Robert Panther

Richtig einsteigen:
Datenbanken entwickeln mit SQL Server 2008 (R2) Express

Mit SQL Server 2008 R2 Express auf DVD

Alles was Einsteiger benötigen:

- Viele Übungen und Insider-Tipps
- Hintergrundwissen und Konzepte
- Software zum sofortigen Loslegen

Microsoft Press

Robert Panther

Richtig einsteigen: Datenbanken entwickeln mit SQL Server 2008 R2 Express

Microsoft Press

Robert Panther – Richtig einsteigen: Datenbanken entwickeln mit SQL Server 2008 R2 Express

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
12 11 10

ISBN 978-3-86645-218-3, PDF-eBook-ISBN 978-3-86645-385-2

Umschlaggestaltung: Hommer Design GmbH, Haar (www.HommerDesign.com)
Fachlektorat, Layout und Satz: Haselier IT-Services, Amsterdam
Korrektorat: Frauke Wilkens, München
Gesamtherstellung: Kösel, Krugzell (www.KoeselBuch.de)

Inhaltsverzeichnis

Teil III
Datenbankentwicklung

Teil V
Erweiterte Funktionen

Anhänge

Vorwort

Um es vorwegzunehmen: Ja, ich bin bekennender »SQL-Junkie«. Ich beschäftige mich schon so lange mit SQL-basierten Datenbanksystemen, dass ich gelegentlich gerne erzähle, dass ich zweisprachig aufgewachsen bin: Meine Mutter sprach mit mir Deutsch, mein Vater SQL (`INSERT Haferbrei INTO Kleinkind`, `UPDATE Kleinkind SET Milchzaehne=Milchzaehne+1` und dergleichen), aber das ist natürlich doch ein wenig übertrieben. Denn als IBM 1975 die Datenbankabfragesprache SEQUEL (den Vorgänger von SQL) auf den Markt brachte, war ich schließlich schon reif für die Grundschule und damit schon über das Sprechenlernen hinaus. SQL habe ich dann doch erst einige Jahre später gelernt.

Dennoch begleitet mich der SQL Server bereits sehr lange und zwar seit dem Jahr 1995, in dem mit der Version 6.0 die erste richtige Microsoft-Version des Produkts das Licht der Welt erblickte (bis dahin war es ja eher eine Gemeinschaftsproduktion von Microsoft und Sybase). Seitdem hat sich viel getan, alle paar Jahre kommt eine neue Version des Produkts auf den Markt und neben neuen Features, die hinzukommen, wird der SQL Server auch mit jeder neuen Version leistungsfähiger, sodass man davon ausgehen kann, dass selbst die aktuelle Express Edition den vollwertigen SQL Server in der 6.0er-Version bei Weitem übertrifft.

Aber auch die neuen Features, die mit jeder Version dazukommen, machen das Produkt immer wieder aufs Neue interessant und lassen einen – selbst nach langjähriger Erfahrung – immer wieder neue Dinge hinzulernen. Daher hat es mich sehr gefreut, als Sandra Michel von Microsoft Press bei mir anfragte, ob ich ein Buch zur Express Edition des Microsoft SQL Server übernehmen könnte.

Das Ergebnis von zahlreichen nächtlichen Schreibattacken, während derer neben vielen Litern Espresso auch ganze Heerscharen an diversen Weingummitieren – um kein einzelnes Produkt in den Vordergrund zu stellen – ihr Leben lassen mussten (um als Nervennahrung zu dienen), halten Sie nun in den Händen.

Bevor ich mit dem eigentlichen Buchtext beginne, will ich aber noch ein Dankeschön an verschiedene Personen loswerden:

Vor allem bedanke ich mich bei meiner Frau Birgit für ihre Toleranz, da sie in den letzten Monaten meine Aufmerksamkeit nur in sehr geringem Maße erhielt, nachdem ich – obwohl körperlich zu Hause anwesend – meist in Gedanken in den Buchtext vertieft war.

Ein großes Dankeschön geht natürlich auch an Sandra Michel von Microsoft Press für ihr Vertrauen, mir dieses Projekt zu übertragen und ihre Geduld, was das Verschieben diverser Abgabetermine anbelangt. Ebenso gilt mein Dank Rainer G. Haselier, der mich sowohl als Fachlektor unterstützt hat als auch den Satz des Buchtextes übernommen hat.

Des Weiteren bedanke ich mich natürlich auch bei allen anderen Freunden und Kollegen, die mich bei diesem Buch in diversen Gesprächen durch interessante Anregungen direkt oder indirekt unterstützt haben.

Robert Panther,
Königstein im Juni 2009

Vorwort zur 2. Ausgabe

So wie das Vorwort der ersten Ausgabe mit Danksagungen geendet hat, möchte ich an dieser Stelle gleich damit fortsetzen und mich bei den Lesern der ersten Ausgabe bedanken, die dieses Buch gekauft und damit die zweite Ausgabe – etwas über ein Jahr nach der ersten – überhaupt ermöglicht haben. Inzwischen hat sich natürlich auch bei Microsoft einiges getan. Im April ist die aktualisierte Fassung von SQL Server 2008 mit dem Namenszusatz R2 erschienen. Auch wenn sich die Neuerungen primär auf die größeren Editionen beziehen, gibt es auch einige Auswirkungen für die Express Edition, die in dieser zweiten Ausgabe natürlich berücksichtigt wurden. Da die ursprüngliche 2008er-Variante momentan aber nach wie vor häufiger installiert ist, ist es sicher beruhigend zu wissen, dass die meisten in diesem Buch enthaltenen Informationen unverändert für beide Varianten gelten. Dort, wo sich die Informationen gezielt auf eine der beiden Versionen beziehen, wird explizit darauf hingewiesen. Auf der Buch-DVD sind beide Versionen der Software enthalten, sodass Sie selbst wählen können, welche Sie verwenden möchten.

Ganz gleich, für welche Variante Sie sich entscheiden, haben Sie mit diesem Buch und der dazu gehörenden DVD alles, was Sie benötigen, um erfolgreich in die Welt von Microsoft SQL Server einzusteigen. So bleibt mir nur, Ihnen viel Spaß beim Lesen und Entdecken von SQL Server zu wünschen.

Robert Panther,
Königstein im September 2010

Kapitel 1

Einleitung

In diesem Kapitel lernen Sie

- warum und für welche Zielgruppe dieses Buch entstanden ist
- wie dieses Buch aufgebaut ist
- welche Formatierungen und Symbole im Text verwendet werden

1.1 Warum dieses Buch?

Mit SQL Server 2008 Express bzw. dem neueren SQL Server 2008 R2 Express bietet Microsoft eine kostenfrei erhältliche Version des aktuellen SQL Server, die bereits einen beachtlichen Funktionsumfang enthält und damit für viele Projekte bestens geeignet ist.

Wenn Sie bereits mit einer älteren Version von SQL Server gearbeitet haben, werden Sie die vielen interessanten Neuerungen schätzen lernen, mit denen der bereits vielfach bewährte SQL Server 2005 weiter verbessert wurde.

Wenn Sie stattdessen bisher mit Microsoft Access oder der MSDB einfache Datenbankanwendungen erstellt haben, ist SQL Server 2008 R2 Express der nächste logische Schritt, um diese Anwendungen auf eine skalierbare Plattform zu portieren. Denn wenn die Express-Edition einmal nicht mehr ausreichen sollte, können Sie die Datenbank problemlos auf einer größeren Variante von SQL Server einspielen, ohne die Anwendung ändern zu müssen.

Und auch wenn Sie bisher noch gar keine Erfahrung mit dem Entwickeln von Datenbankanwendungen haben, ist SQL Server 2008 R2 Express das ideale Produkt, um in diese Thematik einzusteigen.

Mit diesem Buch erhalten Sie einen strukturierten Einstieg in die Möglichkeiten, die SQL Server 2008 R2 Express bietet. Selbst wenn Sie später mit einer größeren Variante von Microsoft SQL Server arbeiten sollten, können Sie auf das hier vermittelte Grundwissen aufbauen und müssen lediglich die zusätzlichen Features der größeren Versionen dazulernen. Somit ist dieses Buch auch als Grundlagenbuch für alle anderen Editionen von SQL Server geeignet.

1.2 Aufbau des Buches

Dieses Buch gliedert sich in fünf größere Abschnitte, die durch einen Anhang ergänzt werden.

- **Teil I, Einführung**, gibt einen kurzen Überblick über Microsoft SQL Server. Nach einer Beschreibung der Historie werden die Neuerungen der 2008er-Variante vorgestellt, gefolgt von einem Vergleich der verschiedenen Editionen von SQL Server. Danach wird die Installation der Express Edi-

tion beschrieben. Abgerundet wird dieser Abschnitt des Buches durch einen Überblick über die wichtigsten Tools.

- **Teil II, Datenbankgrundlagen**, vermittelt die wichtigsten Grundlagen, die Sie benötigen, um mit Datenbank-Management-Systemen wie SQL Server 2008 Express zu arbeiten. Nach ein paar allgemeinen Grundlagen folgt eine Beschreibung, wie Sie die wichtigsten Datenbankobjekte wie Datenbanken, Tabellen, Sichten und Indizes anlegen und verwalten können. Abgerundet wird dieser Teil durch eine Einführung in die wichtigsten SQL-Anweisungen zum Abfragen, Einfügen, Ändern und Löschen von Daten.
- **Teil III, Datenbankentwicklung**, befasst sich mit fortgeschritteneren Themen der Datenbankentwicklung. Ein Kapitel widmet sich der erweiterten SQL-Programmierung wie beispielsweise komplexeren SQL-Abfragen, ein weiteres der Programmierung von Triggern, gespeicherten Prozeduren und Funktionen. Dazu werden auch die Besonderheiten von SQL-Skripts behandelt und erläutert, wie man diese debuggen oder gar eine Fehlerbehandlung zur Laufzeit implementieren kann.
- **Teil IV, Datenbankadministration**, führt Sie in die Nutzung der administrativen Möglichkeiten von SQL Server ein. Dabei wird das Anlegen, Ändern und Löschen von Datenbanken, Tabellen, Indizes etc. sowohl über die Benutzeroberfläche als auch mit der Sprache SQL beschrieben. Aber auch andere wichtige administrative Themen wie die Benutzer- und Rechteverwaltung sowie das Sichern und Wiederherstellen von Daten kommen hier nicht zu kurz.
- **Teil V, Erweiterte Funktionen**, erklärt die Nutzung von erweiterten Möglichkeiten von SQL Server Express. Das ist einerseits das Zusammenspiel mit .NET, um zur Datenbank auch komfortable Anwendungen entwickeln zu können. Andererseits werden die Reporting-Features beschrieben, die Sie mit SQL Server Express Advanced Edition nutzen können. Abschließend folgt ein Kapitel, in dem die Möglichkeiten zur Zusammenarbeit mit den größeren (und kleineren) Editionen von SQL Server aufgezeigt werden. So lässt sich SQL Server Express beispielsweise als Client für ein Replikationsszenario verwenden. Aber auch die Compact Edition, die automatisch mit SQL Server Express installiert wird, wird behandelt. Das letzte Kapitel dieses Teils schließlich beschreibt die Verwendung von Datenebenenanwendungen, die mit SQL Server 2008 R2 eingeführt wurden.
- Im **Anhang** finden Sie eine kurze Referenz der wichtigsten SQL-Befehle. Dazu eine ausführliche Beschreibung des Inhalts der Buch-DVD gefolgt von einem Überblick über weitere Informationsquellen zum Thema im Internet. Abgeschlossen wird der Anhang durch ein Glossar, in dem die wichtigsten Fachbegriffe erklärt sind, sowie den obligatorischen Index.

Aufbau der einzelnen Kapitel

Die einzelnen Kapitel des Buches sind folgendermaßen aufgebaut:

- **Lernziele** Sie finden am Anfang jedes Kapitels eine Übersicht zu den Lernzielen und Themen des Kapitels.
- **Schrittfolgen** Die Anleitungen für das Erstellen der Datenbanken sind als Schrittfolgen dargestellt. Auch wenn Sie die Beispieldatenbanken und -projekte von der Website zum Buch herunterladen können, empfehle ich Ihnen, alle Beispiele selbst zu erstellen, indem Sie Schritt für Schritt des beschriebenen Ablaufs nachvollziehen. Sie werden auf diese Weise schneller mit der Oberfläche der diversen SQL Server Tools sowie der Syntax von T-SQL vertraut.

- **Fazit** Kurz vor Ende jedes Kapitels finden Sie eine kurze Zusammenfassung der wichtigsten Lerninhalte, damit Sie sich das gerade Erlernte noch einmal ins Gedächtnis zurückrufen können.
- **Übungen** Nach dem Fazit folgt meist noch ein Abschnitt mit Übungen zu dem jeweiligen Kapitel. Manchmal handelt es sich dabei um einfache Fragen, manchmal enthält eine Übung eine Aufgabe, um die im Kapitel durchgeführte Datenbankoperation noch weiter zu verfeinern. Alle Antworten finden Sie auf der Website zu diesem Buch. Versuchen Sie, die Übungen selbstständig durchzuführen, und schauen Sie sich erst dann die Lösungen an. So werden Sie schneller in der Lage sein, eigene Datenbankprojekte zu realisieren.

1.3 Die Beispieldatenbank

Um einen echten Praxisbezug zu erreichen, wird im gesamten Buch eine einheitliche Beispieldatenbank verwendet, die im Laufe des Textes immer weiter entwickelt wird. Dabei wurde bewusst vermieden, die zweitausendundzweite Adressverwaltung zu entwerfen. Dennoch wurde ein relativ gängiges Anwendungsbeispiel verwendet, damit die Beispiele auch für jeden möglichst gut nachvollziehbar sind. Und zwar geht es um eine Mediendatenbank, mit der Sie Bücher, Audio-CDs und DVDs verwalten können.

Ich selbst kenne die Notwendigkeit einer solchen Datenbank nur zu gut, gehöre ich doch zu den vielen »Jägern und Sammlern«, die über eine recht große Anzahl an CDs und DVDs verfügen. Hin und wieder passiert es dann, dass ich eine interessante CD oder DVD günstig erstehe, um zu Hause dann festzustellen, dass diese bereits in meinem Regal stand. (Zum Glück kommt das allerdings nicht allzu oft vor.)

Um derlei Missgeschick künftig völlig auszuschließen, benötigt man also eine Mediendatenbank (möglichst auch in einer Offlinevariante, die man dann beim nächsten Besuch des Elektronikmarkts seines Vertrauens beispielsweise im Mobiltelefon parat hat). Wenn man dann noch eine einfache Suchoberfläche zur Verfügung hat, um mit wenigen Tastendrücken nachzusehen, ob ein Titel schon in der privaten Sammlung enthalten ist, ist das Problem gelöst.

Im Vordergrund der meisten Beispiele in diesem Buch wird allerdings weniger die Benutzeroberfläche der zugehörigen Anwendung, sondern vielmehr die Datenbank selbst stehen.

1.4 Schreibweisen

Dieses Buch verwendet einige wenige Konventionen. Wenn Sie mit ihnen vertraut sind, können Sie einfacher mit dem Buch arbeiten:

- **Fette Schrift** Text, den Sie in den Schrittfolgen eintippen müssen, wird in fett gedruckten Buchstaben dargestellt. Auch in Codelistings werden Änderungen, die in einem bestimmten Schritt im Quellcode vorgenommen wurden, fett, aber dann in Listingschrift, dargestellt.
- *Kursive Schrift* Die kursive Schrift wird für Dateinamen verwendet, bezeichnet Elemente der Benutzeroberfläche (in den meisten Fällen also des Menüs des SQL Server Management Studios, Dialogfelder und auch die Namen der Eigenschaften, wie sie im Management Studio oder in der IDE erscheinen) und wird in seltenen Fällen auch für Hervorhebungen verwendet. Auch Datenbankobjekte wie die Namen von Datenbanken, Tabellen, Feldern etc. werden in kursiver Schrift dargestellt.

- `Listingschrift` Die Listingschrift wird verwendet, wenn Codefragmente aus dem Beispielcode abgedruckt werden. Im Fließtext werden die Begriffe in Listingschrift gedruckt, die auf Stellen im Quellcode verweisen.

Zusätzlich finden Sie im ganzen Buch verteilt Absätze, die mit den folgenden Symbolen versehen sind und die auf bestimmte, hilfreiche Elemente hinweisen:

	Wichtig	Absätze mit diesem Icon enthalten wichtige Informationen, auf die Sie unbedingt achten sollten.
	Hinweis	Absätze mit diesem Icon enthalten weiterführende Informationen, die man im Hinterkopf behalten sollte.
	Tipp	In den Tipp-Absätzen finden Sie Informationen dazu, wie Sie Dinge besonders einfach und zeitsparend machen können.
	Best Practices	Mit diesem Symbol markierte Absätze enthalten Beschreibungen zu gängigen Vorgehensweisen, die sich in der Praxis bewährt haben.
	Hintergrundinfo	Absätze, die dieses Icon besitzen, liefern wichtige Hintergrundinformationen. Die Informationen sind nicht erforderlich, um eine bestimmte Schrittfolge durchzuführen, sie vertiefen jedoch Ihr Wissen über die Zusammenhänge, in denen eine bestimmte Aktion steht.
	Internet-Links	Mit diesem Icon werden Links zu weiterführenden Kapiteln aus anderen Microsoft Press-Büchern gekennzeichnet, die Sie kostenlos herunterladen können und die Themen, die in diesem Buch nur angerissen werden können, vertiefen. Dieses Icon wird auch verwendet, um Sie auf andere Websites im Internet hinzuweisen, auf denen Sie unterhaltsame und nützliche Informationen finden, die mit dem Thema, das behandelt wird, in Zusammenhang stehen.

Bei der Wahl von Fachbegriffen aus dem Umfeld von SQL Server wird in der Regel vorrangig die deutschsprachige Bezeichnung verwendet. Zusätzlich wird an vielen Stellen aber auch auf die englischsprachige Variante hingewiesen, da Sie diese in vielen – teilweise selbst deutschsprachigen – Quellen finden und dadurch die Suche nach weiterführenden Informationen zu einem bestimmten Thema im Internet erleichtert wird. Beispiel: Sichten (engl. Views)

1.5 DVD, Softlinks und Website zum Buch

Damit Sie das in diesem Buch Erlernte sofort ausprobieren können, liegt eine DVD bei, auf der alles enthalten ist, was Sie benötigen, um gleich loszulegen. Das betrifft einerseits die dargestellten Programmbeispiele, aber auch die Installationsdateien für SQL Server 2008 Express bzw. SQL Server 2008 R2 Express.

Die Installation der Software ist in *Kapitel 3 – Erste Schritte mit SQL Server 2008 Express* ausführlich beschrieben. Eine ausführliche Beschreibung der sonstigen enthaltenen Dateien finden Sie im Anhang dieses Buches.

Unter der Webadresse *www.vsxpress.de* finden Sie die Website zum Buch, wo neben eventuellen Fehlerkorrekturen zum Text (nobody is perfect) auch aktualisierte Beispieldateien sowie zusätzliche Downloads zu finden sind, die in den jeweiligen Kapiteln beschrieben sind. Dies sind unter anderem weitere Beispielprojekte und Probekapitel aus anderen Microsoft Press-Büchern, mit denen Themen, die in diesem Buch aus Platzgründen nur einführend erläutert werden können, vertieft werden.

Im Text des Buches finden Sie außerdem einige Softlinks, die Sie auf interessante, unterhaltsame oder lehrreiche Websites verweisen. Sie können die Softlinks entweder direkt auf der Startseite von *www.vsxpress.de* eingeben oder in Ihren Browser die angegebene Adresse eintippen, um die jeweilige Website zu öffnen.

Um die Softflinks direkt in die Adressleiste Ihres Browsers einzugeben, verwenden Sie folgende Syntax: *http://go.vsxpress.de/?linkid=**id***, wobei Sie dann ***id*** durch die angegebene Link-ID ersetzen.

1.6 Zusammenfassung

Dieses einführende Kapitel hat Ihnen einen Überblick gegeben, wie dieses Buch aufgebaut ist und welche Schreibweisen darin verwendet werden.

Es wurde selbstverständlich alles unternommen, um die Richtigkeit des Buchinhalts und des Bonusmaterials sicherzustellen. Etwaige Korrekturen und Änderungen finden Sie unter folgender Adresse:

http://www.microsoft-press.de/support.asp

Wenn Sie Kommentare, Fragen oder Anregungen zum Inhalt dieses Buches oder des Begleitmaterials bzw. Fragen haben, die Sie auf den oben angegebenen Websites nicht klären konnten, senden Sie eine E-Mail an folgende Adresse bei Microsoft Press:

presscd@microsoft.com

oder per Post an:

Microsoft Press
Konrad-Zuse-Str. 1
85716 Unterschleißheim

Beachten Sie, dass Microsoft unter diesen Adressen keinen Support für Softwareprodukte leistet.

So, nun wünsche ich aber viel Vergnügen beim Lesen dieses Buches und Entdecken der zahlreichen Möglichkeiten von SQL Server 2008 Express.

Robert Panther

Kapitel 2

Der Microsoft SQL Server

In diesem Kapitel lernen Sie

- wie SQL Server zu dem geworden ist, was er heute ist
- welche Neuerungen in der aktuellen 2008er-Version dazukamen
- was sich bei SQL Server 2008 R2 geändert hat
- wodurch sich die verschiedenen Editionen des SQL Server voneinander unterscheiden

2.1 Historie des Microsoft SQL Server

Microsoft SQL Server kann inzwischen auf eine bewegte Vergangenheit zurückblicken, die inzwischen über 20 Jahre lang ist. Diese lange Zeit belegt natürlich auch, dass es sich bei diesem Datenbank-Management-System um ein sehr ausgereiftes Produkt handelt.

Sybase und die Anfänge des Microsoft SQL Server

Bereits 1988/89 kam die erste Version des SQL Server als Gemeinschaftsproduktion von Ashton-Tate, Sybase und Microsoft auf den Markt, damals noch primär für das Betriebssystem OS/2 gedacht. Der Funktionsumfang entsprach etwa der damaligen Sybase Version 3.0 für Unix.

Das nächste wesentliche Release – immer noch für OS/2 – erschien vier Jahre später mit der Version 4.2, dicht gefolgt von der Version 4.21 für Windows NT, die zeitgleich mit Windows NT 3.1 erschien. Funktionsumfang und Leistungsfähigkeit waren noch recht beschränkt, doch nicht zuletzt die intuitive Oberfläche machten dieses Produkt zumindest als Datenbank für kleine Abteilungen interessant.

Microsoft SQL Server entsteht

Im Jahr 1994 beendeten Sybase und Microsoft ihre Zusammenarbeit. Microsoft erkämpfte sich die Rechte für alle Versionen für Microsoft-Betriebssysteme, während Sybase das Produkt unter dem Namen Sybase Adaptive Server weiterentwickelte. Ein Jahr später – nachdem große Teile der Datenbank-Engine des SQL Server komplett neu programmiert waren – erschien mit der Version 6.0 der erste »richtige« Microsoft SQL Server. Sowohl Leistungsumfang als auch Performance waren stark verbessert, wodurch Microsoft SQL Server eine ernst zu nehmende Alternative für die etablierten Datenbank-Management-Systeme anderer Hersteller wurde, die meist ein Vielfaches kosteten.

Im Jahre 1996 wurde SQL Server 6.0 durch die Version 6.5 abgelöst, die allerdings keine wesentlichen neuen Features brachte, sondern vielmehr eine Menge an Detailverbesserungen.

Auch wenn Sybase seit der Version 6.0 keinen Einfluss mehr auf die Produktentwicklung hatte, so waren bis zur Version 7.0, die 1998 erschien, immer noch Teile des alten Quellcodes darin enthalten. Mit der Version 7.0 allerdings wurde der Kern komplett neu entwickelt. Ein Jahr später kamen die OLAP Tools dazu, mit denen man in Kombination mit den ebenfalls integrierten Data Transformation Services ein Data Warehouse aufbauen konnte, was bis dahin nur wesentlich teureren Softwarelösungen vorbehalten war.

Der SQL Server wird erwachsen

Ab der Version 8.0 – die im Jahr 2000 erschien – wurde der offizielle Produktname umgestellt, sodass die Software nun als SQL Server 2000 auf den Markt kam. Intern wurden die Versionsnummern aber auch nach dem alten Konzept weitergeführt, sodass es seitdem eine interne Versionsnummer und eine offizielle Produktbezeichnung gibt.

Die wesentlichen Neuerungen des SQL Server 2000 waren eine deutlich verbesserte Performance, Skalierbarkeit und Zuverlässigkeit, womit das Produkt für den Unternehmenseinsatz interessanter wurde. Parallel dazu wurden aber auch die Lizenzkosten deutlich angehoben, lagen aber immer noch deutlich unter denen der Konkurrenz. So wundert es nicht, dass SQL Server 2000 ein Jahr nach seinem Erscheinen Oracle von der Spitzenposition der Datenbank-Management-Systeme für Windows verdrängte.

Etwas später kamen dann die Reporting Services als kostenfreies Add-On hinzu, womit SQL Server 2000 alle Komponenten für eine komplette Business Intelligence-Lösung beinhaltete. Im Jahr 2003 erschien erstmals auch eine 64-Bit-Version von SQL Server (bei den darauffolgenden Versionen wurde diese immer nahezu zeitgleich mit der 32-Bit-Version ausgeliefert).

SQL Server bekommt neue Tools

Erst fünf Jahre nach dem Erscheinen von SQL Server 2000 kam mit dem SQL Server 2005 das nächste große Release auf den Markt und wusste gleich zu Beginn mit einer Fülle an neuen Features zu überzeugen. Neben einer wiederum verbesserten Datenbankperformance wurden komplett neue Client-Tools mitgeliefert. So wurde beispielsweise der Enterprise Manager des SQL Server 2000 ab 2005 durch das SQL Server Management Studio abgelöst. Aber auch im Bereich Business Intelligence gab es bahnbrechende Neuerungen: Die alten Data Transformation Services wurden durch die sehr viel leistungsfähigeren SQL Server Integration Services (SSIS) abgelöst. Als Tool für OLAP und Data Mining wurden die SQL Server Analysis Services (SSAS) eingeführt. Konsequenterweise wurden natürlich auch die bereits für den SQL Server 2000 erhältlichen SQL Server Reporting Services (SSRS) in das Produkt integriert. Dazu kam eine Vielzahl an weiteren Features wie .NET Framework-Integration, XML-Unterstützung, Service Broker und Notification Services.

Im August 2008 erschien dann die aktuelle Version des SQL Server, der SQL Server 2008 (auch unter dem Codename Katmai bekannt). Im Vergleich zur Vorgängerversion aus dem Jahre 2005 stellt SQL Server 2008 eine weitere Evolutionsstufe dar. Die meisten Bestandteile, die sich beim SQL Server 2005 schon bewährt haben, wurden beibehalten und konsequent weiterentwickelt. Die Änderungen im Detail sind allerdings so vielfältig, dass es Sinn macht, sich diese genauer anzusehen. Daher folgt dazu weiter hinten in diesem Kapitel ein separater Abschnitt.

Im April 2010 erschien dann entgegen ursprünglichen Erwartungen nicht der SQL Server 2010, sondern der SQL Server 2008 R2, der auch den inoffiziellen Beinamen »BI Refresh« erhielt, da diese Version im Wesentlichen Erweiterungen im Bereich der Business Intelligence-Komponenten erhielt. Dass aber auch in anderen Bereichen einiges geändert wurde und sich einige dieser Änderungen auch auf die Express-Editionen auswirken, zeigt der entsprechende Abschnitt weiter hinten in diesem Kapitel. Die Hauptversionsnummer wurde jedoch bewusst beibehalten, da die eigentliche Datenbank-Engine von SQL Server 2008 nahezu unverändert übernommen wurde.

Kleiner Überblick über die wichtigsten Versionen und Builds

Insbesondere bevor die jeweiligen SQL Server offiziell auf dem Markt erscheinen, werden diese meist nur mit ihren Codenamen bezeichnet, die von Microsoft schon während der Entwicklung des Produkts vergeben werden (der eigentliche Produktname wird erst später festgelegt). Da diese Codenamen auch in vielen Blogs und Internetforen auftauchen, ist es sinnvoll, diese schon einmal gehört zu haben, um entscheiden zu können, ob sich ein Artikel auch auf die selbst genutzte Version von SQL Server bezieht.

Tabelle 2.1: Die Historie des SQL Server im Überblick

Jahr	Produktname	Version	Codename
1988/89	SQL Server 1.0	1.0 (für OS/2)	-
1992	SQL Server 4.2	4.2 (OS/2)	-
1993	SQL Server 4.21	4.21 (Windows NT)	-
1995	SQL Server 6.0	6.0	SQL95
1996	SQL Server 6.5	6.5	Hydra
1998	SQL Server 7.0	7.0	Sphinx
1999	SQL Server 7.0 OLAP Tools	7.0	Plato
2000	SQL Server 2000	8.0	Shiloh
2003	SQL Server 2000 (64 Bit)	8.0	Liberty
2005	SQL Server 2005	9.0	Yukon
2008	SQL Server 2008	10.0	Katmai
2010	SQL Server 2008 R2	10.5	Kilimanjaro

Natürlich gab es zu den meisten oben genannten Versionen noch diverse Updates und Service Packs. So ist im April 2010 das Service Pack 1 für SQL Server 2008 herausgekommen, bevor – ziemlich genau ein Jahr später – die Version R2 von SQL Server 2008 erschien.

Leider zeigt ein bereits installierter SQL Server nicht die Nummer des Service Packs, sondern nur die interne Buildnummer an, daher hier eine kleine Übersicht der Service Packs zu den neueren SQL Server Versionen.

Tabelle 2.2: Zuordnungstabelle SQL Server-Buildnummer und -Service-Packs

Erscheinungstermin	Produktname	Buildnummer	Service Pack
SQL	Server 2000	8.00.194	-
	SQL Server 2000 SP1	8.00.384	SP1 ▸

Tabelle 2.2: Zuordnungstabelle SQL Server-Buildnummer und -Service-Packs *(Fortsetzung)*

Erscheinungstermin	Produktname	Buildnummer	Service Pack
	SQL Server 2000 SP2	8.00.534	SP2
	SQL Server 2000 SP3	8.00.760	SP3
	SQL Server 2000 SP3a	8.00.760	SP3a
06.06.2005	SQL Server 2000 SP4	8.00.2039	SP4
20.12.2005 S	QL Server 2005	9.00.1399	-
18.04.2006	SQL Server 2005 SP1	9.00.2047	SP1
06.03.2007	SQL Server 2005 SP2	9.00.3042	SP2
15.12.2008	SQL Server 2005 SP3	9.00.4035	SP3
22.08.2008 SQL	Server 2008	10.00.1600.22	-
16.03.2009	SQL Server 2008 (kumulatives Update 4)	10.00.1798 -	
23.02.2009	SQL Server 2008 SP1 (CTP)	10.00.2520.00	SP1 (CTP)
07.04.2009	SQL Server 2008 SP1	10.00.2531.00	SP1
November 2009	SQL Server 2008 R2 (CTP)	10.50.1352.12	-
20.04.2010	SQL Server 2008 R2	10.50.1600.1	-

Hintergrundinfo: CTPs, RTMs, Service Packs, Hot Fixes und kumulative Updates

Bei den Versio nsbezeichnungen ak tueller Microsoft-Produkte we rden verschiedene Begriffe verwendet, die zu Verwirrung führen können. Daher werden die Begriffe hier erklärt:

- **Community Technology Preview (CTP)** Hierbei handelt es sich um Vorabversionen, die bereit s den endgültig en Funktionsumfang beinhal ten, aber der Benutzer-Community schon zur Verfügung gestellt werden, noch bevor sie zur eigentlichen Produktion freigegeben sind.
- **Release Candidate (RC)** Ein Re lease Candidate ist die letzte Testversion, in der alle bereit s bekannten Fehler behoben s ind. Werden beim Te st Fehl er gefunden, so wird ein weiterer Release Candidate erstel lt. Dies er Proze ss wird so lange wi ederholt, bis der aktuellste R e-lease Candidate fehlerfrei ist.
- **Ready To Manufacturing (RTM)** Bei einer RTM-Version handelt es sich bereits um die endgültige Version des Produkts, die auch in derselben Form in Produktion geht.
- **Service Pack** Ein Service Pack ist ein größeres Updat e eines Produkts, das nicht selten auch wesentliche Erweiterungen des Funktionsumfangs beinha ltet. Ein Servic e Pack beinhal tet normalerweise alle bis dahin erschienenen Updates sowie vorherige Service Packs.
- **Hotfix** Während neue Service Packs nur in relativ großen Zeitabständen verfügbar sind, erstellt Microsoft sehr häufig kleinere Updat es, die kleine re Fehler beheben. Insbesondere wenn es sich um sicherheitsrelevante Fehler handelt, erscheinen diese sehr zeitnah nach deren Behebung in Form von Hotfixes.
- **Kumulatives Update** Da sich Hotfixes und andere Updates nach einiger Zeit ansammeln, werden die se von Zeit zu Zeit in kumulati ven Updates zusammengefasst, die alle Updates seit dem letzte n Service Pack beinhalten. Dami t spart man sich die Einzelinstallation von zahlreichen separaten Updates.

2.2 Neuerungen bei SQL Server 2008

Mit dem SQL Server 2008 hält eine ganze Reihe an Erweiterungen und Verbesserungen Einzug in das Produkt. Dabei handelt es sich im Gegensatz zum Wechsel von SQL Server 2000 zu 2005 nun eher um eine Evolution, da es weniger grundlegende Änderungen, sondern mehr konsequente Weiterentwicklungen einzelner Komponenten gibt, bei denen wohl auch viele Kritikpunkte der SQL Server-Benutzer-Community berücksichtigt wurden.

Ein zentraler Aspekt bei vielen dieser Neuerungen ist das Ziel von Microsoft, mit den verschiedenen Editionen von SQL Server eine einheitliche Datenbanktechnologie zur Verfügung zu stellen, die von sehr kleinen Systemen (beispielsweise auf Mobile Devices) bis hin zu hochverfügbaren Unternehmensservern skalierbar ist. Wie weiter hinten in diesem Kapitel beim Vergleich der verschiedenen Editionen ersichtlich wird, ist dies auch gut gelungen.

Doch beschränken wir uns vorerst auf die Neuerungen, die für die Express Edition relevant sind:

Neue Datentypen

Einige der wichtigsten Neuerungen von SQL Server 2008 basieren auf den neuen Datentypen, die mit dieser Produktgeneration eingeführt wurden. So gibt es als Alternative zum altbewährten *datetime*-Datentyp nun endlich auch getrennte Datentypen für Uhrzeit *(time)* und Datum *(date)*.

Aber auch komplexere Datentypen sind hinzugekommen, was nicht zuletzt der konsequenteren Integration der Common Language Runtime zu verdanken ist. Denn hierdurch können Datentypen auf Basis von .NET-Klassen erzeugt werden.

Die vielleicht interessantesten neuen Datentypen befassen sich mit der Speicherung von Geodaten. So gibt es einen Typ *Geometry,* der zweidimensionale geometrische Daten verarbeitet. Analog dazu gibt es den Datentyp *Geography,* mit dem man geografische Daten in Form von Längen- und Breitengrad speichern kann.

Um mit hierarchischen Daten arbeiten zu können, ist der neue Datentyp *HierarchyId* hinzugekommen, der es ermöglicht, ganze Hierarchien in einer selbstreferenzierenden Tabelle besser abbilden zu können.

Zur effektiveren Integration von unstrukturierten Daten, die beispielsweise als Dateien vorliegen, bietet SQL Server nun das Konstrukt der Filestreams an. Damit lassen sich Dateien in einem Ordner des Dateisystems ablegen, der aber vom SQL Server verwaltet wird und damit stets konsistent zur Datenbank selbst bleibt. Bisher gab es zur Lösung dieses Problems zwei gängige Varianten, die aber beide große Nachteile hatten. Entweder man speicherte die Dateien komplett in binären Datenbankfeldern, was zu einer extremen Vergrößerung der Datenbank führen kann, die dadurch sehr unhandlich wurde. Oder man speicherte in der Datenbank lediglich eine Verknüpfung auf eine Datei, die dann direkt im Dateisystem abgelegt wurde. Im letzteren Fall musste man aber selbst dafür Sorge tragen, dass Verknüpfungen und Dateien immer konsistent bleiben (also beispielsweise die Datei nicht verschoben oder gelöscht werden kann, ohne dass die Verknüpfung darauf auch aktualisiert wird). Beim Filestream wird der direkte Dateizugriff auf das entsprechende Verzeichnis aber vom SQL Server blockiert, sodass der Zugriff nur kontrolliert über die Mittel des SQL Server erfolgen kann.

Sonstige Neuerungen für SQL Server 2008 Express

Die meisten Änderungen im Bereich Administration beziehen sich auf die größeren Editionen von SQL Server. Aber auch für die Express Edition gibt es ein paar Neuerungen in diesem Umfeld. So lassen sich mit dem neuen Policy Based Management einheitliche Richtlinien für Server oder gar ganze Servergruppen definieren.

Mehr hat sich allerdings im Bereich der Datenbankentwicklung getan. Das SQL Server Management Studio unterstützt nun endlich IntelliSense, was einerseits Arbeit spart und andererseits die Gefahr von Tippfehlern minimiert. Dazu lassen sich SQL-Skripts nun so – wie es viele Entwickler auch von diversen Visual Studio her gewohnt sind – direkt im Management Studio mit Haltepunkten versehen und debuggen.

Als wichtigster neuer SQL-Befehl kommt die *Merge*-Anweisung hinzu, die umgangssprachlich gerne auch als »Upsert« bezeichnet wird, weil sie INSERT und UPDATE in einer Anweisung vereint. Dadurch kann mit einer einzigen Anweisung eine Zieltabelle aufgrund von einer Importtabelle mit demselben Primärschlüssel aktualisiert werden. Ist der Primärschlüssel bereits vorhanden, wird die vorhandene Zeile aktualisiert. Fehlt der Primärschlüssel, wird die fehlende Zeile eingefügt. Im Gegensatz zum SQL-Standard bietet die Microsoft-Variante des *Merge*-Befehls sogar noch die optionale Erweiterung, in der Zieltabelle vorhandene Zeilen zu löschen, wenn diese in der Importtabelle fehlen.

Es gibt aber auch noch ein paar weitere Neuerungen, wie beispielsweise die Möglichkeit, beim Aufruf einer gespeicherten Prozedur ganze Tabellen als Parameter zu übergeben – die sogenannten Table-valued Parameter.

Neue Features für die größeren Editionen von SQL Server 2008

Dazu gibt es noch eine ganze Reihe an neuen Features, die den größeren Editionen (Workgroup, Standard und Enterprise Edition) vorbehalten sind. Der Vollständigkeit halber sollen diese hier auch kurz erwähnt werden, ohne allzu detailliert darauf einzugehen.

So kommt ab der Web bzw. Workgroup Edition die Möglichkeit dazu, mithilfe des Performance Data Collectors bei nur geringer zusätzlicher Systemlast Performancedaten zu sammeln, die dann in entsprechenden Reports (dem Performance Data Warehouse) im SQL Server Management Studio abgerufen werden können.

Ab der Standard Edition gibt es zusätzlich die Plan Guides, durch die man Optimizer Hints definieren kann, die automatisch auf Abfragen angewendet werden, die einer vorgegebenen Form entsprechen. Damit lassen sich die Vorteile von Optimizer Hints auch dann nutzen, wenn die Abfrage selbst nicht geändert werden kann.

Mit der Enterprise Edition gibt es dann noch eine ganze Reihe von weiteren Features wie beispielsweise die transparente Datenbankverschlüsselung oder die ebenso transparente Kompression von Datenbanken und Backups. Dazu kommt der neue Resource Governor, mit dem man die maximale Prozessor- und Speicherauslastung für selbst definierte Ressourcenpools festlegen kann. Durch die Zuordnung von Ressourcenpools zu Workloads, denen wiederum Logins zugeordnet werden, lässt sich damit genau steuern, wie sehr bestimmte Anwendungen und Benutzer das Gesamtsystem belasten dürfen.

Zu den bereits genannten Neuerungen gibt es noch zahlreiche Erweiterungen und Verbesserungen im Business Intelligence-Umfeld. Bei den Analysis Services gibt es nun Performanceverbesserungen für

berechnete Kennzahlen sowie ein proaktives Caching. Im Bereich Data Mining gibt es ein neues Add-In für Microsoft Excel. Auch bei den Integration Services gibt es Performanceverbesserungen – beispielsweise durch ein verbessertes Thread Handling und das Caching von Lookup-Tabellen. Dazu kommt endlich die Möglichkeit, Skript-Tasks und Skriptkomponenten neben Visual Basic .NET auch in C# zu programmieren.

Bei den Reporting Services gibt es eine Reihe von Neuerungen, die sich teilweise auch auf die Express Edition beziehen. Hier sind vor allem die optisch ansprechenderen Chart Controls (die von Dundas lizenziert wurden) zu nennen sowie das neue Tablix Control, das die Vorteile von Tabelle und Matrix vereint. Generell gibt es mit dem SQL Server 2008 nun die Möglichkeit, die Reporting Services zu nutzen, ohne dass dafür ein Internet Information Server installiert sein muss.

Die Liste der Neuerungen ist sicherlich nicht vollständig. Zu zahlreich sind die vielen Detailverbesserungen, die SQL Server mit der 2008er-Generation erhalten hat. Über die wesentlichsten Neuerungen sollten Sie nun aber informiert sein.

2.3 Neuerungen bei SQL Server 2008 R2

Die auf den ersten Blick auffälligste Neuerung sind sicherlich die beiden neuen Editionen (Datacenter und Parallel Data Warehouse), die weiter hinten in diesem Kapitel etwas ausführlicher beschrieben werden. Abgesehen davon beziehen sich die meisten Neuerungen des R2-Releases auf Erweiterungen im Business Intelligence-Umfeld. Dazu gibt es jedoch auch eine ganze Reihe an Änderungen, die nichts mit Business Intelligence zu tun haben und von denen ein Teil sogar die Express Edition betrifft.

Neue Business Intelligence-Features für SQL Server 2008 R2

Das aufsehenerregendste neue Feature ist sicherlich, das ab der Enterprise Edition optional herunterladbare PowerPivot, das sich wahlweise in Excel oder SharePoint integriert und das extrem schnelle Ad-hoc-Auswertungen großer Datenmengen ermöglicht, indem die Daten komprimiert im Hauptspeicher gehalten werden. Somit können Poweruser einfache Auswertungen auf OLAP-Cubes direkt in der gewohnten Excel-Umgebung erstellen und ausführen.

Die zweite große Änderung nennt sich StreamInsight und ist eine ab der Datacenter Edition verfügbare Technologie, die das Auswerten von Massendaten in Echtzeit ermöglicht. Technologie heißt in diesem Fall allerdings, dass es sich wirklich nur um einen Serverdienst mit einer entsprechenden Programmierschnittstelle handelt, die von einer selbst zu entwickelnden Anwendung bedient werden muss. Dafür kann dann aber – im Gegensatz zu einem Data Warehouse, das immer neu aufgebaut werden muss – in Echtzeit auf Ereignisse und neu eintreffende Daten reagiert werden.

Die restlichen Neuerungen im BI-Bereich beziehen sich vor allem auf die Reporting Services. So gibt es nun ab der Standard Edition die Möglichkeit, über eine Shared Component Library Data Sets und Report Parts separat zu speichern und somit einfach in anderen Berichten wiederzuverwenden.

Sogar ab der Express Edition (mit Advanced Services) wird die Verwendung von geografischen Karten in Berichten unterstützt. Dazu gibt es ab der Standard bzw. Workgroup Edition den neuen Report Builder 3.0 zur Erstellung von Ad-hoc-Reports.

Sonstige neue Features der größeren Editionen von SQL Server 2008 R2

Ab der Enterprise Edition ist es nun möglich, per Multi-Server-Management mehrere Server (ab Workgroup bzw. Web Edition) zentral und übersichtlich in einem Utility Control Point zu verwalten. Über das SQL Server Utility können sowohl Datenebenenanwendungen als auch der Gesundheitszustand von SQL Server-Instanzen überwacht werden. Dabei können anpassbare Auslastungsschwellwerte und Richtlinien verwendet werden.

Ebenfalls ab der Enterprise Edition verfügbar sind die Master Data Services, mit denen Stammdaten für Anwendungen zentral verwaltet und verteilt werden können.

Passend zum Produktnamen erhält SQL Server 2008 R2 auch eine bessere Unterstützung für Windows Server 2008 R2. Somit werden ab der Datacenter Edition bis zu 256 logische Prozessoren unterstützt (vor R2 waren maximal 64 möglich). Dazu kommt eine bessere Unterstützung von Hyper-V sowie Windows PowerShell 2.0.

Eine der wenigen Änderungen, die sich auf die Datenbank-Engine bezieht, ist die Möglichkeit der Unicode-Komprimierung für die Datentypen *nchar* und *nvarchar* (allerdings nicht *nvarchar(max)* oder *ntext).* Diese in Kombination mit Zeilen- oder Seitenkomprimierung nutzbare Technologie bewirkt, dass die genannten Unicode-Datentypen auch nur dann wirklich zwei Bytes pro Zeichen benötigen, wenn auch erweiterte Zeichen genutzt werden. Damit dürfte der Speicherbedarf nur unwesentlich höher sein als bei den Nicht-Unicode-Datentypen. Auch dieses Feature ist erst ab der Enterprise Edition nutzbar.

Neue Features von SQL Server 2008 R2 Express

Für die Express Edition gibt es – abgesehen von der oben bereits erwähnten Kartendarstellung bei den Reporting Services – im Wesentlichen drei relevante Neuerungen. Die erste ist die Unterstützung von Sysprep, die eine automatisierte Installation unter Verwendung von Standardwerten ermöglicht.

Das zweite Feature ist die Unterstützung von SQL Azure, durch die in der Cloud liegende Datenbanken beispielsweise direkt vom Management Studio aus angesprochen werden können.

Die umfangreichste Neuerung aus Sicht der Express Edition ist die Einführung der Datenebenenanwendungen (engl. Data-tier Applications), mit denen Datenstrukturen einer Anwendung (und Änderungen daran) einfacher verwaltet und bereitgestellt werden können. Dieses Thema wird daher in *Kapitel 16 – Datenebenenanwendungen* ausführlich behandelt.

2.4 Die verschiedenen SQL Server-Editionen im Vergleich

Vom Microsoft SQL Server gibt es mittlerweile eine ganze Reihe an unterschiedlichen Editionen, von denen die in diesem Buch behandelte Express Edition nur eine Variante darstellt. Um abzuschätzen, welche Edition für welches Datenbankprojekt die sinnvollste ist, wurde hier eine kleine Übersicht zusammengestellt:

Tabelle 2.3: Obergrenzen der verschiedenen SQL Server 2008-Editionen für CPUs, Hauptspeicher und Datenbankgröße

	Edition	max. CPUs	max. RAM	DB-Größe
Kostenfreie Editionen	Compact Edition 3.5	unbegrenzt	unbegrenzt	4 GB
	Express Edition	1	1 GB	4 GB
Spezialeditionen	Web Edition	4	unbegrenzt	unbegrenzt
	Workgroup Edition	2	4 GB	unbegrenzt
	Developer Edition	unbegrenzt	unbegrenzt	unbegrenzt
Haupteditionen	Standard Edition	4	unbegrenzt	unbegrenzt
	Enterprise Edition	unbegrenzt	unbegrenzt	unbegrenzt

Bei der neueren R2-Version von SQL Server 2008 sind zwei neue Editionen hinzugekommen, wodurch sich die Obergrenzen und Beschränkungen etwas verschoben haben. So wurden zugunsten der größeren Editionen bei der bisher unbeschränkten Enterprise Edition neue Obergrenzen für CPU, Hauptspeicher und Datenbankgröße eingeführt, während diese bei der Express Edition etwas großzügiger gefasst wurden, da hiermit nun Datenbanken mit bis zu 10 GB verwaltet werden können.

Tabelle 2.4: Obergrenzen der verschiedenen SQL Server 2008 R2-Editionen für CPUs, Hauptspeicher und Datenbankgröße

	Edition	max. CPUs	max. RAM	DB-Größe
Kostenfreie Editionen	Compact Edition 3.5	unbegrenzt	unbegrenzt	4 GB
	Express Edition	1	1 GB	10 GB
Spezialeditionen	Web Edition	4	64 GB	524 TB
	Workgroup Edition	2	4 GB	524 TB
	Developer Edition	unbegrenzt	unbegrenzt	unbegrenzt
Haupteditionen	Standard Edition	4	64 GB	524 TB
	Enterprise Edition	8	2 TB	524 TB
Premium-Editionen	Datacenter	unbegrenzt	unbegrenzt	unbegrenzt
	Parallel Data Warehouse	unbegrenzt	unbegrenzt	unbegrenzt

Hinweis: Hardware- und Betriebssystembeschränkungen beachten

Auch dort, wo in der obigen Tabelle »unbegrenzt« angegeben ist, gelten natürlich die Obergrenzen, die vom Betriebssystem und der verwendeten Hardware vorgegeben werden. So können beispielsweise die 32-Bit-Versionen von Windows ohne zusätzliche Tricks nicht mehr als 3 GB Hauptspeicher verwalten, auch wenn Sie die Enterprise Edition des SQL Server einsetzen.

SQL Server Compact Edition

Während alle anderen Editionen des SQL Server aufeinander aufbauen (und jeweils alle Features der kleineren Editionen beinhalten), nimmt SQL Server Compact Edition eine Sonderstellung ein. Hierbei handelt es sich nicht um eine Datenbank, die von einem speziellen Datenbankdienst verwaltet wird,

sondern eher um ein einfaches Datenbankmodul, das prozessintern verwendet werden kann und für die lokale Speicherung von Daten ausgelegt ist.

Dadurch lässt es sich leicht in Anwendungen einbetten und ist sogar für mobile Geräte auf Basis von Windows Mobile (wie beispielsweise Pocket PCs und Smartphones) nutzbar. Hier ist auch die Herkunft des Produkts zu sehen, das seine Ursprünge im SQL Server CE bzw. später dem SQL Server Mobile hat. Dieses Datenbankformat war jedoch ausschließlich auf mobilen Geräten nutzbar, was das Entwerfen von Datenbanken nicht gerade vereinfachte. Mit der Ablösung des SQL Server Mobile durch die Compact Edition wurde diese Beschränkung aufgehoben, sodass es endlich ein einheitliches Datenformat für mobile Geräte und Desktop-PCs gab.

Wesentlicher Nachteil bei diesem Produkt ist, dass sich die Compact Edition aufgrund der prozessinternen Nutzung nicht für Mehrbenutzer-Szenarios eignet. Ideal dagegen ist die Compact Edition für selbst entwickelte Desktop- oder Pocket PC- bzw. Smartphone-Anwendungen, die eine Möglichkeit der internen Datenspeicherung benötigen (beispielsweise für die Speicherung von Konfigurationseinstellungen).

Genau wie die Express Edition kann auch die Compact Edition kostenlos heruntergeladen, verwendet und zusammen mit eigenen Anwendungen weiterverteilt werden.

SQL Server Compact Edition wird in Abschnitt *15.3 – Die SQL Server Compact Edition* noch einmal ausführlicher behandelt.

SQL Server Express Edition

Die Express Edition ist ebenfalls kostenfrei erhältlich. Im Gegensatz zur Compact Edition handelt es sich hierbei jedoch um ein vollwertiges Multi-User-Datenbank-Management-System, das auch einen dedizierten Datenbank-Serverdienst beinhaltet. Obwohl der Schwerpunkt auf die lokale Datenspeicherung gelegt wurde, ist die Express Edition auch für Client-Server-Szenarien nutzbar. Damit bildet SQL Server Express (den es seit der 2005er-Generation des SQL Server gibt) das Nachfolgeprodukt der Microsoft Data Engine (MSDE). Die zugrunde liegende Technik ist jedoch dieselbe wie die der größeren SQL Server-Editionen, wodurch eine einfache Skalierbarkeit gewährleistet wird. Eine existierende Datenbank kann also einfach in eine größere Server-Edition »umgehängt« werden, wenn die Leistung der Express Edition einmal nicht mehr ausreichen sollte.

Aber selbst von der Express Edition gibt es mittlerweile drei Untervarianten. Die eigentliche Express Edition beinhaltet keine Administrationsoberfläche. Mit *SQL Server Express with Tools* kommt die Grundvariante von SQL Server Management Studio dazu. *SQL Server Express with Advanced Services* schließlich beinhaltet außerdem die Reporting Services, mit denen sich ansprechende Berichte erstellen und verwalten lassen.

SQL Server Web Edition

Die SQL Server Web Edition ist – wie der Name schon vermuten lässt – speziell für den Einsatz auf Webservern vorgesehen. In den meisten Punkten entspricht der Leistungsumfang in etwa der Workgroup Edition. Lediglich bei den Entwicklungstools sowie dem Management Studio (das auch hier nur in der Grundvariante enthalten ist) orientiert sich die Ausstattung an der Express Edition.

Das Alleinstellungsmerkmal im Vergleich zu den anderen Editionen ist aber sicherlich das Lizenzierungsmodell, da bei dieser Edition nicht nach Clients lizenziert wird (was sich beim Webeinsatz auch schwer ermitteln ließe), sondern ein Betrag pro CPU und Monat der Nutzung anfällt.

SQL Server Workgroup Edition

Die Workgroup Edition ist vor allem für den Einsatz auf Filial- oder Abteilungsservern gedacht, die ihre Daten bei Bedarf mit einem zentralen SQL Server (auf dem dann eine größere Edition des SQL Server läuft) abgleichen können.

Für diese und alle größeren Editionen erfolgt die Lizenzierung wahlweise entweder auf Basis der Prozessoranzahl oder zusammengesetzt aus einem Basispreis plus Gebühren für die Client-Zugriffslizenzen.

SQL Server Standard Edition

Die SQL Server Standard Edition ist – wie der Name schon vermuten lässt – als Standardlösung für produktive Server gedacht. Im Gegensatz zur größeren Enterprise Edition richtet sich die Standard Edition aber primär an kleine und mittelgroße Unternehmen.

Von der Standard Edition gibt es mit *Microsoft SQL Server 2008 Standard Edition for Small Business* noch eine Untervariante, die ein paar zusätzlichen Einschränkungen unterliegt. Die wesentlichste dieser Einschränkungen ist die Begrenzung auf maximal 75 User, was aber für viele kleinere Unternehmen natürlich völlig ausreichend ist.

SQL Server Enterprise Edition

Die Enterprise Edition ist bei SQL Server 2008 (vor R2) die größte Ausführung des SQL Server und vor allem für den Einsatz in großen Unternehmen konzipiert. Selbstverständlich bietet die Enterprise Edition alle verfügbaren Features der kleineren Editionen, wobei vor allem großer Wert auf Ausfallsicherheit und Hochverfügbarkeit gelegt wurde.

Da die Nutzung der Enterprise Edition mit recht hohen Lizenzkosten verbunden ist, gibt es auch eine Evaluationsversion hierzu, mit der man das Produkt 180 Tage lang kostenfrei testen kann. Diese Zeit sollte sicherlich ausreichen, um auch komplexere Testszenarien aufbauen zu können.

Mit SQL Server 2008 R2 erfährt auch die Enterprise Edition einige – wenn auch recht weit gefasste – Beschränkungen bzgl. CPU, Hauptspeicher und maximaler Datenbankgröße. Da sich die Anzahl der zu verwendenden CPUs auf physikalische CPUs (nicht auf Prozessorkerne) bezieht, dürfte auch der auf den ersten Blick etwas niedrig erscheinende Wert von acht CPUs für die meisten Umgebungen völlig ausreichen.

SQL Server Datacenter Edition (nur SQL Server 2008 R2)

Mit SQL Server 2008 R2 wurden zwei neue Premium-Editionen eingeführt, falls die Enterprise Edition doch mal nicht ausreichen sollte. So löst die Datacenter Edition nunmehr die Enterprise Edition als diejenige mit dem größten Funktionsumfang ab. Es gibt quasi keine Beschränkungen mehr für verwendete CPUs (mit Windows Server 2008 R2 sind bis zu 256 CPU-Kerne nutzbar!) oder nutzbaren Hauptspeicher. Dazu kommen ein paar zusätzliche Optionen wie beispielsweise das neue StreamInsight, mit dem sich eine Echtzeit-Datenverarbeitung von komplexen Ereignissen realisieren lässt.

SQL Server Parallel Data Warehouse Edition (nur SQL Server 2008 R2)

Die Parallel Data Warehouse Edition ist vom Funktionsumfang mit der Datacenter Edition vergleichbar, aber auf massiv parallele Verarbeitung von Datenbanken bis zu über einem Petabyte (= 1.000 TB) ausgelegt. Da spätestens hier auch eine optimale Abstimmung der Hardware auf die Anforderungen nötig ist, ist diese Edition nicht als selbst zu installierende Software verfügbar, sondern wird ausschließlich durch Microsoft-Partner als fertig konfigurierte Komplettlösung zusammen mit zertifizierter Hardware angeboten.

Um die massiv parallele Verarbeitung zu nutzen, können mehrere Server wahlweise als sogenannte Control Nodes oder Compute Nodes miteinander arbeiten. Dazu ist eine Partitionierung von extrem großen Tabellen über mehrere Knoten hinweg (Sharding) möglich.

SQL Server Developer Edition

Die Developer Edition stellt wieder eine Besonderheit dar, da sie alle Features der Enterprise Edition beinhaltet, dies jedoch zu einem kleinen Bruchteil der Lizenzkosten. Die Ursache dafür liegt darin, dass die Developer Edition nicht zum produktiven Serverbetrieb lizenziert ist, sondern lediglich zur Entwicklung von Datenbanken bzw. Datenbankanwendungen. Dazu gibt es noch eine weitere Abweichung, welche die Developer Edition interessant macht: Während die Enterprise Edition nur auf den Server-Varianten von Windows eingesetzt werden kann, lässt sich die Developer Edition auch auf den Arbeitsplatzversionen – wie beispielsweise Windows XP – installieren und verwenden.

Daher ist es in vielen Entwicklerteams üblich, dass die Entwickler lokal mit einer Developer Edition entwickeln und lediglich als Staging- und Testserver die größeren Editionen wie beispielsweise Standard oder Enterprise Edition zum Einsatz kommen.

SQL Azure

In Zusammenhang mit der neuen Windows Azure Service Platform wird natürlich auch ein entsprechender Datenbank-Service angeboten. Dieser war ursprünglich unter dem Namen SQL Data Services geplant, wurde inzwischen aber durch SQL Azure abgelöst. In beiden Fällen handelt es sich aber nicht um ein Produkt, das Sie auf einem lokalen Server installieren können, sondern um einen Service, der in einem Microsoft Data Center gehostet wird.

SQL Azure wird in Abschnitt *15.4 – SQL Azure* ausführlicher behandelt.

2.5 Übungen zu diesem Kapitel

In diesem Abschnitt finden Sie einige Übungen zu diesem Kapitel. Die richtigen Antworten, Lösungen und eventuellen Programmcode finden Sie auf der Website *www.vsxpress.de.*

Übung 2.1

Wozu dient der mit SQL Server 2008 neu eingeführte Filestream-Datentyp?

Übung 2.2

Welche der drei verfügbaren Express Editionen (Express, Express with Tools, Express with Advanced Services) können Sie verwenden, wenn Sie die SQL Server Reporting Services nutzen möchten?

Übung 2.3

Wodurch unterscheiden sich die Enterprise und die Developer Edition von SQL Server 2008?

2.6 Zusammenfassung

Nach einem kurzen Überblick über die Historie des Produkts hat dieses Kapitel Sie mit den Neuerungen der 2008er-Generation des SQL Server vertraut gemacht:

- **maximale Skalierbarkeit** – vom Smartphone bis hin zum hochverfügbaren Unternehmensserver
- **neue Datentypen** – *Date, Time, Geometry, Geography, HierarchyId, Filestream*
- **Policy Based Management** – serverübergreifendes richtlinienbasiertes Management
- **IntelliSense** – Unterstützung bei der Eingabe von SQL-Anweisungen
- **T-SQL-Debugging** – Unterstützung bei der Analyse durch Haltepunkte etc.
- **Merge-Anweisung** – SQL-Anweisung, die INSERT und UPDATE (bei Bedarf auch DELETE) in einem Schritt durchführt
- **Table-valued Parameter** – ganze Tabellen können nun als Parameter an eine Stored Procedure übergeben werden

Dazu kommen zahlreiche Neuerungen, die für die größeren Editionen von SQL Server verfügbar sind.

Anschließend haben Sie einen Überblick über die verschiedenen Editionen des SQL Server bekommen:

- **SQL Server Compact Edition** – kostenfreies, prozessintern verwendetes Datenbankmodul, das in Form von Assemblies mit eigenen Anwendungen verteilt werden kann
- **SQL Server Express Edition** – kostenfreie Variante des SQL Server, die vor allem für Privatanwender sowie zum Kennenlernen der SQL Server-Produktfamilie geeignet ist

- **SQL Server Web Edition** – für den Einsatz auf Webservern optimierte Variante, die nicht pro Client sondern ausschließlich pro Prozessor und Nutzungsmonat lizenziert wird
- **SQL Server Workgroup Edition** – Variante des SQL Server, die für kleine Filial- und Abteilungsserver gedacht ist
- **SQL Server Standard Edition** – Standardvariante des SQL Server, die für kleinere Unternehmen konzipiert ist
- **SQL Server Enterprise Edition** – Edition, die für große Unternehmen gedacht und auf Stabilität und optimale Skalierbarkeit ausgelegt ist (maximaler Funktionsumfang für SQL Server 2008)
- **SQL Server Developer Edition** – gleicher Funktionsumfang wie die Enterprise Edition, aber nur zu Entwicklungszwecken lizenziert (bei SQL Server 2008 R2 entspricht der Funktionsumfang der Developer Edition dem der Datacenter Edition)
- **SQL Server Datacenter Edition (nur für SQL Server 2008 R2)** – Edition mit dem maximalen Funktionsumfang, die für große Unternehmen gedacht und auf Hochverfügbarkeit ausgelegt ist
- **SQL Server Parallel Data Warehouse Edition (nur für SQL Server 2008 R2)** – vom Funktionsumfang mit der Datacenter Edition vergleichbar, wird aber als Komplettlösung zusammen mit zertifizierter Hardware angeboten
- **SQL Azure** – Cloud-basierte Variante des SQL Server

Nun wird es Zeit, die Express Edition von SQL Server zu installieren, damit Sie das Produkt gleich ausprobieren können.

Hinweis: SQL 2008 oder SQL 2008 R2?

Die meisten in diesem Buch beschriebenen Inhalte beziehen sich sowohl auf SQL Server 2008 als auch auf SQL Server 2008 R2. Daher wird der Einfachheit halber als Produkt nur SQL Server 2008 genannt. Lediglich dann, wenn ein beschriebenes Feature ausschließlich für die neuere SQL Server 2008 R2-Version verfügbar ist, wird an den entsprechenden Stellen ausdrücklich darauf hingewiesen.

Kapitel 3

Erste Schritte mit SQL Server 2008 Express

In diesem Kapitel lernen Sie

- welche Systemvoraussetzungen für die Installation von SQL Server 2008 Express erfüllt sein müssen
- welche Einstellungen bei der Installation vorzunehmen sind
- welches die wichtigsten Tools für SQL Server sind und was Sie damit machen können

3.1 Systemvoraussetzungen

Um SQL Server installieren zu können, muss eine Reihe von Systemvoraussetzungen gegeben sein, die sich sowohl auf die Hardware als auch die Software Ihres PCs beziehen. Je nachdem, welche Variante von SQL Server Express Sie installieren möchten, unterscheiden sich diese ein wenig. Sofern aber im Folgenden nicht explizit darauf hingewiesen wird, gelten die Beschreibungen für alle drei Varianten (SQL Server 2008 Express, SQL Server 2008 Express with Tools und SQL Server 2008 Express with Advanced Services). Auch für SQL Server 2008 R2 Express haben sich die Systemvoraussetzungen nur minimal geändert, wie in der Auflistung unten zu sehen ist.

Hardwarevoraussetzungen

Für 32-Bit-Systeme gilt:

- Computer mit Intel- oder kompatiblem 1-GHz- oder schnellerem Prozessor (2 GHz oder schneller wird empfohlen. Nur ein einzelner Prozessor wird unterstützt.)

Für 64-Bit-Systeme gilt:

- Prozessor mit 1,4 GHz oder schneller (2 GHz oder schneller wird empfohlen. Nur ein einzelner Prozessor wird unterstützt.)

Für SQL Server 2008 Express (auf 32- oder 64-Bit-Systemen):

- Mindestens 256 MB RAM (1 GB oder mehr wird empfohlen)
- 1 GB freier Festplattenspeicherplatz

Für SQL Server 2008 Express with Tools oder SQL Server 2008 Express with Advanced Services (auf 32- oder 64-Bit-Systemen):

- Mindestens 512 MB RAM (1 GB oder mehr wird empfohlen)
- 1,9 GB freier Festplattenspeicherplatz

Für SQL Server 2008 R2 Express, Express with Tools oder Express with Advanced Services (auf 32- oder 64-Bit-Systemen):

- Mindestens 512 MB RAM (2 GB oder mehr wird empfohlen)
- 2,2 GB freier Festplattenspeicherplatz

Generell muss man die Prozessorfrequenz nicht ganz so genau nehmen, wichtiger sind der freie Plattenplatz, damit das Produkt überhaupt installiert werden kann, sowie die Angaben zum Hauptspeicher, damit SQL Server auch in vertretbarer Geschwindigkeit ausgeführt wird.

Will man die SQL Server-Performance durch Aufrüstung der Hardware steigern, so lässt sich das am einfachsten durch Aufrüsten des Hauptspeichers erreichen.

Softwarevoraussetzungen

Im Gegensatz zu den größeren Varianten von SQL Server, die (mit Ausnahme der Developer Edition) nur auf den Windows Server-Varianten lauffähig sind, können Sie den SQL Server 2008 Express auch auf den gängigen Desktopbetriebssystemen verwenden.

Unterstützte Betriebssysteme:

- Windows Server 2003 mit Service Pack 2
- Windows Server 2008 (für SQL 2008 R2 mit Service Pack 2)
- Windows XP mit Service Pack 2 oder Service Pack 3
- Windows Vista mit/ohne Service Pack 1 (für SQL 2008 R2 mit Service Pack 2)
- Windows 7

Hinweis: Betriebssysteme und Service Packs

Je nach Quelle finden sich unterschiedliche Angaben zu den benötigten Service Packs. Generell ist es jedoch zu empfehlen, immer das aktuellste final verfügbare (also keine Betas!) Service Pack zum verwendeten Betriebssystem zu installieren.

Neben dieser Grundvoraussetzung sind noch ein paar weitere Komponenten erforderlich, die allesamt auf der Buch-DVD enthalten sind.

- .NET Framework 3.5 SP1
- Windows Installer 4.5 Redistributable
- Windows PowerShell 1.0

Auch wenn die Installation der hier genannten Komponenten vom Setup-Programm überprüft wird, ist es sinnvoll, diese vorher zu installieren, um Zeit bei der eigentlichen Installation des SQL Server zu sparen.

Die Installation dieser drei Komponenten ist selbsterklärend. Lediglich die Lizenzbedingungen müssen akzeptiert werden. Beim Windows Installer und der Windows PowerShell ist jedoch zu beachten, dass die richtige Version installiert wird, da es hier – je nach Betriebssystem und Prozessortyp – verschiedene Varianten gibt.

3.2 Installation

Nachdem die entsprechenden Voraussetzungen erfüllt sind, kann endlich die eigentliche Installation von SQL Server beginnen. Je nach benötigter Funktionalität haben Sie drei verschiedene Varianten von SQL Server 2008 Express zur Auswahl. Sofern von Unternehmensseite nichts dagegen spricht, wird die Installation der umfangreichsten Variante (SQL Server 2008 Express with Advanced Services) empfohlen, die auch im Folgenden beschrieben wird. Bei den kleineren Varianten fällt der eine oder andere Installationsschritt weg.

Dazu ist zu entscheiden, ob Sie das ältere SQL Server 2008 Express oder das neuere SQL Server 2008 R2 Express verwenden, die sich auf der Buch-DVD in unterschiedlichen Ordnern befinden. Die Installation beider Varianten unterscheidet sich jedoch nur in Details.

Installation der Advanced Edition

Auf der Buch-DVD befinden sich Installationsordner für sowohl SQL Server 2008 Express (*\SQLServer2008*) als auch für SQL Server 2008 R2 Express (*\SQLServer2008R2*). In diesen Ordnern liegt jeweils die Datei *SQLEXPRADV_x86_DEU.exe*, mit der Sie die Installation der Advanced Edition starten können (die entsprechende Datei für SQL Server 2008 Express heißt *SQLEXPR_x86_DEU.exe*, die für SQL Server 2008 Express with Tools *SQLEXPRWT_x86_DEU.exe*).

Internet-Link: SQL Server 2008 Express zum Download

Wenn Sie die aktuellste Version von SQL Server 2008 Express aus dem Internet herunterladen möchten, können Sie den Softlink **sql0301** nutzen. Auf derselben Website können Sie SQL Server 2008 Express auch online registrieren.

Um den Softlink zu nutzen, geben Sie auf der Startseite von *http://www.vsxpress.de* die angegebene Link-ID ein und klicken dann auf *Link öffnen*. Alternativ können Sie auch direkt in der Adressleiste Ihres Browsers die folgende URL eingeben und dabei *id* durch die angegebene Link-ID ersetzen:

http://go.vsxpress.de/?linkid=sql0301

Hinweis: 32- und 64-Bit Versionen

Aus Platzgründen sind auf der Buch-DVD nur die jeweiligen 32-Bit-Versionen der Installationsdateien enthalten (erkennbar an der Endung x86). Sollten Sie die 64-Bit-Variante installieren wollen, finden Sie die Links zu den entsprechenden Dateien auf der Microsoft-Produktseite, die über den oben angegebenen Softlink erreichbar ist.

Nach einiger Zeit – in der die Installationsdateien entpackt und in einem temporären Verzeichnis auf der Festplatte abgelegt werden – erscheint dann das SQL Server-Installationscenter. Über das Menü auf der linken Seite lassen sich verschiedene Bereiche aufrufen, die jeweils verschiedene Auswahlmöglichkeiten bieten. Angefangen von der Vorbereitung einer Installation (*Planen*), über die Installation selbst und die Wartung danach bis hin zu einer Übersicht mit Links auf weitere Informationsquellen zum Thema.

Wenn Sie sichergehen wollen, können Sie die auf der ersten Seite angebotene Systemkonfigurationsprüfung ausführen oder aber gleich auf der zweiten Seite des Installationscenters mit der eigentlichen Installation beginnen.

Abbildung 3.1: Das SQL Server-Installationscenter

Nachdem wiederum einige Tests zur Systemkonfiguration (hier Setupunterstützungsregeln genannt) durchgeführt wurden, folgt ein Dialogfeld, bei dem Sie normalerweise den Product Key eingeben müssten. Da es sich bei der Express Edition aber um eine lizenzgebührenfreie Version von SQL Server handelt, können Sie dieses Dialogfeld einfach mit *Weiter* überspringen. Anschließend werden Sie dazu aufgefordert, die Lizenzbedingungen zu bestätigen, bevor es mit der Auswahl der zu installierenden Features weitergeht.

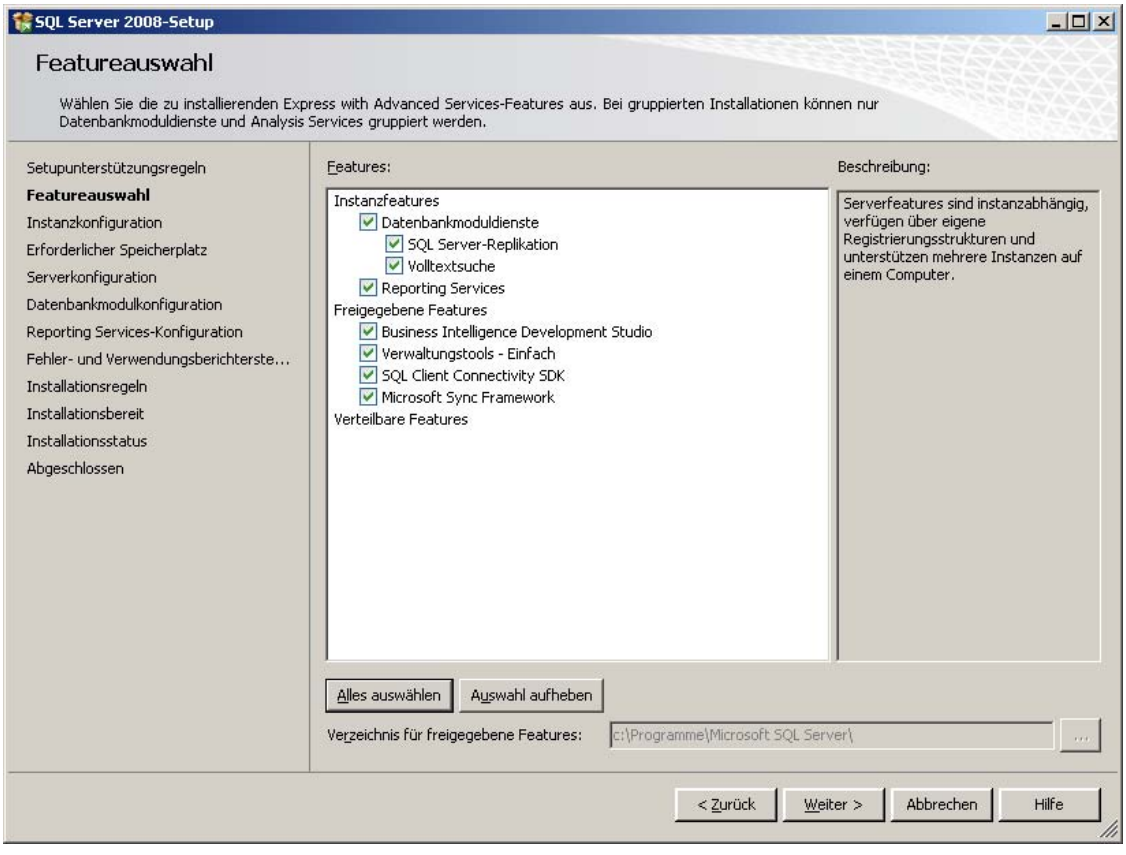


Abbildung 3.2: Die Auswahl der zu installierenden Features

Die Datenbankmoduldienste sind Grundvoraussetzung, wenn Sie mit dem SQL Server arbeiten wollen. Die Features SQL Server-Replikation, Volltextsuche und Reporting Services sind optional, einige davon werden aber später in diesem Buch noch benötigt, sodass Sie auch diese auswählen sollten.

Unter *Freigegebene Features* sind diverse Zusatzoptionen auswählbar, von denen zumindest die einfachen Verwaltungstools (dahinter verbirgt sich das SQL Server Management Studio Express) auf jeden Fall installiert werden sollten. Das Business Intelligence Development Studio benötigen Sie, um Reports für die Reporting Services zu erstellen, während es sich bei dem SQL Client Connectivity SDK und dem Microsoft Sync Framework um Tools für die Verbindung zu anderen Datenbankservern handelt.

Wenn Sie genügend Platz auf der Festplatte haben, empfiehlt es sich, alle diese Tools gleich mitzuinstallieren, denn eine spätere Nachinstallation einzelner Bestandteile ist zwar möglich, kostet aber immer viel Zeit.

Nach der Auswahl der freigegebenen Features, für die Sie auch ein Zielverzeichnis angeben können, geht es weiter mit der Konfiguration der SQL Server-Instanz.

Hintergrundinfo: SQL Server-Instanzen

Die Begriffe Server, SQL Server und SQL Server-Instanz werden oft durcheinandergebracht, daher finden Sie an dieser Stelle eine Klärung der Bedeutungen:

- **Server** bezeichnet normalerweise die Gesamtheit aus einer Serverhardware, mit darauf laufendem Serverbetriebssystem (also beispielsweise Windows Server 2003 oder 2008).
- **SQL Server** wird im alltäglichen Sprachgebrauch oft für die Gesamtheit aus SQL Server-Software und der Hardware, auf der diese läuft, verwendet. Streng genommen bezeichnet der Begriff SQL Server allerdings nur den SQL Server-Dienst, der auf dieser Hardware läuft und die Datenbank bzw. den Zugriff darauf zur Verfügung stellt. Verwirrung kann insbesondere dadurch aufkommen, dass oft die Versionsnummern für den SQL Server und das Betriebssystem, das auf der Serverhardware läuft (also beispielsweise Windows 2003 Server oder Windows 2008 Server) durcheinandergebracht werden.
- **SQL Server-Instanz** ist als Begriff schwieriger klar verständlich zu definieren. Im Prinzip geht es dabei darum, dass auf einer Hardware mehrere SQL Server-Instanzen (derselben oder unterschiedlicher Editionen und Versionen) laufen können, die unabhängig voneinander agieren. Dies wird beispielsweise gerne genutzt, um Entwicklungs- und Testumgebung klar voneinander zu trennen, auch wenn es dafür nur eine Serverhardware gibt. Dazu erhalten die SQL Server-Instanzen separate Namen. Der Zugriff auf diese benannten Instanzen erfolgt dann durch Angabe des Rechnernamens (oder dessen IP-Adresse) gefolgt vom Namen der Instanz. Eine einzige Instanz pro Rechner kann als Standardinstanz definiert werden, für die dann die Angabe des Rechnernamens (oder dessen IP-Adresse) ausreicht. Dies ist auch der Standardfall, wenn Sie auf einem Rechner nur eine Instanz installieren, wobei es prinzipiell zu empfehlen ist, generell benannte Instanzen zu verwenden.

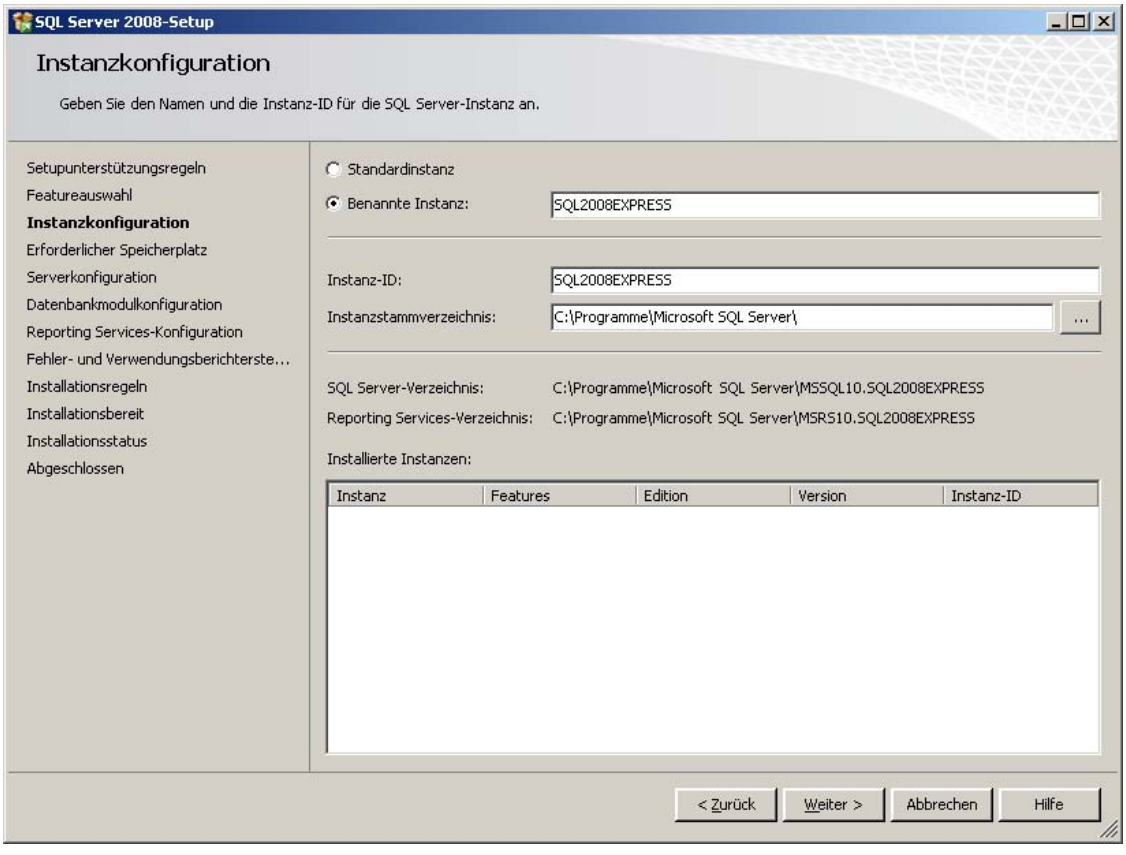


Abbildung 3.3: Die Instanzkonfiguration

Im oberen Bereich des Dialogfelds können Sie auswählen, ob es sich bei dem zu installierenden SQL Server um die Standardinstanz oder eine benannte Instanz handeln soll und im letzteren Fall gleich den Instanznamen angeben. Falls irgendwann einmal mehrere Instanzen auf dem Rechner installiert sein sollten, ist es sinnvoll, im Instanznamen sowohl die Version als auch die Edition des SQL Server unterzubringen, in diesem Fall also beispielsweise **SQL2008EXPRESS**. Der Instanzname wird auch automatisch als Instanz-ID übernommen, was sich zwar ändern lässt, aber nicht nötig ist. Zusätzlich zu Instanzname und Instanz-ID können Sie noch das Stammverzeichnis für die SQL Server-Instanz angeben, aus dem sich auch die Verzeichnisse für den SQL Server und die Reporting Services ergeben, die darunter angezeigt werden. Zur Übersicht über bereits vorhandene SQL Server-Instanzen sind diese im unteren Bereich des Fensters aufgelistet.

Nach diesem Schritt folgt eine Anzeige des benötigten Speicherplatzes auf dem Systemlaufwerk, dem Installationsverzeichnis und dem Instanzverzeichnis. Bei Bedarf können Sie über die entsprechende Schaltfläche einen Schritt zurückgehen und ein anderes Verzeichnis für die Installation auswählen.

Anschließend geht es weiter mit der Konfiguration der Serverdienste.

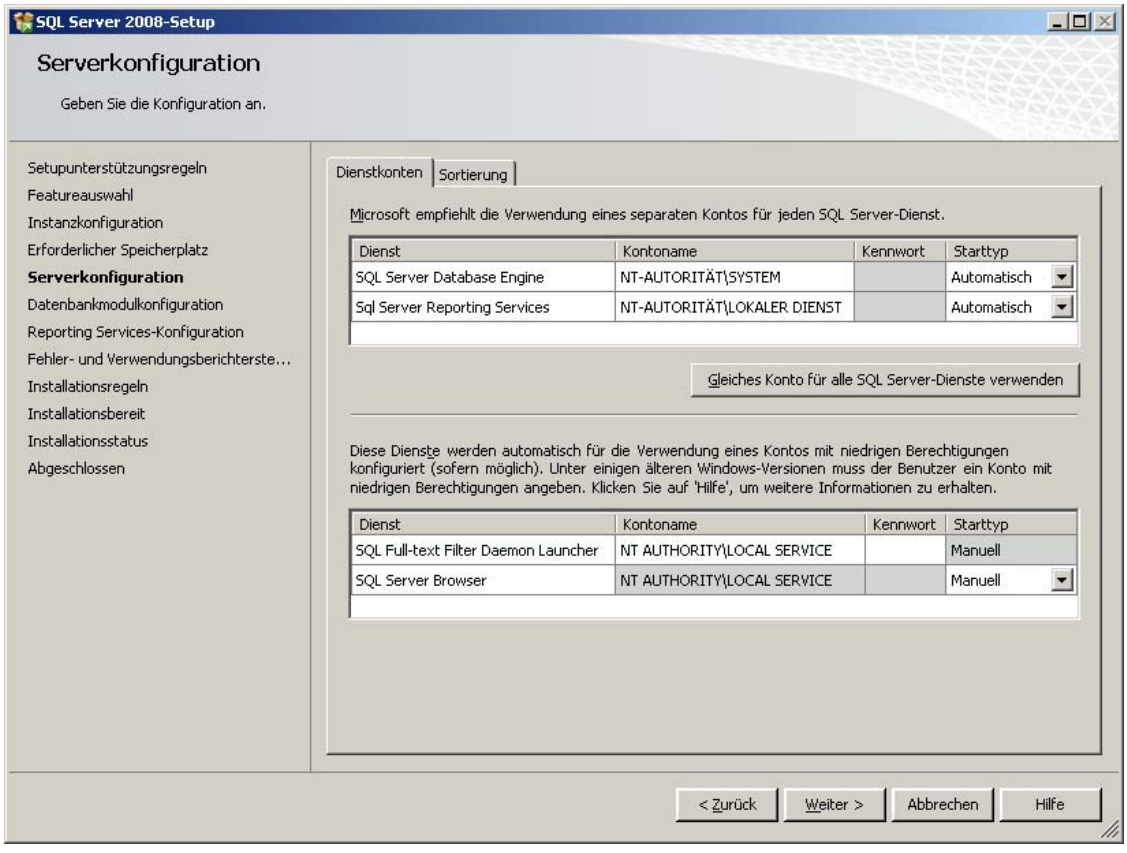


Abbildung 3.4: Konfiguration der Serverdienste

Hier ist anzugeben, wie mit den vom SQL Server benötigten Diensten verfahren werden soll. Dies sind die SQL Server Database Engine (der eigentliche SQL Server-Prozess), die SQL Server Reporting Services (die zur Bereitstellung von Berichten dienen), der SQL Full-text Filter Daemon Launcher (wird im Zusammenhang mit der Volltextsuche benötigt) und der SQL Server Browser.

Hintergrundinfo: Zusätzliche Dienste der größeren SQL Server-Editionen

Wenn Sie eine größere Edition von SQL Server installieren, erscheinen an dieser Stelle natürlich auch die anderen Dienste (sofern Sie die entsprechenden Features zur Installation ausgewählt haben). Dies sind der SQL Server Agent, die SQL Server Analysis Services sowie die SQL Server Integration Services.

Für jeden dieser Dienste ist der Starttyp auszuwählen, der angibt, ob der Dienst beim Systemstart (also bereits bevor ein Benutzer angemeldet ist) automatisch gestartet wird, manuell gestartet werden muss oder komplett deaktiviert ist. Zumindest der SQL Server-Dienst sollte auf automatisch gestellt werden (bei Bedarf auch die Reporting Services), die anderen Dienste können auf manuell eingestellt werden.

Außerdem muss für jeden Dienst ein Benutzerkonto angegeben werden, unter dem (also auch mit dessen Berechtigungen) der Dienst läuft. Für produktive Datenbankserver in großen Netzwerken wird generell empfohlen, separate User für jeden Dienst zu erstellen und diesen gezielt nur die Berechtigun-

gen zuzuweisen, die für den jeweiligen Dienst benötigt werden. (In diesem Fall wären hier auch die entsprechenden Kennwörter mit anzugeben.) Da wir es hier aber mit einem lokalen Datenbankserver zu tun haben, reicht es völlig aus, Standardbenutzerkonten zu verwenden. Verwenden Sie also für den SQL Server-Dienst das *System*-Benutzerkonto und für alle anderen das Benutzerkonto *Lokaler Dienst* (das für den SQL Server-Dienst selbst leider nicht verwendet werden kann).

Hintergrundinfo: Standardbenutzerkonten

Sofern Sie keine selbst erstellten Benutzerkonten verwenden, stehen Ihnen folgende Standardbenutzerkonten zur Verfügung:

- **System (Local System)** ist ein lokales Systemkonto, das über weitreichende Rechte auf dem Rechner selbst, aber nicht im Netzwerk verfügt.
- **Lokaler Dienst (Local Service)** ist vergleichbar mit Local System, allerdings mit dem Unterschied, dass dieses Konto keine Berechtigung zur Anmeldung an die Benutzeroberfläche des Betriebssystems hat, da es speziell für die Nutzung von Diensten vorgesehen ist.
- **Netzwerkdienst (Network Service)** ist ein Systemkonto, das im Gegensatz zu den beiden zuvor genannten auch über die Berechtigung für Netzwerkzugriffe verfügt.

Egal für welche Benutzerkonten und Starttypen Sie sich hier entscheiden, können Sie diese Einstellungen später problemlos wieder ändern, ohne eine erneute Installation durchführen zu müssen.

Auch beim nächsten Schritt geht es um Konten und Berechtigungen, allerdings nun darum, mit welchen Konten sich eine Anwendung oder ein Benutzer an den Datenbankserver anmelden kann. Hier sind zwei verschiedene Arten der Authentifizierung möglich:

- **Windows-Authentifizierung** bedeutet, dass die Authentifizierung durch das Betriebssystem erfolgt, also Benutzer und Passwörter des Betriebssystems verwendet werden, um sicherzustellen, dass der angemeldete Benutzer auch der ist, für den er sich ausgibt.
- **SQL Server-Authentifizierung** bedeutet, dass die Authentifizierung durch ausschließlich vom SQL Server verwaltete Benutzer und Passwörter erfolgt. Allerdings lässt sich dieser Authentifizierungsmodus nicht allein auswählen, sodass er lediglich zusammen mit der Windows-Authentifzierung als sogenannter *Gemischter Modus* ausgewählt werden kann.

Wenn Sie den gemischten Modus verwenden, so ist noch ein Kennwort für den SQL-Systemadministrator-User anzugeben. Dieser Benutzer hat den Benutzernamen *sa* und wird automatisch nach der Installation angelegt, sofern Sie den gemischten Authentifizierungsmodus nutzen.

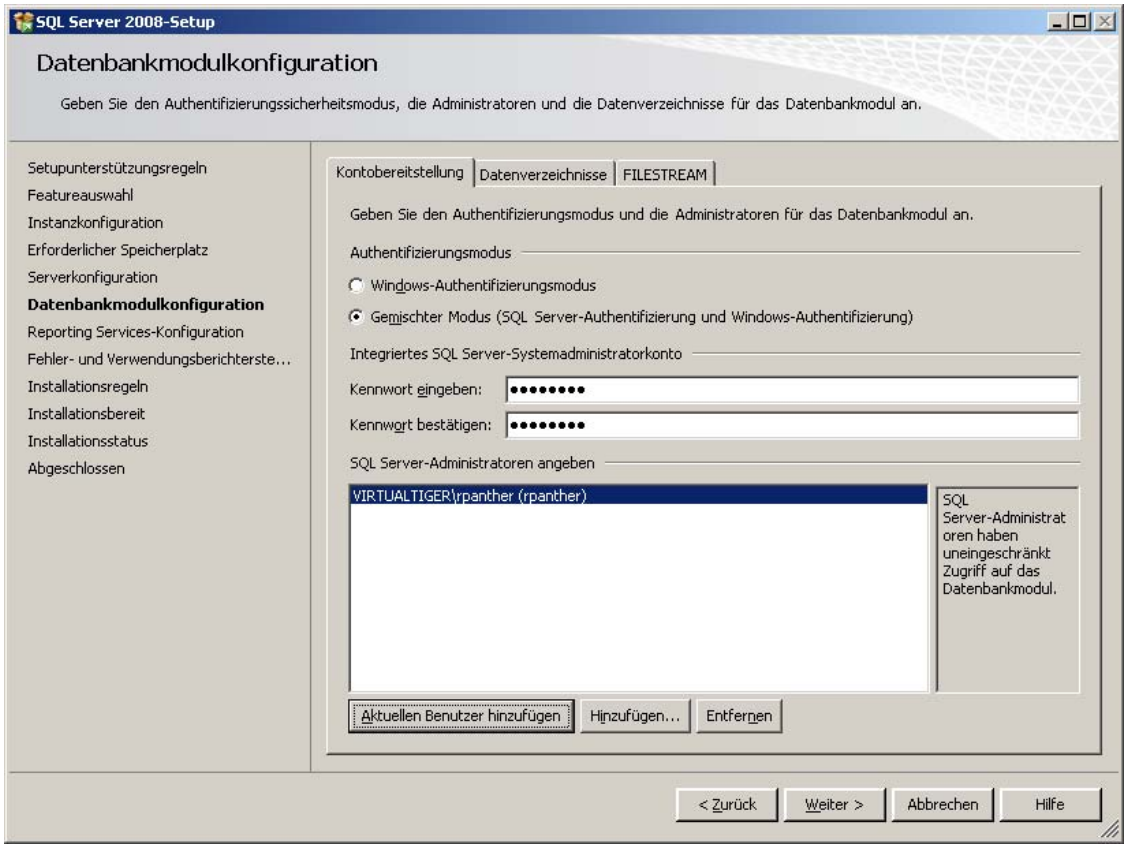


Abbildung 3.5: Auswahl des Authentifizierungsmodus und Konfiguration der Administratoren

Unabhängig vom gewählten Authentifizierungsmodus empfiehlt es sich, auch einen Betriebssystembenutzer zum Administrator zu machen. Im Normalfall wird man hier über die Schaltfläche *Aktuellen Benutzer hinzufügen* den Benutzer zum Administrator machen, der auch die Installation gestartet hat. Alternativ lassen sich aber auch ein oder mehrere andere Benutzer angeben bzw. auswählen.

Auf der Registerkarte *Datenverzeichnisse* können Sie verschiedene Verzeichnisse für Benutzerdatenbanken, Benutzerdatenbankprotokolle, temporäre Datenbanken (und deren Protokolle) sowie Datenbanksicherungen angeben. Auf einem produktiven Datenbankserver mit mehreren Festplattensystemen macht es aus Performance-Gründen auch Sinn, die gerade genannten Verzeichnisse auf getrennte Platten zu verteilen. Auf einem »normalen« PC sieht das natürlich anders aus, sodass Sie hier lediglich das Datenstammverzeichnis auf eine zweite Platte (oder Partition) legen sollten, sofern Sie über eine solche verfügen. Die anderen Verzeichnisse werden dann entsprechend darunter gehängt.

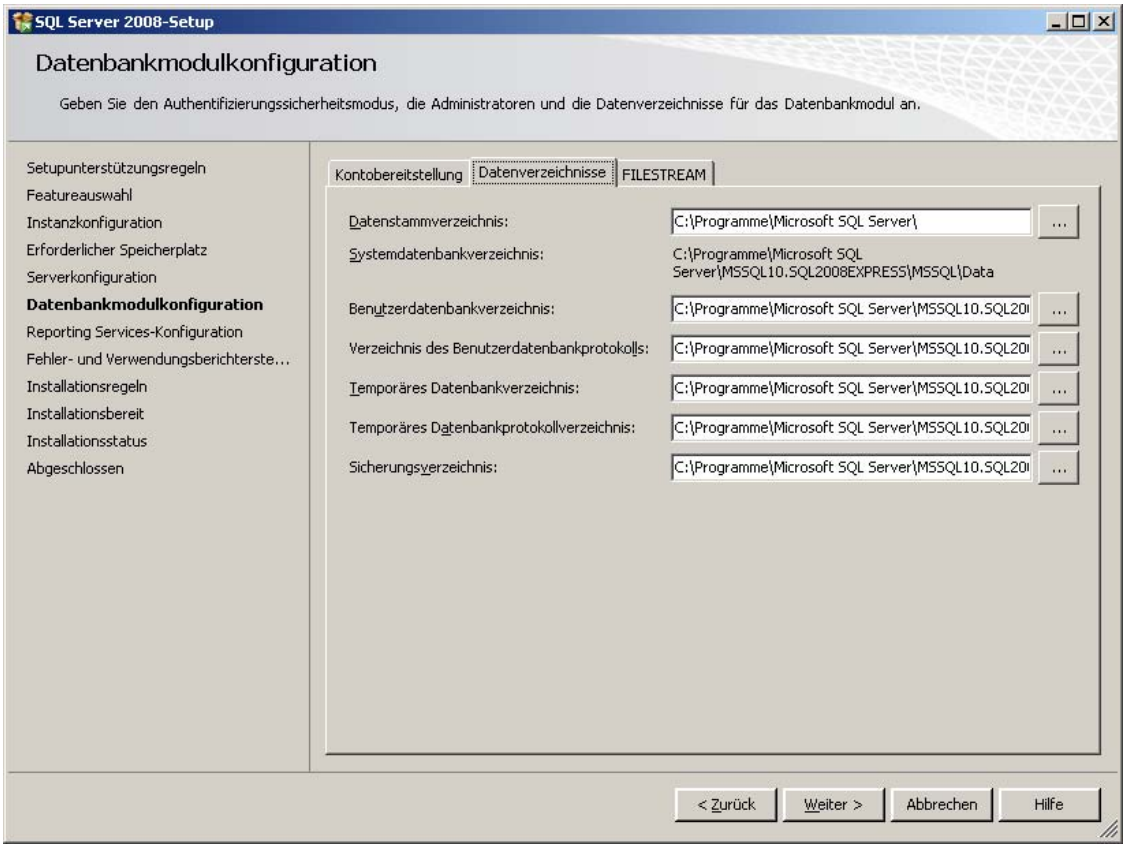


Abbildung 3.6: Konfiguration der zu verwendenden Verzeichnisse

Die Einstellungen auf der Registerkarte *Filestream* können Sie unverändert übernehmen, sodass im nächsten Schritt die Konfiguration der Reporting Services auszuwählen ist. Der Einfachheit halber belassen Sie es hier bei der Voreinstellung *Standardkonfiguration des systemeigenen Modus*.

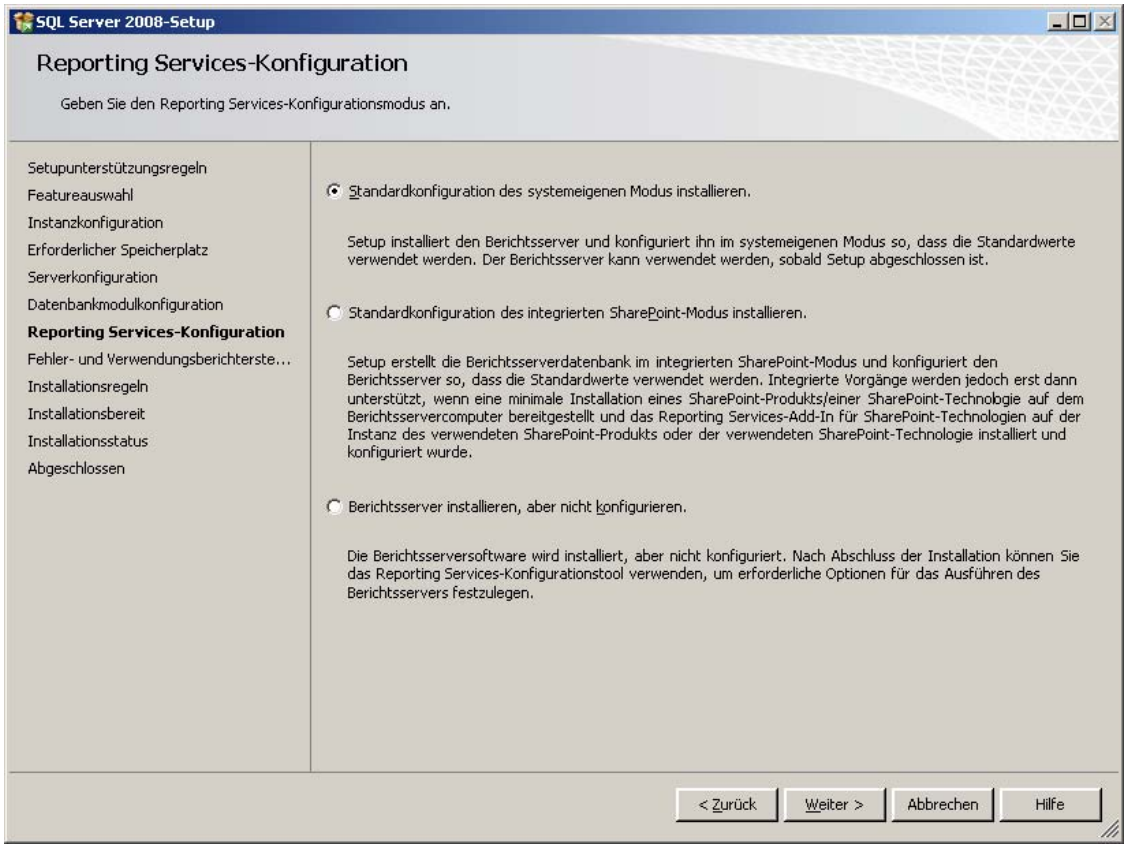


Abbildung 3.7: Auswahl des Konfigurationsmodus für die Reporting Services

Nachdem Sie sich im nächsten Schritt entschieden haben, ob der SQL Server Windows und SQL Server-Fehlerberichte sowie anonyme Daten zur Featureverwendung an Microsoft senden darf, erfolgt eine weitere Prüfung von einigen Installationsvoraussetzungen (hier Installationsregeln genannt).

Sofern hier alle Voraussetzungen erfüllt sind, folgt eine Zusammenfassung, mit der Sie alle vorgenommenen Installationseinstellungen noch einmal überprüfen können. Anschließend startet die eigentliche Installation von SQL Server.

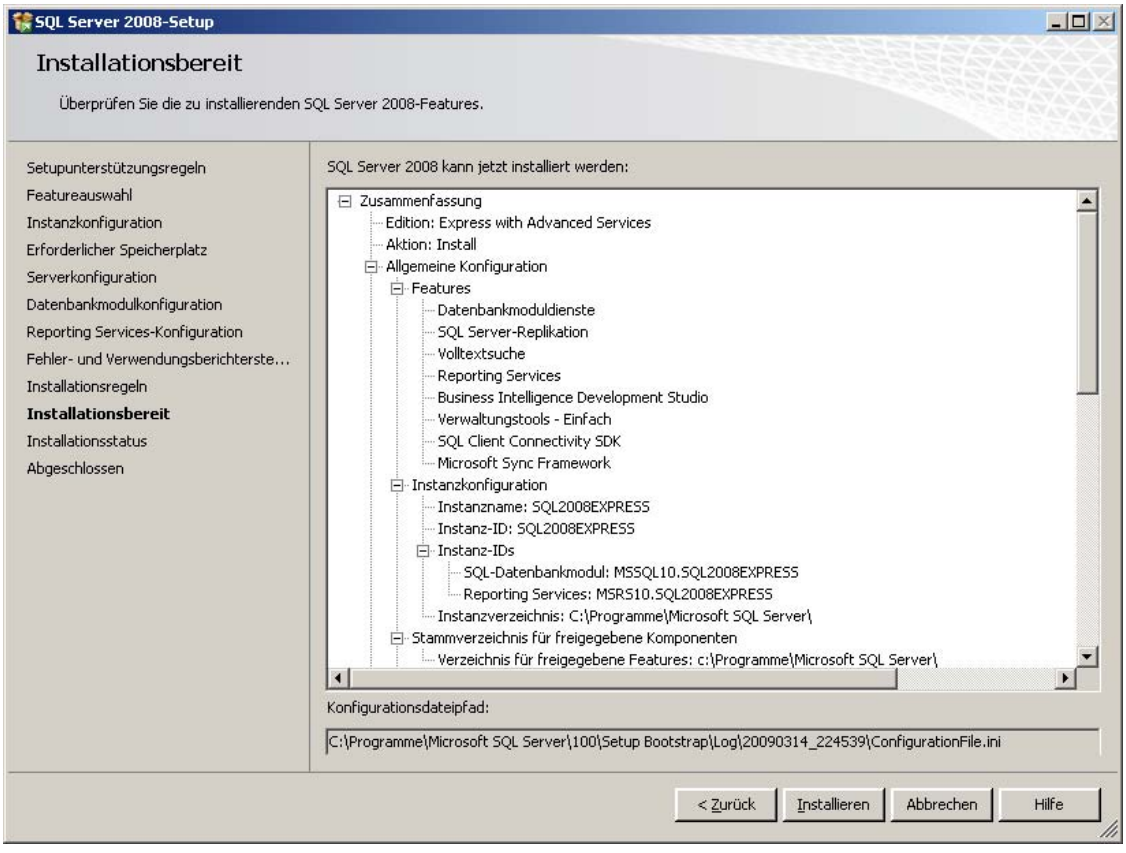


Abbildung 3.8: Die Zusammenfassung der Installationseinstellungen

Aktualisieren von SQL Server

Nach der Installation von SQL Server sollten Sie diesen noch auf den aktuellen Stand bringen. Dazu bietet Microsoft in der Regel kleinere Updates an, die von Zeit zu Zeit zu kumulativen Updates oder gar Service Packs zusammengefasst werden. Dadurch entfällt das lästige Installieren von zahlreichen Einzelupdates, sodass nur ein einziges Gesamtupdate eingespielt werden muss.

Aktueller Stand zu dem Zeitpunkt, zu dem ich dieses Kapitel schreibe, ist das Service Pack 1, das Sie ebenfalls auf der Buch-DVD finden (Dateiname: *SQLServer2008SP1-KB968369-x86-DEU.exe*). Die Größe von etwa 275 MB resultiert daraus, dass diese Datei auch die Aktualisierungen für die anderen Editionen des SQL Server beinhaltet. Alternativ gibt es auch eine aktualisierte Express Edition in die bereits das Service Pack integriert ist. Diese ist allerdings weder für Express Edition with Tools noch für Express Edition with Advanced Services, sondern nur für die Grundvariante der Express Edition erhältlich. Da in diesem Buch die Express Edition with Advanced Services verwendet wird, muss also das große Gesamtupdate installiert werden.

Nachdem Sie die oben genannte Datei gestartet haben, läuft die Installation weitgehend selbsterklärend ab. Sie müssen lediglich die Lizenzbedingungen akzeptieren und bestätigen, welche Komponenten von

SQL Server 2008 Express aktualisiert werden sollen. Vor der eigentlichen Installation erfolgt noch eine kurze Zusammenfassung. Dabei zeigt sich, dass automatisch erkannt wurde, welche Edition und Bestandteile des SQL Server installiert sind, sodass Sie davon ausgehen können, dass nur installierte Komponenten aktualisiert werden und nicht solche, die von der Express Edition ohnehin nicht genutzt werden können. Wenn Sie nun auf *Update* klicken, läuft die eigentliche Installation des Service Packs dann relativ schnell durch und schon ist Ihr SQL Server auf dem aktuellen Stand.

Hinweis: Prüfen auf aktuellere Updates

Eventuell sind zu dem Zeitpunkt, wenn Sie diesen Text lesen, bereits weitere Updates verfügbar. Prüfen Sie daher sicherheitshalber über die entsprechende Option im SQL Server-Installationscenter nach, ob es bereits neuere Updates und Service Packs für den SQL Server 2008 bzw. SQL Server 2008 R2 gibt. Alternativ können Sie die Update-Funktion auch über das Startmenü aufrufen: *Start/Alle Programme/Windows Update*

3.3 Die wichtigsten SQL Server-Tools

Nachdem nun SQL Server installiert ist, möchte ich Ihnen kurz die wichtigsten Tools vorstellen, damit Sie den SQL Server auch gleich ausprobieren können.

SQL Server-Installationscenter

Das SQL Server-Installationscenter haben Sie bereits bei der gerade beschriebenen Installation des SQL Server kennengelernt. Dieses Programm wird aber auch nach einer erfolgreichen Installation eventuell noch einmal benötigt, um beispielsweise eine beschädigte Installation (falls beispielsweise versehentlich eine Datei gelöscht wurde) wieder zu reparieren oder aber um auf eine größere Edition umzusteigen.

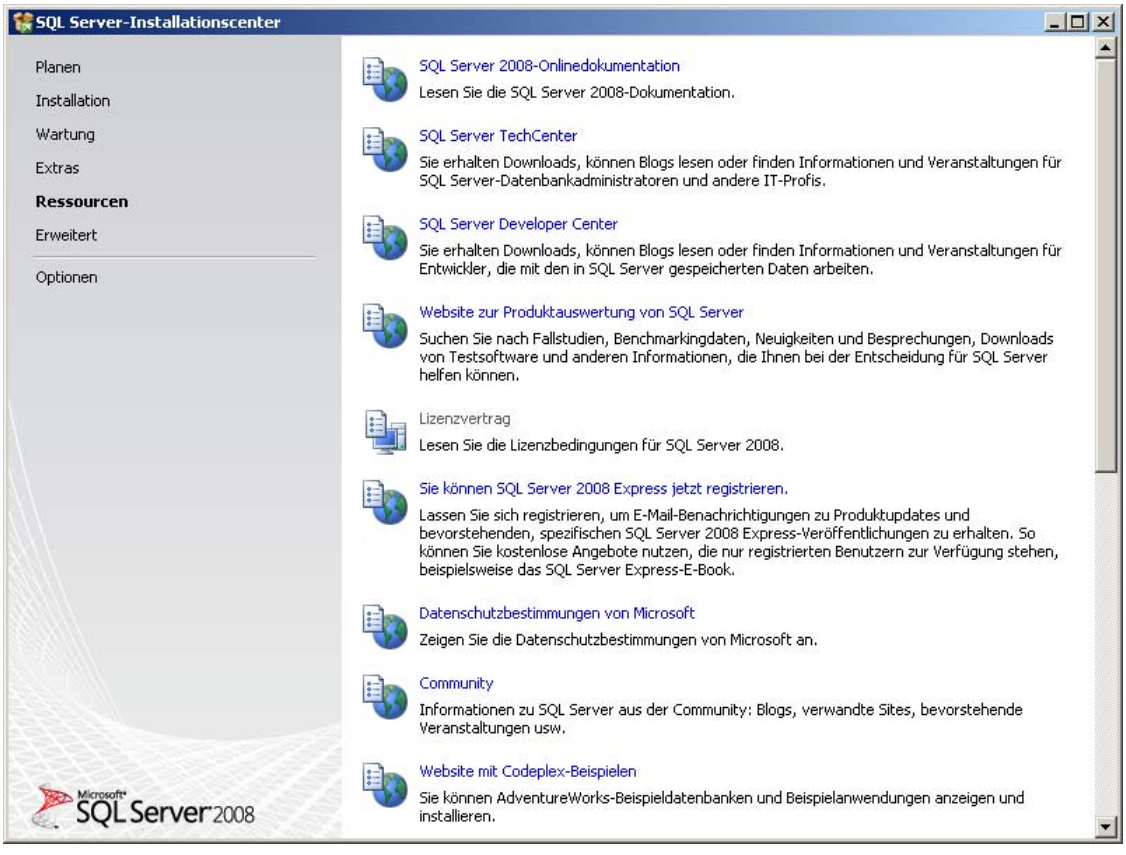


Abbildung 3.9: Die Zusammenstellung der verschiedenen SQL Server-Ressourcen im Installationscenter

Auf der Seite *Ressourcen* finden Sie eine ganze Auflistung an Links unter denen weitere Informationen zum SQL Server zu finden sind. Angefangen von der Onlinedokumentation, über diverse Websites mit Downloads und Zusatzinformationen bis hin zu offiziellen Dokumenten wie beispielsweise dem Lizenzvertrag, dem Sie die genauen Bestimmungen zur Nutzung von SQL Server entnehmen können.

SQL Server-Konfigurations-Manager

Dieses Tool dient – wie der Name schon andeutet – zur Konfiguration Ihrer SQL Server-Installation. Dabei ist der Konfigurations-Manager genau genommen kein eigenes Tool, sondern bedient sich der Microsoft Management Console, um darin die wichtigsten SQL Server-Konfigurationseinstellungen zu verwalten. Auf diesem Weg können Sie beispielsweise einige bei der Installation vorgenommene Einstellungen prüfen und gegebenenfalls anpassen.

Nach dem Aufruf des Konfigurations-Managers bekommen Sie in einer Baumstruktur auf der linken Seite drei Hauptbereiche zu sehen, auf die ich im Folgenden näher eingehen will.

SQL Server-Dienste

Hier werden alle Dienste aufgeführt, die vom SQL Server verwendet werden. Dazu sehen Sie jeweils den aktuellen Status, den Startmodus sowie den Account, unter dem der Dienst läuft. Sie erinnern sich sicherlich, dass Sie diese Einstellungen während der Installation vorgeben mussten.

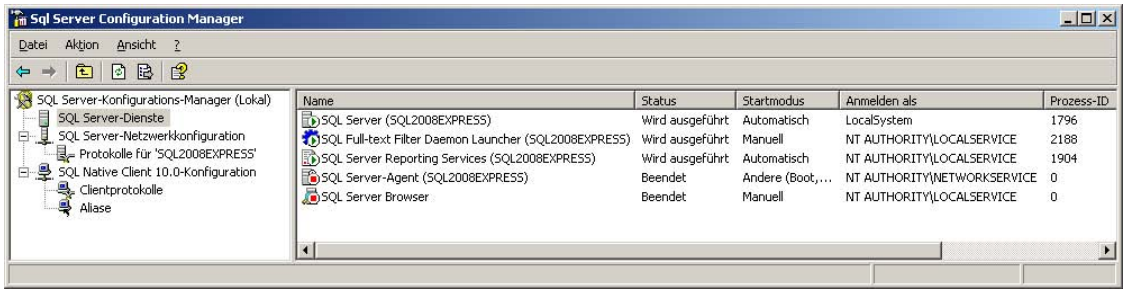


Abbildung 3.10: Die Anzeige der SQL Server-Dienste im Konfigurations-Manager

Die Starttypen haben folgende Bedeutung:

- **Automatisch** – Der Dienst wird bei jedem Systemstart automatisch ausgeführt (auch wenn keine Benutzeranmeldung erfolgt).
- **Manuell** – Der Dienst wird erst dann gestartet, wenn Sie das explizit tun.
- **Deaktiviert** – Der Dienst ist inaktiv und lässt sich nicht starten, bevor der Starttyp geändert wurde.

Über einen Rechtsklick auf einen der Einträge können Sie den jeweiligen Dienst starten oder beenden oder alternativ die Eigenschaften aufrufen, bei denen Sie die anderen oben genannten Einstellungen anpassen können.

Hinweis: Alternative zur Anzeige der ausgeführten Dienste

Die Dienste einstellungen können Sie auch über den vom Betriebssystem bereitgestellten Weg (Windows-Startmenü, dann Einstellungen/Systemsteuerung/Verwaltung/Dienste) vornehmen, allerdings sehen Sie dann alle Dienste (nicht nur die des SQL Server), sodass es hier deutlich schwerer ist, die Übersicht zu behalten.

SQL Server-Netzwerkkonfiguration

Unter diesem Punkt finden Sie einen Eintrag für jede lokal installierte SQL Server-Instanz. Klicken Sie eine davon an (zumindest die installierte SQL Server 2008 Express-Instanz sollte hier auftauchen), werden im Detailbereich auf der rechten Seite die verfügbaren Kommunikationsprotokolle mit deren Status angezeigt. Dies sind:

- **Shared Memory** – Zwei Prozesse nutzen einen gemeinsamen Teil des Hauptspeichers, daher ist keine Kommunikation zwischen verschiedenen Rechnern möglich.
- **Named Pipes** – Basiert auf benannten Datenströmen nach dem FIFO-Prinzip (First In/First Out) und kann bidirektional auch zwischen verschiedenen Rechnern genutzt werden.

- **TCP/IP (Transmission Control Protocol/Internet Protocol)** – Betriebssystemunabhängige Familie von Netzwerkprotokollen, auf der auch das Internet basiert, Identifizierung von Rechnern erfolgt durch IP-Adressen.
- **VIA (Virtual Interface Architecture)** – Modernes, von Microsoft, Intel und Compaq entwickeltes Netzwerkprotokoll, das auf geringe Prozessorlast und optimale Netzwerkauslastung ausgelegt ist.

Über einen rechten Mausklick auf den Namen eines der Protokolle können Sie den Status von *Aktiviert* auf *Deaktiviert* (oder zurück) ändern. Nutzen Sie den Server nur lokal, reicht es völlig aus, wenn nur Shared Memory aktiv ist, damit andere Anwendungen mit dem SQL Server-Dienst kommunizieren können. Wollen Sie von anderen Rechnern im Netzwerk darauf zugreifen, sollten Sie zumindest noch Named Pipes und/oder TCP/IP aktivieren. VIA dagegen verspricht zwar die beste Performance, ist aber von den genannten Netzwerkprotokollen das am wenigsten verbreitete.

SQL Native Client 10.0-Konfiguration

In der Native Client-Konfiguration finden sich unter dem Knoten *Clientprotokolle* fast dieselben Einstellungen wie gerade beschrieben, allerdings beziehen sich diese darauf, mit welchen Protokollen Sie über den ebenso installierten Native Client auf einen entfernten SQL Server zugreifen können. Die ebenfalls angegebene Reihenfolge definiert, in welcher Reihenfolge die jeweiligen Protokolle genutzt werden, um einen Verbindungsaufbau zu versuchen.

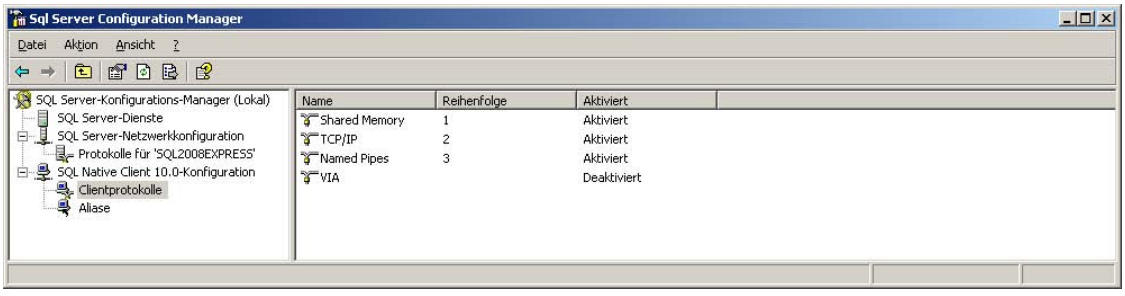


Abbildung 3.11: Übersicht der Clientprotokolle im Konfigurations-Manager

Über den Knoten *Aliase* können Sie einen neuen Alias einrichten, mit dem versucht wird, über einen bestimmten Port und ein bestimmtes Protokoll auf einen verbundenen SQL Server zuzugreifen.

Da wir den SQL Server aber erst einmal nur lokal nutzen möchten, brauchen Sie sich um die Native Client-Konfiguration vorerst nicht zu kümmern.

Hinweis: Konfigurations-Manager für Reporting Services

Für die SQL Server Reporting Services steht ein eigener Konfigurations-Manager zur Verfügung, auf den im Kapitel zu den Reporting Services näher eingegangen wird.

SQL Server Management Studio

Das SQL Server Management Studio ist die integrierte Benutzeroberfläche, mit der Sie die meisten Aktionen für den SQL Server durchführen können. Er beinhaltet sowohl eine grafische Oberfläche, um einzelne Datenbankobjekte auszuwählen und zu bearbeiten, als auch ein Abfragefenster, in dem Sie SQL-Kommandos eingeben und direkt ausführen können. (Wenn Sie schon mit SQL Server 2000 gearbeitet haben, werden Sie feststellen, dass das neue Management Studio die Funktionalität des alten Enterprise Manager und des Query Analyzer in einem Tool vereint.) Damit wird dies sicherlich auch das SQL Server-Tool sein, was Sie in Zukunft am häufigsten verwenden werden.

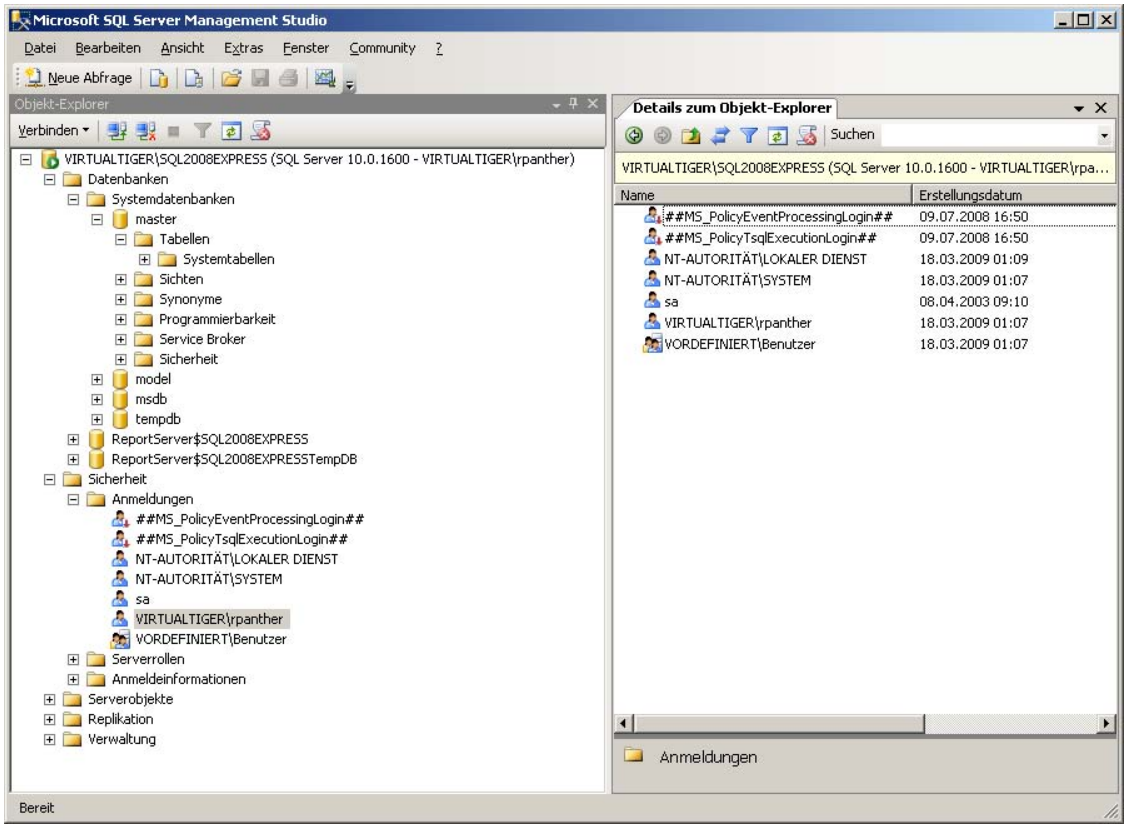


Abbildung 3.12: Das SQL Server Management Studio

Über die Verwendung von SQL Server Management Studio werden Sie in den nächsten Kapiteln viele Details erfahren, daher nun erst mal weiter mit der Übersicht.

SQL Server Business Intelligence Development Studio

Das Business Intelligence Development Studio nutzt die von Visual Studio bekannte Benutzeroberfläche um mit Business-Intelligence-Projekten (SQL Server Analysis Services, SQL Server Integration Services und SQL Server Reporting Services) zu arbeiten. Da von SQL Server Express hiervon nur die Reporting

Services unterstützt werden (und auch diese nur von der Advanced Edition), wird darauf im entsprechenden Abschnitt noch detailliert eingegangen.

Sie sollten sich allerdings nicht davon verwirren lassen, dass bei der Installation von SQL Server auch ein Unterpunkt Visual Studio 2008 in Ihrem Startmenü auftaucht. Aufgrund der gemeinsam genutzten Benutzeroberfläche beinhaltet dieser im Prinzip ebenfalls das Business Intelligence Development Studio. Also unabhängig davon, ob Sie im Menü *Alle Programme/Microsoft SQL Server 2008* das *SQL Server Business Intelligence Development Studio* aufrufen oder im Menü *Alle Programme/Visual Studio 2008* die Anwendung *Visual Studio 2008*, wird die Entwicklungsumgebung Visual Studio 2008 gestartet, die – sofern Sie neben dem SQL Server 2008 Express nicht zusätzlich noch eine Programmiersprache (wie beispielsweise Visual C# 2010 Express) installiert haben – ausschließlich die Business-Intelligence-Projekttypen zur Auswahl anbietet.

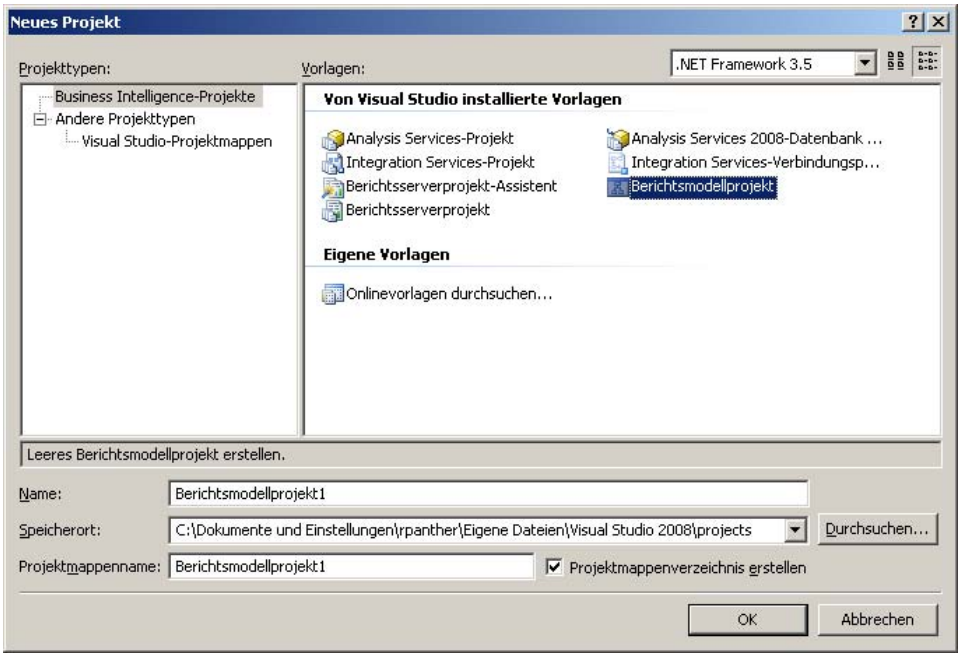


Abbildung 3.13: Die im Business Intelligence Studio verfügbaren Projekttypen

Von den angebotenen Projekttypen sind allerdings für die Express Edition von SQL Server nur die Berichtsvorlagen relevant, die Vorlagen für Analysis Services- und Integration Services-Projekte bleiben den größeren SQL Server-Editionen vorbehalten.

SQL Server-Import/Export-Assistent

Auch wenn die SQL Server Integration Services, die zum Erstellen von komplexen Datenübertragungen prädestiniert sind, nicht Bestandteil der Express Edition sind, so gibt es mit dem Import/Export-Assistenten zumindest eine einfache Möglichkeit, verschiedene Datenquellen anzubinden.

Das Tool taucht unter der Bezeichnung *Daten importieren und exportieren (32-Bit)* im Windows-Startmenü auf und bietet – wie der Name schon vermuten lässt – sowohl die Möglichkeit des Imports von Daten aus einer beliebigen Datenquelle in den SQL Server als auch die des Exports von SQL Server-Daten in ein beliebiges Datenziel.

Als mögliche Datenquellen stehen dabei diverse OLE DB und .NET Framework Data Provider (z.B. ODBC, Oracle, SQL Server) sowie Excel-, Access- und Flatfile-Quellen zur Verfügung. Natürlich darf auch der SQL Server Native Client nicht fehlen, der den direktesten (und damit performantesten) Zugriff auf den SQL Server bietet. Wird eine Datenbankquelle angegeben, lassen sich die zu kopierenden Daten entweder aufgrund von Tabellen, Sichten oder SQL-Abfragen auswählen.

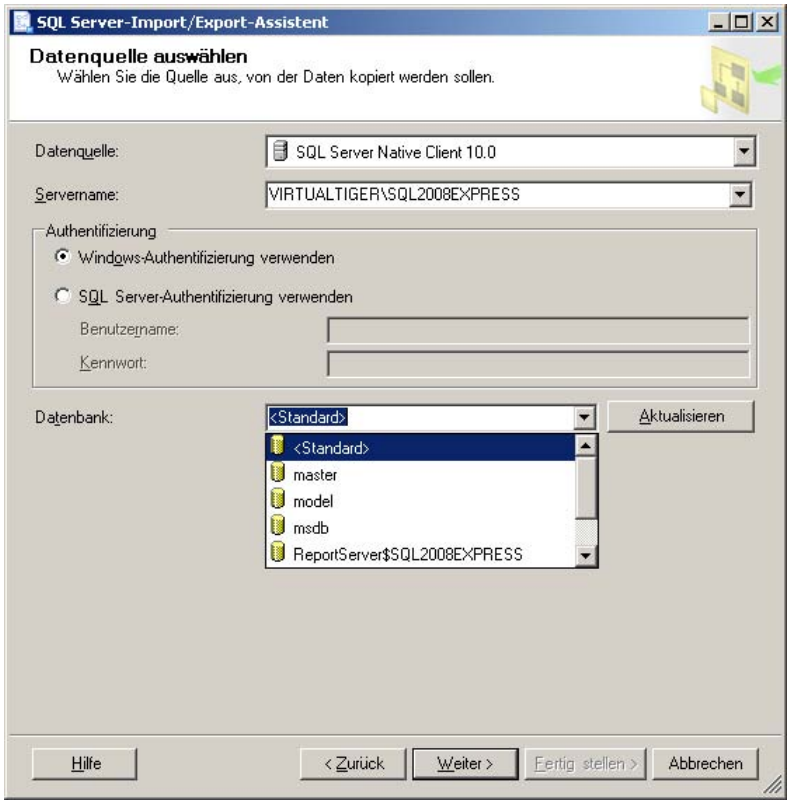


Abbildung 3.14: Der SQL Server-Import/Export-Assistent

Hinweis: Alternative zum Verschieben bzw. Kopieren von ganzen Datenbanken

Wenn ganze Datenbanken von einem auf einen anderen SQL Server verschoben oder kopiert werden sollen, so wird statt der Nutzung des Import-/Export-Assistenten die Verwendung des Assistenten zum Kopieren von Datenbanken innerhalb des SQL Server Management Studio empfohlen.

SQLCMD

Das letzte Tool, das für diese Übersicht erwähnenswert ist, hat keinen Eintrag im Windows-Startmenü, da es sich hierbei um ein reines Kommandozeilentool handelt. Mit der ausführbaren Programmdatei *SQLCMD.EXE*, die im Verzeichnis *C:\Programme\Microsoft SQL Server\100\Tools\Binn* zu finden ist, können Sie SQL-Abfragen oder ganze SQL-Skripts ausführen lassen. Dazu übergeben Sie dem Programm neben dem Namen der SQL-Skriptdatei die Verbindungsdaten für den SQL-Server (also den Namen des SQL Server sowie Benutzer und Passwort) und das SQL-Skript wird auf dem entsprechenden Server ausgeführt, sofern die Berechtigungen des Benutzers dies zulassen.

Das ginge natürlich mit dem SQL Server Management Studio deutlich komfortabler, jedoch ist SQLCMD eher dafür gedacht, Abläufe zu automatisieren, indem die Ausführung des SQL-Skripts in Batchdateien eingebunden werden kann.

Hinweis: Für SQL Server 2008 Express nicht nutzbare Tools

Im Startmenü sind im Untermenü für den SQL Server 2008 noch zwei weitere Programme zu finden, die mit dem SQL Server 2008 Express aber leider nicht nutzbar sind, da sie eigentlich für die größeren Editionen vorgesehen sind. Dies sind der Data Profile Viewer und das Execute Package Utility im Startmenü-Ordner *Alle Programme/Microsoft SQL Server 2008/Integration Services*. Warum diese Tools trotzdem mit installiert werden und im Startmenü auftauchen, ist leider nicht nachvollziehbar.

3.4 Übungen zu diesem Kapitel

In diesem Abschnitt finden Sie einige Übungen zu diesem Kapitel. Die richtigen Antworten, Lösungen und den Programmcode finden Sie auf der Website *www.vsxpress.de.*

Übung 3.1

Womit können Sie Ihre Systemhardware am einfachsten aufrüsten, damit der SQL Server möglichst schnell arbeiten kann?

Übung 3.2

Wie können Sie verschiedene Versionen oder Editionen von SQL Server auf einem Rechner installieren?

Übung 3.3

Welcher Dienst muss für die Verwendung des SQL Server auf jeden Fall gestartet sein?

Übung 3.4

Welche zusätzlichen Möglichkeiten bietet der gemischte Authentifizierungsmodus im Gegensatz zum Windows-Authentifizierungsmodus?

Übung 3.5

Welche Varianten gibt es, um das Konto, unter dem ein SQL Server-Dienst ausgeführt wird, nachträglich zu ändern?

3.5 Zusammenfassung

In diesem Kapitel haben Sie gelernt, welche Systemvoraussetzungen nötig sind, um den SQL Server 2008 in der Express Edition zu installieren. Dazu haben Sie erfahren, welche Einstellungen bei der Installation vorzunehmen sind:

- **Featureauswahl** – Die Datenbankmoduldienste und Verwaltungstools sollten auf jeden Fall ausgewählt werden, der Rest ist wünschenswert (sofern genug Platz vorhanden), aber nicht zwingend notwendig.
- **Instanzkonfiguration** – Im Zweifelsfall immer eine benannte Instanz verwenden, aus deren Instanzname sowohl die Version als auch die Edition des SQL Server hervorgeht (z.B. SQL2008EXPRESS).
- **Dienstkonten** – Auswahl von Starttyp und verwendetem Konto für die verschiedenen SQL Server-Dienste. Die Database Engine sollte unter dem lokalen Systemkonto laufen und automatisch gestartet werden, die anderen Dienste laufen unter dem lokalen Dienstkonto und können manuell gestartet werden.
- **Authentifizierungsmodus** – Im Zweifelsfall immer den gemischten Modus auswählen, um sich alle Möglichkeiten offenzuhalten und den eigenen Netzwerkuser als SQL Server-Administrator definieren.
- **Reporting Services-Konfiguration** – Standardkonfiguration des systemeigenen Modus.

Am Ende des Kapitels haben Sie einen Überblick über die wichtigsten SQL Server-Tools erhalten:

- **SQL Server-Installationscenter** – Oberfläche zum Installieren von SQL Server, aber auch zur Vorbereitung, Reparatur oder Anpassung einer bestehenden Installation
- **SQL Server Konfigurations-Manager** – Tool zum Konfigurieren der verschiedenen Dienste und Kommunikationsprotokolle, die vom SQL Server verwendet werden
- **SQL Server Management Studio** – Komplexes Verwaltungstool, mit dem nahezu alle Datenbankoperationen komfortabel mit der Maus gesteuert werden können, aber auch die Eingabe und Ausführung von SQL-Kommandos möglich ist
- **SQL Server Business Intelligence Development Studio** – Entwicklungsoberfläche, die für die Entwicklung von Business-Intelligence-Projekten (basierend auf SQL Server Analysis Services, Integration Services und Reporting Services) verwendet werden kann. Für die Express Edition von SQL Server sind davon allerdings nur die Reporting Services verfügbar.

- **SQL Server-Import/Export-Assistent** – Tool zum Import von Daten aus einer beliebigen Datenquelle in den SQL Server bzw. Export von SQL Server-Daten in ein beliebiges Datenziel
- **SQLCMD** – Kommandozeilentool zum Ausführen von einzelnen Anweisungen oder ganzen SQL-Skriptdateien

Nachdem SQL Server nun betriebsbereit auf Ihrem Rechner vorliegt, benötigen Sie ein paar theoretische Grundlagen, um sinnvoll mit ihm arbeiten zu können. Diese Grundlagen sind Thema der nächsten Kapitel.

Kapitel 4

Allgemeine Datenbankgrundlagen

In diesem Kapitel lernen Sie

- die wichtigsten Grundlagen zum Arbeiten mit relationalen Datenbanken
- das Erstellen und Ändern von Datenbanken und Tabellen
- wie man mit dem SQL Server Management Studio Daten anzeigen und ändern kann
- was es mit Primärschlüsseln und Indizes auf sich hat

4.1 Erstellen von Datenbanken und Tabellen

Auch wenn der Microsoft SQL Server umgangssprachlich gerne als Datenbank bezeichnet wird, handelt es sich dabei streng genommen um ein Datenbank-Management-System (also ein System zur Verwaltung von Datenbanken)[1]. Eine Datenbank dagegen ist eine Sammlung unterschiedlicher Daten, die thematisch in irgendeiner Form zusammengehören. Diese wiederum setzt sich vor allem aus Tabellen zusammen, die jeweils Daten in einer einheitlichen Struktur enthalten.

In unsere Beispielanwendung übersetzt heißt dies, dass es eine Datenbank (nennen wir sie einfach »MediaBase«) gibt, die verschiedene Tabellen für Bücher, CDs und DVDs enthält.

Jede dieser Tabellen setzt sich aus Zeilen und Spalten zusammen, wobei die Zeilen alle dieselbe Struktur haben, während die Spalten jeweils eine Eigenschaft eines Objekts (z.B. den Titel eines Buches) beschreiben.

Allerdings gibt es auch zwischen Datenbanken und Tabellen noch eine Schicht, die vielen nicht so geläufig ist: die Datenbankschemata. Bei einem Datenbankschema handelt es sich um eine Gruppe von Datenbankobjekten wie beispielsweise Tabellen und Sichten, die eine Untergruppe der Datenbank bilden und daher auch ein gemeinsames Präfix bekommen. Wenn nicht explizit ein Schema angegeben wird, so wird standardmäßig das dbo-Schema verwendet. Der Sinn eines Datenbankschemas ist zweierlei: Erstens wird so die Übersicht gesteigert, da alle Tabellen eines Schemas im Management Studio untereinander stehen, und zweitens lassen sich so leichter Rechte auf alle Tabellen eines Schemas erteilen, ohne gleich die ganze Datenbank freigeben zu müssen.

---

[1] Da der Begriff Datenbank-Management-System ein wenig unhandlich ist, kann man als Kurzform auch Datenbanksystem verwenden.

Anlegen einer Datenbank

Aber beginnen wir erst einmal mit dem Erstellen der Datenbank:

1. Starten Sie SQL Server 2008 Management Studio und verbinden Sie sich mit der lokalen Serverinstanz *SQL2008Express*.

Abbildung 4.1: Verbindung mit SQL Server herstellen

2. Sofern nicht bereits sichtbar, blenden Sie über den Menüpunkt *Ansicht/Objekt Explorer* den Objekt-Explorer ein.
3. Klicken Sie im Objekt-Explorer mit der rechten Maustaste auf *Datenbanken* und wählen Sie den Befehl *Neue Datenbank* aus.
4. Es erscheint das Dialogfeld zum Erstellen einer neuen Datenbank. Im linken Bereich können Sie eine von drei möglichen Seiten mit Einstellungen auswählen: *Allgemein*, *Optionen* und *Dateigruppen*.
5. Nehmen Sie auf der Seite *Allgemein* folgende Einstellungen vor:
   - Datenbankname: MediaBase
   - Besitzer: <Standard>
   - Zeilendaten (erste Zeile in der Liste mit Datenbankdateien)

     Logischer Name: MediaBase

     Anfangsgröße (MB): 20

     Automatische Vergrößerung: 10%, unbeschränkte Vergrößerung (diese Einstellung können Sie vornehmen, wenn Sie auf die Schaltfläche mit den drei Punkten hinter dem entsprechenden Feld klicken)
   - Protokoll (zweite Zeile in der Liste mit Datenbankdateien)

     Logischer Name: MediaBase_log

     Anfangsgröße (MB): 10

     Automatische Vergrößerung: 10%, unbeschränkte Vergrößerung

Wenn Sie möchten, können Sie hier auch noch die Pfade, in denen die Datenbankdateien abgelegt werden, prüfen und bei Bedarf anpassen.

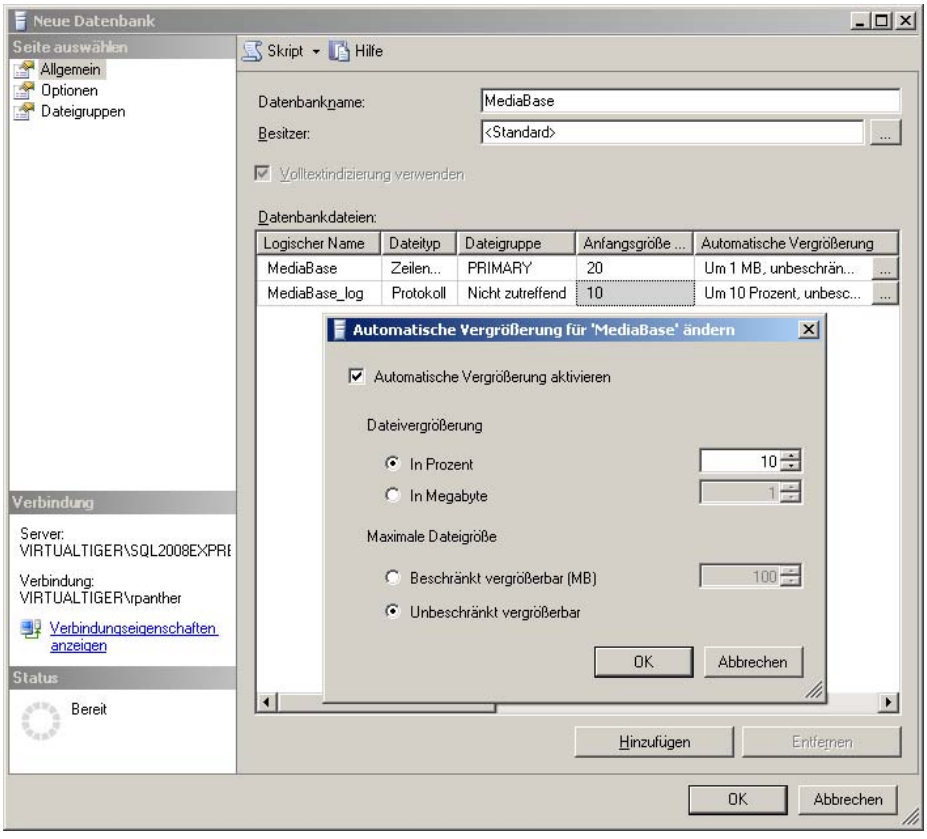


Abbildung 4.2: Das Dialogfeld zum Erstellen einer neuen Datenbank

6. Überprüfen (und gegebenenfalls korrigieren) Sie auf der Seite *Optionen* die folgenden Einstellungen:
   - Sortierung: <Serverstandard>
   - Wiederherstellungsmodell: Vollständig
   - Kompatibilitätsgrad: SQL Server 2008 (100)
7. Überprüfen (und gegebenenfalls korrigieren) Sie auf der Seite *Dateigruppen* die folgenden Einstellungen:
   - Name: PRIMARY
   - Dateien: 1
   - Standard: (angekreuzt)
8. Klicken Sie auf *OK* und die neue Datenbank wird angelegt.

Sie haben nun eine neue Datenbank mit Namen *MediaBase* angelegt. Bevor wir beginnen, mit dieser Datenbank zu arbeiten, möchte ich Ihnen kurz die gerade vorgenommenen Einstellungen erläutern:

- **Datenbankname** ist die Bezeichnung, über die Sie später auf die Datenbank zugreifen. Der Einfachheit halber empfiehlt es sich, hier eine Bezeichnung ohne Leerzeichen und Sonderzeichen (wie beispielsweise Umlaute) zu verwenden, da dies sonst an anderer Stelle zu Problemen führen kann.
- **Besitzer** ist der Datenbankbenutzer, der automatisch volle Berechtigungen auf die gesamte Datenbank erhält. Mit der Einstellung <*Standard*> ist das automatisch der Benutzer, mit dem Sie während der Erstellung mit dem Datenbankserver verbunden sind (also Sie selbst).
- **Datenbankdateien** Datenbanken werden in mehreren Dateien auf der Festplatte abgelegt. Im Normalfall reichen zwei Dateien aus – eine für die eigentlichen Daten (hier Zeilendaten genannt) und eine für das Protokoll. An dieser Stelle können Sie den Namen und die Anfangsgröße für diese Dateien definieren und auch festlegen, in welchen Schritten und bis zu welcher Maximalgröße die Dateien bei Bedarf automatisch wachsen dürfen. Ein Anlegen von weiteren Dateien macht vor allem aus zwei Gründen Sinn:
  - **Platz** Wenn auf der Festplatte, auf der die Dateien bisher liegen, kaum noch Platz frei ist, stellt man diese auf nicht mehr wachsend und fügt eine weitere Datei auf einer zweiten Festplatte hinzu.
  - **Performance** Durch das Anlegen von mehreren Dateien auf mehreren Platten kann man unter Umständen Performance gewinnen, sofern auf diese Dateien parallel zugegriffen werden kann (mehrere Partitionen auf derselben Platte machen hier also keinen Sinn).

Hintergrundinfo: Zeilendaten und Protokoll

Auch wenn dieses Thema an anderer Stelle noch einmal ausführlicher behandelt werden wird, möchte ich hier kurz erläutern, was es mit den Zeilendaten- und Protokolldateien auf sich hat.

- **Zeilendaten** sind die eigentlichen Daten, die in der Datenbank (oder in den Zeilen der Tabellen einer Datenbank) enthalten sind. Daher sollte diese Datei so groß gewählt werden, dass sie die zu erwartende Menge der Daten aufnehmen kann, da die automatische Vergrößerung zusätzlich Zeit kostet und daher nicht unnötig verwendet werden sollte.
- **Protokoll** Im Protokoll (oder auch Transaktionslog) werden Änderungen an den Zeilendaten zwischengespeichert, bevor diese endgültig in den Zeilendaten abgelegt werden. Dieser Mechanismus ist aus verschiedenen Gründen sinnvoll: So sind dadurch einerseits Transaktionen möglich, und andererseits kann mithilfe des Protokolls (bzw. einer Sicherung desselben) nach einem Ausfall wieder ein Datenbankstand zu einem nahezu frei wählbaren Zeitpunkt hergestellt werden (sofern gewisse Rahmenbedingungen erfüllt sind). Beide Themen werden aber in späteren Kapiteln noch ausführlicher angesprochen. Im Moment reicht es völlig aus, zu wissen, dass die Größe des Transaktionslogs von Häufigkeit und Umfang der Datenänderungen abhängt. Je mehr und umfangreichere Änderungen anfallen, desto größer wird das Protokoll.

- **Sortierung** legt den Modus der Sortierung (case sensitive, accent sensitive etc.) für die gesamte Datenbank fest. Sofern Sie hier beim Installieren des SQL Servers die richtige Auswahl getroffen haben (empfehlenswert ist beispielsweise *Latin1_General_CI_AS*), können Sie die Voreinstellung <*Serverstandard*> übernehmen. Wesentlich ist, dass möglichst alle Ihre Datenbanken dieselbe Sortierung haben sollten, da es zu Problemen kommt, wenn Sie in einer Abfrage Tabellen mit unterschiedlichen Sortierungen verwenden wollen.

- **Wiederherstellungsmodell** steuert den Grad, in dem Änderungen an Daten mitprotokolliert werden, um bei Bedarf die Datenbank zu einem bestimmten Stand wiederherstellen zu können. *Vollständig* gibt Ihnen hier alle Freiheiten; die Alternativen *Einfach* und *Massenprotokolliert* schreiben weniger Daten, bieten dafür aber auch weniger Möglichkeiten bei Backup & Restore. Dieses Thema wird detaillierter in einem späteren Kapitel des Buches behandelt.
- **Kompatibilitätsgrad** Sollten Sie Datenbanken und/oder Skripts von einer älteren Version des SQL Servers übernehmen, können Sie an dieser Stelle die Kompatibilität zu *SQL Server 2000 (80)* oder *SQL Server 2005 (90)* erzwingen, allerdings unter Verzicht auf die neuen Features und Datentypen. Daher sollte man möglichst die Voreinstellung *SQL Server 2008 (100)* beibehalten.
- **Dateigruppen** Bei den Dateigruppen handelt es sich um eine logische Zwischenschicht, die zwischen Datenbankobjekten (wie beispielsweise Tabellen) und Datenbankdateien liegt. Beim Erstellen von Datenbankobjekten können Sie nur angeben, in welche Dateigruppe das Objekt angelegt wird. Die Datenbankdateien selbst werden bei ihrer Erstellung ebenfalls Dateigruppen zugeordnet. Sofern es sich nicht um sehr große Datenbanken handelt (und das wird bei der Express Edition normalerweise nicht der Fall sein), werden Sie voraussichtlich nur mit einer Dateigruppe arbeiten, die standardmäßig *PRIMARY* heißt.

Klicken Sie nun im Management Studio den Knoten *Datenbanken* an, um die neu erstellte Datenbank zu sehen. Eventuell müssen Sie mit einem Rechtsklick auf den Punkt *Datenbanken* erst die Option *Aktualisieren* auswählen, damit die Darstellung auf den aktuellen Stand gebracht wird. Dieser Schritt funktioniert auf jeder Ebene des Objekt-Explorers und ist insbesondere dann notwendig, wenn Datenbankobjekte von anderen Benutzern oder per SQL-Skript erstellt wurden.

Mit einem Rechtsklick auf die neu erstellte Datenbank können Sie über die Option *Eigenschaften* die gerade vorgenommenen Einstellungen überprüfen. Teilweise lassen sich diese hier auch ändern.

Anlegen von Tabellen

Nachdem nun eine Datenbank vorliegt, ist es an der Zeit, darin auch ein paar Tabellen anzulegen.

1. Klicken Sie im Objekt-Explorer mit der rechten Maustaste auf den Knoten *Datenbanken/MediaBase/Tabellen* und wählen Sie die Option *Neue Tabelle* aus.
2. Im Detailbereich erscheint nun die Entwurfsansicht für Tabellen, bei der Sie die benötigten Spalten direkt angeben können.
3. Tragen Sie in der ersten Zeile als Spaltenname *ID* ein und wählen Sie als Datentyp *int* aus. Das Feld *NULL-Werte zulassen* sollte nicht angekreuzt werden.
4. Tragen Sie in derselben Weise die weiteren Felder in die nächsten Zeilen ein (die Bedeutung der verschiedenen Datentypen wird weiter unten erklärt):

Tabelle 4.1: Die Felder der Tabelle *Buch*

Spaltenname	Datentyp	NULL-Werte zulassen	Sonstiges
ID	int	nein	Primärschlüssel
ISBN10	char(10)	ja	
ISBN13	char(14)	ja	
Autor	varchar(80)	ja	
Titel	varchar(80)	ja	
Verlag	varchar(80)	ja	
Seiten	int	ja	
Auflage	tinyint	ja	
Sprache	varchar(20)	ja	
Hardcover	bit	ja	
Erscheinungsjahr	int	ja	
Bewertung	tinyint	ja	
Beschreibung	varchar(MAX)	ja	

5. Wenn Sie alle Felder erfasst haben, klicken Sie noch einmal auf die erste Zeile (in der die Spalte mit dem Namen *ID* definiert ist) und klicken anschließend das kleine gelbe Schlüsselsymbol in der Symbolleiste an. Damit wird das Feld *ID* als Primärschlüssel definiert. (Was es damit genau auf sich hat, wird weiter hinten in diesem Kapitel erläutert.)

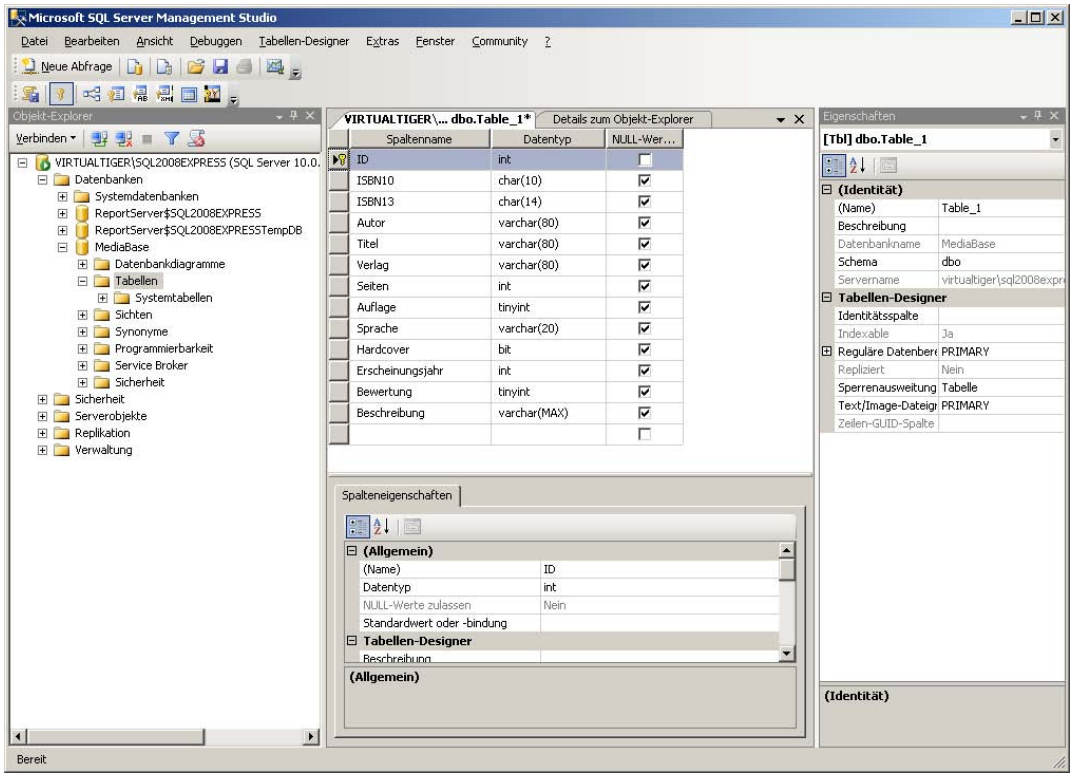


Abbildung 4.3: Die Tabelle im Entwurfsmodus

6. Um die Tabellendefinition zu speichern, klicken Sie in der Symbolleiste auf das Diskettensymbol oder wählen Sie im *Datei*-Menü den Befehl *Table_1 speichern* aus. Der Name *Table_1* ist dabei nur ein automatisch generierter Vorschlag, den Sie beim Speichern überschreiben können (und sollten).
7. Geben Sie als Tabellennamen *Buch* an und schließen Sie anschließend die Tabellenentwurfsansicht durch einen Klick auf das kleine Kreuz in der rechten oberen Ecke. (Achtung! Hier ist natürlich die rechte obere Ecke des Detailbereichs gemeint, mit dem anderen Kreuz ganz rechts oben schließen Sie das gesamte SQL Server Management Studio.)
8. Wenn Sie nun im Objekt-Explorer mit dem +-Zeichen vor den Tabellen die nächste Ebene einblenden, sollte dort auch die neue Tabelle *Buch* stehen. Da Sie nicht explizit ein Schema angegeben haben, wird automatisch das Standardschema *dbo* verwendet, sodass die Tabelle als *dbo.Buch* angezeigt wird.
9. Nach demselben Vorgehen, mit dem Sie die Tabelle *Buch* erstellt haben, können Sie nun zwei weitere Tabellen erstellen:

Tabelle 4.2: Die Felder der Tabelle *CD*

Spaltenname	Datentyp	NULL-Werte zulassen	Sonstiges
ID	int	nein	Primärschlüssel
Titel	varchar(80)	ja	
Interpret	varchar(80)	ja	
Songs	varchar(MAX)	ja	
AnzahlCDs	tinyint	ja	
Erscheinungsjahr	int	ja	
Bewertung	tinyint	ja	

Tabelle 4.3: Die Felder der Tabelle *DVD*

Spaltenname	Datentyp	NULL-Werte zulassen	Sonstiges
ID	int	nein	Primärschlüssel
Titel	varchar(80)	ja	
Originaltitel	varchar(80)	ja	
Schauspieler	varchar(MAX)	ja	
Produzent	varchar(80)	ja	
Regie	varchar(80)	ja	
Land	varchar(20)	ja	
Dauer	tinyint	ja	
Laendercode	tinyint	ja	
Ton	varchar(20)	ja	
Untertitel	varchar(20)	ja	
Extras	varchar(MAX)	ja	
AnzahlDVDs	tinyint	ja	
Erscheinungsjahr	int	ja	
Bewertung	tinyint	ja	

Hinweis: Schreibweise von Tabellen- und Feldnamen

Die Tabellennamen wurden alle bewusst im Singular gewählt, auch wenn klar ist, dass in der Tabelle *Buch* mehrere Bücher gespeichert werden (genau wie bei den CDs und DVDs auch). Lediglich bei den Feldnamen wurde hier und da bewusst der Plural verwendet, nämlich genau dann, wenn ein Feld auch mehrere der bezeichneten Elemente enthält. (So werden beispielsweise im Feld *Songs* mehrere Songtitel abgespeichert.)

Auch auf Leerzeichen und Umlaute sollte bei Tabellen- und Feldnamen möglichst verzichtet werden. SQL Server kann damit zwar umgehen, jedoch kann dies an anderen Stellen (z.B. bei Programmen, die mit diesen Daten arbeiten) zu Problemen führen. Daher wurde bei der *DVD*-Tabelle das Feld für den Ländercode mit *Laendercode* bezeichnet.

Setzt sich ein Name aus mehreren Wörter n zusammen, so können Sie diese entwede r mit einem Unterst rich (»_«) voneinander trennen oder aber die als Pascal Casing beka nnte Schreibweise v erwenden, die darauf beruht, dass der erste Buchstabe jedes Wortes großgeschrieben wird.[1]

Im Objekt-Explorer sollten nun drei Tabellen *dbo.Buch*, *dbo.CD* und *dbo.DVD* zu sehen sein.

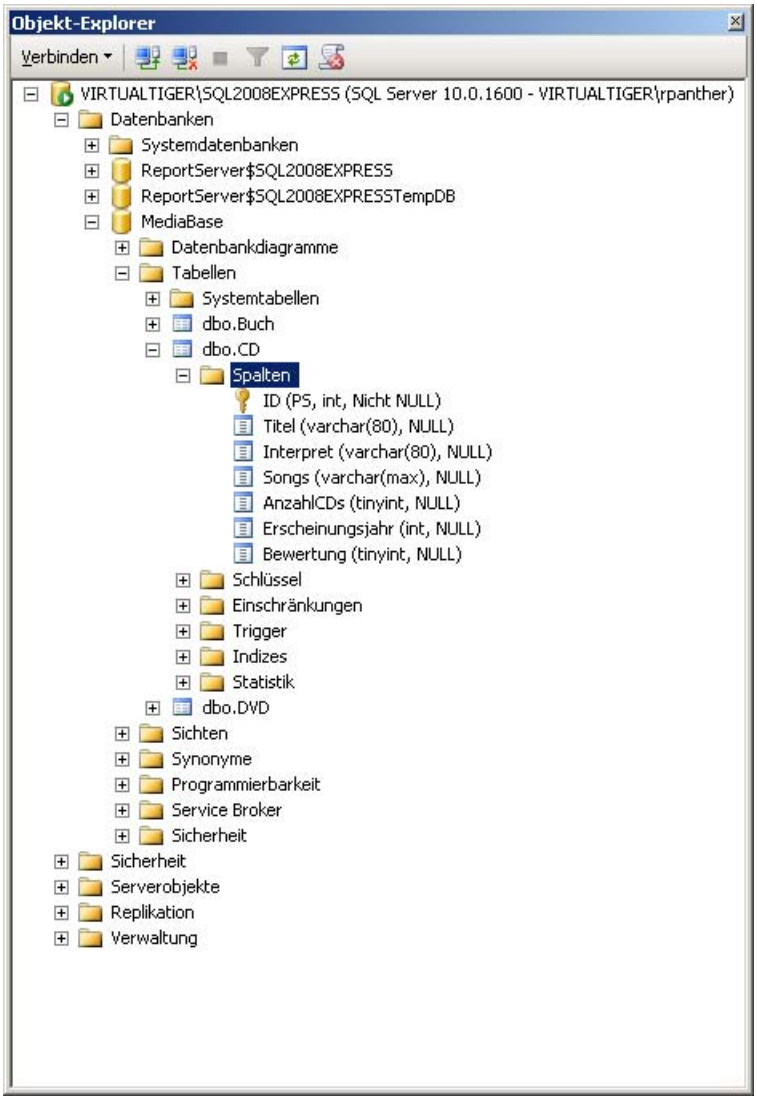


Abbildung 4.4: Die drei neu angelegten Tabellen im Objekt-Explorer

[1] Ich verwende primär Pascal Casing und nutze den Unterstrich nur, um ein Präfix vom eigentlichen Bezeichner zu trennen.

Spalten und Datentypen

Auch wenn die Spalten der Tabellen, die Sie gerade angelegt haben, im Entwurfsmodus als verschiedene Zeilen angezeigt wurden, handelt es sich dabei natürlich doch um Spalten (die später auch als solche sichtbar sind), während die Zeilen einer Tabelle dann Datensätzen (gleicher Struktur) entsprechen.

Ich möchte an dieser Stelle aber noch etwas detaillierter auf die verschiedenen Eigenschaften einer Spalte eingehen. Neben dem Namen, der relativ frei wählbar ist, ist hier vor allem der Datentyp relevant, denn jede Spalte benötigt einen Datentyp, der vorgibt, welche Inhalte für diese Spalte zugelassen sind.

SQL Server 2008 bietet eine ganze Menge an Datentypen an. Diese lassen sich unterteilen in numerische Datentypen, alphanumerische Datentypen, binäre Datentypen, Zeit- und Datumstypen sowie eine ganze Reihe weiterer spezieller Datentypen.

Lassen Sie sich von der Vielzahl der aufgelisteten Datentypen nicht erschrecken. Für die meisten (und insbesondere bei kleineren) Anwendungen reicht eine Handvoll der im Folgenden genannten Typen völlig aus. Daher will ich hier auch nicht allzu detailliert auf alle Datentypen eingehen und mich primär auf die gängigsten konzentrieren. Eine komplette Auflistung aller Datentypen finden Sie im Anhang des Buches.

Tipp: Datentypen im Management Studio anzeigen

Irgendwo in den Tiefen der Onlinehilfe ist natürlich auch eine Übersicht der SQL Server-Datentypen zu finden. Einfacher geht es aber, wenn Sie im Objekt-Explorer unterhalb einer beliebigen Datenbank den Zweig *Programmierbarkeit/Typen/Systemdatentypen* öffnen. Darunter bekommen Sie alle verfügbaren Datentypen (nach Kategorien geordnet) angezeigt.

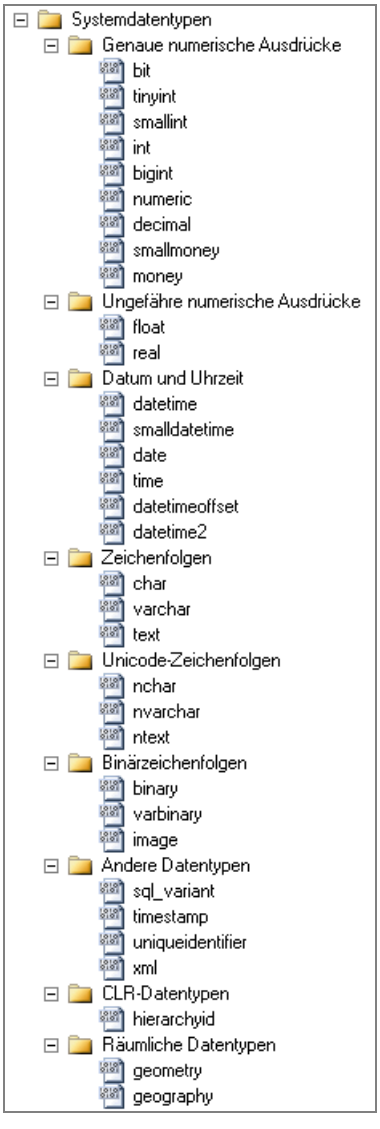


Abbildung 4.5: Anzeige der Datentypen im SQL Server Management Studio

Numerische Datentypen

Die numerischen Datentypen lassen sich in drei Untergruppen unterteilen. Da sind einmal die ganzzahligen Datentypen mit unterschiedlichen Wertebereichen. Angefangen vom Typ *bit*, der nur die Werte 0 und 1 (bzw. *false* und *true*) speichern kann, über *tinyint* (0 bis 255), *smallint* und *int* bis hin zu *bigint*. Mit den Typen *bit*, *tinyint* und *int* kommt man für die meisten Anwendungen allerdings gut aus, wobei *bit* primär für logische Felder verwendet wird, deren Inhalte Ja oder Nein repräsentieren.

Die zweite Gruppe numerischer Datentypen sind die Kommazahlen fester Genauigkeit, zu denen auch die Währungstypen *money* und *smallmoney* gehören (letzterer beschränkt sich allerdings nicht – wie der Name vielleicht irrtümlich vermuten lässt – auf Kleingeld, sondern kann immerhin Werte von –214.748,3648 bis 214.748,3647 aufnehmen, was in der Währung Euro beispielsweise für manch ein Einfamilienhaus ausreichend ist). Beide Währungstypen (*money* und *smallmoney*) speichern ihre Werte mit vier Nachkommastellen, was die Gefahr von Rundungsfehlern im Cent-Bereich minimiert. Für die anderen beiden numerischen Datentypen mit fester Genauigkeit wird die Anzahl der Nachkommastellen explizit angegeben. So entspricht *decimal*(10,2) beispielsweise einer zehnstelligen Zahl, von denen zwei Nachkommastellen sind.

Als dritte Gruppe gibt es dann noch die Gleitkommazahlen mit ungefährer Genauigkeit, mit denen sich dafür extrem hohe Zahlenwerte darstellen lassen. Hier stehen *float*(n) und *real* zur Auswahl wobei *real* mit einem Zahlenbereich von –3,4E+38 bis +3,4E+38 für die meisten Fälle ausreichen dürfte.

Alphanumerische Datentypen

Für die Auswahl des richtigen alphanumerischen Datentyps gilt es, zwei Entscheidungen zu treffen; aus der Kombination der Antworten ergibt sich dann der passende Datentyp. Das eine ist die Länge des Datenfeldes. Diese kann entweder konstant, variabel (mit einer definierten Obergrenze) oder völlig unbestimmt sein. In der Praxis werden Sie Datentypen fester Länge nur relativ selten (und dann ganz gezielt) verwenden, da die wenigsten typischen Datenbankinhalte eine immer gleich bleibende Feldgröße aufweisen und Sie somit unnötig Speicherplatz »verschenken« würden.

Neben der Länge des Datenfeldes steht die Frage, ob ein Zeichensatz mit einem Byte pro Zeichen ausreicht (der regional unterschiedlich ist, um die nationalen Sonderzeichen zu berücksichtigen) oder ob der alle internationalen Sonderzeichen umfassende Zeichensatz (Unicode) benötigt wird, der pro Zeichen allerdings zwei Byte Speicherplatz erfordert.

Tabelle 4.4: Entscheidungstabelle für alphanumerische Datentypen

	feste Länge	variable Länge (mit Obergrenze)	variable Länge (ohne Obergrenze)[1]
kein Unicode	char(n) varchar(n)		varchar(max)
Unicode	nchar(n) nvarchar(n)		nvarchar(max)

Der Datentyp *text* und dessen Unicode-Äquivalent *ntext* werden nur noch aus Kompatibilitätsgründen unterstützt und sollten nicht mehr verwendet werden (dafür gibt es seit SQL Server 2005 varchar(max) und nvarchar(max)).

Tipp: Unicode- und Nicht-Unicode-Datentypen nicht mischen!

Die Entscheidung, ob Sie Unicode-Datentypen benötigen oder nicht, sollten Sie möglichst einmal für die gesamte Datenbank treffen. Eine gemischte Verwendung von Unicode- und Nicht-Unicode-Feldern kann zu Problemen führen, die beispielsweise eine schlechtere Performance zur Folge haben.

[1] Die Angabe »ohne Obergrenze« ist streng genommen nicht ganz korrekt, denn es gibt eine technische Obergrenze, die bei ca. 2 GB (genau 2.147.483.647 Byte) liegt.

Binäre Datentypen

Die binären Datentypen enthalten mit dem Typ *image* auch einen nicht mehr aktuellen Datentyp, der nicht mehr verwendet werden sollte. Stattdessen sind die neueren Typen *binary(n), varbinary(n)* und *varbinary(max)* vorzuziehen. In der Praxis werden binäre Datentypen verwendet, um ganze Dateien – wie beispielsweise Grafiken oder Sounds – direkt in der Datenbank zu speichern. Aus diesem Grund ist *varbinary(max)* sicherlich der meistverwendete dieser Typen, da die Dateien ja auch völlig unterschiedliche Größen haben können. Mit der *Filestream*-Option gibt es für diesen Datentyp noch eine interessante Erweiterung, auf die in einem späteren Kapitel noch genauer eingegangen wird.

Temporale Datentypen

Die Datentypen zur Speicherung von temporalen Informationen beschränkten sich lange Zeit auf den Typ *datetime*, der Datum und Uhrzeit in einem Feld ablegte. Erst mit SQL Server 2008 kamen getrennte Datentypen für Zeit (*time*) und Datum (*date*) hinzu. Dazu gibt es noch ein paar weitere Varianten von *datetime,* die einen anderen Zeitraum umfassen (*smalldatetime* und *datetime2(n))* oder zusätzlich noch Informationen über die entsprechende Zeitzone speichern *(datetimeoffset(n)).*

Sonstige Datentypen

Neben den bisher erwähnten allgemeinen Datentypen gibt es noch eine ganze Menge spezieller Datentypen, mit denen man XML-Dokumente (*xml*), Global Unique Identifiers (*uniqueidentifier*), geografische Informationen und vieles mehr speichern kann. Die interessantesten dieser Datentypen werden an späterer Stelle noch einmal ausführlich behandelt.

NULL-Werte und Defaults

Neben der Auswahl des richtigen Datentyps ist noch zu definieren, ob für das entsprechende Feld NULL-Werte zugelassen werden. Dabei ist nicht die Zahl 0 gemeint, vielmehr handelt es sich bei NULL um einen Wert, der einen nicht definierten Zustand repräsentiert und damit für Felder verwendet werden kann, die noch nicht gefüllt wurden (beispielsweise, weil sie unbekannt sind). In vielen Fällen, macht es jedoch Sinn, eine Spalte zum Pflichtfeld zu erklären, also bereits auf Datenbankebene dafür zu sorgen, dass hier ein Wert angegeben werden muss. Dies erreicht man, indem man das Häkchen bei *NULL-Werte zulassen* für die entsprechende Spalte entfernt. Dadurch wird bei jedem Speichern einer Zeile geprüft, ob alle Pflichtfelder belegt sind, ansonsten wird das Speichern verweigert.

Hinweis: NULL ≠ "" ≠ 0

Beachten Sie auch, dass NULL weder der Zahl 0 noch einer leeren Zeichenkette "" entspricht. Für NULL-Werte ist später im Programmcode oder in SQL-Abfragen immer eine Sonderbehandlung erforderlich. So ist NULL generell weder gleich noch ungleich einem konstanten Wert. (Hier liegt der Vergleich zum Roulette nahe, denn auch dort ist die 0 weder rot noch schwarz, sondern eben nur 0.) Stattdessen muss explizit geprüft werden, ob ein Feld den Wert NULL enthält.

Eine einfache Variante, mit NULL-Werten umzugehen, liegt darin, diese bei der Tabellendefinition gar nicht erst zuzulassen und stattdessen einen Standardwert (Default-Wert) zu definieren, der automatisch

angenommen wird, wenn für die entsprechende Spalte kein Inhalt angegeben wird. Damit ist sichergestellt, dass eine Spalte immer einen gültigen Wert enthält, und man kann auf eine umständliche Sonderbehandlung von NULL-Werten verzichten.

Hinweis: SQL-Funktionen für Default-Werte nutzen

Neben konstanten Werten kann man auch die integrierten SQL Server-Funktionen nutzen, um Felder mit dynamischen Default-Werten zu füllen. So bietet es sich beispielsweise an, für ein Feld, in dem der Zeitpunkt der letzten Änderung gespeichert wird, die Funktion `getdate()` als Standardwert anzugeben, mit der immer der aktuelle Zeitpunkt (Datum und Uhrzeit) gesetzt wird.

4.2 Anzeigen und Ändern von Daten

Nachdem nun die Tabellen erstellt sind, sollen diese natürlich auch mit Daten gefüllt werden.

Dies kann einerseits mit der Datenbanksprache SQL geschehen (darauf wird an anderer Stelle in diesem Buch noch ausführlich eingegangen). Am komfortabelsten geht dies aber direkt mit dem SQL Server Management Studio.

Ändern von Tabelleninhalten

Da die Tabellen bisher ja noch keine Daten beinhalten, macht es Sinn, hier gleich ein paar Zeilen einzugeben.

1. Suchen Sie im Objekt-Explorer den Eintrag für die Tabelle *Buch* und klicken Sie diesen mit der rechten Maustaste an. Wählen Sie im anschließend erscheinenden Kontextmenü die Option *Oberste 200 Zeilen bearbeiten* aus, um die Tabelle im Bearbeitungsmodus zu öffnen.
2. In der folgenden Ansicht sehen Sie die Tabelle *Buch* mit allen Spalten dargestellt (evtl. müssen Sie den Detailbereich nach rechts scrollen, um auch die hinteren Spalten sehen zu können).
3. Geben Sie nun in der ersten Zeile folgende Werte ein:
   - ID: 1
   - ISBN10: 3866452047
   - ISBN13: 978-3866452046
   - Autor: Robert Panther
   - Titel: Datenbanken entwickeln mit SQL Server 2008 Express Edition
   - Verlag: Microsoft Press
   - Auflage: 1
   - Sprache: deutsch
   - Hardcover: False
   - Erscheinungsjahr: 2009

Sobald Sie die ersten Eingaben vorgenommen haben, erscheint in dem kleinen grauen Kästchen links neben der Tabellenzeile ein kleines Stiftsymbol, das anzeigt, dass hier ungespeicherte Änderungen vorliegen. In den einzelnen Feldern der Zeile sehen Sie kleine rote Kreise mit Ausrufezeichen darin, die im Prinzip dasselbe (ungespeicherte Informationen) auf Feldebene anzeigen. In dem Moment, in dem Sie in die nächste Zeile springen oder aber das Stiftsymbol anklicken, wird die Zeile gespeichert und die kleinen Symbole verschwinden wieder.

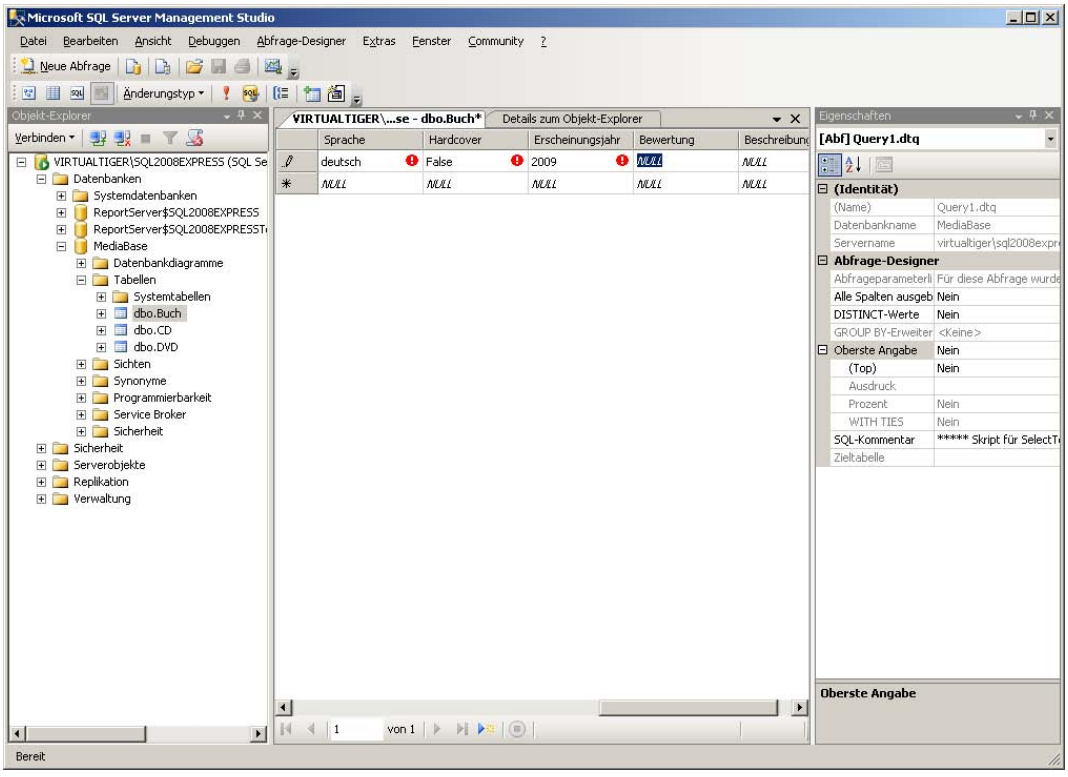


Abbildung 4.6: Die Tabelle *Buch* im Bearbeitungsmodus

Hinweis: Begrenzte Zeilenanzahl im Auswahl- und Bearbeitungsmodus

Sowohl der Auswahl- als auch der Bearbeitungsmodus von Tabellen ist im SQL Server Management Studio standardmäßig auf eine bestimmte Zeilenanzahl begrenzt (1000 Zeilen im Auswahlmodus, 200 Zeilen im Bearbeitungsmodus). Dies wurde so eingerichtet, um in Client-/Server-Betrieb zu vermeiden, dass unnötig viele Zeilen gesperrt werden, wenn jemand einfach nur einen Blick auf die Tabelle werfen will. Diese Standardeinstellung lässt sich bei Bedarf aber recht einfach umgehen.

Im Bearbeitungsmodus sehen Sie auf der rechten Seite bei den Eigenschaften eine Einstellung *Oberste Angabe*. Wenn Sie durch Anklicken des Pluszeichens davor die Unteroptionen aufklappen, können Sie hier die Einstellung *(Top)* auf *Nein* setzen. Wenn Sie anschließend das rote Ausrufezeichen in der Symbolleiste anklicken, wird die Anzeige aktualisiert und es werden alle Zeilen eingeblendet.

4. Beenden Sie den Bearbeitungsmodus der Tabelle, indem Sie auf das kleine Kreuz in der rechten oberen Ecke des Detailbereichs klicken.
5. Fügen Sie auf dieselbe Weise ein paar Zeilen in die Tabellen *CD* und *DVD* ein.

Anzeigen von Daten

Nachdem Sie nun Daten erfasst haben, möchten Sie diese auch wieder anzeigen können. Dies geht natürlich einerseits, indem Sie auch hierfür den Bearbeitungsmodus verwenden. Einfacher geht es aber im Auswahlmodus:

1. Suchen Sie im Objekt-Explorer den Eintrag für die Tabelle *Buch* und klicken Sie diesen mit der rechten Maustaste an. Wählen Sie im anschließend erscheinenden Kontextmenü die Option *Oberste 1000 Zeilen auswählen* aus, um die Tabelle im Auswahlmodus zu öffnen.
2. Der Detailbereich wird nun vertikal geteilt: Im unteren Bereich sehen Sie die ersten 1.000 Zeilen der Tabelle (evtl. müssen Sie auch hier ein wenig nach oben/unten bzw. links/rechts scrollen, um alle Zeilen und Spalten sehen zu können). Im oberen Bereich sehen Sie die SQL-Abfrage, die automatisch ausgeführt wurde, um diese Daten anzuzeigen.

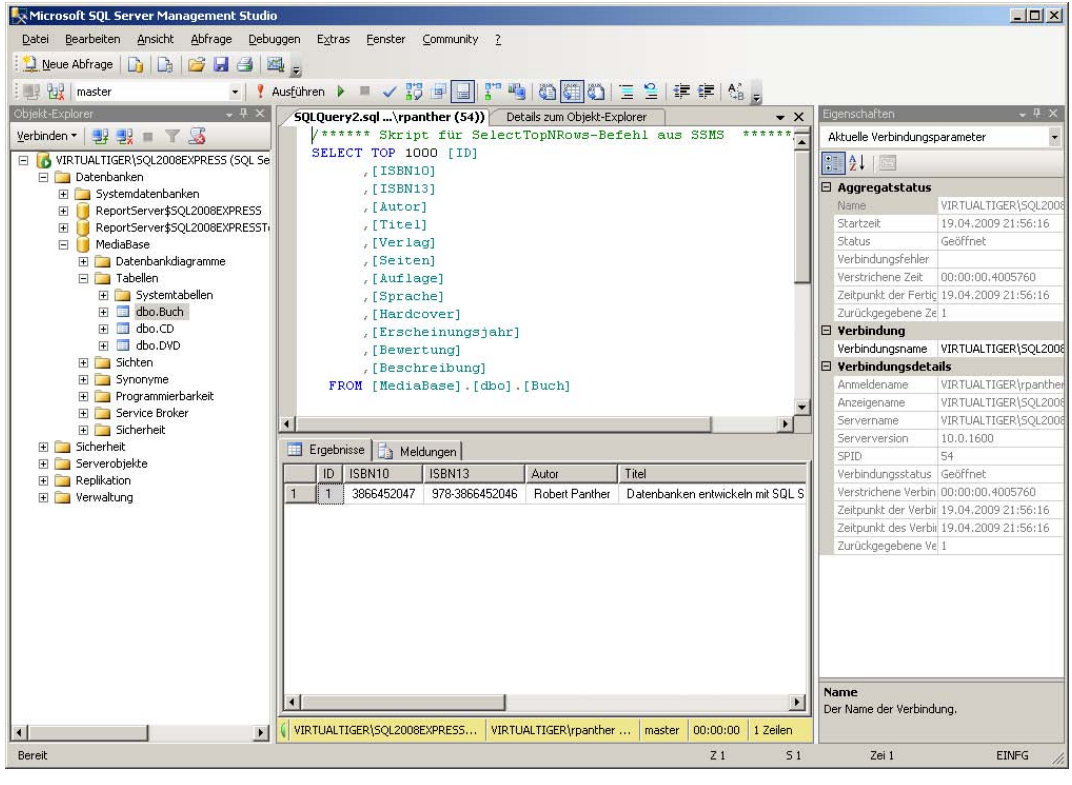


Abbildung 4.7: Die Tabelle *Buch* im Auswahlmodus

3. Löschen Sie den Teil *TOP 1000* aus der Abfrage, sodass in der entsprechenden Zeile nur noch *SELECT [ID]* steht, und führen Sie die Abfrage durch Anklicken des roten Ausrufezeichens in der Symbolleiste erneut aus. Das Ergebnis im unteren Bereich ändert sich scheinbar nicht, allerdings würden nun alle Zeilen angezeigt, auch wenn mehr als 1.000 Zeilen in der Tabelle enthalten wären.
4. Schließen Sie den Auswahlmodus der Tabelle wieder, indem Sie das kleine Kreuz in der rechten oberen Ecke des Detailbereichs anklicken.

Hinweis: Abfrage für Auswahlmodus anpassen

Sie können die Abfrage für den Auswahlmodus auch weiter anpassen, indem Sie beispielsweise Spaltennamen entfernen, deren Inhalte Sie nicht sehen möchten, oder aber eine Bedingung hinzufügen, um nur einen Teil der Zeilen zu sehen. Die genaue Syntax für die SQL SELECT-Anweisung wird in Kapitel 6, *Kleine Einführung in SQL* behandelt.

4.3 Bearbeiten von Datenbanken und Tabellen

Zu Beginn dieses Kapitels haben Sie gelernt, wie Sie Datenbanken und Tabellen anlegen können. Aber was ist, wenn sich die Anforderungen später ändern? SQL Server bietet die Möglichkeit, sowohl Datenbanken als auch Tabellen nachträglich anzupassen.

Ändern von Datenbankeinstellungen

Eine Änderung der Datenbank sollte eher selten notwendig sein – zumal sich viele Eigenschaften im Nachhinein nicht mehr ändern lassen. Wenn aber beispielsweise der Fall eintritt, dass die Datenbankdatei so sehr angewachsen ist, dass kein Platz mehr auf der Festplatte vorhanden ist, sollten Sie eine zweite Datenbankdatei auf einer weiteren Platte (die hoffentlich verfügbar ist) anlegen und diese mit der Datenbank verbinden.

1. Klicken Sie mit der rechten Maustaste im Objekt-Explorer auf die Datenbank und wählen Sie die Option *Eigenschaften* aus. Daraufhin erscheint ein Dialogfeld ähnlich dem, mit dem Sie die Datenbank erstellt haben.
2. Auf der Seite *Allgemein* sehen Sie die aktuelle Datenbankgröße sowie den noch verfügbaren Speicherplatz. Mit dem letztgenannten Wert ist nicht der freie Platz auf dem Datenträger, sondern der unbelegte Platz in den bereits vorhandenen Datenbankdateien gemeint.

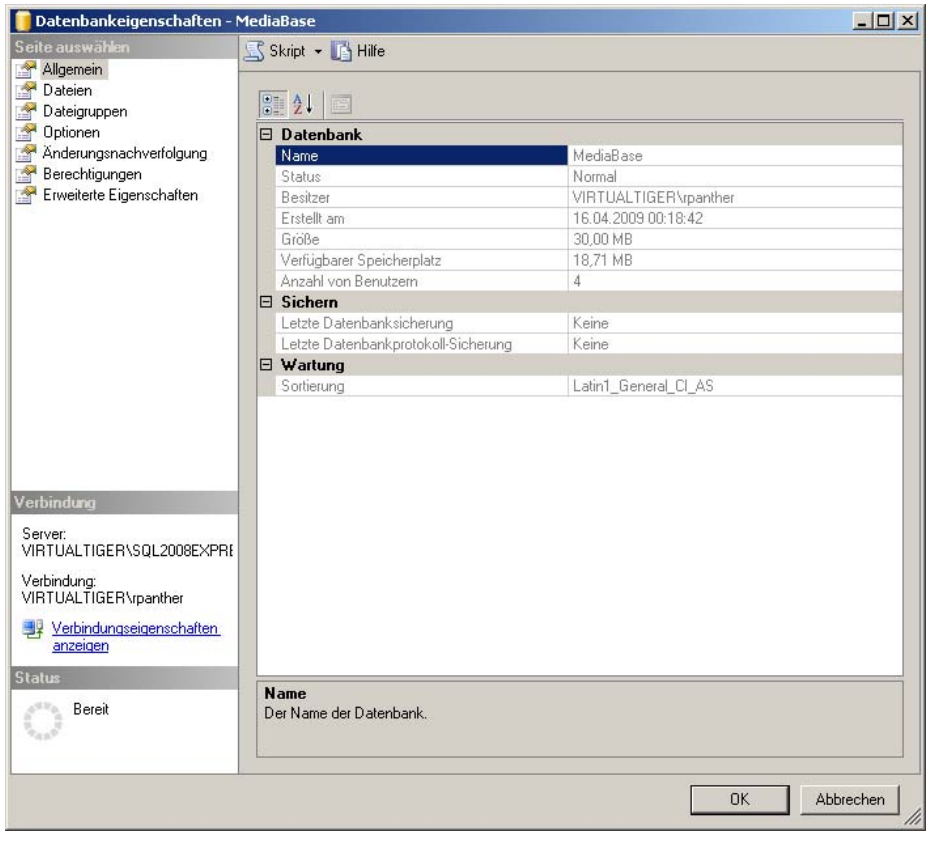


Abbildung 4.8: Datenbankeigenschaften

3. Wechseln Sie zur Seite *Dateien*. Hier sehen Sie die bereits genutzte Zeilendaten- und Protokolldatei. Wenn Sie auf die drei Punkte in der Spalte *Automatische Vergrößerung* klicken, können Sie dort das Häkchen bei *Automatische Vergrößerung aktivieren* entfernen, da ja ohnehin kein Platz mehr auf der Platte ist.
4. Nachdem Sie dies für beide Dateien ausgeführt haben, klicken Sie auf der Seite *Dateien* auf die Schaltfläche *Hinzufügen*. Es erscheint ein weiterer Eintrag in der Liste der Datenbankdateien. Aktualisieren Sie diesen Eintrag mit folgenden Einstellungen:
   - Logischer Name: MediaBase2
   - Dateityp: Zeilendaten
   - Dateigruppe: PRIMARY
   - Anfangsgröße: 20 MB
   - Automatische Vergrößerung (MB): Um 10 Prozent, unbeschränkte Vergrößerung

5. Verfahren Sie ebenso, um eine zweite Protokolldatei mit den folgenden Einstellungen hinzuzufügen:
   - Logischer Name: MediaBase2_log
   - Dateityp: Protokoll
   - Dateigruppe: Nicht zutreffend
   - Anfangsgröße: 10 MB
   - Automatische Vergrößerung (MB): Um 10 Prozent, unbeschränkte Vergrößerung
6. Wenn Sie jetzt wieder auf die Seite *Allgemein* zurückwechseln, sollten Sie dort sehen, dass der verfügbare Speicherplatz durch die neu hinzugefügten Dateien nun wieder deutlich größer geworden ist.

Anpassen der Felddefinitionen einer Tabelle

Ähnlich einfach wie das Bearbeiten von Datenbanken können Sie auch Tabellen nachträglich ändern. Die gängigsten nachträglichen Änderungen sind sicherlich das Hinzufügen eines neuen Feldes und die Vergrößerung eines bestehenden Feldes. Aber auch das Setzen von Standardwerten geschieht auf demselben Weg. Diese und andere Aktionen sind mit SQL Server glücklicherweise problemlos durchführbar, ohne dass man die Tabelle neu anlegen muss.

1. Suchen Sie im Objekt-Explorer den Eintrag für die Tabelle *DVD* und klicken Sie diesen mit der rechten Maustaste an. Wählen Sie im anschließend erscheinenden Kontextmenü die Option *Entwerfen* aus, um die Tabellendefinition zu bearbeiten.
2. Ändern Sie den Typ der Spalte *Dauer* von *tinyint* auf *int*, um auch die wenigen Filme, die länger als 255 Minuten sind, korrekt speichern zu können.
3. Klicken Sie auf die erste freie Zeile und definieren Sie dort eine neue Tabellenspalte mit den folgenden Einstellungen:
   - Spaltenname: Kategorie
   - Datentyp: varchar(80)
   - NULL-Werte zulassen: ja
4. Klicken Sie die Spalte *AnzahlDVDs* an und geben Sie bei den Spalteneigenschaften im unteren Teil des Detailbereichs bei der Eigenschaft *Standardwert oder -bindung* den Wert *1* ein. Damit haben Sie den Default-Wert für diese Spalte auf *1* festgelegt.
5. Setzen Sie auf dieselbe Weise den Standardwert für die Spalte *Laendercode* auf *2* (das ist der Ländercode für DVDs, die in Europa verkauft werden).

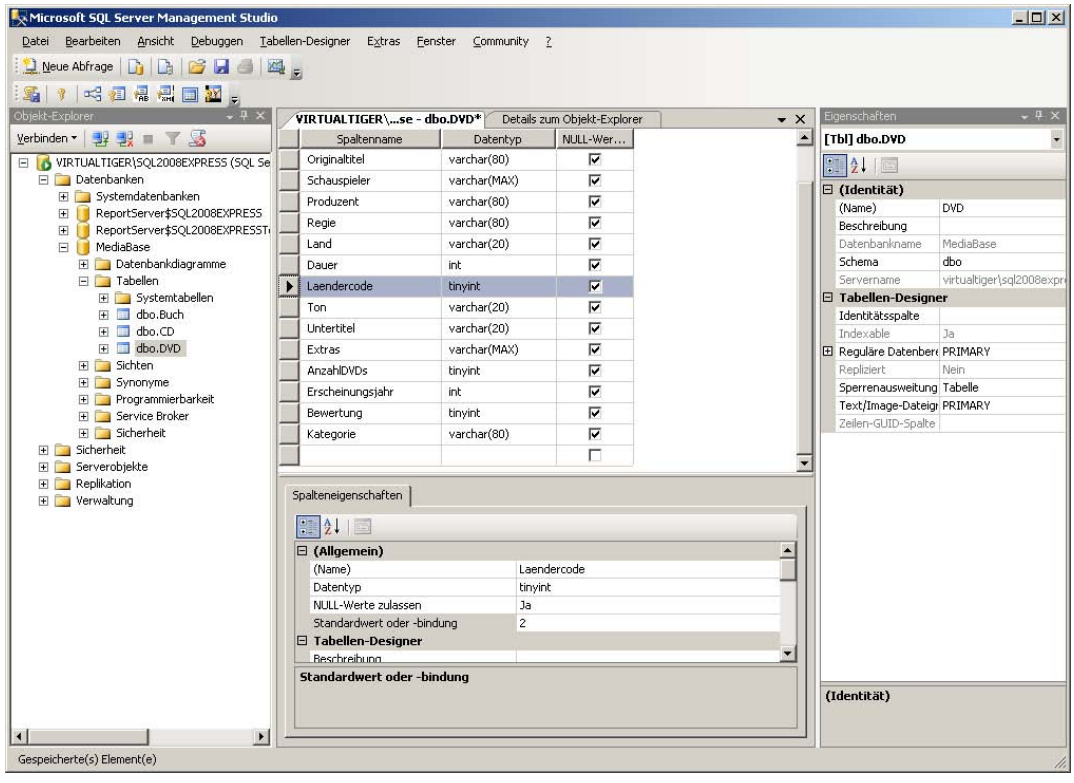


Abbildung 4.9: Nachträgliche Änderung des Tabellenentwurfs

6. In der Registerkarte des Detailbereichs ist neben dem Tabellennamen (*DVD*) ein Sternchen zu sehen, das anzeigt, dass es hier noch nicht gespeicherte Änderungen gibt. Klicken Sie nun auf das Diskettensymbol in der Symbolleiste, um die vorgenommenen Änderungen dauerhaft zu speichern.
7. Schließen Sie den Entwurfsmodus, indem Sie das kleine Kreuz in der rechten oberen Ecke des Detailbereichs anklicken.

4.4 Primärschlüssel

Beim Anlegen der Tabellen haben wir ein Feld mit *ID* bezeichnet und über das kleine gelbe Schlüsselsymbol als Primärschlüssel definiert. Damit wurde festgelegt, dass jede Zeile der Tabelle über die Angabe der ID eindeutig identifiziert werden kann (ähnlich wie man mit einem Namen einen Menschen mehr oder weniger eindeutig identifizieren kann – sofern man die üblichen Sammelbegriffe wie Müller, Meier, Schmidt mal unberücksichtigt lässt). Durch diese Primärschlüsseldefinition wird also sichergestellt, dass kein ID-Wert (in einer Tabelle) doppelt vorkommen kann.

Wenn Sie im Objekt-Explorer unterhalb von der Tabelle *Buch* den Bereich *Schlüssel* weiter aufklappen, sehen Sie dort einen Eintrag *PK_Buch*, der ebenfalls mit einem kleinen gelben Schlüsselsymbol davor dargestellt ist, das den Primarschlüssel der Tabelle *Buch* repräsentiert.

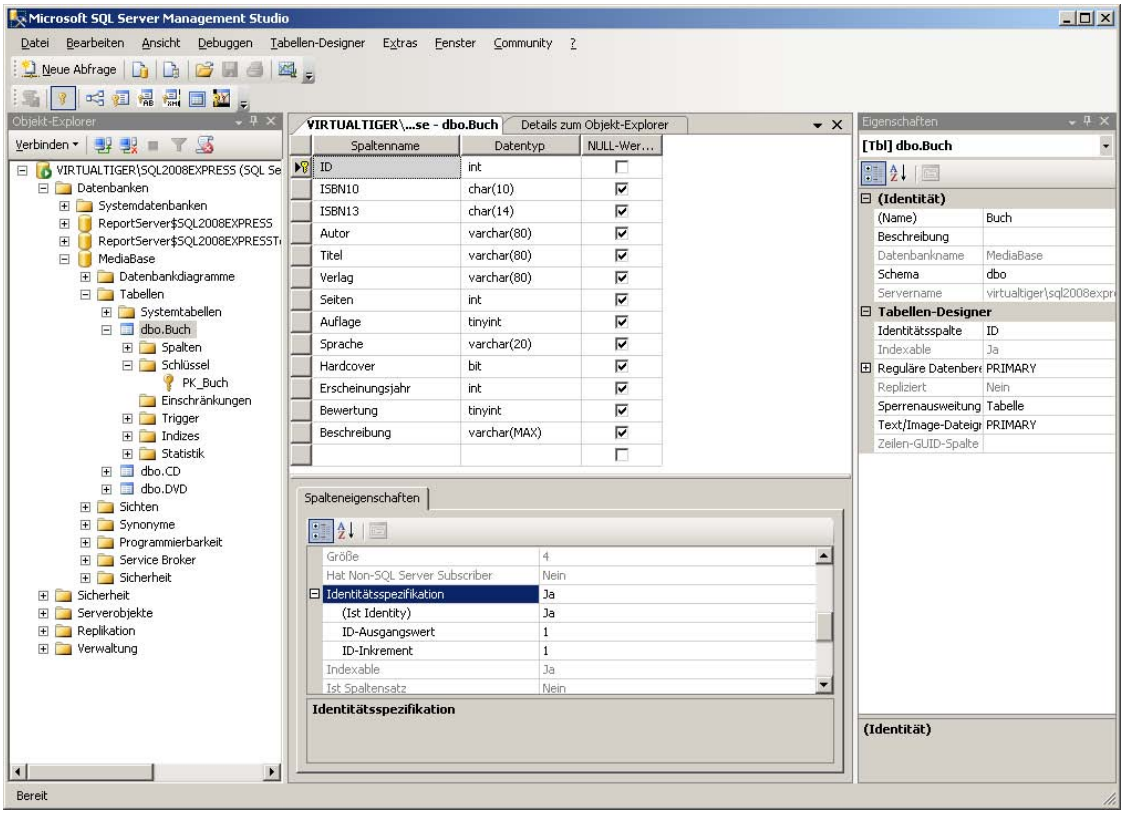


Abbildung 4.10: Setzen der Identitätsspezifikation

Nun ist es aber müßig, die IDs jedes Mal, wenn Sie eine neue Zeile erfassen, von Hand zu vergeben. Insbesondere dann, wenn die Anzeige der Daten vielleicht nach einem anderen Feld sortiert ist, ist es ja auch nicht auf den ersten Blick ersichtlich, welches die höchste bereits vergebene ID ist. Die Lösung für dieses Dilemma ist die Verwendung der Identity-Eigenschaft. Damit wird ein numerisches Feld automatisch mit fortlaufenden Zahlen gefüllt.

Passen wir nun die ID-Felder der drei angelegten Tabellen so an, dass diese mithilfe der Identity-Eigenschaft automatisch gefüllt werden.

1. Suchen Sie im Objekt-Explorer den Eintrag für die Tabelle *Buch* und klicken Sie diesen mit der rechten Maustaste an. Wählen Sie im anschließend erscheinenden Kontextmenü die Option *Entwerfen* aus, um die Tabellendefinition zu bearbeiten.
2. Klicken Sie nun in die Zeile, in der das ID-Feld definiert wird, und vergrößern Sie bei Bedarf den Bereich unten, in dem die erweiterten Eigenschaften des ausgewählten Feldes angezeigt werden.
3. Suchen Sie in den Eigenschaften den Eintrag *Identitätsspezifikation* und klappen Sie dazu die Untereinträge durch Anklicken des Pluszeichens davor auf.
4. Wenn Sie nun den Untereintrag *(Ist Identity)* auf *Ja* setzen, definieren Sie damit, dass die Werte für dieses Feld künftig automatisch vergeben werden. Mit den Einstellungen *ID-Ausgangswert* und

ID-Inkrement können Sie bei Bedarf auch den Startwert und die Schrittweise anpassen, was im Normalfall aber nicht sinnvoll ist.

5. Wenn Sie nun versuchen, die Änderungen zu speichern, erhalten Sie folgende Fehlermeldung:

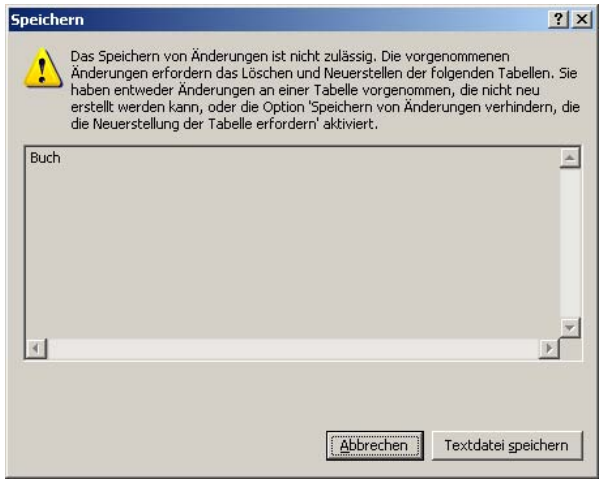


Abbildung 4.11: Fehlermeldung beim Speichern von Änderungen des Tabellenentwurfs

6. Um die Änderungen doch speichern zu können, klicken Sie im Menü *Extras* auf den Punkt *Optionen* und wählen dort den Bereich *Designer* aus. Hier können Sie das Häkchen vor der Einstellung *Speichern von Änderungen verhindern, die die Neuerstellung der Tabelle erfordern* entfernen und das Dialogfeld mit *OK* wieder schließen.

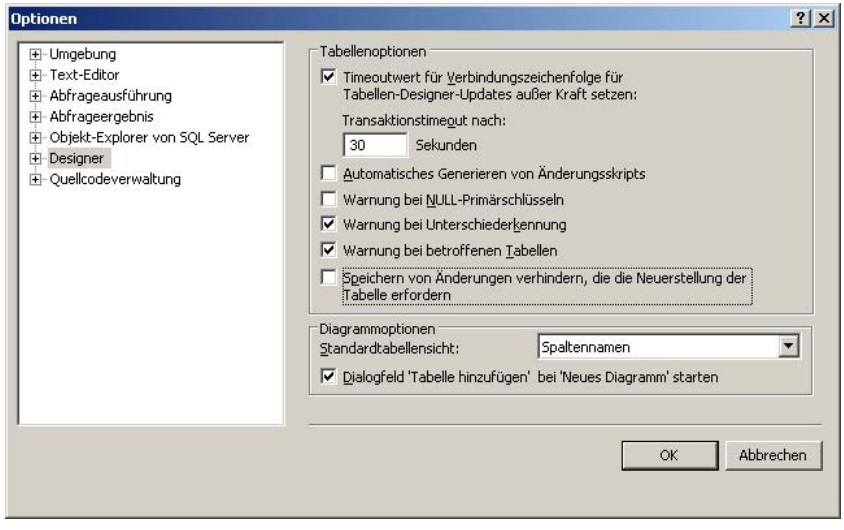


Abbildung 4.12: Die Optionen des Designers im SQL Server Management Studio

7. Nun lassen sich auch die Änderungen am Tabellenentwurf speichern.

8. Wenn Sie die Tabelle nun wieder im Bearbeitungsmodus öffnen, werden Sie feststellen, dass das Feld *ID* nun nicht mehr von Ihnen eingegeben werden kann, sondern in dem Moment automatisch gesetzt wird, wenn Sie einen neuen Datensatz speichern.

Hinweis: Global eindeutige Werte durch Uniqueidentifier

Die gerade genannten IDs sind nur innerhalb einer Tabelle eindeutig. Wenn Sie Primärschlüssel benötigen, die auch tabellenübergreifend (und sogar rechnerübergreifend) eindeutig sind, können Sie anstelle eines numerischen Datentyps den Typ *Uniqueidentifier* verwenden. Dabei handelt es sich um eine 32-stellige hexadezimale Zahl (entspricht 16 Byte), deren Werte Sie über die *newid()*-Funktion setzen können. Die so generierten Werte sind praktisch weltweit eindeutig, weshalb diese Werte auch als *globally unique identifier* bezeichnet werden. Damit eignet sich dieser Datentyp hervorragend für Daten, die auf verschiedenen (nicht miteinander verbundenen) Rechnern erfasst und später in einer Tabelle zusammengeführt werden, ohne dass dabei die Primärschlüssel neu gesetzt werden müssen.

Den Effekt des automatischen Vorbelegens des Primärschlüssels können Sie bei Unique Identifiern erreichen, indem Sie die *newid()*-Funktion als Standardwert des Feldes eintragen.

4.5 Indizes

Unabhängig davon, ob Sie für das ID-Feld die Identity-Eigenschaft nutzen oder nicht, wurde durch die Definition als Primärschlüssel dafür automatisch ein Index erstellt. Dies lässt sich nachvollziehen, wenn man im Objekt-Explorer unterhalb der Tabelle *Buch* den Zweig *Indizes* aufklappt und dort den Eintrag *PK_Buch (gruppiert)* vorfindet. Es wurde also ein gruppierter Index mit Namen *PK_Buch* erstellt.

Doch was hat es mit Indizes genau auf sich?

Funktionsweise von Indizes

Indizes sind Hilfskonstrukte, die insbesondere das Suchen und Sortieren nach bestimmten Feldern beschleunigen. Dabei unterscheidet man zwischen gruppierten (clustered) Indizes und nicht gruppierten (non clustered) Indizes.

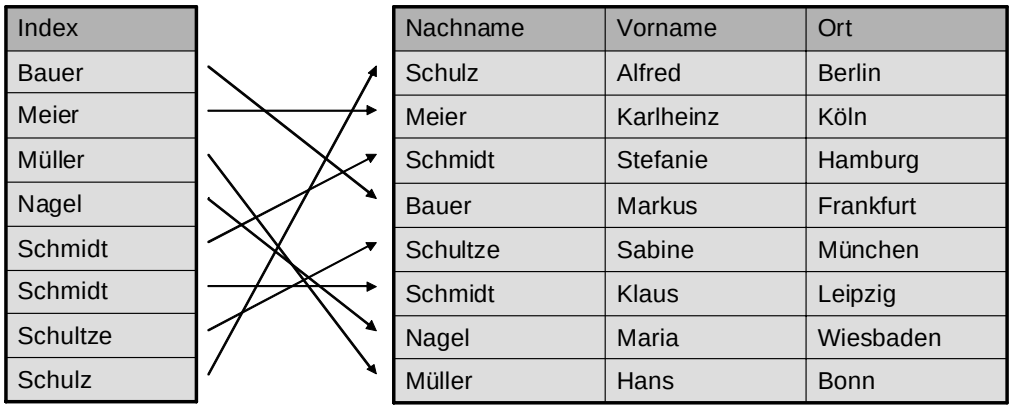
Am einfachsten lässt sich das am Beispiel eines Buches erklären: Wenn Sie in einem Buch nach allen Vorkommen eines bestimmten Begriffs suchen, müssten Sie ohne einen Index das gesamte Buch komplett lesen. Übertragen auf die Datenbankwelt entspricht dies dem kompletten Lesen einer gesamten Tabelle (auch Table Scan genannt). Dies kostet natürlich extrem viel Zeit, was es mithilfe eines Index zu optimieren gilt.

Wenn das Buch aber über ein Stichwortverzeichnis verfügt, können Sie dort – aufgrund der alphabetischen Sortierung des Stichwortverzeichnisses – gezielt nachschlagen, um dann die dort beim gesuchten Begriff angegebene Seite direkt aufzuschlagen. Die Entsprechung bei den Datenbanken ist ein nicht gruppierter Index, in dem aufgrund seiner Struktur gezielt gesucht werden kann (man nennt dies auch Index Seek).

Für den Fall, dass die genaue Schreibweise des gesuchten Begriffs nicht bekannt ist oder dieser auch als Bestandteil eines zusammengesetzten Begriffs vorkommen kann, muss das gesamte Stichwortverzeichnis gelesen werden, um die Verweise auf die relevanten Seiten zu finden (was immer noch deutlich weniger Aufwand ist, als das ganze Buch lesen zu müssen). Datenbankseitig wäre dies ein nicht gruppierter Index, der komplett gelesen werden muss, weil die vorsortierte Reihenfolge nicht sinnvoll genutzt werden kann (Index Scan).

Im Gegensatz zur gängigen Praxis bei Büchern gibt es bei Tabellen oft mehr als einen Index. Dazu können Datenbankindizes auch aus mehreren Feldern bestehen, während ein Stichwortverzeichnis in einem Buch normalerweise aus ein bis maximal zwei Ebenen besteht.

In vereinfachter Form kann man sich einen nicht gruppierten Index als sortierte Liste vorstellen, deren Einträge auf die entsprechende Zeile in der eigentlichen Tabelle verweisen.

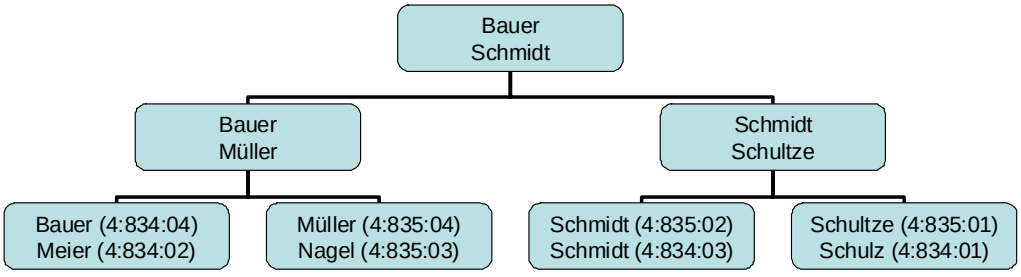


Index
Bauer
Meier
Müller
Nagel
Schmidt
Schmidt
Schultze
Schulz

Nachname	Vorname	Ort
Schulz	Alfred	Berlin
Meier	Karlheinz	Köln
Schmidt	Stefanie	Hamburg
Bauer	Markus	Frankfurt
Schultze	Sabine	München
Schmidt	Klaus	Leipzig
Nagel	Maria	Wiesbaden
Müller	Hans	Bonn

Abbildung 4.13: Nicht gruppierter Index (vereinfachte Darstellung)

Die eigentliche technische Umsetzung ist allerdings etwas komplexer: Die eigentlichen Zeilendaten der Tabelle sind auf 8 KB große Speicherseiten verteilt. Der Index liegt in Form eines balancierten Baumes vor, dessen Blätter auf die Speicherseiten verweisen. Somit kann mit nur wenigen Vergleichen die richtige Speicherseite gefunden werden.



Datei 4

Seite 834

	Nachname	Vorname	Ort
01	Schulz	Alfred	Berlin
02	Meier	Karlheinz	Köln
03	Schmidt	Stefanie	Hamburg
04	Bauer	Markus	Frankfurt

Seite 835

	Nachname	Vorname	Ort
01	Schultze	Sabine	München
02	Schmidt	Klaus	Leipzig
03	Nagel	Maria	Wiesbaden
04	Müller	Hans	Bonn

Abbildung 4.14: Nicht gruppierter Index (korrekte Darstellung)

Bei einem gruppierten Index dagegen liegen die Daten selbst in sortierter Form vor. Die Entsprechung bei einem Buch wäre beispielsweise ein Lexikon, das ja auch nach den Schlagwörtern sortiert ist. Dadurch kann der Zugriff auf die Daten noch etwas schneller erfolgen als über einen nicht gruppierten Index. Außerdem ist zu beachten, dass es natürlich nur einen gruppierten Index pro Tabelle geben kann, denn die Daten selbst können ja nur in einer Reihenfolge sortiert sein.

Während Indizes die Abfrage von Daten deutlich beschleunigen können, werden Datenänderungen (einfügen, ändern oder löschen) etwas langsamer, da neben den eigentlichen Daten auch die Indizes aktualisiert werden müssen. Besonders aufwendig sind diese Aktualisierungen bei gruppierten Indizes. Aufgrund dieser Tatsache sollte man Indizes bewusst einsetzen und gezielt für solche Felder definieren, nach denen oft gesucht oder sortiert wird. Insbesondere für gruppierte Indizes bieten sich solche Felder an, deren Inhalte sich nachträglich nicht mehr oder nur sehr selten ändern.

Erstellen von Indizes

Erstellen wir nun einige zusätzliche Indizes für die Tabelle *Buch*.

1. Suchen Sie im Objekt-Explorer den Eintrag für die Tabelle *Buch* und klicken Sie darunter den Eintrag *Indizes* mit der rechten Maustaste an. Wählen Sie im anschließend erscheinenden Kontextmenü die Option *Neuer Index* aus, um das Dialogfeld für die Indexerstellung zu öffnen.
2. Geben Sie im Feld *Indexname* die Bezeichnung *IX_Titel* an.
3. Lassen Sie den *Indextyp* auf *Nicht gruppiert* eingestellt, denn für die Tabelle gibt es ja bereits einen gruppierten Index.

4. Wenn Sie das Kästchen *Eindeutig* anklicken, können Sie erreichen, dass jeder Inhalt des Index nur einmal vorkommen darf. Da diese Option für einzelne Felder auch direkt auf Feldebene gesetzt werden kann, lassen Sie das Kästchen hier leer.
5. Fügen Sie über die Schaltfläche *Hinzufügen* das Feld *Titel* zum Index hinzu, das dann in der Tabelle *Indexschlüsselspalten* angezeigt wird

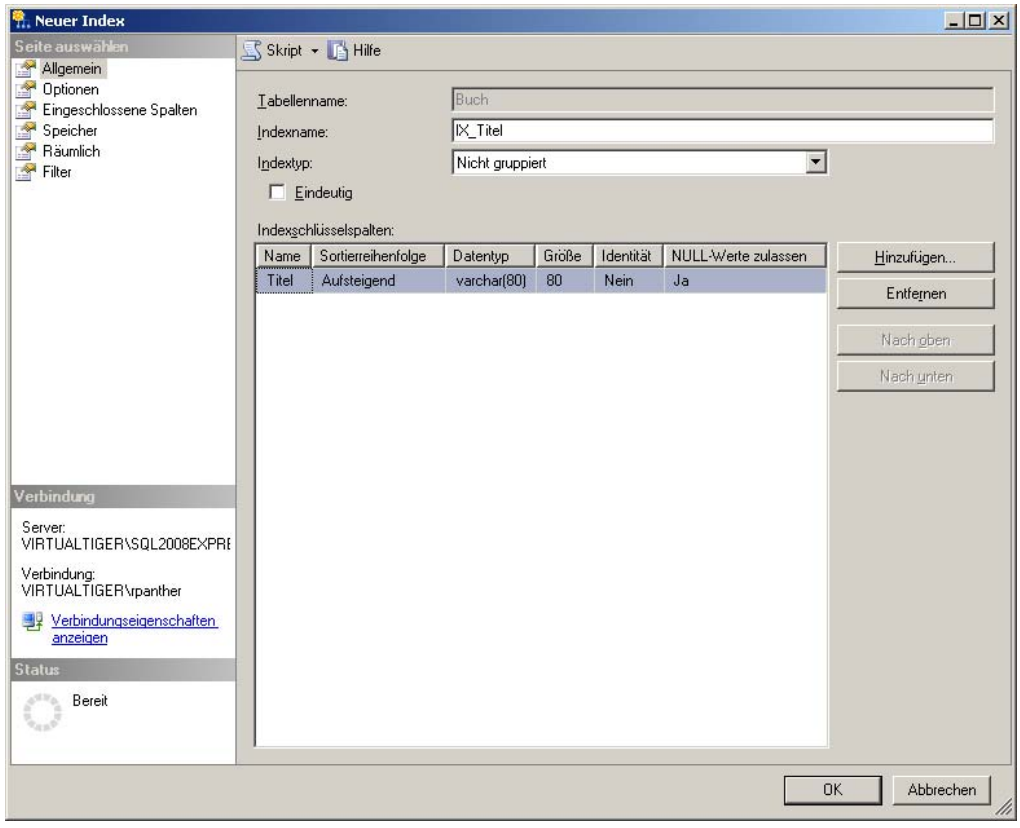


Abbildung 4.15: Das Dialogfeld zum Erstellen von Indizes

Hinweis: Mehrspaltige Indizes

Sie können auch Indizes erstellen, die mehrere Spalten beinhalten. Hier ist dann die Reihenfolge maßgeblich, die Sie mit den entsprechenden Schaltflächen *Nach oben* und *Nach unten* modifizieren können. Wenn Sie den Index auch nutzen möchten, um nur nach einer der angegebenen Spalten zu suchen, sollte diese zuerst in der Liste stehen.

6. Wenn Sie das Dialogfeld nun mit *OK* schließen, wird der Index erstellt und daraufhin im Objekt-Explorer angezeigt: *IX_Titel (nicht eindeutig, nicht gruppiert)*.
7. Erstellen Sie nun nach demselben Verfahren drei weitere Indizes für die Spalten *ISBN10*, *ISBN13* und *Autor*.

4.6 Übungen zu diesem Kapitel

In diesem Abschnitt finden Sie einige Übungen zu diesem Kapitel. Die richtigen Antworten und Lösungen finden Sie wie immer auf der Website *www.vsxpress.de.*

Übung 4.1

Was können Sie tun, wenn auf der aktuellen Partition kein Platz mehr ist, um die Datenbankdatei weiter automatisch zu vergrößern?

Übung 4.2

Fügen Sie für die Tabellen *Buch* und *CD* ebenfalls eine Spalte *Kategorie* mit den folgenden Einstellungen hinzu:

- Datentyp: varchar(80)
- NULL-Werte zulassen: ja

Übung 4.3

Setzen Sie in der Tabelle *Buch* den Standardwert für das Feld *Sprache* auf *deutsch*.

Setzen Sie in der Tabelle *CD* den Standardwert für das Feld *AnzahlCDs* auf *1*.

Übung 4.4

Definieren Sie die Spalte *ID* in den Tabellen *CD* und *DVD* als Identitätsspalte.

Übung 4.5

Erstellen Sie folgende Indizes für die Tabelle *CD*:

- IX_Titel (Spalte: Titel, nicht eindeutig, nicht gruppiert)
- IX_Interpret (Spalte: Interpret, nicht eindeutig, nicht gruppiert)

Erstellen Sie folgende Indizes für die Tabelle *DVD*:

- IX_Titel (Spalte: Titel, nicht eindeutig, nicht gruppiert)
- IX_Originaltitel (Spalte: Originaltitel, nicht eindeutig, nicht gruppiert)

4.7 Zusammenfassung

Sie haben in diesem Kapitel die wichtigsten Grundlagen kennengelernt, um mit SQL Server Management Studio Datenbanken und Tabellen anzulegen und zu bearbeiten.

Dabei wurden beim Erstellen von Datenbanken sowohl die Initialgrößen von Zeilendaten- und Protokolldatei festgelegt als auch deren automatische Vergrößerung konfiguriert. Anschließend wurden Tabellen angelegt und dabei die wichtigsten Datentypen verwendet:

- Numerische Datentypen: bit, tinyint, int, money, real
- Alphanumerische Datentypen: char(n), varchar(n), varchar(max)
  bzw. deren Unicode-Entsprechungen: nchar(n), nvarchar(n), nvarchar(max)
- Zeit- und Datumstypen: date, time, datetime
- Sonstige Datentypen: varbinary(max), uniqueidentifier, xml

Mit dem SQL Server Management Studio kann man sowohl Datenbankeinstellungen nachträglich ändern als auch Tabellenstrukturen anpassen. Davor wurden aber die Inhalte der erstellten Tabellen angezeigt und bearbeitet.

Um die Zeilen einer Tabelle später einfacher identifizieren zu können, wurden Primärschlüssel definiert, die mithilfe der Identitätseigenschaft automatisch einen eindeutigen Wert zugewiesen bekommen. Diese Primärschlüssel wurden implizit als gruppierte Indizes angelegt, nach denen die Zeilendaten sortiert angelegt werden. Anschließend wurden zusätzlich einige nicht gruppierte Indizes angelegt, durch die man schneller nach den darin verwendeten Spalten suchen kann.

Damit wurden in diesem Kapitel die wichtigsten Mittel behandelt, um mit einzelnen Tabellen arbeiten zu können. Wie man Tabellen miteinander in Beziehung setzen kann, ist Thema des nächsten Kapitels.

Kapitel 5

Eine Tabelle kommt selten allein

In diesem Kapitel lernen Sie

- wozu Relationen und Fremdschlüssel verwendet werden
- wie und warum man Datenbanken normalisiert
- die Verwendung von Datenbankdiagrammen
- wie man Datenbankabfragen erzeugen kann, ohne SQL-Code schreiben zu müssen
- die Erstellung und Verwendung von Sichten

Bis jetzt haben wir lediglich mit einzelnen Tabellen gearbeitet, die in keinem direkten Zusammenhang stehen. Im »richtigen Leben« dürfte das eher der Ausnahmefall sein, zumal relationale Datenbanksysteme ihre Stärken eben genau dann ausspielen können, wenn die Tabellen miteinander in Beziehung stehen.

5.1 Relationen und Fremdschlüssel

Nehmen wir uns einmal die Tabelle *CD* vor. Hier sind alle Songtitel, die auf der CD enthalten sind, in einem einzigen Feld abgespeichert. Das ist zwar einerseits relativ einfach in der Handhabung, aber auch sehr unflexibel.

Die bessere Variante liegt darin, eine separate Tabelle anzulegen, in der ein Datensatz pro Track gespeichert ist, zusammen mit einem Hinweis, zu welcher CD dieser Track gehört. Dieser Hinweis stellt eine Beziehung zwischen beiden Tabellen dar – eine sogenannte Relation.

Zur technischen Umsetzung dieser Relation wird ein Feld benötigt, das auf den Primärschlüssel der CD-Tabelle verweist. Da dieses Feld auf den Primärschlüssel einer »fremden« Tabelle verweist, wird es Fremdschlüssel genannt.

Hintergrundinfo: Verschiedene Typen von Relationen

Je nach Anzahl der passenden Datensätze auf beiden Seiten der Relation gibt es verschiedene Relationstypen:

- **1:1-Relation** Jeder Datensatz der Tabelle A hat genau einen passenden Datensatz in der Tabelle B. Dies wird realisiert, indem ein Fremdschlüssel in einer der beiden Tabellen auf den Primärschlüssel der anderen Tabelle zeigt. Dieser Relationstyp tritt in der Praxis nicht sehr häufig auf, da man für diesen Fall die Felder der Tabellen A und B auch in einer Tabelle zusammen ablegen kann.
- **1:n-Relation** Für jeden Datensatz in der Tabelle A gibt es n (einen oder mehrere) passende Datensätze in der Tabelle B. Dies wird realisiert, indem ein Fremdschlüssel in der Tabelle B auf den Primärschlüssel der Tabelle A verweist. Ein Praxisbeispiel dafür sind die Tabellen *CD* und *CDTracks,* deren Relation weiter unten in diesem Kapitel angelegt wird.
- **m:n-Relation** Für jeden Datensatz der Tabelle A gibt es einen oder mehrere Datensätze in der Tabelle B, für jeden Datensatz der Tabelle B können aber auch mehrere Datensätze in der Tabelle A existieren. Die Umsetzung im Datenmodell sieht so aus, dass zwischen den Tabellen A und B eine Hilfstabelle eingefügt wird, die Fremdschlüssel sowohl für die Tabelle A als auch für die Tabelle B beinhaltet. Ein typisches Beispiel aus der Praxis findet sich in fast jeder Adressverwaltung. Hier können an einer Adresse mehrere Personen wohnen, aber es kann auch für jede Person mehrere Adressen geben (Privatadresse, Firmenadresse etc.).

Legen wir nun zuerst die neue Tabelle mit dem Namen *CDTrack* an:

1. Starten Sie SQL Server 2008 Management Studio und verbinden Sie sich mit der lokalen Serverinstanz *SQL2008Express*.
2. Suchen Sie im Objekt-Explorer die Datenbank *MediaBase* und legen Sie dort die Tabelle *CDTrack* mit den folgenden Feldern an:

Tabelle 5.1: Die Felder der Tabelle *CDTrack*

Spaltenname	Datentyp	NULL-Werte zulassen	Sonstiges
ID	int	nein	Primärschlüssel
idCD	int	nein	
CDNr	tinyint	ja	Standardwert 1
TrackNr	tinyint	ja	
Titel	varchar(80)	ja	
Interpret	varchar(80)	ja	
Dauer	time	ja	
Bewertung	tinyint	ja	

Während die Bedeutung der meisten Felder offensichtlich ist, sollte erwähnt werden, dass die Spalte *TrackNr* die Nummer des Tracks auf der CD enthält. Die Spalte *CDNr* dagegen ist für die laufende CD-Nummer einer Mehrfach-CD (z.B. Doppel-CD) vorgesehen und erhält den Standardwert *1*, da die meisten CD-Pakete natürlich einfache CDs sind.

3. Bevor Sie die Tabelle speichern, achten Sie darauf, die Spalte *ID* als Primärschlüssel zu definieren und für diese Spalte auch die Identitätsspezifikation mit Ausgangswert *1* und Inkrement *1* zu setzen.

Best Practices: Benennung von Tabellen

Sie fragen sich vielleicht, warum die Tabelle nicht einfach *Track* – oder gar *Tracks* – genannt wurde. Die Einzahl wurde für den Tabellennamen verwendet, weil ein Datensatz genau einem Track entspricht (auch wenn die gesamte Tabelle später natürlich mehrere Tracks beinhaltet). Das alte Feld *Songs* in der Tabelle *CD* dagegen beinhaltete mehrere Songtitel in einem Datensatz (daher wurde dafür die Mehrzahl verwendet).

Das *CD* wurde der Tabelle vorangestellt, um zu verdeutlichen, dass diese Tabelle Detailinfos zu den Daten der Tabelle *CD* enthält. Dies ist zwar nicht zwingend notwendig, macht die Darstellung im Objekt-Explorer aber deutlich übersichtlicher, da die Tabellen *CD* und *CDTrack* somit direkt untereinander stehen.

Nachdem nun die Tabelle erstellt ist, muss noch die Verbindung zwischen den Tabellen *CD* und *CDTrack* hergestellt werden:

1. Klicken Sie mit der rechten Maustaste im Objekt-Explorer unterhalb der neuen Tabelle *dbo.CDTrack* auf den Eintrag *Schlüssel* und wählen Sie die Option *Neuer Fremdschlüssel*. Es erscheint das Dialogfeld *Fremdschlüsselbeziehungen*, in dem bereits ein Fremdschlüssel eingetragen wurde, der nur noch angepasst werden muss.
2. Klicken Sie auf die Zeile *Tabellen- und Spaltenspezifikation* und anschließend auf die Schaltfläche mit den drei Punkten, die in dieser Zeile erscheint. Dadurch erscheint ein weiteres Dialogfeld, in dem Sie die Zieltabelle und die Spalten festlegen können, mit denen die Relation erstellt wird. Stellen Sie die Primärschlüsseltabelle auf *CD* und die Spalte dazu auf *ID*.

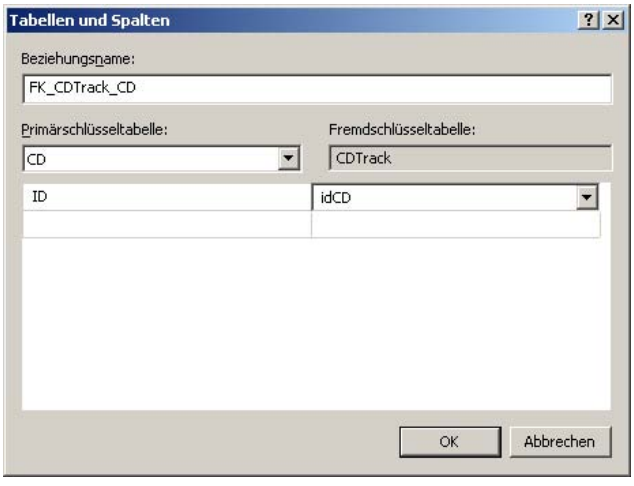


Abbildung 5.1: Definition der Primär- und Fremdschlüsselspalten

3. Die Fremdschlüsseltabelle ist auf *CDTrack* voreingestellt. Ändern Sie hier die darunter angezeigte Spalte auf *idCD*. Der Beziehungsname wird automatisch auf *FK_CDTrack_CD* geändert, um zu

verdeutlichen, dass hier ein Fremdschlüssel (FK = Foreign Key) definiert wird, der von der Tabelle *CDTrack* auf die Tabelle *CD* verweist.

4. Schließen Sie das Dialogfeld mit *OK*. Im Dialogfeld für Fremdschlüsselbeziehungen wurde nun der Name sowie die Tabellen- und Spaltenspezifikation (sofern Sie diese mit dem Pluszeichen davor aufklappen) aktualisiert.

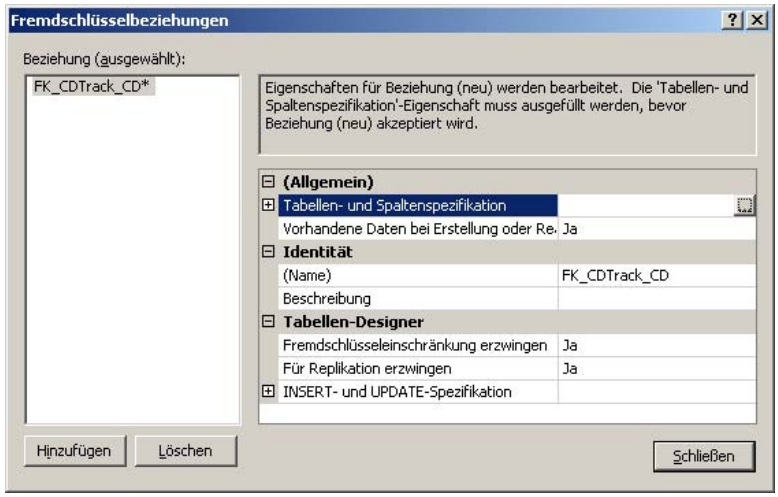


Abbildung 5.2: Das Dialogfeld zum Definieren der Fremdschlüsselbeziehungen

5. Über den Knoten *INSERT- und UPDATE-Spezifikation* weiter unten können Sie sogenannte Integritätsregeln auswählen, die festlegen, wie beim Aktualisieren oder Löschen des übergeordneten Datensatzes (also des entsprechenden Eintrags in der *CD*-Tabelle) verfahren werden soll. *Regel aktualisieren* gibt an, was geschieht, wenn eine ID in der *CD*-Tabelle geändert wurde (was durch die Identitätsspezifikation eigentlich nicht vorkommen kann). *Regel löschen* definiert, was mit den Fremdschlüsseln geschieht, wenn ein Datensatz in der *CD*-Tabelle, auf den diese verweisen, gelöscht wird: Mit *Keine Aktion* erfolgt keine Änderung, mit *Weitergabe* werden auch die entsprechenden Datensätze in der *CDTrack*-Tabelle automatisch gelöscht und mit den Optionen *NULL festlegen* und *Standard festlegen* wird der Fremdschlüssel (*idCD*) auf NULL bzw. den definierten Standardwert gesetzt.

Hinweis: Prüfung der referenziellen Integrität

Sofern die Einstellung *Fremdschlüsseleinschränkung erzwingen* aktiviert ist, erfolgt eine Fehlermeldung, sobald ein Zustand entstehen würde, der die über Fremdschlüssel definierte referenzielle Integrität verletzt. Würde also nach Löschung eines CD-Datensatzes in der Tabelle *CDTrack* ein Verweis auf eine nicht vorhandene CD übrig bleiben, wird die Löschung des CD-Datensatzes verhindert (es sei denn, die Lösch-Weitergabe wurde aktiviert, wodurch auch die entsprechenden *CDTrack*-Einträge automatisch gelöscht werden, oder aber der Fremdschlüssel wird durch die entsprechende Spezifikation automatisch auf einen anderen gültigen Wert gesetzt).

Dieselbe Prüfung erfolgt natürlich auch, wenn ein Datensatz in die Tabelle *CDTrack* eingefügt oder darin geändert wird (daher auch der Name *INSERT- und UPDATE-Spezifikation*).

6. Schließen Sie nun das Dialogfeld für die Definition der Fremdschlüsselbeziehungen über die Schaltfläche *Schließen*. Sie befinden sich nun im Entwurfsmodus der Tabelle *CDTrack*. Damit der gerade definierte Fremdschlüssel auch wirklich gespeichert wird, müssen Sie in der Symbolleiste auf das Diskettensymbol zum Speichern klicken oder den Entwurfsmodus verlassen und die Frage, ob die Änderungen an der Tabelle *dbo.CDTrack* gespeichert werden sollen, mit *Ja* beantworten. Anschließend erfolgt noch ein Hinweis, dass sich die Änderungen ebenfalls auf die Tabelle *dbo.CD* auswirken, der ebenfalls positiv zu bestätigen ist.

Nun sollten Sie noch ein paar Indizes für die Tabelle *CDTrack* definieren:

1. Erstellen Sie zuerst separate Indizes für die Spalten Titel (*IX_Titel*) und Interpret (*IX_Interpret*), damit in diesen später schneller gesucht werden kann.
2. Erstellen Sie einen weiteren Index *IX_idCD* für die Fremdschlüsselspalte. Mit diesem wird später ein schnellerer Zugriff auf alle Titel, die zu einer CD gehören, möglich.
3. Wenn Sie nun im Objekt-Explorer die Bereiche *Schlüssel* und *Indizes* für die Tabelle *CDTrack* aktualisieren und einblenden, sollten Sie folgende Darstellung erhalten:

Abbildung 5.3: Schlüssel und Indizes der Tabelle *dbo.CDTrack*

4. Da die Songtitel nun in der separaten Tabelle *CDTrack* gespeichert werden, können Sie nun die Spalte *Songs* aus der Tabelle *CD* löschen, indem Sie im Objekt-Explorer die Spaltenauflistung dieser Tabelle suchen, die Spalte *Songs* mit der rechten Maustaste anklicken und die Option *löschen* auswählen. (Sollten Sie hier bereits Informationen erfasst haben, sollten Sie diese vorher natürlich in die Tabelle *CDTrack* übernehmen.)

5.2 Normalisierung

Alternativ zu der im vorigen Abschnitt beschriebenen Aufteilung auf zwei Tabellen hätte man natürlich auch alle Informationen in der *CDTrack*-Tabelle ablegen können, indem man diese um den Namen der CD, das Erscheinungsjahr etc. ergänzt. Dies hätte allerdings den Nachteil, dass alle diese Informationen mehrfach (redundant) – nämlich einmal pro Track – gespeichert würden, was natürlich unnötig Platz belegt und auch noch ein paar weitere Probleme mit sich bringt.

Das bewusste Aufteilen von logisch zusammengehörenden Daten auf mehrere Tabellen, um eine saubere Struktur zu erhalten, nennt sich Normalisierung. Damit werden vor allem folgende Ziele erreicht:

- **Vermeidung von redundanten Informationen** Informationen, die für mehrere Zeilen gleich sind, werden nicht mehrfach gespeichert, sondern in einer separaten Tabelle abgelegt.
- **Einsparung von Speicherplatz** Dadurch, dass identische Informationen nicht mehrfach gespeichert werden, wird Speicherplatz eingespart.
- **Änderungen an nur einer Stelle nötig** Sollte eines der Felder, die für mehrere Zeilen gleich sind, geändert werden müssen, muss diese Information in der normalisierten Form nur noch an einer Stelle geändert werden, wobei sich diese Änderung trotzdem automatisch auf alle dazugehörenden Zeilen auswirkt.

Hintergrundinfo: Normalformen

In der Datenbanktheorie wird die Normalisierung über verschiedene Normalformen definiert, die bestimmte Bedingungen an die Datenbankstruktur enthalten. Die wichtigsten dieser Normalformen (die alle aufeinander aufbauen) sind die ersten drei:

- **1. Normalform** Jedes Feld der Tabelle muss einen atomaren Wertebereich haben. Damit ist gemeint, dass zur Erfüllung der 1. Normalform keine zusammengesetzten Felder erlaubt sind.

  Beispiel: Ein Namensfeld, das Vor- und Nachname enthält, verletzt die erste Normalform. Ebenso sieht es bei Adressfeldern aus, die Straße und Hausnummer oder Postleitzahl und Ort enthalten. Damit die erste Normalform erfüllt wird, müssen diese Informationen auf separate Felder verteilt werden.

- **2. Normalform** Jedes Nichtschlüsselfeld ist vom Primärschlüssel (nicht nur von einem Teil davon) voll funktional abhängig. Außerdem muss die Bedingung der 1. Normalform erfüllt sein.

  Beispiel: Eine Tabelle hat als Primärschlüssel die Kombination der Felder *Nachname* und *Vorname*. Wenn diese Tabelle Felder beinhaltet, die lediglich vom Nachnamen abhängig sind (z.B. die Adresse), ist die 2. Normalform nicht erfüllt. Dazu müssten diese Felder in eine zweite Tabelle ausgelagert werden, deren Primärschlüssel der Nachname ist.

- **3. Normalform** Alle Nichtschlüsselfelder müssen direkt (nicht indirekt) vom Primärschlüssel abhängen. Außerdem müssen die Bedingungen der 2. Normalform erfüllt sein.

  Beispiel: In einer Adresstabelle hängt die Postleitzahl direkt vom Primärschlüssel ab. Der Ortsname dagegen hängt nicht vom Primärschlüssel, sondern vielmehr von der Postleitzahl ab (zumindest wenn man für dieses Beispiel vernachlässigt, dass es Postleitzahlen gibt, die mehrere Orte betreffen). Damit dieses Modell die dritte Normalform erfüllt, muss eine separate Tabelle angelegt werden, die alle Ortsnamen enthält und deren Primärschlüssel die Postleitzahl ist.

Es existiert noch eine ganze Menge weiterer Normalformen, die allerdings sehr speziell werden und deren Erläuterung an dieser Stelle sicherlich zu weit führen würde.

Es gibt aber auch Situationen, in denen redundante Informationen bewusst in Kauf genommen werden oder manchmal sogar gezielt denormalisiert wird (also normalisiert vorliegende Tabellen in eine Gesamttabelle aufgelöst werden). Letzteres ist beispielsweise Grundlage für Data Warehouses, in denen die Daten für die Anzeige und Auswahl optimiert aufbereitet werden, wofür allerdings zusätzlicher Speicherplatz (in teilweise erheblichem Ausmaß) benötigt wird.

Die für Sie im Moment relevantere Variante ist aber das bewusste Akzeptieren von Redundanzen, wenn der für die Normalisierung nötige Verwaltungsaufwand stärker ins Gewicht fällt als der Vorteil, den man durch die Normalisierung erhält. Das ist insbesondere dann der Fall, wenn die redundanten Felder eine relativ geringe Größe haben und nur selten geändert werden. Es würde beispielsweise nicht unbedingt Sinn machen, eine separate Tabelle anzulegen, in der alle verfügbaren Interpreten gespeichert sind, da sich diese für eine CD im Nachhinein nicht mehr ändern. Auch die zu erwartende Speicherplatzersparnis wäre verhältnismäßig gering im Vergleich zu der Komplexität, die durch eine weitere Tabelle hinzukommt. Allgemein lässt sich sagen, dass die Normalisierung vor allem die sinnvolle Verwaltung von Datenänderungen unterstützt, während denormalisierte Daten eher für lesenden Zugriff Sinn machen können.

Generell sollte man die Grundprinzipien der Normalisierung zwar kennen, sie aber nicht allzu dogmatisch anwenden, sondern immer nach gesundem Menschenverstand entscheiden, wie die Felder auf Tabellen aufzuteilen sind.

5.3 Datenbankdiagramme

Zur übersichtlichen Darstellung der Zusammenhänge von Tabellen können Sie im SQL Server Management Studio Datenbankdiagramme nutzen. Dabei wählen Sie zuerst die Tabellen aus, die in einem Diagramm dargestellt werden sollen. Anschließend werden die Tabellen mit ihren Relationen dargestellt. Gerade bei komplexeren Datenbanken macht es Sinn, Diagramme für einzelne Teilbereiche des Datenmodells zu definieren. Bei überschaubaren Datenmodellen dagegen können Sie auch alle Tabellen auf einmal darstellen.

Erstellen von Datenbankdiagrammen

Erstellen Sie nun ein Datenbankdiagramm, das alle bisher erstellten Tabellen und deren Relationen darstellt:

1. Im Objekt-Explorer finden Sie unterhalb des Eintrags für die Datenbank *MediaBase* einen Punkt *Datenbankdiagramme*. Sobald Sie diesen anklicken, erhalten Sie eine Meldung, dass hierfür noch ein paar Unterstützungsobjekte angelegt werden müssen, die Sie mit *Ja* bestätigen. Die Erstellung dieser Unterstützungsobjekte kann einen Moment dauern, ist aber nur einmalig erforderlich.

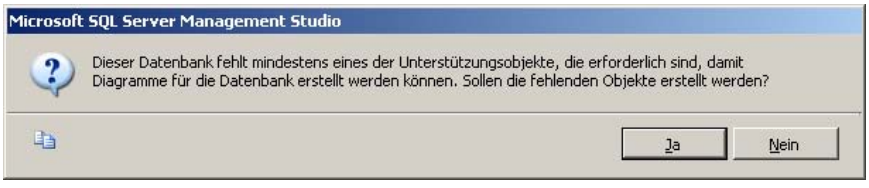


Abbildung 5.4: Rückfrage zur Erstellung von Datenbankunterstützungsobjekten

2. Nachdem die Unterstützungsobjekte angelegt sind, klicken Sie mit der rechten Maustaste in den Detailbereich (oder auf den Eintrag *Datenbankdiagramme* im Objekt-Explorer) und wählen die Option *Neues Datenbankdiagramm*.

3. Es erscheint ein Dialogfeld, mit dem Sie komfortabel Tabellen zum Diagramm hinzufügen können. Da die Datenbank noch recht übersichtlich ist, können Sie hier alle vier Tabellen auswählen und über die Schaltfläche *Hinzufügen* im Datenbankdiagramm einbinden. Anschließend schließen Sie das Dialogfeld mit der entsprechenden Schaltfläche, um das erstellte Diagramm zu sehen.

Tipp: Mehrfachauswahl in Listen

Um mehrere Einträge in einer Liste auszuwählen, klicken Sie zuerst den ersten Eintrag an. Danach können Sie entweder alle weiteren Einträge mit gedrückter Strg-Taste anklicken, oder aber Sie klicken den letzten Eintrag mit gedrückter Shift-Taste an, damit alle Einträge zwischen dem ersten und letzten Eintrag ausgewählt werden. Beide Vorgehensweisen sind nicht nur im SQL Server Management Studio, sondern in den meisten Windows-Anwendungen verfügbar.

4. In dem Diagramm werden alle vier Tabellen mit ihren Spalten angezeigt. Zwischen den Tabellen *CD* und *CDTrack* wird durch eine Verbindungslinie sogar die Relation zwischen den Tabellen dargestellt. Sie können die Tabellen nun beliebig anordnen und das gesamte Diagramm über das Zoomen-Feld der Symbolleiste vergrößern oder verkleinern, bis Sie alle vier Tabellen auf einmal im Blick haben.
5. Speichern Sie das Diagramm nun durch einen Klick auf das Diskettensymbol in der Symbolleiste und geben Sie als Namen *MediaBase_Gesamt* an. Das Diagramm erscheint nun auch im Objekt-Explorer, sodass Sie es von hier künftig wieder öffnen können.

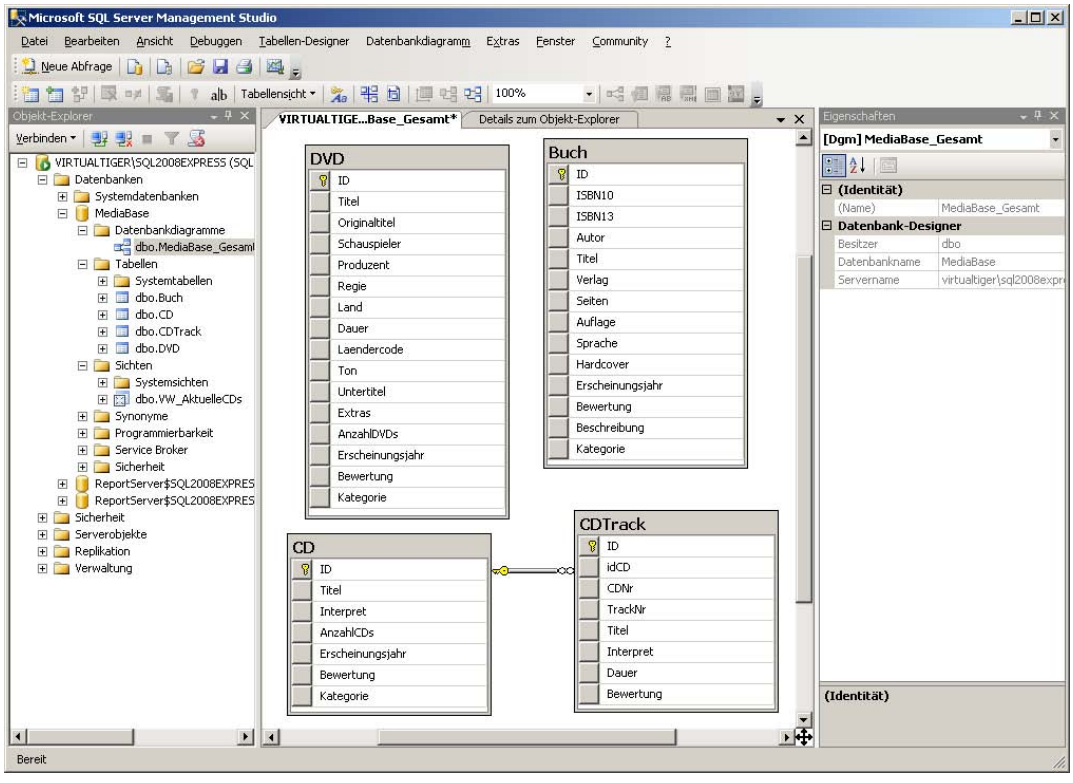


Abbildung 5.5: Das fertige Datenbankdiagramm zur Datenbank *MediaBase*

Vielleicht haben Sie bemerkt, dass während der Arbeit mit dem Diagramm im oberen Bereich des Management Studios eine neue Symbolleiste eingeblendet ist, die spezielle Funktionen zum Arbeiten mit Datenbankdiagrammen bietet (dieselben Optionen sind alternativ auch über das Menü *Datenbankdiagramm* erreichbar). So können Sie über das Symbol *a|b* eine Textanmerkung an beliebiger Stelle im Diagramm einfügen. Über *Tabellenansicht* können Sie die Darstellung zuvor ausgewählter Tabellen anpassen, sodass neben den Feldnamen auch die Datentypen angezeigt werden. Über die Option *Benutzerdefinierte Einstellungen ändern* lässt sich die Darstellung um recht viele zusätzliche Informationen erweitern (sofern Ihr Monitor groß genug ist). Diesem Problem können Sie aber begegnen, wenn Sie über den Menüpunkt *Ansicht/Ganzer Bildschirm* auf Vollbilddarstellung umschalten, wodurch auch der Objekt-Explorer ausgeblendet wird. Wenn Sie nun noch die Eigenschaften am rechten Bildrand ausblenden, haben Sie fast den ganzen Bildschirm für das Datenbankdiagramm zur Verfügung.

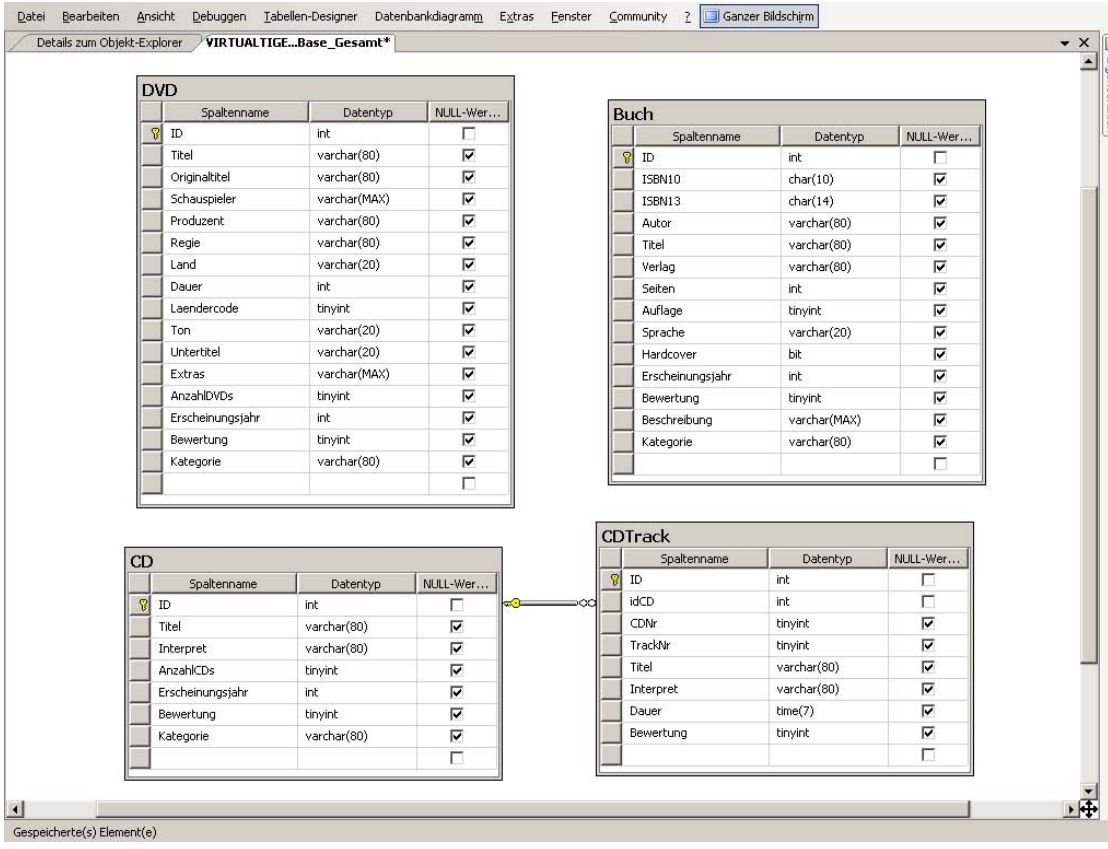


Abbildung 5.6: Das Datenbankdiagramm mit Spalten und Datentypen in Vollbilddarstellung

Auf diesem Weg können Sie Datenbankdiagramme nutzen, um sich einen Überblick zu verschaffen, oder das Datenmodell zu dokumentieren (die Diagramme lassen sich auch ausdrucken). Doch mit Datenbankdiagrammen können Sie noch mehr machen.

Ändern von Datenstrukturen mit Datenbankdiagrammen

Sie können sowohl Feldnamen als auch Datentypen direkt im Datenbankdiagramm ändern. Über einen Klick mit der rechten Maustaste auf die Tabelle im Diagramm oder mit dem Menü *Tabellen-Designer* können Sie Spalten löschen, neue Spalten hinzufügen, Spalteneigenschaften ändern sowie Beziehungen zwischen Tabellen und Indizes verwalten. All diese Änderungen werden erst dann in der Datenbank ausgeführt, wenn Sie das Diagramm speichern.

Es gibt aber noch eine weitere interessante Aktion, die Sie mit einem Datenbankdiagramm ausführen können. Wenn Sie eine Datenbankänderung noch nicht gespeichert haben und im Menü *Tabellen-Designer* den Punkt *Änderungsskript generieren* aufrufen (oder das entsprechende Symbol in der Symbolleiste anklicken), wird ein SQL-Skript generiert, das alle noch nicht durchgeführten Änderungen beinhaltet. Dieses können Sie als SQL-Datei speichern und bei Bedarf auch in anderen Datenbanken ausführen.

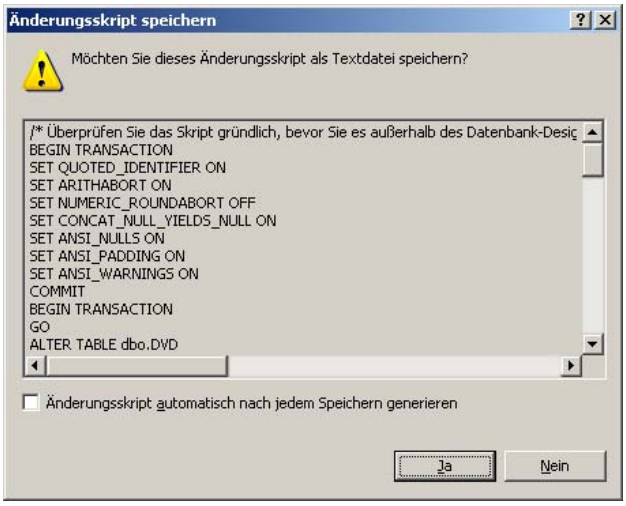


Abbildung 5.7: Das Dialogfeld mit dem generierten Änderungsskript

Best Practices: SQL-Skripts generieren, um Test- und Produktionsumgebungen anzupassen

Die Möglichkeit zum Generieren von Änderungsskripts können Sie nutzen, um Datenbankstrukturen von Test- und Produktionsumgebungen zu aktualisieren, wenn Sie die Änderungen erst auf einer separaten Entwicklungsumgebung ausführen.

Dabei bleiben die Daten der anderen Umgebungen erhalten, denn die generierten Skripts erstellen die Tabellen wenn nötig zwar neu, übertragen dabei aber alle Daten, die vorher in der Tabelle enthalten waren. (Genau genommen wird zuerst eine temporäre Kopie der Tabelle mit der neuen Struktur erstellt, dann die Daten aus der alten Tabelle in die neue Tabelle kopiert, die alte Tabelle gelöscht und schließlich die neue Tabelle umbenannt, sodass sie wieder den ursprünglichen Namen hat.) Sie können die Strukturen der verschiedenen Umgebungen also anpassen, aber trotzdem mit unterschiedlichen Datenständen arbeiten, um Testdaten von Produktionsdaten zu trennen.

5.4 Abfragen

Wenn Sie nun mit Daten aus mehreren Tabellen arbeiten möchten, so können Sie diese mit einer SQL-Anweisung vom Server abfragen. Dies wird allerdings erst im nächsten Kapitel behandelt. Wenn Sie sich aber nicht mit der SQL-Syntax herumschlagen möchten, können Sie auch mithilfe von SQL Server Management Studio und des darin enthaltenen Abfrage-Designers eine solche Abfrage interaktiv erstellen.

1. Wählen Sie in der Symbolleiste die Option *Neue Abfrage* aus und verbinden Sie sich mit der SQL Server-Instanz *SQL2008Express*.
2. Nun erscheint im Detailbereich der Abfrage-Editor, mit dem Sie SQL-Abfragen eingeben können. In der neu erschienenen Symbolleiste sehen Sie, dass die Verbindung momentan auf die Datenbank *master* besteht. Klappen Sie die Liste mit einem Klick auf und ändern Sie die Verbindung durch Auswahl der Datenbank *MediaBase* auf die gleichnamige Datenbank.

3. Da wir uns mit dem manuellen Erfassen von SQL-Anweisungen erst im nächsten Kapitel beschäftigen, wählen Sie im neu erschienenen Menü *Abfrage* den Punkt *Abfrage in Editor entwerfen* aus. Es erscheint der Abfrage-Designer, bei dem gleich ein Dialogfeld zum Hinzufügen von Tabellen geöffnet ist. Darin sollten die von Ihnen erstellten Tabellen *Buch*, *CD*, *CDTrack* und *DVD* zu sehen sein (falls nicht, sind Sie wahrscheinlich nicht mit der richtigen Datenbank verbunden).

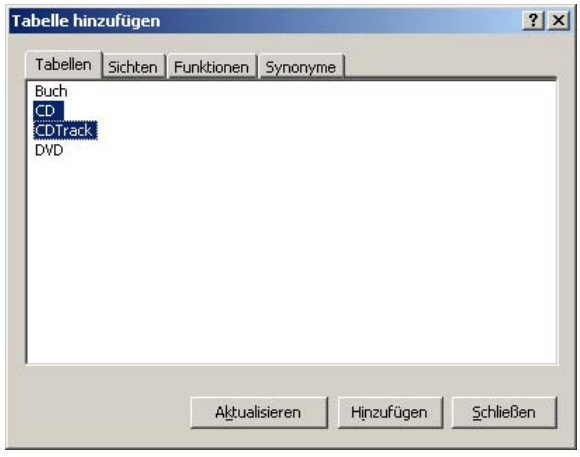


Abbildung 5.8: Das Dialogfeld zum Hinzufügen von Tabellen

4. Wählen Sie nun die Tabelle *CD* aus und klicken Sie auf die Schaltfläche *Hinzufügen*. Verfahren Sie genauso mit der Tabelle *CDTrack*. Alternativ können Sie auch erst beide Tabellen auswählen (siehe Tipp: *Mehrfachauswahl in Listen* weiter oben) und anschließend auf die Schaltfläche *Hinzufügen* klicken.
5. Schließen Sie das Dialogfeld über die Schaltfläche *Schließen*. Sie sehen nun den Abfrage-Designer, der in drei Bereiche aufgeteilt ist: Im oberen Bereich sind die beiden von Ihnen ausgewählten Tabellen mit ihrer Verknüpfung zu sehen. Im unteren Bereich steht die vom Designer automatisch generierte SQL-Abfrage. Dazwischen ist eine – momentan leere – Tabelle, in der alle von der Abfrage zurückzugebenden Spalten aufgelistet sind.
6. Vergrößern Sie die Ansicht der Tabellen im oberen Drittel so, dass alle Spalten zu sehen sind, und klicken Sie anschließend in der Liste der Tabelle *CD* das Kästchen vor dem Eintrag * *(Alle Spalten)* an und in der Liste der Tabelle *CDTrack* die Spalten *CDNr*, *TrackNr*, *Titel* und *Interpret*. Sie werden feststellen, dass auch die anderen beiden Bereiche des Abfrage-Designers automatisch um die ausgewählten Spalten erweitert werden.
7. Weil in der Tabelle *CDTrack* mit *Titel* und *Interpret* zwei Spalten ausgewählt wurden, deren Bezeichnungen auch in der Tabelle *CD* vorhanden sind, wurden hier automatisch Aliasnamen (*Expr1* und *Expr2*) vergeben, um die Felder voneinander unterscheiden zu können. Überschreiben Sie diese Aliasnamen in der Tabelle in der Mitte nun mit den aussagekräftigeren Bezeichnungen *TrackTitel* und *TrackInterpret*.
8. Damit die Abfrage auch sortierte Daten ausgibt, klicken Sie in das Feld *Sortiertyp* zur Spalte *TrackTitel*. In der nun über einen kleinen Pfeil aufklappbaren Liste wählen Sie die Option *Aufsteigend*, damit die *TrackTitel* in aufsteigender Reihenfolge sortiert werden.

9. Klicken Sie nun in das Feld *Filter* zur Spalte *CDNr* und geben Sie dort eine *1* ein. Damit werden von Mehrfach-CDs nur die Titel der ersten CD angezeigt. Der Abfrage-Designer sollte nun wie folgt aussehen:

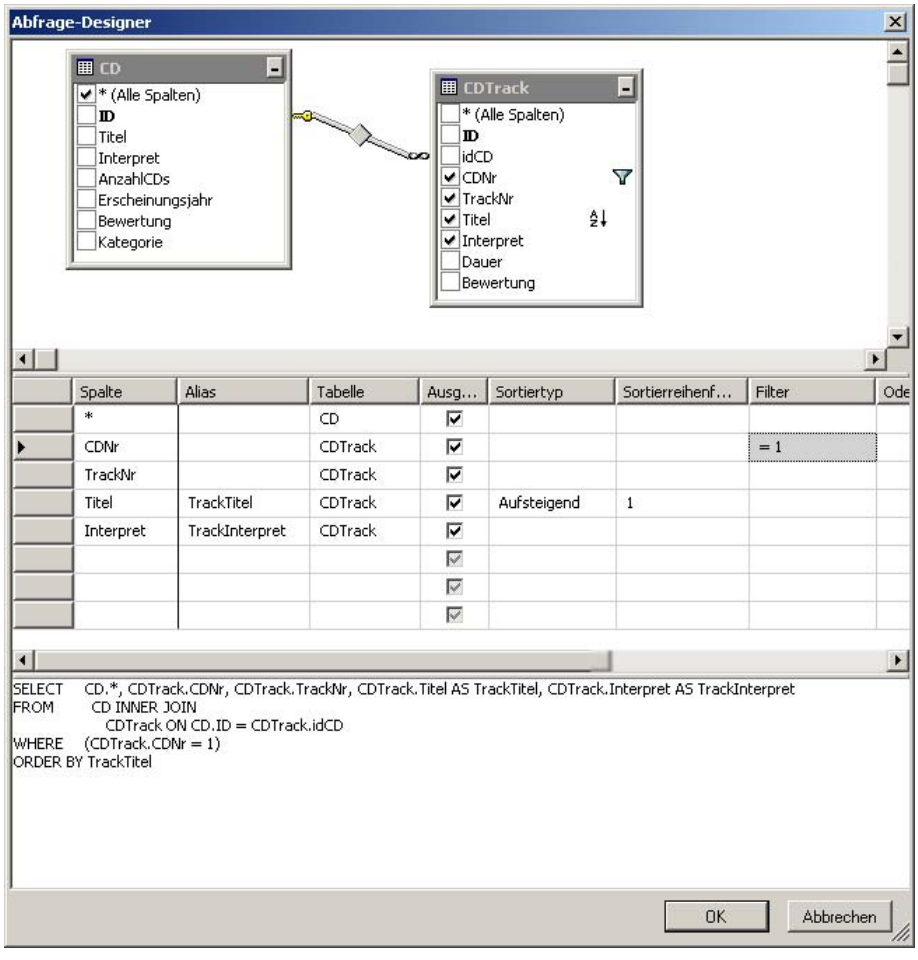


Abbildung 5.9: Der Abfrage-Designer

10. Schließen Sie den Designer mit der Schaltfläche *OK* und Sie sehen im Abfrage-Editor nur noch den SQL-Quelltext, den Sie durch einen Klick auf die entsprechend bezeichnete Schaltfläche ausführen können. Et voilà – Sie haben Ihre erste Abfrage entworfen, ohne auch nur ein Wort SQL zu schreiben.
11. Nach der Ausführung der Abfrage können Sie diese durch einen Klick auf das kleine Kreuz in der rechten oberen Ecke des Detailbereichs wieder schließen.

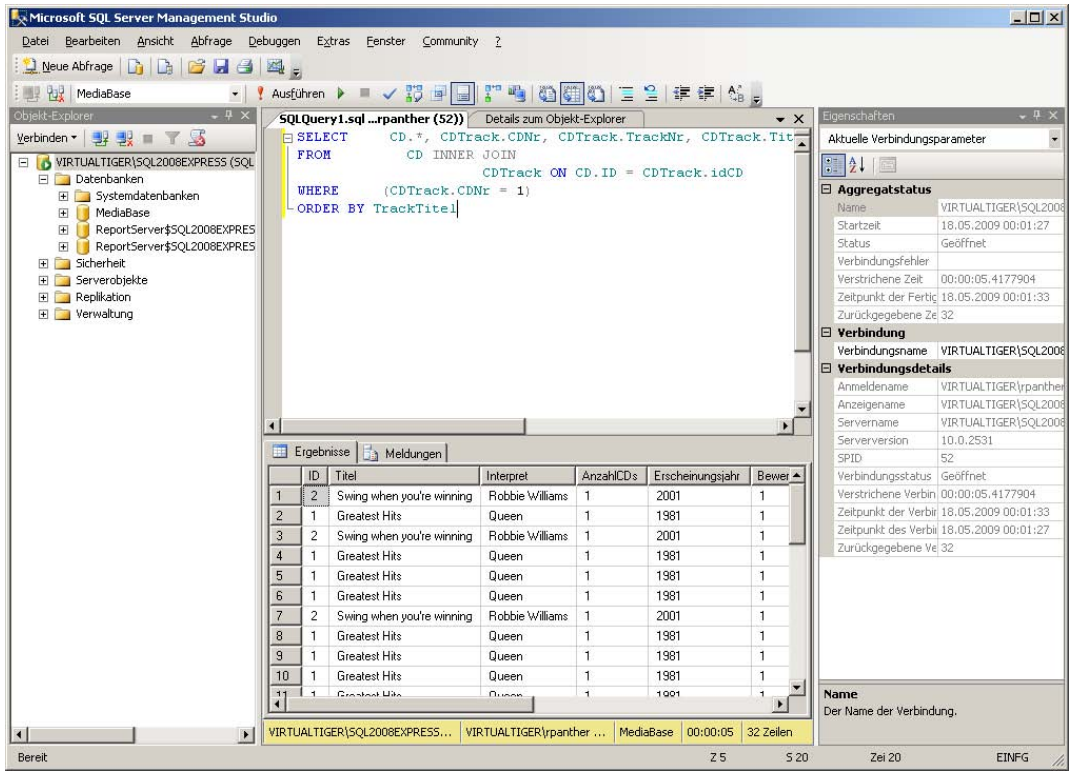


Abbildung 5.10: Der Abfrage-Editor mit der generierten Abfrage

5.5 Sichten (Views)

Nun drängt sich die Befürchtung auf, dass es auf Dauer sehr mühsam und umständlich werden kann, für jede Abfrage immer wieder dieselben Daten (z.B. eine CD mit dazugehörigen Tracks) aus den verschiedenen Tabellen zusammenzusuchen. SQL Server bietet aber ein passendes Mittel, um diesen Vorgang künftig weitgehend zu automatisieren: die Sichten (Views).

Mit Sichten lässt sich ein frei definierbarer Ausschnitt aus einer oder mehreren Tabellen definieren, die dann später auf einmal abgerufen werden. Dabei werden nicht die Daten selbst noch einmal zwischengespeichert, sondern lediglich die Definition, wie und woher die Sicht die Daten abrufen soll. Sichten können bei Bedarf sowohl die Auswahl der Zeilen anhand von Bedingungen einschränken als auch nur einen Teil der Spalten einer Tabelle zurückliefern.

Sichten auf eine Tabelle

Beginnen wir mit einer einfachen Sicht, die Daten aus einer einzigen Tabelle abruft:

1. Suchen Sie im Objekt-Explorer den Eintrag *Sichten* unterhalb der Datenbank *MediaBase* und klicken Sie diesen mit der rechten Maustaste an.
2. Wählen Sie im darauf erscheinenden Kontextmenü die Option *Neue Sicht* aus.
3. Es erscheint ein Dialogfeld, in dem Sie die Tabellen für die Sicht auswählen können. Wählen Sie hier die Tabelle CD aus und klicken Sie anschließend erst auf *Hinzufügen* und dann auf *Schließen*.
4. Die folgende Ansicht erinnert ein wenig an den Abfrage-Designer: Im oberen Bereich sehen Sie die beteiligten Tabellen mit ihren Spalten (und ein Kontrollkästchen vor jedem Spaltennamen). Darunter finden sich wieder die tabellarische Darstellung der in der Sicht verwendeten Spalten, sowie die der Abfrage zugrunde liegende SQL-Abfrage. Neu dagegen ist der Ergebnisbereich darunter, der das Ergebnis der Abfrage anzeigt, sobald Sie diese zum Testen ausführen.
5. Klicken Sie nun in der Diagrammdarstellung das Kästchen vor dem Sternchen an (das Sternchen symbolisiert alle Spalten der Tabelle). Klicken Sie anschließend ebenfalls die Kästchen vor den Feldern *Titel* und *Erscheinungsjahr* an, da Sie diese später für die Sortierung und einen Filter benötigen.
6. Sowohl das Sternchen als auch die Felder *Titel* und *Erscheinungsjahr* wurden damit in der Tabelle in der Mitte des Detailbereichs hinzugefügt und auch der generierte SQL-Befehl wurde aktualisiert. Entfernen Sie nun das Häkchen in der Spalte *Ausgabe* für die Felder *Titel* und *Erscheinungsjahr*, da diese ja schon durch die Angabe des Sternchens mit im Ergebnis ausgegeben werden.
7. Wählen Sie für die Spalte *Titel* den Sortiertyp *Aufsteigend* aus und geben Sie für die Spalte *Erscheinungsjahr* als Filter *>2000* an. Damit werden nur CDs ausgewählt, die nach dem Jahr 2000 erschienen sind und diese nach dem Titel sortiert. In der Diagrammansicht werden die beiden Spalten ebenfalls durch Symbole markiert, aus denen hervorgeht, dass die Spalte *Titel* für die Sortierung und die Spalte *Erscheinungsjahr* für einen Filter verwendet wird.
8. Führen Sie die Abfrage nun aus, indem Sie auf das kleine rote Ausrufezeichen in der Symbolleiste klicken. Je nachdem, welche Testdaten Sie in der Tabelle erfasst haben, werden nun im Ergebnisbereich die passenden Zeilen angezeigt.
9. Speichern Sie die Sicht, indem Sie das Diskettensymbol anklicken, und geben Sie der Sicht den Namen *VW_AktuelleCDs* (das Präfix *VW* kommt von View und wird verwendet, damit später schneller zu erkennen ist, dass es sich hierbei um eine Sicht und nicht um eine Tabelle handelt). Anschließend können Sie die Sicht schließen, indem Sie das kleine Kreuz in der rechten oberen Ecke des Detailbereichs anklicken.

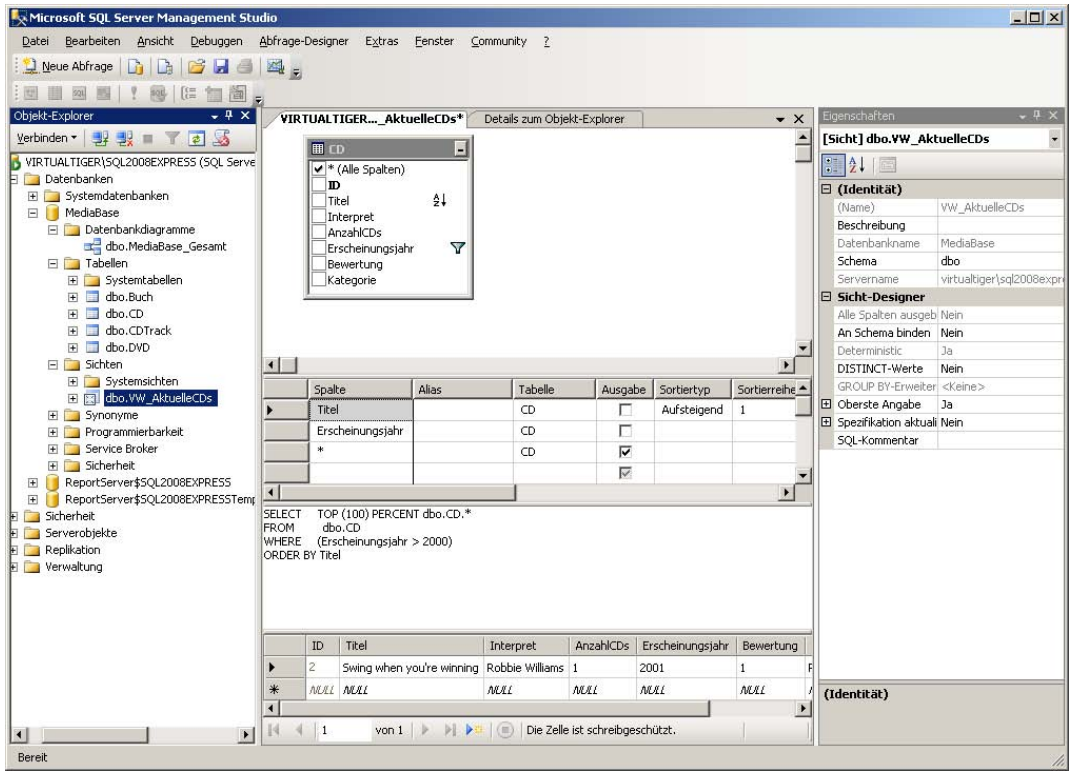


Abbildung 5.11: Der Entwurfsmodus für Sichten

Sichten, die mehrere Tabellen nutzen

Noch interessanter wird es, wenn Sichten definiert werden, die sich Daten aus mehreren Tabellen zusammensuchen. Bei unserem Beispiel bietet es sich an, eine Sicht zu erstellen, die alle CDs mit den dazugehörigen Tracks auswählt:

1. Öffnen Sie die gerade erstellte Sicht durch einen Klick mit der rechten Maustaste auf den Sichtnamen im Objekt-Explorer.
2. Nachdem die Sichtdefinition geöffnet wurde, sehen Sie, dass nun jedes einzelne Feld angekreuzt ist, das Sternchen allerdings nicht mehr. Dies wurde automatisch durchgeführt, ändert aber nichts am Ergebnis der Sicht.
3. Rufen Sie den Menüpunkt *Abfrage-Designer/Tabelle hinzufügen* auf (oder klicken Sie das entsprechende Symbol in der Symbolleiste an), um das Fenster mit den verfügbaren Tabellen wieder einzublenden.
4. Wählen Sie hier die Tabelle *CDTrack* aus und klicken Sie nacheinander auf die Schaltflächen *Hinzufügen* und *Schließen*. Die Tabelle *CDTrack* erscheint nun ebenfalls im Abfrage-Designer und wird automatisch mit der Tabelle *CD* verbunden.

5. Klicken Sie nun die Kontrollkästchen vor allen Spalten der Tabelle *CDTrack* bis auf die Spalten *ID* und *idCD* an, um diese ebenfalls im Ergebnis anzuzeigen.
6. Spalten mit den Namen *Titel*, *Interpret* und *Bewertung* existieren bereits in der Tabelle *CD*, daher wird diesen Spalten in der Tabelle *CDTrack* automatisch ein Alias zugewiesen, damit sie von den anderen Feldern unterscheidbar sind. Da diese Aliasse (*Expr1*, *Expr2* und *Expr3*) nicht besonders aussagekräftig sind, ersetzen Sie sie durch die Bezeichnungen *TrackTitel*, *TrackInterpret* und *TrackBewertung*. Nun können Sie die Abfrage ausführen und erhalten das entsprechende Ergebnis im unteren Bereich angezeigt.
7. Speichern und schließen Sie die überarbeitete Sicht nun.

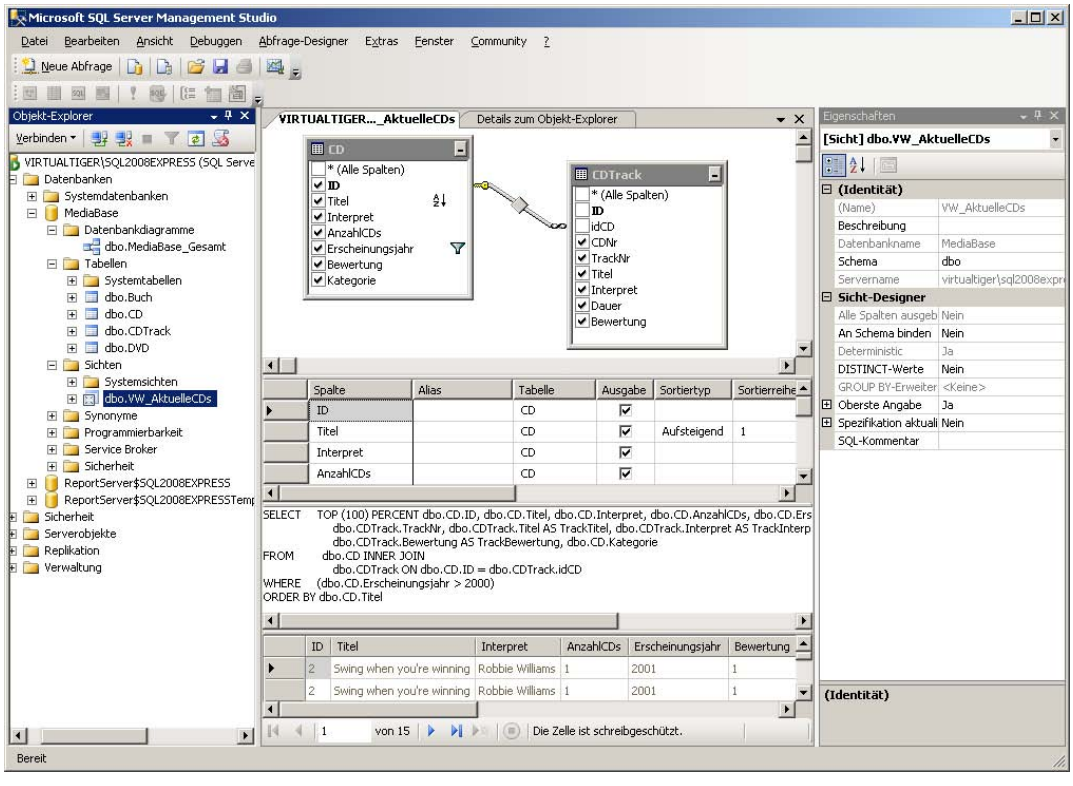


Abbildung 5.12: Die erweiterte Sicht mit Spalten aus zwei Tabellen

Um Daten für eine Sicht abzurufen, können Sie – genau wie bei Tabellen auch – deren Namen mit der rechten Maustaste anklicken und die Option *Oberste 1000 Zeilen auswählen* anklicken. Insgesamt lässt sich mit Sichten recht ähnlich arbeiten wie mit echten Tabellen.

Lediglich bei Sichten, die auf mehreren Tabellen basieren, gibt es einige Einschränkungen beim Ändern von Daten zu beachten. Das Hinzufügen und Ändern von Einträgen ist beispielsweise nur dann erlaubt, wenn sich die Änderungen auf lediglich eine der hinter der Sicht verborgenen Tabellen beziehen. Das funktioniert beim Ändern von Zeilen unter Umständen noch ganz gut, das Einfügen von Zeilen in Sichten wird – zumindest in Kombination mit einer Fremdschlüsseleinschränkung – fast unmöglich. Aus

demselben Grund lassen sich auch keine Zeilen in Sichten, die auf mehreren Tabellen basieren, entfernen. Stattdessen sollte man diese Operationen direkt in den zugrunde liegenden Basistabellen ausführen.

5.6 Übungen zu diesem Kapitel

In diesem Abschnitt finden Sie einige Übungen zu diesem Kapitel. Die richtigen Antworten und Lösungen finden Sie wie immer auf der Website *www.vsxpress.de.*

Übung 5.1

Wie können Sie verhindern, dass Datensätze mit ungültigen Fremdschlüsseln entstehen, sobald die Datensätze mit den entsprechenden Primärschlüsseln gelöscht werden?

Übung 5.2

Welcher Normalform entspricht die Tabelle *CD*, bevor deren Inhalte auf zwei Tabellen (*CD* und *CDTrack*) aufgeteilt wurden?

Übung 5.3

Erstellen Sie ein Datenbankdiagramm *MediaBase_CD*, das lediglich die Tabellen *CD* und *CDTrack* enthält.

Übung 5.4

Erstellen Sie eine Abfrage, die alle CD-Titel nach Interpreten und Erscheinungsjahr (Letzteres in absteigender Reihenfolge, sodass die neueren CDs zuerst erscheinen) sortiert zurückliefert.

Übung 5.5

Erstellen Sie eine Sicht *VW_Fachbuch*, die alle Bücher der Kategorie *Fachbuch* nach Autoren sortiert zurückliefert.

5.7 Zusammenfassung

In diesem Kapitel haben Sie gelernt, wie Sie mehrere Tabellen miteinander in Beziehung setzen können. Mithilfe von Fremdschlüsseln haben Sie Relationen definiert. Dazu haben Sie etwas Theorie über die Normalisierung von Datenbanken erfahren, welche die Grundlage für ein gutes Datenbankdesign bildet.

Datenbankdiagramme können Sie einerseits nutzen, um einen besseren Überblick über Datenstrukturen zu erhalten (oder diese damit zu dokumentieren). Andererseits lassen sich Datenbankdiagramme auch ändern, um damit ein Delta-Skript zu erzeugen, das die vorgenommenen Änderungen auch auf anderen Datenbanken (und Servern) durchführen kann.

Im Abschnitt über Abfragen haben Sie erfahren, wie sich SQL-Abfragen generieren lassen, ohne dass Sie SQL-Code selbst schreiben müssen. Nach einem ähnlichen Verfahren haben Sie anschließend Sichten, die auf mehreren Tabellen basieren, definiert, um diese komfortabel abzufragen. Mit einigen Einschränkungen sind sogar Datenänderungen über Sichten möglich.

Nachdem bis zu diesem Punkt alle Datenbankoperationen ausgeführt wurden, ohne dass Sie selbst eine einzige Zeile SQL schreiben mussten, werden im folgenden Kapitel einige SQL-Grundlagen vermittelt, mit denen Sie noch mehr Leistung und Flexibilität als über die grafische Benutzeroberfläche bekommen.

Kapitel 6

Kleine Einführung in SQL

In diesem Kapitel lernen Sie

- wozu man SQL benötigt
- wie Sie mit SQL Daten abfragen können
- wie Sie mit SQL Daten einfügen, ändern und löschen können
- wie Sie mit SQL Sichten definieren können
- was bei der Verwendung von Sichten zu beachten ist

6.1 Was ist eigentlich SQL?

SQL steht für Structured Query Language und ist aus der 1975 von IBM entwickelten Datenbank-Abfragesprache SEQUEL (Structured English Query Language) entstanden. Aus diesem Grund wird SQL auch heute oft noch wie das englische Wort »sequel« (Fortsetzung) ausgesprochen, was eigentlich auch ganz gut passt, denn zu dem SQL von damals gab es in den Jahren danach einige Fortsetzungen.

Seit 1986 wurde SQL durch das ANSI (American National Standards Institute) standardisiert. Dieser Standard wurde 1987 dann auch durch die ISO (International Organization for Standardization) aufgegriffen.

Tabelle 6.1: Die verschiedenen SQL-Standards im Überblick

Jahr	Produktname
1975	SEQUEL, der Vorgänger von SQL entsteht
1986	SQL1 wird als ANSI-Standard verabschiedet
1987	SQL1 wird als ISO-Standard verabschiedet
1992	SQL-92 / SQL2 wird als ISO-Standard verabschiedet
1999	SQL:1999 / SQL3 wird als ISO-Standard verabschiedet
2003	SQL:2003 wird als ISO-Standard verabschiedet
2006	SQL:2006 wird als ISO-Standard verabschiedet

Seit vielen Jahren ist SQL die Abfragesprache für relationale Datenbanken und wird von allen namhaften Herstellern unterstützt. Dabei verlief die Historie meist so, dass zuerst im SQL-Standard gewisse Vorgaben definiert wurden, die dann nach und nach von den verschiedenen Herstellern in den eigenen Produkten implementiert wurden.

Der Begriff Abfragesprache ist allerdings etwas irreführend, denn SQL geht weit über reine Abfragen hinaus. Denn SQL umfasst neben der klassischen SELECT-Anweisung (zum Auswählen von Daten) auch drei weitere Anweisungsgruppen:

- DML (Data Manipulation Language) zum Ändern von Daten, genauer Einfügen, Ändern und Löschen von Daten
- DDL (Data Definition Language) zum Verwalten (Anlegen, Ändern, Löschen) von Datenbankobjekten wie Tabellen, Sichten, Indizes, aber auch Datenbanken selbst
- DCL (Data Control Language) zum Verwalten von Benutzern, Rollen und Zugriffsrechten

Die SELECT-Anweisung selbst wird gelegentlich auch als DQL (Data Query Language) bezeichnet, diese Abkürzung ist jedoch bei Weitem nicht so geläufig wie die anderen drei.[1]

Der von Microsoft verwendete SQL-Dialekt heißt T-SQL (Transact-SQL) und basiert ebenfalls auf dem ISO- bzw. ANSI-Standard. Dazu kommen jedoch zahlreiche Erweiterungen.

Es gibt verschiedene Stellen, an denen SQL eingesetzt werden kann. So können SQL-Anweisungen in Form von Inline SQL bzw. Embedded SQL in Anwendungen eingebunden sein. Mit Dynamic SQL können auch SQL-Anweisungen zur Laufzeit zusammengesetzt und dann ausgeführt werden.

Als weitere Variante können Sie mithilfe des Kommandozeilentools SQLCMD SQL-Anweisungen, oder gar ganze SQL-Skripts, die in Dateien gespeichert sind, ausführen. In den meisten Fällen werden Sie zur Ausführung von SQL-Befehlen das SQL Server Management Studio verwenden, da dieses eine sehr komfortable Benutzeroberfläche für die Eingabe und Ausführung von SQL-Kommandos bietet.

6.2 SQL-Anweisungen im Management Studio ausführen

Bevor ich mit der Beschreibung der wichtigsten SQL-Anweisungen beginne, möchte ich noch kurz erläutern, wo man diese am besten ausprobieren kann. Bereits seit der 2005er-Generation bietet Microsoft mit dem SQL Server Management Studio eine einheitliche Oberfläche, mit der man sowohl Datenbanken verwalten als auch SQL-Anweisungen – oder gar komplette Skripts – komfortabel erfassen und ausführen kann. Wenn Sie bereits mit einer SQL Server-Instanz verbunden sind, brauchen Sie dazu nur auf die Schaltfläche *Neue Abfrage* in der linken oberen Ecke zu klicken und schon erscheint im Detailbereich ein Abfragefenster. Dazu gibt es eine entsprechende Symbolleiste, die Ihnen die wichtigsten Möglichkeiten für das Arbeiten mit Abfragen liefert.

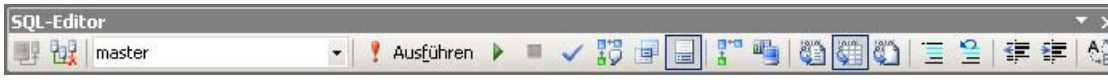


Abbildung 6.1: Die Symbolleiste zum Abfrage-Editor

[1] Sowohl die Befehle der DDL als auch die der DCL werden nicht in diesem Kapitel, sondern weiter hinten im Buch behandelt. Die Anweisungen der DDL werden in Kapitel 10 *Datenbankadministration mit SQL* beschrieben, die DCL wird in Kapitel 11 *Benutzer, Rollen und Rechte* berücksichtigt.

Mit den beiden Symbolen links kann man eine Verbindung zu einem SQL Server herstellen (was im Normalfall schon geschehen ist, wenn Sie sich im Abfrage-Editor befinden, daher ist das erste Symbol in Abbildung 6.1 inaktiv) bzw. die bestehende Verbindung ändern. Letzteres macht beispielsweise dann Sinn, wenn Sie eine Abfrage nacheinander auf mehreren Servern ausführen möchten.

Rechts daneben ist eine Auswahlliste für die Datenbank, in der die Abfrage ausgeführt werden soll. Wenn Sie nicht vorher – beispielsweise über den Objekt-Explorer – eine andere Datenbank ausgewählt haben, ist hier die *master*-Datenbank voreingestellt (bzw. die Datenbank, die für den verwendeten Login als Standarddatenbank angegeben wurde). Wenn Sie die Liste aufklappen, werden alle Datenbanken angezeigt, auf die Sie zugreifen können. Alternativ können Sie aber auch direkt im Skript die Datenbank wechseln, indem Sie die `use`-Anweisung nutzen.

Beispiel: `use MediaBase`

Dank IntelliSense bekommen Sie nach Eingabe des Schlüsselworts `use` gefolgt vom ersten Zeichen des Datenbanknamens unmittelbar die möglichen Datenbanken zur Auswahl angezeigt, die mit dem bereits eingegebenen Zeichen beginnen.

Doch kommen wir zurück zur Symbolleiste: Rechts neben der Datenbankauswahl befindet sich ein rotes Ausrufezeichen gefolgt vom Text *Ausführen*. Wie die Bezeichnung schon erahnen lässt, können Sie mit diesem Symbol die eingegebenen SQL-Anweisungen ausführen, was im unteren Bereich des Fensters durch eine entsprechende Meldung quittiert wird.

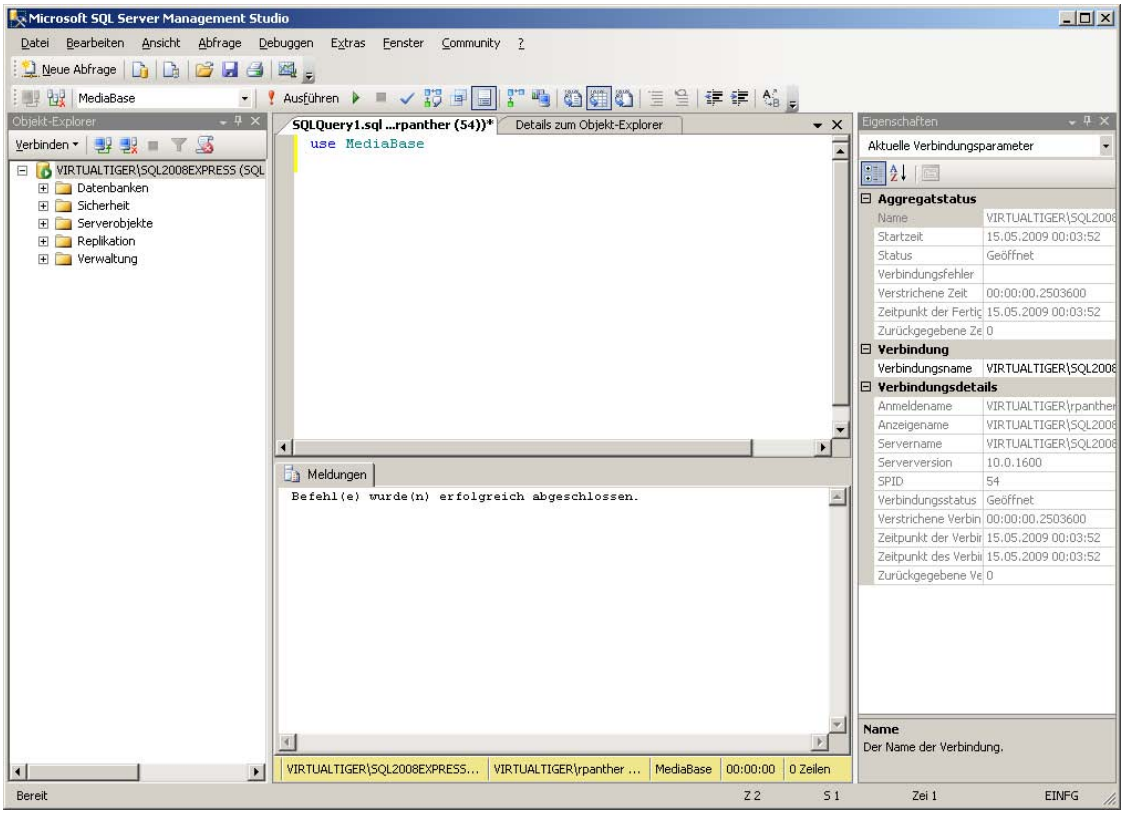


Abbildung 6.2: Abfrage-Editor mit eingeblendeter Ergebnismeldung

Tipp: Teile eines SQL-Skripts ausführen

Wenn im Abfragefenster mehrere SQL-Anweisungen eingegeben wurden, werden die sie ausgeführt, sofern Sie die entsprechende Schaltfläche in der Symbolleiste anklicken. Um nur einen Teil der Anweisungen auszuführen, markieren Sie diese mit der Maus (bzw. den Cursortasten). Wenn Sie nun auf die *Ausführen*-Schaltfläche klicken, werden lediglich die markierten Anweisungen ausgeführt.

Mit dem kleinen grünen Pfeil direkt neben der *Ausführen*-Schaltfläche wird die Abfrage zwar ebenfalls ausgeführt, allerdings im Debug-Modus. Was es damit genau auf sich hat, wird in Abschnitt 8.4, *Debuggen von SQL-Skripts* im Detail erläutert.

Rechts neben dem grünen Pfeil ist ein kleines Viereck zu sehen, das während der Ausführung einer Abfrage rot dargestellt wird. Mit diesem Symbol können Sie die Ausführung einer Abfrage unterbrechen. Das blaue Häkchen rechts daneben dient zum Analysieren einer Abfrage. So können Sie einfach testen, ob eine Abfrage syntaktische Fehler beinhaltet, ohne die Abfrage ausführen zu müssen.

Dies soll im Moment ausreichen, damit wir endlich mit den ersten Abfragen loslegen können. Die weiteren Symbole werden weiter hinten im Buch ausführlich erläutert.

Abschließend sollte aber noch die Statusleiste unterhalb des Ergebnis- und Meldungsbereichs erwähnt werden, in der neben den Verbindungsdaten (Datenbankinstanz und Login) und der Datenbank auch die Dauer und Zeilenanzahl der letzten Abfrageausführung zu sehen. Aber nun weiter mit SQL.

6.3 Datenbankabfragen mit SELECT

Die sicherlich bekannteste, aber auch wichtigste SQL-Anweisung ist das `SELECT`-Statement. Mit dieser Anweisung lassen sich Daten aus einer oder mehreren Tabellen abfragen. Angefangen von sehr einfachen einzeiligen Anweisungen bis hin zu komplexen mehrseitigen Abfragen mit verschachtelten Unterabfragen, Joins und Gruppierungen bildet der `SELECT`-Befehl das Herzstück der Sprache SQL.

Dabei setzen sich SQL-Anweisungen aus mehreren Bestandteilen (sogenannten Klauseln) zusammen, die jeweils durch ein Schlüsselwort eingeleitet werden. Die meisten dieser Klauseln sind optional, sodass wir mit einer einfachen Variante beginnen, bevor wir uns den komplizierteren `SELECT`-Abfragen zuwenden.

Abfragen auf einer Tabelle

Die einfachste Form der `SELECT`-Anweisung ist die folgende: `SELECT * FROM xyz`

Dies bedeutet, dass alle Zeilen und Spalten der Tabelle `xyz` abgefragt werden sollen. Dabei ist das `xyz` natürlich durch einen existierenden Tabellennamen zu ersetzen. Das Sternchen ist ein Platzhalter, der alle Felder der verwendeten Tabelle(n) repräsentiert. Würde man diesen Platzhalter nicht verwenden, so müsste man die gewünschten Spaltennamen (durch Kommata getrennt) aufzählen.

Beispiel: `SELECT Autor, Titel FROM Buch`

Tipp: Drag & Drop spart Tipparbeit

Wenn Sie sich das Tippen von Tabellennamen, Feldnamen etc. sparen wollen, können Sie die entsprechenden Objekte auch im Objekt-Explorer auswählen und per Drag & Drop ins Abfragefenster ziehen.

Mit dem Zusatz `DISTINCT` lässt sich das Ergebnis einer Abfrage so einschränken, dass doppelt auftretende Zeilen nur noch einmal erscheinen. Dazu ist das entsprechende Schlüsselwort direkt hinter `SELECT` anzugeben.

Beispiel: `SELECT DISTINCT Autor FROM Buch`

Wenn die angezeigten Ergebniszeilen sortiert werden sollen, verwendet man die sogenannte `ORDER BY`-Klausel. Diese steht am Ende einer `SELECT`-Anweisung, gefolgt von den Namen der Spalten (durch Komma getrennt), nach denen sortiert werden soll.

Beispiel: `SELECT * FROM Buch ORDER BY Autor, Titel`

Dabei wird standardmäßig in aufsteigender Reihenfolge sortiert, was man auch explizit durch den Zusatz `asc` (als Kurzform für ascending = aufsteigend) dokumentieren kann. Will man stattdessen in absteigender Reihenfolge sortieren, so ist der Zusatz `desc` (für descending = absteigend) zu nutzen.

Beispiel: `SELECT * FROM Buch ORDER BY Autor ASC, Erscheinungsjahr DESC`

Best Practices: SQL-Codeformatierung

Bei SQL-Anweisungen spielt die Groß-/Kleinschreibung eigentlich keine Rolle. Dennoch ist es weit verbreitete Praxis, die SQL-Schlüsselwörter komplett großzuschreiben, um diese optisch besser von Objektnamen (Tabellennamen, Feldnamen etc.) abzuheben.

Sobald die SQL-Anweisungen etwas komplexer werden, bietet es sich an, diese auf mehrere Zeilen zu verteilen. Dies wird dann üblicherweise so umbrochen, dass jede Zeile mit einer neuen Klausel beginnt. Für die oben stehende Abfrage würde das dann wie folgt aussehen:

```
SELECT *
FROM Buch
ORDER BY Autor ASC, Erscheinungsjahr DESC
```

Zum Eingrenzen der Ergebnismenge kann in der `WHERE`-Klausel eine (oder mehrere) Filterbedingung angegeben werden. Dabei nutzt man meist ein oder mehrere Spaltennamen in Kombination mit den üblichen Vergleichsoperatoren.

Beispiel: `SELECT * FROM CD WHERE Interpret = 'Queen'`

Mit den Verknüpfungsoperatoren `AND` und `OR` lassen sich mehrere Bedingungen kombiniert einsetzen:

Beispiele:

```
SELECT *
FROM CD
WHERE Interpret = 'Queen' AND Erscheinungsjahr > 1990

SELECT *
FROM CD
WHERE Interpret = 'Queen' OR Interpret = 'Police'
```

Geht es aber – wie im letzten Beispiel – darum, verschiedene Möglichkeiten für ein Feld abzufragen, kann man auch den Operator `IN` verwenden und dazu in Klammern eine kommagetrennte Liste von möglichen Werten angeben. Damit würde folgende Abfrage dasselbe Ergebnis liefern:

```
SELECT * FROM CD WHERE Interpret IN ('Queen', 'Police')
```

Für den Vergleich von Zeichenketten gibt es aber noch einen weiteren nützlichen Operator. Mit dem Schlüsselwort `LIKE` können Sie die Platzhalter _ (für genau ein beliebiges Zeichen) und `%` (für beliebig viele beliebige Zeichen) verwenden.

Bei einer Abfrage einer Personentabelle würde damit die Bedingung `WHERE Nachname LIKE 'M_ller'` sowohl die Nachnamen *Müller* als auch *Möller* oder *Miller* erfassen. Wenn Sie auch Namen wie *Moeller* und *Mueller* abdecken wollen, müssten Sie schon folgende Form nutzen: `WHERE Nachname LIKE 'M%ller'`, wodurch aber auch der Name *Meisenstaller* mit erfasst würde.

Als weitere Möglichkeit können Sie beim `LIKE`-Operator auch verschiedene Zeichen in eckigen Klammern angeben, von denen eines vorkommen muss. Mit der folgenden Abfrage würden beispielsweise die Namen *Müller* und *Möller* gefunden, nicht aber *Miller* oder Ähnliche:

```
SELECT * FROM Adressen WHERE Nachname LIKE 'M[üö]ller'
```

Insbesondere bei sehr großen Tabellen macht es Sinn, die Ergebnismenge einzuschränken. Hierfür gibt es bei der `SELECT`-Anweisung den optionalen Zusatz `TOP`, der auf zweierlei Art verwendbar ist. Entweder gibt man die Anzahl der Zeilen an oder aber einen prozentuellen Wert gefolgt vom Schlüsselwort `PERCENT`, der definiert, welcher Teil des Gesamtergebnisses zurückgegeben werden soll.

Um beispielsweise die ersten zehn Prozent aller CD-Tracks zu erhalten, könnte man folgende Abfrage verwenden:

```
SELECT TOP 10 PERCENT *
FROM CDTrack
```

Oft verwendet man den `TOP`-Zusatz auch in Verbindung mit der `ORDER BY`-Klausel, um genau steuern zu können, welche Zeilen man im Ergebnis erhält. Die folgende Anweisung liefert beispielsweise die drei aktuellsten CDs zurück:

```
SELECT TOP 3 *
FROM CD
ORDER BY Erscheinungsjahr DESC
```

Anstatt die Beschränkung für jede SQL-Anweisung anzugeben, lässt sich die Eingrenzung auf eine bestimmte Zeilenzahl mit der `ROWCOUNT`-Option auch generell für alle Abfragen einer Session (SQL Server-Verbindung) festlegen. Nutzen Sie die folgende Anweisung, um für alle folgenden Abfragen maximal fünf Zeilen zurückzugeben:

```
SET ROWCOUNT 5
```

Um künftig wieder alle Ergebniszeilen einer Abfrage zu erhalten, können Sie dies mit der Anweisung `SET ROWCOUNT 0` wieder aufheben.

Aggregierungsfunktionen und Gruppierungen

Für einfache Berechnungen über mehrere Zeilen stellt SQL Server ein paar Funktionen zur Verfügung. Die gängigsten davon sind die folgenden:

- `COUNT` gibt die Anzahl der Zeilen zurück
- `AVG` gibt den Durchschnittswert des Feldes zurück
- `MAX` gibt den größten in der Tabelle vorkommenden Wert des Feldes zurück
- `MIN` gibt den kleinsten in der Tabelle vorkommenden Wert des Feldes zurück
- `SUM` liefert die Summe der Werte des Feldes in der Tabelle

Dabei wird für die meisten Funktionen ein Feldname in Klammern als Parameter angegeben. Lediglich die `COUNT`-Funktion wird meist in der Form `COUNT(*)` genutzt, da es hier eigentlich nur um die Anzahl der Zeilen geht. Geben Sie hier doch einen Spaltennamen an, so werden nur diejenigen Zeilen gezählt, bei denen das Feld einen Wert ungleich `NULL` hat. Die beiden folgenden Abfragen liefern damit dasselbe Ergebnis:

```
SELECT COUNT(*) FROM Buch WHERE Auflage IS NOT NULL
SELECT COUNT(Auflage) FROM Buch
```

Das Ergebnis einer so berechneten Spalte wird mit dem Titel *(kein Spaltenname)* angezeigt. Dies kann man beheben, indem man in der Abfrage eine explizite Bezeichnung für die berechnete Spalte vergibt.

```
SELECT COUNT(*) AS AnzahlBuecher
FROM Buch
```

Bei den bisher beschriebenen Fällen wird die Berechnung immer über alle Zeilen der Tabelle ausgeführt. In der Praxis wird aber normalerweise eher ein Wert für eine Gruppe von Zeilen benötigt. Um dies zu erreichen, lassen sich die Ergebnisse mit der `GROUP BY`-Klausel anhand der darin angegebenen Spalten gruppieren. Die folgende Abfrage liefert beispielsweise die Anzahl Bücher pro Autor:

```
SELECT COUNT(*) AS Anzahl, Autor
FROM Buch
GROUP BY Autor
```

Dabei ist zu beachten, dass in der Feldauswahl für das Ergebnis – abgesehen von den Aggregierungsfunktionen – nur Spalten angegeben werden dürfen, die auch in der `GROUP BY`-Klausel stehen.

Mit einer zusätzlichen `HAVING`-Klausel lassen sich die Ergebniszeilen weiter filtern. Der Unterschied zu einer `WHERE`-Klausel liegt darin, dass sich eine `WHERE`-Klausel immer auf einzelne Zeilen bezieht, während `HAVING` die Ergebnisse einer Gruppierung filtert. Damit lässt sich die Abfrage von oben über eine `HAVING`-Klausel auf Autoren, von denen es mehr als ein Buch in der Tabelle gibt, eingrenzen:

```
SELECT COUNT(*) AS Anzahl, Autor
FROM Buch
GROUP BY Autor
HAVING COUNT(*)>1
```

Abfragen auf mehreren Tabellen

Die bisher behandelten Abfragen wurden nur auf jeweils einer einzelnen Tabelle ausgeführt. Interessanter wird es aber, wenn mehrere Tabellen in einer Abfrage gemeinsam genutzt werden.

Spalten aus mehreren Tabellen mit JOIN verknüpfen

Die gängigste Variante hierfür ist eine Verknüpfung per `INNER JOIN`. Hierbei werden die beiden Tabellennamen in der `FROM`-Klausel durch die Schlüsselwörter `INNER JOIN` verbunden, gefolgt vom Schlüsselwort `ON`, nach dem die Felder angegeben sind, mit denen die Tabellen verknüpft werden.

```
SELECT *
FROM CD INNER JOIN CDTrack ON CD.ID = CDTrack.idCD
```

Der Stern repräsentiert in diesem Fall alle Felder aller beteiligten Tabellen. Soll dies auf alle Felder lediglich einer Tabelle oder gar einzelne Felder eingegrenzt werden, so kann man hier das Sternchen oder explizite Feldnamen in Kombination mit dem Tabellennamen aufzählen. Die folgende Abfrage listet beispielsweise alle Felder der Tabelle *CD* auf und lediglich die Felder *TrackNr* und *Titel* aus der Tabelle *CDTrack*:

```
SELECT CD.*, CDTrack.TrackNr, CDTrack.Titel
FROM CD INNER JOIN CDTrack ON CD.ID = CDTrack.idCD
```

Wenn Sie nun zahlreiche Felder explizit angeben, so ist es mitunter recht unübersichtlich, jedes Mal den kompletten Tabellennamen mit anzugeben (insbesondere dann, wenn dieser deutlich länger ist als im obigen Beispiel). Um diesem Umstand abzuhelfen, können Sie innerhalb der Abfrage Aliasnamen für die einzelnen Tabellen definieren. In der Praxis bewährt haben sich hier Kürzel mit zwei bis drei Buchstaben. Damit würde die folgende Abfrage dasselbe Ergebnis liefern wie das vorige Beispiel:

```
SELECT CD.*, CDT.TrackNr, CDT.Titel
FROM CD INNER JOIN CDTrack as CDT ON CD.ID = CDT.idCD
```

Für die Tabelle *CD* wurde hier natürlich kein Alias definiert, weil dieser Tabellenname ja ohnehin schon kurz genug ist.

Neben dem gerade verwendeten `INNER JOIN`, bei dem passende Daten in beiden Tabellen erforderlich sind, gibt es noch weitere `JOIN`-Arten (dabei sind die in Klammern angegebenen Schlüsselwörter optional):

- `(INNER) JOIN` – alle Datensätze der ersten Tabelle, die entsprechende Datensätze in der zweiten Tabelle haben (und umgekehrt)
- `LEFT (OUTER) JOIN` – alle Datensätze der ersten Tabelle, mit entsprechenden Daten aus der zweiten Tabelle (sofern vorhanden)
- `RIGHT (OUTER) JOIN` – alle Datensätze der zweiten Tabelle, mit entsprechenden Daten aus der ersten Tabelle (sofern vorhanden)
- `FULL (OUTER) JOIN` – Kombination aus `LEFT JOIN` und `RIGHT JOIN`: alle Datensätze beider Tabellen, mit entsprechenden Daten der jeweils anderen Tabelle (sofern vorhanden)
- `CROSS JOIN` – jeder Datensatz der ersten Tabelle wird mit jedem Datensatz der zweiten Tabelle verknüpft

In der Praxis werden FULL OUTER JOIN und insbesondere CROSS JOIN aber eher selten verwendet. Ich persönlich vermeide nach Möglichkeit auch RIGHT OUTER JOINs, indem ich diese zu LEFT OUTER JOINs umforme, was komplexe Abfragen besser lesbar macht, als wenn LEFT JOIN und RIGHT JOIN gemischt verwendet werden.

Wenn man in dem obigen Beispiel den INNER JOIN durch einen LEFT JOIN ersetzt, werden auch CDs mit ausgegeben, zu denen keine Tracks erfasst sind:

```
SELECT CD.*, CDT.TrackNr, CDT.Titel
FROM CD LEFT JOIN CDTrack as CDT ON CD.ID = CDT.idCD
```

Die Felder *TrackNr* und *Titel* werden für solche CDs dann als *NULL*-Werte ausgegeben.

Zeilen aus mehreren Tabellen mit UNION verknüpfen

Neben der Möglichkeit, Tabellen über JOIN-Bedingungen zu verknüpfen, gibt es noch eine weitere grundlegende Variante, Daten aus verschiedenen Tabellen mithilfe einer Abfrage zu verbinden. Der UNION SELECT wird verwendet, um Abfragen, welche Ergebnisse mit derselben Struktur zurückliefern, miteinander zu verbinden.

Für unsere Beispieltabellen wären beispielsweise folgende Abfragen verwendbar:

```
SELECT Titel, Interpret, Erscheinungsjahr FROM CD
SELECT Titel, Autor, Erscheinungsjahr FROM Buch
SELECT Titel, Produzent, Erscheinungsjahr FROM DVD
```

Alle drei liefern drei Spalten zurück, von denen die ersten beiden alphanumerisch sind und die dritte vom Typ integer ist. Verbinden Sie die Abfragen nun mit dem Schlüsselwort UNION, werden die Ergebnisse in einer Abfrage zusammengefasst:

```
SELECT Titel, Interpret, Erscheinungsjahr FROM CD
UNION SELECT Titel, Autor, Erscheinungsjahr FROM Buch
UNION SELECT Titel, Produzent, Erscheinungsjahr FROM DVD
```

Dabei werden die Feldnamen der ersten Tabelle (bzw. Teilabfrage) übernommen. Das zweite Feld heißt also *Interpret*, auch wenn es sowohl Autoren als auch Produzentennamen beinhaltet. Zur besseren Unterscheidung der Datenquelle kann man jeder Teilabfrage ein Typ-Feld mit einem konstanten Wert hinzufügen:

```
SELECT Titel, Interpret AS InterpretAutorProduzent, Erscheinungsjahr, 'CD' as Quelle FROM CD
UNION SELECT Titel, Autor, Erscheinungsjahr, 'Buch' FROM Buch
UNION SELECT Titel, Produzent, Erscheinungsjahr, 'DVD' FROM DVD
```

Eine Besonderheit der UNION-Anweisung liegt darin, dass die Daten implizit sortiert und doppelte Ergebniszeilen ausgefiltert werden (also ein impliziter DISTINCT auf die gesamte Ergebnismenge angewendet wird). Wenn dies nicht gewünscht ist, lässt sich das durch den Zusatz ALL verhindern, wodurch die Zeilen aller drei Teilabfragen auch in der Reihenfolge der Teilabfragen zurückgegeben werden.

```
SELECT Titel, Interpret AS InterpretAutorProduzent, Erscheinungsjahr, 'CD' as Quelle FROM CD
UNION ALL SELECT Titel, Autor, Erscheinungsjahr, 'Buch' FROM Buch
UNION ALL SELECT Titel, Produzent, Erscheinungsjahr, 'DVD' FROM DVD
```

Das reicht vorerst einmal zum Thema Abfragen von Daten. Kommen wir nun zu den wichtigsten Anweisungen, um Daten zu bearbeiten.

6.4 Daten bearbeiten mit UPDATE, INSERT und DELETE

Die Anweisungen `UPDATE`, `INSERT` und `DELETE` bilden die wesentlichen Bestandteile der sogenannten Data Manipulation Language (DML) und werden oft auch als CRUD-Anweisungen bezeichnet (CReate, Update, Delete).

INSERT und SELECT INTO zum Einfügen von Daten

Die `INSERT`-Anweisung gibt es in zwei Grundformen. Mit der ersten Variante fügen Sie eine einzelne Zeile ein und geben dabei die Inhalte der einzelnen Felder an.

```
INSERT INTO [Tabelle] ([Felder])
VALUES ([Daten])
```

Um der Tabelle *Buch* einen neuen Eintrag hinzuzufügen, können Sie eine Abfrage der folgenden Form verwenden:

```
INSERT INTO Buch (Autor, Titel, Hardcover)
VALUES ('Robert Panther', 'Programmieren mit dem .NET Compact Framework', 1)
```

Dabei müssen nicht alle Spalten der Tabelle angegeben werden, lediglich die als *Nicht NULL* deklarierten Felder müssen auf jeden Fall aufgeführt werden (sofern dafür kein Standardwert definiert ist).

Mit der zweiten Variante der `INSERT`-Anweisung können mehrere Zeilen auf einmal eingefügt werden, indem man eine entsprechende `SELECT`-Abfrage angibt. Dabei können die Daten auch aus einer komplett anderen Tabelle stammen, sofern die Datentypen kompatibel sind. Wenn Sie beispielsweise zu allen erfassten DVDs die entsprechende Soundtrack-CD gekauft haben, können Sie diese mit folgender Anweisung einfach erfassen:

```
INSERT INTO CD (Titel)
SELECT Titel FROM DVD
```

Alternativ zur `INSERT`-Anweisung gibt es auch einen anderen SQL-Befehl, mit dem Sie Daten einfügen und dabei sogar gleichzeitig die Zieltabelle anlegen können. Über die `INTO`-Klausel können Sie eine beliebige `SELECT`-Abfrage so erweitern, dass das Ergebnis nicht ausgegeben, sondern in eine neue Tabelle geschrieben wird.

```
SELECT [Felder]
INTO [Zieltabelle]
FROM [Quelltabelle]
WHERE [Bedingung]
```

Mit der folgenden Anweisung können Sie einen Teil der Tabelle *Buch* in eine neue Tabelle mit Namen *MeineBuecher* kopieren:

```
SELECT *
INTO MeineBuecher
FROM Buch
WHERE Autor = 'Robert Panther'
```

Genau wie bei der INSERT INTO-Anweisung kann auch beim SELECT … INTO ein beliebiger SELECT verwendet werden, der beispielsweise auch auf mehreren Tabellen basieren darf. Da hier aber auf Basis der Abfrage eine neue Tabelle erstellt wird, müssen die Bezeichnungen der Spalten eindeutig sein.

Die folgende Anweisung würde also nicht funktionieren, da sowohl aus der Tabelle *CD* als auch aus der Tabelle *CDTrack* eine Spalte mit Namen *Titel* zurückgegeben wird:

```
SELECT CD.*, CDT.TrackNr, CDT.Titel
INTO CDMitTracks
FROM CD LEFT JOIN CDTrack as CDT ON CD.ID = CDT.idCD
```

Wenn Sie die Abfrage aber wie folgt anpassen, funktioniert es, da die Spalte *Titel* aus der Tabelle *CDTrack* nun den neuen Spaltennamen *TrackTitel* bekommen hat:

```
SELECT CD.*, CDT.TrackNr, CDT.Titel as TrackTitel
INTO CDMitTracks
FROM CD LEFT JOIN CDTrack as CDT ON CD.ID = CDT.idCD
```


UPDATE zum Ändern von Daten

Für Datenänderungen stellt die Sprache SQL die UPDATE-Anweisung zur Verfügung. Dabei wird zuerst angegeben, welche Tabelle geändert werden soll. In einer SET-Klausel erfolgen dann die eigentlichen Änderungen. Am Ende folgt noch eine WHERE-Klausel, in der eingeschränkt wird, für welche Zeilen der Tabelle die Änderungen durchgeführt werden.

```
UPDATE [Tabelle]
SET [Zuweisung]
WHERE [Bedingung]
```

Die folgende Anweisung setzt das Feld *Verlag* für alle Zeilen, in denen das Feld nicht belegt war (also den Wert *NULL* enthält), auf die konstante Zeichenkette *(unbekannt)*:

```
UPDATE MeineBuecher
SET Verlag = '(unbekannt)'
WHERE Verlag IS NULL
```

Wichtig: Unbedingt WHERE-Klausel verwenden!

Bei der Verwendung des UPDATE-Befehls ist die Verwendung einer passenden WHERE-Klausel extrem wichtig, da sich die Änderungen sonst auf alle Datensätze der Tabelle auswirken, was in den meisten Fällen nicht beabsichtigt ist.

Innerhalb der SET-Klausel kann man auch Bezug auf Felder der Tabelle selbst nehmen, was unter anderem dafür genutzt werden kann, um den Wert eines numerischen Feldes zu erhöhen. Um beispielsweise in der Tabelle *MeineBuecher* die Seitenanzahl um die vier Umschlagseiten zu erhöhen, können Sie folgende Abfrage verwenden:

```
UPDATE MeineBuecher
SET Seiten = Seiten + 4
```

In diesem Fall wird sogar bewusst auf eine WHERE-Klausel verzichtet. Allerdings muss man beachten, dass diese UPDATE-Anweisung nicht mehrfach ausgeführt wird, da sonst natürlich auch das Feld *Seiten* mehrfach erhöht wird.

In der SET-Klausel können auch mehrere Änderungen auf einmal durchgeführt werden, wenn diese mit Kommata voneinander getrennt werden.

```
UPDATE MeineBuecher
SET Sprache='deutsch', Hardcover=1
WHERE Titel='Programmieren mit dem .NET Compact Framework'
```

Hinweis: UPDATE und DELETE mit JOINs verwenden

Sowohl UPDATE als auch DELETE können mit Tabellenverknüpfungen kombiniert werden, um auch spezielle Anforderungen realisieren zu können. Diese komplexeren Formen der entsprechenden SQL-Anweisungen werden in einem späteren Kapitel behandelt.

DELETE und TRUNCATE TABLE zum Löschen von Daten

Um Daten zu löschen, wird die DELETE-Anweisung verwendet. In der Grundform reicht es aus, neben der Tabelle eine entsprechende WHERE-Klausel anzugeben, mit der die zu löschenden Zeilen eingeschränkt werden.

```
DELETE FROM [Tabelle]
WHERE [Bedingung]
```

Die folgende Anweisung löscht alle Bücher aus der Tabelle *MeineBuecher*, die das Hardcover-Kennzeichen gesetzt haben:

```
DELETE FROM MeineBuecher
WHERE Hardcover=1
```

Wichtig: Unbedingt WHERE-Klausel verwenden!

Wie beim UPDATE ist auch bei der DELETE-Anweisung die Verwendung einer passenden WHERE-Klausel extrem wichtig, da sonst alle Datensätze der Tabelle gelöscht werden, was nur in den wenigsten Fällen beabsichtigt sein dürfte.

Sollten Sie wirklich einmal alle Datensätze einer Tabelle löschen wollen, so können Sie statt des DELETE-Befehls die Anweisung TRUNCATE TABLE (gefolgt vom Tabellennamen) verwenden. Da die Löschoperation im Gegensatz zum DELETE beim TRUNCATE TABLE nicht protokolliert wird, läuft sie – insbesondere bei großen Tabellen – deutlich schneller ab.

```
TRUNCATE TABLE [Tabelle]
```

Hinweis: DELETE und TRUNCATE TABLE löschen nur Daten, keine Tabellen

Sowohl DELETE als auch TRUNCATE TABLE löschen nur die Datenzeilen, nicht die Tabellen selbst. Die SQL-Anweisungen zum Anlegen und Löschen von Tabellen werden in Kapitel 10, *Datenbankadministration mit SQL* behandelt.

6.5 Erstellen und Verwenden von Sichten

Sichten können das Arbeiten mit Daten erheblich vereinfachen. Dabei gibt es allerdings ein paar Besonderheiten zu beachten, auf die ich in diesem Abschnitt näher eingehen will. Bevor Sie aber mit Sichten arbeiten, müssen erst einmal Sichten existieren. Im vorigen Kapitel haben Sie schon gelernt, wie man Sichten mithilfe der grafischen Oberfläche im SQL Server Management Studio erstellen kann. Wenn Sie allerdings ohnehin mit SQL-Anweisungen arbeiten, geht dies auch etwas schneller.

Erstellen von Sichten

Eine Sicht lässt sich beinahe genauso leicht erstellen wie eine SQL-Abfrage. Dazu müssen Sie vor die entsprechende SQL-Anweisung, für die Sie eine Sicht erstellen wollen, lediglich die Schlüsselwörter `CREATE VIEW` gefolgt vom Namen der Sicht und dem Schlüsselwort `AS` setzen.

Erstellen wir nun eine Sicht, die Ihnen alle CD-*Titel* (sowie deren *ID* und *Erscheinungsjahr*) Ihrer Sammlung zurückgibt, die ab 2005 erschienen sind. Geben Sie dazu folgende Anweisung in ein Abfragefenster ein, nachdem Sie sich mit der Datenbank *MediaBase* verbunden haben:

```
CREATE VIEW dbo.VW_CDsAb2005 AS
SELECT ID, Titel, Erscheinungsjahr
FROM dbo.CD
WHERE (Erscheinungsjahr > 2004)
```

Best Practices: Erst die Abfrage testen, dann die Sicht erstellen

Sie können am einfachsten sicherstellen, dass die Sicht auch so arbeitet, wie Sie es beabsichtigen, wenn Sie zuerst die zugehörige `SELECT`-Abfrage erstellen und diese aufrufen. Erst wenn die syntaktischen und logischen Fehler behoben sind, setzen Sie die `CREATE VIEW`-Anweisung davor und erstellen dadurch eine auf der getesteten Abfrage basierende Sicht.

Wollen Sie eine Sicht später anpassen, können Sie die Sicht mit der `DROP`-Anweisung löschen und mit der neuen Definition neu anlegen. Mit den folgenden Anweisungen können Sie der Sicht ein weiteres Feld hinzufügen:

```
DROP VIEW dbo.VW_CDsAb2005

CREATE VIEW dbo.VW_CDsAb2005 AS
SELECT ID, Titel, Interpret, Erscheinungsjahr
FROM dbo.CD
WHERE (Erscheinungsjahr > 2004)
```

Wenn Sie diese beiden Anweisungen in einem Schritt ausführen, erhalten Sie folgende Fehlermeldung:

'CREATE VIEW' muss die erste Anweisung in einem Abfragebatch sein.

Dies können Sie umgehen, indem Sie entweder beide Anweisungen separat ausführen (indem vorher nur die jeweils auszuführende Anweisung markiert wird). Alternativ können Sie auch die Anweisung `GO` zwischen die beiden SQL-Kommandos setzen, die dafür sorgt, dass die Anweisungen wie separate SQL-Skripts (Abfragebatches) interpretiert werden.

Noch einfacher lässt sich die Sicht nachträglich ändern, indem Sie die `ALTER VIEW`-Anweisung verwenden:

```
ALTER VIEW dbo.VW_CDsAb2005 AS
SELECT ID, Titel, Interpret, Erscheinungsjahr
FROM dbo.CD
WHERE (Erscheinungsjahr > 2004)
```

Das funktioniert natürlich nur, wenn die entsprechende Sicht schon existiert. Ist keine Sicht mit dem angegebenen Namen vorhanden, erhalten Sie eine entsprechende Fehlermeldung.

Verwenden von Sichten in SELECT-Abfragen

So einfach wie das Erstellen ist auch die Nutzung von Sichten. Diese können in SQL-Abfragen fast genau so verwendet werden wie Tabellen, da diese zur Ausführung aufgelöst werden. Das bedeutet, dass die Sichtnamen vor der Ausführung intern durch die Abfrage ersetzt werden, auf deren Basis die Sicht definiert ist.

Insbesondere bei `SELECT`-Anweisungen gibt es eigentlich keinen Unterschied. Für die gerade erstellte Sicht liefert die Abfrage `SELECT * FROM VW_CDsAb2005` dasselbe Ergebnis wie die Abfrage, die der Sicht hinterlegt ist.

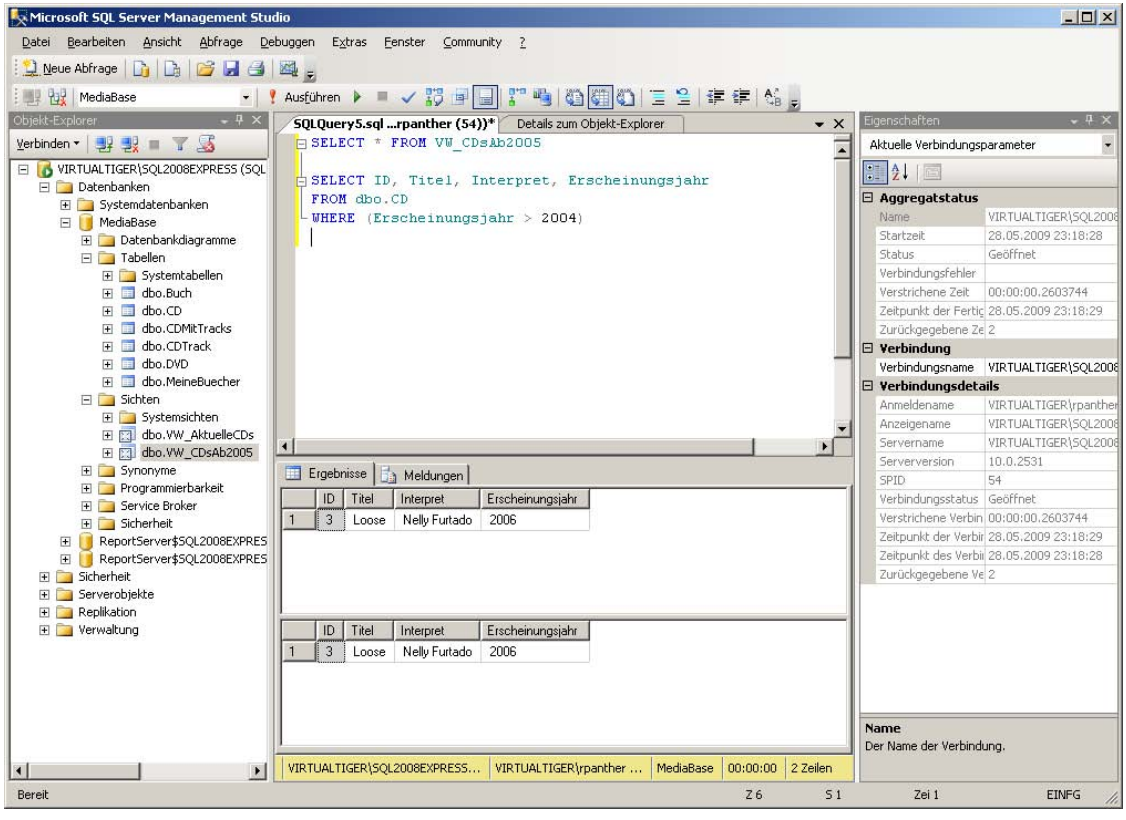


Abbildung 6.3: Zwei Abfragen – dasselbe Ergebnis

Dabei ist zu beachten, dass trotz der Angabe des Sternchens nicht alle Felder der Tabelle zurückgegeben werden, sondern nur diejenigen Felder, die in der Sicht definiert sind. Somit können Sie Sichten nutzen, um Felder zu verstecken.

Best Practices: Sichten nutzen, um technische Felder zu verbergen

In der Praxis werden Sichten tatsächlich oft genutzt, um technische Felder (also solche, die keinen fachlichen Hintergrund haben) vor den Nutzern der Sicht zu verbergen. So stellen beispielsweise in größeren Entwicklungsteams Datenbankentwickler Sichten zur Verfügung, die keine ID-Felder, interne Kennzeichen etc. nach außen geben. Diese Sichten werden dann von Anwendungsentwicklern genutzt, um fachliche Anforderungen umzusetzen, ohne direkt auf die Tabellen zuzugreifen.

Sichten lassen sich aber auch – genauso wie Tabellen selbst – mit weiteren SQL-Klauseln kombiniert nutzen, um diese beispielsweise mit weiteren Bedingungen einzuschränken, mit anderen Tabellen zu verknüpfen etc.

Die folgende Abfrage liefert alle CDs der Erscheinungsjahre 2005 bis 2007, wobei die Untergrenze (2005) durch die Sicht definiert wird, während die Obergrenze (< 2008) in der Abfrage hinzugefügt wurde:

```
SELECT *
FROM VW_CDsAb2005
WHERE (Erscheinungsjahr < 2008)
```

Die folgende Abfrage liefert alle CDs ab 2005 mit den dazugehörenden Track-Titeln:

```
SELECT *
FROM VW_CDsAb2005 AS CD05
INNER JOIN dbo.CDTrack AS TRK ON CD05.ID=TRK.idCD
```

Dabei wurde *CD05* als Alias für die Sicht verwendet, damit bei der `JOIN`-Bedingung nicht der komplette Name der Sicht wiederholt werden muss. Die Feldangaben in der `SELECT`-Klausel lassen sich natürlich auch genauer spezifizieren, sodass beispielsweise beide in der Sicht definierten Felder plus einige ausgewählte Felder aus der Tabelle *CDTrack* zurückgeliefert werden:

```
SELECT CD05.*, TRK.Titel as TrackTitel, TRK.Interpret as TrackInterpret
FROM VW_CDsAb2005 AS CD05
INNER JOIN dbo.CDTrack AS TRK ON CD05.ID=TRK.idCD
```

Verwenden von Sichten für Datenänderungsoperationen

Bei Datenänderungsoperationen auf Sichten sind einige Einschränkungen zu berücksichtigen. Sofern sich die Sicht auf lediglich eine Tabelle bezieht, können Sie Zeilen ändern und löschen, als wenn Sie in der Tabelle selbst arbeiten würden. Beim Einfügen von Zeilen ist zu beachten, dass alle Pflichtfelder der Tabelle (also solche, die als *NOT NULL* deklariert sind) Bestandteil der Sicht sind, damit für sie bei der `INSERT`-Anweisung ein Wert angegeben werden kann. Das ist natürlich für Felder mit Identitätsspezifikation und Felder, die einen Standardwert in der Tabellendefinition hinterlegt haben, nicht nötig, da diese ja automatisch einen Wert zugewiesen bekommen.

Da für die *CD*-Tabelle lediglich die *ID*-Spalte als Pflichtfeld definiert ist, diese aber über eine Identitätsspezifikation automatisch Werte erhält, können Sie mit der View *VW_CDsAb2005* problemlos weitere Zeilen einfügen:

```
INSERT INTO VW_CDsAb2005 (Interpret, Titel, Erscheinungsjahr)
VALUES ('Erwin Mustermann', 'Best Of Erwin Mustermann', 2003)
```

Wenn Sie die Sicht danach noch einmal abfragen, werden Sie feststellen, dass die neue Zeile nicht angezeigt wird. Das liegt daran, dass das beim `INSERT` angegebene Erscheinungsjahr (2003) durch die `WHERE`-Klausel `Erscheinungsjahr > 2004` in der Sicht ausgeschlossen wird. Nun könnte man sich fragen, warum der SQL Server dann das Einfügen des Datensatzes über die Sicht überhaupt zugelassen hat. Das liegt daran, dass die Bedingungen einer Sicht bei Änderungsoperationen normalerweise nicht überprüft werden. Dies lässt sich aber ändern, indem Sie die Sicht mit `CHECK OPTION` erstellen. Setzen wir nun diese Option nachträglich mit dem `ALTER VIEW`-Befehl:

```
ALTER VIEW dbo.VW_CDsAb2005 AS
SELECT ID,Titel, Interpret, Erscheinungsjahr
FROM dbo.CD
WHERE (Erscheinungsjahr > 2004)
WITH CHECK OPTION
```

Wenn Sie nun noch einmal versuchen, über die Sicht eine Zeile einzufügen, die ein Erscheinungsjahr beinhaltet, das nicht größer als 2004 ist, wird die Aktion mit folgender Fehlermeldung quittiert:

Meldung 550, Ebene 16, Status 1, Zeile 1

Fehler beim Einfügen oder Aktualisieren, da die Zielsicht WITH CHECK OPTION angibt oder sich auf eine Sicht erstreckt, die WITH CHECK OPTION angibt, und mindestens eine Ergebniszeile nicht der CHECK OPTION-Einschränkung entsprach.

Diese Einschränkung gilt ebenfalls für `UPDATE`- oder `DELETE`-Anweisungen, die sich auf Zeilen der Tabelle beziehen, die durch die `WHERE`-Bedingung in der Sicht ausgeschlossen sind.

Noch etwas komplizierter wird es bei Datenänderungsaktionen auf Sichten, die auf mehreren Tabellen basieren. Hierbei dürfen sich die Änderungen nur auf eine der in der Sicht verwendeten Tabellen beziehen.

Bereits in Kapitel 5 wurde eine Sicht *VW_AktuelleCDs* definiert, die sich auf Daten aus den Tabellen *CD* und *CDTrack* bezieht. Falls die Sicht noch nicht auf dem Server vorhanden ist, können Sie diese mit folgender Anweisung erstellen:

```
CREATE VIEW dbo.VW_AktuelleCDs
AS
SELECT TOP (100) PERCENT dbo.CD.ID, dbo.CD.Titel, dbo.CD.Interpret, dbo.CD.AnzahlCDs, dbo.CD.Erscheinungsjahr,
         dbo.CD.Bewertung, dbo.CDTrack.CDNr,
         dbo.CDTrack.TrackNr, dbo.CDTrack.Titel AS TrackTitel, dbo.CDTrack.Interpret AS TrackInterpret,
dbo.CDTrack.Dauer,
         dbo.CDTrack.Bewertung AS TrackBewertung, dbo.CD.Kategorie
FROM dbo.CD
         INNER JOIN dbo.CDTrack ON dbo.CD.ID = dbo.CDTrack.idCD
WHERE dbo.CD.Erscheinungsjahr > 2000
ORDER BY dbo.CD.Titel
```

Eine DELETE-Anweisung ist hier praktisch nicht mehr verwendbar, da sich die Sicht auf zwei Tabellen bezieht und nicht in beiden Tabellen mit einer Anweisung gelöscht werden kann. Die Anweisung DELETE FROM VW_AktuelleCDs WHERE ID=2 würde beispielsweise mit folgender Fehlermeldung quittiert:

Meldung 4405, Ebene 16, Status 1, Zeile 1

Die Sicht oder Funktion 'VW_AktuelleCDs' kann nicht aktualisiert werden, da die Änderung sich auf mehrere Basistabellen auswirkt.

Dieselbe Einschränkung gilt für INSERT- und UPDATE-Anweisungen. Das Einfügen einer neuen Zeile mit folgender Anweisung funktioniert dagegen, da hier lediglich Felder aus der Tabelle *CD* verwendet werden:

```
INSERT INTO VW_AktuelleCDs (Interpret, Titel, Erscheinungsjahr)
VALUES ('Erwin Mustermann', 'The Very Best Of Erwin Mustermann', 2007)
```

Ein INSERT auf Felder der Tabelle *CDTrack* ist dagegen über die View *VW_AktuelleCDs* nicht möglich, da das Pflichtfeld *idCD* nicht Bestandteil der Sicht ist und damit nicht gesetzt werden kann. Der Befehl

```
INSERT INTO VW_AktuelleCDs (TrackTitel)
VALUES ('SingSang')
```

wird daher mit folgender Fehlermeldung quittiert:

Meldung 515, Ebene 16, Status 2, Zeile 1

Der Wert NULL kann in die 'idCD'-Spalte, 'MediaBase.dbo.CDTrack'-Tabelle nicht eingefügt werden. Die Spalte lässt NULL-Werte nicht zu. Fehler bei INSERT.

Beim UPDATE lassen sich Felder aus beiden beteiligten Tabellen ändern, aber eben nicht mit einer Anweisung. Um die Bewertung für eine CD und deren Tracks auf die Note 1 zu setzen, funktionieren also folgende Anweisungen:

```
UPDATE VW_AktuelleCDs
SET Bewertung=1 WHERE ID=3
-- setzt die Bewertung für die CD

UPDATE VW_AktuelleCDs
SET TrackBewertung=1 where ID=3
-- setzt die Bewertung für die CD-Tracks
```

Der Versuch, beide Felder mit einer Anweisung zu setzen, schlägt dagegen fehl:

```
UPDATE VW_AktuelleCDs
SET Bewertung=1, TrackBewertung=1 where ID=3
```

ergibt folgende Fehlermeldung:

Meldung 4405, Ebene 16, Status 1, Zeile 1

Die Sicht oder Funktion 'VW_AktuelleCDs' kann nicht aktualisiert werden, da die Änderung sich auf mehrere Basistabellen auswirkt.

Möglich ist allerdings, in einem UPDATE auf eine Sicht die Felder einer Tabelle zu nutzen, um die Felder einer anderen Tabelle zu ändern. Somit lässt sich beispielsweise die Gesamtbewertung der CD als Einzelbewertungen für die verschiedenen Tracks übernehmen:

```
UPDATE VW_AktuelleCDs
SET TrackBewertung=Bewertung
WHERE TrackBewertung IS NULL
```

Mit der zusätzlichen `WHERE`-Klausel wird erreicht, dass die Änderung nur für diejenigen Tracks erfolgt, für die noch keine individuelle Bewertung gespeichert war.

6.6 Übungen zu diesem Kapitel

In diesem Abschnitt finden Sie einige Übungen zu diesem Kapitel. Die richtigen Antworten, Lösungen und den Programmcode finden Sie wie immer auf der Website *www.vsxpress.de.*

Übung 6.1

Wie können Sie erreichen, dass nur ein Teil eines SQL-Skripts im Abfrage-Editor des SQL Server Management Studio ausgeführt wird?

Übung 6.2

Schreiben Sie eine SQL-Abfrage, welche die Anzahl aller in der Datenbank erfassten CDs zurückliefert.

Übung 6.3

Erstellen Sie eine Abfrage, die alle CD-Titel mit Anzahl der darauf enthaltenen Tracks ausgibt.

Übung 6.4

Erstellen Sie einige `INSERT`-Anweisungen, um jeweils mindestens ein Buch, eine DVD und eine CD (mit Tracks) in die jeweiligen Tabellen einzufügen.

Übung 6.5

Stellen Sie sich vor, Sie würden alle Ihre Bücher, die Ihnen gar nicht gefallen, vernichten. Um dies auch in der Datenbank zu berücksichtigen, schreiben Sie eine Abfrage, die alle Bücher mit einer Bewertung schlechter als 4 (einem Wert größer als 4) löscht.

Übung 6.6

Erstellen Sie eine Sicht mit Namen *VW_BestOfMedia*, die Ihnen die Titel aller Bücher, CDs und DVDs mit Bewertung 1 in einer Gesamtliste zurückgibt.

6.7 Zusammenfassung

In diesem Kapitel haben Sie gelernt, wie Sie SQL-Anweisungen mit dem SQL Server Management Studio absetzen können. Danach wurden die wichtigsten SQL-Anweisungen behandelt:

- `SELECT` zum Abfragen von Daten aus einer oder mehreren Tabellen
- `INSERT` zum Einfügen von Daten in eine Tabelle
- `UPDATE` zum Ändern von Daten in einer Tabelle
- `DELETE` zum Löschen von Daten aus einer Tabelle

Im letzten Abschnitt des Kapitels wurde auf die Erstellung und Nutzung von Sichten eingegangen. Für die Nutzung von Sichten sind folgende Punkte zu beachten:

- Sichten können im Prinzip verwendet werden wie Tabellen, allerdings mit ein paar Einschränkungen.
- Damit Sichten für Einfügeoperationen verwendet werden können, müssen sie alle Pflichtfelder der beteiligten Tabellen enthalten, die weder über eine Identitätsspezifikation noch über einen Standardwert verfügen.
- Löschoperationen sind nur für solche Sichten sinnvoll anwendbar, die auf lediglich einer Tabelle basieren.
- Einfüge- und Änderungsaktionen über Sichten können nicht Felder in mehreren Tabellen (die von einer Sicht verwendet werden) schreiben.

An diesem Punkt haben Sie mit den Kapiteln 1 bis 6 die wichtigsten Grundlagen erlernt, um mit dem SQL Server arbeiten zu können. Je nachdem, ob Sie sich mehr für Datenbankentwicklung interessieren oder eher etwas über die Administration von Datenbanken erfahren möchten, können Sie nun mit *Teil III – Datenbankentwicklung* fortfahren, oder gleich zu *Teil IV – Datenbankadministration* springen.

Kapitel 7

Erweiterte SQL-Programmierung

In diesem Kapitel lernen Sie

- wie Sie mithilfe von Unterabfragen komplexe SELECT-Abfragen schreiben können
- welche Möglichkeiten es gibt, um komplexere Datenänderungsanweisungen zu schreiben
- wie man mit dem neuen MERGE-Befehl effizient Tabellen abgleichen kann

7.1 Komplexe SQL-SELECTs

Im vorigen Kapitel haben Sie gelernt, wie man einfache SQL-Abfragen schreiben kann. Die Möglichkeiten der SQL SELECT-Anweisungen gehen jedoch deutlich weiter. In vielen Umgebungen werden Abfragen verwendet, die über mehrere Tausend Zeichen lang sind und damit über mehrere Bildschirmseiten gehen (und das nicht nur aufgrund einer langen Aufzählung von Feldnamen).

Fallunterscheidung mit CASE

Oft gibt es Situationen, in denen in Abhängigkeit von einem Feld die Daten oder die Berechnung für ein weiteres Feld variieren sollen. Dies lässt sich realisieren, indem Sie alle möglichen Fälle in verschiedenen SELECT-Abfragen abhandeln und diese dann mit UNION verbinden. Als konkretes Beispiel fragen wir alle Zeilen der Tabelle *dbo.Buch* ab und je nachdem, ob das Kennzeichen Hardcover gesetzt ist oder nicht, wird in einer weiteren Spalte der Text *Hardcover* oder *Softcover* ausgegeben. Unter Verwendung von UNION SELECT sieht die entsprechende Abfrage wie folgt aus:

```
SELECT *, 'Hardcover'
FROM dbo.Buch
WHERE Hardcover=1
UNION
SELECT *, 'Softcover'
FROM dbo.Buch
WHERE Hardcover=0
```

Wenn Sie die Abfrage nun ausführen, erhalten Sie alle Zeilen der Tabelle *dbo.Buch* und jeweils dahinter eine Spalte, die im Klartext aussagt, ob das Buch ein Hardcover oder ein Softcover hat. Diese Variante der Abfrage funktioniert zwar, bringt jedoch zwei wesentliche Nachteile mit sich:

- Die Tabelle wird bei der Ausführung mehrfach durchlaufen, was nicht gerade zu einer optimalen Gesamtperformance beiträgt.
- Der größte Teil der Abfrage muss für jeden Fall wiederholt werden, was die Abfrage unübersichtlich und schwer lesbar macht.

Um diesem Dilemma abzuhelfen, kann man eine Fallunterscheidung innerhalb einer `SELECT`-Klausel nutzen. Diese hat die Grundform `CASE WHEN Bedingung THEN Wert1 ELSE Wert2 END` und liefert `Wert1` zurück, wenn die Bedingung erfüllt ist, und `Wert2`, wenn die Bedingung nicht erfüllt ist.

Angewendet auf das oben genannte Beispiel ergibt sich damit die folgende Abfrage:

```
SELECT *, CASE
                  WHEN Hardcover=1 THEN 'Hardcover'
  ELSE                 'Softcover'
         END AS Cover
FROM dbo.Buch
```

Die Abfrage liefert bei Ausführung dasselbe Ergebnis wie die erste Variante (eventuell in unterschiedlicher Reihenfolge der Zeilen), läuft aber deutlich schneller ab, weil die Tabelle *dbo.Buch* nur einmal durchlaufen werden muss. Auch wenn dies bei den momentan verwendeten Beispieltabellen noch keine Rolle spielt, weil die Zeilenanzahl noch sehr gering ist, darf man diesen Aspekt nicht vernachlässigen, denn im produktiven Einsatz können die Datenmengen schnell anwachsen.

Hinweis: Fallunterscheidungen schachteln

Wenn Sie einmal mehr als zwei Fälle unterscheiden müssen, lassen sich die Fallunterscheidungen auch beliebig schachteln. Dabei sollte man möglichst Klammern verwenden, damit eindeutig klargestellt ist, welche `THEN`- und `ELSE`-Zweige sich auf welche `CASE`-Bedingung beziehen.

Es können durchaus mehrere `WHEN`-Bedingungen verwendet werden, sodass man das oben gezeigte Beispiel noch verfeinern kann, indem man die Fälle `0` und `NULL` explizit unterscheidet:

```
SELECT *, CASE
                  WHEN Hardcover=1 THEN 'Hardcover'
                  WHEN Hardcover=0 THEN 'Softcover'
  ELSE                 '(unbekannt)'
         END AS Cover
FROM dbo.Buch
```

Sofern sich die Bedingungen alle auf dasselbe Feld beziehen, gibt es auch noch eine verkürzte Schreibweise, bei der das zu prüfende Feld nur einmal angegeben wird und hinter dem Schlüsselwort `WHEN` nur noch der entsprechende Wert angegeben wird:

```
SELECT *, CASE Hardcover
                  WHEN 1 THEN 'Hardcover'
                  WHEN 0 THEN 'Softcover'
  ELSE                 '(unbekannt)'
         END AS Cover
FROM dbo.Buch
```

Unterabfragen

Im letzten Kapitel wurde gezeigt, wie Sie mehrere Tabellen durch `JOIN`s und `UNION SELECT`s miteinander verbinden können. Es gibt aber noch eine weitere Möglichkeit, mehrere Tabellen oder Sichten in einer Abfragen unterzubringen: die Unterabfragen (Sub-SELECTs).

Die Idee von Unterabfragen liegt darin, das Ergebnis einer Abfrage in einer übergeordneten Abfrage weiterzuverwenden. Dabei unterscheidet man zwischen zwei Arten von Unterabfragen: sogenannte korrelierte Unterabfragen (die mit der aufrufenden Abfrage verknüpft sind) und nicht korrelierte Unterabfragen, die auch ohne die übergeordnete Abfrage ausführbar sind.

Beginnen wir der Einfachheit halber zuerst mit der letztgenannten Variante. Wenn Sie beispielsweise über eine Abfrage die Titel aller CDs ausgeben wollen, die mehr als 13 Tracks beinhalten, können Sie damit beginnen, eine Abfrage zu erstellen, die alle CD-IDs ausgibt, zu denen es mehr als 13 Datensätze in der Tabelle *dbo.CDTrack* gibt:

```
SELECT idCD
FROM dbo.CDTrack
GROUP BY idCD
HAVING COUNT(*)>13
```

Diese Abfrage wollen wir später als Unterabfrage verwenden. Der Vorteil einer nicht korrelierten Unterabfrage ist, dass Sie diese erst separat entwickeln und testen können und erst dann in die übergeordnete Abfrage einbauen, wenn sie das richtige Ergebnis liefert.

Führen Sie die Abfrage nun aus. Wenn das Ergebnis Ihren Erwartungen entspricht, können Sie an die übergeordnete Abfrage gehen. Hier wird der bereits aus dem vorigen Kapitel bekannte `IN`-Operator genutzt, allerdings nun nicht mit einer konstanten Wertmenge, sondern mit der Unterabfrage:

```
SELECT Titel, Interpret
FROM dbo.CD
WHERE ID IN
 (SEL        ECT idCD
 FROM         dbo.CDTrack
         GROUP BY idCD
 HAVI        NG COUNT(*)>13)
```

Neben dem `IN`-Operator gibt es noch einen zweiten Operator, der oft im Zusammenhang mit Unterabfragen verwendet wird. Der `EXISTS`-Operator wird normalerweise bei korrelierten Unterabfragen verwendet und ist genau dann wahr, wenn die Unterabfrage mindestens eine Ergebniszeile liefert. Das letzte Beispiel lässt sich leicht in eine korrelierte Unterabfrage mit `EXISTS` umwandeln:

```
SELECT Titel, Interpret
FROM dbo.CD
WHERE EXISTS
 (SEL        ECT idCD
 FROM         dbo.CDTrack
         WHERE idCD = dbo.CD.ID
         GROUP BY idCD
 HAVI        NG COUNT(*)>13)
```

Um die Unterabfrage mit der aufrufenden Abfrage zu verknüpfen, wurde eine zusätzliche `WHERE`-Bedingung eingebaut, die sicherstellt, dass für jeden Durchlauf der Tabelle *dbo.CD* nur diejenigen Einträge der Tabelle *dbo.CDTrack* gezählt werden, die auch die entsprechende *idCD* haben.

Genauso gut hätte man die Bedingung auch zur HAVING-Klausel der Unterabfrage hinzufügen können:

```
SELECT Titel, Interpret
FROM dbo.CD
WHERE EXISTS
 (SEL         ECT idCD
 FROM          dbo.CDTrack
         GROUP BY idCD
         HAVING COUNT(*)>13 AND idCD = dbo.CD.ID)
```

Das Ergebnis der Gesamtabfrage ist für alle drei genannten Varianten dasselbe und selbst der verwendete Ausführungsplan ist in allen drei Fällen gleich. Trotzdem ist die erste Variante zu bevorzugen, da diese besser lesbar und auch besser wartbar ist.

Zusätzlich zu IN und EXISTS können Sie Unterabfragen auch wie normale Funktionen verwenden, sofern Sie sicherstellen, dass die Unterabfrage nur einen einzigen Wert liefert. Wenn wir die Unterabfrage nun so umformulieren, dass sie die Anzahl Tracks für die jeweilige *CD* zurückliefert, würde dies wie folgt aussehen (und nach wie vor dasselbe Ergebnis liefern):

```
SELECT Titel, Interpret
FROM dbo.CD
WHERE (SELECT count(*)
 FROM          dbo.CDTrack
         WHERE idCD = dbo.CD.ID)>13
```

In dieser Form können Sie die Unterabfrage aber auch in der SELECT-Klausel verwenden, um beispielsweise die Anzahl der Tracks für jede *CD* auszugeben:

```
SELECT Titel, Interpret,
 (SEL         ECT count(*)
 FROM          dbo.CDTrack
         WHERE idCD = dbo.CD.ID) AS Anzahl
FROM dbo.CD
```

Dasselbe Ergebnis lässt sich allerdings auch mit einer geschickten Kombination aus LEFT JOIN und GROUP BY erzielen:

```
SELECT CD.Titel, CD.Interpret, count(CDTrack.ID) as Anzahl
FROM dbo.CD
         LEFT JOIN dbo.CDTrack ON CD.ID=CDTrack.idCD
GROUP BY CD.Titel, CD.Interpret
```

Der LEFT JOIN ist in diesem Fall nötig, da ein INNER JOIN sonst die CDs weglassen würde, zu denen keine Tracks gespeichert sind. Die Angabe einer Spalte aus der Tabelle *CDTrack* für die COUNT-Funktion ist wichtig, da bei einem COUNT(*) für die CDs ohne Tracks der Wert 1 zurückgeliefert würde (denn der LEFT JOIN erzeugt ja auch für diese CDs eine Zeile, bei denen die Felder der Tabelle *CDTrack* allerdings mit NULL gefüllt sind). Der Aufruf COUNT(CDTrack.ID) dagegen liefert die Anzahl der Zeilen, für die das Feld *CDTrack.ID* nicht NULL ist.

Es gibt noch eine weitere Möglichkeit, dasselbe Ergebnis zu erzielen: Eine Unterabfrage, welche die CD-ID und deren *TrackAnzahl* zurückgibt, kann über einen LEFT JOIN mit der Tabelle *CD* verbunden werden. Diese Variante hat den Charme, dass die Unterabfrage für sich selbst lauffähig ist und daher auch allein getestet werden kann, bevor sie in die übergeordnete SELECT-Abfrage eingebaut wird.

```
SELECT CD.Titel, CD.Interpret, TrackAnzahl.Anzahl
FROM dbo.CD
 LEFT          JOIN
  (SEL                 ECT idCD, count(*) AS Anzahl
  FROM                  dbo.CDTrack
                    GROUP BY idCD) AS TrackAnzahl
         ON CD.ID = TrackAnzahl.idCD
```

Es gibt aber noch einen weiteren Unterschied zu beachten: Da die Unterabfrage für die CDs, zu denen keine Tracks gespeichert sind, auch keine Zeilen liefert, enthält das Feld *TrackAnzahl.Anzahl* in der übergeordneten Abfrage aufgrund des `LEFT JOIN`s nicht – wie bei den anderen Varianten der Abfrage – den Wert 0, sondern stattdessen `NULL`.

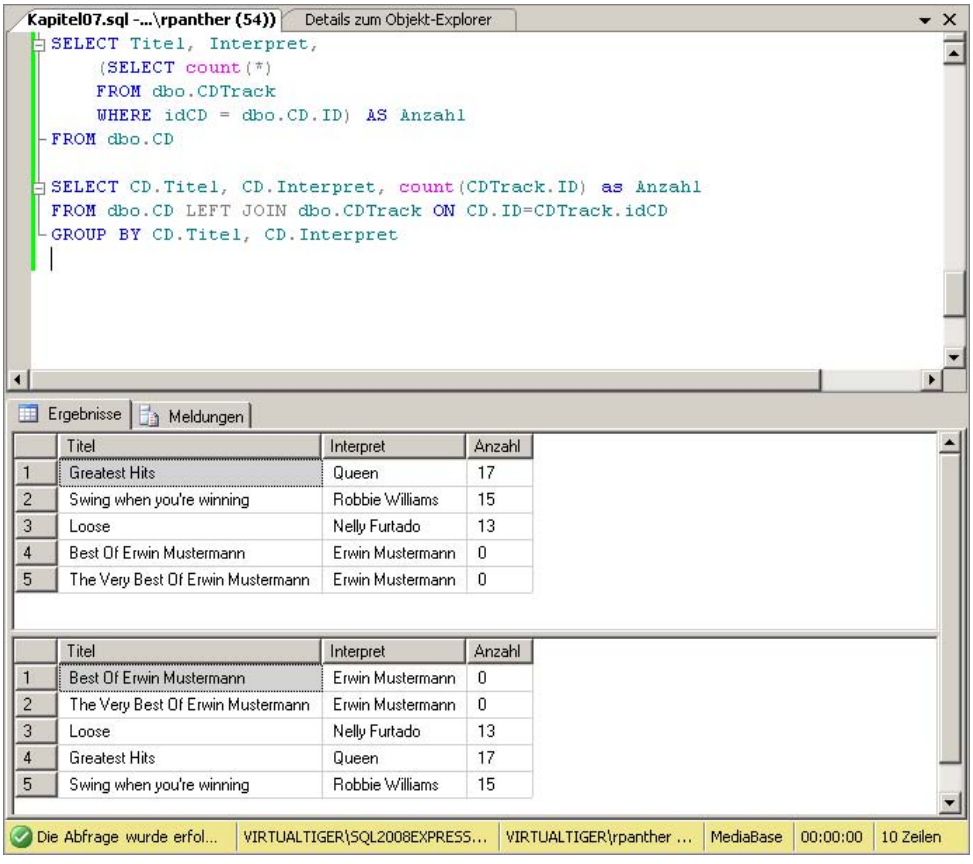


Abbildung 7.1: Zwei Abfragen – dasselbe Ergebnis

Generell empfiehlt es sich, für eine bessere Performance auf Unterabfragen möglichst zu verzichten und diese beispielsweise durch einen `JOIN` zu ersetzen. Allerdings sollte dies im Einzelfall überprüft werden, da hier auch andere Faktoren (wie die Größe der Tabellen, Indizes etc.) eine nicht unbedeutende Rolle spielen.

Hinweis: Unterabfragen schachteln

Genau wie Fallunterscheidungen lassen sich auch Unterabfragen nahezu beliebig schachteln. Da dies aber weder der Lesbarkeit der Abfrage noch der Performance zuträglich ist, sollte man diese Möglichkeit nur im Notfall nutzen.

Aggregierungsfunktionen mit COMPUTE nutzen

Bereits im letzten Kapitel haben Sie die Verwendung von Aggregierungsfunktionen im Zusammenhang mit der `GROUP BY`-Klausel kennengelernt. Nachteil dabei ist, dass durch die Gruppierung die Detailinformationen nicht mehr im Ergebnis auftauchen. Mit der `COMPUTE`-Klausel gibt es hierzu jedoch eine Alternative, über die – mithilfe der bekannten Aggregierungsfunktionen – Gesamtwerte zusätzlich zu den eigentlichen Abfrageergebnissen angezeigt werden können.

Dazu wird einfach am Ende der Abfrage die `COMPUTE`-Klausel hinzugefügt, in der Aggregierungen auf beliebigen Feldern durchgeführt werden. Im einfachsten Fall können Sie durch den Zusatz von `COMPUTE COUNT(ID)` die Anzahl der Ergebniszeilen ausgeben. (`COUNT(*)` ist in der `COMPUTE`-Klausel leider nicht verwendbar, sodass hier zumindest eine Tabellenspalte angegeben werden muss.)

Da mit einer solchen Abfrage zwei separate Ergebnisse ausgegeben werden, ist die Angabe der aggregierten Spalten – im Gegensatz zur Verwendung von Aggregierungsfunktionen mit `GROUP BY` – unabhängig von der `SELECT`-Klausel.

Die folgende Abfrage:

```
SELECT *
FROM dbo.CD
COMPUTE count(ID)
```

liefert damit dasselbe Ergebnis wie die beiden folgenden Abfragen zusammen:

```
SELECT *
FROM dbo.CD

SELECT count(ID)
FROM dbo.CD
```

Allerdings besteht hierbei der kleine, aber feine Unterschied, dass in der Variante mit zwei Anweisungen die Tabelle *dbo.CD* zweimal durchlaufen wird, was zulasten der Performance geht.

Probieren wir einmal ein etwas komplexeres Beispiel aus:

1. Öffnen Sie im SQL Server Management Studio ein Abfragefenster und verbinden Sie sich mit der lokalen SQL Server-Instanz *SQL2008Express* und der Datenbank *MediaBase* (z.B. durch die Anweisung `use MediaBase`).
2. Führen Sie im Abfrage-Editor folgende SQL-Anweisung aus:

   ```
   SELECT *
   FROM dbo.CD
   COMPUTE count(ID), min(Erscheinungsjahr), max(Erscheinungsjahr)
   ```

   Das Ergebnis ist eine Liste aller CDs gefolgt von einer Zeile, in der die Anzahl der Datensätze sowie das älteste und das jüngste Erscheinungsjahr zu sehen sind.

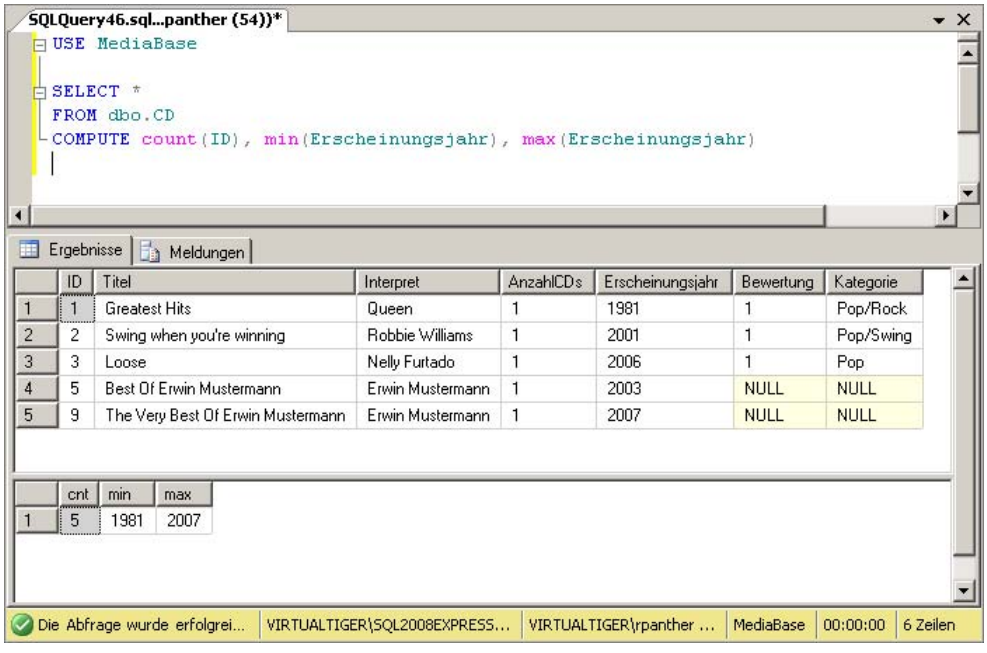


Abbildung 7.2: SELECT-Abfrage mit COMPUTE

Mit der `COMPUTE`-Anweisung lassen sich aber auch Zwischensummen berechnen. Dazu können Sie mit dem Zusatz `BY feldliste` eine Art Gruppierung in der `COMPUTE`-Klausel festlegen. Voraussetzung ist allerdings, dass die Abfrage davor auch genau nach diesen Gruppenfeldern sortiert ist. Innerhalb der Gruppe kann dann eine weitere Sortierung erfolgen, indem man zusätzliche Sortierfelder in der `ORDER BY`-Klausel angibt.

3. Erweitern Sie die Abfrage um eine `ORDER BY`-Klausel sowie den `BY`-Zusatz in der `COMPUTE`-Klausel, sodass diese wie folgt aussieht:

```
SELECT *
FROM dbo.CD
ORDER BY Interpret, Titel
COMPUTE count(ID), min(Erscheinungsjahr), max(Erscheinungsjahr) BY Interpret
```

4. Wenn Sie die Abfrage nun erneut ausführen, erhalten Sie für jeden Interpreten in der Tabelle die beiden oben genannten Ergebnisse.

7.2 Komplexe INSERTs, UPDATEs und DELETEs

Ähnlich wie die `SELECT`-Anweisung lassen sich auch die Befehle zum Ändern von Dateninhalten mit Fallunterscheidungen, Unterabfragen etc. versehen.

INSERT auf Basis von mehreren Tabellen

Bereits im vorigen Kapitel wurde mit der Anweisung `SELECT ... INTO` eine neue Tabelle *CDMitTracks* erzeugt, die sowohl Daten aus der Tabelle *CD* als auch aus der Tabelle *CDTrack* beinhaltet. Wenn Sie diese Tabelle nun mit der folgenden Anweisung leeren:

```
TRUNCATE TABLE CDMitTracks
```

kann sie später wieder mit einem `INSERT` gefüllt werden, dessen `SELECT`-Teil einen `JOIN` beinhaltet:

```
INSERT INTO CDMitTracks
SELECT CD.*, CDT.TrackNr, CDT.Titel as TrackTitel
FROM CD LEFT JOIN CDTrack as CDT ON CD.ID = CDT.idCD
```

Somit können Sie die Daten jederzeit mit denen aus den ursprünglichen Tabellen wieder aktualisieren. Alternativ wäre es natürlich auch möglich, die Tabelle komplett zu löschen und mit einem erneuten `SELECT ... INTO` wieder anzulegen. Diese Methode hat allerdings den Nachteil, dass sämtliche abhängigen Objekte wie beispielsweise Indizes ebenfalls wieder neu erstellt werden müssten.

Dadurch, dass die zweite Hälfte des `INSERT INTO`-Befehls eine beliebige `SELECT`-Anweisung ist, können Sie hier nach Bedarf auch alle erweiterten Möglichkeiten dieser Anweisung – wie beispielsweise Unterabfragen, Fallunterscheidungen mit `CASE` etc. – nutzen.

UPDATE auf Basis von mehreren Tabellen

Sie erinnern sich vielleicht noch an ein Beispiel aus dem letzten Kapitel, in dem ein `UPDATE` auf eine Sicht ausgeführt wurde, die Daten aus mehreren Tabellen beinhaltet:

```
UPDATE VW_AktuelleCDs
SET TrackBewertung=Bewertung
WHERE TrackBewertung IS NULL
```

Dasselbe Ergebnis können Sie auch ohne eine Sicht erzielen. Zwei mögliche Varianten dazu möchte ich nun vorstellen. Die Variante, die der Sicht am ehesten entspricht, ist die Verwendung eines JOINs. Dazu wird in der `UPDATE`-Anweisung eine `FROM`-Klausel angegeben – ähnlich wie Sie das von `SELECT`-Abfragen bereits kennen. Nach dem `UPDATE` selbst wird dann die Tabelle angegeben, in der die Datenänderungen ausgeführt werden. Denn wie bei einem `UPDATE` auf eine Sicht können Sie auch hier nur Daten in einer der beteiligten Tabellen ändern. Damit sieht die Anweisung, welche die Bewertung aus der *CD*-Tabelle auf die *CDTracks* anwendet (sofern hier noch keine gesetzt ist), wie folgt aus:

```
UPDATE dbo.CDTrack
SET CDTrack.Bewertung=CD.Bewertung
FROM dbo.CD
         INNER JOIN dbo.CDTrack ON CD.ID=CDTrack.idCD
WHERE CDTrack.Bewertung IS NULL
```

Die Einschränkung auf CDs ab einem gewissen Erscheinungsjahr habe ich hier der Einfachheit halber weggelassen. Alternativ dazu kann man auch den Wert, auf den das Feld *Bewertung* gesetzt wird, über eine (korrelierte) Unterabfrage auslesen:

```
UPDATE dbo.CDTrack
SET Bewertung=(SELECT Bewertung FROM dbo.CD WHERE CD.ID=CDTrack.idCD)
WHERE CDTrack.Bewertung IS NULL
```

DELETE auf Basis von mehreren Tabellen

Ähnlich wie beim Ändern von Daten lassen sich auch beim `DELETE` Unterabfragen, JOINs etc. nutzen. Auch hier kann man in einer zusätzlichen `FROM`-Klausel mehrere Tabellen mit entsprechenden JOIN-Bedingungen angeben.

Vorher erstellen wir jedoch eine Kopie der Tabelle *dbo.CD*, um die bereits angelegten Beispieldaten nicht zu zerstören:

```
SELECT * INTO dbo.CD2 FROM dbo.CD
```

Hinweis: Transaktionen

Um Änderungsaktionen gefahrlos ausprobieren zu können, ohne die Daten dauerhaft zu verändern, würde man normalerweise eher Transaktionen nutzen, die man dann abschließend wieder rückgängig machen kann, um den Ausgangszustand wiederherzustellen. Dieses Thema wird allerdings erst im nächsten Kapitel behandelt, sodass hier stattdessen mit einer Tabellenkopie gearbeitet wird.

Mit der folgenden Abfrage lassen sich nun in der Tabelle *dbo.CD2* alle CDs löschen, zu denen keine Tracks in der Tabelle *dbo.CDTrack* erfasst sind:

```
DELETE FROM dbo.CD2
FROM dbo.CD2 LEFT JOIN dbo.CDTrack ON CD2.ID = CDTrack.idCD
WHERE CDTrack.idCD IS NULL
```

Wenn Sie danach einen Blick in die Tabelle *dbo.CD2* werfen, werden Sie feststellen, dass nur noch die Einträge übrig geblieben sind, zu denen es auch Datensätze in der Tabelle *dbo.CDTrack* gibt. Etwas verwirrend ist bei der verwendeten Anweisung die doppelte `FROM`-Klausel, wobei die erste angibt, in welcher Tabelle gelöscht wird, und die zweite definiert, welche Tabellen gelesen werden, um die zu löschenden Zeilen zu ermitteln. Für die erste `FROM`-Klausel ist das Schlüsselwort allerdings optional. Wenn Sie dieses also weglassen, wird die Anweisung dadurch besser lesbar (der Aufbau entspricht dann im Prinzip der `UPDATE`-Anweisung, nur ohne `SET`-Klausel):

```
DELETE dbo.CD2
FROM dbo.CD2 LEFT JOIN dbo.CDTrack ON CD2.ID = CDTrack.idCD
WHERE CDTrack.idCD IS NULL
```

Dasselbe Ergebnis lässt sich auch auf eine andere Weise – mit einer Unterabfrage – erzielen. Um dies auszuprobieren, muss davor allerdings die Tabelle *dbo.CD2* gelöscht und neu aufgebaut werden, damit wieder Daten zum Löschen vorhanden sind:

```
DROP TABLE dbo.CD2
SELECT * INTO dbo.CD2 FROM dbo.CD
```

Nun können Sie die folgende SQL-Anweisung ausführen, die in einer Unterabfrage alle CD-IDs auflistet, zu denen Tracks gespeichert sind, und diese Informationen nutzt, um alle CDs aus der Tabelle *dbo.CD2* zu löschen, deren IDs nicht in dieser Liste auftauchen:

```
DELETE FROM dbo.CD2
WHERE ID NOT IN (SELECT DISTINCT idCD FROM dbo.CDTrack)
```

Wenn Sie anschließend durch einen Blick in die Tabelle *dbo.CD2* das Ergebnis überprüft haben, können Sie diese wieder mit der `DROP TABLE`-Anweisung löschen.

7.3 Daten abgleichen mit dem MERGE-Befehl

Oft kommt es in der Praxis vor, dass Daten in eine Tabelle geschrieben werden sollen, von denen nicht bekannt ist, ob die entsprechenden Zeilen bereits existieren und diese nur aktualisiert werden müssen, oder aber komplett neu anzulegen sind.

Das im Folgenden verwendete Beispiel nutzt eine Tabelle *dbo.BuchImport*, um damit die Daten der bestehenden Tabelle *dbo.Buch* zu aktualisieren. Bevor Sie das Beispiel selbst ausprobieren können, benötigen Sie allerdings noch eine Tabelle *dbo.BuchImport*. Um diese zu erstellen, verwenden Sie folgende Anweisung:

```
SELECT Top 1 *
INTO dbo.BuchImport
FROM dbo.Buch
```

Ändern Sie nun den Inhalt dieser Zeile – damit es später auch etwas zu aktualisieren gibt – mit folgender Anweisung:

```
UPDATE dbo.BuchImport
SET Kategorie = 'Fachbuch'
```

Fügen Sie außerdem eine neue Zeile hinzu, die ein Buch enthält, das noch nicht in der ursprünglichen Tabelle vorhanden ist:

```
INSERT INTO dbo.BuchImport (Autor, Titel, Kategorie)
VALUES ('Robert Panther', 'Das Pocket PC Programmierhandbuch', 'Fachbuch')
```

Wenn Sie die Tabelle *dbo.BuchImport* nun öffnen (vorher muss die Ansicht im Objekt-Explorer noch aktualisiert werden), sehen Sie zwei Zeilen: die neu hinzugefügte sowie die aus der Tabelle *dbo.Buch* kopierte, bei der allerdings das Feld *Kategorie* geändert wurde.

Die klassische Variante (ohne MERGE)

Um nun die Tabelle *dbo.Buch* mit den Daten einer Tabelle *dbo.BuchImport* zu aktualisieren, würde man folgende `UPDATE`-Anweisung verwenden:

```
UPDATE dbo.Buch
SET Buch.Autor = BuchImport.Autor,
 Buch         .Titel = BuchImport.Titel,
          Buch.Kategorie = BuchImport.Kategorie
FROM dbo.Buch
          INNER JOIN dbo.BuchImport ON Buch.Titel=BuchImport.Titel
```

Anschließend wäre noch ein `INSERT` nötig, um die noch fehlenden Zeilen zu ergänzen:

```
INSERT INTO dbo.Buch (Autor, Titel, Kategorie)
SELECT Autor, Titel, Kategorie
FROM dbo.BuchImport
WHERE Titel NOT IN (SELECT Titel FROM dbo.Buch)
```

(Der Einfachheit halber werden in den oben dargestellten Abfragen nur die Felder *Autor*, *Titel* und *Kategorie* geschrieben. Für einen vollständigen Abgleich müsste man hier noch alle anderen Felder – bis auf das *ID*-Feld, das durch eine Identitätsspezifikation automatisch gefüllt wird – ergänzen.)

Um auch die Zeilen zu entfernen, die in der Tabelle *dbo.Buch* vorhanden sind, aber in der aktuelleren Tabelle *dbo.BuchImport* nicht, wäre noch eine separate DELETE-Anweisung nötig:

```
DELETE FROM dbo.Buch
WHERE Titel NOT IN (SELECT Titel FROM dbo.BuchImport)
```

Die neue Variante (mit MERGE)

Seit SQL Server 2008 gibt es mit dem neuen MERGE-Befehl endlich eine Möglichkeit, alle drei Aktionen in einer Anweisung gemeinsam auszuführen. Dadurch wird auch die Ausführungsgeschwindigkeit deutlich erhöht, was die Anweisung auch für das Aktualisieren von Data Warehouses interessant macht.

Der Grundaufbau der Anweisung ist relativ einfach: Nach dem Schlüsselwort MERGE INTO wird angegeben, welche Tabelle aktualisiert werden soll. Darauf folgt die USING-Klausel, in der die Datenquelle sowie die zugehörige JOIN-Bedingung angegeben sind. Anschließend folgen bis zu drei Blöcke, die dem Schema WHEN Bedingung THEN Aktion folgen. Dabei sind als Bedingungen folgende Varianten zulässig:

- WHEN MATCHED – es gibt eine Entsprechung der Zeile in beiden Tabellen.
- WHEN NOT MATCHED – die Zeile fehlt in der zu aktualisierenden Datei.
- WHEN NOT MATCHED BY SOURCE – zur Zeile der zu aktualisierenden Datei ist in der Quelldatei keine Entsprechung vorhanden.

Hinweis: Geänderte Syntax!

In den früheren Betas und Technology Previews des SQL Servers 2008 hieß die Klausel WHEN NOT MATCHED BY SOURCE noch WHEN SOURCE NOT MATCHED. Diese alte Syntax ist daher auch noch in vielen Dokumentationen und Büchern zu finden, sie ist aber nicht mehr gültig.

Aus den gerade geschilderten Bestandteilen ergibt sich für die Beispieldatenbank folgende Anweisung:

```
MERGE INTO dbo.Buch
USING BuchImport ON Buch.Titel=BuchImport.Titel
WHEN MATCHED
         THEN UPDATE SET Buch.Autor = BuchImport.Autor,
  Buch                  .Titel = BuchImport.Titel,
  Buch                  .Kategorie = BuchImport.Kategorie
WHEN NOT MATCHED
         THEN INSERT (Autor, Titel, Kategorie)
         VALUES (BuchImport.Autor, BuchImport.Titel, BuchImport.Kategorie)
WHEN NOT MATCHED BY SOURCE THEN
 DELE         TE;
```

Wichtig!

Wie in vielen anderen Programmiersprachen (wie z.B. C#, Java etc.) schon lange üblich, ist es mittlerweile auch in SQL möglich, Anweisungen mit einem Semikolon abzuschließen. Während dies in T-SQL bei den meisten Anweisungen optional ist, ist dies beim MERGE-Befehl zwingend erforderlich.

Um die Auswirkungen des MERGE-Befehls zu prüfen, macht es Sinn, sowohl vor der Ausführung als auch nach der Ausführung den Inhalt der Tabelle *dbo.Buch* über eine einfache SELECT-Anweisung auszugeben.

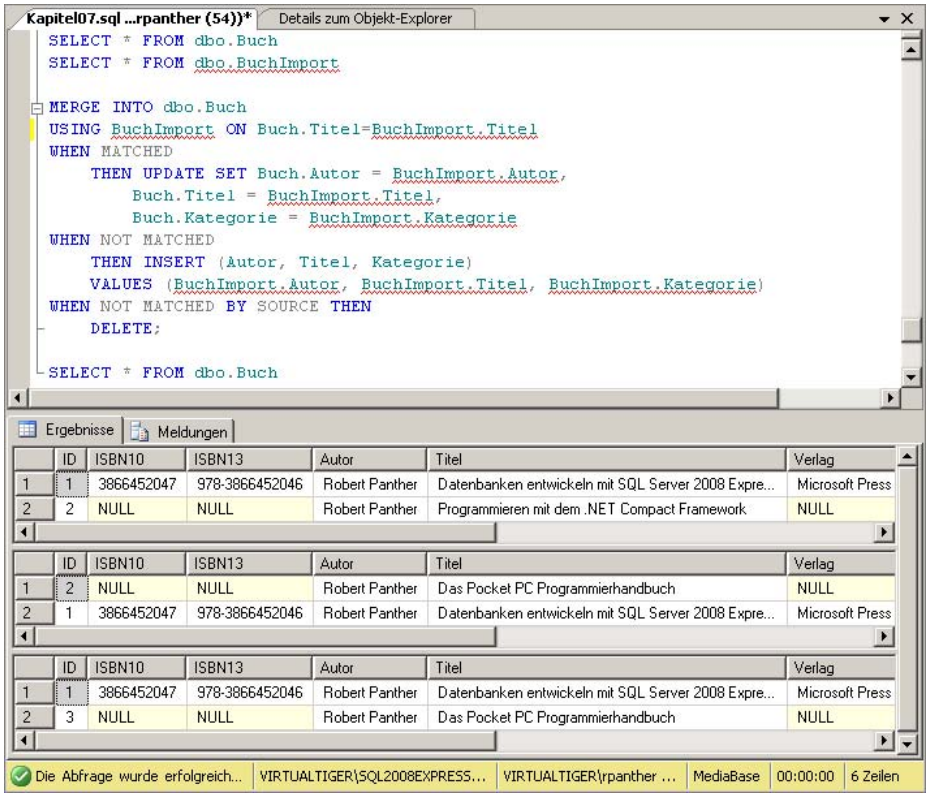


Abbildung 7.3: Der MERGE-Befehl und seine Auswirkungen

Das Ergebnis des MERGE-Befehls lässt sich nun leicht nachprüfen. Der Datensatz mit der *ID* 1 wurde aktualisiert, wie am Feld *Kategorie* leicht zu sehen ist (dazu müssen Sie allerdings etwas nach rechts scrollen). Der Zeile mit der *ID* 2 wurde durch die Option NOT MATCHED BY SOURCE gelöscht, weil der Titel nicht in der Tabelle *dbo.BuchImport* vorhanden war, und die Zeile zum Buch mit dem Titel *Das Pocket PC Programmierhandbuch* ist neu hinzugekommen, hat nun aber durch die Identitätsspezifikation die neue *ID* 3 bekommen.

Lassen Sie sich bei der Abbildung nicht von den roten Linien irritieren, mit denen die Vorkommen des Tabellennamens *BuchImport* unterlegt sind. Da die Tabelle *dbo.BuchImport* erst innerhalb des SQL-Skripts erzeugt wurde, weiß IntelliSense nicht, dass es sich dabei um einen gültigen Tabellennamen handelt.

Hintergrundinfo: Mehr als nur Standard-SQL

Wie die meisten SQL-Anweisungen ist auch der MERGE-Befehl keine Erfindung von Microsoft, sondern im ISO-SQL-Standard definiert. Microsoft hat diese Definition implementiert, aber noch um die Option WHEN NOT MATCHED BY SOURCE erweitert.

7.4 Übungen zu diesem Kapitel

In diesem Abschnitt finden Sie einige Übungen zu diesem Kapitel. Die richtigen Antworten, Lösungen und den Programmcode finden Sie wie immer auf der Website *www.vsxpress.de.*

Übung 7.1

Entwickeln Sie eine `SELECT`-Anweisung, die zu allen Titeln der Tabelle *dbo.Buch* angibt, ob auch eine entsprechende DVD mit demselben Titel gespeichert ist.

Übung 7.2

Schreiben Sie eine `UPDATE`-Anweisung, die das Feld *Kategorie* der Tabelle *dbo.Buch* mit Kategorieeinträgen von gleichnamigen Filmen aus der Tabelle *dbo.DVD* aktualisiert, sofern das Feld vorher leer war.

Übung 7.3

Entwerfen Sie eine `MERGE`-Anweisung, welche die Tabelle *dbo.DVD* mit einer Tabelle *dbo.DVDImport* abgleicht, ohne bereits bestehende Datensätze in der Tabelle *dbo.DVD* zu löschen. Benutzen Sie für die Verknüpfung die Felder *Titel* und *Erscheinungsjahr*. Abzugleichen sind die Felder *Titel*, *Schauspieler*, *Produzent* und *Kategorie*.

7.5 Zusammenfassung

In diesem Kapitel haben Sie gelernt, wie man mithilfe von `CASE WHEN` komplexere `SELECT`-Abfragen entwerfen kann. Dasselbe gilt für Unterabfragen, die entweder mit den Operatoren `IN` oder `EXISTS` verbunden werden oder aber nur einen einzigen Wert zurückliefern dürfen.

Auch für die üblichen Datenänderungsanweisungen `INSERT`, `UPDATE` und `DELETE` ist die Nutzung von `CASE WHEN` und Unterabfragen möglich. Dazu können Sie aber auch direkt mehrere Tabellen nutzen, wenn eine entsprechende `FROM`-Klausel angegeben wird.

Zu guter Letzt haben Sie den `MERGE`-Befehl kennengelernt, mit dem sich zwei Tabellen abgleichen lassen, ohne dafür separate `INSERT`-, `UPDATE`- und `DELETE`-Anweisungen ausführen zu müssen.

Kapitel 8

SQL-Skripts

In diesem Kapitel lernen Sie

- wie Sie SQL-Skripts nutzen können, um komplexere Aufgabenstellungen zu lösen
- die Nutzung von Variablen, um Skripts flexibler zu gestalten
- wie Sie mithilfe von Fallunterscheidungen und Schleifen den Ablauf eines Skripts steuern können
- was es mit Sperren, Transaktionen und Deadlocks auf sich hat

8.1 Arbeiten mit SQL-Skripts

Bisher haben wir nur mit einzelnen SQL-Anweisungen gearbeitet. Für komplexere Aufgabenstellungen reicht dies allerdings oft nicht aus. Dafür bietet SQL Server die Möglichkeit, ganze Skripts von SQL-Anweisungen – quasi SQL-Programme – zu schreiben, um diese dann in einem Schritt auszuführen.

SQL-Skripts werden – wie einzelne SQL-Anweisungen auch – im Abfrage-Editor des SQL Server Management Studios entworfen, mit dem man sie auch speichern und wieder laden kann. Als Dateiendung wird dabei *.SQL* verwendet. Alternativ kann man SQL-Skripts auch mit jedem beliebigen Texteditor – wie beispielsweise Notepad – schreiben und mit der Dateiendung *.SQL* speichern.

Zur Ausführung von SQL-Skripts gibt es vor allem zwei Varianten:

- Das SQL Server Management Studio bietet den Vorteil, dass man die Skripts hier sowohl schreiben als auch direkt ausführen kann und auch gleich das Ergebnis angezeigt bekommt.
- Mit dem Kommandozeilentool SQLCMD lassen sich sowohl einzelne SQL-Anweisungen als auch ganze SQL-Skripts ausführen. Dabei muss allerdings neben der SQL-Anweisung bzw. der SQL-Skriptdatei auch die Verbindung zur entsprechenden Datenbank mit als Parameter übergeben werden. Diese Variante macht allerdings nur für DDL- und DML-Anweisungen Sinn, da sie für `SELECT`-Abfragen keine Möglichkeit bietet, das Ergebnis anzuzeigen.

Hinweis: Zusätzliche Möglichkeiten mit den größeren Editionen von SQL Server

Die größeren Editionen des SQL Server bieten noch ein paar weitere Möglichkeiten zur Ausführung von SQL-Skripts an. So können diese als Bestandteil von SQL Server Agent-Jobs zeitgesteuert ausgeführt werden oder alternativ auch in SQL Server Integration Services-Pakete integriert werden.

In den meisten Fällen werden Sie jedoch mit dem SQL Server Management Studio arbeiten, zumal dieses inzwischen auch die Möglichkeit bietet, SQL-Skripts zu debuggen (also schrittweise auszuführen). Diese Funktionalität wird weiter hinten in diesem Kapitel genauer erläutert.

Im Zusammenspiel mit SQL-Skripts gibt es außerdem noch ein paar Besonderheiten zu beachten:

SQL-Anweisungen können bekanntlich generell mehrzeilig sein. Um nun eindeutig zu kennzeichnen, dass eine Anweisung zu Ende ist und in der nächsten Zeile auch eine neue Anweisung beginnt, gibt es bei T-SQL mittlerweile die Möglichkeit, die Anweisungen mit einem Semikolon abzuschließen. Bis auf wenige Ausnahmen (wie beispielsweise der `MERGE`-Befehl) ist die Verwendung des Semikolons allerdings optional.

Wenn Sie mehrere Anweisungen in einem Schwung ausführen, wird der SQL Server Datenbankobjekte, die innerhalb des SQL-Skripts erstellt wurden, nicht automatisch für folgende Anweisungen verfügbar haben. Die folgende Kombination von Anweisungen würde also zu einer Fehlermeldung führen, da die Sicht beim unteren `SELECT` noch nicht bekannt ist:

```
CREATE VIEW VW_NeueBuecher AS
SELECT * FROM dbo.Buch WHERE Erscheinungsjahr > 2008

SELECT * FROM VW_NeueBuecher
```

Als Lösung hierfür können Sie nach der entsprechenden `CREATE`-Anweisung den `GO`-Befehl einfügen, der dafür sorgt, dass die Änderungen verarbeitet werden, bevor mit der nächsten Anweisung fortgefahren wird. Damit wird sichergestellt, dass die neu erstellten Tabellen, Sichten etc. auch in weiteren Befehlen verwendbar sind:

```
CREATE VIEW VW_NeueBuecher AS
SELECT * FROM dbo.Buch WHERE Erscheinungsjahr > 2008
GO

SELECT * FROM VW_NeueBuecher
```

Genauso werden Sie eine Fehlermeldung erhalten, wenn irgendwo in der Mitte eines SQL-Skripts eine `CREATE`-Anweisung auftaucht, da diese immer am Anfang eines SQL-Batches stehen müssen. Auch dieses Problem können Sie lösen, indem Sie vor die entsprechende `CREATE`-Anweisung einen `GO`-Befehl einfügen, der dafür sorgt, dass nach der `GO`-Anweisung quasi ein neuer SQL-Batch beginnt.

Die Kombination

```
SELECT * FROM dbo.Buch

CREATE VIEW VW_NeueBuecher AS
SELECT * FROM dbo.Buch WHERE Erscheinungsjahr > 2008
```

führt also zu einer Fehlermeldung; mit einem `GO` dazwischen funktioniert es dann:

```
SELECT * FROM dbo.Buch
GO

CREATE VIEW VW_NeueBuecher AS
SELECT * FROM dbo.Buch WHERE Erscheinungsjahr > 2008
```

8.2 Variablen

Wie in fast jeder Programmiersprache, kann man auch in T-SQL Variablen deklarieren, um Daten für eine spätere Verwendung in derselben Sitzung zwischenzuspeichern. Vor der ersten Verwendung müssen Variablen deklariert werden. Dazu wird das Schlüsselwort `DECLARE` gefolgt vom Variablennamen und

dem Datentyp verwendet (zwischen Variablenname und Typ kann optional auch das Schlüsselwort `AS` stehen, was zwar unnötig ist, die Anweisung aber etwas besser lesbar macht). Der Variablenname ist im Prinzip frei wählbar, muss jedoch mit einem @-Zeichen beginnen. Als Datentyp sind alle Typen verwendbar, die Sie auch für die Definition von Tabellenspalten nutzen können (also `int`, `char(n)`, `varchar(n)`, `datetime` etc.)

Für die Zuweisung von Variablenwerten gibt es zwei gängige Methoden. Die erste ist die `SET`-Anweisung, die vor allem genutzt wird, um einer Variablen einen konstanten oder berechneten Wert zuzuweisen.

Um beispielsweise eine Variable `@CDAnzahl` zu deklarieren und dieser einen konstanten Wert zuzuweisen, der anschließend mit einer `SELECT`-Anweisung wieder ausgegeben wird, können Sie folgende Anweisungen verwenden:

```
DECLARE @CDAnzahl AS int
SET @CDAnzahl = 0
SELECT @CDAnzahl
```

Die zweite Methode, einer Variablen einen Wert zuzuweisen, wird verwendet, wenn der Wert aus einer SQL-Abfrage kommt. In diesem Fall können Sie die Zuweisung direkt in der Abfrage vornehmen:

```
DECLARE @CDAnzahl AS int
SELECT @CDAnzahl = COUNT(*) FROM dbo.CD
```

In diesem Fall wird von der `SELECT`-Anweisung noch keine Ausgabe erzeugt, da das Ergebnis quasi in die Variable umgeleitet wird. Zur Anzeige muss noch eine weitere `SELECT`-Abfrage auf die Variable ergänzt werden:

```
DECLARE @CDAnzahl AS int
SELECT @CDAnzahl = COUNT(*) FROM dbo.CD
SELECT @CDAnzahl AS AnzahlCDs
```

Systemvariablen

Neben der Möglichkeit, eigene Variablen zu definieren, stellt SQL Server auch eine Reihe von Systemvariablen zur Verfügung, die man beispielsweise in eigenen Abfragen verwenden kann.

Systemvariablen sind an dem doppelten @ als Präfix erkennbar. Probieren Sie einmal folgende Abfrage im SQL Server Management Studio aus:

```
SELECT @@VERSION AS Version, @@LANGUAGE AS Sprache, @@SERVERNAME AS SQLServer
```

Die Variable `@@VERSION` gibt die genaue Version des benutzten SQL Servers (inklusive Build-Nummer und Service Pack) zurück, `@@LANGUAGE` zeigt die verwendete Sprache an und `@@SQLSERVERNAME` schließlich liefert den Namen der SQL Server-Instanz.

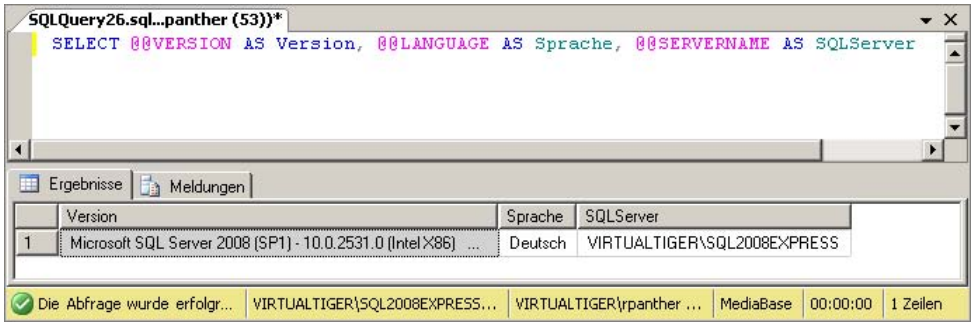


Abbildung 8.1: Abfrage einiger Systemvariablen

Eine vollständige Auflistung der verfügbaren Systemvariablen finden Sie im Anhang A.6, *Systemobjekte*.

Tabellenvariablen und temporäre Tabellen

Zusätzlich zu den bereits behandelten Datentypen gibt es noch zwei weitere, die nicht für Tabellenspalten, sondern ausschließlich für Variablen verwendet werden können. Das ist einerseits der `CURSOR`, der im nächsten Kapitel behandelt wird, und andererseits der Typ `TABLE`, mit dem ganze Tabellen in Variablen abgelegt werden können. Dazu müssen Sie bei der Deklaration der Tabellenvariablen die darin enthaltenen Felder mit angeben. Anschließend lässt sich die Tabellenvariable wie eine normale Tabelle nutzen:

```
DECLARE @CDTitel AS TABLE (ID int, Titel varchar(80))
INSERT INTO @CDTitel SELECT ID, Titel FROM dbo.CD
SELECT * FROM @CDTitel
```

Tabellenvariablen werden – wie andere Variablen auch – im Hauptspeicher des Servers gehalten und nur dann, wenn diese zu groß werden, in der *tempdb* abgelegt. Dies geschieht jedoch transparent, sodass Sie sich nicht selbst darum kümmern müssen.

Hintergrundinfo: Lebensdauer von Variablen

Variablen bleiben innerhalb einer Session erhalten. Das hat zur Folge, dass Sie eine Fehlermeldung erhalten, wenn Sie zuerst die Variable deklarieren, diese Anweisung einzeln ausführen und anschließend versuchen, die Variable zu verwenden. Obwohl Sie sich noch im selben Abfragefenster und damit in derselben Datenbankverbindung befinden, wird für jede Ausführung – auch wenn Sie sich nur auf einen Teil des SQL-Skripts bezieht – eine neue Session geöffnet und anschließend wieder geschlossen.

Um dem Problem der kurzen Lebensdauer von Tabellenvariablen zu begegnen, stellt SQL Server noch eine alternative Möglichkeit zur Verfügung: die temporären Tabellen. Diese werden automatisch in der *tempdb* gespeichert und müssen auch nicht wie Tabellenvariablen deklariert werden. Stattdessen erzeugt man sie meist mit einer `SELECT INTO`-Anweisung auf Basis einer SQL-Abfrage oder über die `CREATE TABLE`-Anweisung[1], mit der man auch normale Tabellen erstellen kann.

[1] Die `CREATE TABLE`-Anweisung wird in Kapitel 10 *Datenbankadministration mit SQL* behandelt.

Temporäre Tabellen gibt es in zwei verschiedenen Ausprägungen: Lokale temporäre Tabellen sind durch das Präfix # im Tabellennamen gekennzeichnet. Sie bleiben während einer Datenbankverbindung erhalten, sind aber für andere Verbindungen nicht zugreifbar. Globale temporäre Tabellen erhalten als Präfix ## für den Tabellennamen und sind auch für andere Verbindungen zugreifbar. Sie werden erst dann wieder entfernt, wenn die letzte Verbindung, die auf die Tabelle zugegriffen hat, geschlossen wird.

Hintergrundinfo: Tabellen direkt in der tempdb anlegen

Alternativ zu den gerade beschriebenen temporären Tabellen kann man auch Tabellen explizit in der *tempdb* anlegen. Hier ist dann kein Präfix für den Tabellennamen nötig und die Tabellen bleiben so lange erhalten, bis der Datenbankdienst gestoppt und neu gestartet wird (da dadurch die gesamte *tempdb* neu aufgebaut wird).

Nun ein kleines Beispiel, um die Unterschiede der verschiedenen Varianten von temporären Tabellen zu verdeutlichen:

1. Öffnen Sie im SQL Server Management Studio ein Abfragefenster und verbinden Sie sich mit der lokalen SQL Server-Instanz und der Datenbank *MediaBase* (z.B. durch die Anweisung `use MediaBase`).
2. Erstellen Sie eine lokale temporäre Tabelle, welche die Titel aller gespeicherten CDs enthält:

```
SELECT ID, Titel
INTO #CDTitel
FROM dbo.CD
```

3. Erstellen Sie mit derselben Abfrage eine globale temporäre Tabelle, indem Sie das Präfix des Tabellennamens anpassen:

```
SELECT ID, Titel
INTO ##CDTitel
FROM dbo.CD
```

4. Erstellen Sie mit derselben Abfrage eine Tabelle in der *tempdb*, indem Sie den Tabellennamen anpassen und um den Datenbank- und Schemanamen erweitern:

```
SELECT ID, Titel
INTO tempdb.dbo.CDTitel
FROM dbo.CD
```

5. Fragen Sie alle drei Varianten von temporären Tabellen ab und Sie erhalten drei Mal alle CDTitel aufgelistet:

```
SELECT * FROM #CDTitel
SELECT * FROM ##CDTitel
SELECT * FROM tempdb.dbo.CDTitel
```

6. Öffnen Sie ein neues Abfragefenster und führen Sie die drei `SELECT`-Anweisungen zur Anzeige der CDTitel einzeln erneut aus.
7. Bei der Abfrage auf die Tabelle *#CDTitel* erhalten Sie die Fehlermeldung *Ungültiger Objektname '#CDTitel'*, da diese lokale temporäre Tabelle für andere Verbindungen nicht sichtbar ist.
8. Schließen Sie nun alle Abfragefenster und öffnen Sie danach ein neues Abfragefenster, in dem Sie wieder alle drei SQL `SELECT`-Anweisungen einzeln ausführen.

9. Nun erhalten Sie sowohl bei der lokalen als auch bei der globalen temporären Tabelle die Fehlermeldung, da auch die globale temporäre Tabelle gelöscht wurde, als die letzte offene Verbindung, die diese Tabelle genutzt hat, geschlossen wurde.
10. Die Abfrage auf die Tabelle *CDTitel* in der Datenbank *tempdb* funktioniert allerdings immer noch. Diese Tabelle wird erst nach einem Neustart des SQL Server-Dienstes automatisch entfernt.

Best Practices: Verwendung von Tabellenvariablen und temporären Tabellen

Sowohl Tabellenvariablen als auch temporäre Tabellen werden meist dazu genutzt, in komplexeren SQL-Skripts Zwischenergebnisse zu speichern und dadurch umfangreiche Abfragen in mehrere kleine Abfragen aufteilen zu können. Das bringt insbesondere dann einen signifikanten Vorteil, wenn ein so gespeichertes Zwischenergebnis von mehreren Abfragen genutzt werden kann.

8.3 Fallunterscheidungen und Schleifen

Auch Fallunterscheidungen und Schleifen sind zentrale Bestandteile fast jeder Programmiersprache, da sich mit diesen Konstrukten auch nicht lineare Abläufe gestalten lassen.

Fallunterscheidung mit IF

Mit der `CASE WHEN`-Anweisung haben Sie bereits ein Konstrukt kennengelernt, mit dem eine Fallunterscheidung realisierbar ist, allerdings ist diese nur innerhalb einer SQL-Abfrage verwendbar. Wenn es darum geht, zu unterscheiden, welche Anweisungen überhaupt ausgeführt werden, ist – wie in den meisten Programmiersprachen – die `IF`-Anweisung das Mittel der Wahl.

```
IF (SELECT COUNT(*) FROM dbo.Buch) > 10
    PRINT 'Viele Bücher, nur lesen muss man sie auch! '
ELSE
    PRINT 'Der Trend geht zum Zweitbuch!'
```

Bei der Gelegenheit habe ich auch zum ersten Mal in diesem Buch die `PRINT`-Anweisung verwendet, die nichts anderes macht, als den angegebenen Text im Ergebnisbereich auszugeben.

Hinweis: Gemeinsamkeiten und Unterschiede zwischen SELECT und PRINT

Beide Anweisungen kann man nutzen, um Werte anzuzeigen. Allerdings kann die `PRINT`-Anweisung nicht direkt auf Tabellen zugreifen und gibt das Ergebnis direkt im Meldungsfenster aus, während die `SELECT`-Anweisung auch einen konstanten Meldungstext wie eine Tabelle (mit einer Zeile und Spalte) interpretiert und diesen im Ergebnisbereich ausgibt. Im Meldungsfenster steht dann stattdessen:

(1 Zeile(n) betroffen)

Anweisungsblöcke mit BEGIN ... END

Bei den oben gezeigten Beispielen beinhaltete sowohl der IF-Zweig als auch der ELSE-Zweig jeweils nur eine einzige SQL-Anweisung. Wenn hier einmal mehrere Anweisungen benötigt werden, sind diese in die Schlüsselwörter BEGIN und END einzufassen, wodurch ein Anweisungsblock definiert wird.

```
IF (SELECT COUNT(*) FROM dbo.Buch) > 10
    BEGIN
        PRINT 'Viele Bücher, nur lesen muss man sie auch! '
        PRINT '(vielleicht hilft es aber auch, sie unter das Kopfkissen zu legen)'
    END
ELSE
    BEGIN
        PRINT 'Der Trend geht zum Zweitbuch!'
        PRINT '... manch einer hat sogar drei!'
    END
```

Anweisungsblöcke werden auch für Funktionen und gespeicherte Prozeduren verwendet, die im nächsten Kapitel behandelt werden.

WHILE-Schleifen

Ein weiteres wesentliches Konstrukt einer jeden Programmiersprache sind die Schleifen. Mit diesen kann eine Wiederholung einer Anweisung (oder eines Blocks aus mehreren Anweisungen) erreicht werden.

Das folgende Beispiel soll dies verdeutlichen:

1. Öffnen Sie im SQL Server Management Studio ein Abfragefenster und verbinden Sie sich mit der lokalen SQL Server-Instanz und der Datenbank *MediaBase* (z.B. durch die Anweisung `use MediaBase`).
2. Führen Sie folgende SQL-Anweisungen aus, um der Tabelle *dbo.Buch* so lange leere Einträge hinzuzufügen, bis diese 100 Zeilen enthält.

   ```
   WHILE (SELECT COUNT(*) FROM dbo.Buch) < 100
       INSERT INTO dbo.Buch (Titel) VALUES (NULL)
   ```

3. Überprüfen Sie das Ergebnis durch Eingabe der folgenden Abfrage:

   ```
   SELECT * FROM dbo.Buch
   ```

4. Entfernen Sie die hinzugefügten Zeilen nun wieder, indem Sie die folgende Anweisung ausführen:

   ```
   DELETE FROM dbo.Buch WHERE Titel IS NULL
   ```

Natürlich lasen sich auch beim WHILE-Konstrukt ganze Blöcke von Anweisungen wiederholen, die in BEGIN und END eingefasst sind. In diesem Zusammenhang bietet WHILE aber noch zwei Zusatzanweisungen, mit denen man die Codeausführung weiter beeinflussen kann.

- BREAK beendet die Schleife und fährt mit den Anweisungen danach fort
- CONTINUE überspringt die folgenden Anweisungen der Schleife und fährt mit dem nächsten Schleifendurchlauf fort (sofern die Schleifenbedingung immer noch erfüllt ist)

Probieren Sie hierzu das folgende Beispielskript einmal aus:

```
DECLARE @Zaehler int = 0
WHILE @Zaehler < 20
    BEGIN
        SET @Zaehler = @Zaehler+1
        IF @Zaehler = 13
            CONTINUE
        PRINT @Zaehler
    END
```

Zu Beginn wird eine Variable `@Zaehler` deklariert und auf 0 gesetzt. Anschließend wird die Schleife wiederholt, solange der Wert der Variablen kleiner als 20 ist. Innerhalb der Schleife wird `@Zaehler` um eins erhöht und mit einer `PRINT`-Anweisung ausgegeben. Lediglich dann, wenn die Zahl 13 erreicht ist, wird die `PRINT`-Anweisung übersprungen (quasi die Ausnahme für Abergläubische). Wenn Sie das Skript starten, werden demnach die Zahlen 1 bis 20 – allerdings ohne die Zahl 13 – ausgegeben.

Wenn Sie nun die `CONTINUE`-Anweisung durch `BREAK` ersetzen, wird die Schleife bei der Zahl 13 komplett verlassen (bevor die Zahl angezeigt wird), sodass nur die Zahlen von 1 bis 12 ausgegeben werden.

8.4 Debuggen von SQL-Skripts

Inzwischen bietet das SQL Server Management Studio auch die Möglichkeit, SQL-Skripts zu debuggen, um beispielsweise zu prüfen, welcher Weg bei einer `IF`-Abfrage eingeschlagen wird. Dabei können sowohl die Skripts selbst, als auch von Ihnen benutzte Funktionen und gespeicherte Prozeduren debuggt werden.

Das Debuggen selbst wird über den Menüpunkt *Debuggen/Debuggen starten* oder den kleinen grünen Pfeil in der Symbolleiste (direkt neben dem *Ausführen*-Symbol) gestartet. Während des Debuggens werden Objekt-Explorer und registrierte Server automatisch ausgeblendet. Dafür werden im unteren Bereich des SQL Server Management Studio zwei neue Bereiche mit den Überschriften *Lokal* und *Aufrufliste* eingeblendet (dazu gleich mehr). Außerdem wird eine zusätzliche Symbolleiste eingeblendet, mit der man die wichtigsten Aktionen für das Debuggen schnell erreichen kann.

Abbildung 8.2: Die *Debuggen*-Symbolleiste

Die Optionen der *Debuggen*-Symbolleiste finden sich auch alle im *Debuggen*-Menü wieder (wo auch die entsprechende Tastenkombination zu sehen ist, mit der man die Aktion alternativ ausführen kann). Im Einzelnen sind dies folgende:

- *Debuggen starten/weiter* – Starten des Debug-Modus bzw. mit der Skriptausführung im Debug-Modus fortfahren
- *Alle unterbrechen* – Anhalten der Skriptausführung im Debug-Modus
- *Debuggen beenden* – Abbrechen von Skriptausführung und Debug-Modus
- *Nächste Anweisung anzeigen* – Anzeige der nächsten Anweisung, die ausgeführt würde, allerdings ohne diese auszuführen

- *Einzelschritt* – Ausführen der nächsten Anweisung; ist dies eine Funktion oder gespeicherte Prozedur, springt der Debugger in den Quelltext derselben
- *Prozedurschritt* – Ausführen der nächsten Anweisung; ist dies eine Funktion oder gespeicherte Prozedur, wird diese komplett ausgeführt, ohne dass deren Quelltext angezeigt wird
- *Ausführen bis Rücksprung* – Fortführen der Skriptausführung bis zum Rücksprung aus einer Funktion bzw. Prozedur; befindet sich die Skriptausführung gerade nicht in einer Funktion oder Prozedur, wird das Skript bis zum Ende ausgeführt
- *Haltepunkte* – Anzeige von Haltepunkten in einem separaten Fenster im unteren Bereich

Schrittweise Ausführung

Spielen wir nun gemeinsam einmal ein Beispiel durch, das die schrittweise Ausführung von SQL-Skripten und dabei verfügbaren Möglichkeiten demonstriert:

1. Zuerst benötigen Sie natürlich ein SQL-Skript. Geben Sie daher folgenden Text in einem Abfragefenster ein, das mit dem lokalen SQL Server verbunden ist:

```
DECLARE @Zaehler AS int

SET @Zaehler = 0

WHILE @Zaehler < 10
BEGIN
    SET @Zaehler = @Zaehler+1
 PRIN   T @Zaehler
END
```

2. Wenn Sie das Skript nun auf dem üblichen Weg ausführen, bekommen Sie die Zahlen von 1 bis 10 im Ergebnisbereich angezeigt.
3. Starten Sie nun den Debug-Modus durch Anklicken des entsprechenden Symbols oder Auswahl des entsprechenden Menüpunkts im *Debuggen*-Menü. Die Anzeige wechselt in den Debug-Modus und die erste Anweisung des Skripts (die Deklaration der Variablen `@Zaehler`) wird ausgeführt.

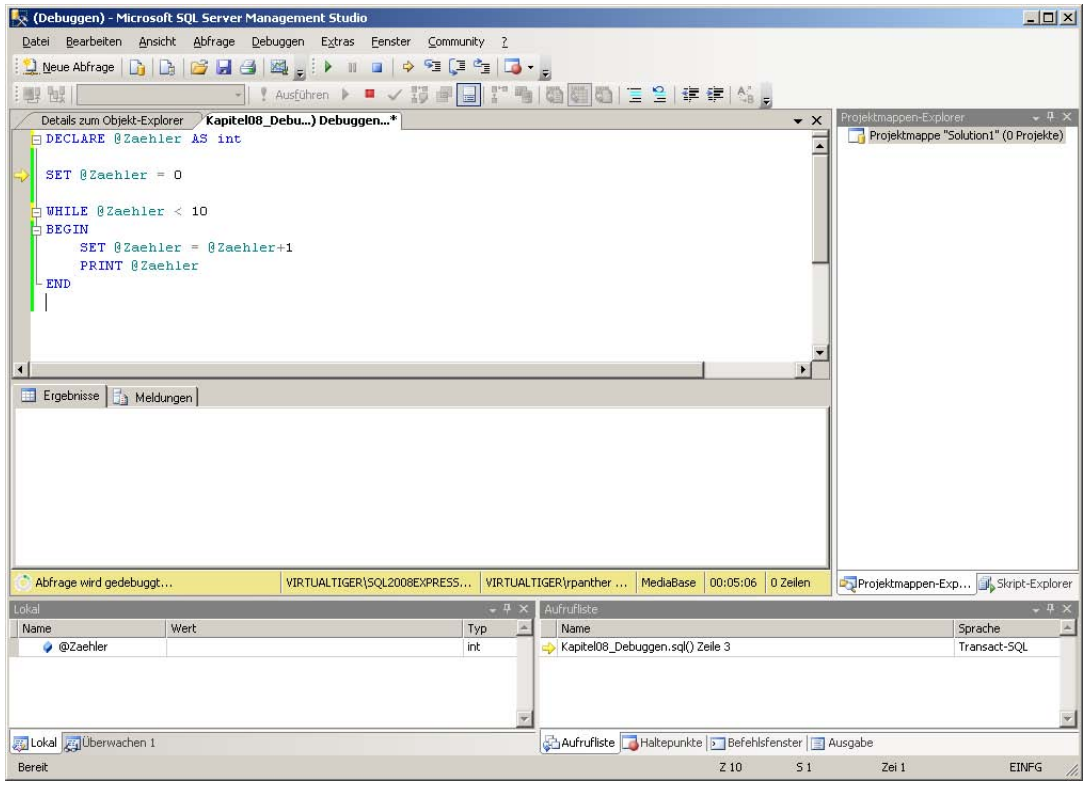


Abbildung 8.3: SQL Server Management Studio im Debug-Modus

Im Debug-Modus markiert der kleine gelbe Pfeil die SQL-Anweisung, die als Nächstes ausgeführt wird. Im unteren Bereich ist auf der linken Seite eine Auflistung aller deklarierten lokalen Variablen mit ihrem Typ und Wert zu sehen (da die Variable `@Zaehler` gerade erst deklariert wurde, hat sie im Screenshot in Abbildung 8.3 noch keinen Wert bekommen). Auf der rechten Seite ist die Aufrufliste zu sehen, die darüber informiert, in welchem Skript (Dateiname, Prozedur- oder Funktionsname etc.) und welcher Zeile sich die Ausführung gerade befindet.

4. Wenn Sie nun mehrfach die Taste F11 drücken (was dem Einzelschritt entspricht), können Sie sehen, wie der gelbe Pfeil wandert und die Variable zuerst den Wert 0 zugewiesen bekommt, der dann bei jedem Schleifendurchlauf erhöht wird.
5. Wenn Sie nun ein umfangreiches Skript mit so vielen Variablen debuggen, dass die Liste unter *Lokal* zu unübersichtlich wird, gibt es zwei Möglichkeiten, die wir nun beide ausprobieren. Mit einem Klick mit der rechten Maustaste auf einen Variablennamen im Skript und Auswahl der Option *Schnellüberwachung* im dann erscheinenden Kontextmenü, erscheint ein separates Dialogfeld, das Ihnen den Typ und aktuellen Wert der Variablen anzeigt.

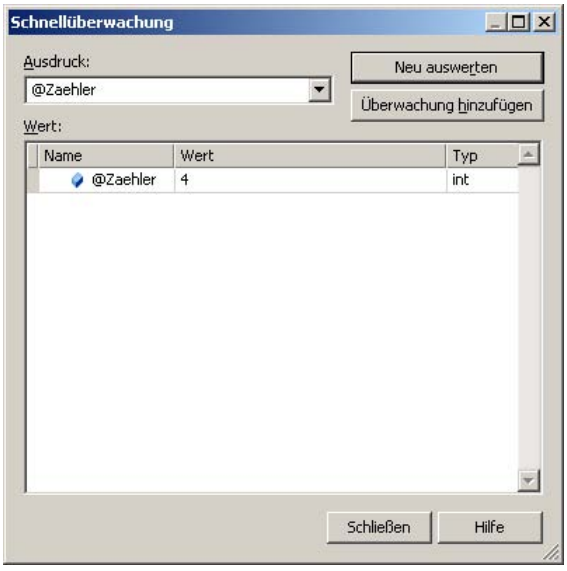


Abbildung 8.4: Das Dialogfeld *Schnellüberwachung*

6. Klicken Sie hier auf die Schaltfläche *Überwachung hinzufügen*, wird die Variable auch zur Liste *Überwachen 1* hinzugefügt, die sich auf einer Registerkarte hinter *Lokal* verbirgt und nun automatisch in den Vordergrund gestellt wird. Schließen Sie nun das Dialogfeld *Schnellüberwachung* wieder.
7. Um Überwachungen hinzuzufügen, können Sie alternativ auch sowohl im Skript selbst als auch in der Liste *Lokal* eine Variable mit der rechten Maustaste anklicken und im darauf erscheinenden Kontextmenü den Befehl *Überwachung hinzufügen* auswählen. Somit können Sie sich bei vielen Variablen hier eine Art Favoritenliste zusammenstellen, in denen die wichtigsten Variablen während der Skriptausführung im Debug-Modus auf einen Blick sichtbar sind.
8. Klicken Sie nun auf das Symbol *Weiter* in der *Debuggen*-Symbolleiste, um den Rest des Skripts bis zum Ende auszuführen.

Breakpoints (Haltepunkte) nutzen

Bei langen SQL-Skripts kann es natürlich sehr lästig werden, diese zeilenweise bis zur kritischen Stelle auszuführen, die eigentlich überprüft werden soll. Um diesem Dilemma abzuhelfen, gibt es sogenannte Haltepunkte (Breakpoints). Ein Breakpoint kann an beliebiger Stelle im SQL-Skript gesetzt werden, indem man entweder mit der Maus auf den grauen Bereich links von der Zeile klickt oder aber mit der rechten Maustaste auf die Zeile klickt und im Kontextmenü die Option *Haltepunkt/Haltepunkt einfügen* auswählt. Auf dieselbe Weise lassen sich auch Haltepunkte wieder entfernen oder temporär deaktivieren (bzw. später wieder aktivieren).

1. Setzen Sie einen Breakpoint in der Zeile mit der `PRINT`-Anweisung.
2. Starten Sie den Debug-Modus durch Anklicken des entsprechenden Symbols oder Auswahl des entsprechenden Menüpunkts im *Debuggen*-Menü.

3. Klicken Sie nun auf das Symbol *Weiter* und die Ausführung des Skripts wird bis zu der Zeile fortgeführt, in der der Breakpoint gesetzt wurde.
4. Klicken Sie noch einmal auf das Symbol *Weiter* und das Skript wird an derselben Stelle angehalten, nachdem die Schleife einmal durchlaufen wurde.
5. Hinter der bereits erwähnten *Aufrufliste* befinden sich weitere Registerkarten. Holen Sie nun die mit *Haltepunkte* bezeichnete Registerkarte in den Vordergrund und Sie sehen dort alle Haltepunkte (momentan also nur einen) aufgelistet.
6. In der Liste *Haltepunkte* können Sie Haltepunkte ebenfalls entfernen oder temporär deaktivieren. Außerdem wird hier angezeigt, wie oft der Haltepunkt schon durchlaufen wurde. Dazu gibt es noch die Möglichkeit, im Skript an die Stelle zu springen, wo der Haltepunkt definiert wurde.
7. Um nicht alle zehn Schleifendurchläufe mit einem separaten Klick bestätigen zu müssen, deaktivieren Sie die Haltepunkte nun durch Entfernen des Häkchens und fahren im Skript durch Anklicken des Symbols *Weiter* in der Symbolleiste fort.
8. Wenn das Skript komplett ausgeführt wurde, können Sie den Haltepunkt wieder komplett entfernen.

8.5 Fehlerbehandlung in SQL-Skripts

Mit den gerade behandelten Methoden zum Debuggen von Skripts können Sie Fehlern Schritt für Schritt manuell auf die Schliche kommen. Aber gerade in Skripts, die künftig häufig – und mit verschiedenen Daten – ausgeführt werden, ist es unmöglich, alle möglichen Fehler vorherzusehen oder gar im Vorfeld zu beseitigen. Um für einen kontrollierten Umgang mit Fehlern zu sorgen, bietet SQL Server zwei grundlegende Möglichkeiten:

- Fehler generieren mit `RAISERROR`
- Fehler abfangen mit `TRY...CATCH`

RAISERROR

Sie haben sicherlich schon versehentlich verschiedene Fehlermeldungen im Abfrage-Editor des SQL Server Management Studio erzeugt. Wenn Sie sich beispielsweise bei einem Tabellennamen vertippen und daher die folgende Abfrage eingeben

```
SELECT * FROM dbo.Huch
```

erscheint eine Fehlermeldung ähnlich der folgenden:

Meldung 208, Ebene 16, Status 1, Zeile 1
Ungültiger Objektname 'dbo.Huch'.

Dabei ist die Meldungsnummer 208 eine eindeutige ID für allgemeine Fehler (in diesem Fall für die Meldung *Ungültiger Objektname*). Die Ebene (in diesem Fall 16) steht für den Schweregrad des Fehlers und die 1 kennzeichnet den Status. Danach folgen die Zeilennummer im Skript, an welcher der Fehler auftritt, sowie die Fehlermeldung noch einmal im Klartext.

Die Ebene (der Schweregrad) des Fehlers liegt zwischen 0 und 25, wobei die höheren Nummern auch die schwerwiegenderen Fehler bezeichnen. Fehlerebenen ab 20 werden als fatale Fehler eingestuft, nach denen die Client-Verbindung automatisch unterbrochen wird. Der Status kann zwischen 0 und 255 liegen und kann benutzt werden, um gleiche Fehler, die an unterschiedlichen Servern auftreten, voneinander zu unterscheiden.

Mit der `RAISERROR`-Anweisung können Sie eine solche Fehlermeldung nun explizit erzeugen. Dabei werden sowohl die Fehlermeldung als auch Ebene und Status als Parameter übergeben. Die oben dargestellte Fehlermeldung lässt sich beispielsweise nahezu identisch künstlich erzeugen, wenn Sie folgendes SQL-Kommando ausführen:

```
RAISERROR ('Ungültiger Objektname dbo.Huch', 16, 1)
```

In der Fehlermeldung, die Sie dadurch erhalten, ist ein wesentlicher Unterschied zu sehen:

Die Nummer der Fehlermeldung ist nicht 208, sondern 50000. Das liegt daran, dass Sie den Fehlermeldungstext direkt als Parameter übergeben haben. Damit handelt es sich um einen individuellen Fehler, der automatisch die Fehlermeldungsnummer 50000 bekommt.

Alternativ zum Fehlermeldungstext könnten Sie auch die Fehlernummer übergeben, wobei die Standard-Fehlermeldungsnummern hier ausgeschlossen sind. Stattdessen können Sie eine eigene Fehlermeldung speichern und dieser eine Nummer zwischen 13.000 und 2.147.483.647 geben, die aber ungleich 50.000 sein muss. Aus diesem Grund werden meist nur Nummern ab 50.001 für eigene Fehlermeldungen verwendet. Zum Erzeugen von eigenen Fehlermeldungen kann man die Systemprozedur[2] `sp_addmessage` verwenden, die als Parameter die Fehlernummer, die Ebene und den Fehlertext übergeben bekommt. Dazu kommt als weiterer Parameter noch die Sprache, sodass die Fehlermeldung immer in der richtigen Sprache des Clients ausgegeben werden kann. Dieser Parameter wird benötigt, da die Fehlermeldungstexte immer erst in der US-amerikanischen Variante erfasst werden müssen, bevor andere Sprachvarianten hinzukommen. Führen Sie daher folgende Anweisungen aus, um eine neue Fehlermeldung mit den dazugehörenden Fehlertexten in Englisch und Deutsch zu erstellen:

```
EXEC sp_addmessage 50001, 16, 'Custom Error', us_english
EXEC sp_addmessage 50001, 16, 'Benutzerdefinierter Fehler', german
```

Diese Fehlermeldung können Sie nun auch für die `RAISERROR`-Anweisung nutzen. Der Aufruf

```
RAISERROR (50001, 16, 1)
```

gibt Ihnen die folgende Fehlermeldung zurück:

Meldung 50001, Ebene 16, Status 1, Zeile 1

Benutzerdefinierter Fehler

Wenn Sie die `RAISERROR`-Anweisung von einem englischsprachigen Client aus aufrufen, erhalten Sie automatisch die englischsprachige Fehlermeldung.

Hinweis: Vorhandene Fehlermeldungen anzeigen

Eine Übersicht über alle bereits vorhandenen Fehlermeldungen bekommen Sie, wenn Sie die Systemsicht `sys.messages` öffnen. In dieser Sicht sind auch die von Ihnen selbst erstellten Meldungen enthalten.

[2] Systemprozeduren werden detailliert im nächsten Kapitel behandelt.

TRY ... CATCH

Die zweite Möglichkeit, eine Fehlerbehandlung in SQL-Skripts zu realisieren, ist das Abfangen von Fehlern mit dem `TRY...CATCH`-Konstrukt. Damit können Sie einen Block von SQL-Anweisungen in die Anweisungen `BEGIN TRY` und `END TRY` einfassen. Sollte nun in diesem Bereich ein Fehler auftreten, wird die Ausführung nicht abgebrochen, sondern mit den Anweisungen im `CATCH`-Block (der mit `BEGIN CATCH` und `END CATCH` eingefasst ist) fortgesetzt. Hier kann dann eine individuelle Fehlerbehandlung – wie beispielsweise ein Protokollieren des Fehlers in einer Datenbanktabelle – erfolgen.

Testen wir dies gemeinsam an einem einfachen Beispiel (einer Division durch 0):

1. Führen Sie in einem Abfragefenster die folgende SQL-Anweisung aus:

```
SELECT 1/0
```

2. Sie erhalten die folgende Fehlermeldung:

   Meldung 8134, Ebene 16, Status 1, Zeile 1
   Fehler aufgrund einer Division durch Null.

3. Erweitern Sie das SQL-Skript nun so, dass die `SELECT`-Anweisung in einen `TRY`-Block eingefasst ist:

```
BEGIN TRY
 SELE   CT 1/0
END TRY

BEGIN CATCH
    PRINT 'Division durch 0!'
END CATCH
```

4. Wenn Sie das Skript nun ausführen, erhalten Sie kein Ergebnis, aber im Meldungsfenster steht folgender Text:

   (0 Zeile(n) betroffen)
   Division durch 0!

Für eine effektivere Fehlerbehandlung ist natürlich noch etwas mehr nötig. So gibt es einige Systemfunktionen[3], die Ihnen helfen können, mehr Informationen über den aufgetretenen Fehler zu bekommen. Ergänzen Sie den `CATCH`-Block dazu mit folgender Abfrage (direkt hinter der `PRINT`-Anweisung):

```
SELECT
       ERROR_NUMBER() AS ErrorNumber,
       ERROR_SEVERITY() AS ErrorSeverity,
       ERROR_STATE() as ErrorState,
       ERROR_PROCEDURE() as ErrorProcedure,
       ERROR_LINE() as ErrorLine,
       ERROR_MESSAGE() as ErrorMessage;
```

Fehlernummer (`ErrorNumber`), Schweregrad (`ErrorSeverity`) und Status (`ErrorState`) dürften Ihnen genauso bekannt vorkommen wie die Fehlermeldung (`ErrorMessage`). Dazu wird noch die Zeilennummer (`ErrorLine`) sowie – falls vorhanden – der Name der gespeicherten Prozedur (`ErrorProcedure`) ausgegeben, in welcher der Fehler aufgetreten ist.

---

[3] Systemfunktionen werden detailliert im nächsten Kapitel behandelt. Im Moment reicht es aus zu wissen, dass diese wie Systemvariablen verwendet werden können.

Best Practices: Fehler protokollieren

Es hat sich bewährt, die gerade gezeigten Fehlerinformationen zusammen mit dem Zeitpunkt, an dem der Fehler aufgetreten ist, in einer Fehlerprotokolltabelle zu speichern. Damit können Sie auch im Nachhinein noch nachvollziehen, wann welche Fehler aufgetreten sind.

Alternativ zu den beschriebenen Funktionen erhält auch die Systemvariable `@@ERROR` die Fehlernummer des zuletzt aufgetretenen Fehlers. Allerdings wird diese wieder auf 0 zurückgesetzt, sobald sie einmal abgefragt wurde. Mit den Systemfunktionen können Sie die Fehlerinformationen auch mehrfach abfragen, solange Sie sich innerhalb des `CATCH`-Blocks befinden.

8.6 Sperren, Transaktionen und Deadlocks

Sperren, Transaktionen und Deadlocks sind drei Begriffe, die eng zusammengehören und deren Bedeutung daher am besten auch zusammen zu erklären ist.

Sperren

Insbesondere beim Schreiben von Daten ist es wichtig, dass die entsprechenden Datenobjekte vorher für den Zugriff von anderen Benutzern oder Prozessen gesperrt werden. Dies kann auf Datensatzebene geschehen, auf Speicherseiten- (in Kapitel 4 wurde bereits beschrieben, dass Datenzeilen auf 8 KB großen Speicherseiten angeordnet sind), Tabellen- oder sogar Datenbankebene, sofern eine kritische Anzahl an Elementen der untergeordneten Kategorie gesperrt ist. Wenn also viele Zeilen, die in derselben Speicherseite abgelegt sind, gesperrt werden, wird stattdessen die gesamte Speicherseite gesperrt. Sind viele Speicherseiten zu einer Tabelle gesperrt, wird stattdessen automatisch die ganze Tabelle gesperrt und wenn viele Tabellen gesperrt sind, wird dies in eine komplette Datenbanksperre umgewandelt. Man nennt diesen Vorgang Lock Escalation (Lock = engl. für Sperre).

Dabei wird zwischen verschiedenen Arten von Sperren unterschieden. Die wichtigsten Sperrtypen sind:

- Shared Lock (S) – lesender Zugriff
- Update Lock (U) – Ankündigung eines schreibenden Zugriffs
- Exclusive Lock (X) – schreibender Zugriff

Wie der Name schon andeutet, können gleichzeitig mehrere Shared Locks auf einen Speicherbereich bestehen. Diese verhindern nur einen gleichzeitigen schreibenden Zugriff. Mit einem Update Lock allein kann noch kein Schreibzugriff erfolgen, allerdings wird dieser automatisch in einen Exclusive Lock umgewandelt, sobald alle anderen Sperren freigegeben sind. Der Exclusive Lock schließlich gewährleistet den alleinigen Zugriff auf den Speicherbereich und ermöglicht damit den schreibenden Zugriff.

Transaktionen

Trotz automatischer Lock Escalation kann es oft auch Sinn machen, Datensätze aus verschiedenen Tabellen (und evtl. sogar Datenbanken) explizit gemeinsam zu sperren, um zusammenhängende Änderungen durchführen zu können, bevor ein anderer Prozess oder Benutzer darauf zugreifen kann. Hierfür

gibt es Transaktionen, die dafür sorgen, dass die Sperren auf die beteiligten Datenbankobjekte so lange aufrechterhalten werden, bis die gesamte Transaktion beendet ist. Die Bestätigung einer beendeten Transaktion nennt sich `COMMIT` und bewirkt, dass die Änderungen, die in der Transaktion durchgeführt wurden, dauerhaft gespeichert werden. Alternativ lässt sich auch ein sogenannter `ROLLBACK` ausführen, der ebenfalls die Sperren wieder freigibt, vorher aber alle Änderungen der Transaktion wieder rückgängig macht. Ein `ROLLBACK` wird auch automatisch ausgelöst, falls innerhalb der Transaktion ein Fehler auftritt, oder gar der SQL Server-Dienst beendet wird (im Extremfall auch wenn der Server abstürzt und neu gebootet wird). Damit erhält die Transaktion einen »ganz oder gar nicht«-Charakter, da entweder alle darin enthaltenen Änderungen endgültig ausgeführt werden (bei erfolgreichem `COMMIT`) oder alle begonnenen Änderungen wieder rückgängig gemacht werden (durch einen expliziten oder automatischen `ROLLBACK`).

Hinweis: Implizite Transaktionen

Jede einzelne SQL-Anweisung wird implizit als eine Transaktion ausgeführt. Wenn Sie also beispielsweise alle Datensätze einer Tabelle mit einem `UPDATE` ändern und irgendwo in der Mitte tritt ein Fehler auf, so werden auch die bereits durchgeführten Änderungen automatisch wieder rückgängig gemacht.

Typisches Beispiel für eine Transaktion ist eine Geldüberweisung: Im ersten Schritt wird der Kontostand des ersten Kontos um den Betrag x reduziert, anschließend der Kontostand des zweiten Kontos um den Betrag x erhöht. Findet dazwischen ein Serverausfall statt, so würde das ohne Transaktionen bedeuten, dass der Geldbetrag für beide Konten verloren ist. Bei einem von einer Transaktion automatisch ausgeführten `ROLLBACK` aber, wäre der Betrag wieder auf dem ursprünglichen Konto vorhanden und könnte erneut überwiesen werden.

In T-SQL gibt es drei Anweisungen, mit denen Sie Transaktionen steuern können:

- `BEGIN TRANSACTION` startet eine neue Transaktion.
- `COMMIT TRANSACTION` bestätigt das erfolgreiche Ende der Transaktion und führt die Änderungen endgültig durch.
- `ROLLBACK TRANSACTION` macht alle Änderungen der Transaktion wieder rückgängig.

Führen wir nun ein Beispiel durch, um das Arbeiten mit Transaktionen auszuprobieren:

1. Öffnen Sie im SQL Server Management Studio ein Abfragefenster und verbinden Sie sich mit der lokalen SQL Server-Instanz und der Datenbank *MediaBase* (z.B. durch die Anweisung `use MediaBase`).
2. Lassen Sie mit einer Abfrage ID, Titel und Erscheinungsjahr der Tabelle *dbo.Buch* anzeigen:

   ```
   SELECT ID, Titel, Erscheinungsjahr FROM dbo.Buch
   ```

3. Öffnen Sie nun eine Transaktion, indem Sie die folgende Anweisung ausführen:

   ```
   BEGIN TRANSACTION
   ```

4. Führen Sie anschließend mit den folgenden Anweisungen eine Datenänderung durch und prüfen Sie die Auswirkung der Änderung:

   ```
   UPDATE dbo.Buch SET Erscheinungsjahr = 2010
   SELECT ID, Titel, Erscheinungsjahr FROM dbo.Buch
   ```

5. Brechen Sie nun die Transaktion ab und prüfen Sie anschließend den Inhalt der Tabelle *dbo.Buch:*

```
ROLLBACK TRANSACTION
SELECT ID, Titel, Erscheinungsjahr FROM dbo.Buch
```

6. Die Änderungen sind durch den `ROLLBACK` wieder rückgängig gemacht und die ursprünglichen Daten werden angezeigt. Hätten Sie anstelle des `ROLLBACK` ein `COMMIT TRANSACTION` ausgeführt, wären die Änderungen dauerhaft und unwiderruflich durchgeführt worden.

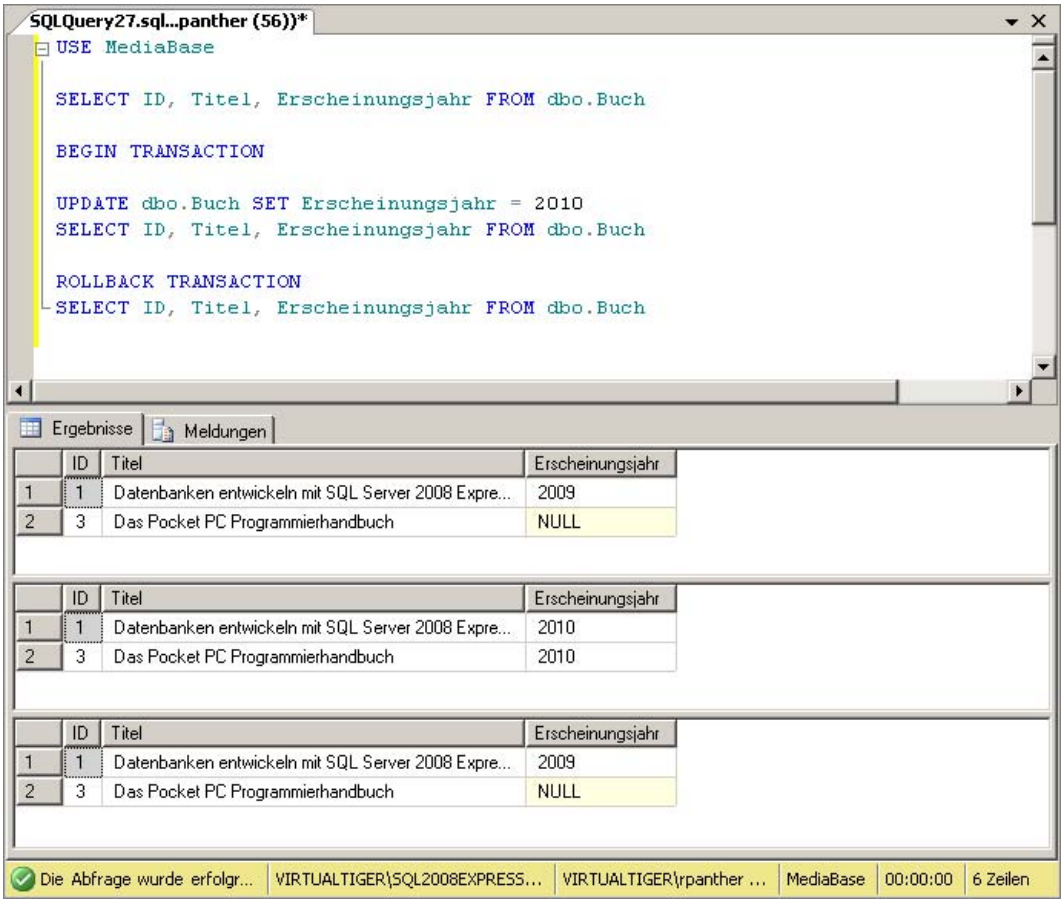


Abbildung 8.5: Die Daten der Tabelle vor, während und nach Abbruch der Transaktion

Best Practices: Transaktionen zum Ausprobieren nutzen

Sie können Transaktionen auch zweckentfremden, um Datenänderungsanweisungen in SQL gefahrlos ausprobieren zu können. Dazu starten Sie mit `BEGIN TRANSACTION` eine neue Transaktion, führen anschließend Ihre `INSERT`-, `UPDATE`- oder `DELETE`-Anweisungen durch und prüfen, ob das Ergebnis Ihren Erwartungen entspricht. Ist dies der Fall, können Sie die Änderungen mit `COMMIT TRANSACTION` bestätigen. Falls sich aber ein Fehler in die entsprechende Anweisung eingeschlichen hat, können Sie die Änderungen mit `ROLLBACK TRANSACTION` rückgängig machen und die Anweisung korrigieren.

Transaktionen können auch geschachtelt werden, was im Prinzip schon bei der ersten expliziten Transaktion geschieht, da jede SQL-Anweisung innerhalb dieser Transaktion wiederum eine (implizite) Transaktion darstellt. Aber auch explizite Transaktionen können ineinander geschachtelt werden, wobei sich jede ROLLBACK- bzw. COMMIT-Anweisung immer auf die zuletzt geöffnete der noch offenen Transaktionen bezieht.

Über die Systemvariable @@TRANCOUNT können Sie die Anzahl der offenen Transaktionen abfragen, wie das in Abbildung 8.6 dargestellte SQL-Skript und dessen Ergebnis zeigen.

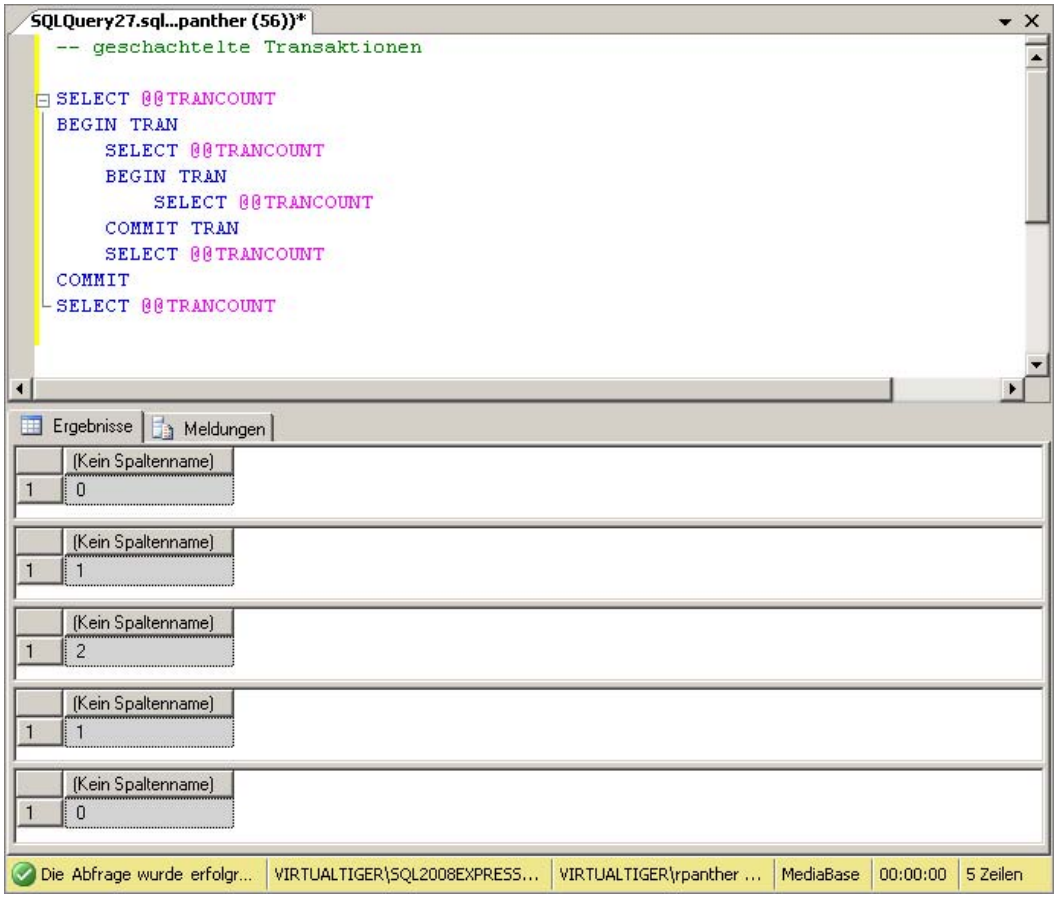


Abbildung 8.6: Geschachtelte Transaktionen und die @@TRANCOUNT-Systemvariable

Sie sehen in dieser Abbildung außerdem, dass sich der Begriff TRANSACTION auch als TRAN abkürzen lässt. Bei COMMIT und ROLLBACK kann man dieses Schlüsselwort sogar ganz weglassen, da sich diese Anweisungen immer auf Transaktionen beziehen.

Deadlocks

So vorteilhaft Sperren und Transaktionen auch sein mögen, bringen sie doch auch ein neues Problem mit sich: die Deadlocks. Dabei handelt es sich um eine Situation, die eintreten kann, wenn zwei Transaktionen (aus unterschiedlichen Prozessen) sich gegenseitig blockieren.

SQL Server erkennt Deadlocks allerdings automatisch und löst sie auf, indem einer der Prozesse zum Deadlock-Opfer erklärt und abgebrochen wird. Dadurch wird für die dazugehörige Transaktion automatisch ein Rollback ausgeführt, während die andere Transaktion problemlos durchlaufen kann, da sie ja nun nicht mehr blockiert wird. Die Entscheidung, welche Transaktion abgebrochen wird, trifft SQL Server primär aufgrund der bereits verursachten Transaktionskosten (damit ist die Kombination aus Prozessorzeit und Speicher gemeint, die von der Transaktion belegt wird). Da ein Rollback in der Regel um ein Vielfaches aufwändiger ist, als die ursprüngliche Ausführung der Anweisungen selbst, wird der kostengünstigere Prozess abgebrochen. Nachdem der teurere Prozess durchgelaufen ist, kann die abgebrochene Transaktion erneut ausgeführt werden. Dies geschieht jedoch nicht automatisch, sondern muss manuell (oder durch eine Anwendung gesteuert) initiiert werden.

Probieren wir auch dies anhand eines einfachen Beispiels aus:

1. Öffnen Sie im SQL Server Management Studio ein Abfragefenster und verbinden Sie sich mit der lokalen SQL Server-Instanz und der Datenbank *MediaBase* (z.B. durch die Anweisung `use MediaBase`).
2. Beginnen Sie hier eine Transaktion und führen Sie eine Änderung auf der Tabelle dbo.Buch aus:

   ```
   BEGIN TRANSACTION
   UPDATE dbo.Buch SET Bewertung = 1
   ```

3. Öffnen Sie ein weiteres Abfragefenster, das ebenfalls mit der lokalen SQL Server-Instanz und der Datenbank *MediaBase* verbunden ist.
4. Beginnen Sie hier ebenfalls eine Transaktion und führen Sie eine Änderung auf der Tabelle *dbo.CD* aus:

   ```
   BEGIN TRANSACTION
   UPDATE dbo.CD SET Bewertung = 2
   ```

5. Wechseln Sie nun zurück zum ersten Abfragefenster und führen Sie hier ebenfalls eine Änderung auf der Tabelle *dbo.CD* aus:

   ```
   UPDATE dbo.CD SET Bewertung = 1
   ```

   Sie werden jetzt keine direkte Bestätigung erhalten, dass das `UPDATE` ausgeführt wurde, da die Verbindung nun versucht, die Tabelle *dbo.CD* exklusiv zu sperren, diese aber von der anderen Verbindung bereits gesperrt ist.
6. Wechseln Sie wieder zurück zum zweiten Abfragefenster und führen Sie hier eine Änderung auf der Tabelle *dbo.Buch* aus:

   ```
   UPDATE dbo.Buch SET Bewertung = 2
   ```

7. Sie haben nun einen Deadlock erzeugt, da die erste Verbindung die Tabelle *dbo.Buch* gesperrt hat und auf Freigabe der Tabelle *dbo.CD* wartet, während die zweite Verbindung die Tabelle *dbo.CD* gesperrt hat und auf Freigabe der Tabelle *dbo.Buch* wartet. Nach einer kurzen Pause wird einer der beiden Prozesse zum Deadlockopfer erklärt und automatisch abgebrochen, sodass im anderen Pro-

zess auch die zweite `UPDATE`-Anweisung ausgeführt werden kann. Als Ergebnis der abgebrochenen Abfrage erhalten Sie eine Meldung der folgenden Art:

Meldung 1205, Ebene 13, Status 51, Zeile 1
Die Transaktion (Prozess-ID 56) befand sich auf Sperre (Ressourcen aufgrund eines anderen Prozesses in einer Deadlock-Situation) und wurde als Deadlock-Opfer ausgewählt. Führen Sie die Transaktion erneut aus.

8. Durch Abfragen der Systemvariablen `@@TRANCOUNT` können Sie verifizieren, dass bei der abgebrochenen Transaktion aufgrund des automatischen `ROLLBACK` keine Transaktion mehr offen ist, während im anderen Abfragefenster (in dem beide `UPDATE`-Anweisungen erfolgreich ausgeführt wurden) die Transaktion noch geöffnet ist. Diese können Sie nun ebenfalls mit der `ROLLBACK`-Anweisung abbrechen.

Da die Erkennung von Deadlocks Zeit erfordert und die abgebrochenen Skripts der Deadlockopfer erneut ausgeführt werden müssen, sollte man Deadlocks möglichst von vornherein vermeiden oder das Deadlock-Risiko zumindest auf ein Minimum reduzieren. Dafür gibt es vor allem zwei einfache Grundregeln:

- Halten Sie Transaktionen so klein wie möglich. Insbesondere sollte innerhalb einer Transaktion nicht auf eine Anwenderaktion (wie beispielsweise eine Eingabe) gewartet werden.
- Definieren Sie eine einheitliche Reihenfolge, in der Tabellen geändert werden.

8.7 Übungen zu diesem Kapitel

In diesem Abschnitt finden Sie einige Übungen zu diesem Kapitel. Die richtigen Antworten, Lösungen und den Programmcode finden Sie wie immer auf der Website *www.vsxpress.de.*

Übung 8.1

Schreiben Sie ein SQL-Skript, das alle Tracks der CD mit der ID 1 in eine (lokale) temporäre Tabelle (*#CD1Tracks*) speichert und anschließend aus dieser temporären Tabelle die Tracks einmal nach *TrackNr* sortiert und einmal alphabetisch sortiert ausgibt.

Übung 8.2

Entwerfen Sie ein SQL-Skript, das die Jahre 1980 bis 2009 in einer Schleife durchläuft und für jedes Jahr prüft, ob CDs gespeichert sind, die in diesem Jahr erschienen sind. Ist dies der Fall, so sind die entsprechenden Titel und Interpreten (der CD, nicht der einzelnen Tracks) auszugeben, ansonsten die Meldung *keine CD aus diesem Jahr gespeichert*.

Übung 8.3

Debuggen Sie das Skript aus der vorhergehenden Übung und beobachten Sie, wie die verwendeten Variablenwerte sich verändern.

Übung 8.4

Schreiben Sie ein SQL-Skript, das ausgibt, wie viel Prozent der gespeicherten DVDs mit 1 oder 2 bewertet wurden. Fügen Sie diesem Skript eine Fehlerbehandlung hinzu, um den Fall abzufangen, falls noch keine DVDs erfasst wurden. Statt dem üblichen Fehler wegen Division durch 0 soll dann der Text *keine Daten zum Auswerten vorhanden* ausgegeben werden.

Übung 8.5

Zwei Transaktionen in separaten Prozessen greifen zuerst auf die Tabelle *dbo.Buch* und anschließend auf die Tabelle *dbo.DVD* zu. Welche der beiden Transaktionen kann komplett durchlaufen?

8.8 Zusammenfassung

In diesem Kapitel haben Sie viele Möglichkeiten kennengelernt, die Sie nutzen können, um mit SQL Server auch komplexere Skripts schreiben und ausführen zu können. Während die reine Ausführung auch mit dem Kommandozeilentool *SQLCMD* möglich ist, bietet der Abfrage-Editor des SQL Server Management Studio zahlreiche Hilfestellungen, mit denen Sie SQL-Skripts nicht nur besser entwerfen, sondern auch testen können. Dazu zählt das zeilenweise Abarbeiten sowie die Verwendung von Haltepunkten. Eventuell auftretende Fehler können Sie mit einem `TRY...CATCH`-Konstrukt abfangen oder eigene Fehler mit der `RAISERROR`-Anweisung erzwingen.

An SQL-Sprachfeatures wurden Systemvariablen sowie selbst definierbare Variablen behandelt, die auch ganze Tabellen beinhalten können. Für Letzteres sind als Alternative aber auch temporäre Tabellen verwendbar.

Zur Steuerung des Skriptablaufs können Fallunterscheidungen mit der `IF`-Anweisung oder `WHILE`-Schleifen verwendet werden, die sich entweder auf einzelne Zeilen oder ganze Anweisungsblöcke beziehen, die mit `BEGIN` und `END` eingefasst sind.

Zum Schluss des Kapitels wurde noch ein Thema behandelt, das vor allem für Mehrbenutzerumgebungen eine wichtige Rolle spielt: die Zusammenhänge von Sperren, Transaktionen und Deadlocks. Transaktionen können aber auch genutzt werden, um die Korrektheit von Datenänderungen zu prüfen, bevor diese endgültig ausgeführt werden.

Kapitel 9

Gespeicherte Prozeduren, Funktionen, Trigger und Cursor

In diesem Kapitel lernen Sie

- wie Sie mithilfe von gespeicherten Prozeduren oft benötigte SQL-Skripts speichern und parametrisieren können
- welche Möglichkeiten Funktionen bieten, um immer wiederkehrende Berechnungen zu vereinfachen
- wie man mit Triggern Aufgaben automatisieren kann
- was es mit SQL-Cursorn auf sich hat

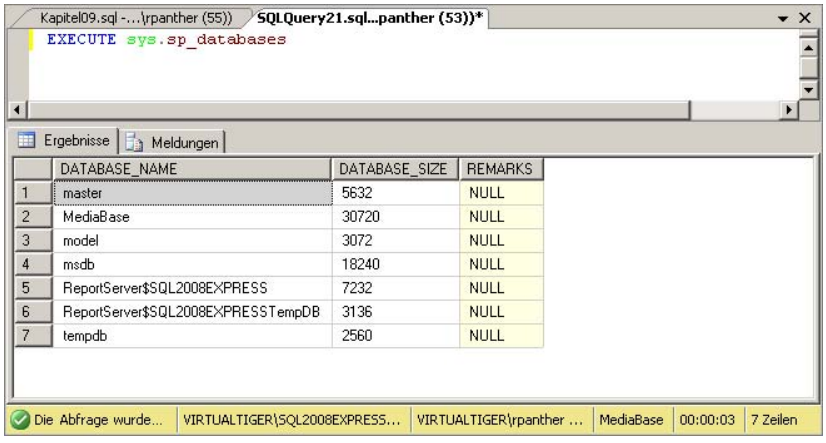
9.1 Systemprozeduren und -funktionen

SQL Server stellt eine ganze Reihe von vorgefertigten Systemprozeduren und -funktionen zur Verfügung, die man direkt verwenden kann.

Systemprozeduren

Systemprozeduren sind in großer Menge vorhanden, werden für alltägliche Aufgaben aber eher selten benötigt. Daher folgt an dieser Stelle nur eine Beschreibung der Nutzung von Systemprozeduren anhand einiger ausgewählter Beispiele.

Zum Aufrufen von gespeicherten Prozeduren wird der `EXECUTE`-Befehl (oder kurz: `EXEC`) verwendet. Über die Systemprozedur `sys.sp_databases` erhalten Sie beispielsweise eine Auflistung aller Datenbanken mit ihrer Größe (in Megabytes).



```
EXECUTE sys.sp_databases
```

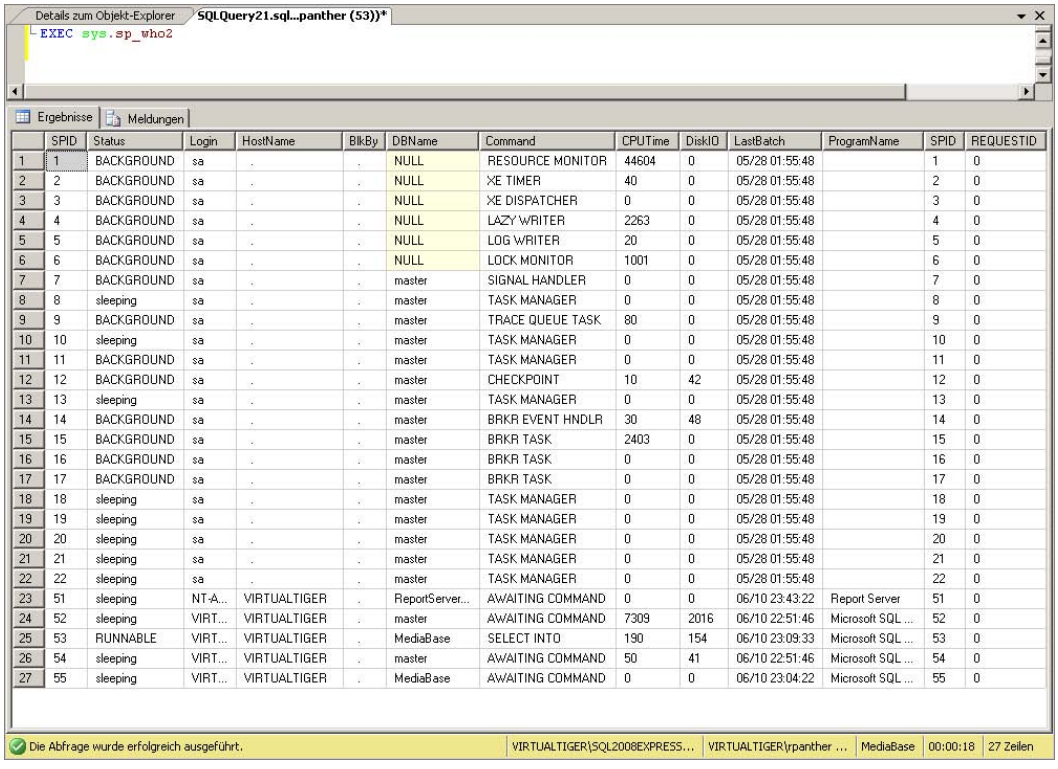
Ergebnisse | Meldungen

	DATABASE_NAME	DATABASE_SIZE	REMARKS
1	master	5632	NULL
2	MediaBase	30720	NULL
3	model	3072	NULL
4	msdb	18240	NULL
5	ReportServer$SQL2008EXPRESS	7232	NULL
6	ReportServer$SQL2008EXPRESSTempDB	3136	NULL
7	tempdb	2560	NULL

Die Abfrage wurde... | VIRTUALTIGER\SQL2008EXPRESS... | VIRTUALTIGER\rpanther ... | MediaBase | 00:00:03 | 7 Zeilen

Abbildung 9.1: Das Ergebnis der Systemprozedur `sys.sp_databases`

Eine weitere hilfreiche Systemprozedur ist `sys.sp_who` bzw. deren Nachfolger `sys.sp_who2`, die alle SQL Server-Prozesse auflisten. Beide zeigen neben der SQL Server-internen Prozess-ID (SPID) Informationen wie Status, Login-Name, Hostname und Datenbank an. Die neuere Variante `sp_who2` liefert hier allerdings noch ein paar Details mehr, wie beispielsweise die Prozessorzeit (*CPUTime*) und Festplattenaktivität (*DiskIO*) der jeweiligen Prozesse.



```
EXEC sys.sp_who2
```

Ergebnisse | Meldungen

	SPID	Status	Login	HostName	BlkBy	DBName	Command	CPUTime	DiskIO	LastBatch	ProgramName	SPID	REQUESTID
1	1	BACKGROUND	sa	.	.	NULL	RESOURCE MONITOR	44604	0	05/28 01:55:48		1	0
2	2	BACKGROUND	sa	.	.	NULL	XE TIMER	40	0	05/28 01:55:48		2	0
3	3	BACKGROUND	sa	.	.	NULL	XE DISPATCHER	0	0	05/28 01:55:48		3	0
4	4	BACKGROUND	sa	.	.	NULL	LAZY WRITER	2263	0	05/28 01:55:48		4	0
5	5	BACKGROUND	sa	.	.	NULL	LOG WRITER	20	0	05/28 01:55:48		5	0
6	6	BACKGROUND	sa	.	.	NULL	LOCK MONITOR	1001	0	05/28 01:55:48		6	0
7	7	BACKGROUND	sa	.	.	master	SIGNAL HANDLER	0	0	05/28 01:55:48		7	0
8	8	sleeping	sa	.	.	master	TASK MANAGER	0	0	05/28 01:55:48		8	0
9	9	BACKGROUND	sa	.	.	master	TRACE QUEUE TASK	80	0	05/28 01:55:48		9	0
10	10	sleeping	sa	.	.	master	TASK MANAGER	0	0	05/28 01:55:48		10	0
11	11	BACKGROUND	sa	.	.	master	TASK MANAGER	0	0	05/28 01:55:48		11	0
12	12	BACKGROUND	sa	.	.	master	CHECKPOINT	10	42	05/28 01:55:48		12	0
13	13	sleeping	sa	.	.	master	TASK MANAGER	0	0	05/28 01:55:48		13	0
14	14	BACKGROUND	sa	.	.	master	BRKR EVENT HNDLR	30	48	05/28 01:55:48		14	0
15	15	BACKGROUND	sa	.	.	master	BRKR TASK	2403	0	05/28 01:55:48		15	0
16	16	BACKGROUND	sa	.	.	master	BRKR TASK	0	0	05/28 01:55:48		16	0
17	17	BACKGROUND	sa	.	.	master	BRKR TASK	0	0	05/28 01:55:48		17	0
18	18	sleeping	sa	.	.	master	TASK MANAGER	0	0	05/28 01:55:48		18	0
19	19	sleeping	sa	.	.	master	TASK MANAGER	0	0	05/28 01:55:48		19	0
20	20	sleeping	sa	.	.	master	TASK MANAGER	0	0	05/28 01:55:48		20	0
21	21	sleeping	sa	.	.	master	TASK MANAGER	0	0	05/28 01:55:48		21	0
22	22	sleeping	sa	.	.	master	TASK MANAGER	0	0	05/28 01:55:48		22	0
23	51	sleeping	NT-A...	VIRTUALTIGER	.	ReportServer...	AWAITING COMMAND	0	0	06/10 23:43:22	Report Server	51	0
24	52	sleeping	VIRT...	VIRTUALTIGER	.	master	AWAITING COMMAND	7309	2016	06/10 22:51:46	Microsoft SQL ...	52	0
25	53	RUNNABLE	VIRT...	VIRTUALTIGER	.	MediaBase	SELECT INTO	190	154	06/10 23:09:33	Microsoft SQL ...	53	0
26	54	sleeping	VIRT...	VIRTUALTIGER	.	master	AWAITING COMMAND	50	41	06/10 22:51:46	Microsoft SQL ...	54	0
27	55	sleeping	VIRT...	VIRTUALTIGER	.	MediaBase	AWAITING COMMAND	0	0	06/10 23:04:22	Microsoft SQL ...	55	0

Die Abfrage wurde erfolgreich ausgeführt. | VIRTUALTIGER\SQL2008EXPRESS... | VIRTUALTIGER\rpanther ... | MediaBase | 00:00:18 | 27 Zeilen

Abbildung 9.2: Das Ergebnis der Systemprozedur `sys.sp_who2`

Sollte ein Prozess von einem anderen blockiert werden, so ist dessen Prozess-ID in der Spalte *BlkBy* zu finden.

Tipp: Blockierungen aufheben

Wenn Sie mit der Systemprozedur `sys.sp_who2` feststellen, dass ein Prozess von einem anderen blockiert wird, können Sie den blockierenden Prozess mit der SQL-Anweisung `KILL` abbrechen. Dabei muss lediglich die SPID des abzubrechenden Prozesses als Parameter übergeben werden.

Mit der Systemprozedur `sys.sp_renamedb` können Sie Datenbanken umbenennen. Dazu werden sowohl der alte als auch der neue Name der Datenbank als Parameter übergeben. Mit der folgenden Anweisung könnten Sie der Datenbank *MediaBase* einen neuen Namen geben:

```
EXEC sys.sp_renamedb MediaBase, MediaBaseReloaded
```

Dabei wird lediglich der Name der Datenbank geändert, die Dateinamen bleiben in der ursprünglichen Form erhalten.[1]

Eine Systemprozedur muss auf jeden Fall noch erwähnt werden. Die Prozedur `sys.sp_sqlexec` führt beliebige SQL-Kommandos aus, die als Parameter übergeben werden:

Beispiel: `EXEC sys.sp_sqlexec 'SELECT * FROM dbo.Buch'`

Das mag auf den ersten Blick unsinnig erscheinen, denn dann könnte man ja die SQL-Anweisung auch ohne Verwendung der Systemprozedur eingeben. Der Sinn von `sys.sp_sqlexec` liegt allerdings darin, dynamisches SQL zu ermöglichen. Das bedeutet, dass man eine SQL-Anweisung in einer Variablen zusammenbaut und diese dann an `sys.sp_sqlexec` als Parameter übergibt.

Probieren Sie doch einmal das folgende SQL-Skript aus:

```
DECLARE @SQLCommand AS VARCHAR(255)
DECLARE @SQLTable AS VARCHAR(64)
SET @SQLTable = 'dbo.Buch'
SET @SQLCommand = 'SELECT * FROM ' + @SQLTable
EXEC sys.sp_sqlexec @SQLCommand
```

Das Ergebnis ist wiederum die Anzeige der Tabelle *dbo.Buch*, wobei dieses Skript leicht so angepasst werden kann, dass damit verschiedene Tabellen angezeigt werden.

Die wichtigsten Systemfunktionen

Während Systemprozeduren eher selten benötigt werden, spielen Systemfunktionen eine größere Rolle in der alltäglichen Arbeit mit SQL, zumal man diese oft als Bestandteil von `SELECT`-Abfragen verwendet. Da Funktionen nur einen Wert zurückliefern, diesen aber nicht automatisch anzeigen, wird dazu auch eine aufrufende Anweisung, die diesen Funktionswert weiter verarbeitet, benötigt.

Die einfachste und sicherlich bekannteste Systemfunktion ist die `GETDATE()`-Funktion, die den aktuellen Zeitpunkt (Datum mit Uhrzeit) zurückgibt. So erhalten Sie nach Ausführung der Abfrage

[1] Falls Sie die oben gezeigte Anweisung ausprobieren, denken Sie daran, dies durch dieselbe Anweisung mit vertauschten Parametern wieder rückgängig zu machen, damit der Datenbankname zu den weiteren Beispielen in diesem Buch passt.

```
SELECT GETDATE()
```

ein Ergebnis in der folgenden Form: 2009-06-11 23:24:32.500

Mit den Funktionen `YEAR()`, `MONTH()` und `DAY()` lässt sich der jeweilige Datumsbestandsteil des übergebenen Datums herausfinden. Zusammen mit der `GETDATE()`-Funktion lassen sich damit die einzelnen Bestandteile des aktuellen Datums selektieren:

```
SELECT GETDATE() AS Datum, YEAR(GETDATE()) AS Jahr, MONTH(GETDATE()) AS Monat, DAY(GETDATE()) AS Tag
```

Noch flexibler lassen sich einzelne Bestandteile eines Datums mit der `DATEPART()`-Funktion herausfiltern. Dazu wird neben dem Datum selbst noch die Zeiteinheit übergeben.

Tabelle 9.1: SQL Server-Zeiteinheiten

Bezeichnung(en)	Bedeutung
DAY, D, DD	Tag (des Monats)
DAYOFYEAR, DY, Y	Tag (des Jahres)
HOUR, HH	Stunde
ISO_WEEK, ISOWK, ISOWW	
MONTH, M, MM	Monat
MICROSECOND, MCS	Mikrosekunde
MILLISECOND, MS	Millisekunde
MINUTE, MI, N	Minute
NANOSECOND, NS	Nanosekunde
QUARTER, QQ, Q	Quartal
SECOND, SS, S	Sekunde
TZOFFSET, TZ	
WEEKDAY, DW, W	Tag (der Woche)
WEEK, WK, WW	Woche
YEAR, YY, YYYY	Jahr

Somit gibt die folgende Anweisung die aktuelle Kalenderwoche aus:

```
SELECT DATEPART(WEEK, GETDATE()) AS KW
```

Als weitere Datumsfunktionen, die ebenfalls die oben aufgelisteten Zeiteinheiten verwenden, sind noch `DATEADD()` und `DATEDIFF()` zu erwähnen. Die erste Funktion addiert zu einem übergebenen Datum ein Intervall einer bestimmten Zeiteinheit hinzu, während die zweite Funktion die Differenz zwischen zwei Datumswerten in einer übergebenen Zeiteinheit errechnet.

Die folgende Anweisung addiert zum aktuellen Datum 14 Tage:

```
SELECT DATEADD(DAY, 14, GETDATE())
```

Die `DATEDIFF()`-Funktion kann beispielsweise gut verwendet werden, um die Tage bis Weihnachten zu berechnen:

```
SELECT DATEDIFF(DAY, GETDATE(), '24.12.2009')
```

Neben den Datumsfunktionen gibt es natürlich auch noch eine Reihe weiterer Systemfunktionen. So sind – wie in fast jeder Programmiersprache – die wichtigsten mathematischen Funktionen (wie beispielsweise `ABS()`, `SIN()`, `COS()`, `LOG()`, `EXP()`, `PI()` etc.) vertreten.

Noch wichtiger sind die zahlreichen Funktionen zum Arbeiten mit Zeichenketten. Die Funktionen `LEFT()` und `RIGHT()` geben die ersten bzw. letzten Zeichen einer Zeichenkette zurück, wobei als Parameter die Anzahl der Zeichen und die Zeichenkette selbst übergeben werden. Will man eine Teilzeichenkette in der Mitte herausschneiden, kann man mit der `SUBSTRING`-Funktion neben der Zeichenkette die Position und Anzahl der Zeichen übergeben. Das folgende Beispiel veranschaulicht die Arbeitsweise der drei Funktionen:

```
SELECT SUBSTRING('ABCDE', 2, 3), LEFT('ABCDE', 2), RIGHT('ABCDE',3)
```

Wenn Sie die Länge einer Zeichenkette ermitteln wollen, bietet sich die `LEN()`-Funktion an. Die folgende Abfrage gibt alle Buchtitel zusammen mit ihrer jeweiligen Länge an:

```
SELECT Titel, LEN(Titel) FROM dbo.Buch
```

Dies funktioniert allerdings nur, weil das Feld *Titel* als `VARCHAR(80)` deklariert wurde. Hätten Sie das Feld als `CHAR(80)` angelegt, würden die Feldinhalte mit Leerzeichen aufgefüllt und es würde für alle Zeilen die Länge 80 zurückgegeben. Aber auch hier gibt es Systemfunktionen, die Abhilfe schaffen. Mit den Funktionen `LTRIM()` und `RTRIM()` können Sie führende und folgende Leerzeichen abschneiden, sodass Sie für das gerade beschriebene Problem die `RTRIM()`- und `LEN()`-Funktion kombiniert einsetzen würden:

```
SELECT Titel, LEN(RTRIM(Titel)) FROM dbo.Buch
```

Mit den Funktionen `UPPER()` und `LOWER()` können Sie eine Zeichenkette komplett in Groß- bzw. Kleinbuchstaben umwandeln. Probieren Sie einmal folgende Abfrage aus:

```
SELECT Autor, UPPER(Autor), LOWER(Autor) FROM dbo.Buch
```

Dazu gibt es noch eine Reihe an allgemeinen Funktionen, die verschiedenste Bereiche abdecken. Darunter sind einige, die Informationen über die aktuelle Verbindung ausgeben. So gibt `USER_NAME()` den aktuellen Benutzer zurück, `HOST_NAME()` den Servernamen und `APP_NAME()` den Namen der Anwendung, mit der die Verbindung zum Datenbankserver hergestellt wurde.

```
SELECT USER_NAME() AS Benutzer, HOST_NAME() AS Host, APP_NAME() AS Anwendung
```

Mit der Funktion `NEWID()` können Sie eine neue GUID (Globally Unique Identifier) erzeugen. Dabei handelt es sich um eine weltweit eindeutige Kombination aus verschiedenen Zeichen, die gerne verwendet wird, um Zeilen in Tabellen, die später einmal zusammengeführt werden, eindeutig zu identifizieren. Zum Speichern verwendet SQL Server dafür einen eigenen Datentyp, der `uniqueidentifier` heißt.

```
SELECT NEWID()
```

Hinweis: Übersicht über Systemprozeduren und -funktionen

Die oben gezeigten Systemprozeduren und -funktionen stellen lediglich eine Auswahl dar. Wenn Sie eine komplette Übersicht über alle verfügbaren Systemprozeduren und -funktionen benötigen, können Sie diese im Objekt-Explorer in einer beliebigen Datenbank unter *Programmierbarkeit/Gespeicherte Prozeduren/Gespeicherte Systemprozeduren* bzw. *Programmierbarkeit/Funktionen/Systemfunktionen* finden. Letztere sind sogar noch in weitere Kategorien unterteilt.

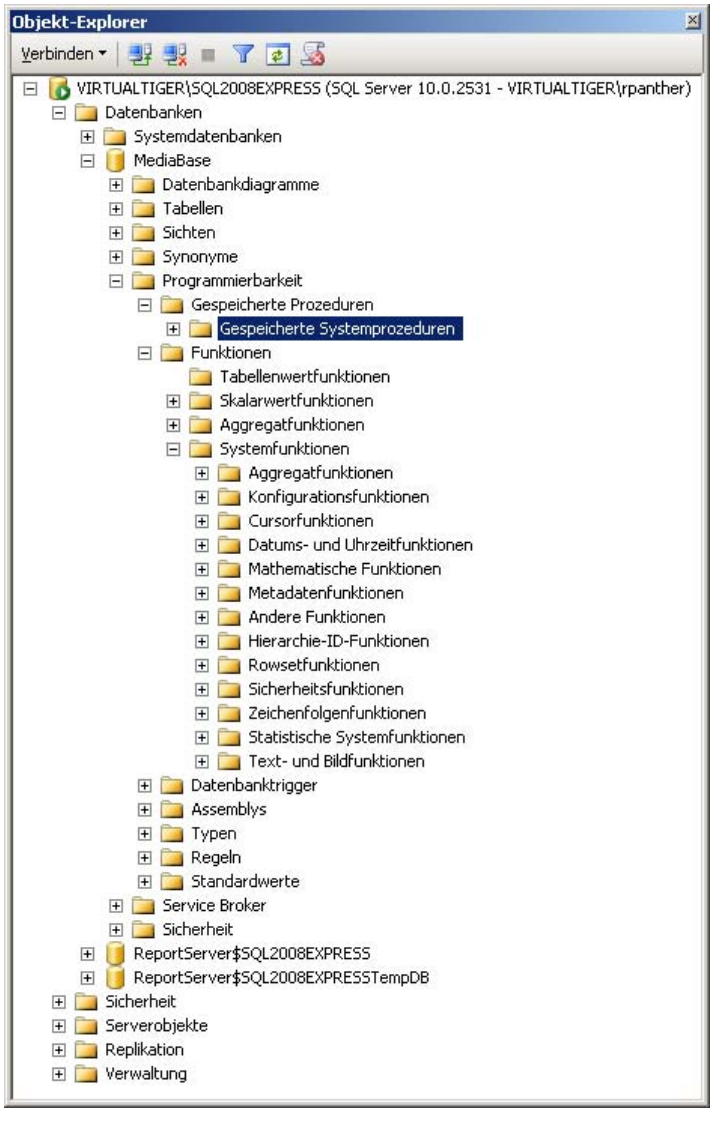


Abbildung 9.3: Die Systemprozeduren und -funktionen im Objekt-Explorer

9.2 Benutzerdefinierte gespeicherte Prozeduren

Jetzt, wo Sie die wichtigsten Systemprozeduren und -funktionen kennengelernt haben, wollen wir einmal darangehen, eigene Prozeduren und Funktionen zu entwickeln.

Einfache gespeicherte Prozeduren

Die einfachste Variante stellen gespeicherte Prozeduren ohne Parameter dar. Dazu können Sie ein beliebiges SQL-Skript mit `BEGIN` und `END` eingrenzen und davor die Anweisung `CREATE PROCEDURE [Prozedurname] AS` setzen. Starten wir mit einem einfachen Beispiel:

1. Stellen Sie im SQL Server Management Studio eine Verbindung zur lokalen Serverinstanz *SQL2008Express* her und öffnen Sie im Objekt-Explorer ein Abfragefenster.
2. Verbinden Sie sich mit der Datenbank *MediaBase* und führen Sie folgende Anweisung aus, um die gespeicherte Prozedur zu erstellen:

```
CREATE PROCEDURE dbo.BuecherAnzeigen AS
BEGIN
    SELECT * FROM dbo.Buch
END
GO
```

3. Rufen Sie die Prozedur nun auf, indem Sie die folgende Anweisung ausführen:

```
EXEC dbo.BuecherAnzeigen
```

4. Es wird eine Liste mit allen gespeicherten Büchern angezeigt.

Richtig Sinn macht eine gespeicherte Prozedur natürlich nur, wenn der Inhalt etwas komplexer wird.

Wichtig: Gespeicherte Prozeduren und Funktionen immer mit GO abschließen

Die Definition von benutzerdefinierten gespeicherten Prozeduren und Funktionen sollte immer mit der `GO`-Anweisung abgeschlossen werden. Sonst kann es leicht vorkommen, dass darauf folgende Anweisungen mit in die gespeicherte Prozedur (oder Funktion) integriert werden.

Gespeicherte Prozeduren mit Parametern

Um die Prozedur aus dem letzten Beispiel etwas universeller nutzbar zu machen, erweitern wir sie nun so, dass sie nicht alle Bücher anzeigt, sondern nur die von einem bestimmten Autor. Der Name des Autors soll als Parameter an die Funktion übergeben werden. Die Deklaration dieses Parameters erfolgt zwischen dem Prozedurnamen und dem Schlüsselwort `AS`:

1. Führen Sie im Abfragefenster (verbunden mit der Datenbank *MediaBase*) die folgende Anweisung aus, um die gespeicherte Prozedur zu erstellen:

```
CREATE PROCEDURE dbo.BuecherVonAutor
 @Aut    or varchar(80)
AS
BEGIN
    SELECT * FROM dbo.Buch WHERE Autor = @Autor
END
GO
```

2. Rufen Sie die Prozedur nun auf, indem Sie die folgende Anweisung ausführen:

```
EXEC dbo.BuecherVonAutor 'Robert Panther'
```

3. Es wird eine Liste mit allen Büchern des Autors *Robert Panther* angezeigt. Variieren Sie den Aufruf der Abfrage nun, indem Sie andere Autorennamen als Parameter übergeben.

Parameter lassen sich auch mit Standardwerten (Defaults) versehen, wodurch sie zu optionalen Parametern werden (vorausgesetzt, es folgen keine weiteren nicht optionalen Parameter). Passen wir nun die zuerst erstellte Prozedur so an, dass diese als optionale Parameter die Angabe des Anfangs von Titel und Autorennamen akzeptiert:

1. Öffnen Sie im Objekt-Explorer den Zweig *Datenbanken/MediaBase/Programmierbarkeit/Gespeicherte Prozeduren*. Hier müssten nun die beiden bisher erstellen gespeicherten Prozeduren (*dbo.BuecherAnzeigen* und *dbo.BuecherVonAutor*) aufgelistet sein.
2. Klicken Sie nun die gespeicherte Prozedur *dbo.BuecherAnzeigen* mit der rechten Maustaste an und wählen Sie die Option *Ändern*. Daraufhin wird im Abfragefenster ein Skript für die Änderung der gespeicherten Prozedur generiert, das wie folgt aussieht:

```
USE [MediaBase]
GO
/****** Object:  StoredProcedure [dbo].[BuecherAnzeigen]    Script Date: 06/12/2009 17:01:07 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[BuecherAnzeigen] AS
BEGIN
    SELECT * FROM dbo.Buch
END
GO
```

3. Passen Sie dieses Skript nun wie folgt an:

```
ALTER PROCEDURE [dbo].[BuecherAnzeigen]
    @TitelAnfang varchar(80) = '',
    @AutorAnfang varchar(80) = ''
AS
BEGIN
 SELE   CT *
    FROM dbo.Buch
    WHERE Titel LIKE @TitelAnfang + '%'
              AND Autor LIKE @AutorAnfang + '%'
END
```

4. Wenn Sie das Skript ausführen, wird die gespeicherte Prozedur um die optionalen Parameter erweitert, die beide als Standardwert eine leere Zeichenkette erhalten. Probieren Sie nun folgende Aufrufe aus:

```
EXEC dbo.BuecherAnzeigen
EXEC dbo.BuecherAnzeigen 'Das'
EXEC dbo.BuecherAnzeigen 'Das', 'Ro'
```

5. Je nach Büchern, die Sie in Ihrer Datenbank erfasst haben, wird das Ergebnis variieren. Auf jeden Fall erhalten Sie bei der ersten Variante des Aufrufs eine Liste mit allen Büchern, während die zweite Variante nur die Bücher anzeigt, die mit *Das* beginnen. Die dritte Variante schließlich zeigt alle Bücher, deren Titel mit *Das* beginnt und die von einem Autor verfasst sind, dessen Name mit *Ro* beginnt.

Problematisch ist dabei, dass Sie auf diese Weise keine Möglichkeit haben, nur den zweiten Parameter anzugeben und für den ersten Parameter den Standardwert zu nutzen. Um dies dennoch realisieren zu können, bietet die `EXECUTE`-Anweisung die Möglichkeit, bei den Parametern deren Namen mit anzugeben. Damit lässt sich auch von der vorgegebenen Reihenfolge der Parameter abweichen. Probieren Sie einmal folgende Aufrufe der gespeicherten Prozedur aus:

```
EXEC dbo.BuecherAnzeigen @AutorAnfang='Ro'
EXEC dbo.BuecherAnzeigen @AutorAnfang='Ro', @TitelAnfang='Das'
```

Gespeicherte Prozeduren mit OUTPUT-Parametern

Gespeicherte Prozeduren können auch Werte zurückgeben. Im Gegensatz zu Funktionen kann eine Prozedur sogar mehrere Rückgabewerte haben. Rückgabeparameter werden mit dem Zusatz `OUTPUT` (oder dessen Kurzform `OUT`) als solche gekennzeichnet. Erweitern wir nun die Prozedur *dbo.BuecherVonAutor* so, dass neben der Anzeige der Bücher selbst auch die Anzahl der gefundenen Bücher zurückgegeben wird.

1. Führen Sie im Abfrage-Editor das folgende SQL-Skript aus, um die gespeicherte Prozedur anzupassen:

```
ALTER PROCEDURE [dbo].[BuecherVonAutor]
 @Aut    or varchar(80),
    @Anzahl int OUTPUT
AS
BEGIN
    SELECT * FROM dbo.Buch WHERE Autor = @Autor
    SET @Anzahl = (SELECT COUNT(*) FROM dbo.Buch WHERE Autor = @Autor)
END
GO
```

2. Um den `OUTPUT`-Parameter nach Ausführung der gespeicherten Prozedur auch auswerten zu können, benötigen wir eine Variable, die vorher deklariert werden muss. Nach Rücksprung aus der gespeicherten Prozedur kann diese dann wieder abgefragt werden. Führen Sie die folgenden Anweisungen aus, um die gespeicherte Prozedur zu testen:

```
DECLARE @Zeilen AS int
EXEC dbo.BuecherVonAutor 'Robert Panther', @Zeilen OUTPUT
SELECT @Zeilen AS ZeilenAnzahl
```

Wichtig: OUTPUT auch beim Aufruf der Prozedur verwenden

Beachten Sie, dass auch beim Aufruf der Prozedur der Zusatz `OUTPUT` beim entsprechenden Parameter angegeben werden muss. Ohne diesen Zusatz funktioniert der Aufruf zwar auch, die Variable bekommt dann aber keinen Wert zugewiesen, sodass diese immer `NULL` enthalten würde.

9.3 Benutzerdefinierte Funktionen

Während gespeicherte Prozeduren selbstständig ausführbar sind, liefern Funktionen einen Wert zurück und können überall dort eingesetzt werden, wo auch der zurückgegebene Wert passen würde. Man unterscheidet zwischen drei verschiedenen Typen von Funktionen:

- Skalarfunktionen
- Tabellenwertfunktionen
- Aggregatfunktionen

Diese drei Varianten sollen im Folgenden etwas genauer beleuchtet werden. Insbesondere die ersten beiden Typen lassen sich auch besonders leicht nutzen, um mithilfe einer benutzerdefinierten Logik spezielle Anforderungen zu realisieren.

Skalarwertfunktionen (oder kurz: Skalarfunktionen)

Diese Variante von Funktionen entspricht dem, was man sich normalerweise unter einer Funktion vorstellt, denn von ihnen wird ein einfacher (skalarer) Wert als Ergebnis zurückgeliefert. Damit bietet sich vor allem die Verwendung in Abfragen an, wie das folgende Beispiel zeigt:

```
SELECT Titel, GETDATE() FROM dbo.Buch
```

Hier wurde die Ihnen bereits bekannte Systemfunktion `GETDATE()` verwendet, die das aktuelle Datum (mit Uhrzeit) zurückliefert. Mehr Sinn macht die Verwendung einer Funktion innerhalb einer Abfrage natürlich, wenn diese zusammen mit einem Feld aus der Abfrage verwendet wird, sodass für jede Zeile der Tabelle andere Werte entstehen. Die folgende Abfrage verwendet beispielsweise die Systemfunktionen `GETDATE()` und `YEAR()`, um das Alter eines Buches in Jahren zu bestimmen:

```
SELECT Titel, Erscheinungsjahr, YEAR(GETDATE())-Erscheinungsjahr AS BuchAlter FROM dbo.Buch
```

Über eine benutzerdefinierte Funktion lässt sich diese Abfrage noch etwas eleganter gestalten:

```
SELECT Titel, Erscheinungsjahr, dbo.UFN_JahreSeit(Erscheinungsjahr) AS BuchAlter FROM dbo.Buch
```

Nun muss nur noch die Berechnung selbst als Funktion implementiert werden:

1. Stellen Sie im SQL Server Management Studio eine Verbindung zur lokalen Serverinstanz *SQL2008Express* her und öffnen Sie im Objekt-Explorer die Datenbank *MediaBase*.
2. Wenn Sie nun den Zweig *Programmierbarkeit* öffnen, finden Sie darunter einen Eintrag *Funktionen*, der die verschiedenen Funktionstypen beinhaltet.
3. Klicken Sie den Unterpunkt *Skalarwertfunktionen* mit der rechten Maustaste an und wählen Sie die Option *Neue Skalarwertfunkion*.
4. Es wird automatisch ein Gerüst für eine Skalarwertfunktion erstellt, das Sie nur noch füllen müssen:

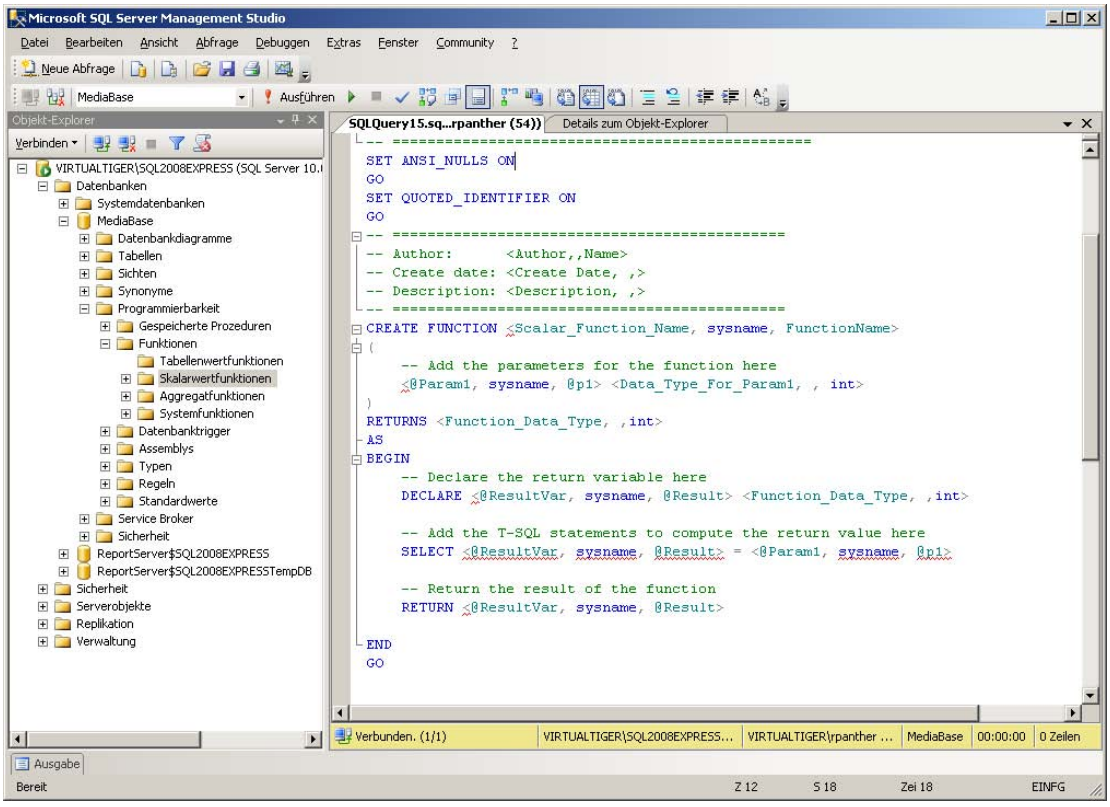


Abbildung 9.4: Das Gerüst für eine Skalarwertfunktion

5. Passen Sie die Vorlage nun so an, dass folgende Funktionsdefinition übrig bleibt:

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- ============================================
-- Author:    Robert Panther
-- Create date:         09.06.2009
-- Description:         ermittelt die Differenz des aktuellen Datums
--                      zu einer übergebenen Jahreszahl
-- ============================================
CREATE FUNCTION dbo.UFN_JahreSeit
(
    -- Add the parameters for the function here
 @Jah   reszahl int
)
RETURNS int
AS
BEGIN
    -- Return the result of the function
 RETU   RN YEAR(GETDATE())-@Jahreszahl
END
GO
```

Best Practices: Kommentarblöcke in Funktionen und gespeicherten Prozeduren

Auch wenn der einleitende Block von Kommentarzeilen optional ist, sollten Sie sich angewöhnen, diesen für alle benutzerdefinierten gespeicherten Prozeduren und Funktionen zu verwenden, um dort zumindest in ein paar Stichwörtern zu beschreiben, was die jeweilige Routine macht. In Umgebungen, in denen mehrere Entwickler gespeicherte Prozeduren und Funktionen erstellen, ist auch die Angabe des Autors wichtig, sodass man gleich sieht, an wen man sich wenden kann, wenn später einmal Änderungen oder Erweiterungen notwendig werden.

6. Führen Sie das Skript aus, um die Funktion zu erstellen.
7. Schließen Sie das Abfragefenster und führen Sie in einem weiteren Abfragefenster die folgende Anweisung aus, um die soeben erstellte Funktion auszuprobieren:

```
SELECT Titel, Erscheinungsjahr, dbo.UFN_JahreSeit(Erscheinungsjahr) AS BuchAlter
FROM dbo.Buch
```

8. Als Ergebnis sollten Sie eine Liste der Buchtitel mit Erscheinungsjahr und Alter in Jahren erhalten.

Tabellenwertfunktionen

Im Gegensatz zu Skalarwertfunktionen liefern Tabellenwertfunktionen ganze Tabellenobjekte als Ergebnis zurück. Ein vergleichbares Ergebnis können Sie erzielen, wenn Sie eine Tabellenvariable erstellen, diese in einer Prozedur als `OUTPUT`-Parameter zurückgeben und dann im Funktionsaufruf die Tabellenvariable verwenden. Die Nutzung einer Tabelle als Parameter ist allerdings erst seit SQL Server 2008 möglich.

Erstellen wir nun eine Tabellenwertfunktion, die alle Datensätze der Tabelle *CDTrack* zurückliefert, die zu einer als Parameter übergebenen CD gehören:

1. Stellen Sie im SQL Server Management Studio eine Verbindung zur lokalen Serverinstanz *SQL2008Express* her und öffnen Sie im Objekt-Explorer die Datenbank *MediaBase*.
2. Wenn Sie nun den Zweig *Programmierbarkeit* öffnen, finden Sie darunter einen Eintrag *Funktionen*, der die verschiedenen Funktionstypen beinhaltet.
3. Klicken Sie unter *Programmierbarkeit* den Unterpunkt *Funktionen/Tabellenwertfunktionen* mit der rechten Maustaste an und wählen Sie die Option *Neue Inline-Tabellenwertfunktion*.
4. Es wird automatisch ein Gerüst für eine Tabellenwertfunktion erstellt, das Sie wie folgt anpassen:

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- ============================================
-- Author:    Robert Panther
-- Create date:         09.06.2009
-- Description:         ermittelt alle Tracks zu einer CD
-- ============================================
CREATE FUNCTION dbo.UFN_TracksFromCD
(
    -- Add the parameters for the function here
 @idC   D int
```

```
)
RETURNS TABLE
AS
RETURN
(
    -- Add the SELECT statement with parameter references here
    SELECT * FROM dbo.CDTrack WHERE idCD=@idCD
)
GO
```

5. Führen Sie die `CREATE FUNCTION`-Anweisung aus, um die Tabellenwertfunktion zu erstellen.
6. Da die Funktion eine Tabelle als Ergebnis zurückgibt, kann sie wie eine Tabelle verwendet werden. Führen Sie die folgende SQL-Abfrage aus, um die Funktion zu testen:

```
SELECT *
FROM dbo.UFN_TracksFromCD(1)
```

7. Als Ergebnis bekommen Sie alle Tracks zur idCD 1 ausgegeben.

Im Gegensatz zu den gerade verwendeten Inline-Tabellenwertfunktionen gibt es noch die Tabellenwertfunktionen mit mehreren Anweisungen. Während die erste Variante einfach das Ergebnis einer `SELECT`-Abfrage zurückgibt, wird in der zweiten Variante zuerst unter Angabe der Tabellenspalten eine Tabellenvariable als Rückgabewert deklariert. Anschließend wird die Tabellenvariable mit einer oder mehreren Anweisungen gefüllt. Die oben gezeigte Tabellenwertfunktion sieht in dieser alternativen Form wie folgt aus:

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:                  Robert Panther
-- Create date:    09.06.2009
-- Description:    ermittelt alle Tracks zu einer CD
-- =============================================
CREATE FUNCTION dbo.UFN_TracksFromCD2
(
          -- Add the parameters for the function here
 @idC         D int
)
RETURNS @Ergebnis TABLE
          (ID int, idCD int, CDNr tinyint, Titel varchar(80), Interpret varchar(80), Dauer time(7), Bewertung
tinyint)
AS
BEGIN
          INSERT INTO @Ergebnis (ID, idCD, CDNr, Titel, Interpret, Dauer, Bewertung)
          SELECT ID, idCD, CDNr, Titel, Interpret, Dauer, Bewertung
          FROM dbo.CDTrack
          WHERE idCD=@idCD

 RETU         RN
END
GO
```

Diese Variante ist auf den ersten Blick etwas umständlicher und unübersichtlicher, bietet aber andererseits mehr Flexibilität, da man die Ergebnistabelle auch mit einer Folge von Anweisungen füllen kann.

Best Practices: Tabellenwertfunktion zur Realisierung einer Sicht mit Parametern

Tabellenwertfunktionen können gut genutzt werden, wenn Sie eigentlich eine Sicht verwenden würden, dieser aber Parameter übergeben wollen. Daher sind auch einige vordefinierte Sichten der älteren SQL Server-Versionen inzwischen als Tabellenwertfunktionen implementiert.

Um eine Tabellenwertfunktion mit anderen Tabellen zu verknüpfen, wird der `APPLY`-Operator verwendet, den es in zwei Ausprägungen gibt:

- `CROSS APPLY` entspricht einem `INNER JOIN`
- `OUTER APPLY` entspricht einem `LEFT OUTER JOIN`

Dabei werden die Felder, mit denen die Tabellen verknüpft werden, als Parameter an die Funktion übergeben. Innerhalb der Funktion muss dann natürlich dafür gesorgt werden, dass die richtigen Zeilen für den übergebenen Parameter selektiert werden. In unserer Beispielfunktion `dbo.UFN_TracksFromCD` von weiter oben ist dies natürlich der Fall, sodass wir diese Funktion gut mit der Tabelle *dbo.CD* verknüpfen können. So liefert die Abfrage

```
SELECT *
FROM dbo.CD CROSS APPLY dbo.UFN_TracksFromCD(CD.ID)
```

alle CDs mit ihren dazugehörigen Tracks und damit dasselbe Ergebnis wie die einfache Abfrage:

```
SELECT *
FROM dbo.CD INNER JOIN dbo.CDTrack ON CD.ID=CDTrack.idCD
```

Wenn Sie dagegen auch die CDs sehen wollen, zu denen keine Tracks gespeichert sind, würden Sie anstelle des `INNER JOIN` einen `LEFT JOIN` verwenden. Beim `APPLY`-Operator können Sie dasselbe erreichen, indem Sie die `OUTER APPLY`-Variante nutzen:

```
SELECT *
FROM dbo.CD OUTER APPLY dbo.UFN_TracksFromCD(CD.ID)
```

In der hier dargestellten Form macht die Verwendung der Tabellenwertfunktion natürlich recht wenig Sinn, da die Komplexität der Gesamtabfrage dadurch höher wird, was sich auch in einer schlechteren Performance niederschlagen kann. Wenn die Funktion selbst aber komplexer ist und nicht nur aus einer `SELECT`-Abfrage besteht, lässt sich das `APPLY`-Konstrukt nicht mehr so einfach durch einen `JOIN` ersetzen.

Aggregatfunktionen

Die systemeigenen Aggregatfunktionen wie beispielsweise `COUNT`, `SUM`, `MAX`, `MIN` und `AVG` haben Sie ja bereits kennengelernt. Seit SQL Server 2005 bietet der Microsoft SQL Server allerdings auch die Möglichkeit, mithilfe von .NET eigene Aggregatfunktionen zu erstellen, die man dann in einzelne Datenbanken integrieren kann. So wäre beispielsweise eine Funktion vorstellbar, die Zeichenketten zusammenfügt.

Da das Erstellen einer solchen Aggregatfunktion aber eher eine .NET-lastige Angelegenheit ist, die noch dazu recht selten benötigt wird, verzichte ich hier darauf, dieses Thema detailliert zu behandeln. Im Moment reicht es aus zu wissen, dass die Möglichkeit für benutzerdefinierte Aggregatfunktionen existiert.

Hinweis: Einschränkungen von Funktionen

Im Gegensatz zu gespeicherten Prozeduren unterliegen Funktionen zahlreichen Einschränkungen. So ist einerseits keine Fehlerbehandlung mit `TRY…CATCH` möglich. Dazu sind auch keine Datenänderungen durchführbar (wodurch es leider auch unmöglich gemacht wird, eventuell auftretenden Fehler in einer Fehlertabelle zu protokollieren). Sollte eines dieser Features benötigt werden, verwenden Sie stattdessen eine gespeicherte Prozedur mit `OUTPUT`-Parameter.

9.4 Trigger

Mithilfe von Triggern lassen sich Reaktionen auf Datenänderungsoperationen einfach automatisieren. Ein Trigger ist eine spezielle Form einer gespeicherten Prozedur, die automatisch ausgeführt wird, sobald ein bestimmtes Ereignis eintritt. Dabei unterscheidet man zwischen zwei Varianten von Triggern:

- DML-Trigger
- DDL-Trigger

Die häufiger eingesetzten DML-Trigger reagieren auf `INSERT`-, `UPDATE`- und `DELETE`-Anweisungen und werden meist dazu genutzt, Statistiken zu aktualisieren oder ein Protokoll bzw. eine Datenhistorie mitzuführen.

DDL-Trigger werden dagegen ausgelöst, sobald eine entsprechende Änderung an der Datenstruktur ausgeführt wird. Eine genaue Beschreibung der DDL-Trigger erfolgt in Kapitel 10, *Datenbankadministration mit SQL*, daher nun zurück zu den DML-Triggern.

DML-Trigger werden auf Basis einer Tabelle erstellt. Hier sind drei Ereignisse möglich, für die ein entsprechender Trigger definiert werden kann:

- `INSERT`-Trigger – ein oder mehrere Datensätze werden der Tabelle hinzugefügt
- `UPDATE`-Trigger – ein oder mehrere Datensätze der Tabelle werden geändert
- `DELETE`-Trigger – ein oder mehrere Datensätze der Tabelle werden gelöscht

Ein einfacher UPDATE-Trigger

Die Arbeitsweise eines DML-Triggers wird schnell klar, wenn man dies einfach mal ausprobiert.

1. Stellen Sie im SQL Server Management Studio eine Verbindung zur lokalen Serverinstanz *SQL2008Express* her und öffnen Sie im Objekt-Explorer die Datenbank *MediaBase*.
2. Erstellen Sie nun eine neue Tabelle mit Namen *dbo.AenderungsLog*, die folgende Felder enthält:

Tabelle 9.2: Die Felder der Tabelle *dbo.AenderungsLog*

Spaltenname	Datentyp	NULL-Werte zulassen	Sonstiges
ID	int	nein	Primärschlüssel
Zeitpunkt	datetime	nein	Standardwert: getdate()
Aenderung	varchar(MAX)	ja	

Für die Spalte *ID* sollte auch eine Identitätsspezifikation festgelegt werden, sodass man diese nicht manuell setzen muss.

3. Suchen Sie nun im Objekt-Explorer die Tabelle *dbo.Buch* und klicken Sie das kleine Pluszeichen davor an.
4. Klicken Sie in der daraufhin erscheinenden nächsten Ebene den Unterpunkt *Trigger* mit der rechten Maustaste an und wählen Sie die Option *Neuer Trigger*.
5. Im Abfragefenster erscheint ein Gerüst für eine Anweisung, mit der ein Trigger erstellt werden kann. Passen Sie dieses Gerüst nun wie folgt an und führen Sie das SQL-Skript anschließend aus:

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- ============================================
-- Author:              Robert Panther
-- Create date:         14.06.2009
-- Description:         Änderungshistorie für dbo.Buch schreiben
-- ============================================
CREATE TRIGGER dbo.TRG_ChangeLog
   ON  dbo.Buch
   AFTER UPDATE
AS
BEGIN
 SET    NOCOUNT ON;

   INSERT INTO AenderungsLog (Aenderung) VALUES ('Zeile in dbo.Buch geändert!')
END
GO
```

6. Wenn Sie jetzt im Objekt-Explorer den Zweig *MediaBase/Tabellen/dbo.Buch/Trigger* aufklappen, erscheint dort der neu angelegte Trigger mit Namen *TRG_ChangeLog*.

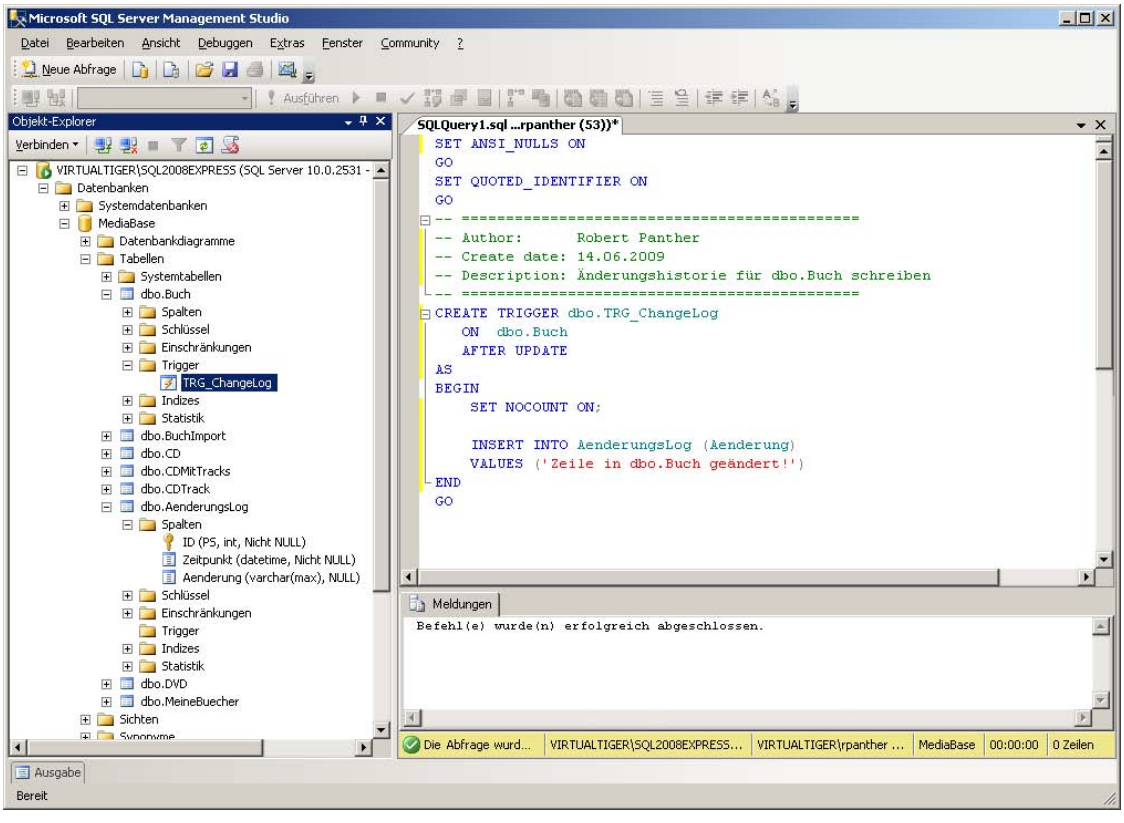


Abbildung 9.5: Definition eines UPDATE-Triggers für die Tabelle *dbo.Buch*

7. Um zu testen, dass der Trigger auch wirklich funktioniert, führen Sie nun mit folgender SQL-Anweisung eine Änderung der Daten in der Tabelle *dbo.Buch* durch:

```
UPDATE dbo.Buch
SET Bewertung=3
WHERE Bewertung IS NULL
```

8. Wenn Sie nun einen Blick in die Tabelle *dbo.AenderungsLog* werfen, werden Sie feststellen, dass dort eine Zeile eingefügt wurde, die neben dem vorher festgelegten Kommentar auch den Zeitpunkt der UPDATE-Ausführung beinhaltet.

Hinweis: Ein Trigger-Ereignis pro DML-Anweisung

Unabhängig davon, ob von der UPDATE-Anweisung keine, eine oder mehrere Zeilen betroffen waren, wird genau ein Eintrag in die Tabelle *dbo.AenderungsLog* geschrieben, da der Trigger pro Anweisung (nicht pro Datensatz) aufgerufen wird.

Sie können dies leicht überprüfen, indem Sie die UPDATE-Anweisung noch einmal ausführen. Aufgrund der Bedingung WHERE Bewertung IS NULL wird nun keine Zeile mehr geändert. In die Tabelle *dbo.AenderungsLog* wird trotzdem ein Datensatz geschrieben.

Kombinierte DML-Trigger

Zusätzlich zum UPDATE-Trigger können Sie natürlich auch einen entsprechenden INSERT- und DELETE-Trigger erstellen, indem Sie statt AFTER UPDATE eine der beiden anderen DML-Operationen angeben. Ein Trigger kann sogar auf mehrere DML-Ereignisse reagieren, indem Sie zwei oder alle drei DML-Operationen hier aufführen. Passen wir nun den Trigger entsprechend an:

1. Klicken Sie den Trigger *TRG_ChangeLog* im Objekt-Explorer mit der rechten Maustaste an und wählen Sie die Option *Ändern*.
2. Es wird eine entsprechende ALTER TRIGGER-Anweisung im Abfragefenster generiert, die Sie wie folgt anpassen und anschließend ausführen:

```
USE [MediaBase]
GO
/****** Object:  Trigger [dbo].[TRG_ChangeLog]    Script Date: 06/14/2009 14:45:14 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER TRIGGER [dbo].[TRG_ChangeLog]
   ON  [dbo].[Buch]
   AFTER UPDATE, INSERT, DELETE
AS
BEGIN
 SET    NOCOUNT ON;

    INSERT INTO AenderungsLog (Aenderung)
    VALUES ('Datenänderung in Tabelle dbo.Buch!')
END
```

3. Führen Sie anschließend die beiden folgenden SQL-Anweisungen aus, um den Trigger zu testen:

```
INSERT INTO dbo.Buch (Titel) VALUES ('TriggerTest')
DELETE FROM dbo.Buch WHERE Titel='TriggerTest'
```

4. Auch wenn die Zeile, die hinzugefügt wurde, anschließend gleich wieder gelöscht wurde, hat der Trigger für jede der beiden DML-Operationen einen Eintrag in der Tabelle *dbo.AenderungsLog* hinzugefügt.

Verwendung von geänderten Daten im Trigger

Für ein richtiges Änderungsprotokoll wäre es natürlich sinnvoll, auch die geänderten Daten selbst mit zu speichern. Dafür sind innerhalb eines Triggers zwei temporäre Tabellen verfügbar, mit denen man die gelöschten (DELETED) und hinzugefügten (INSERTED) Zeilen abfragen kann. Bei einer UPDATE-Operation stehen die alten Werte in der DELETED-Tabelle, die aktualisierten in der INSERTED-Tabelle.

Nutzen wir nun diese Tabellen, um den gerade erstellten Trigger weiter zu verfeinern:

1. Klicken Sie den Trigger *TRG_ChangeLog* im Objekt-Explorer mit der rechten Maustaste an und wählen Sie die Option *Ändern*.
2. Passen Sie den Trigger wie folgt an:

```
USE [MediaBase]
GO
/****** Object:  Trigger [dbo].[TRG_ChangeLog]    Script Date: 06/14/2009 14:45:14 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER TRIGGER [dbo].[TRG_ChangeLog]
   ON  [dbo].[Buch]
   AFTER UPDATE, INSERT, DELETE
AS
BEGIN
 SET    NOCOUNT ON;

    DECLARE @Aenderungen AS varchar(max)

    SET @Aenderungen='Datenänderung in Tabelle dbo.Buch: '
             + (SELECT ltrim(str(count(*))) FROM DELETED) + ' Zeilen gelöscht, '
             + (SELECT ltrim(str(count(*))) FROM INSERTED) + ' Zeilen hinzugefügt'

    INSERT INTO AenderungsLog (Aenderung)
 VALU   ES (@Aenderungen)
END
```

Die Systemfunktionen `str()` und `ltrim()` werden dabei benötigt, um den numerischen Wert, der von `count(*)` zurückgegeben wird, in eine Zeichenkette umzuwandeln (`str()`) und anschließend dessen führende Leerzeichen zu entfernen (`ltrim()`).

3. Führen Sie nun die folgenden SQL-Anweisungen aus, um eine Zeile hinzuzufügen, zu ändern und anschließend wieder zu löschen:

```
INSERT INTO dbo.Buch (Titel) VALUES ('TriggerTest')
UPDATE dbo.Buch SET Bewertung=1 WHERE Titel='TriggerTest'
DELETE FROM dbo.Buch WHERE Titel='TriggerTest'
```

4. Werfen Sie anschließend einen Blick in die Tabelle *dbo.AenderungsLog* und Sie werden sehen, dass beim `UPDATE` eine Zeile gelöscht und wieder eingefügt wurde.

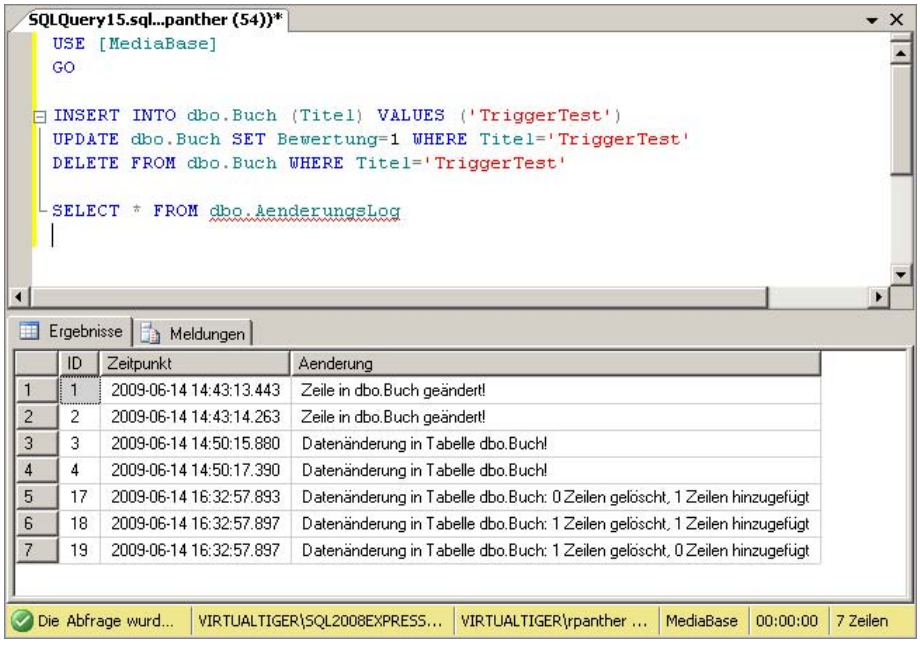


Abbildung 9.6: Verwendung der temporären Tabellen *DELETED* und *INSERTED* im Trigger

Hinweis: Ausführungsreihenfolge von Triggern

Behalten Sie bei Triggern stets die Ausführungsreihenfolge im Hinterkopf. Die eigentliche DML-Operation wird ausgeführt, bevor der Trigger selbst aktiv wird. Allerdings geschieht dies innerhalb einer impliziten Transaktion, sodass Sie mit einem expliziten `ROLLBACK` innerhalb des Triggers auch die DML-Anweisung rückgängig machen können, die den Trigger aufgerufen hat.

INSTEAD OF-Trigger

Bei der Definition des Triggers können Sie anstelle des Schlüsselwortes `AFTER` auch das Schlüsselwort `FOR` verwenden, was aber auf die Funktionsweise des Triggers keinerlei Auswirkungen hat. Es gibt allerdings noch eine dritte Alternative: Wenn Sie den Trigger mit `INSTEAD OF` deklarieren, wird nur der Trigger ausgeführt, aber nicht die aufrufende DML-Operation. Auf diesem Weg lässt sich beispielsweise ein Löschen von Daten komplett verhindern, wie folgendes Beispiel zeigt:

1. Erstellen Sie mit folgendem SQL-Skript einen Trigger für die Tabelle *dbo.DVD*:

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- ============================================
-- Author:              Robert Panther
-- Create date:         14.06.2009
-- Description:         Löschen von DVDs verhindern
-- ============================================
```

```
CREATE TRIGGER TRG_DVDLoeschenVerhindern
   ON dbo.DVD
   INSTEAD OF DELETE
AS
BEGIN
 SET    NOCOUNT ON;
    PRINT 'Löschen abgelehnt!'
END
GO
```

2. Wenn Sie nun versuchen, die Inhalte der Tabelle *dbo.DVD* mit einer `DELETE`-Anweisung zu löschen werden Sie die im `PRINT`-Befehl definierte Fehlermeldung erhalten. (Achtung: Ein `TRUNCATE TABLE` würde dagegen funktionieren, da dann der `DELETE`-Trigger nicht greift.)

Hinweis: Trigger nur gezielt einsetzen!

Trigger sind sehr mit Vorsicht zu genießen, da sie insbesondere Massenoperationen stark verlangsamen. Dazu besteht die Gefahr, durch – bewussten oder unbewussten – Einsatz von rekursiven Triggern zumindest theoretisch endlose Datenbankoperationen zu erzeugen. Glücklicherweise ist die Rekursionstiefe von Triggern bei SQL Server Express Edition auf 32 begrenzt.

9.5 SQL-Cursor

Die Sprache SQL basiert im Allgemeinen auf mengenorientierter Verarbeitung. Die SQL-Anweisung sagt dem SQL Server, was er tun soll, gibt aber nicht direkt vor, wie. Dabei sind meist mehrere Zeilen oder gar Tabellen betroffen.

In der klassischen Programmierung wird allerdings eher von satzweiser Verarbeitung ausgegangen. Um dies auch für den SQL Server zu ermöglichen, wurden die SQL-Cursor entwickelt. Dabei handelt es sich um ein Sprachkonstrukt, mit dem zuerst ein sogenannter Cursor (für CURrent Set Of Records) auf Basis einer `SELECT`-Anweisung definiert wird. Anschließend kann der Cursor in einer Schleife satzweise durchlaufen werden, um beliebige Anweisungen für jeden einzelnen Datensatz des Cursors auszuführen.

Ein einfacher Cursor

Hier ein Beispiel für einen SQL-Cursor, der die Tabelle *dbo.Buch* satzweise durchläuft und den Titel jedes Buches mit einer `PRINT`-Anweisung ausgibt.

```
DECLARE CUR_Buch CURSOR
FOR SELECT ID, ISBN10, ISBN13, Autor, Titel FROM dbo.Buch

DECLARE @ID int, @ISBN10 char(10), @ISBN13 char(14), @Autor varchar(80), @Titel varchar(80)

OPEN CUR_Buch

FETCH NEXT FROM CUR_Buch INTO @ID, @ISBN10, @ISBN13, @Autor, @Titel
```

```
WHILE @@FETCH_STATUS = 0
BEGIN
 PRIN         T @Titel

         FETCH NEXT FROM CUR_Buch INTO @ID, @ISBN10, @ISBN13, @Autor, @Titel
END
CLOSE CUR_Buch
DEALLOCATE CUR_Buch
```

Zu Beginn wird die Cursor-Variable *CUR_Buch* deklariert und dieser gleich die `SELECT`-Abfrage zugeordnet, auf der der Cursor basieren soll. Anschließend wird für jede Spalte der Abfrage auch eine entsprechende lokale Variable deklariert, die später die Werte der gerade gelesenen Zeile enthält. Mit der `OPEN`-Anweisung wird der Cursor geöffnet und anschließend über `FETCH NEXT` der Inhalt der ersten Zeile in die entsprechenden Variablen gelesen.

Nun wird eine `WHILE`-Schleife durchlaufen, solange die Systemvariable `@@FETCH_STATUS=0` ist (was bedeutet, dass noch eine Zeile gelesen werden konnte). Innerhalb der Schleife wird die Variable, die den Titel des Buches enthält, mit einer `PRINT`-Anweisung angezeigt, bevor die nächste Datenzeile wieder in die entsprechenden Variablen gelesen wird.

Wenn die Abfrage komplett durchlaufen ist und damit die Schleife beendet wird, wird zuerst der Cursor geschlossen und anschließend über die `DEALLOCATE`-Anweisung die Cursor-Variable wieder freigegeben.

Anstelle der `PRINT`-Anweisung können innerhalb der `WHILE`-Schleife natürlich auch sehr viel aufwendigere Aktionen erfolgen, zumal die reine Ausgabe der Buchtitel auch über eine einfache `SELECT`-Abfrage möglich gewesen wäre.

Wenn Sie möchten, können Sie das oben gezeigte SQL-Skript einmal eingeben und ausführen oder gar zeilenweise debuggen, um die Funktionsweise des Cursors besser zu verstehen.

Cursor und Trigger kombiniert verwenden

Innerhalb von Triggern lassen sich Cursor gut verwenden, um die temporären Tabellen `INSERTED` und `DELETED` zu durchlaufen. Mit diesem Verfahren lässt sich beispielsweise der *dbo.TRG_ChangeLog*-Trigger aus dem vorigen Abschnitt so erweitern, dass nicht die Anzahl der geänderten Zeilen, sondern eine kommagetrennte Liste der IDs gespeichert wird:

1. Klicken Sie den Trigger *TRG_ChangeLog* im Objekt-Explorer mit der rechten Maustaste an und wählen Sie die Option *Ändern*.
2. Passen Sie den Trigger wie folgt an:

```
USE [MediaBase]
GO
/****** Object:  Trigger [dbo].[TRG_ChangeLog]    Script Date: 06/14/2009 14:45:14 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER TRIGGER [dbo].[TRG_ChangeLog]
   ON  [dbo].[Buch]
   AFTER UPDATE, INSERT, DELETE
AS
BEGIN
```

```
SET    NOCOUNT ON;

   DECLARE @Aenderungen AS varchar(max)

   DECLARE @IDsInserted AS varchar(max) = ''
   DECLARE CUR_Inserted CURSOR FOR SELECT ID FROM INSERTED
   DECLARE @ID int

OPEN    CUR_Inserted
   FETCH NEXT FROM CUR_Inserted INTO @ID

   WHILE @@FETCH_STATUS = 0
BEGI   N
            SET @IDsInserted = @IDsInserted + ltrim(str(@ID)) + ', '

            FETCH NEXT FROM CUR_Inserted INTO @ID
END
CLOS   E CUR_Inserted
DEAL   LOCATE CUR_Inserted

   SET @Aenderungen='Hinzugefügte IDs: ' + @IDsInserted

   INSERT INTO AenderungsLog (Aenderung)
 VALU   ES (@Aenderungen)
END
```

3. Um den neuen Trigger nun sinnvoll testen zu können, sollten natürlich mehrere Zeilen in einer Anweisung hinzugefügt werden. Führen Sie dazu den folgenden SQL-Befehl aus:

```
INSERT INTO dbo.Buch (Titel) VALUES ('TriggerTest1'),('TriggerTest2')
```

4. Anschließend können Sie die neuen Zeilen wieder aus der Tabelle löschen und durch einen Blick in die Tabelle *dbo.Aenderungslog* sicherstellen, dass der angepasste Trigger nun die IDs der beiden neu hinzugefügten Zeilen protokolliert hat.

```
DELETE FROM dbo.Buch WHERE Titel LIKE 'TriggerTest%'
SELECT * FROM dbo.AenderungsLog
```

Hinweis: Nur im Notfall verwenden!

SQL-Cursor gelten zusammen mit den weiter oben beschriebenen Tr iggern als das »Enf ant terrible« der T-SQL-Sprache. Beide können die Gesamtperformance eines Datenban kservers extrem belasten, sodass mögliche Alternativen im Zweifelsfall vorzuziehen sind.

SQL-Cursor lassen sich oft durch geschickt formulierte komplexe SQL-Abfragen ersetzen.

Im Zusammenhang mit Cursorn gibt es noch zahlreiche weitere Optionen und Möglichkeiten, deren detaillierte Darstellung den Rahmen dieses Kapitels sicherlich sprengen würde. Für die meisten Zwecke sollten Sie ohnehin ohne die Verwendung von Cursorn auskommen.

9.6 Übungen zu diesem Kapitel

In diesem Abschnitt finden Sie einige Übungen zu diesem Kapitel. Die richtigen Antworten, Lösungen und den Programmcode finden Sie wie immer auf der Website *www.vsxpress.de.*

Übung 9.1

Erstellen Sie eine gespeicherte Prozedur mit Namen *dbo.USP_MedienAnzeigen()*, die alle Bücher, CDs und DVDs ausgibt, die einen Titel haben, der als Parameter übergeben wird. Der Medientyp (Buch, CD oder DVD) soll dabei als separate Spalte aufgelistet sein.

Übung 9.2

Entwerfen Sie eine benutzerdefinierte Skalarwertfunktion mit Namen *dbo.UFN_LastRelease*, die das letzte Veröffentlichungsjahr eines Autors bzw. Interpreten zurückgibt. Dazu sollen sowohl die Tabelle *dbo.Buch* als auch die Tabelle *dbo.CD* berücksichtigt werden.

Übung 9.3

Schreiben Sie eine benutzerdefinierte Tabellenwertfunktion mit Namen *dbo.UFN_CDTitelVon*, die alle gespeicherten CD-Titel eines als Parameter übergebenen Interpreten zurückgibt.

Übung 9.4

Erweitern Sie den Trigger *dbo.TRG_ChangeLog* so, dass nicht nur die IDs der hinzugefügten Zeilen, sondern auch die der gelöschten Zeilen protokolliert werden.

Übung 9.5

Schreiben Sie eine SQL-Abfrage, die dasselbe Ergebnis (allerdings in einer Ergebnistabelle) liefert wie das folgende cursorbasierte Skript:

```
DECLARE CUR_CD CURSOR
FOR SELECT ID, Titel, Interpret FROM dbo.CD

DECLARE @ID int, @Titel varchar(80), @Interpret varchar(80)

OPEN CUR_CD

FETCH NEXT FROM CUR_CD INTO @ID, @Titel, @Interpret

WHILE @@FETCH_STATUS = 0
```

```
BEGIN
        SELECT @Titel AS Titel, @Interpret AS Interpret, (SELECT COUNT(*) FROM dbo.CDTrack WHERE idCD=@ID) AS
AnzahlTracks

        FETCH NEXT FROM CUR_CD INTO @ID, @Titel, @Interpret
END
CLOSE CUR_CD
DEALLOCATE CUR_CD
```

9.7 Zusammenfassung

In diesem Kapitel haben Sie sowohl die Nutzung von vordefinierten als auch die Erstellung von eigenen gespeicherten Prozeduren und Funktionen kennen gelernt. Viele Probleme lassen sich sowohl mit einer Sicht als auch mit einer gespeicherten Prozedur oder einer Funktion lösen, daher an dieser Stelle eine kleine Übersicht, wo die Unterschiede liegen:

Aufruf, Eingabe und Ausgabe von Daten

- Funktionen (egal welchen Typs) haben einen Rückgabewert.
- Sowohl Sichten als auch Tabellenwertfunktionen geben Datenzeilen zurück und können damit wie Tabellen auch in Abfragen weiterverwendet werden.
- Gespeicherte Prozeduren geben Daten lediglich aus, können aber dazu auch `OUTPUT`-Parameter haben, die vom aufrufenden Skript weiterverwendet werden können.

Komplexität

- Sichten beinhalten ausschließlich Abfragen.
- In gespeicherten Prozeduren kann eigentlich alles geschehen (inklusive Fehlerbehandlung, Aufruf von externen Programmen etc.).
- Funktionen können zwar mehrere Anweisungen enthalten, dürfen aber keine Daten ändern und unterstützen auch keine Fehlerbehandlung.

Performance

- Sichten werden erst mit der Abfrage, in der sie verwendet werden, aufgelöst und dann zusammen mit diesen ausgeführt (es wird dynamisch ein Ausführungsplan für die Gesamtabfrage – inklusive Sicht – ermittelt). Diese Variante ist sinnvoller, wenn Daten (Zeilen oder Spalten), die in der Sicht abgefragt werden, von der aufrufenden Abfrage nicht genutzt werden.
- Für gespeicherte Prozeduren und Funktionen werden eigene Ausführungspläne vorbereitet. Diese werden also getrennt von der aufrufenden Abfrage behandelt. Diese Variante ist beispielsweise dann sinnvoller, wenn die Prozedur oder Funktion häufig mit denselben Parametern aufgerufen wird und das Ergebnis damit zwischengespeichert werden kann, sodass die Prozedur bzw. Funktion weniger häufig durchlaufen werden muss.

Zusätzlich zu den gängigeren Themen wie gespeicherte Prozeduren und Funktionen wurden auch noch exotischere Themen wie Trigger und Cursor behandelt, die sich allerdings beide sehr negativ auf die Gesamtperformance auswirken können und daher nur ganz gezielt und mit äußerster Vorsicht eingesetzt werden sollten.

Unter Umständen kann aber sogar ein DML-Trigger Sinn machen, der einen Cursor verwendet, um alle eingefügten oder gelöschten Zeilen zu durchlaufen.

Mit diesem Kapitel ist der Teil zur Datenbankentwicklung erst einmal abgeschlossen. In den folgenden Kapiteln wird näher auf die Administration von SQL-Datenbanken eingegangen, was gerade im Umfeld der Express Edition oft vernachlässigt wird.

Kapitel 10

Datenbankadministration mit SQL

In diesem Kapitel lernen Sie

- wie man SQL-Skripts im SQL Server Management Studio generieren lassen kann
- wie Sie Datenbanken mit SQL erstellen und entfernen können
- wie Sie Datenbankobjekte wie Tabellen, Indizes etc. mit SQL-Anweisungen erzeugen und löschen können
- was es mit DDL-Triggern auf sich hat

10.1 Skriptgenerierung oder »SQL ist überall«

So banal diese Überschrift klingt, so viel Wahrheit steckt auch darin. Hinter eigentlich allem, was ein SQL Server macht, stecken SQL-Anweisungen. Somit können Sie auch alle Einstellungen, die Sie komfortabel mit dem SQL Server Management Studio vornehmen, alternativ auch direkt als SQL-Befehl eingeben. Bevor wir uns also den SQL-Anweisungen der Data Definition Language (DDL) zuwenden, möchte ich kurz darauf eingehen, welche Möglichkeiten SQL Server bietet, das im Hintergrund generierte SQL zu nutzen oder sogar explizit SQL-Anweisungen für bestimmte Aktionen generieren zu lassen.

Skriptgenerierung aus Dialogfeldern heraus

Wie gerade erwähnt, lässt sich für eigentlich alles, was Sie im SQL Server Management Studio komfortabel über die Benutzeroberfläche machen, auch ein SQL-Skript generieren. Deutlich wird dies dadurch, dass in den meisten Dialogfeldern, die Sie vom SQL Server Management Studio angeboten bekommen, neben dem Hilfe-Symbol auch ein Skript-Symbol zu finden ist. Mit diesem können Sie die im Dialogfeld vorgenommenen Einstellungen auch als SQL-Skript generieren lassen, um diese beispielsweise später auszuführen.

Probieren Sie dies nun aus, indem Sie eine neue Datenbank über die Oberfläche erstellen, dazu dann aber das generierte Skript im Abfrage-Editor ausführen:

1. Stellen Sie im SQL Server Management Studio eine Verbindung zur lokalen Serverinstanz *SQL2008Express* her.

2. Klicken Sie im Objekt-Explorer mit der rechten Maustaste auf den Eintrag *Datenbanken* und wählen Sie die Option *Neue Datenbank,* und das entsprechende Dialogfeld erscheint.
3. Nehmen Sie auf der Seite *Allgemein* folgende Einstellungen vor, allerdings ohne anschließend auf *OK* zu klicken:
   - Datenbankname: MediaBase2
   - Besitzer: <Standard>
   - Zeilendaten (erste Zeile in der Liste mit Datenbankdateien)
     Logischer Name: MediaBase2
     Anfangsgröße (MB): 20
     Automatische Vergrößerung: 10 Prozent, unbeschränkte Vergrößerung (diese Einstellung können Sie vornehmen, wenn Sie auf die Schaltfläche mit den drei Punkten hinter dem entsprechenden Feld klicken)
   - Protokoll (zweite Zeile in der Liste mit Datenbankdateien)
     Logischer Name: MediaBase2_log
     Anfangsgröße (MB): 10
     Automatische Vergrößerung: 10 Prozent, unbeschränkte Vergrößerung

   Wenn Sie möchten, können Sie hier auch noch die Pfade, in denen die Datenbankdateien abgelegt werden, prüfen und bei Bedarf anpassen.

   Die Einstellungen auf den Seiten *Optionen* und *Dateigruppen* können Sie unverändert lassen.
4. Klicken Sie nun in der Symbolleiste des Dialogfeldes auf den kleinen Pfeil zwischen den Schaltflächen für Skript und Hilfe und es klappt ein kleines Menü auf, in dem Sie auswählen können, was mit dem generierten Skript geschehen soll. Im Wesentlichen können Sie das Skript speichern, in die Zwischenablage legen oder – und das ist der Standardfall, der auch ausgeführt wird, wenn Sie direkt auf die Skript-Schaltfläche klicken – das Skript in einem neuen Abfragefenster öffnen. (Die vierte Variante – *Skript für Aktion in Auftrag schreiben* – ist abgeblendet, da diese nur in Zusammenhang mit dem SQL Server Agent Sinn macht, der für die Express Edition nicht verfügbar ist.)

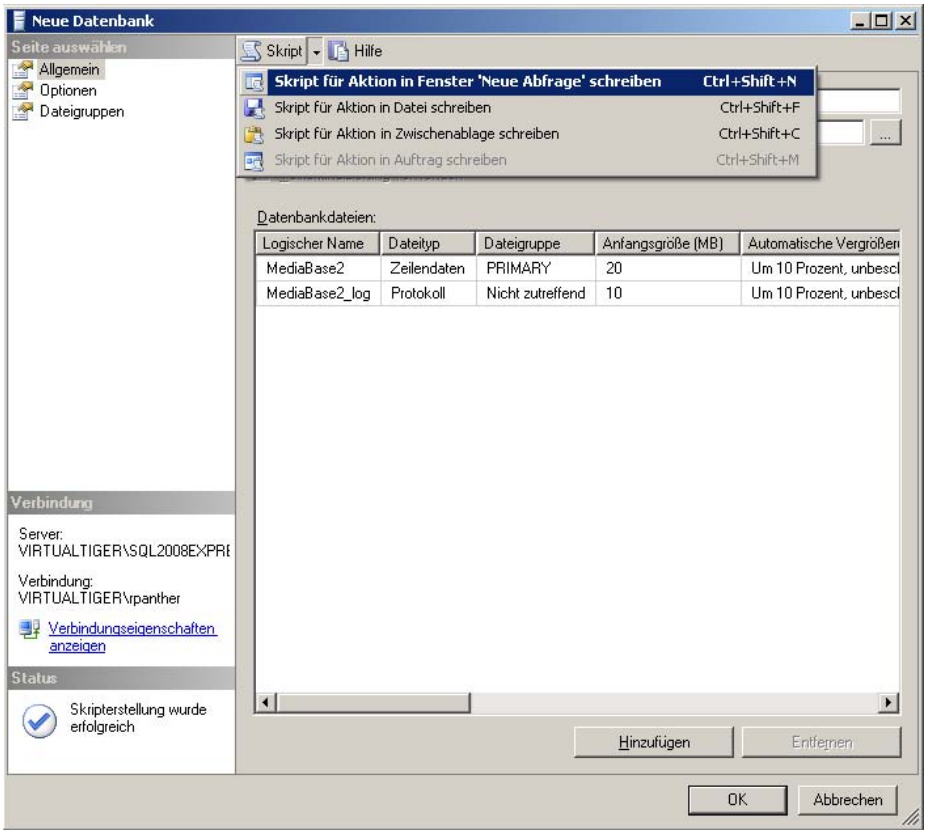


Abbildung 10.1: Menü zur Generierung von SQL-Skripts aus einem Dialogfeld

5. Wählen Sie nun die Standardoption: *Skript für Aktion in Fenster ‚Neue Abfrage' schreiben* und im Hintergrund sehen Sie bereits, dass ein neues Abfragefenster geöffnet und mit einem Skript gefüllt wird.
6. Klicken Sie anschließend auf *Abbrechen*, damit die Datenbank nicht bereits vom Dialogfeld erstellt wird.
7. Schauen Sie sich nun das generierte Skript einmal in Ruhe an:

```
CREATE DATABASE [MediaBase2] ON  PRIMARY
( NAME = N'MediaBase2', FILENAME = N'c:\Programme\Microsoft SQL
Server\MSSQL10.SQL2008EXPRESS\MSSQL\DATA\MediaBase2.mdf' , SIZE = 20480KB , FILEGROWTH = 10%)
 LOG ON
( NAME = N'MediaBase2_log', FILENAME = N'c:\Programme\Microsoft SQL
Server\MSSQL10.SQL2008EXPRESS\MSSQL\DATA\MediaBase2_log.ldf' , SIZE = 10240KB , FILEGROWTH = 10%)
GO
ALTER DATABASE [MediaBase2] SET COMPATIBILITY_LEVEL = 100
GO
ALTER DATABASE [MediaBase2] SET ANSI_NULL_DEFAULT OFF
GO
ALTER DATABASE [MediaBase2] SET ANSI_NULLS OFF
GO
ALTER DATABASE [MediaBase2] SET ANSI_PADDING OFF
```

```
GO
ALTER DATABASE [MediaBase2] SET ANSI_WARNINGS OFF
GO
ALTER DATABASE [MediaBase2] SET ARITHABORT OFF
GO
ALTER DATABASE [MediaBase2] SET AUTO_CLOSE OFF
GO
ALTER DATABASE [MediaBase2] SET AUTO_CREATE_STATISTICS ON
GO
ALTER DATABASE [MediaBase2] SET AUTO_SHRINK OFF
GO
ALTER DATABASE [MediaBase2] SET AUTO_UPDATE_STATISTICS ON
GO
ALTER DATABASE [MediaBase2] SET CURSOR_CLOSE_ON_COMMIT OFF
GO
ALTER DATABASE [MediaBase2] SET CURSOR_DEFAULT  GLOBAL
GO
ALTER DATABASE [MediaBase2] SET CONCAT_NULL_YIELDS_NULL OFF
GO
ALTER DATABASE [MediaBase2] SET NUMERIC_ROUNDABORT OFF
GO
ALTER DATABASE [MediaBase2] SET QUOTED_IDENTIFIER OFF
GO
ALTER DATABASE [MediaBase2] SET RECURSIVE_TRIGGERS OFF
GO
ALTER DATABASE [MediaBase2] SET  DISABLE_BROKER
GO
ALTER DATABASE [MediaBase2] SET AUTO_UPDATE_STATISTICS_ASYNC OFF
GO
ALTER DATABASE [MediaBase2] SET DATE_CORRELATION_OPTIMIZATION OFF
GO
ALTER DATABASE [MediaBase2] SET PARAMETERIZATION SIMPLE
GO
ALTER DATABASE [MediaBase2] SET  READ_WRITE
GO
ALTER DATABASE [MediaBase2] SET RECOVERY SIMPLE
GO
ALTER DATABASE [MediaBase2] SET  MULTI_USER
GO
ALTER DATABASE [MediaBase2] SET PAGE_VERIFY CHECKSUM
GO
USE [MediaBase2]
GO
IF NOT EXISTS (SELECT name FROM sys.filegroups WHERE is_default=1 AND name = N'PRIMARY') ALTER DATABASE
[MediaBase2] MODIFY FILEGROUP [PRIMARY] DEFAULT
GO
```

Neben der `CREATE DATABASE`-Anweisung, die den wichtigsten Teil des Skripts darstellt, wurden vor allem zahlreiche Anweisungen der Form `ALTER DATABASE [MediaBase2] SET ...` generiert. Diese setzen die diversen Eigenschaften, die Sie auf der *Optionen*-Seite des Dialogfeldes *Neue Datenbank* einstellen konnten.

8. Nun könnten Sie das Skript ausführen oder speichern, um es beispielsweise auf einem anderen Server laufen zu lassen. Im Moment können Sie das Fenster aber einfach schließen, da wir ja bereits eine Beispieldatenbank haben.

Alternativ zur Skriptgenerierung aus einem Dialogfeld kann man auch über den Objekt-Explorer Skripts zu fast allen Datenbankobjekten – einschließlich der Datenbank selbst – generieren.

Skriptgenerierung über den Objekt-Explorer

Über den Objekt-Explorer können Sie auch an vielen Stellen Skripts generieren lassen. Dazu müssen Sie nur das entsprechende Objekt mit der rechten Maustaste anklicken und aus dem daraufhin erscheinenden Kontextmenü die Option *Skript für ... als ...* auswählen. Die Ziele für das Skript sind dabei dieselben wie bei der Skriptgenerierung aus einem Dialogfeld: Abfrage-Editor, Datei, Zwischenablage und Agentenauftrag (der hier interessanterweise sogar auswählbar ist, obwohl bei der Express Edition von SQL Server keine Möglichkeit vorhanden ist, diesen Auftrag dann irgendwie weiter zu nutzen).

Probieren wir das einmal für das Erstellen einer Tabelle aus:

1. Stellen Sie im SQL Server Management Studio eine Verbindung zur lokalen Serverinstanz *SQL2008Express* her.
2. Suchen Sie in der Datenbank *MediaBase* die Tabelle *dbo.Buch* und klicken Sie diese mit der rechten Maustaste an.
3. Wählen Sie im Kontextmenü die Option *Skript für Tabelle als/CREATE in/Neues Abfrage-Editor-Fenster* aus.

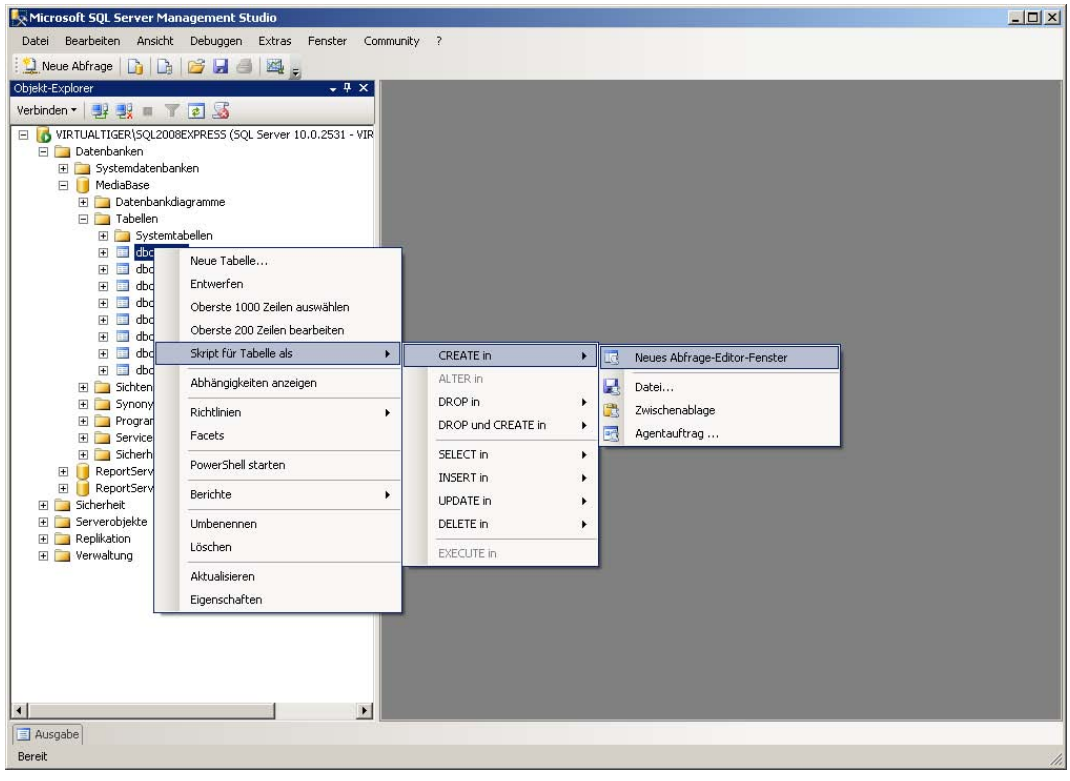


Abbildung 10.2: Generierung von SQL-Skripts über den Objekt-Explorer

4. Es wird ein neues Abfragefenster geöffnet, das mit einem Skript zur Erstellung der Tabelle *dbo.Buch* gefüllt wird. Schauen Sie sich auch dieses Skript in Ruhe an:

```
USE [MediaBase]
GO

/****** Object:  Table [dbo].[Buch]    Script Date: 06/16/2009 00:42:49 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

SET ANSI_PADDING ON
GO

CREATE TABLE [dbo].[Buch](
    [ID] [int] IDENTITY(1,1) NOT NULL,
 [ISB    N10] [char](10) NULL,
 [ISB    N13] [char](14) NULL,
    [Autor] [varchar](80) NULL,
    [Titel] [varchar](80) NULL,
 [Ver    lag] [varchar](80) NULL,
    [Seiten] [int] NULL,
    [Auflage] [tinyint] NULL,
```

```
 [Spr    ache] [varchar](20) NULL,
    [Hardcover] [bit] NULL,
    [Erscheinungsjahr] [int] NULL,
    [Bewertung] [tinyint] NULL,
    [Beschreibung] [varchar](max) NULL,
 [Kat    egorie] [varchar](80) NULL,
CONSTRAINT [PK_Buch] PRIMARY KEY CLUSTERED
(
 [ID]     ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON,
ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

SET ANSI_PADDING OFF
GO

ALTER TABLE [dbo].[Buch] ADD  CONSTRAINT [DF_Buch_Sprache]  DEFAULT ('deutsch') FOR [Sprache]
GO
```

5. Nun könnten Sie das Skript speichern, um es beispielsweise auf einem anderen Server laufen zu lassen. Ein direktes Ausführen würde fehlschlagen, da bereits eine Tabelle mit dem Namen *dbo.Buch* vorhanden ist und diese daher nicht neu erstellt werden kann. Im Moment können Sie das Fenster aber einfach schließen.

Ihnen ist vielleicht aufgefallen, dass auf diesem Wege neben den `CREATE`-Skripts weitere Skriptarten erstellt werden können. So können Sie `DROP`-Skripts erstellen, um die entsprechenden Datenbankobjekte zu löschen, kombinierte `DROP`- und `CREATE`-Skripts, mit denen die Objekte zuerst gelöscht und dann neu angelegt werden, aber auch Skripts für `SELECT`, `INSERT`, `UPDATE` und `DELETE`. Diese erstellen jeweils ein Skriptgerüst für die entsprechende Aktion zur passenden Tabelle, das anschließend noch bearbeitet werden muss. Während das `DELETE`-Skript sehr kurz (und eine Generierung damit eigentlich überflüssig) ist, beinhalten die Skripts für `SELECT`; `INSERT` und `UPDATE` bereits alle Spaltennamen, was unter Umständen sehr viel Tipparbeit sparen kann.

Für gespeicherte Prozeduren können Sie nach demselben Vorgehen auch `EXECUTE`-Skripts erzeugen lassen, die bereits die verfügbaren Parameter mit auflisten.

Hinweis: Skript für Datenbank mit Datenbankobjekten erstellen

Während der gerade demonstrierte Weg lediglich ein Skript für ein Objekt erstellt, gibt es auch eine einfache Möglichkeit, ein Skript generieren zu lassen, das sowohl die Datenbank als auch alle darin enthaltenen Objekte erstellt. Dieses rufen Sie auf, indem Sie die Datenbank im Objekt-Explorer mit der rechten Maustaste anklicken und im Kontextmenü die Option *Tasks/Skripts generieren* auswählen. Daraufhin erscheint ein Assistent, der Sie durch die weiteren Schritte führt.

Skriptgenerierung mit dem Vorlagen-Explorer

Es gibt noch eine weitere Möglichkeit, SQL-Skripts zu generieren. Über das Menü *Ansicht* können Sie den *Vorlagen-Explorer* einblenden, der standardmäßig auf der rechten Seite des Management Studio erscheint. Hier sind nun zu verschiedensten SQL Server-Objekten Skriptvorlagen zu finden, die Sie einfach mit einem Doppelklick aufrufen können.

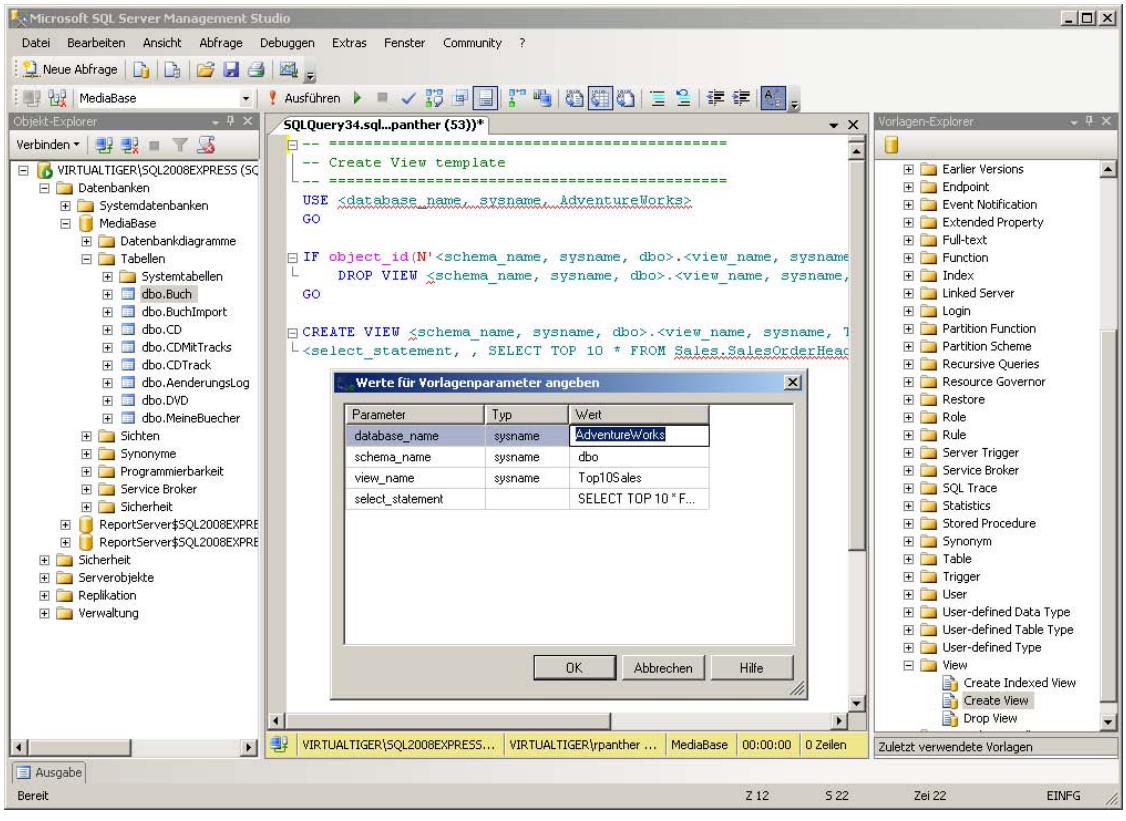


Abbildung 10.3: Der Vorlagen-Explorer im Einsatz

Für die variablen Teile sind dann Platzhalter eingefügt, die Sie entweder von Hand überschreiben können oder aber Sie rufen im *Abfrage*-Menü die Option *Werte für Vorlagenparameter angeben* auf (oder klicken das entsprechende Symbol in der Symbolleiste an). Dadurch erscheint ein Dialogfeld, in dem Sie gezielt nach den einzugebenden Werten gefragt werden.

Hinweis: Skripts aus Datenbankdiagrammen erstellen

An eine vierte Variante zur Skripterstellung soll hier nur kurz erinnert werden, da diese bereits in Kapitel 5.3, *Datenbankdiagramme* behandelt wurde: Wenn Sie in einem Datenbankdiagramm Änderungen vornehmen, diese aber noch nicht speichern, können Sie ein Skript generieren lassen, das genau diese Änderungen beinhaltet.

10.2 Verwalten von Datenbanken

Nach allem Komfort von Management Studio-Oberfläche und Skriptgenerierung wollen wir uns nun ansehen, was sich hinter den generierten SQL-Kommandos der DDL genau verbirgt und wie man diese – wenn man sich auf die wesentlichen Optionen beschränkt – auch relativ schnell eingeben kann.

Hinweis: Generierte Skripts vs. manuell erstellte Skripts

Wie bei den vorhergegangenen Beispielen in diesem Kapitel zu sehen war, können generierte Skripts mitunter recht lang werden. Das liegt daran, dass sie meist alle Optionen beinhalten (selbst die, bei denen die Standardvorgabe übernommen wurde). Wenn Sie dagegen ein Skript – beispielsweise zur Erstellung einer Datenbank – von Hand eingeben, kommen Sie im Normalfall mit erheblich weniger Text aus.

So wie `INSERT`, `UPDATE` und `DELETE` die zentralen Kommandos der Data Manipulation Language (DML) sind, mit denen Sie Zeilen einfügen, ändern oder löschen können, gibt es auch in der Data Definition Language (DDL) drei grundlegende Befehle. Dies sind die Anweisungen `CREATE`, `ALTER` und `DROP`, mit denen Sie Datenbanken anlegen, ändern und löschen können. Mit dem passenden Zusatz können Sie mit diesen drei Anweisungen aber auch alle möglichen anderen Datenbankobjekte wie beispielsweise Tabellen, Sichten, gespeicherte Prozeduren etc. anlegen, ändern und löschen. Doch bleiben wir erst einmal bei den Datenbanken.

Datenbanken erstellen

Um eine Datenbank zu erstellen, verwenden Sie die Anweisung `CREATE DATABASE` gefolgt vom Namen der Datenbank. Danach folgen einige Angaben, die steuern, wo die Datenbankdateien gespeichert werden und wie sie konfiguriert werden. Schauen wir uns dazu einfach noch einmal die entsprechende Anweisung aus dem weiter oben generierten Skript an:

```
CREATE DATABASE [MediaBase2] ON  PRIMARY
( NAME = N'MediaBase2', FILENAME = N'c:\Programme\Microsoft SQL
Server\MSSQL10.SQL2008EXPRESS\MSSQL\DATA\MediaBase2.mdf' , SIZE = 20480KB , FILEGROWTH = 10%)
 LOG ON
( NAME = N'MediaBase2_log', FILENAME = N'c:\Programme\Microsoft SQL
Server\MSSQL10.SQL2008EXPRESS\MSSQL\DATA\MediaBase2_log.ldf' , SIZE = 10240KB , FILEGROWTH = 10%)
GO
```

In den zwei Klammern werden die Einstellungen für die eigentliche Datenbankdatei (in diesem Fall *MediaBase2.mdf*) und die Protokolldatei (*MediaBase2_log.ldf*) festgelegt. Diese umfassen jeweils den logischen Namen (`NAME`), den kompletten Dateipfad (`FILENAME`) sowie die Größe (`SIZE`) und den Faktor, um den die Datei bei Bedarf automatisch vergrößert wird (`FILEGROWTH`).

Hintergrundinfo: Präfix N vor einer Zeichenkette

Das Präfix N vor den Zeichenketten für die Einstellungen `NAME` und `FILENAME` sorgt lediglich dafür, dass die Zeichenkette unverändert übernommen wird und Sonderzeichen wie beispielsweise der Schrägstrich (\) nicht als Escape-Sequenz interpretiert werden.

Die Einstellungen für die Datenbankdatei werden eingeleitet mit den Schlüsselwörtern `ON PRIMARY`, die bewirken, dass die angegebene Datenbankdatei in der primären Dateigruppe angelegt wird. Im Umfeld des SQL Server Express werden Sie im Normalfall nur mit einer Dateigruppe pro Datenbank arbeiten, sodass Sie dies als Standardeinstellung ansehen können.

Die Einstellungen für die Protokolldatei werden durch die Schlüsselwörter `LOG ON` eingeleitet.

Vereinfacht kann man sich die Syntax merken als:

```
CREATE DATABASE Datenbankname
ON PRIMARY (Datenbankdatei-Einstellungen)
LOG ON (Logdatei-Einstellungen)
```

Die `CREATE DATABASE`-Anweisung allein reicht völlig aus, um eine funktionierende Datenbank anzulegen, mit der Sie direkt arbeiten können. Probieren wir das einmal aus:

1. Stellen Sie im SQL Server Management Studio eine Verbindung zur lokalen Serverinstanz *SQL2008Express* her und öffnen Sie ein neues Abfragefenster.
2. Führen Sie nun die folgende Anweisung aus, um eine neue Datenbank zu erstellen:

```
CREATE DATABASE CrashTestDummy ON  PRIMARY
( NAME = N'CrashTestDummy', FILENAME = N'c:\CrashTestDummy.mdf' , SIZE = 20480KB , FILEGROWTH = 10%)
 LOG ON
( NAME = N'CrashTestDummy_log', FILENAME = N'c:\CrashTestDummy_log.ldf' , SIZE = 10240KB , FILEGROWTH = 10%)
```

Hinweis: Pfade für Datenbank- und Protokolldateien

Der Einfachheit halber – und nicht zuletzt, um Ihnen Tipparbeit zu sparen – wurde hier das Stammverzeichnis von *C:* als Pfad für die Dateien gewählt, da diese in einer späteren Übung ohnehin wieder gelöscht werden.

In einer produktiven Umgebung sollten Sie hierfür ein separates Datenbankverzeichnis nutzen (evtl. auch ein weiteres Verzeichnis für die Protokolldatei), das möglichst sogar auf einer anderen Partition als der Systempartition liegt.

3. Aktualisieren Sie die Liste der Datenbanken im Objekt-Explorer, indem Sie mit der rechten Maustaste auf den Knoten *Datenbanken* klicken und anschließend *Aktualisieren* auswählen. Nun sollte auch hier die Datenbank *CrashTestDummy* zu sehen sein.
4. Klicken Sie die Datenbank mit der rechten Maustaste an und wählen Sie die Option *Eigenschaften*. Hier können Sie überprüfen, mit welchen Einstellungen die Datenbank angelegt wurde.
5. Schauen Sie außerdem mit dem Windows-Explorer einmal in das Stammverzeichnis von Laufwerk C:\. Sie werden dort die beiden Dateien (Datenbankdatei und Protokolldatei) in der vorgegebenen Größe vorfinden.

Auch wenn die so erstellte Datenbank bereits komplett einsatzbereit ist, möchte ich vorher noch ein paar Einstellungen anpassen.

Datenbanken anpassen

Zum Ändern der Datenbank verwenden Sie die Anweisung `ALTER DATABASE`. Das bezieht sich aber nicht auf die in der Datenbank enthaltenen Objekte oder gar deren Daten, denn für Letzteres gibt es ja die

Befehle der DML. Stattdessen können Sie mit `ALTER DATABASE` die Einstellungen der Datenbank ändern. Wie Sie bereits in dem zu Beginn des Kapitels generierten Skript gesehen haben, wird die `ALTER DATABASE`-Anweisung meist in Kombination mit dem Schlüsselwort `SET` verwendet, um anschließend einer bestimmten Einstellung einen Wert zuzuweisen. Über die folgende Anweisung können Sie beispielsweise die Kompatibilität zu einer älteren SQL-Version einschalten:

```
ALTER DATABASE CrashTestDummy SET COMPATIBILITY_LEVEL = 90
```

Dabei entspricht die angegebene Zahl der internen Versionsnummer ohne den Punkt. 90 steht also für Version 9.0 was wiederum für SQL Server 2005 steht. Die interne Versionsnummer für SQL 2008 ist 10.0, hier wäre also der Wert 100 anzugeben, der aber für neue Datenbanken natürlich auch voreingestellt ist.

Die folgende Anweisung versetzt die Datenbank in den schreibgeschützten Modus:

```
ALTER DATABASE CrashTestDummy SET READ_ONLY
```

Die Standardeinstellung können Sie durch Setzen des ursprünglichen Wertes wiederherstellen:

```
ALTER DATABASE CrashTestDummy SET  READ_WRITE
```

Auch um die Datenbank für administrative Zwecke in den Einbenutzerbetrieb umzuschalten, gibt es eine Einstellung:

```
ALTER DATABASE CrashTestDummy SET SINGLE_USER
```

Diese Einstellung lässt sich mit folgender Anweisung wieder aufheben:

```
ALTER DATABASE CrashTestDummy SET MULTI_USER
```

Es gibt noch zahlreiche weitere Einstellungen, deren Aufzählung hier sicherlich zu weit führen würde. Sie finden diese allerdings einerseits in dem generierten Skript weiter oben in diesem Kapitel. Alternativ können Sie die Einstellungen auch über die Seite *Optionen* des Dialogfeldes *Datenbankeigenschaften* einsehen. Hier sind aus den Auswahllisten dann auch die alternativen Werte ersichtlich.

Neben diesen Einstellungen bietet die `ALTER DATABASE`-Anweisung aber noch weitere Möglichkeiten. So können Sie mit der folgenden Anweisung die Sortierreihenfolge ändern (hier in den französischen Standard):

```
ALTER DATABASE CrashTestDummy COLLATE French_CI_AI
```

Eine andere Variante der Anweisung kann verwendet werden, um den Namen der Datenbank nachträglich zu ändern:

```
ALTER DATABASE CrashTestDummy MODIFY NAME = CrashDummy
```

Dazu gibt es noch eine Reihe weiterer Varianten, mit denen die Einstellungen der Dateien angepasst oder neue Dateigruppen hinzugefügt werden können.

Datenbanken löschen

Der einfachste SQL-Befehl im Zusammenhang mit Datenbanken ist die Anweisung `DROP DATABASE`. Diese erhält als Parameter lediglich den Datenbanknamen und die Datenbank wird gelöscht.

```
DROP DATABASE datenbankname
```

Probieren Sie diese Anweisung nun aus, um die gerade erstellte *CrashTestDummy*-Datenbank wieder zu entfernen:

1. Stellen Sie im SQL Server Management Studio eine Verbindung zur lokalen Serverinstanz *SQL2008Express* her und öffnen Sie ein neues Abfragefenster.
2. Führen Sie nun die folgende Anweisung aus, mit der die Datenbank wieder entfernt wird:

   ```
   DROP DATABASE CrashTestDummy
   ```
3. Dies löscht einerseits die Datenbank vom Server (damit dies im Objekt-Explorer sichtbar wird, müssen Sie die Ansicht wieder aktualisieren), aber auch die für die Datenbank verwendeten Dateien werden wieder entfernt, wie Sie leicht mit dem Windows-Explorer nachprüfen können.

10.3 Verwalten von Datenbankobjekten

So wie Sie die `CREATE`-, `ALTER`- und `DROP`-Anweisungen auf Datenbanken angewendet haben, lassen sich damit auch Objekte innerhalb einer Datenbank bearbeiten.

Tabellen

Die sicherlich wichtigsten Datenbankobjekte sind dabei natürlich die Tabellen, da in diesen die Daten gespeichert werden. Um eine Tabelle mit der Data Definition Language zu erstellen, verwenden Sie eine Anweisung der folgenden Form:

```
CREATE TABLE tabellenname (spaltendefinition)
```

Dabei ist die Spaltendefinition eine kommagetrennte Liste aus Spaltennamen, denen nach einem Leerzeichen der Datentyp folgt sowie die optionale Angabe, ob die Spalte `NULL` sein darf oder nicht (`NOT NULL`).

Beispiel:

```
CREATE TABLE dbo.Adresse
         (ID int NOT NULL,
         Name varchar(80) NOT NULL,
         Strasse varchar(80) NULL,
         PLZ char(5) NULL,
         Wohnort varchar(80) NULL)
```

Möchten Sie für eine Spalte eine Identitätsspezifikation festlegen, geben Sie das Schlüsselwort `IDENTITY` gefolgt von Startwert und Schrittweite (in Klammern) hinter dem entsprechenden Datentyp an:

```
ID int IDENTITY(1,1) NOT NULL
```

Außerdem lassen sich direkt in der `CREATE TABLE`-Anweisung `CONSTRAINTS` festlegen, was vor allem für die Definition des Primärschlüssels sinnvoll ist.

```
CONSTRAINT PK_Adresse PRIMARY KEY (ID)
```

Insgesamt ergibt sich damit die folgende Anweisung:

```
CREATE TABLE dbo.Adresse
         (ID int IDENTITY(1,1) NOT NULL,
```

```
Name varchar(80) NOT NULL,
Strasse varchar(80) NULL,
PLZ char(5) NULL,
Wohnort varchar(80) NULL,
CONSTRAINT PK_Adresse PRIMARY KEY (ID))
```

Die `ALTER TABLE`-Anweisung kann beispielsweise genutzt werden, um der Tabelle nachträglich neue Spalten hinzuzufügen.

```
ALTER TABLE dbo.Adresse ADD Geburtstag datetime
```

Hinweis: Neue Spalten an beliebiger Position hinzufügen

Tabellenspalten, die mit der `ALTER TABLE`-Anweisung hinzugefügt werden, werden immer als letzte Spalte hinten angehängt. Wenn Sie eine Spalte an beliebiger Position einfügen möchten, benutzen Sie den Entwurfsmodus (im Objekt-Explorer Tabelle mit der rechten Maustaste anklicken, dann *Entwerfen* auswählen). Hier können Sie die Spaltenreihenfolge beliebig verändern und beim Speichern wird die Tabelle entsprechend umorganisiert, indem die Daten erst in eine Hilfstabelle mit der neuen Spaltenreihenfolge kopiert werden. Anschließend wird die Quelltabelle gelöscht und die Hilfstabelle umbenannt. Dies geschieht allerdings automatisch, sodass Sie sich nicht selbst darum kümmern müssen.

Auch Fremdschlüssel lassen sich mithilfe von DDL-Anweisungen erstellen. Da der einzig sinnvolle Fremdschlüssel für unsere Beispieldatenbank aber bereits existiert, beginnen wir ausnahmsweise einmal mit dem Löschen eines Fremdschlüssels und legen diesen anschließend wieder neu an:

1. Stellen Sie im SQL Server Management Studio eine Verbindung zur lokalen Serverinstanz *SQL2008Express* her und öffnen Sie ein neues Abfragefenster, das mit der Datenbank *MediaBase* verbunden ist.
2. Führen Sie nun die folgende Anweisung aus, um den Fremdschlüssel *FK_CDTrack_CD* (der von der Tabelle *dbo.CDTrack* auf die Tabelle *dbo.CD* verweist) zu entfernen:

   ```
   ALTER TABLE dbo.CDTrack
   DROP CONSTRAINT FK_CDTrack_CD
   ```

   Das `DROP CONSTRAINT` resultiert daher, dass ein Fremdschlüssel für SQL Server eine Sonderform eines Constraints (also einer Einschränkung) ist und daher in den SQL-Anweisungen auch als Einschränkung behandelt wird.
3. Im Objekt-Explorer können Sie nun im Zweig *MediaBase/Tabellen/dbo.CDTrack/Schlüssel* prüfen, dass der Fremdschlüssel wirklich entfernt wurde (vorher die Ansicht im Objekt-Explorer aktualisieren).
4. Führen Sie nun die folgende Anweisung aus, um den Fremdschlüssel *FK_CDTrack_CD* wieder neu anzulegen:

   ```
   ALTER TABLE dbo.CDTrack
   WITH CHECK ADD  CONSTRAINT FK_CDTrack_CD
   FOREIGN KEY(idCD)
   REFERENCES dbo.CD(ID)
   ```

Das Löschen einer Tabelle geht denkbar einfach, indem man die Anweisung `DROP TABLE` gefolgt vom Tabellennamen angibt:

```
DROP TABLE dbo.Adresse
```

Indizes

Auch wenn Sie mit dem Erstellen der Tabelle meist gleichzeitig den Primärschlüssel und damit in der Regel auch einen gruppierten Index erstellen, ist es bei den meisten Tabellen sinnvoll, noch zusätzliche (nicht gruppierte Indizes) zu erstellen. Dies geht mit der `CREATE INDEX`-Anweisung relativ einfach, wenn Sie sich an folgende Grundform halten:

```
CREATE INDEX indexname ON tabellenname (feldliste)
```

Erstellen wir nun einen Index für die Tabelle *dbo.Buch*, der die Kombination aus *Verlag*, *Erscheinungsjahr*, *Autor* und *Titel* beinhaltet:

1. Stellen Sie im SQL Server Management Studio eine Verbindung zur lokalen Serverinstanz *SQL2008Express* her und öffnen Sie ein neues Abfragefenster, das mit der Datenbank *MediaBase* verbunden ist.
2. Führen Sie nun die folgende Anweisung aus, um den neuen Index zu erstellen:

   ```
   CREATE INDEX IX_VerlagErscheinungsjahrAutorTitel
   ON dbo.Buch
       (Verlag, Erscheinungsjahr, Autor, Titel)
   ```

Sie erinnern sich vielleicht, dass ein Index intern in einer Baumstruktur gespeichert wird. Durch Datenänderungen wird dieser Indexbaum nach und nach weniger balanciert (man sagt hierzu auch der Index wird fragmentiert). Um dem entgegenzuwirken, kann man den Index mit der `ALTER INDEX`-Anweisung wieder ausbalancieren. Dazu gibt es zwei Varianten:

```
ALTER INDEX indexname ON tabellenname REORGANIZE
```

und

```
ALTER INDEX indexname ON tabellenname REBUILD
```

Mit der Variante `REORGANIZE` wird die Blattebene des Index neu organisiert, mit `REBUILD` wird der Index komplett neu aufgebaut. Das dauert zwar etwas länger, sorgt aber für einen noch besser ausbalancierten (und damit effektiver nutzbaren) Index.

Anstelle des Indexnamens können Sie auch das Schlüsselwort `ALL` angeben, um alle Indizes einer Tabelle zu bearbeiten. Da die Tabelle *dbo.Buch* inzwischen einiges Indizes hat, nutzen wir hier diese Variante, um all diese Indizes neu zu erstellen:

1. Stellen Sie im SQL Server Management Studio eine Verbindung zur lokalen Serverinstanz *SQL2008Express* her und öffnen Sie ein neues Abfragefenster, das mit der Datenbank *MediaBase* verbunden ist.
2. Führen Sie nun die folgende Anweisung aus, um alle Indizes der Tabelle *dbo.Buch* neu aufzubauen:

   ```
   ALTER INDEX ALL ON dbo.Buch REBUILD
   ```

Alternativ zum `REBUILD` könnten Sie natürlich die Indizes auch mit der `DROP`-Anweisung löschen und anschließend mit `CREATE INDEX` neu anlegen, dafür müssten Sie aber einerseits alle Indizes einzeln bearbeiten und andererseits die `INDEX`-Definitionen vorliegen haben.

Best Practices: Regelmäßige Indexpflege

Da der interne Abfrageoptimierer (der die Ausführungspläne für SQL-Abfragen erstellt) stark fragmentierte Indizes nicht nutzt, ist es wichtig, alle Indizes regelmäßig zu reorganisieren oder gar neu aufzubauen. Hier hat es sich bewährt, ein SQL-Skript zu bauen, das einen `REBUILD` aller Indizes durchführt und diese regelmäßig (je nach Aktivität der Datenbank, maximal einmal pro Woche) auszuführen. Mit den größeren Editionen von SQL Server lässt sich diese regelmäßige Ausführung mithilfe des SQL Server Agents auch automatisieren. Beim SQL Server Express können Sie das Skript aber mithilfe des Kommandozeilentools *SQLCMD* und den *Geplanten Tasks* des Windows-Betriebssystems ebenso zeitgesteuert automatisch ausführen lassen.

Am einfachsten ist natürlich wieder das Löschen eines Index. Hier wird lediglich die Anweisung `DROP INDEX` gefolgt vom Namen des Index benötigt:

```
DROP INDEX IX_VerlagErscheinungsjahrAutorTitel
```

Sichten, Funktionen, gespeicherte Prozeduren und Trigger

Da sowohl Sichten als auch Funktionen, gespeicherte Prozeduren und Trigger ohnehin nur aus SQL-Skripts bestehen, ist die Bearbeitung derselben über die DDL verhältnismäßig einfach. Sobald Sie eines der genannten Objekte über den Objekt-Explorer oder auch über den Vorlagen-Explorer erstellen, erhalten Sie ein Skriptgerüst als Vorlage für eine `CREATE`-Anweisung.

Beispiel:

```
CREATE VIEW dbo.VW_Dummy AS SELECT * FROM dbo.Buch WHERE Erscheinungsjahr > 1980
```

Um nun ein solches Objekt im Nachhinein anzupassen, können Sie dasselbe Skript verwenden und müssen dabei nur das `CREATE` gegen ein `ALTER` austauschen. Bei Ausführung des Skripts wird das entsprechende Objekt dann ohnehin gelöscht und neu angelegt.

Beispiel:

```
ALTER VIEW dbo.VW_Dummy AS SELECT * FROM dbo.Buch WHERE Erscheinungsjahr > 1990
```

Das Entfernen mit der `DROP`-Anweisung ist natürlich noch einfacher, da hier nach dem Schlüsselwort `DROP` und dem Objekttyp nur noch der Name des Objekts angegeben werden muss.

Beispiel:

```
DROP VIEW dbo.VW_Dummy
```

Dasselbe Prinzip gilt auch für Funktionen, gespeicherte Prozeduren und Trigger. Entsprechende Beispiele hierzu sind in Kapitel 9 zu finden.

10.4 DDL-Trigger

Die generelle Arbeitsweise von Triggern haben Sie bereits im vorigen Kapitel kennengelernt. Wie bei den dort vorgestellten DML-Triggern handelt es sich auch bei DDL-Triggern um eine Sonderform der gespeicherten Prozeduren, die beim Eintreten bestimmter Ereignisse automatisch gestartet werden.

Der Unterschied liegt allerdings darin, dass die auslösenden Ereignisse bei den DDL-Triggern keine Datenänderungsoperationen (DML) sind, sondern Anweisungen, mit denen die Datenstrukturen geändert werden (DDL).

Dabei wird – je nach auslösenden Ereignissen – zwischen Servertriggern und Datenbanktriggern unterschieden. Letztere finden Sie im Objekt-Explorer bei der jeweiligen Datenbank unter *Programmierbarkeit/Datenbanktrigger*. Servertrigger dagegen sind unter *Serverobjekte/Trigger* aufgelistet.

Servertrigger

Servertrigger beziehen sich auf Änderungen an serverweit bekannten Objekten wie beispielsweise das Erstellen, Verändern oder Löschen von Datenbanken oder Logins. Hierzu stehen folgende Ereignisse zur Verfügung:

Tabelle 10.1: Serverseitige Triggerereignisse

Objekt	Ereignisse
Anmeldung	CREATE_LOGIN, ALTER_LOGIN, DROP_LOGIN
Datenbank	CREATE_DATABASE, ALTER_DATABASE, DROP_DATABASE
HTTP-Endpoint CR	EATE_HTTP_ENDPOINT, DROP_HTTP_ENDPOINT
Serverzugriff G	RANT_SERVER_ACCESS, DENY_SERVER_ACCESS, REVOKE_SERVER_ACCESS
Zertifikat CR	EATE_CERT, ALTER_CERT, DROP_CERT

Die Grundform der Anweisung, um einen Servertrigger zu erstellen, ist die folgende:

```
CREATE TRIGGER triggername ON ALL SERVER
FOR ereignisse
AS
BEGIN
  ...
END
```

Wenn Sie mehrere serverseitige Ereignisse in einem Trigger abfangen wollen, ohne alle einzeln aufzuzählen, können Sie stattdessen vordefinierte DDL-Ereignisgruppen angeben. Die Ereignisgruppe `DDL_SERVER_LEVEL_EVENTS` fasst alle serverseitigen Triggerereignisse zusammen.

Datenbanktrigger

Bei Datenbanktriggern beziehen sich die Ereignisse auf Strukturänderungen innerhalb der Datenbank. Hierzu zählt das Erstellen, Ändern oder Löschen von Tabellen, Sichten und vielem mehr. Insgesamt stehen folgende Ereignisse zur Verfügung:

Tabelle 10.2: Datenbankseitige Triggerereignisse

Objekt	Ereignisse
Anwendungsrolle CR	EATE_APPLICATION_ROLE, ALTER_APPLICATION_ROLE, DROP_APPLICATION_ROLE
Assembly	CREATE_ASSEMBLY, ALTER_ASSEMBLY, DROP_ASSEMBLY
Benutzer CR	EATE_USER, ALTER_USER, DROP_USER
Contract CR	EATE_CONTRACT, ALTER_CONTRACT, DROP_CONTRACT
Database	GRANT_DATABASE, DENY_DATABASE, REVOKE_DATABASE
Datentyp CRE	ATE_TYPE, DROP_TYPE
Ereignisbenachrichtigung CR	EATE_EVENT_NOTIFICATION, DROP_EVENT_NOTIFICATION
Funktion CR	EATE_FUNCTION, ALTER_FUNCTION, DROP_FUNCTION
gesp. Prozedur	CREATE_PROCEDURE, ALTER_PROCEDURE, DROP_PROCEDURE
Index	CREATE_INDEX, ALTER_INDEX, DROP_INDEX
MessageType CR	EATE_MESSAGE_TYPE, ALTER_ MESSAGE_TYPE, DROP_ MESSAGE_TYPE
Partitiongunktion CR	EATE_PARTITION_FUNCTION, ALTER_PARTITION_FUNCTION, DROP_PARTITION_FUNCTION
Partitionschema CR	EATE_PARTITION_SCHEME, ALTER_PARTITION_SCHEME, DROP_PARTITION_SCHEME
Queue CRE	ATE_QUEUE, ALTER_QUEUE, DROP_QUEUE
RemoteServiceBinding CR	EATE_REMOTE_SERVICE_BINDING, ALTER_REMOTE_SERVICE_BINDING, DROP_REMOTE_SERVICE_BINDING
Rolle CRE	ATE_ROLE, ALTER_ROLE, DROP_ROLE
Route CR	EATE_ROUTE, ALTER_ROUTE, DROP_ROUTE
Schema	CREATE_SCHEMA, ALTER_SCHEMA, DROP_SCHEMA
SecExpr CRE	ATE_SECEXPR, DROP_SECEXPR
Service CR	EATE_SERVICE, ALTER_SERVICE, DROP_SERVICE
Sicht CR	EATE_VIEW, ALTER_VIEW, DROP_VIEW
Statistiken CR	EATE_STATISTICS, UPDATE_STATISTICS, DROP_STATISTICS
Synonym CR	EATE_SYNONYM, DROP_SYNONYM
Tabelle	CREATE_TABLE, ALTER_TABLE, DROP_TABLE
Trigger	CREATE_TRIGGER, ALTER_TRIGGER, DROP_TRIGGER
XML-Schema CRE	ATE_XML_SCHEMA, ALTER_XML_SCHEMA, DROP_XML_SCHEMA

Die Grundform der Anweisung, um einen Datenbanktrigger zu erstellen, ist die folgende:

```
CREATE TRIGGER triggername ON DATABASE
FOR ereignisse
AS
BEGIN
  ...
END
```

Wie bei den serverseitigen Ereignissen können Sie auch hier – anstatt einzelne Ereignisse aufzuzählen – vordefinierte DDL-Ereignisgruppen nutzen. Die Ereignisgruppe `DDL_DATABASE_LEVEL_EVENTS` fasst alle datenbankseitigen Triggerereignisse zusammen.

Was wurde eigentlich geändert?

Während bei DML-Triggern die gelöschten und eingefügten Zeilen im Trigger abgefragt werden können, stellen DDL-Trigger über die Funktion `EVENTDATA()` eine XML-Struktur zur Verfügung, in der die Datenstrukturänderung beschrieben ist. Je nach auslösender Aktion kann diese variieren, ein paar Elemente sind allerdings immer gleich:

- *PostTime* – Zeitpunkt, zu dem das Ereignis ausgelöst wurde
- *SPID* – SQL Server-Prozess-ID des Prozesses, der das Ereignis ausgelöst hat
- *ComputerName* – Name des Computers, der das Ereignis ausgelöst hat

Für datenbankseitige Trigger kommen noch folgende dazu:

- *DatabaseName* – Name der Datenbank, die das Ereignis ausgelöst hat
- *UserName* – Name des Benutzers, der das Ereignis ausgelöst hat
- *LoginName* – Name der Anmeldung, die das Ereignis ausgelöst hat

Diese Elemente werden – je nach Ereignis – noch durch ereignisspezifische Angaben ergänzt.

Erstellen Sie nun einen Datenbanktrigger, der alle Strukturänderungen der Datenbank *MediaBase* in einer Tabelle *dbo.DDLAenderungsLog* protokolliert. Die Logtabelle enthält dabei die klassischen Antworten auf die Fragen: Wer?, Wann?, Was?

1. Stellen Sie im SQL Server Management Studio eine Verbindung zur lokalen Serverinstanz *SQL2008Express* her und öffnen Sie im Objekt-Explorer die Datenbank *MediaBase*.
2. Erstellen Sie nun eine neue Tabelle mit Namen *dbo.DDLAenderungsLog*, die folgende Felder enthält:

   Tabelle 10.3: Die Felder der Tabelle *dbo.DDLAenderungsLog*

Spaltenname	Datentyp	NULL-Werte zulassen	Sonstiges
ID int		nein	Primärschlüssel
User varchar(128	)	nein	Standardwert: system_user
Zeitpunkt datetime		nein	Standardwert: getdate()
Aenderung XM	L	ja	

   Für die Spalte *ID* sollte auch eine Identitätsspezifikation festgelegt werden, sodass man diese nicht manuell setzen muss.
3. Geben Sie im Abfragefenster die folgende Anweisung ein, um einen DDL-Trigger zu erzeugen:

```
CREATE TRIGGER TRG_Datenbank ON DATABASE
FOR DDL_DATABASE_LEVEL_EVENTS
AS
BEGIN
 PRIN   T 'Datenbank-Änderung!'
```

```
    INSERT INTO MediaBase.dbo.DDLAenderungsLog (Aenderung) VALUES (EVENTDATA())
END
```

4. Führen Sie nun ein paar Änderungen an der Datenbankstruktur durch:

```
CREATE VIEW dbo.VW_EinBuch
AS SELECT TOP 1 * FROM dbo.Buch

DROP VIEW dbo.VW_EinBuch
```

5. Überprüfen Sie anschließend durch einen Blick in die Tabelle *dbo.DDLAenderungsLog*, ob die Änderungen auch korrekt protokolliert wurden. Wenn Sie die Tabelle über den Objekt-Explorer geöffnet haben, können Sie das XML-Feld anklicken, um dessen kompletten Inhalt zu sehen. Für den ersten Eintrag in der Tabelle erhält man beispielsweise folgende XML-Daten:

```
<EVENT_INSTANCE>
  <EventType>CREATE_VIEW</EventType>
  <PostTime>2009-06-19T23:33:27.330</PostTime>
  <SPID>53</SPID>
  <ServerName>VIRTUALTIGER\SQL2008EXPRESS</ServerName>
  <LoginName>VIRTUALTIGER\rpanther</LoginName>
  <UserName>dbo</UserName>
  <DatabaseName>MediaBase</DatabaseName>
  <SchemaName>dbo</SchemaName>
  <ObjectName>VW_EinBuch</ObjectName>
  <ObjectType>VIEW</ObjectType>
  <TSQLCommand>
    <SetOptions ANSI_NULLS="ON" ANSI_NULL_DEFAULT="ON" ANSI_PADDING="ON" QUOTED_IDENTIFIER="ON"
ENCRYPTED="FALSE" />
    <CommandText>CREATE VIEW dbo.VW_EinBuch
AS SELECT TOP 1 * FROM dbo.Buch
</CommandText>
  </TSQLCommand>
</EVENT_INSTANCE>
```

6. Zum Schluss können Sie den Trigger mit der folgenden Anweisung wieder entfernen:

```
DROP TRIGGER TRG_Datenbank ON DATABASE
```

Hinweis: FOR, AFTER & INSTEAD OF

Wie bei den DML-Triggern können Sie auch bei der Definition eines DDL-Triggers anstelle des Schlüsselwortes `FOR` das Schlüsselwort `AFTER` verwenden, was sich aber nicht auf die Funktionsweise des Triggers auswirkt. Die Variante `INSTEAD OF` ist dagegen für DDL-Trigger nicht verfügbar.

10.5 Übungen zu diesem Kapitel

In diesem Abschnitt finden Sie einige Übungen zu diesem Kapitel. Die richtigen Antworten, Lösungen und den Programmcode finden Sie wie immer auf der Website *www.vsxpress.de.*

Übung 10.1

Formulieren Sie eine `CREATE TABLE`-Anweisung, um die in Abschnitt 10.4, *DDL-Trigger* verwendete Tabelle *dbo.DDLAenderungsLog* zu erstellen.

Übung 10.2

Schreiben Sie ein SQL-Skript, das alle Indizes für die Tabellen *dbo.Buch*, *dbo.CD*, *dbo.CDTrack* und *dbo.DVD* neu erstellt. Dieses Skript können Sie für eine regelmäßige Indexpflege verwenden.

Übung 10.3

Erstellen Sie einen DDL-Trigger *TRG_MediaBase_Views*, der jedes Erstellen, Ändern oder Löschen einer Sicht in der Datenbank *MediaBase* in der Tabelle *dbo.DDLAenderungsLog* protokolliert.

10.6 Zusammenfassung

Sie haben in diesem Kapitel gelernt, dass eigentlich alle Operationen, die Sie mit dem SQL Server Management Studio vornehmen, intern in SQL umgesetzt werden, bevor der SQL Server diese umsetzen kann. In diesem Zusammenhang haben Sie auch verschiedene Arten der Skriptgenerierung kennengelernt:

- Skriptgenerierung aus Dialogfeldern
- Skriptgenerierung über den Objekt-Explorer
- Skriptgenerierung mit dem Vorlagen-Explorer

Dazu haben Sie die wichtigsten DDL-Befehle kennengelernt und damit sowohl eine Datenbank als auch eine Tabelle mit Indizes darin erstellt.

Am Schluss des Kapitels haben Sie gelernt, wie man mithilfe von DDL-Triggern automatisiert auf Änderungen an den Datenstrukturen reagieren kann.

Für den Rest des Buches werde ich – sofern eine Operation über die Oberfläche des SQL Server Management Studios vorgenommen wird – auch die dazugehörige SQL-Anweisung mit angeben, sodass Sie die Aufgabe alternativ durch Eingabe der SQL-Anweisung im Abfrage-Editor ausführen können.

Kapitel 11

Benutzer, Rollen und Rechte

In diesem Kapitel lernen Sie

- wodurch sich Anmeldungen und Benutzer voneinander unterscheiden
- welche Authentifizierungsmodi es gibt
- wie Sie Benutzern Rechte erteilen oder entziehen
- wie Sie die Rechtevergabe durch Rollen vereinfachen können
- wie Sie Schemas für einfachere Rechtevergabe und bessere Übersicht in der Datenbank verwenden

11.1 Das SQL Server-Rechtesystem

Bisher haben wir ausschließlich mit maximalen Berechtigungen auf dem gesamten Datenbankserver gearbeitet, da der Benutzer verwendet wurde, der den Datenbankserver auch installiert hat. Das ist sicherlich völlig ausreichend, wenn Sie die Datenbank bzw. die dazugehörende Anwendung nur von einem Benutzer verwendet wird. Auch während der Entwicklung einer Anwendung kann man eine Zeit lang so verfahren. Aber spätestens dann, wenn mehrere Benutzer aktiv mit der Datenbank arbeiten oder die Datenbankanwendung sogar an die Endanwender verteilt wird, sollte man sich ernsthaft Gedanken über ein ausgefeilteres Sicherheitskonzept machen.

SQL Server unterscheidet dabei zwischen Anmeldungen (Login) und Datenbankbenutzern (Database User). Beide zusammen (insbesondere aber die Anmeldungen) ergeben die Authentifizierung, mit der die Frage geklärt wird, wer der Benutzer ist, bzw. durch Mechanismen wie Passwortabfragen etc. sichergestellt wird, dass der Benutzer auch der ist, für den er sich ausgibt.

Darauf aufbauend erfolgt dann später die Autorisierung, also die Verwaltung der Rechte für Anmeldungen und Benutzer.

11.2 Anmeldungen und Authentifizierung

Der eigentliche Verbindungsaufbau zum SQL Server geschieht über Anmeldungen. Dabei können zwei Arten der Authentifizierung genutzt werden, die von verschiedenen Systemen verwaltet werden. Bei der Windows-Authentifizierung werden Benutzer und Passwörter vom lokal installierten Betriebssystem (bzw. in Netzwerkdomänen von einem Server, der als Domänencontroller fungiert) verwaltet. Dadurch entfällt die Notwendigkeit einer separaten Anmeldung am SQL-Server, da das Betriebssystem automatisch die Information an den SQL Server weitergibt, welcher User angemeldet ist.

Bei der SQL Server-Authentifizierung dagegen werden Benutzer und deren Passwörter von SQL Server verwaltet. Die entsprechenden Daten werden verschlüsselt in Systemtabellen abgelegt. Standardmäßig ist ein Benutzer mit Namen *sa* (die Abkürzung steht für Systemadministrator) eingerichtet, der volle Rechte auf dem gesamten SQL Server hat. Das Passwort für diesen User haben Sie selbst bei der Installation von SQL Server festgelegt.

Ob die SQL Server-Authentifizierung überhaupt verfügbar ist, hängt davon ab, ob Sie bei der Installation die reine Windows-Authentifizierung oder den gemischten Modus gewählt haben. Wenn Sie sich an die in Abschnitt 3.2, *Installation* beschriebene Installationsanweisung gehalten haben, sind beide Authentifizierungsmodi verfügbar, sodass Ihnen hier alle Möglichkeiten offenstehen.

Auch jedes Mal, wenn Sie sich mit dem SQL Server verbinden, ist auszuwählen, ob die Anmeldung über SQL Server- oder Windows-Authentifizierung erfolgen soll. Im erstgenannten Fall sind Anmeldungsname und Passwort einzugeben, bei der Windows-Authentifizierung werden Benutzernamen (gegebenenfalls mit vorangestelltem Domänennamen) angezeigt und sind nicht änderbar. Wenn Sie in diesem Authentifizierungsmodus einen anderen Windows-Benutzer verwenden möchten, müssen Sie sich zuerst vom Betriebssystem abmelden und mit dem anderen Login neu verbinden, der dann beim nächsten Verbindungsaufbau mit dem SQL Server automatisch verwendet wird.

Abbildung 11.1: Herstellung einer SQL Server-Verbindung mit Windows-Authentifizierung

Erfolgt die Verbindung zum SQL Server nicht über das SQL Server Management Studio, sondern aus einer Applikation für Endanwender, so werden die Anmeldedaten normalerweise in Form einer Verbindungszeichenfolge (hier ist der englische Begriff Connection String eigentlich gebräuchlicher) übertragen. Diese Verbindungszeichenfolge beinhaltet unter anderem den Namen des SQL Servers, den Namen der Serverinstanz (sofern nicht die Standardinstanz verwendet wird), den Authentifizierungsmodus sowie – im Falle der SQL Server-Authentifizierung – den Anmeldenamen und das dazugehörige Passwort.

Wichtig: Nie den sa-Login in einer Verbindungszeichenfolge verwenden!

Selbst wenn die Anwendung volle Rechte auf die Datenbank erhalten soll, sollten Sie nie das *sa*-Passwort in einer Verbindungszeichenfolge hinterlegen. Dies würde eine große Sicherheitslücke darstellen, da der *sa*-Login Vollzugriff auf alle Datenb anken des S ervers (inklusive der Systemdatenbanken) hat. Stattdessen sollte z umindest eine eigene Anmeldung für diese Anwendung erstellt werden, die alle Rechte auf die entsprechende Datenbank erhält.

Anlegen von SQL Server-Anmeldungen

Schauen wir uns nun einmal an, wie neue Anmeldungen angelegt werden, indem wir drei verschiedene SQL Server-Anmeldungen erstellen, die später unterschiedliche Rechte bekommen werden.

1. Stellen Sie im SQL Server Management Studio eine Verbindung zur lokalen Serverinstanz *SQL2008Express* her.
2. Klicken Sie nun mit der rechten Maustaste auf den Eintrag *Sicherheit/Anmeldungen* und wählen Sie die Option *Neue Anmeldung*.
3. Es erscheint das Dialogfeld zum Erstellen einer neuen Anmeldung. Im linken Bereich können Sie eine von fünf möglichen Seiten mit Einstellungen auswählen: *Allgemein*, *Serverrollen, Benutzerzuordnung, Sicherungsfähige Elemente* und *Status*. Im Moment benötigen wir davon allerdings nur die erste Seite.

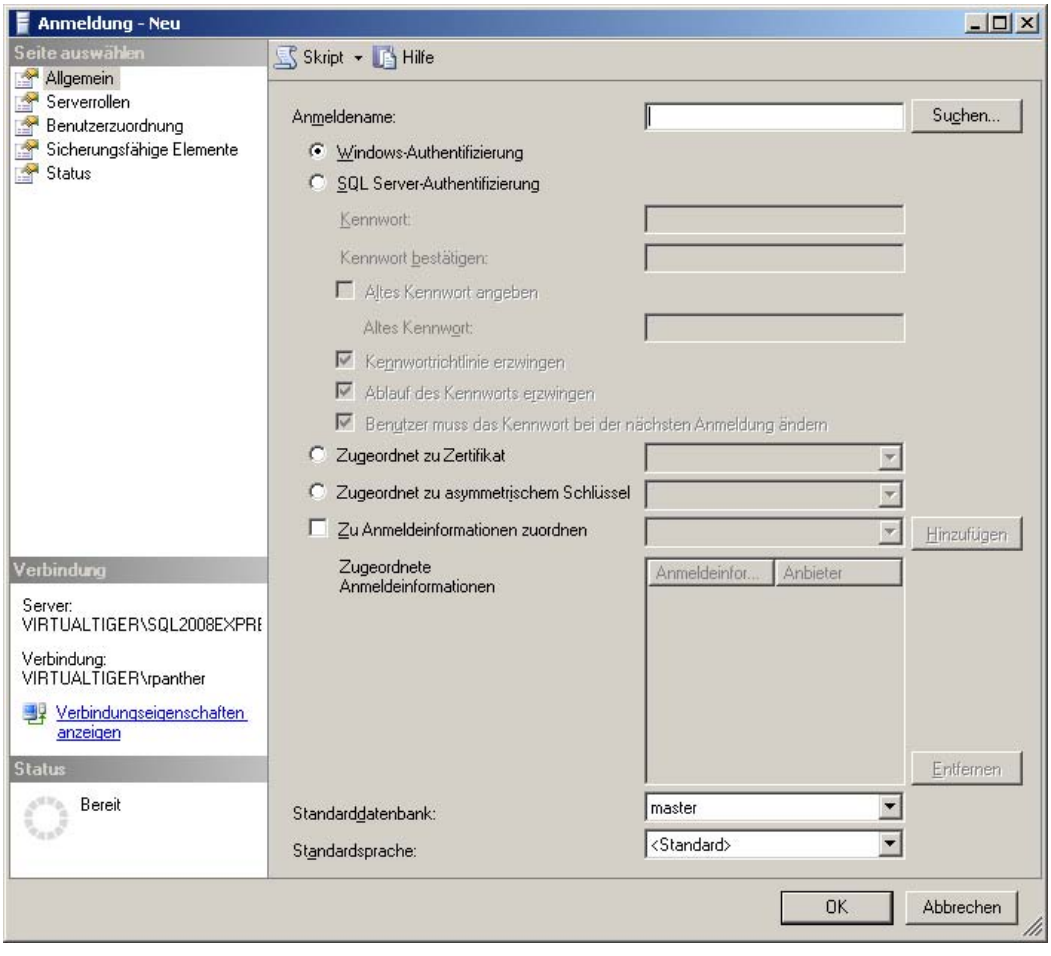


Abbildung 11.2: Das Dialogfeld zum Erstellen einer neuen Anmeldung

4. Nehmen Sie auf der Seite *Allgemein* folgende Einstellungen vor:
   - *Anmeldename*: **MediaBaseReadWrite**
   - *SQL Server-Authentifizierung*
   - *Kennwort*: **mbrw**
   - *Kennwortrichtlinie erzwingen*: nein
   - *Standarddatenbank*: master
   - *Standardsprache*: <Standard>

   Schließen Sie dann das Dialogfeld über die *OK*-Schaltfläche und die Anmeldung wird angelegt.
5. Legen Sie auf dieselbe Weise Anmeldungen mit Namen *MediaBaseReadOnly* (Kennwort: *mbro*) und *MediaBaseAudioOnly* (Kennwort: *mbao*) an.

Wichtig: Keine zu einfachen Passwörter verwenden

Der Einfachheit halber wurden für die Beispielanmeldungen in diesem Kapitel sehr kurze Passwörter verwendet. In der Praxis sollte man dies natürlich nicht tun, sondern stattdessen längere Passwörter verwenden, die neben Kleinbuchstaben auch Großbuchstaben und eventuell ein paar Sonderzeichen oder Ziffern enthalten.

Das Erstellen von Anmeldungen ist natürlich auch mit SQL-Anweisungen durchführbar. Dazu wird die `CREATE LOGIN`-Anweisung der sogenannten Data Control Language (DCL) verwendet. Zum Erstellen der oben genannten drei Anmeldungen wäre das folgende SQL-Skript auszuführen:

```
USE [master]
GO
CREATE LOGIN [MediaBaseReadWrite]
WITH PASSWORD=N'mbrw', DEFAULT_DATABASE=[master], CHECK_EXPIRATION=OFF, CHECK_POLICY=OFF
GO
CREATE LOGIN [MediaBaseReadOnly]
WITH PASSWORD=N'mbro', DEFAULT_DATABASE=[master], CHECK_EXPIRATION=OFF, CHECK_POLICY=OFF
GO
CREATE LOGIN [MediaBaseAudioOnly]
WITH PASSWORD=N'mbao', DEFAULT_DATABASE=[master], CHECK_EXPIRATION=OFF, CHECK_POLICY=OFF
GO
```

Bevor wir uns damit befassen, diesen Anmeldungen auch Datenbankbenutzer zuzuordnen, werfen wir noch einen Blick auf die Windows-Authentifizierung.

Windows-Benutzer und -Gruppen als Anmeldungen anlegen

Damit Windows-Benutzer als Anmeldungen verwendbar sind, müssen diese auch als solche angelegt werden. Das Vorgehen dazu entspricht weitgehend dem zum Anlegen von Anmeldungen für die SQL Server-Authentifizierung. Um auch dies auszuprobieren, legen wir nun einen lokalen Windows-Benutzer im Betriebssystem an[1] und erstellen anschließend dafür eine SQL Server-Anmeldung:

[1] Die Beschreibung für das Anlegen eines Betriebssystembenutzers orientiert sich am Betriebssystem Windows XP. Bei Verwendung einer anderen Windows-Variante kann das Verfahren leicht abweichen.

Hinweis: Lokale Admin-Rechte erforderlich

Damit das Anlegen eines Bet riebssystembenutzers möglich ist, müssen Si e über die entsprechenden Betriebssystemrechte verfügen. Dies ist be ispielsweise dann der Fall, wenn Ihr Betriebssystembenutzer der Gruppe lokale Administratoren angehört.

1. Öffnen Sie das Windows-Startmenü und rufen Sie dort die *Systemsteuerung* auf.
2. Klicken Sie in der *Systemsteuerung* auf den Punkt *Verwaltung* und im anschließend erscheinenden Fenster auf *Computerverwaltung*.
3. In der Computerverwaltung finden Sie in der Baumstruktur auf der linken Seite den Eintrag *Computerverwaltung/System/Lokale Benutzer und Gruppen*. Darunter gibt es zwei Unterpunkte, die mit *Benutzer* und *Gruppen* benannt sind.
4. Klicken Sie mit der rechten Maustaste auf den Knoten *Benutzer* und wählen Sie den Befehl *Neuer Benutzer*.
5. Geben Sie nun die in der Abbildung gezeigten Daten an, um den Benutzer zu erstellen und verwenden Sie dabei ein beliebiges Kennwort (das Sie sich natürlich merken sollten):

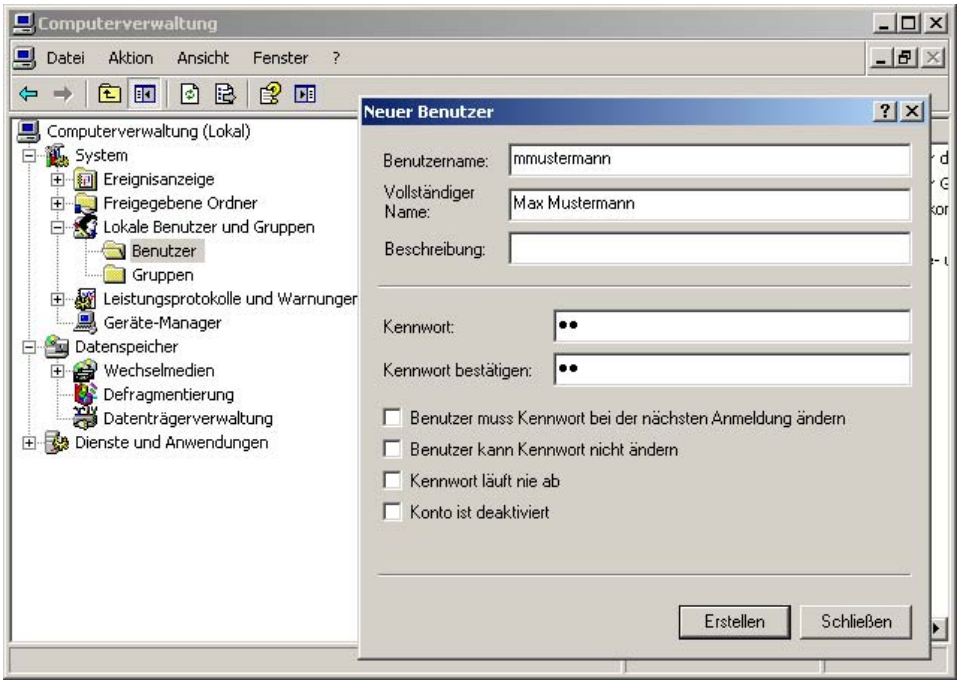


Abbildung 11.3: Das Dialogfeld zum Erstellen eines neuen Betriebssystembenutzers

Klicken Sie anschließend auf *Erstellen,* um den Benutzer anzulegen.

6. Nachdem der Benutzer erstellt ist, soll nun noch eine neue Gruppe erstellt werden, um später auch auf Gruppenebene Zugriff erteilen zu können. Klicken Sie dazu in der Computerverwaltung unter *Lokale Benutzer und Gruppen* mit der rechten Maustaste auf *Gruppen* und wählen Sie den Befehl *Neue Gruppe*.

7. Nennen Sie die Gruppe im daraufhin erscheinenden Dialogfeld *MediaBaseBenutzer* und fügen Sie über die Schaltfläche *Hinzufügen* zu dieser Gruppe den Benutzer **mmustermann** hinzu, bevor Sie die Gruppe über die Schaltfläche *Erstellen* anlegen.
8. Anschließend können Sie die Computerverwaltung und die Systemsteuerung wieder schließen.

Nachdem der Betriebssystembenutzer erstellt ist, wird noch eine SQL Server-Anmeldung benötigt.

1. Stellen Sie dazu im SQL Server Management Studio eine Verbindung zur lokalen Serverinstanz *SQL2008Express* her.
2. Klicken Sie mit der rechten Maustaste auf den Eintrag *Sicherheit/Anmeldungen* und wählen Sie den Befehl *Neue Anmeldung*.
3. Behalten Sie die Voreinstellung *Windows-Authentifizierung* bei und klicken Sie auf die hinter dem Eingabefeld *Anmeldename* stehende Schaltfläche *Suchen*.
4. Es erscheint ein weiteres Dialogfeld, bei dem der aktuelle Rechnername als Suchpfad voreingestellt ist. Geben Sie als Objektname **mmustermann** ein und klicken Sie auf die Schaltfläche *Namen überprüfen*. Wenn Sie sich nicht vertippt haben, erscheint im Eingabefeld dann die Kombination aus Rechnernamen und Windows-Benutzernamen (bei Ihnen wird anstelle des Rechnernamens *VIRTUALTIGER* sicherlich eine andere Bezeichnung stehen).

Abbildung 11.4: Eingabe eines Betriebssystembenutzers

5. Klicken Sie anschließend zweimal auf *OK* und die neue Anmeldung ist erstellt.
6. Fügen Sie auf dieselbe Art auch die Benutzergruppe *MediaBaseBenutzer* als Anmeldung hinzu. Damit die Gruppe gefunden werden kann, müssen Sie vorher über die Schaltfläche *Objekttypen* auch die Suche nach Gruppen aktivieren, indem Sie das entsprechende Kästchen anklicken.

Das Anlegen der Anmeldungen für Windows-Benutzer und Windows-Benutzergruppe können Sie alternativ auch mit den folgenden SQL-Anweisungen durchführen (beachten Sie, dass Sie dazu den Rechnernamen `VIRTUALTIGER` durch den eigenen Rechnernamen ersetzen):

```
USE [master]
GO
CREATE LOGIN [VIRTUALTIGER\mmustermann] FROM WINDOWS WITH DEFAULT_DATABASE=[master]
GO
CREATE LOGIN [VIRTUALTIGER\MediaBaseBenutzer] FROM WINDOWS WITH DEFAULT_DATABASE=[master]
GO
```

Anmeldungen testen

Nun können Sie die neu angelegten Anmeldungen einmal ausprobieren. Um die auf Windows-Authentifizierung basierenden Anmeldungen zu testen, gehen Sie wie folgt vor:

1. Schließen Sie alle offenen Anwendungen und melden Sie sich über das Windows-Startmenü vom Betriebssystem ab (oder nutzen Sie die Option *Benutzer wechseln*).
2. Melden Sie sich nun neu an und verwenden Sie dazu den lokalen Betriebssystembenutzer *MediaBaseBenutzer*. Falls Sie sich beim vorigen Mal mit einem Netzwerkbenutzer angemeldet haben, müssen Sie im Feld *Anmelden an* (das eventuell noch über die Schaltfläche *Optionen* eingeblendet werden muss) den lokalen Computer auswählen. Alternativ können Sie auch vor dem Benutzernamen (mit einem Backslash getrennt) den Namen des lokalen Rechners angeben. Auf meinem Datenbankentwicklungsrechner wäre dies beispielsweise **VIRTUALTIGER\MediaBaseBenutzer**. Dadurch stellen Sie sicher, dass der Benutzer *MediaBaseBenutzer* auf dem lokalen Rechner und nicht in der Domäne gesucht wird, wo er sicherlich nicht existiert (er wurde ja nur lokal auf dem Rechner angelegt).
3. Öffnen Sie das SQL Server Management Studio und stellen Sie eine Verbindung zur lokalen Serverinstanz *SQL2008Express* her. Benutzen Sie dabei die Windows-Authentifizierung.
4. Probieren Sie nun im Objekt-Explorer aus, auf welche Bereiche Sie Zugriff haben. Im Zweig *Datenbanken* sehen Sie zwar die Datenbank *MediaBase*, wenn Sie diesen Zweig allerdings weiter aufklappen möchten, erhalten Sie eine Fehlermeldung, die besagt, dass der Zugriff auf die MediaBase-Datenbank nicht möglich ist. Wenn Sie über die rechte Maustaste versuchen, das *Eigenschaften*-Fenster der Datenbank zu öffnen, wird die Fehlermeldung noch etwas deutlicher.

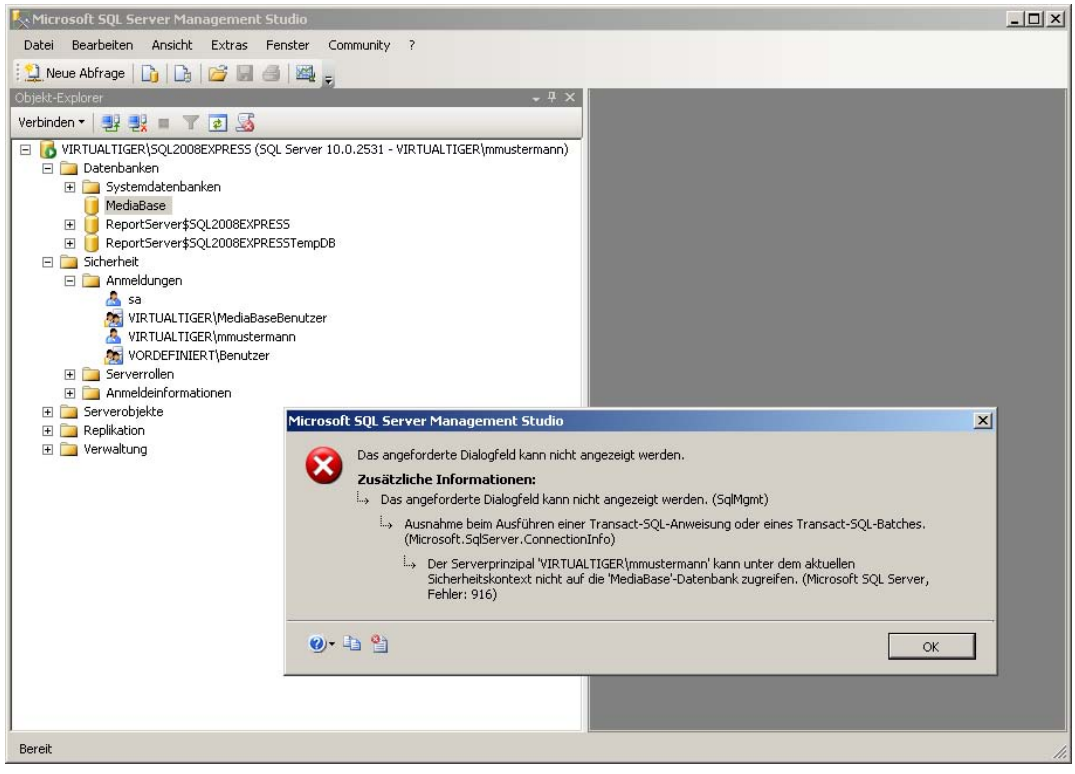


Abbildung 11.5: Fehlgeschlagener Versuch, auf die Eigenschaften der Datenbank *MediaBase* zuzugreifen

Der Grund hierfür liegt darin, dass der Windows-Benutzer *mmustermann* zwar als Anmeldung angelegt wurde (sonst wäre bereits der Verbindungsaufbau zum SQL Server mit einer entsprechenden Fehlermeldung gescheitert), diese Anmeldung aber noch keine Zugriffsrechte auf irgendwelche Datenbanken hat.

5. Schließen Sie das SQL Server Management Studio, melden Sie sich vom Rechner ab und dann mit dem Windows-Benutzer an, mit dem Sie den SQL Server installiert haben.

Hinweis: Lokale Admin-Rechte erforderlich

Wie der Screenshot in Abbildung 11.5 zeigt, sind im Objekt-Explorer auch nicht alle Anmeldungen sichtbar, sondern – abgesehen vom Systemadministrator (*sa*) – nur die gerade verwendete Anmeldung (*mmustermann*) selbst, sowie die Benutzer und Gruppen, auf welche die verwendete Anmeldung Rechte hat. Das sind in diesem Fall die Gruppe *VIRTUALTIGER\MediaBaseBenutzer* sowie die Standardgruppe *VORDEFINIERT\Benutzer*, der automatisch jeder erstellte Benutzer zugeordnet wird.

Sie können das gerade durchgeführte Experiment noch einmal über SQL Server-Authentifizierung wiederholen, indem Sie bei der Anmeldung am SQL Server die SQL Server-Authentifizierung wählen und dazu eine der frisch erstellten SQL-Anmeldungen (*MediaBaseReadWrite*, *MediaBaseReadOnly* oder *MediaBaseAudioOnly*) mit dem entsprechenden Passwort verwenden. Das Ergebnis beim versuchten Zugriff auf die Datenbank *MediaBase* wird dasselbe sein wie zuvor. Da auch für diese Anmeldungen

noch kein Zugriff auf die Datenbank eingerichtet wurde, fehlen auch hier die entsprechenden Berechtigungen.

Im Objekt-Explorer sind unter *Sicherheit* dann sogar nur noch die verwendete Anmeldung selbst sowie der Systemadministrator (*sa*) zu sehen, da die Windows-Authentifizierung nun nicht aktiv ist.

11.3 Verwalten von Datenbankbenutzern

Damit eine Anmeldung Zugriff auf eine Datenbank erhält, muss dieser ein Datenbankbenutzer zugeordnet werden, der eventuell vorher noch erstellt werden muss und entsprechende Rechte auf der Datenbank erhält. Die Datenbankbenutzer sind im Objekt-Explorer bei der jeweiligen Datenbank im Zweig *Sicherheit/Benutzer* zu finden. Hier sind bereits einige Benutzer vordefiniert:

- *dbo* – Database Owner (Vollzugriff auf die gesamte Datenbank)
- *guest* – Gast (lediglich Rechte, sich mit der Datenbank zu verbinden)
- *INFORMATION_SCHEMA* – (wird intern verwendet)
- *sys* – (wird intern verwendet)

Hier könnten Sie durch einen Klick mit der rechten Maustaste auf *Benutzer* und Auswahl von *Neuer Benutzer* einen neuen Datenbankbenutzer anlegen, um diesen später einer Anmeldung zuzuordnen. Doch es gibt auch einen komfortableren Weg, den wir nun ausprobieren wollen:

1. Stellen Sie im SQL Server Management Studio unter Verwendung der Windows-Authentifizierung eine Verbindung zur lokalen Serverinstanz *SQL2008Express* her.
2. Öffnen Sie im Objekt-Explorer den Zweig *Sicherheit/Anmeldungen*.
3. Klicken Sie mit der rechten Maustaste auf den Eintrag *MediaBaseReadWrite* und wählen Sie die Option *Eigenschaften* aus (alternativ dazu wäre auch ein Doppelklick auf den Anmeldungsnamen möglich).
4. Es erscheint das Dialogfeld *Anmeldungseigenschaften*, das weitgehend dem Dialogfeld entspricht, mit dem Sie ursprünglich die Anmeldung erstellt haben. Ändern Sie auf der Seite *Allgemein* die Standarddatenbank auf *MediaBase*. Damit wird zwar das Problem der fehlenden Berechtigung noch nicht gelöst, dies sorgt aber dafür, dass der SQL Server bei Verwendung dieser Anmeldung automatisch versucht, sich mit der Datenbank *MediaBase* zu verbinden.
5. Wechseln Sie nun zur Seite *Benutzerzuordnung*. Hier sind alle verfügbaren Datenbanken aufgelistet und Sie können durch Setzen eines Häkchens Zugriff auf die jeweilige Datenbank erteilen. In der Spalte *Benutzer* wird der entsprechende Datenbankbenutzer zugeordnet. Standardmäßig wird hier in dem Moment, in dem Sie die Datenbank zuordnen, der Name der Anmeldung eingetragen. Dadurch wird implizit ein Datenbankbenutzer mit demselben Namen wie die SQL Server-Anmeldung erstellt, sobald Sie auf *OK* klicken.

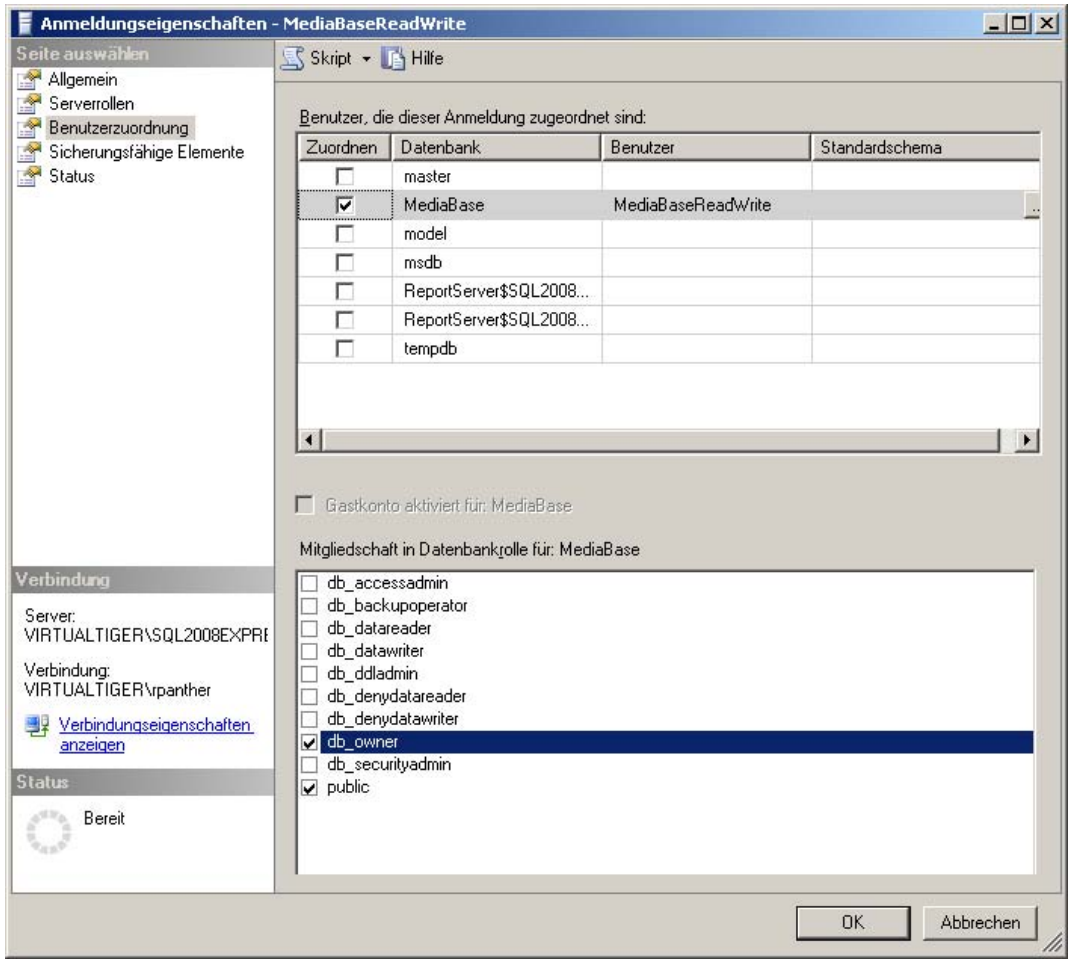


Abbildung 11.6: Die Benutzerzuordnung für die Anmeldung *MediaBaseReadWrite*

6. Im unteren Bereich des Dialogfeldes erfolgt die Rollenzuordnung, über die auf einfachem Weg eine ganze Menge an Berechtigungen erteilt werden kann. Standardmäßig ist hier die Rolle *public* ausgewählt, die lediglich eine Verbindung mit der Datenbank gewährt. Klicken Sie hier zusätzlich noch die Rolle *db_owner* an, damit der Benutzer *MediaBaseReadWrite* Vollzugriff auf die Datenbank *MediaBase* erhält.
7. Klicken Sie auf *OK*, um die Benutzerzuordnung durchzuführen. Anschließend können Sie im Objekt-Explorer unter *MediaBase/Sicherheit/Benutzer* nachprüfen, dass der Datenbankbenutzer wirklich angelegt wurde.
8. Erstellen Sie auf demselben Weg Datenbankbenutzer für die anderen Anmeldungen und ordnen Sie diesen folgende Rollen zu:

- *MediaBaseReadOnly* (Rollen: *public, db_datareader)*
- *MediaBaseAudioOnly* (Rollen: *public*)
- *VIRTUALTIGER\mmustermann* (Rollen: *public*, *db_owner*)
- *VIRTUALTIGER\MediaBaseBenutzer* (Rollen: *public*, *db_owner*)

Den gesamten Vorgang (Ändern der Standarddatenbank, Anlegen eines Datenbankbenutzers und Zuordnung desselben zu einer Anmeldung sowie Hinzufügen der Rolle *db_datareader* für diesen Benutzer) können Sie alternativ auch mit den folgenden SQL-Anweisungen durchführen:

```
USE [master]
GO
ALTER LOGIN [MediaBaseReadOnly] WITH DEFAULT_DATABASE=[MediaBase], DEFAULT_LANGUAGE=[Deutsch],
CHECK_EXPIRATION=OFF, CHECK_POLICY=OFF
ALTER LOGIN [MediaBaseAudioOnly] WITH DEFAULT_DATABASE=[MediaBase], DEFAULT_LANGUAGE=[Deutsch],
CHECK_EXPIRATION=OFF, CHECK_POLICY=OFF
ALTER LOGIN [VIRTUALTIGER\mmustermann] WITH DEFAULT_DATABASE=[MediaBase], DEFAULT_LANGUAGE=[Deutsch]
ALTER LOGIN [VIRTUALTIGER\MediaBaseBenutzer] WITH DEFAULT_DATABASE=[MediaBase], DEFAULT_LANGUAGE=[Deutsch]
GO
USE [MediaBase]
GO
CREATE USER [MediaBaseReadOnly] FOR LOGIN [MediaBaseReadOnly]
CREATE USER [MediaBaseAudioOnly] FOR LOGIN [MediaBaseAudioOnly]
CREATE USER [VIRTUALTIGER\mmustermann] FOR LOGIN [VIRTUALTIGER\mmustermann]
CREATE USER [VIRTUALTIGER\MediaBaseBenutzer] FOR LOGIN [VIRTUALTIGER\MediaBaseBenutzer]
GO
EXEC sp_addrolemember N'db_datareader', N'MediaBaseReadOnly'
EXEC sp_addrolemember N'db_owner', N'VIRTUALTIGER\mmustermann'
EXEC sp_addrolemember N'db_owner', N'VIRTUALTIGER\MediaBaseBenutzer'
GO
```

Für die Rollenzuordnung wird in diesem Skript die gespeicherte Systemprozedur `sp_addrolemember` verwendet. Dies geschieht allerdings nicht für die Rolle *public*, da diese automatisch zugeordnet wird, sobald ein Datenbankbenutzer für die Anmeldung erstellt wird.

Nun können Sie ausprobieren, ob der Zugriff über die gerade erstellten Datenbankbenutzer funktioniert. Der Einfachheit halber nutzen wir nun hierzu die SQL Server-Authentifizierung:

1. Stellen Sie im SQL Server Management Studio unter Verwendung der SQL Server-Authentifizierung eine Verbindung zur lokalen Serverinstanz *SQL2008Express* her. Verwenden Sie dabei die Anmeldung *MediaBaseReadOnly* mit dem entsprechenden Passwort.
2. Öffnen Sie nun im Objekt-Explorer den Zweig *Datenbanken/MediaBase/Tabellen*. Dort werden nun alle Tabellen angezeigt, weil Sie über die Rolle *db_datareader* Lesezugriff auf alle Daten der Datenbank haben.
3. Klicken Sie die Tabelle *dbo.Buch* mit der rechten Maustaste an und wählen Sie die Option *Oberste 200 Zeilen bearbeiten* aus, worauf der Inhalt der Tabelle angezeigt wird. Wenn Sie nun versuchen, eines der Felder zu bearbeiten, erhalten Sie beim Versuch, diese Änderung zu speichern, eine Fehlermeldung, die Sie darauf hinweist, dass die Schreibrechte nicht erteilt wurden.

Abbildung 11.7: Fehlende Schreibrechte

4. Probieren Sie anschließend dieselbe Aktion mit der Anmeldung *MediaBaseReadWrite* aus und das Speichern der Daten sollte funktionieren.
5. Melden Sie sich danach mit der Anmeldung *MediaBaseAudioOnly* an. Jetzt wird im Objekt-Explorer zur Datenbank *MediaBase* keine Tabelle angezeigt, da keine Leseberechtigungen für diese Anmeldung existieren.

Für die meisten Anwendungsfälle – insbesondere im Umfeld von SQL Server Express Edition – reicht eine indirekte Erteilung von Rechten durch die Zuweisung von vordefinierten Rollen für die entsprechenden Datenbanken aus. Bei Bedarf können Sie die Berechtigungen aber auch detaillierter erteilen bzw. verweigern, wie die folgenden Seiten zeigen.

11.4 Rechte und Rollen

Bei der Vergabe von Rechten (und Zuordnung von Rollen) wird zwischen zwei Arten von Rechten unterschieden. Das eine sind die Berechtigungen zum Zugriff auf Datenbankobjekte wie Tabellen, Sichten etc., die für Datenbankbenutzer erteilt werden. Das andere sind allgemeine Berechtigungen, die für den gesamten Server gelten (sogenannte Serverrechte) und daher für Anmeldungen erteilt werden. Für beide Varianten existieren auch entsprechende Rollen, mit deren Hilfe sich einheitliche Berechtigungen für eine Gruppe von Anmeldungen bzw. Datenbankbenutzern vereinfachen lassen.

Serverrechte und -rollen

Serverrollen sind fest definiert und lassen sich nicht erweitern oder ändern. Folgende Rollen stehen in diesem Zusammenhang zur Auswahl:

Tabelle 11.1: Übersicht der Serverrollen

Rolle	Bedeutung und enthaltene Berechtigungen
bulkadmin	Durchführen von Masseneinfügeoperationen (bcp bzw. BULK INSERT)
dbcreator	Erstellen, Ändern und Wiederherstellen von eigenen Datenbanken
diskadmin	Verwalten von Datenträgerdateien
processadmin	Beenden von SQL Server-Prozessen
public	Standardrolle ohne besondere Rechte
securityadmin	Verwalten von Anmeldungen und Berechtigungen
serveradmin	Konfiguration und Herunterfahren des Servers
setupadmin	Hinzufügen von Verbindungsservern
sysadmin	Systemadministrator: Vollzugriff auf den gesamten Server

Die Zuordnung einer Anmeldung zu einer Rolle können Sie im Dialogfeld *Anmeldungseigenschaften* auf der Seite *Serverrollen* vornehmen. Alternativ dazu können Sie auch eine gespeicherte Systemprozedur nutzen. Die folgende SQL-Anweisung würde beispielsweise der Anmeldung *MediaBaseReadWrite* die Serverrolle *diskadmin* zuordnen:

```
EXEC master..sp_addsrvrolemember @loginame = N'MediaBaseReadWrite', @rolename = N'diskadmin'
GO
```

Neben den vordefinierten Serverrollen lassen sich aber explizite Serverrechte an einzelne Anmeldungen erteilen. Auch dies geschieht über das Dialogfeld *Anmeldungseigenschaften* des jeweiligen Benutzers, nun aber auf der Seite *Sicherungsfähige Elemente*. Hier können Sie über die *Suchen*-Schaltfläche bestimmte Objekte wie beispielsweise Server, Endpunkte und Anmeldungen auswählen, um dann auf diese Objekte einzelne Rechte zu *erteilen* oder um diese explizit zu *verweigern*. Die Variante *Mit Erteilung* steht dafür, dass die jeweilige Anmeldung die entsprechende Berechtigung auch an andere Anmeldungen weitergeben darf. Die folgende Abbildung zeigt beispielsweise die Erteilung (mit Weitergabe) der Berechtigung zum *Herunterfahren* des Servers bei gleichzeitiger Verweigerung der Berechtigung zum *Massenvorgänge verwalten*.

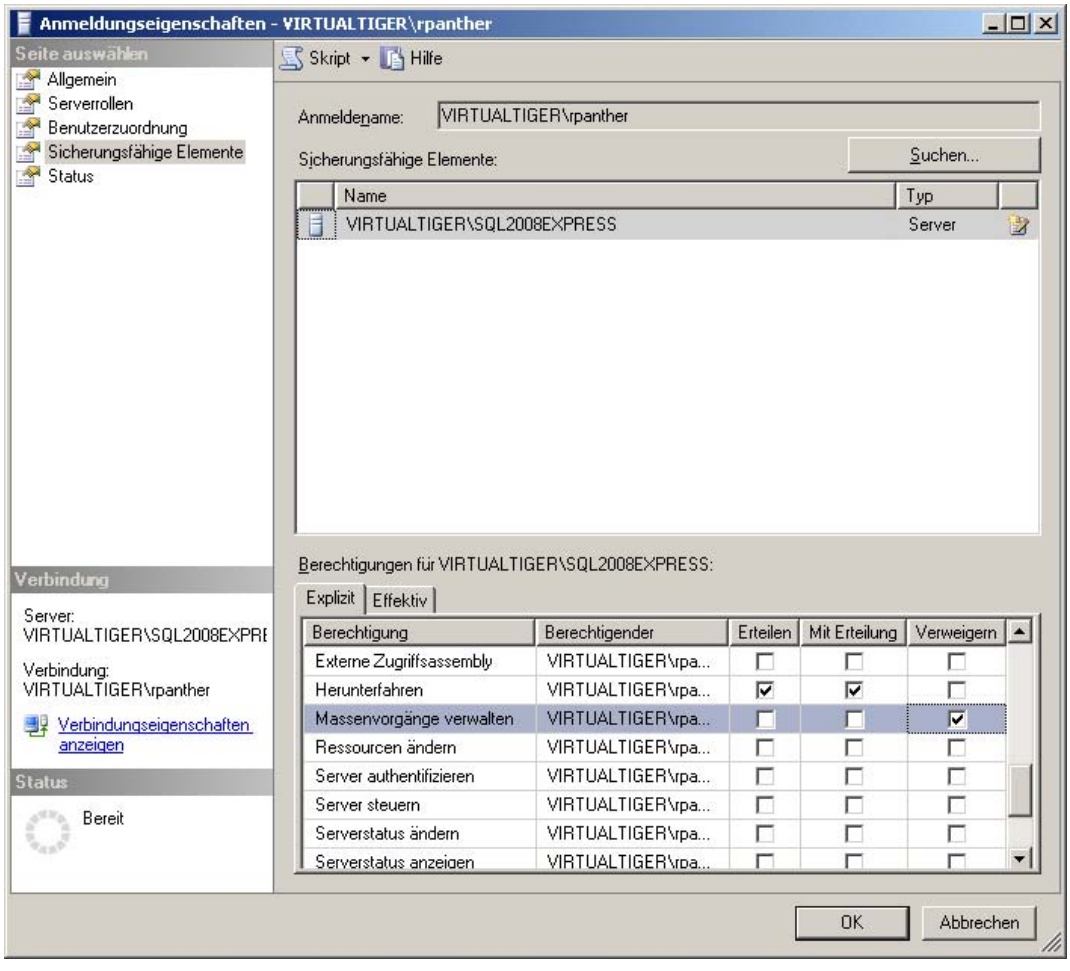


Abbildung 11.8: Erteilung von Serverrechten

Wenn Sie im unteren Bereich die Registerkarte *Effektiv* öffnen, werden die aktuellen Berechtigungen zum oben ausgewählten sicherungsfähigen Element angezeigt.

Das Erteilen und Verweigern der in Abbildung 11.8 dargestellten Rechte ist alternativ auch mit folgendem SQL-Skript ausführbar:

```
use [master]
GO
GRANT SHUTDOWN TO [VIRTUALTIGER\rpanther] WITH GRANT OPTION
DENY ADMINISTER BULK OPERATIONS TO [VIRTUALTIGER\rpanther]
GO
```

Best Practices: Verwendung von Serverrollen und -rechten

Die Bedeutung von Serverrollen und -rechten ist im Umfeld von SQL Server Express eher gering. Im Normalfall sind alle Anmeldungen lediglich der Rolle *public* zugeordnet und nur ein paar einzelne Anmeldungen bekommen zusätzlich noch die Rolle *sysadmin*, um auch administrative Aufgaben durchführen zu können. Einzelne Serverrechte werden selten vergeben.

Die anderen Serverrollen sowie die explizite Vergabe von einzelnen Serverrechten spielen eher im größeren Umfeld eine Rolle, wo sich verschiedene Mitarbeiter dediziert um die Verwaltung von Berechtigungen, das Erstellen von Datenbanksicherungen etc. kümmern, ohne dabei auf andere Bereiche zugreifen zu müssen.

Datenbankrechte und -rollen

Die Erteilung von Datenbankrechten erfolgt nach demselben Prinzip wie bei den Serverrechten, nur mit dem Unterschied, dass hier das *Eigenschaften*-Dialogfeld des entsprechenden Datenbankbenutzers verwendet wird. Auch hier gibt es eine Seite *Sicherungsfähige Elemente*, auf der Sie nach Auswahl der entsprechenden Datenbankobjekte (wie Tabellen, Sichten, Funktionen, gespeicherte Prozeduren etc.) explizit Berechtigungen darauf erteilen (mit und ohne Weitergabe) oder verweigern können. Für Tabellen und Sichten lassen sich die Berechtigungen sogar für einzelne Spalten verwalten.

Dazu können Sie auf der Seite *Allgemein* dem Datenbankbenutzer bestehende Datenbankrollen zuordnen, die bereits über einen vordefinierten Satz von Berechtigungen verfügen.

Tabelle 11.2: Übersicht der vordefinierten Datenbankrollen

Rolle	Bedeutung und enthaltene Berechtigungen
db_accessadmin	Verwalten des Zugriffs für Windows-Anmeldungen & -Gruppen sowie SQL Server-Anmeldungen
db_backupoperator	Durchführen von Datenbanksicherungen
db_datareader	Lesezugriff auf alle Tabellen und Sichten der Datenbank
db_datawriter	Schreibzugriff auf alle Tabellen und Sichten der Datenbank
db_ddladmin	Ausführung von DDL-Anweisungen
db_denydatareader	Verweigert Lesezugriff auf alle Benutzertabellen der Datenbank
db_denydatawriter	Verweigert Schreibzugriff auf alle Benutzertabellen der Datenbank
db_owner	Database Owner: Vollzugriff auf die gesamte Datenbank
db_securityadmin	Verwalten von Rollenmitgliedschaften und Berechtigungen
public	Standardrolle ohne besondere Rechte

Im Gegensatz zu den Serverrollen lassen sich die Datenbankrollen aber auch um selbstdefinierte Rollen erweitern, die dann – genau wie die Benutzer – detaillierte Berechtigungen erhalten können.

Probieren wir dies nun aus, indem wir eine Datenbankrolle *MediaBaseCDRole* erstellen, die den Vollzugriff auf die Tabellen *dbo.CD* und *dbo.CDTrack* ermöglicht:

1. Stellen Sie im SQL Server Management Studio unter Verwendung der Windows-Authentifizierung eine Verbindung zur lokalen Serverinstanz *SQL2008Express* her.

2. Öffnen Sie im Objekt-Explorer den Zweig *Datenbanken/MediaBase/Sicherheit/Rollen/Datenbankrollen*. Sie sehen hier die in Tabelle 11.2 dargestellten Datenbankrollen.
3. Klicken Sie mit der rechten Maustaste auf *Datenbankrollen* und wählen Sie die Option *Neue Datenbankrolle*. Es erscheint das Dialogfeld *Neue Datenbankrolle.*
4. Geben Sie als Namen für die Datenbankrolle **MediaBaseCDRole** ein und klicken Sie auf die Schaltfläche *Hinzufügen*, um der Rolle einen neuen Datenbankbenutzer zuzuordnen.
5. Geben Sie den Namen *MediaBaseAudioOnly* ein und klicken Sie auf *OK*.
6. Wechseln Sie nun zur Seite *Sicherungsfähige Elemente* und klicken Sie auf die Schaltfläche *Suchen*. Wählen Sie im Dialogfeld *Objekte hinzufügen* die Option *Alle Objekte des Typs* und klicken Sie wiederum auf *OK*. Im darauf folgenden Dialogfeld (*Objekttypen auswählen*) markieren Sie den Eintrag *Tabellen* und klicken nochmals auf *OK*.
7. Auf der Seite *Sicherungsfähige Elemente* werden nun alle Tabellen der Datenbank *MediaBase* aufgelistet. Klicken Sie hier die Tabelle *dbo.CD* an und markieren Sie darauf in der Berechtigungsliste unten die gesamte Spalte *Erteilen*. Verfahren Sie dann für die Tabelle *dbo.CDTrack* genauso.
8. Klicken Sie nun auf *OK* und die Rechte werden gemäß den gerade vorgenommenen Einstellungen erteilt.

Das SQL-Skript, mit dem Sie die Datenbankrolle alternativ erstellen können, ist diesmal etwas länger, da eine ganze Menge an Berechtigungen explizit vergeben wird:

```
USE [MediaBase]
GO
CREATE ROLE [MediaBaseCDRole]
GO
EXEC sp_addrolemember N'MediaBaseCDRole', N'MediaBaseAudioOnly'
GO
-- Berechtigungen für Tabelle dbo.CD erteilen
GRANT UPDATE ON [dbo].[CD] TO [MediaBaseCDRole]
GRANT ALTER ON [dbo].[CD] TO [MediaBaseCDRole]
GRANT VIEW CHANGE TRACKING ON [dbo].[CD] TO [MediaBaseCDRole]
GRANT SELECT ON [dbo].[CD] TO [MediaBaseCDRole]
GRANT TAKE OWNERSHIP ON [dbo].[CD] TO [MediaBaseCDRole]
GRANT VIEW DEFINITION ON [dbo].[CD] TO [MediaBaseCDRole]
GRANT INSERT ON [dbo].[CD] TO [MediaBaseCDRole]
GRANT DELETE ON [dbo].[CD] TO [MediaBaseCDRole]
GRANT CONTROL ON [dbo].[CD] TO [MediaBaseCDRole]
GRANT REFERENCES ON [dbo].[CD] TO [MediaBaseCDRole]

-- Berechtigungen für Tabelle dbo.CDTrack erteilen
GRANT UPDATE ON [dbo].[CDTrack] TO [MediaBaseCDRole]
GRANT ALTER ON [dbo].[CDTrack] TO [MediaBaseCDRole]
GRANT VIEW CHANGE TRACKING ON [dbo].[CDTrack] TO [MediaBaseCDRole]
GRANT SELECT ON [dbo].[CDTrack] TO [MediaBaseCDRole]
GRANT TAKE OWNERSHIP ON [dbo].[CDTrack] TO [MediaBaseCDRole]
GRANT VIEW DEFINITION ON [dbo].[CDTrack] TO [MediaBaseCDRole]
GRANT INSERT ON [dbo].[CDTrack] TO [MediaBaseCDRole]
GRANT DELETE ON [dbo].[CDTrack] TO [MediaBaseCDRole]
GRANT CONTROL ON [dbo].[CDTrack] TO [MediaBaseCDRole]
GRANT REFERENCES ON [dbo].[CDTrack] TO [MediaBaseCDRole]
GO
```

Nun können Sie ausprobieren, welche Rechte der Datenbankbenutzer *MediaBaseAudioOnly* durch seine Zugehörigkeit zur Rolle *MediaBaseCDRole* erhalten hat:

1. Stellen Sie im SQL Server Management Studio unter Verwendung der SQL Server-Authentifizierung eine Verbindung zur lokalen Serverinstanz *SQL2008Express* her. Verwenden Sie dabei die Anmeldung *MediaBaseAudioOnly* mit dem entsprechenden Passwort.
2. Öffnen Sie im Objekt-Explorer den Zweig *Datenbanken/MediaBase/Tabellen*. Dort sollten nun nur die beiden Tabellen *dbo.CD* und *dbo.CDTrack* sichtbar sein, auf die Sie mit dem verwendeten Benutzer volle Zugriffsrechte haben.

Best Practices: Verwendung von Datenbankrechten und -rollen

Wenn ein hohes Maß an Sicherheit notwendig ist, hat sich folgendes Vorgehen bereits in vielen Projekten bewährt: Für die von der Anwendung verwendeten Datenbankbenutzer existiert generell kein direkter Zugriff auf die Tabellen. Lesende Zugriffe erfolgen ausschließlich über Sichten, die auch nur die Spalten zur Verfügung stellen, die in der Anwendung benötigt werden. Schreibende Zugriffe erfolgen ausschließlich über gespeicherte Prozeduren. Auf diese Art werden die Möglichkeiten der Datenmanipulation (für den Fall, dass die Anmeldedaten doch einmal in die falschen Hände geraten) sehr gering gehalten, da einerseits nur die von der Anwendung benötigten Daten zugreifbar sind und durch die Verwendung von gespeicherten Prozeduren nur fest definierte Datenänderungen möglich sind.

11.5 Verwendung von Schemas

Sie haben sich vielleicht schon gewundert, warum in vielen Angaben zu Datenbankobjekten, wie Tabellen, Sichten etc. das Präfix *dbo* gefolgt von einem Punkt auftaucht. Dabei handelt es sich um das sogenannte Schema, zu dem das jeweilige Datenbankobjekt gehört. Dabei steht das Standardschema *dbo* für *Database Owner* und bezeichnet damit den Benutzer, der die Datenbank angelegt hat.

In früheren Versionen von SQL Server wurde zu jedem Datenbankobjekt ein Datenbankbenutzer als Besitzer definiert (nämlich der Benutzer, der das Objekt angelegt hat). Inzwischen wird ein Schema als Besitzer jedes Datenbankobjekts angegeben. Lediglich das Schema selbst hat einen Benutzer oder eine Rolle als Besitzer. Dazu wird über die Eigenschaften des Datenbankbenutzers für jeden Benutzer ein Standardschema definiert, das verwendet wird, wenn dieser Benutzer neue Datenbankobjekte erstellt. Auf diesem Weg können auch mehrere Benutzer (die dieses Schema als Standardschema verwenden) quasi als Besitzer des Objekts fungieren.

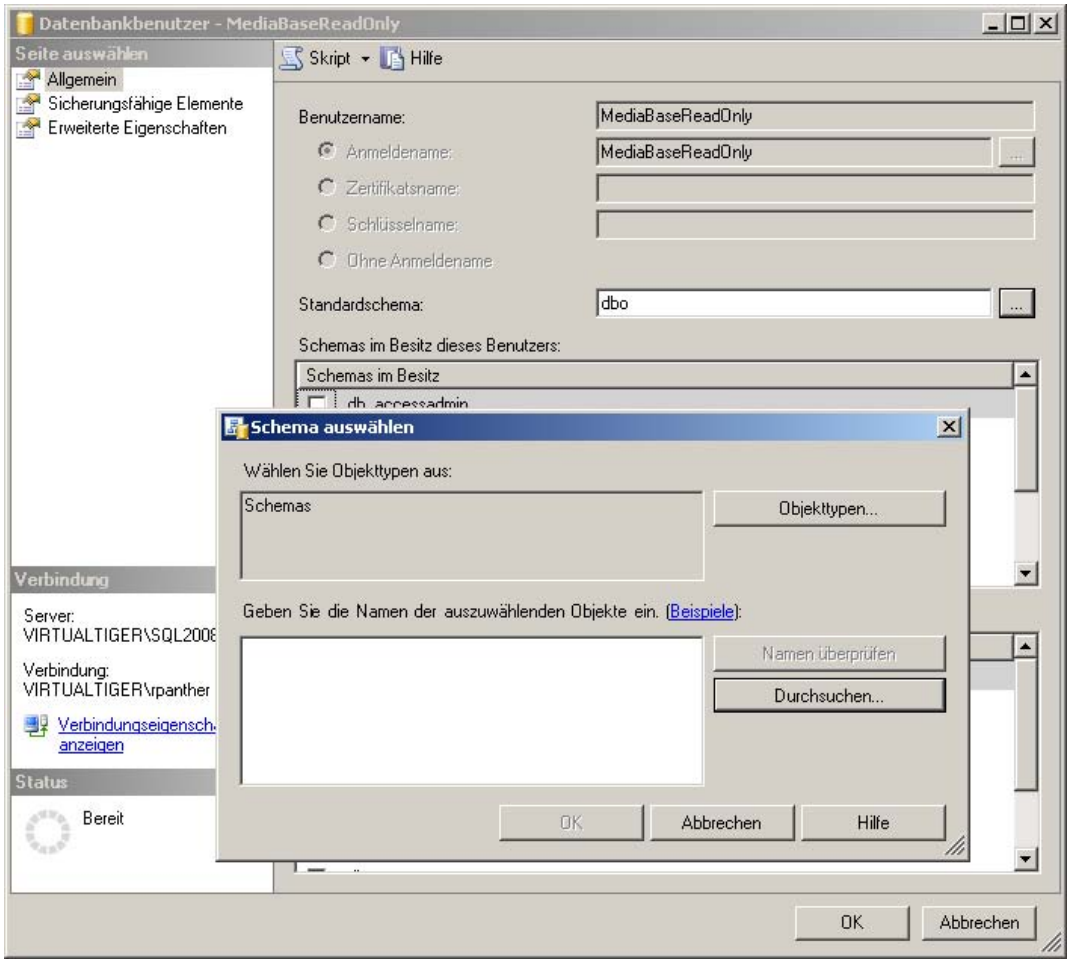


Abbildung 11.9: Auswahl eines Standardschemas für den Benutzer *MediaBaseReadOnly*

Dem ursprünglichen Ansatz, dass Benutzer und Schema identisch sind, ist es wohl zu verdanken, dass auch heute noch für jeden vordefinierten Benutzer und jede vordefinierte Rolle ein entsprechendes Schema existiert.

Neben der Protokollierung des Objektbesitzers werden Schemas primär zu zwei Zwecken verwendet:

- Logische Gruppierung von zusammengehörenden Objekten innerhalb einer Datenbank zur besseren Übersicht
- Einfachere Verwaltung von Berechtigungen, indem diese nicht auf einzelne Datenbankobjekte, sondern auf ein gesamtes Schema erteilt werden können

Schemas erstellen

Die Verwaltung von Schemas ist sehr einfach, da hier so gut wie keine Einstellungen vorzunehmen sind. Probieren wir dies nun einmal aus und erstellen ein neues Schema, um Datenbankobjekte, die später für Auswertungen verwendet werden, von den eigentlichen Daten zu trennen.

1. Stellen Sie im SQL Server Management Studio unter Verwendung der Windows-Authentifizierung eine Verbindung zur lokalen Serverinstanz *SQL2008Express* her.
2. Klicken Sie im Objekt-Explorer den Zweig *Datenbanken/MediaBase/Sicherheit/Schemas* mit der rechten Maustaste an und wählen Sie den Befehl *Neues Schema*.
3. Hier reicht es erst einmal aus, den Schemanamen **Auswertungen** anzugeben. Für den Schemabesitzer wird automatisch der Benutzer verwendet, mit dem Sie gerade arbeiten. Schließen Sie nun das Dialogfeld mit der Schaltfläche *OK* und das neue Schema wird angelegt.

Dieselbe Aktion wäre auch mit folgendem SQL-Skript durchführbar:

```
CREATE SCHEMA Auswertungen
```

Das Löschen eines Schemas geht ebenso einfach (soll aber an dieser Stelle nicht ausgeführt werden):

```
DROP SCHEMA Auswertungen
```

Schemas verwenden

Wenn Sie nun ein Datenbankobjekt in einem SQL-Skript ansprechen wollen, das nicht Bestandteil des *dbo*-Schemas ist, müssen Sie vor dem eigentlichen Objektnamen (mit einem Punkt getrennt) den Namen des Schemas angeben.

Erstellen wir nun eine neue Tabelle im Schema *Auswertungen*:

1. Stellen Sie im SQL Server Management Studio unter Verwendung der Windows-Authentifizierung eine Verbindung zur lokalen Serverinstanz *SQL2008Express* her.
2. Öffnen Sie ein Abfragefenster auf der Datenbank *MediaBase* und geben Sie folgende SQL-Anweisung ein:

```
SELECT *
INTO Auswertungen.AktuelleBuecher
FROM dbo.Buch
WHERE Erscheinungsjahr > 2007
```

3. Wenn Sie nun im Objekt-Explorer die Liste der Tabellen in der Datenbank *MediaBase* aufklappen, werden Sie sehen, dass die neue Tabelle *Auswertungen.AktuelleBuecher* ganz oben in der Liste erscheint (eventuell muss die Ansicht vorher aktualisiert werden). Das liegt daran, dass diese alphabetisch sortiert ist. Wenn Sie also eine Datenbank mit sehr vielen Tabellen haben, erreichen Sie durch die Verwendung von Schemas, dass Tabellen, die zu einem Schema gehören, direkt untereinander stehen (dasselbe gilt natürlich auch für Sichten etc.).

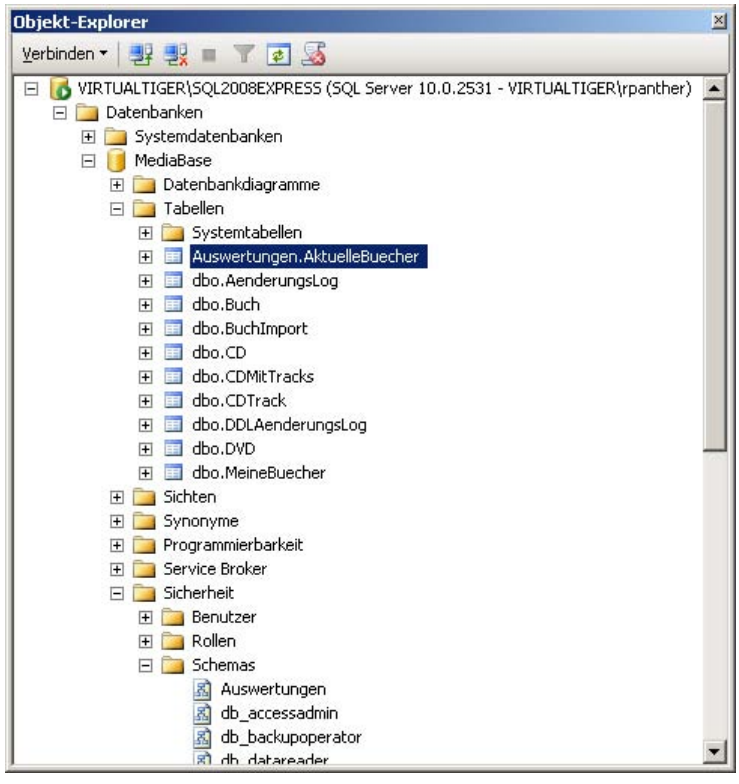


Abbildung 11.10: Das Schema *Auswertungen* und die Tabelle *AktuelleBuecher* im Objekt-Explorer

4. Um die Daten der neuen Tabelle abzufragen, muss natürlich ebenfalls der Schemaname mit angegeben werden:

```
SELECT *
FROM Auswertungen.AktuelleBuecher
```

Sie haben gerade ein Datenbankschema verwendet, um eine bessere Übersicht durch eine Gruppierung der Datenbankobjekte in Schemas zu erzielen.

Best Practices: Verwendung von Datenbankrechten und -rollen

Solange Sie ausschließlich mit dem *dbo*-Schema arbeiten, können Sie die Angabe des Schemas auch weglassen. Es wird jedoch generell empfohlen, Datenbankobjekte immer möglichst mit Angabe des Schemas zu spezifizieren, da dies einerseits Missverständnisse beim Lesen vermeidet und teilweise sogar für eine bessere Performance sorgt, da der SQL Server gleich weiß, in welchem Schema das entsprechende Objekt zu suchen ist (ansonsten wird das Objekt zuerst im Standardschema des angemeldeten Benutzers gesucht und anschließend im *dbo*-Schema, sofern die Berechtigungen dies zulassen).

Berechtigungen für Schemas verwalten

Wie bereits weiter oben erwähnt, liegt ein weiterer Vorteil von Schemas darin, dass diese eine einfachere Vergabe von Berechtigungen ermöglichen. Um einer Anmeldung Zugriff auf alle Objekte eines Schemas zu erteilen, gibt es – neben der müßigen Erteilung der einzelnen Rechte für jedes Objekt, das Bestandteil des Schemas ist – zwei einfache Möglichkeiten:

- Übernahme des Besitzes des entsprechenden Schemas
- Erteilen von expliziten Berechtigungen für das gesamte Schema

Die erstgenannte Variante ist einfacher, bietet aber dafür geringere Flexibilität, weil mit der Besitzübernahme natürlich gleich alle Rechte auf das Schema erteilt sind. Dazu rufen Sie das *Eigenschaften*-Dialogfeld des entsprechenden Benutzers auf und aktivieren hier die gewünschten Schemas in der Liste *Schemas im Besitz dieses Benutzers*.

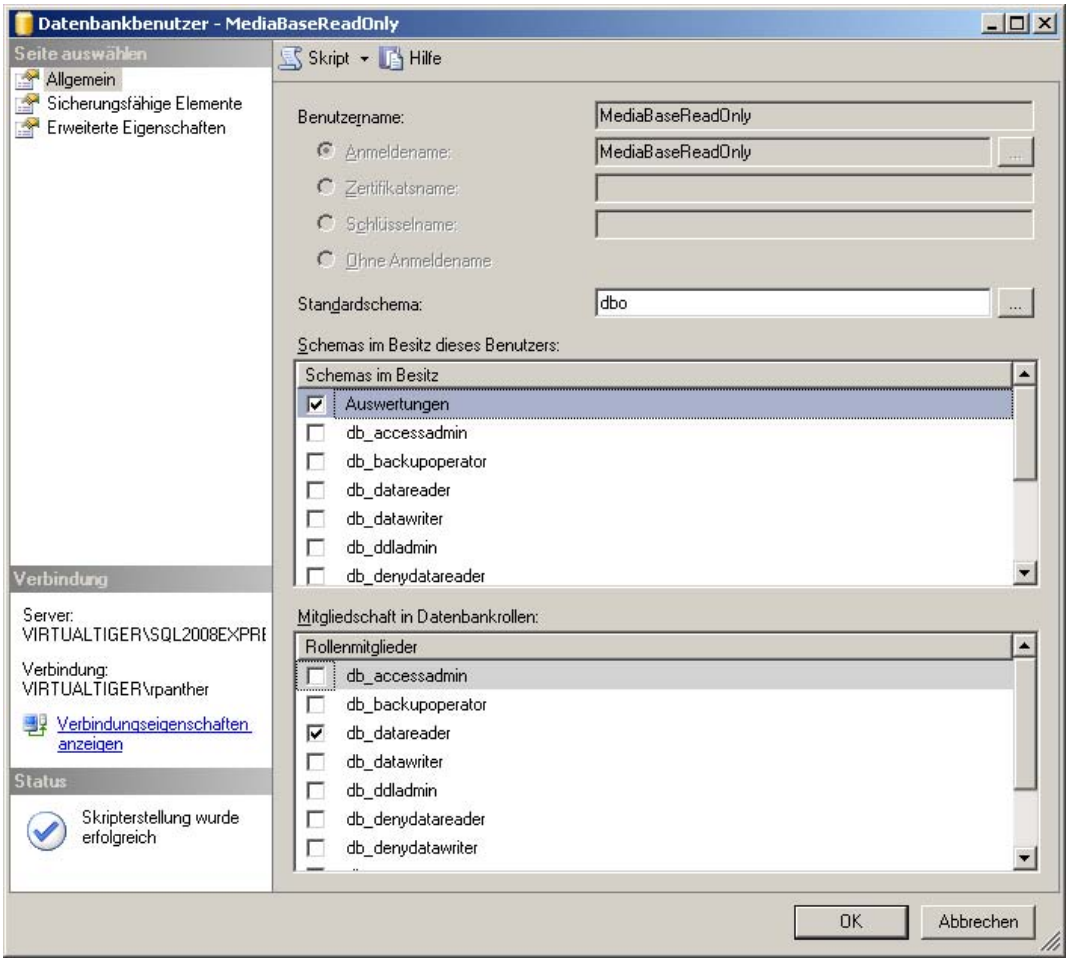


Abbildung 11.11: Besitzübernahme des Schemas *Auswertungen* durch den Benutzer *MediaBaseReadOnly*

Alternativ können Sie auch die folgende SQL-Anweisung verwenden, um beispielsweise dem Benutzer *MediaBaseReadOnly* den Besitz des Schemas *Auswertungen* übernehmen zu lassen:

```
ALTER AUTHORIZATION ON SCHEMA::[Auswertungen] TO [MediaBaseReadOnly]
```

Ein weiterer Nachteil dieser Variante liegt darin, dass nur ein Benutzer oder eine Rolle den Besitz eines Schemas halten kann. Daher nutzen wir nun die zweite Variante (Erteilen von expliziten Berechtigungen), um dem Benutzer *MediaBaseAudioOnly* (der ja bisher nur Zugriff auf die Tabellen *dbo.CD* und *dbo.CDTrack* hat), Lesezugriff für das gesamte Schema *Auswertungen* zu geben:

1. Stellen Sie im SQL Server Management Studio unter Verwendung der Windows-Authentifizierung eine Verbindung zur lokalen Serverinstanz *SQL2008Express* her.
2. Öffnen Sie im Objekt-Explorer den Zweig *Datenbanken/MediaBase/Sicherheit/Schemas*.
3. Klicken Sie mit der rechten Maustaste auf das Schema *Auswertungen* und wählen Sie anschließend *Eigenschaften* aus. Auf der Seite *Allgemein* können Sie ebenfalls den Besitz des Schemas an einen Benutzer (oder eine Rolle) übertragen.
4. Öffnen Sie die Seite *Berechtigungen*.
5. Über die Schaltfläche *Suchen* können Sie *Benutzer* oder *Rollen* auswählen, für die Sie anschließend Rechte erteilen. Suchen Sie hier nach dem Benutzer *MediaBaseAudioOnly*.
6. Geben Sie dem gewählten Benutzer nun Leserechte auf das gesamte Schema, indem Sie das *Erteilen*-Kästchen für die Berechtigung *Auswählen* anklicken.

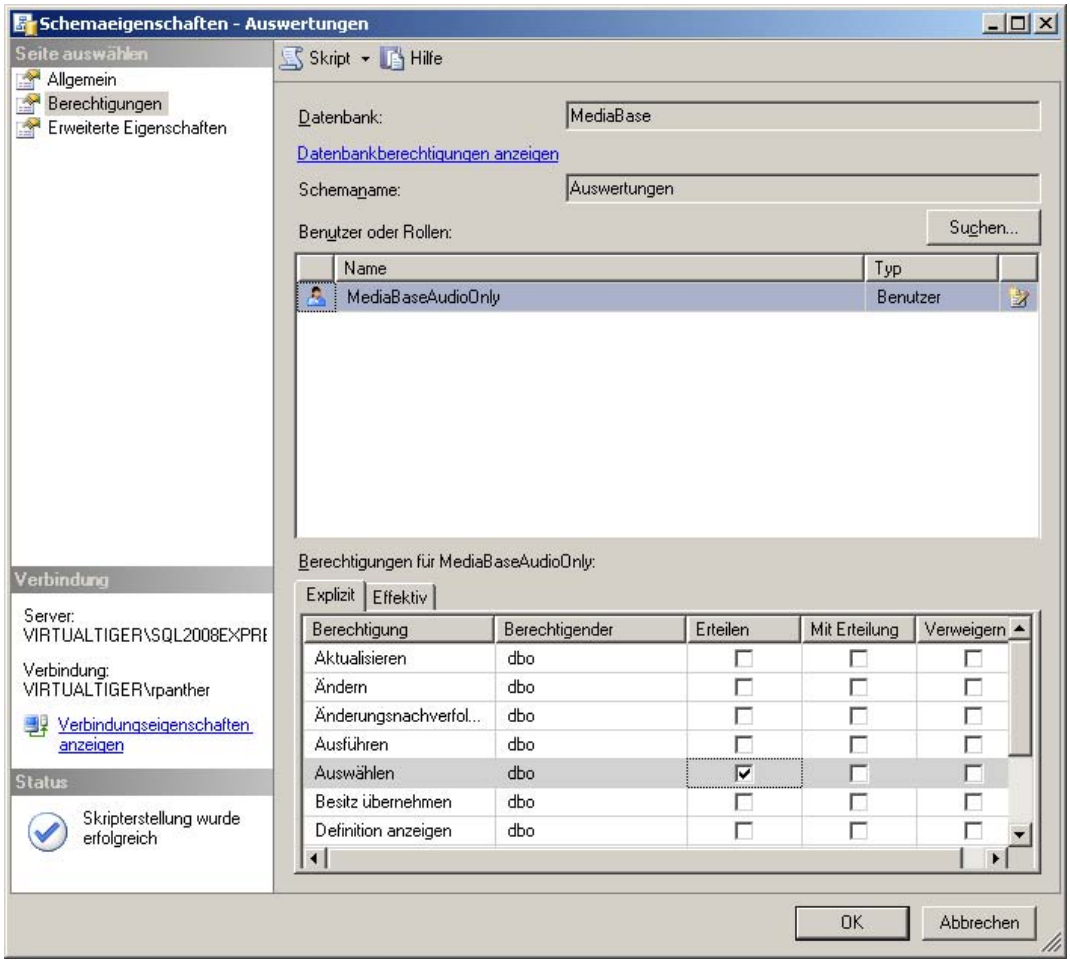


Abbildung 11.12: Erteilung von `SELECT`-Rechten auf das Schema *Auswertungen*

7. Schließen Sie nun das *Schemaeigenschaften*-Dialogfeld durch einen Klick auf *OK*.
8. Wenn Sie sich anschließend über SQL Server-Authentifizierung mit dem Benutzer *MediaBaseAudioOnly* anmelden, werden Sie sehen, dass dieser nun auch Zugriff auf die Tabelle *Auswertungen.AktuelleBuecher* hat, da diese Bestandteil des *Auswertungen*-Schemas ist.

Alternativ hätten Sie diese Aktion auch mit der folgenden SQL-Anweisung durchführen können:

```
GRANT SELECT ON SCHEMA::[Auswertungen] TO [MediaBaseAudioOnly]
```

Wenn Sie zusätzlich die Option *Mit Erteilung* aktivieren möchten, wäre die obige Anweisung um den Zusatz `WITH GRANT OPTION` zu erweitern:

```
GRANT SELECT ON SCHEMA::[Auswertungen] TO [MediaBaseAudioOnly] WITH GRANT OPTION
```

Die entsprechenden Anweisungen zum Entziehen oder Verweigern einer Berechtigung sind `REVOKE` (Zurücknehmen einer vorhandenen Berechtigung) und `DENY` (Verweigern einer Berechtigung, die evtl. durch eine Rollen- oder Gruppenzugehörigkeit gegeben war).

11.6 Übungen zu diesem Kapitel

In diesem Abschnitt finden Sie einige Übungen zu diesem Kapitel. Die richtigen Antworten, Lösungen und den Programmcode finden Sie wie immer auf der Website *www.vsxpress.de.*

Übung 11.1

Erstellen Sie in der Windows-Benutzerverwaltung eine neue Benutzergruppe *SQLServerAdmins* und ordnen Sie dieser Ihren eigenen Betriebssystembenutzer hinzu.

Übung 11.2

Legen Sie im SQL Server Management Studio für die Windows-Benutzergruppe *SQLServerAdmins* eine Anmeldung an und ordnen Sie diese der Serverrolle *sysadmin* zu.

Übung 11.3

Erstellen Sie eine SQL Server-Anmeldung *MediaBaseBooksOnly* und richten Sie dafür auch einen entsprechenden Datenbankbenutzer ein. Erteilen Sie diesem Datenbankbenutzer anschließend Vollzugriff auf die Tabelle *dbo.Buch*.

11.7 Zusammenfassung

In diesem Kapitel haben Sie die wichtigsten Möglichkeiten kennengelernt, um Zugriffe auf Datenbankserver, Datenbanken und Datenbankobjekte zu regeln. Je nach Art der verwendeten Authentifizierung erstellen Sie Ihre Benutzer und Gruppen im Windows-Betriebssystem (Windows-Authentifizierung) oder Anmeldungen im SQL Server (SQL Server-Authentifizierung). Diese Anmeldungen können über vordefinierte Serverrollen serverweite Berechtigungen erhalten.

Datenbankseitig werden den Anmeldungen Datenbankbenutzer zugeordnet, die explizit Rechte erhalten können, aber ebenso Rollen zugeordnet werden können, den sogenannten Datenbankrollen.

Schemas ermöglichen das Gruppieren von Objekten wie Tabellen, Sichten etc. innerhalb einer Datenbank. Darüber wird einerseits eine übersichtlichere Anordnung der Objekte im Objekt-Explorer erreicht. Dazu erleichtern Schemas die effektive Verwaltung von Rechten, da diese nun auf das ganze Schema erteilt werden können und nicht mehr für jedes Datenbankobjekt einzeln vergeben werden müssen.

Kapitel 12

Daten sichern und bewegen

In diesem Kapitel lernen Sie

- was Sie beachten müssen, wenn Sie Datenbankdateien sichern oder kopieren
- was es mit dem Transaktionslog auf sich hat
- wie Sie Datenbanken sichern und wiederherstellen können
- welche Möglichkeiten es gibt, um Datenbanken oder Teile daraus zu importieren und zu exportieren

12.1 Sichern von Datenbankdateien

Ein wichtiges Thema, das gerne vernachlässigt wird, ist das Sichern und Wiederherstellen von Datenbanken. Hierfür gibt es verschiedene Möglichkeiten, die ich Ihnen in diesem Kapitel vorstellen will.

Der naive Backup-Ansatz: Dateien kopieren

Die einfachste Art, eine Datenbank zu sichern, liegt darin, die entsprechenden Datenbank- und Protokolldateien zu kopieren. Problematisch ist dabei, dass der SQL Server-Dienst alle aktiven Datenbank- und Protokolldateien exklusiv sperrt. Um die Dateien kopierbar zu machen, müsste man die entsprechende Datenbank erst offline schalten, wodurch die Dateisperre aufgehoben wird. Nach dem Kopieren kann die Datenbank dann wieder online geschaltet werden.

Probieren wir dies einmal am Beispiel der Datenbank *MediaBase* aus:

1. Öffnen Sie den Windows-Explorer und wechseln Sie zu dem Ordner, in dem die Datenbankdateien abgelegt sind (wenn Sie die Standardeinstellung beibehalten haben, dann ist dies der Ordner *C:\Programme\Microsoft SQL Server\MSSQL10.SQL2008EXPRESS\MSSQL\DATA*). Hier liegen die Dateien sowohl für die Zeilendaten als auch für die Transaktionsprotokolle aller Datenbanken, sofern dies nicht explizit vorher geändert wurde.

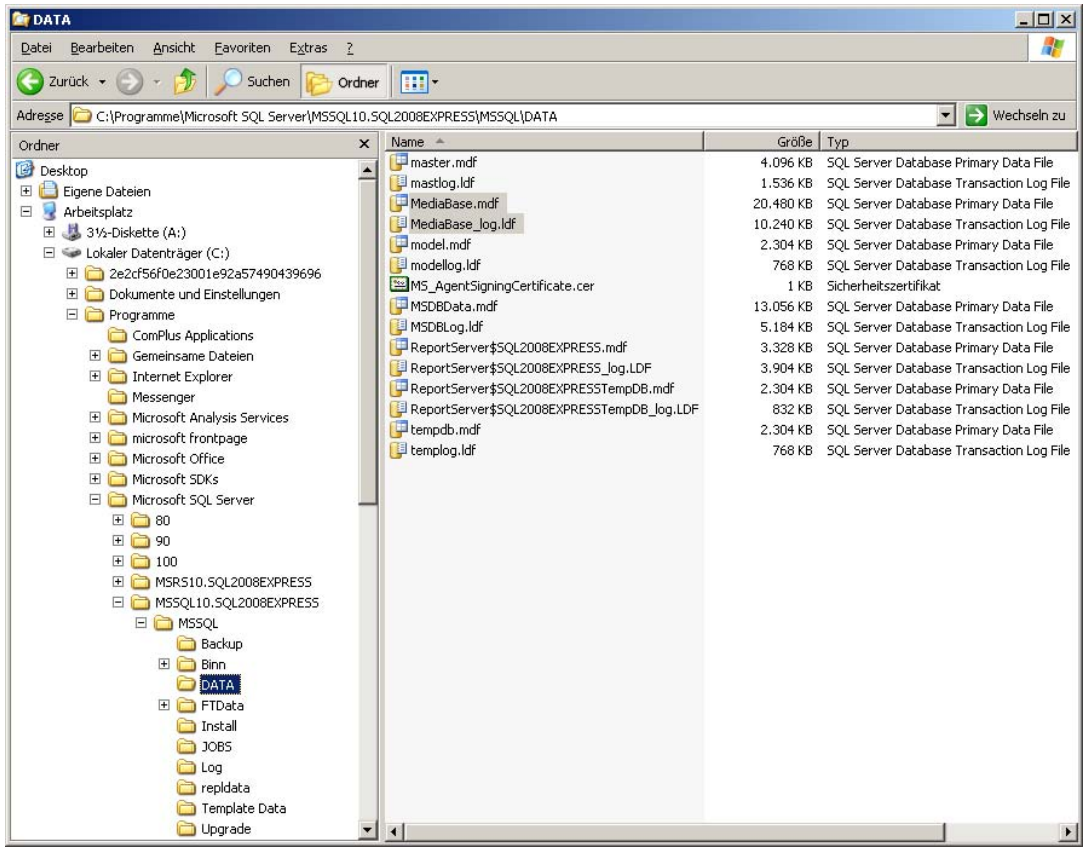


Abbildung 12.1: Das Standarddatenverzeichnis von SQL Server

2. Versuchen Sie nun, die beiden Dateien *MediaBase.mdf* und *MediaBase_log.ldf* in einen anderen Ordner zu kopieren, werden Sie eine Fehlermeldung erhalten, die besagt, dass die Datei von einer anderen Person bzw. einem anderen Programm verwendet wird.
3. Lassen Sie den Windows-Explorer im Hintergrund geöffnet und stellen Sie anschließend im SQL Server Management Studio eine Verbindung zur lokalen Serverinstanz *SQL2008Express* her.
4. Klicken Sie hier die Datenbank *MediaBase* mit der rechten Maustaste an und wählen Sie dann im Kontextmenü die Option *Tasks/Offline schalten* aus. Nach einer kurzen Pause erhalten Sie die Meldung, dass das Offlinesschalten erfolgreich abgeschlossen wurde und die Datenbank wird im Objekt-Explorer entsprechend markiert. Die Datenbank bleibt so zwar sichtbar, mit ihr kann aber nicht mehr gearbeitet werden.

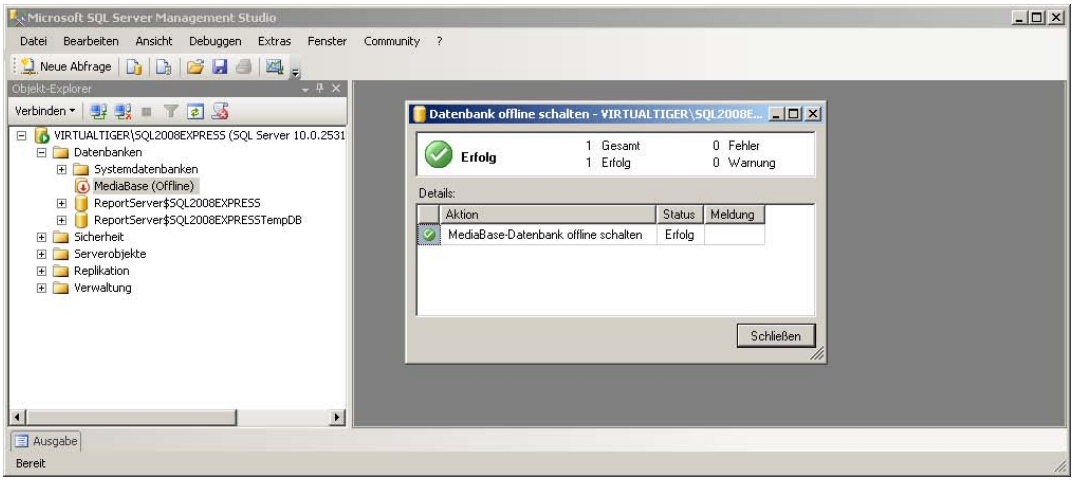


Abbildung 12.2: Die Datenbank *MediaBase* wurde offline geschaltet

5. Kehren Sie zum Windows-Explorer zurück und versuchen Sie erneut die beiden Dateien *MediaBase.mdf* und *MediaBase_log.ldf* in einen anderen Ordner zu kopieren. Diesmal sollte es problemlos funktionieren, da die Dateien nun nicht mehr vom SQL Server-Dienst gesperrt werden.
6. Sobald der Kopiervorgang abgeschlossen ist, können Sie die Datenbank im Objekt-Explorer des Management Studios wieder online schalten, indem Sie sie wieder mit der rechten Maustaste anklicken und die Option *Tasks/Online schalten* auswählen.

Das Online- und Offlineschalten der Datenbank *MediaBase* können Sie alternativ auch mit den folgenden SQL-Anweisungen durchführen:

```
USE master
ALTER DATABASE MediaBase SET OFFLINE
ALTER DATABASE MediaBase SET ONLINE
```

Die `USE`-Anweisung ist notwendig, da die Datenbank *MediaBase* nicht offline genommen werden kann, wenn Sie mit dieser verbunden sind.

Best Practices: Datenbanken offline schalten für File Backups

Gerade in größeren Netzwerken sind oft zentrale Backup-Server im Einsatz, die nachts oder am Wochenende komplette Laufwerke von einzelnen PCs dateiweise sichern. Für diesen Fall kann es sinnvoll sein, zumindest benutzerdefinierte Datenbanken während dieses Zeitfensters offline zu schalten, damit sie mitgesichert werden können. Das Offlineschalten ist allerdings nicht für die Systemdatenbanken verfügbar, sodass hier nur der SQL Server-Dienst beendet werden kann, um die zugehörigen Dateien zugreifbar zu machen.

Das Offline- und Onlineschalten von Datenbanken, um die Dateien kopierbar zu machen, bringt allerdings ein paar gravierende Nachteile mit sich:

- Eine Sicherung der kompletten Datenbankdateien lässt nur ein Wiederherstellen zu dem Zeitpunkt zu, an dem die Sicherung erstellt wurde. Besser wäre es natürlich, wenn man einen beliebigen Zustand wiederherstellen könnte.

- Während die Datenbank offline ist, kann nicht mit ihr gearbeitet werden. Es wird also ein gewisses Wartungsfenster benötigt.
- Um die Datenbank offline schalten zu können, darf keine offene Verbindung zu ihr bestehen. Ansonsten bricht das Offlineschalten ab und die folgende Sicherung schlägt fehl.
- Die Datenbank kann nicht einfach auf einem anderen Server wieder eingespielt werden, da die Pfade und Einträge in den Systemdatenbanken aller Voraussicht nach nicht dazu passen würden.

Zumindest für die beiden letztgenannten Probleme gibt es eine einfache Lösung, indem man die im Folgenden beschriebene Alternative nutzt.

Trennen und Verbinden von Datenbanken

Um eine Datenbank in einen transportfähigen Zustand zu bringen, lässt sich diese vom Server quasi abkoppeln. Das Vorgehen ist fast dasselbe wie beim Offlineschalten, nur dass der Befehl im *Tasks*-Menü des Datenbankkontextmenüs dann *Trennen* heißt. Im Gegensatz zum Offlineschalten ist die Datenbank danach aber nicht mehr im Objekt-Explorer von SQL Server Management Studio sichtbar, da auch alle Verweise in den Systemdatenbanken entfernt wurden.

Nun können Sie die Dateien an einen beliebigen Ort kopieren und anschließend die Datenbank wieder online schalten, wodurch die entsprechenden Einträge in den Systemdatenbanken wieder hergestellt werden. Bei Bedarf können auch die kopierten Dateien auf einem anderen Server (und in einem anderen Verzeichnis) wieder online geschaltet werden und neben den Einträgen in den Systemdatenbanken werden auch die Pfade für die Datenbank- und Protokolldateien automatisch entsprechend angepasst.

Verschieben wir nun die Datenbankdateien der *MediaBase*-Datenbank in ein anderes Verzeichnis:

1. Stellen Sie im SQL Server Management Studio eine Verbindung zur lokalen Serverinstanz *SQL2008Express* her und öffnen Sie im Objekt-Explorer den Zweig, der die *Datenbanken* beinhaltet.
2. Klicken Sie hier die Datenbank *MediaBase* mit der rechten Maustaste an und wählen Sie dann im Kontextmenü den Befehl *Tasks/Trennen* aus.
3. Es erscheint ein Dialogfeld, in dem Sie angeben können, ob bestehende Verbindungen zur Datenbank vorher getrennt werden sollen und ob die Optimierungsstatistiken der Datenbank vor dem Trennen aktualisiert werden sollen.

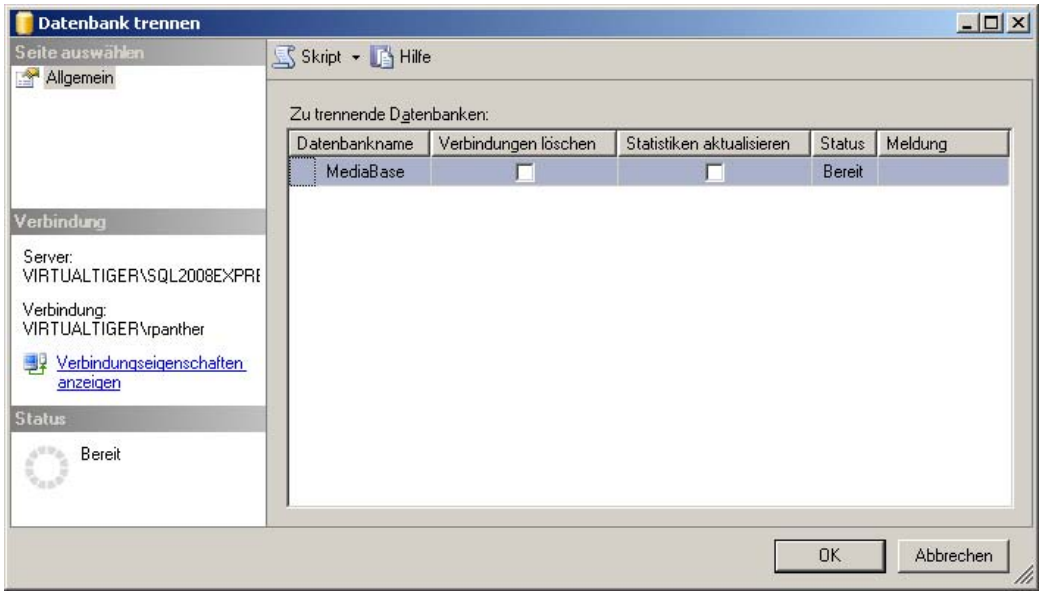


Abbildung 12.3: Das Dialogfeld zum Trennen von Datenbanken

4. Nach einer kurzen Pause ist die Datenbank nicht mehr im Objekt-Explorer sichtbar, obwohl die Datenbankdateien nach wie vor an derselben Stelle liegen.
5. Öffnen Sie nun den Windows-Explorer und erstellen Sie ein neues Verzeichnis, in dem Sie künftig die Benutzerdatenbanken speichern wollen (z.B. *C:\SQLServer*). Verschieben Sie anschließend die Datenbankdateien *MediaBase.mdf* und *MediaBase_log.ldf* in diesen neuen Ordner.
6. Um die Datenbank wieder zu verbinden, klicken Sie im Objekt-Explorer mit der rechten Maustaste auf *Datenbanken* und wählen dann die Option *Anfügen* aus. Es folgt ein Dialogfeld, in dem Sie über die Schaltfläche *Hinzufügen* in einem Verzeichnisbaum eine Datenbankdatendatei auswählen können. Wählen Sie hier die Datei *MediaBase.mdf* an ihrem neuen Standort aus und klicken Sie auf *OK*.

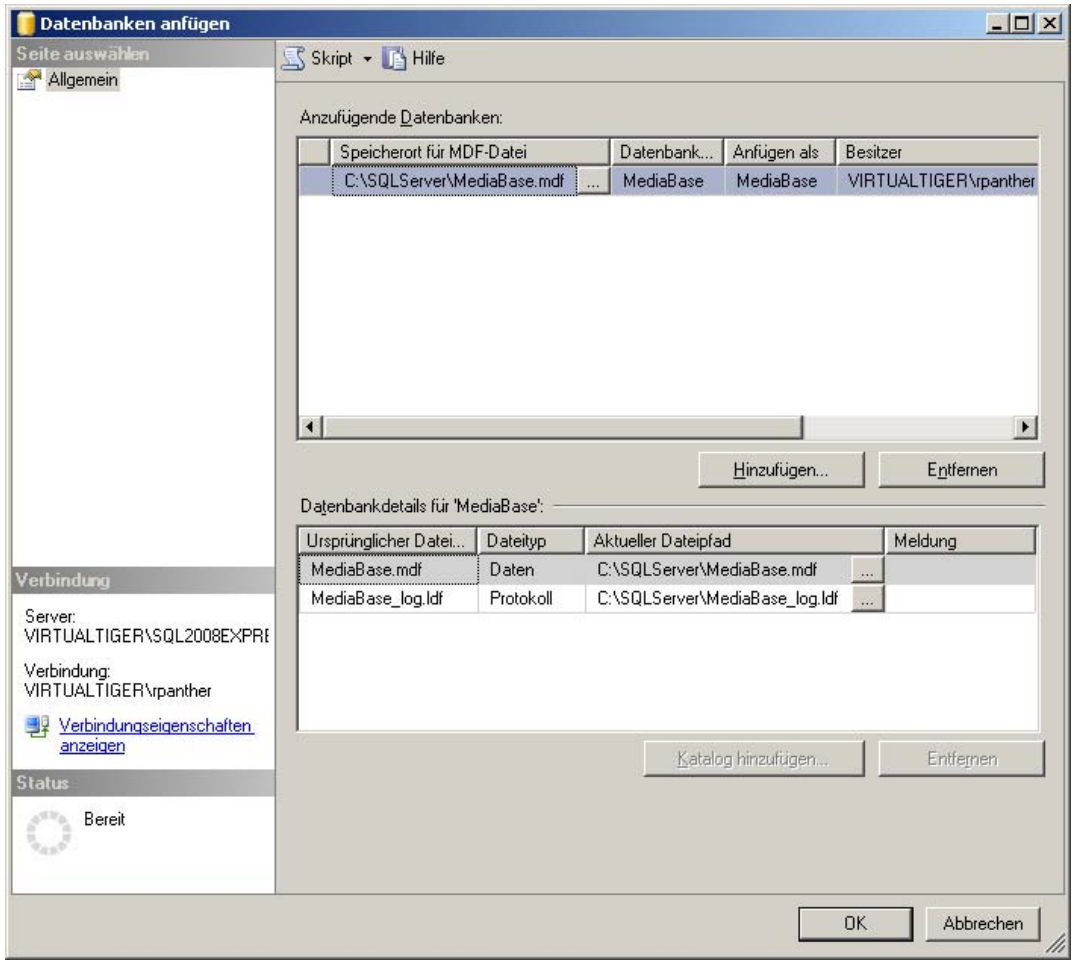


Abbildung 12.4: Das Dialogfeld zum Anfügen von Datenbanken

Hinweis: Datendatei und Protokolldatei in getrennten Verzeichnissen

Die Protokolldatei wird automatisch erkannt, so fern Sie sich entweder im ursprünglichen Verzeichnis oder im selben Verzeichnis wie die dazugehörende Datendatei befindet. Falls Sie die Protokolldatei in ein anderes Verzeichnis verschoben haben, müssen Sie auch diese noch explizit über den Verzeichnisbaum suchen, indem Sie die Schaltfläche mit den drei Punkten vor der Meldung *Nicht gefunden* anklicken.

7. Durch einen weiteren Klick auf *OK* wird die Datenbank wieder angefügt und steht in vollem Umfang zur Nutzung bereit.

Das Trennen der Datenbank *MediaBase* können Sie alternativ auch mit den folgenden SQL-Anweisungen durchführen:

```
USE master
GO
EXEC master.dbo.sp_detach_db @dbname = N'MediaBase'
GO
```

Sollten Sie vorher alle offenen Verbindungen trennen wollen, können Sie die folgende Anweisung ausführen (die sich auch in Verbindung mit dem Offlineschalten von Datenbanken verwenden lässt):

```
ALTER DATABASE MediaBase SET SINGLE_USER WITH ROLLBACK IMMEDIATE
GO
```

Mit der folgenden Variante der `CREATE DATABASE`-Anweisung wird die Datenbank dann im neuen Verzeichnis wieder angefügt:

```
USE master
GO
CREATE DATABASE MediaBase ON
( FILENAME = N'C:\SQLServer\MediaBase.mdf' ),
( FILENAME = N'C:\SQLServer\MediaBase_log.ldf' )
 FOR ATTACH
GO
```

Best Practices: Trennen und Verbinden zum Verschieben von Datenbanken nutzen

Das Trennen und Verbinden von Datenbanken können Sie auch verwenden, um eine Datenbank in ein anderes Verzeichnis umzuziehen. Das kann beispielsweise dann sinnvoll sein, wenn auf Laufwerk *C:* irgendwann der Plattenplatz knapp wird und Sie auf einer anderen Partition noch viel Platz frei haben. Auch aus Gründen der Datenbankperformance ist es sinnvoll, die Datenbankdateien nicht auf demselben Laufwerk zu speichern, auf dem auch die Systemdateien von Betriebssystem und SQL Server liegen. Durch die Verteilung der Dateien auf verschiedene Laufwerke wird auch die Belastung der Platten verteilt, sodass mehr parallele Plattenzugriffe erfolgen können. Wenn Sie noch mehr Festplatten im Zugriff des Servers haben, können Sie auch die Datendateien von den Protokolldateien trennen, um eine noch bessere Parallelität zu bekommen.

So flexibel diese Variante auch ist, bestehen auch hierbei immer noch zwei der weiter oben beschriebenen Probleme, sodass noch eine andere Variante erforderlich ist. Bevor wir uns aber den professionelleren Methoden der Datenbanksicherung zuwenden, muss ich etwas weiter ausholen und die theoretische Basis erläutern, die für das weitere Verständnis erforderlich ist.

12.2 Das Transaktionslog

Wie bereits in Abschnitt 4.1, *Erstellen von Datenbanken und Tabellen* angedeutet wurde, sind zum Speichern von Datenbanken mindestens zwei Dateien notwendig. Eine Zeilendatendatei (mit der Dateiendung .mdf), in der die eigentlichen Daten der Datenbank abgelegt sind, sowie eine Protokolldatei, die oft auch als Transaktionslog bezeichnet wird.

Im Transaktionslog werden alle Datenänderungen zusammen mit dem Zeitpunkt der Änderung in chronologischer Reihenfolge gespeichert. Wie der Name vermuten lässt, bildet dies die technische Grundlage für Transaktionen, da mit diesen Informationen noch offene Transaktionen wieder rückgängig gemacht werden können (siehe Abschnitt 8.6, *Sperren, Transaktionen und Deadlocks).* Andererseits kann über eine Sicherung dieser Informationen, sofern sie vollständig vorhanden sind, eine Wiederherstellung des Datenbankstandes zu einem beliebigen Zeitpunkt realisiert werden. Dies setzt aber voraus, dass beim Anlegen der Datenbank das richtige Wiederherstellungsmodell gewählt wurde. Das Wiederherstellungsmodell bedingt, welche Daten im Transaktionsprotokoll gespeichert werden und wie lange diese dort verbleiben.

Folgende Wiederherstellungsmodelle sind bei SQL Server verfügbar:

- *Einfach* – Im einfachen Wiederherstellungsmodell werden alle Änderungen aus Transaktionen – sobald sie abgeschlossen sind – in der Datenbankdatei gespeichert. Somit bleiben die Protokolldateien hier relativ klein, da sie nur die offenen Transaktionen aufnehmen müssen. Damit ist aber – neben den offenen Transaktionen – nur der aktuelle Datenbestand vorhanden. Es werden keine Informationen über die Historie gespeichert.
- *Vollständig* – Beim vollständigen Wiederherstellungsmodell bleiben neben den offenen auch abgeschlossene Transaktionen im Transaktionsprotokoll erhalten. Diese werden erst in die Datenbankdatei geschrieben, wenn eine Sicherung des Transaktionsprotokolls ausgeführt wurde. Da die Detailinfos dann bei Bedarf aus der Protokollsicherung wieder geholt werden können, werden diese im Transaktionsprotokoll selbst nicht mehr benötigt und der Platz hierfür wird wieder freigegeben. Die Inhalte abgeschlossener Transaktionen im Transaktionsprotokoll bilden damit eine Historie, mit deren Hilfe ein Datenstand zu einem beliebigen Zeitpunkt wiederhergestellt werden kann.
- *Massenprotokolliert* – Dieses Wiederherstellungsmodell entspricht weitgehend der Einstellung *Vollständig.* Lediglich bei Massenoperationen (wie *bcp*, `BULK INSERT`, `SELECT INTO`, `CREATE INDEX` etc.) werden hier weniger Daten ins Transaktionsprotokoll geschrieben, was die Performance der Massenoperation verbessert. Dadurch ist allerdings für diesen Zeitraum auch keine Zeitpunktwiederherstellung mehr möglich.

Das Wiederherstellungsmodell einer Datenbank kann auch im laufenden Betrieb geändert werden, was besonders im Zusammenhang mit der Einstellung *Massenprotokolliert* interessant sein kann.

Die Wahl des richtigen Wiederherstellungsmodells hängt primär davon ab, ob Sie die Möglichkeit des Rücksicherns zu einem beliebigen Zeitpunkt benötigen oder nicht. Für Datenbanken, die problemlos aus einer Komplettsicherung wiederherstellbar sind – beispielsweise weil sich deren Daten nur selten ändern – reicht das einfache Wiederherstellungsmodell völlig aus. Dasselbe gilt, wenn die Detaildaten so unkritisch sind, dass Sie notfalls auch mit dem Stand der letzten Vollsicherung weiterarbeiten können.

Für typische OLTP-Systeme, bei denen die Datenbank sensible Daten enthält, von denen keine noch so kleine Änderung verloren gehen sollte, ist dagegen das vollständige Wiederherstellungsmodell zu wählen.

Die Einstellung *Massenprotokolliert* macht – wie weiter oben beschrieben – eher temporär Sinn. So kann es sinnvoll sein, eine Datenbank, die normalerweise auf *Vollständig* eingestellt ist, für größere Datenimporte kurzzeitig auf *Massenprotokolliert* umzustellen.

12.3 Sichern und Wiederherstellen von Datenbanken

Kommen wir nun zu den von SQL Server explizit für diesen Zweck vorgesehenen Möglichkeiten, Datenbanken zu sichern und wiederherzustellen. Dabei stellt SQL Server zwei grundlegende Varianten von Datenbanksicherungen zur Verfügung:

- *Vollständige Sicherung* – Hier werden alle Daten gesichert, die in der Datenbankdatei vorhanden sind. Dazu kommen die Daten, die sich aus abgeschlossenen Transaktionen im Transaktionslog ergeben. Dadurch, dass keine leeren Speicherbereiche geschrieben werden, benötigt die Sicherungsdatei meist erheblich weniger Speicherplatz als die Summe aus Datenbankdatei und Protokolldatei.
- *Differenzielle Sicherung* – Dieser Sicherungstyp sichert alle seit der letzten vollständigen Sicherung geänderten Daten. Dadurch können Sie häufiger Sicherungen erstellen, ohne jedes Mal den kompletten Speicherplatz für eine Vollsicherung zu benötigen. Zum Wiederherstellen werden dann zuerst die letzte Vollsicherung und anschließend die aktuellste differenzielle Sicherung eingespielt.

Zusätzlich zu den beiden Grundvarianten gibt es – sofern Sie das vollständige oder massenprotokollierte Wiederherstellungsmodell verwenden – noch die Möglichkeit, das Transaktionslog zu sichern. Dies sollte man auch regelmäßig tun, da hierbei der Platz, den die abgeschlossenen Transaktionen im Transaktionsprotokoll beanspruchen, wieder freigegeben wird (man sagt hierzu auch, dass das Protokoll abgeschnitten wird). Dadurch verringert sich zwar nicht die Größe der Protokolldatei, aber der Platz wird wieder für andere (offene) Transaktionen nutzbar, sodass die Protokolldatei zumindest nicht unnötig vergrößert werden muss.

Hintergrundinfo: Komprimierte Sicherungen

Ab SQL Server 2008 Enterprise lässt sich die Sicherung (egal welchen Typs) auch noch komprimiert erzeugen, was weiteren Speicherplatz spart (dafür aber etwas mehr Prozessorlast erzeugt). Für Anwender der Express Edition ist es jedoch sinnvoll zu wissen, dass man mit jeder Edition von SQL Server 2008 komprimierte Sicherungen wiederherstellen kann.

Erstellen wir nun eine Sicherung der Datenbank *MediaBase:*

1. Stellen Sie im SQL Server Management Studio eine Verbindung zur lokalen Serverinstanz *SQL2008Express* her und öffnen Sie im Objekt-Explorer den Zweig, der die *Datenbanken* beinhaltet.
2. Klicken Sie hier die Datenbank *MediaBase* mit der rechten Maustaste an und wählen Sie dann im Kontextmenü den Befehl *Tasks/Sichern* aus.
3. Es erscheint das Dialogfeld *Datenbanken sichern*, das aus zwei Seiten besteht. Die meisten Einstellungen werden auf der Seite *Allgemein* vorgenommen, die Seite *Optionen* hält noch ein paar zusätzliche Parameter bereit. Doch beginnen wir mit der Seite *Allgemein*:

   Als *Quelle* ist bereits die Datenbank *MediaBase* ausgewählt, da Sie über diese das Dialogfeld aufgerufen haben. Das *Wiederherstellungsmodell* wird hier lediglich angezeigt, da dies eigentlich eine Datenbankeigenschaft ist, die aber im direkten Zusammenhang zur Datenbanksicherung steht. Beim *Sicherungstyp* können Sie lediglich zwischen *Vollständig* und *Differenziell* auswählen, wobei für unsere Beispieldatenbank eine Vollsicherung ausreichen sollte. (Das Sichern des Transaktionsprotokolls wird nicht angeboten, da wir uns bei der Datenbank *MediaBase* beim Anlegen für das einfache Wiederherstellungsmodell entschieden haben.) Der Name des Sicherungssatzes wurde bereits vorbelegt, kann aber überschrieben werden. Außerdem können Sie das Feld *Beschreibung* mit einem

Kommentar füllen. In den nächsten Feldern können Sie definieren, nach wie vielen Tagen bzw. zu welchem Datum der Sicherungssatz abläuft und damit wieder überschrieben werden kann. Im Bereich *Ziel* können Sie einerseits auswählen, ob die Sicherung auf die Festplatte oder ein Band erfolgen soll (sofern ein entsprechendes Bandlaufwerk verfügbar ist). Darunter ist ein kompletter Dateipfad angegeben, der für die Sicherung verwendet wird. Dabei wird als Dateiname automatisch *MediaBase.bak* vorgeschlagen. Wenn Ihnen Pfad oder Dateiname nicht gefallen, können Sie die Sicherungsdatei durch die *Entfernen*-Schaltfläche aus der Liste nehmen und über die *Hinzufügen*-Schaltfläche neu angeben. Sollten Sie eine bestehende Sicherungsdatei verwenden, können Sie über die Schaltfläche *Inhalt* anzeigen, welche Sicherungen bereits in der Datei vorhanden sind. Übernehmen Sie hier die Einstellungen wie in Abbildung 12.5 angegeben und wechseln Sie dann zur Seite *Optionen*.

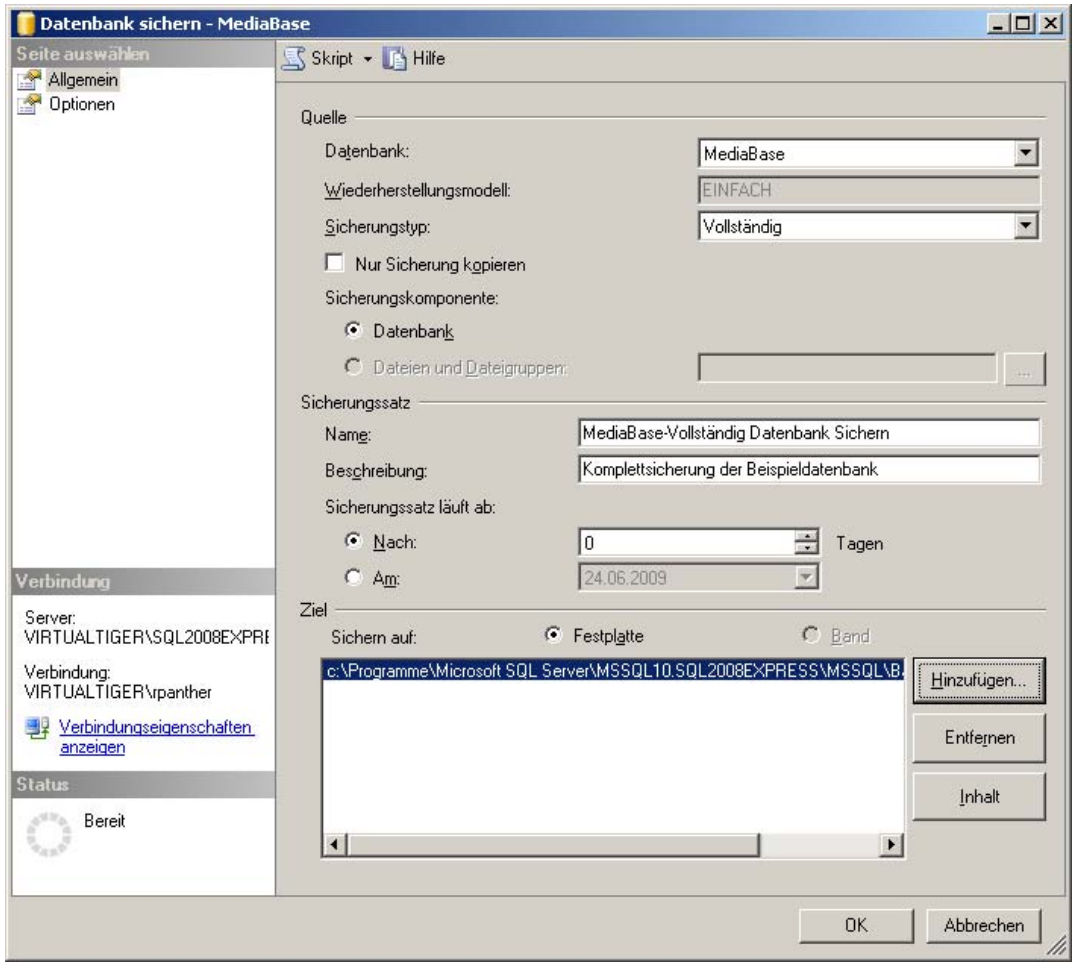


Abbildung 12.5: Das Dialogfeld zum Sichern von Datenbanken

4. Auf der Seite *Optionen* beschäftigen sich die meisten Einstellungen damit, wie mit Mediensätzen und Sicherungssätzen verfahren werden soll.

Hintergrundinfo: Medien, Mediensätze und Sicherungssätze

Zum besseren Verständnis der Optionen ist zunächst die Klärung von ein paar Begriffen nötig:

- **Medien** Ein Medium (oder auch Sicherungsmedium) ist eine Datei auf Festplatte oder Band, welche die Sicherungsdaten aufnimmt.
- **Mediensatz** Ein Mediensatz kann ein oder mehrere Medien enthalten. Letzteres ist dann der Fall, wenn die Sicherungsdaten auf mehrere Dateien (und meist auch Verzeichnisse) verteilt werden, um durch parallele Zugriffe bessere Performance zu erreichen.
- **Sicherungssatz** Ein Sicherungssatz ist die Gesamtheit der Daten, die mit einer Sicherung gespeichert wurde. Ein Mediensatz kann mehrere Sicherungssätze beinhalten, um beispielsweise den Stand der Datenbank zu verschiedenen Zeitpunkten zu speichern.

Die zentrale Frage dabei ist, ob ein neuer Mediensatz erstellt oder ein vorhandener genutzt werden soll. Wird ein neuer Mediensatz erstellt, können Sie dessen Namen und eine Beschreibung angeben. Wenn Sie einen vorhandenen Mediensatz nutzen, müssen Sie auswählen, ob bereits vorhandene Sicherungssätze gelöscht werden sollen (*Alle vorhandenen Sicherungssätze überschreiben*) oder ob der neue Sicherungssatz einfach hinzuzufügen ist (*An vorhandenen Sicherungssatz anfügen*). Des Weiteren können Sie über ein Kontrollkästchen aktivieren, dass das Ablaufdatum von bestehenden Sicherungssätzen im Mediensatz geprüft werden soll. Wählen Sie hier aus, dass ein neuer Mediensatz genutzt werden soll, und geben Sie dafür als Namen **MediaBase Full Backup** an.

Im Bereich *Zuverlässigkeit* können Sie unter anderem angeben, ob die Sicherung überprüft werden soll, nachdem sie erstellt wurde (was immer sinnvoll ist, wenn die Zeit es zulässt). Etwas irreführend ist die Möglichkeit, im Bereich *Komprimierung* auszuwählen, ob die Sicherung komprimiert erfolgen soll oder nicht, da die Backup-Komprimierung von der Express Edition nicht unterstützt wird. Diese Einstellung kann man aber ohnehin am besten auf *Standardservereinstellung verwenden* belassen und die Komprimierung von Sicherungen dann serverweit aktivieren, wenn man die passende SQL Server Edition im Einsatz hat.

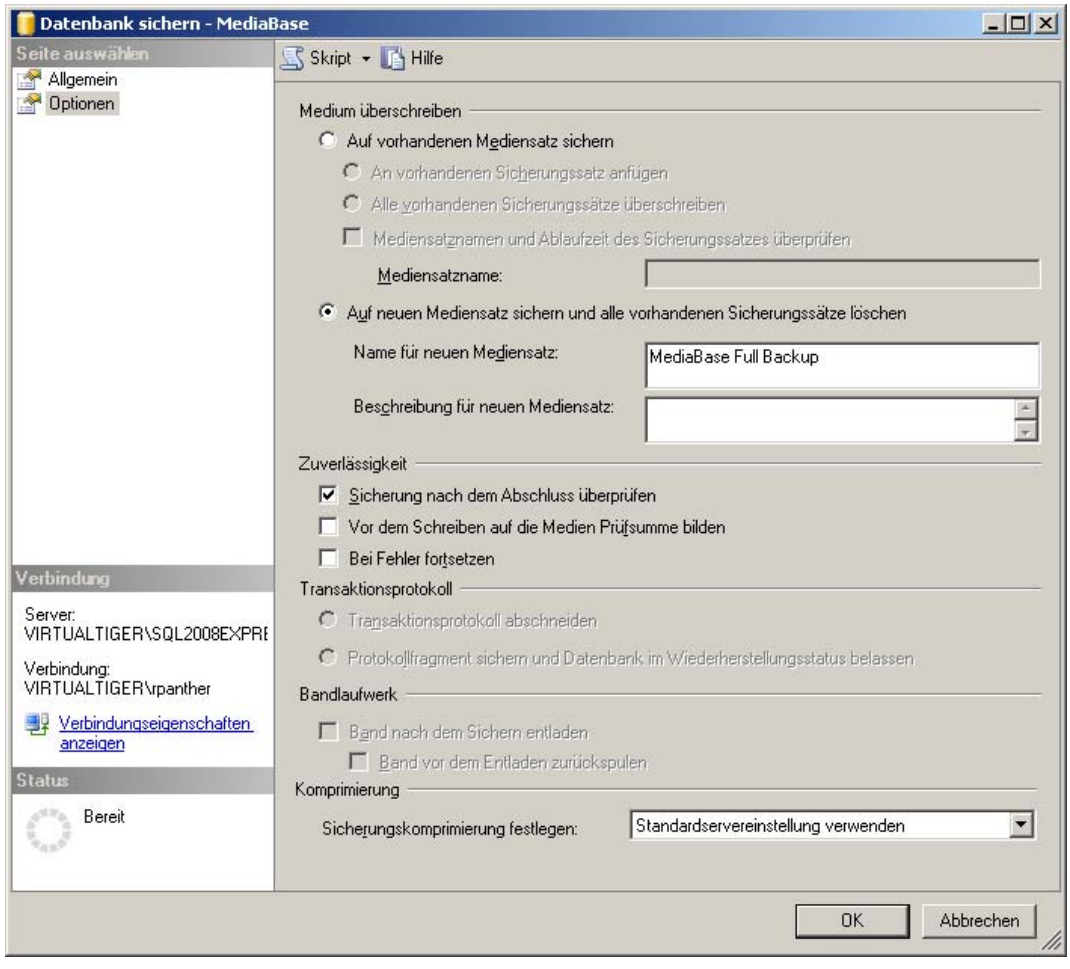


Abbildung 12.6: Die *Optionen*-Seite des Dialogfelds zum Sichern von Datenbanken

5. Klicken Sie auf *OK* und Sie erhalten nach kurzer Zeit die Meldung, dass die Sicherung der Datenbank erfolgreich abgeschlossen wurde.

Die entsprechenden SQL-Anweisungen, um die Sicherung wie beschrieben durchzuführen, sind wie folgt:

```
BACKUP DATABASE [MediaBase] TO  DISK = N'c:\Programme\Microsoft SQL
Server\MSSQL10.SQL2008EXPRESS\MSSQL\Backup\MediaBase.bak' WITH FORMAT, INIT,  MEDIANAME = N'MediaBase Full Backup',
➥NAME = N'MediaBase-Vollständig Datenbank Sichern', SKIP, NOREWIND, NOUNLOAD,  STATS = 10
GO
declare @backupSetId as int
select @backupSetId = position from msdb..backupset where database_name=N'MediaBase' and backup_set_id=(select
➥max(backup_set_id) from msdb..backupset where database_name=N'MediaBase' )
if @backupSetId is null begin raiserror(N'Fehler beim Überprüfen. Sicherungsinformationen für die MediaBase-
➥Datenbank wurden nicht gefunden.', 16, 1) end
RESTORE VERIFYONLY FROM  DISK = N'c:\Programme\Microsoft SQL
Server\MSSQL10.SQL2008EXPRESS\MSSQL\Backup\MediaBase.bak' WITH  FILE = @backupSetId,  NOUNLOAD,  NOREWIND
GO
```

Dabei reicht die erste Anweisung für die eigentliche Sicherung aus, die restlichen Anweisungen werden für die Überprüfung der Sicherung benötigt. Dazu wird erst die aktuellste Sicherungssatz-ID aus den entsprechenden Systemtabellen gelesen und mithilfe dieser ID dann eine Überprüfung der Gültigkeit des Sicherungssatzes durchgeführt.

Bevor wir die Datenbank nun aus der Sicherung wiederherstellen, muss eine Änderung darin durchgeführt werden, damit man auch nachvollziehen kann, dass die Wiederherstellung funktioniert hat. Proben wir also den Ernstfall, indem wir eine Tabelle komplett löschen und dann die Sicherung wieder einspielen:

1. Stellen Sie im SQL Server Management Studio eine Verbindung zur lokalen Serverinstanz *SQL2008Express* her und öffnen Sie ein Abfragefenster.
2. Führen Sie hier im Abfragefenster folgendes Skript aus, um die Tabelle *dbo.CDMitTracks* komplett zu entfernen (alternativ können Sie auch eine beliebige andere Tabelle löschen):

```
USE MediaBase
DROP TABLE dbo.CDMitTracks
GO
```

3. Prüfen Sie im Zweig *Datenbanken/MediaBase/Tabellen*, dass die Tabelle wirklich nicht mehr existiert (eventuell müssen Sie die Ansicht vorher aktualisieren).
4. Klicken Sie nun mit der rechten Maustaste auf die Datenbank *MediaBase* und wählen Sie anschließend die Option *Tasks/Wiederherstellen/Datenbank*. Es erscheint das Dialogfeld zum Wiederherstellen von Datenbanken, das ebenfalls über zwei Seiten mit Einstellungen verfügt.
5. Auf der Seite *Allgemein* ist die Datenbank *MediaBase* bereits ausgewählt. Die Option des Zeitpunkts spielt momentan keine Rolle, da wir keine Transaktionsprotokollsicherung verwenden. Für die Angabe der Quelle haben Sie zwei Möglichkeiten: Die Variante *Aus Datenbank* stellt alle Sicherungen zur Auswahl, die auf diesem Rechner für die ausgewählte Datenbank erstellt wurden. Bei der Variante *Von Medium* müssen Sie dagegen die Backup-Datei selbst heraussuchen. Diese Variante ist erforderlich, wenn Sie eine Sicherung auf einem Server, auf dem sie nicht erstellt wurde, einspielen wollen. Egal welchen Weg Sie zur Auswahl der Sicherungsmedien verwendet haben, ist anschließend noch auszuwählen, welche Sicherungssätze von diesen Medien wiederhergestellt werden sollen. Setzen Sie die Einstellungen so, wie in Abbildung 12.7 dargestellt und wechseln Sie dann zur Seite *Optionen*.

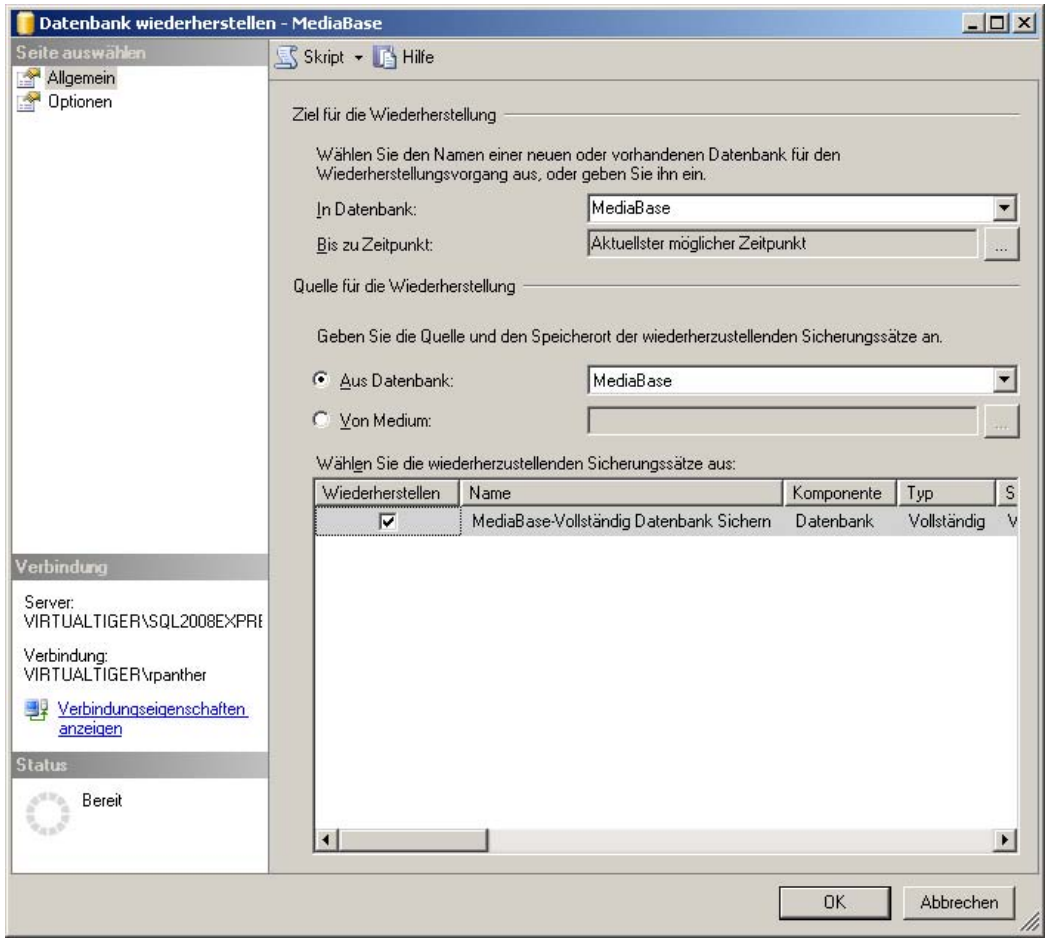


Abbildung 12.7: Das Dialogfeld zum Wiederherstellen von Datenbanken

6. Auf der Seite *Optionen* ist festzulegen, ob eine vorhandene Datenbank überschrieben werden kann. Dazu können Sie detailliert angeben, unter welchen Pfaden und Dateinamen die Datenbank- und Protokolldateien wiederhergestellt werden sollen. Wichtig ist auch die Wahl des *Wiederherstellungsstatus*: Die erste Variante (*RESTORE WITH RECOVERY*) wählen Sie, wenn dies die einzige oder letzte Sicherung ist, die Sie wiederherstellen. Wenn Sie vorhaben, nach dem Einspielen der Voll- oder Differenzialsicherung auch noch Transaktionsprotokolle wiederherzustellen, wählen Sie die zweite Variante (*RESTORE WITH NORECOVERY*). Die dritte Variante (*RESTORE WITH STANDBY*) ist vor allem dann sinnvoll, wenn Sie verhindern möchten, dass nach dem Wiederherstellen gleich mit der Datenbank weitergearbeitet wird, weil Sie beispielsweise vorher noch eine gründliche Fehleranalyse betreiben wollen. Da wir lediglich eine Vollsicherung wiederherstellen wollen, ist hier die Variante *RESTORE WITH RECOVERY* auszuwählen.

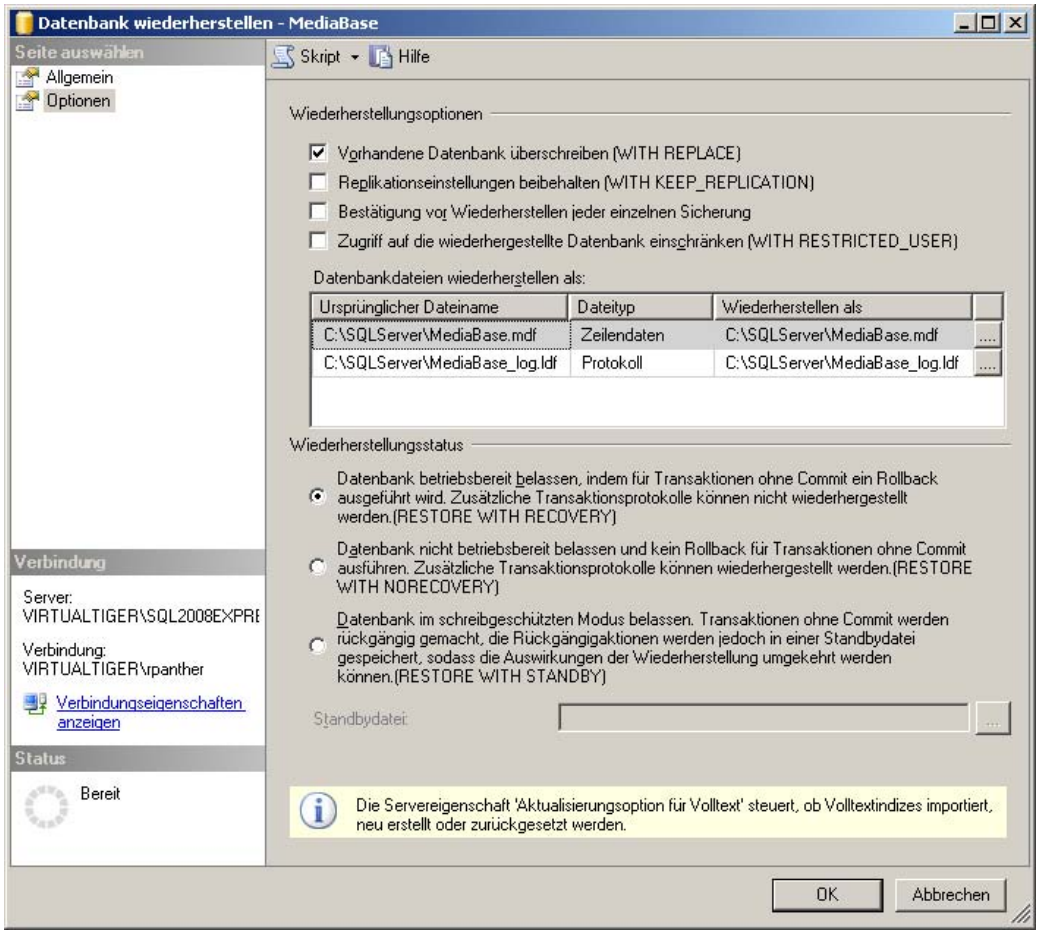


Abbildung 12.8: Die *Optionen*-Seite des Dialogfelds zum Wiederherstellen von Datenbanken

7. Wenn Sie nun auf *OK* klicken, wird die Sicherung wiederhergestellt. Überprüfen Sie anschließend im Objekt-Explorer, dass die gelöschte Datei wieder vorhanden ist.

Das gerade geschilderte Wiederherstellen der Vollsicherung ist alternativ auch mit folgender SQL-Anweisung durchführbar:

```
RESTORE DATABASE [MediaBase] FROM  DISK = N'c:\Programme\Microsoft SQL
Server\MSSQL10.SQL2008EXPRESS\MSSQL\Backup\MediaBase.bak' WITH  FILE = 1,  NOUNLOAD,  REPLACE,  STATS = 10
GO
```

Wahl der richtigen Sicherungsstrategie

Ein wichtiger Faktor im Umfeld der Sicherung von Datenbanken liegt darin, diese auch regelmäßig durchzuführen. Dazu bedarf es – je nach Umfeld und individuellen Anforderungen – einer sinnvollen Sicherungsstrategie, zu der auch das Zusammenspiel von Wiederherstellungsmodell und Sicherungsart gehört. Es folgen ein paar mögliche Beispiele:

Anwendungstyp 1: Referenzdatenbank, die in unregelmäßigen Abständen neu aufgebaut wird

- Wiederherstellungsmodell: einfach
- Sicherung:
  - vollständige Sicherung: jeweils direkt nach dem Befüllen bzw. Aktualisieren
  - Differenzielle Sicherung: (keine)
  - Transaktionsprotokoll-Sicherung: (keine)
- Vorgehen im Notfall:
  - Wiederherstellen der letzten vollständigen Sicherung

Anwendungstyp 2: Unkritische Daten, die sich regelmäßig ändern

- Wiederherstellungsmodell: einfach
- Sicherung:
  - vollständige Sicherung: einmal pro Woche (am Wochenende)
  - Differenzielle Sicherung: täglich (Mo-Fr), jeweils nachts
  - Transaktionsprotokoll-Sicherung: (keine)
- Vorgehen im Notfall:
  - Wiederherstellen der letzten vollständigen Sicherung
  - Wiederherstellen der letzten differenziellen Sicherung

Anwendungstyp 3: Kritische Daten, die sich regelmäßig ändern (großes Transaktionslog)

- Wiederherstellungsmodell: vollständig
- Sicherung:
  - vollständige Sicherung: einmal pro Woche (am Wochenende)
  - Differenzielle Sicherung: (keine)
  - Transaktionsprotokoll-Sicherung: jeweils unmittelbar vor der vollständigen Sicherung sowie täglich (Mo-Fr), jeweils nachts
- Vorgehen im Notfall:
  - Sichern des Transaktionsprotokolls
  - Wiederherstellen der letzten vollständigen Sicherung
  - Wiederherstellen der Transaktionsprotokolle seit der letzten vollständigen Sicherung (bis zum gewünschten Zeitpunkt)

Anwendungstyp 4: Kritische Daten, die sich regelmäßig ändern (kleines Transaktionslog)

- Wiederherstellungsmodell: vollständig
- Sicherung:
  - vollständige Sicherung: einmal pro Woche (am Wochenende)
  - Differenzielle Sicherung: täglich (Mo-Fr), jeweils nachts
  - Transaktionsprotokoll-Sicherung: jeweils unmittelbar vor den vollständigen und differenziellen Sicherungen sowie täglich (Mo-Fr) um 12 Uhr
- Vorgehen im Notfall:
  - Sichern des Transaktionsprotokolls
  - Wiederherstellen der letzten vollständigen Sicherung
  - Wiederherstellen der letzten differenziellen Sicherung
  - Wiederherstellen der Transaktionsprotokolle seit der letzten differenziellen Sicherung (bis zum gewünschten Zeitpunkt)

Best Practices: Automatisierung von Datenbanksicherungen

Während man das Erstellen von Sicherungen bei den größeren Editionen von SQL Server über einen SQL Server Agent-Job (oder gar einen Wartungsplan) automatisieren kann, muss man bei der Express Edition etwas mehr manuelle Vorb ereitungsarbeit leisten. Hier sind separate SQL-Skripts zu erstellen, die dann über das Kommandozeilentool SQLCMD aufgerufen werden können. Die Automatisierung des Aufrufs lä sst sich recht einfach mit der Funktionalität *Geplante Tasks* des Windows-Betriebssystems aufrufen, die in der Systemsteuerung zu finden ist.

12.4 Import und Export von Daten

Eine Aufgabe, die im Umfeld von Datenbanken immer wieder auftaucht, ist der Im- und Export von Daten aus bzw. in andere Formate. Auch wenn hier die größeren Editionen von SQL Server – insbesondere mit den SQL Server Integration Services – sehr aufwendige Varianten zur Verfügung stellen, bietet auch die Express Edition einige Möglichkeiten hierzu an.

Der Import-/Export-Assistent

Die sicherlich komfortabelste Variante, Daten von einer beliebigen Quelle in den SQL Server zu importieren oder aber Daten vom SQL Server in ein beliebiges Ziel zu exportieren, ist der *Import-/Export-Assistent*. Dieser kann entweder aus dem SQL Server Management Studio aufgerufen werden (durch einen Klick mit der rechten Maustaste im Objekt-Explorer auf die Datenbank und anschließende Auswahl von *Tasks/Daten importieren* bzw. *Tasks/Daten exportieren*), oder aber man nutzt den entsprechenden Eintrag im Windows-Startmenü: *Start/Alle Programme/Microsoft SQL Server 2008/Daten importieren und exportieren (32-Bit)*.

Mit dem Import-/Export-Asssistenten können verschiedene Datenquellen bzw. -ziele genutzt werden:

- .Net Framework Data Provider for ODBC
- .Net Framework Data Provider for Oracle
- .Net Framework Data Provider for SQL Server
- Flatfilequelle/-ziel
- Microsoft Access
- Microsoft Excel
- Microsoft OLE DB Provider for Analysis Services 10.0
- Microsoft OLE DB Provider for Data Mining Services
- Microsoft OLE DB Provider for Internet Publishing
- Microsoft OLE DB Provider for OLAP Services 8.0
- Microsoft OLE DB Provider for Oracle
- Microsoft OLE DB Provider for SQL Server
- SQL Server Native Client 10.0

Je nach zusätzlich installierter Software können hier auch noch weitere Datenquellen und -ziele zur Auswahl stehen. Nutzen wir nun den Import-/Export-Assistenten, um die Tabelle *dbo.DVD* in eine Excel-Datei zu exportieren:

1. Rufen Sie den Import-/Export-Assistenten über das Windows-Startmenü auf.
2. Bei den *Datenquellen* stehen – wie in der Liste oben zu sehen – verschiedene Varianten bereit, um auf SQL Server-Datenbanken zuzugreifen. Wählen Sie hier den *SQL Server Native Client 10.0*, da dieser den direktesten (und damit performantesten) Zugriff auf den SQL Server ermöglicht.

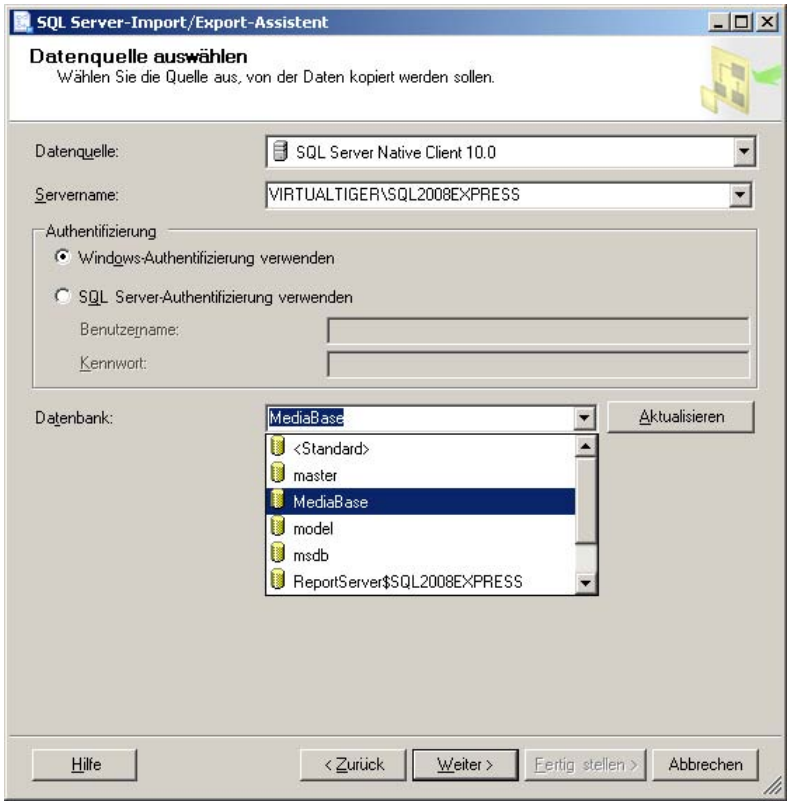


Abbildung 12.9: Auswahl der Datenquelle

3. Wählen Sie nun die lokale Datenbank-Serverinstanz, *Windows-Authentifizierung* sowie die Datenbank *MediaBase* aus und klicken Sie anschließend auf die Schaltfläche *Weiter*.
4. Als Ziel stehen dieselben Varianten wie für die Quelle zur Auswahl. Wählen Sie hier das Ziel *Microsoft Excel* und geben Sie einen gültigen Excel-Dateipfad (z.B. *C:\SQLServer\MediaBaseDVDs.xls)* ein. Als Excel-Version wählen Sie *Microsoft Excel 97-2003,* da diese Variante von den meisten Excel-Versionen verwendet werden kann. Damit die Spaltennamen auch in die Excel-Datei übernommen werden, lassen Sie das entsprechende Kästchen aktiviert und klicken dann auf *Weiter*.
5. Im nächsten Schritt des Assistenten können Sie wählen, ob Sie komplette Tabellen bzw. Sichten kopieren oder lieber eine SQL-Abfrage zur Definition der Datenquelle verwenden wollen. Bestätigen Sie die erste Variante, indem Sie auf *Weiter* klicken.
6. Es folgt eine Liste zur Auswahl der Quelltabellen und -sichten, in der Sie die Tabelle *dbo.DVD* anklicken. Dazu können Sie den Namen für das Ziel ändern, über die Schaltfläche *Vorschau* einen Blick in die Quelltabelle werfen oder über die betreffende Schaltfläche Zuordnungen bearbeiten. Bei Letzterem können Sie die Namen und Datentypen der Zielspalten angeben oder doch noch die SQL-Anweisung zum Abfragen der Quelldaten anpassen. Belassen Sie es jetzt aber bei der Auswahl der Tabelle und klicken Sie auf *Weiter*.

7. Im nächsten Schritt werden Sie aufgefordert, die Datentypzuordnung zu prüfen und anzugeben, wie während des Imports verfahren werden soll, wenn einzelne Spalten nicht zum gewählten Datentyp passen oder aber einfach zu groß für die gewählte Feldlänge sind. Als mögliche Reaktionen stehen hier *Fehler* und *Ignorieren* zur Auswahl. Belassen Sie hier die Voreinstellungen und klicken Sie einmal auf *Weiter*.

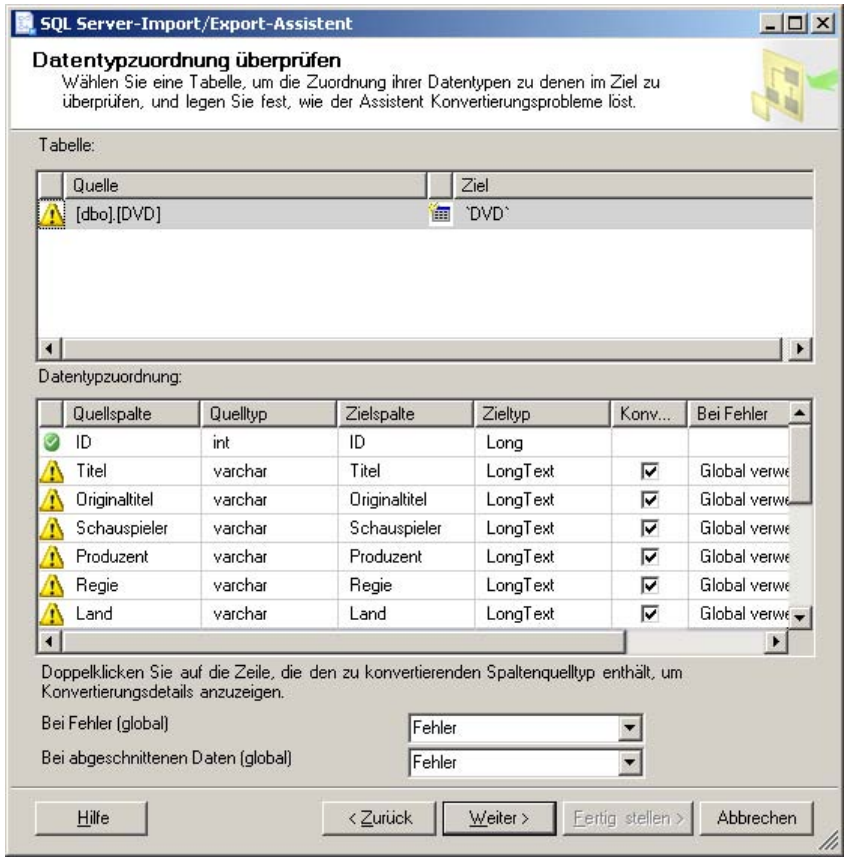


Abbildung 12.10: Prüfung der Datentypzuordnung

8. Als Nächstes könnten Sie – wenn Sie eine Standard, Enterprise oder Developer Edition von SQL Server verwenden würden – wählen, ob der Export sofort ausgeführt werden oder als SSIS-Paket gespeichert werden soll. Da wir hier lediglich mit der Express Edition arbeiten, steht allerdings nur die Option *Sofort ausführen* zur Auswahl, die Sie nun mit einem Klick auf *Weiter* bestätigen.
9. Es folgt eine kurze Zusammenfassung der vorgenommenen Einstellungen. Mit einem Klick auf *Fertig stellen* starten Sie den eigentlichen Exportvorgang, während dessen Sie laufend über den aktuellen Status informiert werden und sich auch eventuell auftretende Fehler- oder Warnhinweise ansehen können. Nach Fertigstellung des Exports können Sie noch einen Bericht dazu anzeigen oder speichern.
10. Anschließend können Sie – sofern Sie Microsoft Excel installiert haben – einen Blick in die exportierte Tabelle werfen.

Hintergrundinfo: SQL Server Integration Services

Auch wenn dieser Dienst eigentlich den größeren Editionen von SQL Server vorbehalten ist, verwendet der Import-/Export-Assistent intern Funktionalitäten der SQL Server Integration Services.

Die eher codelastige Variante zum Import von Daten ist das Masseneinfügen. Hierzu stellt SQL Server zwei Varianten bereit, die im Prinzip dasselbe leisten. Das eine ist das SQL-Kommando `BULK INSERT`, das zweite das Kommandozeilentool *bcp*.

Masseneinfügen per BULK INSERT

Über die Anweisung `BULK INSERT` können Sie Daten aus einem sogenannten Flatfile einlesen. Dabei handelt es sich entweder um eine Rohdatei, in der die Daten Byte für Byte abgelegt sind, oder um eine Textdatei, in der die Zeilen und Spalten entweder aufgrund von festen Feldlängen oder durch vordefinierte Trennzeichen unterteilt sind. Typischer Vertreter hierfür sind die CSV-Dateien (CSV steht für *comma separated values,* also kommagetrennte Werte), in denen die Felder normalerweise durch Semikolons unterteilt sind, während die Datenzeilen – wie in einer Textdatei – durch ein einfaches Return voneinander getrennt sind. Als typische Feldtrennzeichen sind neben dem Semikolon noch das Komma oder der Tabulator sehr verbreitet, wobei man dann immer noch von einer CSV-Datei spricht, auch wenn das Komma dann gar nicht mehr als Trennzeichen verwendet wird.

Die `BULK INSERT`-Anweisung hat folgende Grundform:

```
BULK INSERT zieltabelle FROM quelldatei WITH (einstellungen)
```

Statt der Zieltabelle kann auch eine Sicht angegeben werden und auch die Quelldatei erklärt sich eigentlich von selbst. Komplizierter wird es da im Bereich *Einstellungen,* denn hier können zahlreiche Parameter gesetzt werden, von denen ich die wichtigsten kurz vorstellen will.

Tabelle 12.1: Übersicht der wichtigsten `BULK INSERT`-Einstellungen

Parameter	Bedeutung
`BATCHSIZE`	Anzahl von Zeilen in einer Einfügetransaktion
`FIELDTERMINATOR`	Feldtrennzeichen, Standardeinstellung ist '\t' (Tabstoppzeichen)
`FORMATFILE`	vollständiger Pfad einer Formatdatei
`KEEPIDENTITY`	IDENTITY-Spalten nicht neu generieren
`KEEPNULLS`	leere Spalten in `NULL` umwandeln (sonst Standardwerte)
`ROWTERMINATOR`	Zeilentrennzeichen, Standardeinstellung ist '\r\n' (CR/Linefeed)
`TABLOCK`	steuert exklusive Sperre der gesamten Tabelle während des Masseneinfügens

Probieren wir die `BULK INSERT`-Anweisung nun aus, indem wir zuerst eine Textdatei zum Importieren erzeugen und diese dann per `BULK INSERT` in die Tabelle *dbo.CD* einlesen:

1. Erstellen Sie mit Notepad oder einem beliebigen anderen Texteditor eine Datei mit Namen *CD-Import.txt,* die eine oder mehrere Zeilen enthält, in der Inhalte für alle Felder der Tabelle *dbo.CD* – jeweils mit Semikolons getrennt – angegeben sind. Beispiel:

   ```
   1;Cocker;Joe Cocker;1;1986;2;Pop/Rock
   ```

2. Verbinden Sie sich nun im SQL Server Management Studio über Windows-Authentifizierung mit der lokalen Serverinstanz *SQL2008Express* und öffnen Sie ein neues Abfragefenster.
3. Lassen Sie sich mit folgendem Skript den aktuellen Inhalt der Tabelle *dbo.CD* anzeigen:

```
USE MediaBase
SELECT * FROM dbo.CD
```

4. Führen Sie nun die folgende `BULK INSERT`-Anweisung aus und werfen Sie danach wieder einen Blick in die Tabelle *dbo.CD:*

```
BULK INSERT dbo.CD FROM N'c:\SQLServer\CDImport.txt'
WITH (FIELDTERMINATOR=';', ROWTERMINATOR='\r\n')

SELECT * FROM dbo.CD
```

5. Sie werden sehen, dass die neuen Zeilen aus der Textdatei eingefügt wurden, dabei aber die IDs automatisch durch neu generierte ersetzt wurden. Hätten Sie den `KEEPIDENTITY`-Parameter angegeben, hätte der `BULK INSERT` versucht, die in der Datei angegebenen ID-Werte zu verwenden, was aber zu einem Fehler geführt hätte, wenn dadurch IDs doppelt vergeben worden wären.

Wenden wir uns nun der Alternativmethode für das Masseneinfügen von Daten zu.

BCP – Masseneinfügen über die Kommandozeile

Das Bulk Copy Program *(bcp)* entspricht weitgehend dem Befehl `BULK INSERT`, wird aber von der Kommandozeile aus gestartet. Die Parameter haben zwar andere Namen, entsprechen vom Inhalt aber weitgehend denen vom `BULK INSERT`. Lediglich ein paar Parameter zum Herstellen der Verbindung zum SQL Server kommen hinzu, da von der Kommandozeile aus ja erst eine Verbindung zum Server aufgebaut werden muss.

Hinweis: Kleine Anekdote zur Abkürzung BCP

Ich wurde kürz lich von einem Kunden gefra gt, wofür denn die Abkürzun g *bcp* stehe. In den gängigen Su chmaschinen im Internet sei nic hts Hilfreiches dazu zu finden. Nachdem ich der fragenden Person erklärt hatte, worum es sich bei *bcp* handelt, war ich natürlich neugierig und habe selbst mal in der Suchmaschine meines Vertrauens nach dieser Abkürzung suchen lassen. Als e ines der ersten S uchergebnisse war dann die Bruderschaft Christlicher Pfadfinder aus einer norddeut schen Kleinsta dt zu finden, wä hrend das eigentliche Bulk Copy Progra m erst weiter hinten in der Ergebnisliste auftauchte. (Seitdem müssen aber die i nternen Statistiken aktualisiert worden sein. Inzwischen sehen die Ergebnisse zum Glück etwas anders aus.)

Die Grundform des Aufrufs für das *bcp*-Tool sieht wie folgt aus:

```
bcp tabelle|abfrage in|out|queryout|format datendatei parameter
```

Hier eine Auflistung der wichtigsten Parameter:

Tabelle 12.2: Übersicht der wichtigsten bcp-Parameter

Parameter	Bedeutung
-f	Formatdatei
-b	Batch Size
-t	Feldtrennzeichen
-r	Zeilentrennzeichen
-c	Standardformat (char als Speichertyp, \t als Feldtrennzeichen, \r\n als Zeilentrennzeichen)
-S	Servername\Instanz
-T	Windows-Authentifizierung
-U	Anmeldung
-P	Passwort
-k	leere Spalten in NULL umwandeln (sonst Standardwerte)
-E IDEN	TITY-Spalten nicht neu generieren

Die Entsprechung für die vorhin verwendete BULK INSERT-Anweisung

```
BULK INSERT dbo.CD FROM N'c:\SQLServer\CDImport.txt'
WITH (FIELDTERMINATOR=';', ROWTERMINATOR='\r\n')
```

wäre folgender *bcp*-Aufruf ...

```
bcp MediaBase.dbo.CD in c:\SQLServer\CDImport.txt -c -t ; -r \r\n -T -S .\SQL2008Express
```

Dabei wurde der Parameter -S angegeben, um die zu verwendende Serverinstanz zu spezifizieren, außerdem der Parameter -T, damit die Anmeldung am Server über die Windows-Authentifizierung erfolgt (ansonsten hätte man über -U und -P einen Anmeldenamen und sein Passwort angeben müssen).

An der weiter oben dargestellten Grundform der *bcp*-Syntax sind Ihnen vielleicht die Schlüsselwörter out und queryout aufgefallen. Durch Verwendung dieser Begriffe anstelle des üblichen in lässt sich eine Tabelle (out) oder Abfrage (queryout) aus dem SQL Server in eine Datei exportieren.

Probieren wir nun beide Varianten einmal aus:

1. Erstellen Sie mit Notepad oder einem beliebigen anderen Texteditor eine Datei mit Namen *CD-Export.bat,* um darin die folgenden *bcp*-Anweisungen abzulegen:

   ```
   bcp MediaBase.dbo.CD out c:\SQLServer\CDExport.txt -c -T -S .\SQL2008Express
   bcp "SELECT Titel, Interpret FROM MediaBase.dbo.CDTrack" queryout c:\SQLServer\CDTrackExport.txt -c -T -S
   .\SQL2008Express
   pause
   ```

   Die Anweisung pause dient dazu, dass das Kommandozeilenfenster nach Ausführung der *bcp*-Aufrufe geöffnet bleibt, bis Sie eine Taste drücken, damit Sie genügend Zeit haben, eventuell auftretende Fehlermeldungen zu lesen.

2. Führen Sie die Batchdatei nun im Windows-Explorer durch einen Doppelklick auf die Datei aus und Sie dürften Meldungen wie die in Abbildung 12.11 dargestellten erhalten.

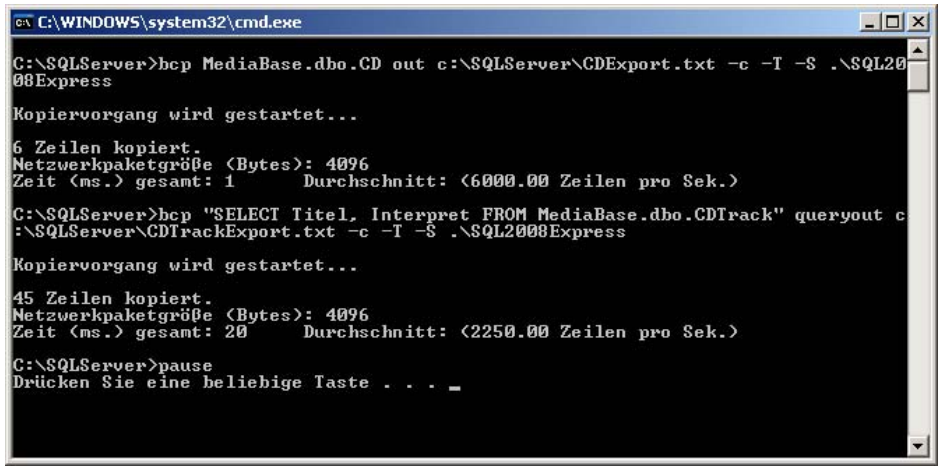


Abbildung 12.11: Ausgabe der Batchausführung

3. Drücken Sie – wie aufgefordert – eine beliebige Taste, damit sich das Kommandozeilenfenster wieder schließt. Im Ordner *C:\SQLServer* sollten nun die beiden Dateien *CDExport.txt* und *CDTrackExport.txt* zu sehen sein, die Sie durch einen Doppelklick öffnen können, um zu prüfen, dass die CD-Daten korrekt exportiert wurden. Dabei wird Ihnen auffallen, dass als Feldtrennzeichen ein Tabulatorzeichen verwendet wurde. Dafür zeichnet sich der Parameter `-c` verantwortlich, der – sofern nicht explizit andere Trennzeichen angegeben werden – als Feldtrennzeichen einen Tabulator und als Zeilentrennzeichen einen Zeilenumbruch vorsieht.

Formatdateien für BULK INSERT und bcp nutzen

Alternativ zu Feldtrennzeichen können für den Im- und Export von Daten mit `bcp` auch Formatdateien verwendet werden, in denen genau angegeben ist, welche Felder mit welchen Datentypen und Feldlängen zu verarbeiten sind. Als angenehmer Nebeneffekt lässt sich dadurch auch eine beliebige Reihenfolge der Felder vorgeben; Sie können sogar Felder weglassen, die nicht im- bzw. exportiert werden sollen.

Da die manuelle Erstellung einer solchen Formatdatei jedoch recht lästig sein kann, bietet *bcp* über das Schlüsselwort `format` (anstelle von `in`, `out` oder `queryout`) die Möglichkeit, eine Formatdatei aufgrund der angegebenen Tabelle zu erstellen.

So lässt sich mit der folgenden Anweisung eine Formatdatei für die Tabelle *dbo.CD* erzeugen:

```
bcp MediaBase.dbo.CD format nul -c -T -S .\SQL2008Express -f c:\SQLServer\CDExport.fmt
```

Die Formatdatei beinhaltet in der ersten Zeile die Versionsnummer des SQL Servers. Danach folgt die Anzahl der Spalten. In den weiteren Zeilen steht jeweils eine laufende Nummer, der Datentyp (für die Datei, nicht der aus der Tabelle), die Größe in Zeichen, das Trennzeichen und die Bezeichnung der Spalte.

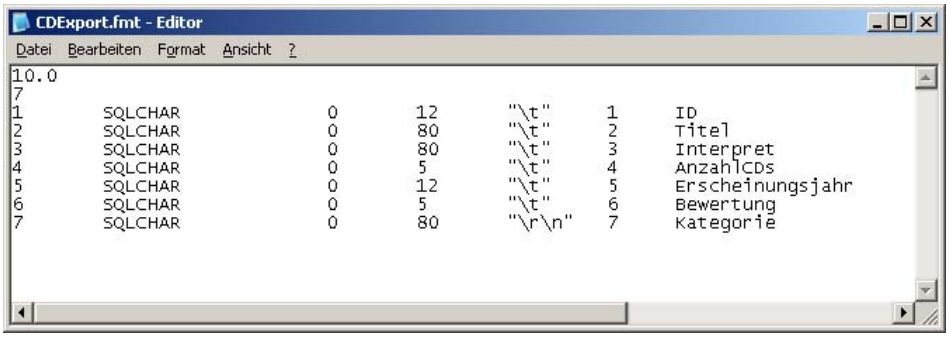


Abbildung 12.12: Formatdatei der Tabelle *dbo.CD*

Diese Formatdatei lässt sich dann später für einen Import mit *bcp* verwenden:

```
bcp MediaBase.dbo.CD in c:\SQLServer\CDImport.txt -T -S .\SQL2008Express -f c:\SQLServer\CDExport.fmt
```

12.5 Übungen zu diesem Kapitel

In diesem Abschnitt finden Sie einige Übungen zu diesem Kapitel. Die richtigen Antworten, Lösungen und den Programmcode finden Sie wie immer auf der Website *www.vsxpress.de.*

Übung 12.1

Erstellen Sie eine differenzielle Sicherung der *MediaBase*-Datenbank, die den vorhandenen Mediensatz nutzt und die bestehenden Sicherungssätze beibehält. Lassen Sie dazu auch die SQL-Anweisung generieren.

Übung 12.2

Exportieren Sie die Tabellen *dbo.CD* und *dbo.CDTrack* mithilfe des Import-/Export-Assistenten in eine Excel-Datei.

12.6 Zusammenfassung

In diesem Kapitel haben Sie verschiedene Varianten kennengelernt, eine Datenbank zu sichern. Angefangen vom Kopieren von Datenbankdateien, die vorher entweder offline geschaltet oder vom SQL Server getrennt wurden, bis hin zum Sichern über die Backup-Funktionalität von SQL Server. Egal welche Variante Sie verwenden, ist es vor allem wichtig, dass Sie Ihre Datenbanken regelmäßig sichern.

Dazu wurden noch die verschiedenen Varianten zum Im- und Exportieren von Daten vorgestellt:

- Der Import-/Export-Assistent
- Die `BULK INSERT`-Anweisung
- Das Kommandozeilenprogramm *bcp*

Während sich der Import-/Export-Assistent eher für einmalige Aktionen eignet, lassen sich sowohl mit `BULK INSERT` als auch über `bcp` entsprechende SQL-Skripts bzw. Batchdateien erstellen, die man später immer wieder ausführen kann.

Damit ist der Teil zum Thema Administration abgeschlossen. Im folgenden und letzten Teil wird gezielt auf verschiedene Spezialthemen und erweiterte Funktionen von Microsoft SQL Server eingegangen.

Internet-Link: Weitere Infos zur Administration von SQL Server 2008

Wenn Sie tiefer in das Thema einsteigen möchten, empfehle ich Ihnen das Buch *Microsoft SQL Server 2008 – Taschenratgeber für Administratoren* von *William R. Stanek*. Genauere Informationen zu diesem Buch erhalten Sie über den Softlink **sql1201**.

Als Probekapitel steht Ihnen Kapitel 7, *Grundlegende Aufgaben der Datenbankverwaltung* über den Softlink **sql1202** zur Verfügung.

Um die Softlink zu nutzen, geben Sie auf der Startseite von *http://www.vsxpress.de* die angegebene Link-ID ein und klicken dann auf *Link öffnen*. Alternativ können Sie auch direkt in der Adressleiste Ihres Browsers die folgenden URLs eingeben:
http://go.vsxpress.de/?linkid=sql1201 bzw.
http://go.vsxpress.de/?linkid=sql1202

Kapitel 13

SQL Server und .NET Framework

In diesem Kapitel lernen Sie

- wie Sie von .NET auf SQL Server zugreifen können
- verschiedene Arten des Datenzugriffs von ADO.NET über LINQ to SQL bis hin zum Entity Framework
- welche Möglichkeiten es gibt, .NET über die CLR-Integration im SQL Server zu nutzen

Es gibt verschiedene Möglichkeiten, den SQL Server im Zusammenhang mit .NET zu nutzen. Diese möchte ich Ihnen hier kurz vorstellen. Da man mit jedem der hier behandelten Themen leicht ein ganzes Buch füllen könnte, ist es an dieser Stelle allerdings nur möglich, die Themen anzureißen oder anhand eines einfachen Beispiels kurz zu demonstrieren. Aus demselben Grund kann hier natürlich auch nicht auf jedes Detail der dargestellten Beispiele eingegangen werden.

Internet-Link: Weitere Infos zur .NET-Programmierung

Wenn Sie tiefer in das Thema einsteigen möchten, kann ich Ihnen die Bücher *Programmieren lernen mit Visual Basic 2008* sowie *Programmieren lernen mit Visual C# 2008* von *Klaus Fahnenstich* und *Rainer G. Haselier* empfehlen. Genauere Informationen zu diesen Büchern erhalten Sie über die Softlinks **sql1301** (Visual Basic) und **sql1302** (Visual C#).

Um die Softlinks zu nutzen, geben Sie auf der Startseite von http://www.vsxpress.de die angegebene Link-ID ein und klicken dann auf *Link öffnen*. Alternativ können Sie auch direkt in der Adressleiste Ihres Browsers die folgenden URLs eingeben:
http://go.vsxpress.de/?linkid=sql1301 bzw. **http://go.vsxpress.de/?linkid=sql1302**

Wichtig: Visual Studio erforderlich!

Da es in diesem Kapitel um die Programmierung mit .NET geht, ist für alle diskutierten Technologien und Beispiele eine Vollversion von Visual Studio nötig, um diese ausprobieren zu können. Das mit SQL Server ausgelieferte Business Intelligence Development Studio liefert zwar die Entwicklungsoberfläche, stellt aber nicht die für die .NET-Entwicklung benötigten Projekttypen zur Verfügung.

13.1 Schichtentrennung und Applikationsaufbau

Während es bei kleineren Anwendungen, die komplett von einem Entwickler implementiert werden, eigentlich nur eine untergeordnete Rolle spielt, hat sich – insbesondere bei großen Softwareentwicklungsprojekten – eine mehrschichtige Architektur bewährt. Die Idee dabei ist, die Benutzeroberfläche der Anwendung von der Datenhaltung zu trennen.

Wenn man diesen Ansatz weiter verfolgt, lässt sich noch eine dritte Schicht herauslösen: die Geschäftslogik. Dies zusammen ergibt dann eine Architektur mit drei Schichten (3-Tier):

- Die *Datenbankschicht* (Database Layer) stellt Objekte und Methoden bereit, um auf die Datenbank zuzugreifen. Sofern die Datenbankschicht wirklich sauber getrennt ist, kann man diese mit relativ wenig Aufwand austauschen. So kann man im Idealfall von einer anderen Datenbank auf eine SQL Server-Datenbank umsteigen, ohne dabei die Geschäftslogik oder die Präsentationsschicht anpassen zu müssen.
- Die *Geschäftslogik* (Business Layer) steuert die Prozesse und implementiert die fachlichen Anforderungen. Dazu nutzt sie die Datenbankschicht, um auf die Datenbank zuzugreifen, und die Präsentationsschicht, um die Ergebnisse darzustellen (oder eine Benutzereingabe einzufordern).
- Die *Präsentationsschicht* (Presentation Layer) stellt die von der Geschäftslogik zur Verfügung gestellten Daten dar. Auch die Präsentationsschicht lässt sich – bei konsequenter Schichtentrennung – gegen eine andere austauschen, sodass man auf diesem Wege mit vertretbarem Aufwand aus einer Desktopanwendung eine Webanwendung oder eine Windows Mobile-Anwendung machen kann.

Ein weiterer großer Vorteil der Schichtentrennung liegt in einer deutlich besseren Wartbarkeit der Anwendung. So können einzelne Schichten von verschiedenen Entwicklerteams programmiert und getestet werden, ohne die anderen Schichten zu beeinflussen. Das klappt allerdings nur durch eine klar definierte Abgrenzung der Schichten mit genau spezifizierten Schnittstellen.

Zum Erstellen einer Datenbankschicht gibt es mittlerweile auch zahlreiche Tools (sowohl von Microsoft als auch von Fremdanbietern), mit denen aufgrund einer bestehenden Datenbank eine komplette Zugriffsschicht mit den wichtigsten Methoden zum Lesen, Einfügen, Ändern und Löschen von Daten generiert werden kann. Auf diesem Weg wird gleich noch ein weiteres Problem gelöst: der Zugriff aus einer objektorientierten Anwendung auf relationale Daten. Daher fasst man die entsprechenden Tools auch unter dem Begriff O/R-Mapper zusammen. Die meisten etablierten O/R-Mapper erstellen für jede Tabelle eine Objektklasse mit entsprechenden Zugriffsmethoden. Die meisten generieren Methoden, um bestehende gespeicherte Prozeduren aufzurufen, einige generieren auch gespeicherte Prozeduren, um die Datenbankzugriffe weiter zu kapseln.

13.2 Zugriff über ADO.NET

ADO.NET – die Weiterentwicklung der ActiveX Data Objects (kurz: ADO) – bildet die Standard-Datenbankschnittstelle des .NET Framework. Dazu werden verschiedene Klassen zur Verfügung gestellt, die es ermöglichen, auf verschiedene Datenbanktypen zuzugreifen (.NET-Datenanbieter) sowie Datenbanken – oder Teile davon – im Speicher zu halten (DataSet), um damit zu arbeiten.

Rund um das DataSet gibt es unter anderem folgende Klassen (die Bestandteile des Namespace `System.Data` sind):

- `DataSet` – kann eine oder mehrere Tabellen und deren Beziehungen zueinander enthalten, quasi eine kleine Datenbank im Hauptspeicher
- `DataTable` – eine Tabelle, die Bestandteil eines DataSets sein kann
- `DataRow` – eine Datenzeile einer Tabelle

Mit diesen Klassen lässt sich komplett im Hauptspeicher arbeiten, ohne irgendwelche Daten in einer echten Datenbank ablegen zu müssen. Für die Verbindung zu bzw. Nutzung von Datenbanken sind verschiedene .NET-Datenanbieter verfügbar, von denen für uns besonders der Namensraum `System.Data.SqlClient` interessant ist, da dieser die Klassen bereitstellt, um auf einen SQL Server zuzugreifen. Die .NET-Datenanbieter stellen unter anderem folgende Klassen zur Verfügung:

- `Connection` – stellt eine Verbindung zu einer Datenbank her
- `Command` – steht für eine SQL-Anweisung, die auf verschiedene Arten verwendet werden kann: einfach ausgeführt (mit oder ohne Rückgabewert), als Datenquelle für einen DataReader etc.
- `DataAdapter` – Schnittstelle zwischen Datenquelle und DataSet, füllt Letzteres und gleicht dessen Daten mit der Datenquelle ab
- `DataReader` – basiert auf einer Abfrage und kann satzweise durchlaufen werden, um einzelne Zeilen einer Abfrage zu lesen

Da die .NET-Datenanbieter datenbankspezifisch sind, haben die entsprechenden Klassen – je nach verwendeter Datenbank – ein spezielles Präfix. Für den SQL Server ist dies erwartungsgemäß `Sql`, sodass die gerade beschriebenen Klassen hier `SqlConnection`, `SqlCommand`, `SqlDataAdapter` und `SqlDataReader` heißen.

Schauen wir uns nun einfach ein kleines Codebeispiel an, in dem – ohne ein DataSet zu nutzen – direkt mit dem .NET-Datenanbieter für SQL Server gearbeitet wird:

```
// Standardverweise
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

// Verweise für ADO.NET-Datenanbieter
using System.Data.SqlClient;

namespace Datenzugriff
{
  public partial class Form1 : Form
  {
     public Form1()
     {
       InitializeComponent();
     }
```

```
    private void btnADOdotNET_Click(object sender, EventArgs e)
    {
        SqlConnection conSQL = new SqlConnection("Database=MediaBase;" +
             Server=.\\SQL2008EXPRESS;Integrated Security=SSPI");

        conSQL.Open();

        SqlCommand cmdSQL1 = new SqlCommand("INSERT INTO Buch (Titel, Autor) VALUES " +
             "('Test', 'Robert Panther')", conSQL);
        cmdSQL1.ExecuteNonQuery();

        SqlCommand cmdSQL2 = new SqlCommand("SELECT ID, Titel FROM Buch WHERE Autor='Robert Panther'", conSQL);
        SqlDataReader drSQL = cmdSQL2.ExecuteReader();
        while (drSQL.Read())
        {
            txtErgebnis.Text = txtErgebnis.Text + drSQL[0].ToString() + "/" + drSQL[1] + "\r\n";
        }
        drSQL.Close();

        SqlCommand cmdSQL3 = new SqlCommand("UPDATE Buch SET Titel='Test (update)' WHERE Titel='Test'", conSQL);
cmdS        QL3.ExecuteNonQuery();

        SqlCommand cmdSQL4 = new SqlCommand("DELETE FROM Buch WHERE Titel='Test (update)'", conSQL);
        cmdSQL4.ExecuteNonQuery();

        conSQL.Close();
    }
}
```

Zu diesem Quelltext gehört eine Windows Forms-Anwendung, die eine Schaltfläche (`btnADOdotNET`) und ein mehrzeiliges Textfeld (`txtErgebnis`) enthält. Durch einen Klick auf die Schaltfläche wird die Ereignisprozedur `btnADOdotNET`-Click aufgerufen, in der die eigentliche Logik stattfindet:

Nachdem über ein Objekt des Typs `SqlConnection` eine Verbindung zur Datenbank `MediaBase` aufgebaut und geöffnet wurde, wird ein Objekt vom Typ `SqlCommand` erstellt, das eine `INSERT`-Anweisung enthält. Diese wird darauf über die Methode `ExecuteNonQuery` (führt eine `Sql`-Anweisung aus, die kein Ergebnis zurückliefert) ausgeführt und fügt dabei einen Datensatz in die Tabelle *Buch* ein.

Anschließend wird ein weiteres `SqlCommand`-Objekt (`cmdSQL2`) erzeugt, das eine `SELECT`-Abfrage beinhaltet. Dieses wird als Quelle für ein `SqlDataReader`-Objekt genutzt, das so lange durchlaufen wird, bis alle Zeilen gelesen sind. Innerhalb der Schleife wird direkt auf die Spalten 0 und 1 der Abfrage zugegriffen, deren Werte in das Textfeld (`txtErgebnis`) geschrieben werden. Anschließend wird der `SqlDataReader` geschlossen.

Auf dieselbe Art, wie weiter oben eine Zeile in die Tabelle *Buch* eingefügt wurde, wird dieselbe Zeile nun über separate `SqlCommand`-Objekte geändert und anschließend gelöscht, bevor am Ende die Verbindung `conSQL` wieder geschlossen wird.

13.3 LINQ to SQL

LINQ ist – wie der Name Language Integrated Query schon vermuten lässt – eine in .NET integrierte Abfragesprache. Diese Abfragesprache gibt es in verschiedenen Implementierungen, die auf unterschiedliche Datenquellen zugreifen:

- LINQ to Objects – zum Zugriff auf beliebige Objekt-Collections
- LINQ to ADO.NET – zum Zugriff auf ADO.NET-Datenquellen
  - LINQ to DataSet – zum Zugriff auf .NET-DataSets
  - LINQ to Entities – zum Zugriff auf das Entity Framework
  - LINQ to SQL – zum Zugriff auf SQL-Datenbanken
- LINQ to XML – zum Zugriff auf XML-Strukturen

Auch wenn man mit den meisten Varianten mehr oder weniger direkt auf SQL Server-Datenbanken zugreifen kann, möchte ich hier kurz mit LINQ to SQL die direkteste Möglichkeit vorstellen.

Hintergrundinfo: Totgesagte leben länger

Seit längerer Zeit kursieren in Entwicklerkreisen Gerüchte, dass LINQ to SQL zugunsten des weiter unten vorgestellten Entity Framework künftig von Microsoft nicht mehr unterstützt wird. Inzwischen hat sich herausgestellt, dass dies nicht (mehr) ganz zutrifft.

Microsoft wird LINQ to SQL zwar nicht mehr maßgeblich weiterentwickeln, aber – zumindest bis auf Weiteres – auch in kommenden .NET-Versionen noch weiter unterstützen. Es schadet also nicht, sich mit LINQ to SQL zu befassen, notfalls kann man später immer noch auf LINQ to Entities umsteigen, ohne dabei die grundlegenden Konzepte von LINQ über Bord werfen zu müssen.

Um LINQ nutzen zu können, benötigen Sie Visual Studio 2008 und das .NET Framework in der Version 3.5 (oder neuer). Wo aber liegt der wesentliche Vorteil von LINQ gegenüber ADO.NET?

Die Antwort ist relativ schnell gegeben: Während man mit ADO.NET SQL-Anweisungen in Form von Zeichenketten verwendet und dadurch viele Fehler erst zur Laufzeit auftreten, stellt LINQ Objekte zur Verfügung, deren Typ bereits während der Entwicklung geprüft werden kann. Als angenehmer Nebeneffekt kann dadurch sogar IntelliSense bei der Eingabe von Abfragen genutzt werden, was Tipparbeit spart und ebenfalls Fehler vermeidet.

LINQ to SQL-Klassen per Quelltext erstellen

LINQ to SQL nutzt ein einfaches O/R-Mapping, in dem für jede Tabelle eine entsprechende Objektklasse erzeugt wird. Schauen wir uns auch dies wieder anhand des bereits bei ADO.NET demonstrierten Beispiels an:

```
// Standardverweise
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
```

```
using System.Linq;
using System.Text;
using System.Windows.Forms;
// Verweise für LINQ to SQL
using System.Data.Linq;
using System.Data.Linq.Mapping;

namespace Datenzugriff
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void btnLINQtoSQL_Click(object sender, EventArgs e)
        {
            // Datenkontext über Connection-String aufbauen
            DataContext db = new DataContext ("Database=MediaBase;" +
                  "Server=.\\SQL2008EXPRESS;Integrated Security=SSPI");

            // typisierte Tabelle erstellen
            Table<Buch> Buecher = db.GetTable<Buch>();

            db.Log = Console.Out;

            // eigentliche LINQ-Abfrage
            IQueryable<Buch> buchQuery =
                from buch in Buecher
                where buch.Autor == "Robert Panther"
                select buch;

            foreach (Buch buch in buchQuery)
            {
                txtErgebnis.Text = txtErgebnis.Text + (buch.ID).ToString() + "/" + buch.Titel + "\r\n";
            }
        }
    }

    [Table(Name = "Buch")]
    public class Buch
    {
        private int _ID;
        [Column(IsPrimaryKey=true, Storage="_ID")]
        public int ID
        {
            get
            {
                return this._ID;
            }
            set
            {
                this._ID = value;
            }
        }
```

```
        private string _Titel;
        [Column(Storage="_Titel")]
        public string Titel
        {
            get
            {
                return this._Titel;
            }
            set
            {
                this._Titel=value;
            }
        }

        private string _Autor;
        [Column(Storage = "_Autor")]
        public string Autor
        {
            get
            {
                return this._Autor;
            }
            set
            {
                this._Autor = value;
            }
        }

        private int _Erscheinungsjahr;
        [Column(Storage = "_Erscheinungsjahr")]
        public int Erscheinungsjahr
        {
            get
            {
                return this._Erscheinungsjahr;
            }
            set
            {
                this._Erscheinungsjahr = value;
            }
        }
    }

}
```

Im unteren Bereich des Quelltextes wird die Klasse `Buch` definiert, in der Eigenschaften für die Spalten `ID`, `Titel`, `Autor` und `Erscheinungsjahr` spezifiziert werden. In der Click-Ereignisprozedur der Schaltfläche `btnLINQtoSQL` wird zuerst ein Datenkontext erstellt, der quasi die Verbindung zur Datenbank `MediaBase` repräsentiert. Anschließend wird über die Methode `GetTable` die Tabelle `Buch` gelesen und an das typisierte Tabellenobjekt `Buecher` gebunden. Danach findet die eigentliche LINQ-Abfrage statt. Die Syntax ist ähnlich wie beim SQL `SELECT`, allerdings in anderer Reihenfolge, da es für IntelliSense mehr Sinn macht, erst die Tabellennamen und dann die entsprechenden Spalten anzugeben. Somit können die in den Tabellen enthaltenen Spalten bereits bei deren Eingabe erkannt bzw. vorgeschlagen werden.

Schließlich wird die über die LINQ-Abfrage definierte Query mit der `foreach`-Anweisung durchlaufen und `ID` und `Titel` jedes Buches in das Textfeld (`txtErgebnis`) geschrieben.

LINQ to SQL-Klassen mit dem Server-Explorer erstellen

Die Klassen für die verschiedenen Tabellen müssen nicht – wie vielleicht aufgrund des Beispiels oben zu befürchten ist – manuell angelegt werden. Hier bietet Visual Studio entsprechende Unterstützung, um den Code aufgrund von bestehenden Tabellen automatisch generieren zu lassen:

1. Klicken Sie dazu im Projektmappen-Explorer mit der rechten Maustaste auf den Projektnamen und wählen Sie anschließend die Option *Hinzufügen/Neues Element*.
2. Im anschließend erscheinenden Dialogfeld wählen Sie die Vorlage *LINQ to SQL-Klassen* und passen bei Bedarf den Namen der zu generierenden *.dbml*-Datei an, bevor Sie auf *Hinzufügen* klicken.
3. Es erscheint der Entwurfsmodus für LINQ to SQL-Klassen, in dem Sie Tabellen aus dem *Server-Explorer* mit der Maus einfach in den Detailbereich ziehen können, um diese dem Modell hinzuzufügen. Mit demselben Vorgehen können Sie auch gespeicherte Prozeduren und Funktionen auf die rechte Seite des Detailbereichs ziehen, um hierfür später Methoden generieren zu lassen.
4. Sobald Sie das Modell speichern und das Projekt kompilieren, werden die entsprechenden Klassen im Hintergrund generiert und stehen sofort zur Benutzung im Quelltext zur Verfügung.

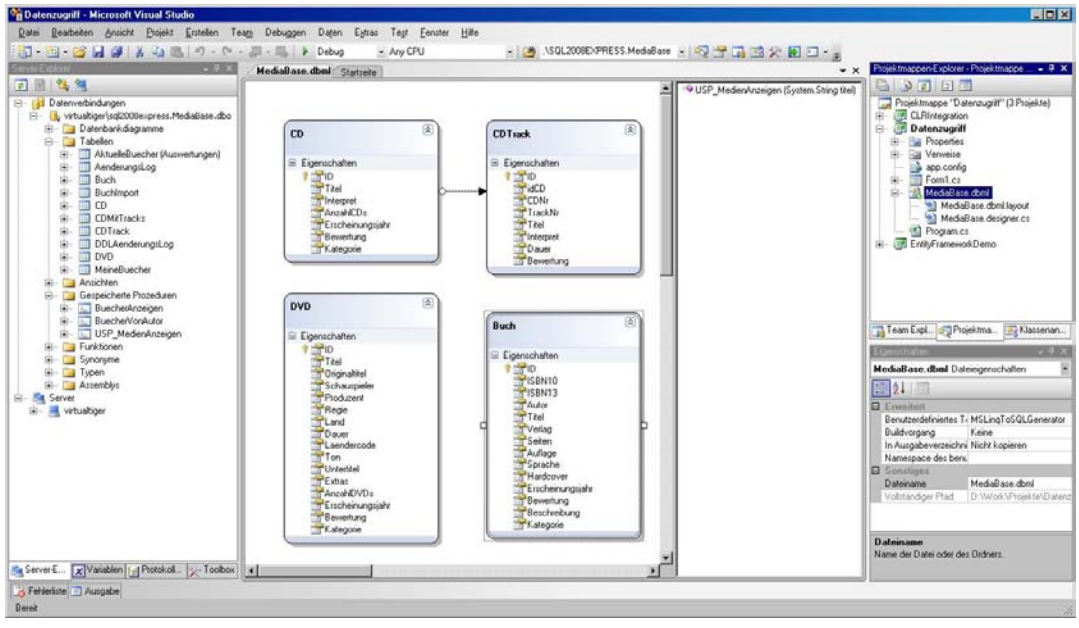


Abbildung 13.1: Der Entwurfsmodus für LINQ to SQL-Klassen

13.4 Das ADO.NET Entity Framework

Am konsequentesten wird die Schichtentrennung mit Nutzung des Entity Framework verfolgt, da hier die größte Unabhängigkeit zwischen Daten- und Objektmodell besteht. Während bei LINQ to SQL für jede Tabelle genau eine Klasse generiert wird, können Sie beim Entity Framework eine Tabelle auf mehrere Klassen verteilen oder aber mehrere Tabellen in einer Klasse zusammenfassen. Um dies zu realisieren, werden innerhalb der Datenzugriffsschicht drei fest definierte Ebenen verwendet:

- Das logische Modell (Storage Model) repräsentiert die Daten, so wie sie auf dem SQL Server gespeichert sind. Dafür wird die Datenspeicherschema-Definitionssprache (Storage Schema Definition Language, SSDL) verwendet.
- Das konzeptionelle Modell (Conceptual Model) nutzt die konzeptionelle Schemadefinitionssprache (Conceptual Schema Definition Language, CSDL), um die Objektstruktur zu beschreiben, auf die das logische Modell abgebildet wird.
- Zwischen logischem und konzeptionellem Modell liegt die Zuordnungsschicht (C-S Mapping), in der die Zuordnung von Tabellen auf Objektklassen und Tabellenspalten auf Objekteigenschaften stattfindet. Dies geschieht mithilfe der Mapping-Spezifikationssprache (Mapping Specification Language, MSL).

Alle drei Schichten werden intern in einer *.edmx*-Datei (im XML-Format) gespeichert. Der Zugriff auf die Datenschicht erfolgt dann entweder in Entity SQL (einer speziellen SQL-Variante zum Zugriff auf Entitäten) oder über die entsprechende LINQ-Variante LINQ to Entities.

Voraussetzung für die Verwendung des ADO.NET Entity Framework ist Visual Studio 2008 und das .NET Framework 3.5 SP1 (oder neuer).

Schauen wir uns nun einmal das generelle Vorgehen zum Erstellen eines Datenzugriffs mithilfe des Entity Framework an:

1. Erstellen Sie in Visual Studio ein neues Projekt vom Typ *Windows Forms-Anwendung*.
2. Um diesem Projekt ein Entity Data Model hinzuzufügen, klicken Sie im Projektmappen-Explorer mit der rechten Maustaste auf den Projektnamen und wählen anschließend die Option *Hinzufügen/Neues Element*.

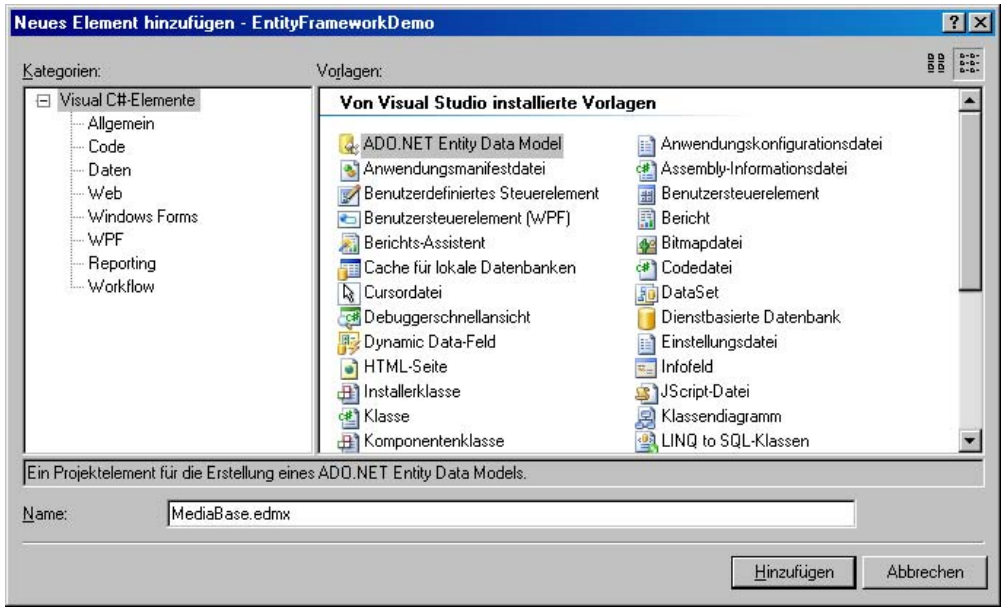


Abbildung 13.2: Hinzufügen eines ADO.NET Entity Data Models

3. Wählen Sie nun die Vorlage *ADO.NET Entity Data Model* und passen Sie bei Bedarf den Namen der zu generierenden *.edmx*-Datei an, bevor Sie auf *Hinzufügen* klicken.
4. Im nächsten Schritt können Sie wählen, ob das Modell aus einer Datenbank generiert oder ein leeres Modell erstellt werden soll. Bestätigen Sie die erste Variante durch einen Klick auf die Schaltfläche *Weiter*.
5. Anschließend ist die Datenverbindung auszuwählen bzw. über die Schaltfläche *Neue Verbindung* neu zu definieren. Geben Sie hier die entsprechende Serverinstanz (z.B. *.\SQL2008EXPRESS)* und Datenbank (*MediaBase*) an. Im unteren Bereich des Dialogfelds können Sie über ein Kontrollkästchen dafür sorgen, dass die Verbindungseinstellungen in der Datei *App.Config* gespeichert werden, wodurch sie später auch für weitere Modelle innerhalb desselben Projekts wieder verwendet werden können.
6. Als Nächstes sind die Datenbankobjekte (Tabellen, Sichten und/oder gespeicherte Prozeduren) auszuwählen, die im Modell berücksichtigt werden sollen. Dazu können Sie auch den Namen des Namespace für das Modell festlegen.

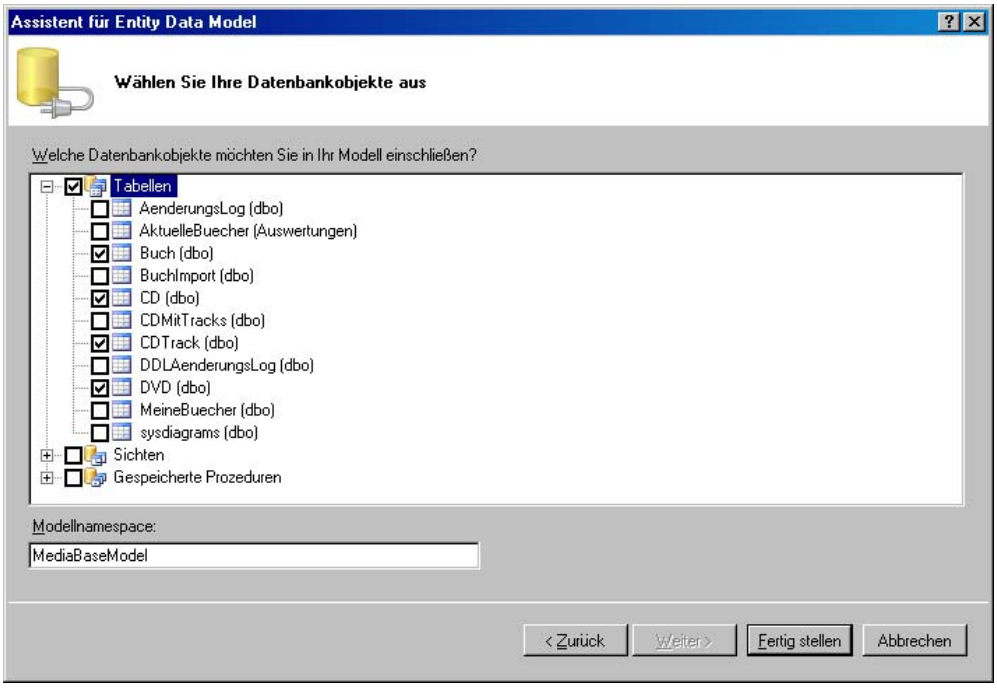


Abbildung 13.3: Die Auswahl der in das Modell zu integrierenden Datenbankobjekte

7. Wenn Sie auf *Fertig stellen* klicken, wird das Entity Data Model generiert und in einer .edmx-Datei abgelegt. Dazu wird die grafische Darstellung des Modells im Detailbereich von Visual Studio angezeigt.

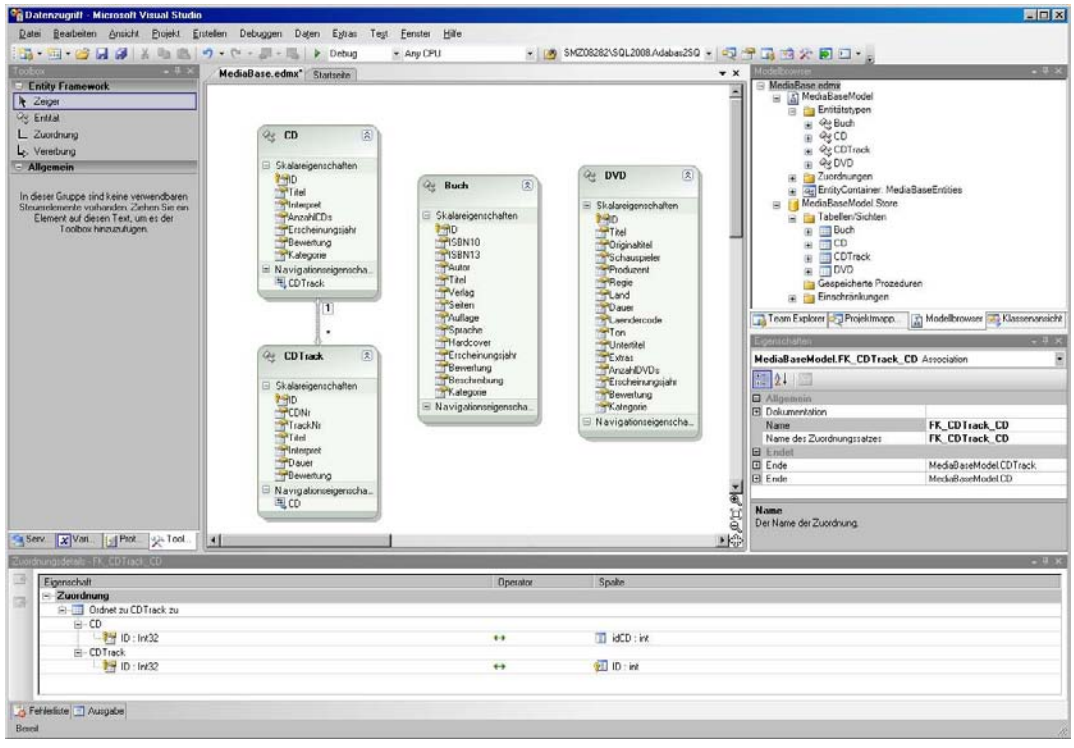


Abbildung 13.4: Das Entity Data Model in der grafischen Darstellung

Hintergrundinfo: Aufbau der *.edmx*-Datei

Wenn es Sie interessiert, wie die *.edmx*-Datei aufgebaut ist, können Sie diese in einem beliebigen Texteditor öffnen. Sie werden feststellen, dass dies reines XML ist.

Übersichtlicher lässt sich dies im Internet Explorer betrachten, indem Sie eine Kopie der Datei erstellen, dieser die Endung xml geben und diese dann mit der Maus in ein zuvor geöffnetes Internet Explorer-Fenster ziehen. Hier sind schnell die folgenden Bereiche zu erkennen, die genau den weiter oben beschriebenen Ebenen der Datenzugriffsschicht entsprechen:

- `<edmx:StorageModels>` – logisches Modell
- `<edmx:ConceptualModels>` – konzeptionelles Modell
- `<edmx:Mappings>` – Zuordnung zwischen logischem und konzeptionellem Modell

8. Nachdem Sie nun das Entity Data Model erstellt haben, fehlt noch die eigentliche Anwendung. Öffnen Sie dazu das Formular *Form1* im Entwurfsmodus und fügen Sie diesem eine *ComboBox* (cboCD) und ein *DataGridView* (grdCDTrack) hinzu.
9. Doppelklicken Sie auf einen leeren Bereich des Formulars, wodurch Sie in die Ereignisprozedur `Form1_Load` gelangen, die Sie mit folgendem Code füllen:

```
private void Form1_Load(object sender, EventArgs e)
{
   mediaBaseContext = new MediaBaseEntities();

   ObjectQuery<CD> cdQuery = mediaBaseContext.CD.Include("CDTrack");

   this.cboCD.DataSource = cdQuery;
   this.cboCD.DisplayMember = "Titel";
}
```

Hierdurch wird ein Kontextobjekt (`mediaBaseContext`) erzeugt, über das Sie auf das Modell zugreifen können. Anschließend wird eine Abfrage (`cdQuery`) erstellt, welche die Objekte des Typs `CD` sowie deren `CDTrack`-Objekte enthält. Diese Abfrage (bzw. deren `Titel`-Spalte) wird als Datenquelle für die `ComboBox` festgelegt.

10. In der Klasse *Form1* muss nun noch das Kontextobjekt deklariert werden:

```
private MediaBaseEntities mediaBaseContext;
```

11. Wechseln Sie nun wieder zur Entwurfsansicht des Formulars und doppelklicken Sie auf die ComboBox, wodurch auch hierfür eine Ereignisprozedur generiert wird, die Sie wie folgt füllen:

```
private void cboCD_SelectedIndexChanged(object sender, EventArgs e)
{
   CD cd = (CD)this.cboCD.SelectedItem;
   grdCDTrack.DataSource = cd.CDTrack;
}
```

Dadurch wird ein CD-Objekt definiert und mit der ausgewählten CD gefüllt. Deren Tracks werden anschließend als Datenquelle für die DataGridView verwendet.

12. Der gesamte Quelltext sollte nun wie folgt aussehen:

```
// Standardverweise
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

// Verweise für ADO.NET Entity Framework
using System.Data.Objects;
using System.Data.Objects.DataClasses;

namespace EntityFrameworkDemo
{
    public partial class Form1 : Form
    {
        private MediaBaseEntities mediaBaseContext;

        public Form1()
        {
            InitializeComponent();
        }
```

```
        private void Form1_Load(object sender, EventArgs e)
        {
            mediaBaseContext = new MediaBaseEntities();

            ObjectQuery<CD> cdQuery = mediaBaseContext.CD.Include("CDTrack");

            this.cboCD.DataSource = cdQuery;
            this.cboCD.DisplayMember = "Titel";
        }

        private void cboCD_SelectedIndexChanged(object sender, EventArgs e)
        {
            CD cd = (CD)this.cboCD.SelectedItem;
            grdCDTrack.DataSource = cd.CDTrack;
        }
    }
}
```

13. Starten Sie nun die Anwendung mit der F5-Taste. Sie sollten dann ein Formular sehen, in dem die in der Datenbank gespeicherten CD-Titel über die ComboBox auswählbar sind. Die Titel der gewählten CD sind in der DataGridView darunter zu sehen.

Abbildung 13.5: Das Beispiel zur Laufzeit

Auch das Entity Framework bietet natürlich noch viele weitere Möglichkeiten, die hier aus Platzgründen nicht genauer beleuchtet werden können. Stattdessen möchte ich nun – nachdem jetzt drei verschiedene Varianten des Datenzugriffs aus .NET heraus behandelt wurden – den Spieß umdrehen und darauf eingehen, wie Sie mit .NET erstellte Routinen in den SQL Server einbinden können.

13.5 Die CLR-Integration von SQL Server

Eines der großen Features, die mit SQL Server 2005 hinzugekommen sind und in der 2008erVersion noch ausgebaut wurden, ist die CLR-Integration. Bei dem Begriff CLR (Common Language Runtime) handelt es sich um die Laufzeitumgebung für .NET-Anwendungen. Bereits mit SQL Server 2000 war es möglich, mit Visual Studio eigene Routinen zu programmieren, die dann in den SQL Server eingebunden wurden. Damals ging dies allerdings nur mit C++-DLLs. Nachdem die die CLR in den SQL Server integriert wurde, ist es erstmals möglich, in einer beliebigen .NET-Sprache (wie beispielsweise VB.NET oder C#) selbst entwickelte Funktionalitäten in den SQL Server zu integrieren. Das betrifft neben gespeicherten Prozeduren und Funktionen auch Trigger sowie benutzerdefinierte Typen und benutzerdefinierte Aggregate. Für Letztere ist es sogar der einzige Weg, diese zu erzeugen.

Hintergrundinfo: Systemeigene CLR-Funktionen

Viele der in SQL Server 2008 neu hinzugekomme nen Datentypen sind intern ebenfalls als .NET-Datentypen realisiert. So sind sowohl die Geo-Datentypen *geography* und *geometry* als auch die Datentypen *Hierarchyld* und *XML* auf diesem Weg entstanden.

Die grobe Vorgehensweise hierzu ist wie folgt:

- Erstellen einer .NET-Assembly mit der passenden Vorlage
- Registrieren der Assembly im SQL Server

Zuvor muss allerdings die CLR-Integration für die verwendete SQL Server-Instanz aktiviert werden, da diese per Default inaktiv ist. Dabei handelt es sich um eine globale Konfigurationsoption, die Sie mit den folgenden Anweisungen setzen können:

```
sp_configure 'clr enabled' , 1;
GO
RECONFIGURE;
GO
```

Die Anweisung `RECONFIGURE` ist notwendig, damit die vorgenommene Einstellung auch aktiviert wird.

Wenn Sie Visual Studio installiert haben, können Sie das Erstellen und Einbinden einer eigenen Funktion einmal ausprobieren:

1. Legen Sie ein neues Projekt mit Namen **CLRIntegration** an und verwenden Sie dazu die Projektvorlage *Visual C#/Datenbank/SQL Server-Projekt* (oder alternativ: *Visual Basic/Datenbank/SQL Server-Projekt*).

Wichtig: Verwechslungsgefahr!

Verwechseln Sie die Vorlage n icht mit der Vorlage *Datenbankprojekt*, die in *Andere Projekttypen/Datenbank* zu finden ist. Diese i st zum Erstel len von Datenbankprojekten gedac ht, die primär SQL-Skripte enthalten.

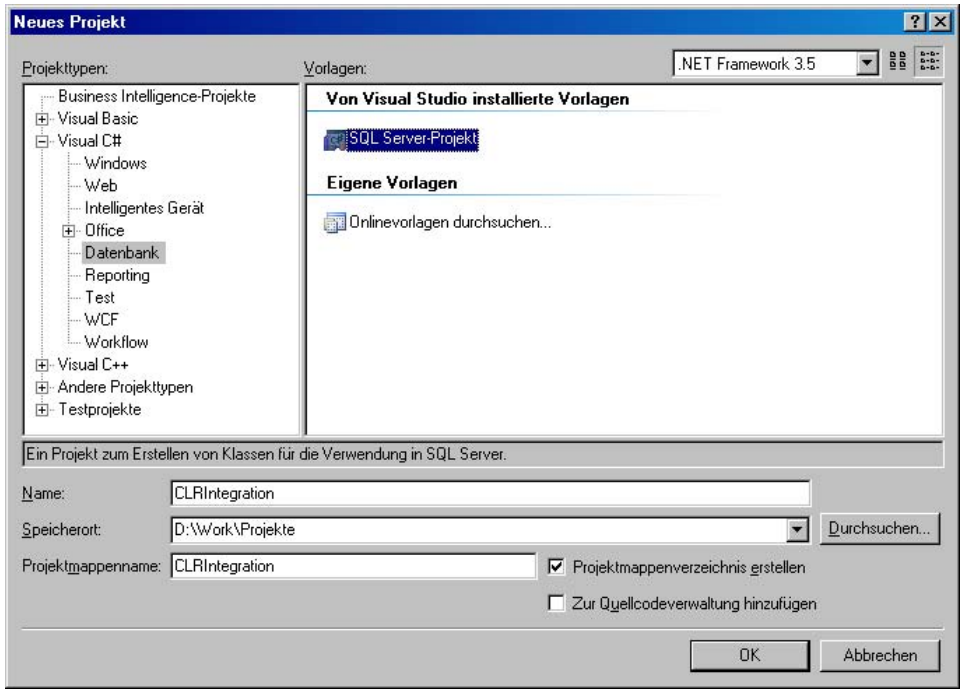


Abbildung 13.6: Die Projektvorlage für die CLR-Integration von SQL Server

2. Im nächsten Schritt werden Sie aufgefordert, einen Datenbankverweis auszuwählen oder – sofern noch nicht vorhanden – über die Schaltfläche *Neuen Verweis hinzufügen* zu erstellen. Erstellen Sie hier einen Verweis auf die Datenbank *MediaBase* Ihrer lokalen Serverinstanz .\SQL2008EXPRESS.
3. Nachdem das Projektgerüst erzeugt ist, klicken Sie mit der rechten Maustaste im Projektmappen-Explorer auf den Projektnamen und wählen *Hinzufügen/Benutzerdefinierte Funktion*.
4. Es erscheint ein Dialogfeld, in dem Sie die Vorlage *Benutzerdefinierte Funktion* bestätigen und einen Namen festlegen können. Geben Sie hier **UFN_CLRTest.cs** ein. (Die Endung .cs resultiert aus der Programmiersprache C#, der Name der Funktion wird dem gerade eingegebenen Namen ohne diese Endung entsprechen.)

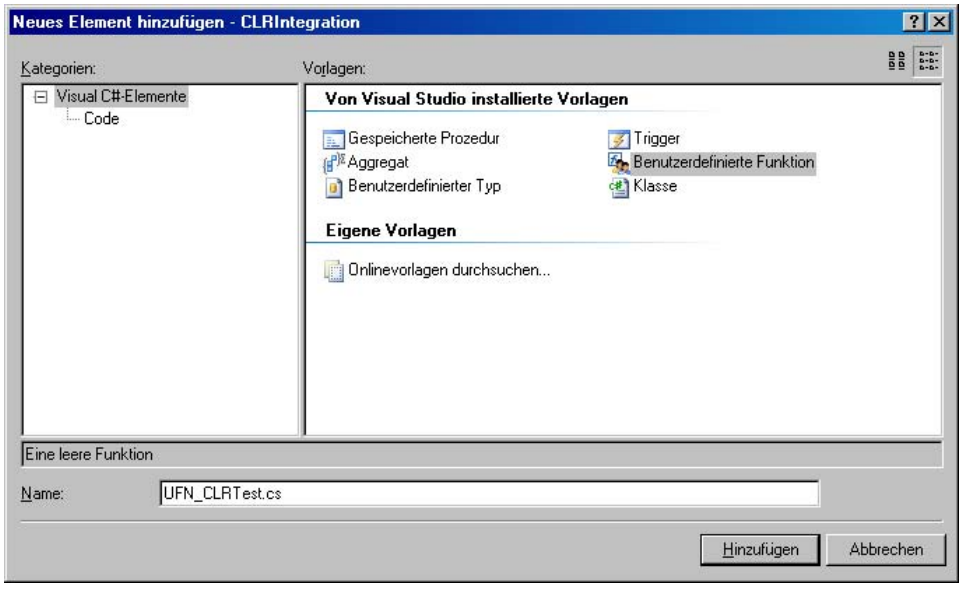


Abbildung 13.7: Die verfügbaren Klassenvorlagen

5. Anschließend wird automatisch ein Gerüst für die benutzerdefinierte Funktion erstellt, das wir zu Testzwecken einfach beibehalten (wenn Sie möchten, können Sie natürlich den Text, der von der Funktion zurückgegeben wird, ändern).

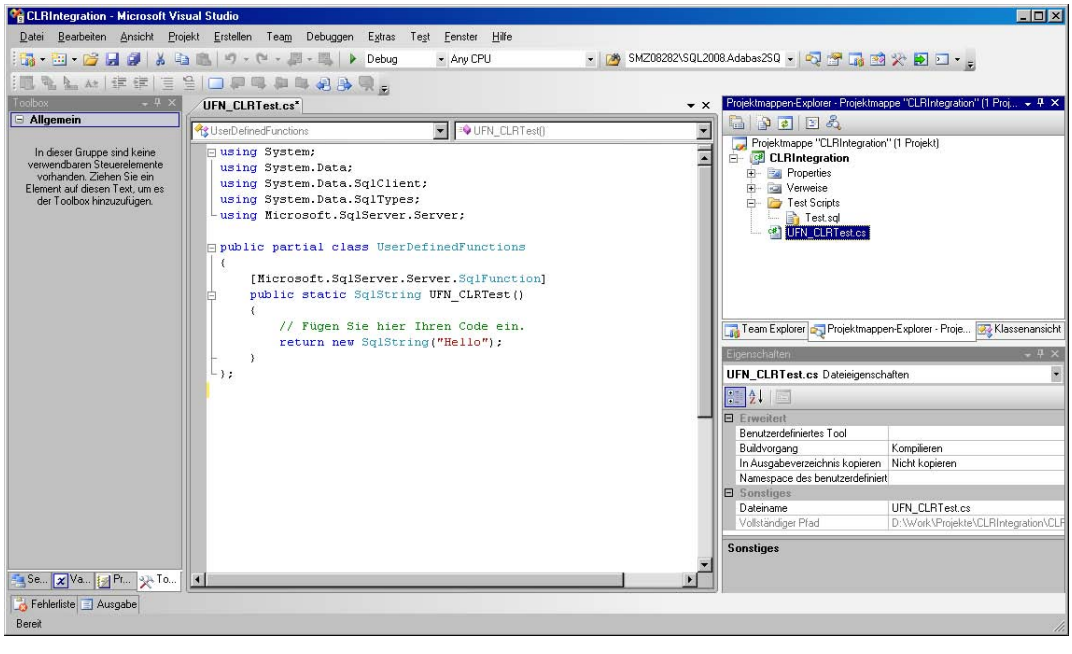


Abbildung 13.8: Das Gerüst für eine benutzerdefinierte Funktion

6. Im Zweig *Test Scripts* des Objekt-Explorers ist eine Datei *Test.sql* zu finden. Hier können Sie SQL-Anweisungen eintragen, um die benutzerdefinierte Funktion zu testen. Ergänzen Sie hier die folgende Abfrage:

   ```
   SELECT dbo.UFN_CLRTest()
   ```

7. Nun können Sie das Projekt erstellen und bereitstellen, indem Sie im Menü *Erstellen* den Punkt *CLRIntegration erstellen* und anschließend *CLRIntegration bereitstellen* auswählen. Dadurch wird die Assembly für die gespeicherte Funktion erzeugt und automatisch auf dem SQL Server registriert.

8. Starten Sie nun das SQL Server Management Studio und öffnen Sie im Objekt-Explorer den Zweig *Programmierbarkeit/Assemblys*. Hier sollte nun die Assembly *CLRIntegration* zu finden sein. Dadurch taucht im Zweig *MediaBase/Programmierbarkeit/Funktionen/Skalarwertfunktionen* auch die darin enthaltene Funktion *UFN_CLRTest* auf, die Sie nun wie jede andere Skalarwertfunktion nutzen können.

9. Testen Sie dies nun, indem Sie in einem Abfragefenster die folgende Abfrage eingeben und ausführen:

   ```
   SELECT dbo.UFN_CLRTest()
   ```

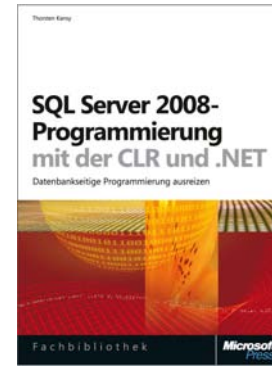
Nachdem Sie nun gelernt haben, wie einfach es ist, mithilfe von .NET selbst entwickelte Funktionalität in den SQL Server zu integrieren, drängt sich natürlich die Frage auf, anhand welcher Kriterien man entscheiden soll, ob beispielsweise eine Funktion eher in .NET oder eher in T-SQL zu implementieren ist. Dazu gibt es bereits seit SQL 2005 eine klare Regel:

- Je aufwendiger die Logik der Routine ist, desto sinnvoller ist der Einsatz von .NET. Insbesondere dann, wenn externe Aufrufe (z.B. von Webdiensten) integriert werden müssen, ist .NET das Mittel der Wahl.
- Je datenintensiver die Funktionalität ist, desto sinnvoller ist der Einsatz von T-SQL, da dieses auf die Verarbeitung von Massendaten hin optimiert ist.

Die Befürchtung, dass T-SQL bald komplett von .NET abgelöst wird, ist also sicherlich fehl am Platz.

Internet-Link: Weitere Infos zur SQL Server-CLR-Integration

Wenn Sie tiefer in das Thema einsteigen möchten, kann ich Ihnen das Buch *SQL Server 2008-Programmierung mit der CLR und .NET* von *Thorsten Kansy* empfehlen. Genauere Informationen zu diesem Buch erhalten Sie über den Softlink **sql1303**.



Als Leseprobe steht Ihnen Kapitel 8 *Skalarfunktionen* über den Softlink **sql1304** zur Verfügung.

Um die Softlink zu nutzen, geben Sie auf der Startseite von *http://www.vsxpress.de* die angegebene Link-ID ein und klicken dann auf *Link öffnen*. Alternativ können Sie auch direkt in der Adressleiste Ihres Browsers die folgenden URLs eingeben:
http://go.vsxpress.de/?linkid=sql1303 bzw.
http://go.vsxpress.de/?linkid=sql1304

13.6 Übungen zu diesem Kapitel

In diesem Abschnitt finden Sie einige Übungen zu diesem Kapitel. Die richtigen Antworten, Lösungen und den Programmcode finden Sie wie immer auf der Website *www.vsxpress.de.*

Übung 13.1

Welche drei Schichten werden bei der 3-Tier-Architektur unterschieden?

Übung 13.2

Von welcher Klasse müssen Sie zuerst ein Objekt erstellen, bevor Sie mit ADO.NET arbeiten können?

Übung 13.3

Wie wird bei LINQ to SQL die Trennung zwischen Datenbankschicht und Anwendungslogik realisiert?

Übung 13.4

Wo wird im ADO.NET Entity Framework die Zuordnung von Tabellen zu Objekten gespeichert?

Übung 13.5

Welche vorbereitenden Schritte müssen Sie auf einem SQL Server ausführen, um eine in .NET erstellte gespeicherte Prozedur nutzen zu können?

13.7 Zusammenfassung

Zu Beginn des Kapitels haben Sie gelernt, warum eine Aufteilung der Anwendungsarchitektur in Datenbankschicht, Geschäftslogik und Präsentationsschicht Sinn machen kann.

Am Beispiel des `SqlDataReader` haben Sie gesehen, wie der Datenzugriff über ADO.NET erfolgen kann. Anschließend wurde dieselbe Beispielanwendung auf Basis von LINQ to SQL erläutert. Aber erst das ADO.NET Entity Framework – das LINQ to SQL mittelfristig ersetzen soll – bietet eine richtige Trennung zwischen Datenbankschicht und Geschäftsobjekten, da diese weitgehend unabhängig voneinander modelliert werden können. Auch wenn LINQ to SQL irgendwann aussterben wird, können Sie die Abfragesyntax von LINQ nach wie vor nutzen, um beispielsweise mit der Implementierung LINQ to Entities auf Klassen des Entity Framework zuzugreifen.

Die gezeigten Beispiele stellen natürlich nur einen minimalen Auszug der jeweiligen Datenzugriffstechnologie dar. Dies sollte jedoch ausreichen, um eine ungefähre Vorstellung der Funktionsweise zu bekommen.

Im letzten Abschnitt des Kapitels wurde die SQL Server CLR-Integration erläutert und demonstriert. Mit dieser kann man gespeicherte Prozeduren, Funktionen, Aggregate, Trigger und sogar eigene Datentypen in .NET entwickeln und anschließend im SQL Server registrieren, sodass diese anschließend wie vergleichbare mit SQL erzeugte Objekte nutzbar sind.

Kapitel 14

Reporting mit SQL Server Express mit Advanced Services

In diesem Kapitel lernen Sie

- was die SQL Server Reporting Services leisten
- wie Sie die SQL Server Reporting Services konfigurieren
- wie man mit dem Report-Designer einen Bericht entwirft

14.1 Überblick über die Reporting Services

Die SQL Server Reporting Services ermöglichen das komfortable Erstellen, Verwalten und Verteilen von Berichten auf Basis von Daten, die beispielsweise vom SQL Server bereitgestellt werden. Im Gegensatz zu Tools wie dem mit vielen Varianten von Visual Studio ausgelieferten Crystal Reports sind die SQL Server Reporting Services jedoch ganz klar für Microsoft SQL Server optimiert (was nicht heißen soll, dass keine anderen Datenquellen unterstützt werden).

Bei den Reporting Services, die mit SQL Server Express mit Advanced Services ausgeliefert werden, handelt es sich um eine leicht abgespeckte Variante, die nicht den vollen Funktionsumfang des Produkts aufweist. So ist beispielsweise der Report Builder, mit dem Ad-hoc-Reports erstellt werden können, nur für die Web, Standard und Enterprise Edition von SQL Server verfügbar. Die Report-Verteilung per E-Mail oder Laufwerksfreigabe ist – genau wie die Möglichkeit, Reports zeitgesteuert zu erstellen und zu abonnieren – der Standard und Enterprise Edition vorbehalten. Ab diesen Editionen ist auch die 4-GB-Speichergrenze aufgehoben, die für die Reporting Services der kleineren Editionen gilt.

Um mit der obigen Aufzählung nicht den Eindruck zu erwecken, dass die Reporting Services der Express Edition keine interessanten Features zu bieten haben, sollte natürlich auch erwähnt werden, welche zentralen Bestandteile und Möglichkeiten es auch mit dieser Edition gibt:

- *Report-Designer* (Bestandteil des *Business Intelligence Development Studio*) zum komfortablen Erstellen von Berichten
- *Report Server* der die Berichte zur Verfügung stellt
- *Report Manager* zum Verwalten von Berichten über ein Web-Frontend

- Nutzung von verschiedenen Datenquellen für Berichte (OLE DB, ODBC, XML, Microsoft SQL Server, Oracle, SAP NetWeaver BI, Hyperion Essbase, TERADATA etc.)
- Export der Berichte nach XML, CSV, TIFF, PDF (Adobe), MHTML (Webarchiv), Excel und Word

Während die Reporting Services der 2005er-Generation von SQL Server noch einen Internet Information Server benötigten, liefert die 2008er-Generation einen eigenen Webserver mit, der ausschließlich für die Reporting Services genutzt wird und daher auch erheblich geringeren Konfigurationsaufwand erfordert. Sollten Sie bereits einen Internet Information Server im Einsatz haben, können Sie die Berichte aber selbstverständlich auch über diesen veröffentlichen und bereitstellen.

14.2 Konfiguration der Reporting Services

Bevor die Reporting Services genutzt werden können, müssen sie erst einmal konfiguriert werden. Bereits bei der Installation des SQL Servers ist die Auswahl zu treffen, ob und wenn ja, in welchem Modus die Reporting Services konfiguriert werden sollten.

Wenn Sie der Anleitung in Kapitel 3 gefolgt sind, wurden die Reporting Services in der *Standardkonfiguration des systemeigenen Modus* eingerichtet. Damit stehen die Reporting Services sofort nach Abschluss der Installation zur Verwendung bereit. Alternativ standen noch zwei weitere Varianten zur Auswahl, auf die ich kurz eingehen will.

Die *Standardkonfiguration des integrierten Sharepoint-Modus* konfiguriert die Reporting Services so, dass sie in Verbindung mit SharePoint verwendet werden können. Sollten Sie kein SharePoint-Produkt verwenden, ist diese Variante nutzlos.

Als dritte Möglichkeit stand die Variante *Berichtsserver installieren, aber nicht konfigurieren* zur Auswahl. Dadurch werden die notwendigen Komponenten der Reporting Services zwar installiert, aber nicht vorkonfiguriert. Um die Konfiguration vorzunehmen, wird der Konfigurations-Manager für Reporting Services verwendet.

Überprüfen bzw. vervollständigen Sie nun die Konfiguration der Reporting Services:

1. Rufen Sie den Reporting Services Konfigurations-Manager auf, der im Windows-Startmenü unter *Alle Programme/Microsoft SQL Server 2008/Konfigurationstools* zu finden ist.
2. Nach Aufruf des Konfigurations-Managers ist zuerst eine Verbindung mit der Berichtsserverinstanz herzustellen. Dabei entspricht der Instanzname dem Namen der SQL Server-Instanz.

Abbildung 14.1: Verbindungsaufbau zu den SQL Server Reporting Services

3. In der Startanzeige des Reporting Server Konfigurations-Managers sehen Sie verschiedene Angaben zu SQL Server-Instanz, Version (Build-Nr.), Berichtsservermodus, Status des Berichtsserverdienstes etc. Prüfen Sie hier, dass der Dienst gestartet ist. Falls nicht, können Sie das über die entsprechende Schaltfläche nachholen.
4. Klicken Sie in der Navigation auf der linken Seite auf den Link *Dienstkonto*. Hier ist das Benutzerkonto auszuwählen, mit dem der Reporting Services-Dienst läuft. Da in Ihrer Konfiguration sowohl SQL Server als auch Reporting Services und der zu verwendende Webserver auf demselben Rechner laufen, reicht hier das Konto *Lokaler Dienst* völlig aus.
5. Klicken Sie in der Navigation auf *Webdienst-URL,* um den zu verwendenden Webserver zu konfigurieren. Hier ist das virtuelle Verzeichnis angegeben, in dem die Berichte später zur Verfügung gestellt werden. Dazu sind die zu verwendende *IP-Adresse* und der *TCP-Port* sowie Parameter zur SSL-Verschlüsselung auszuwählen. Behalten Sie hier ruhig die in Abbildung 14.2 dargestellten Standardwerte bei.

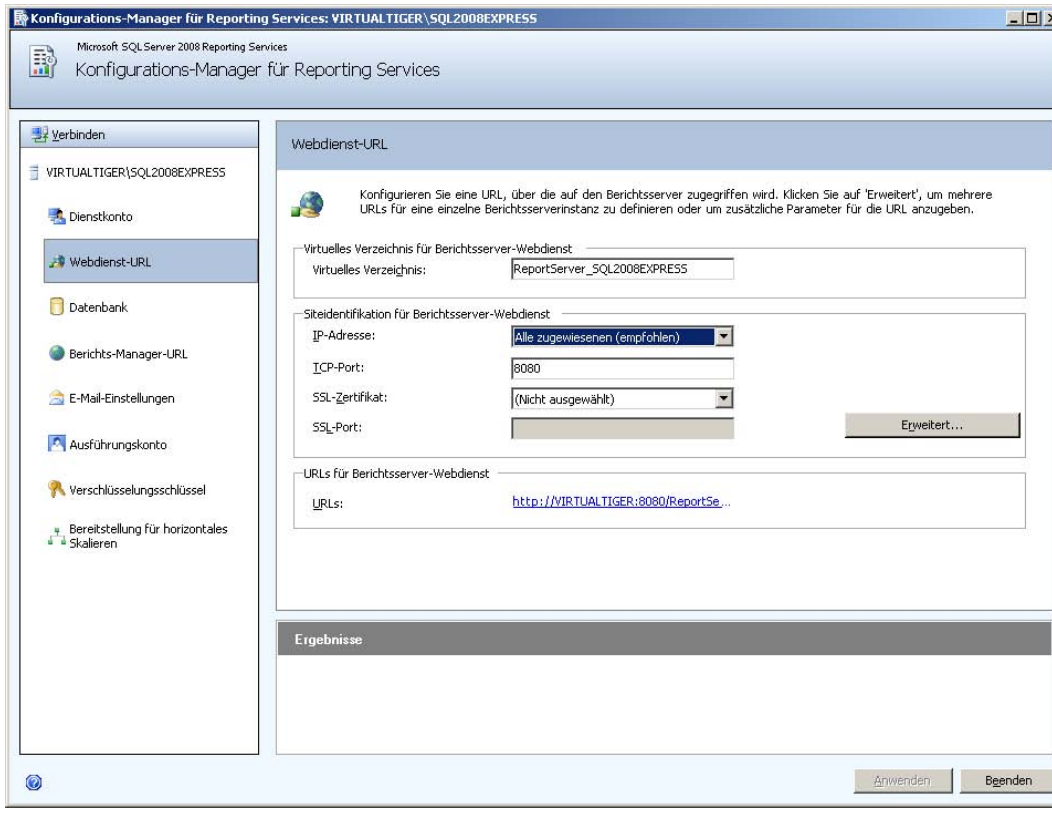


Abbildung 14.2: Konfiguration des Webdienstes

Wenn Sie stattdessen ein noch nicht existierendes virtuelles Verzeichnis angeben, so wird dieses automatisch erstellt, sobald Sie auf die *Anwenden*-Schaltfläche klicken. Im unteren Bereich des Dialogfeldes wird die komplette URL (z.B. *http://virtualtiger:8080/ReportServer_SQL2008EXPRESS*) angezeigt, über die auf das virtuelle Verzeichnis zugegriffen werden kann.

6. Auf der Seite *Datenbank* bekommen Sie die aktuell ausgewählte Berichtsserver-Datenbank sowie die Anmeldeinformationen hierzu angezeigt. Wenn Sie Datenbank oder Anmeldeinformationen ändern möchten, klicken Sie auf die entsprechende Schaltfläche (über die Schaltfläche *Datenbank ändern* lassen sich sogar neue Berichtsserver-Datenbanken erstellen). Lassen Sie die Einstellung hier aber wie vorgegeben.

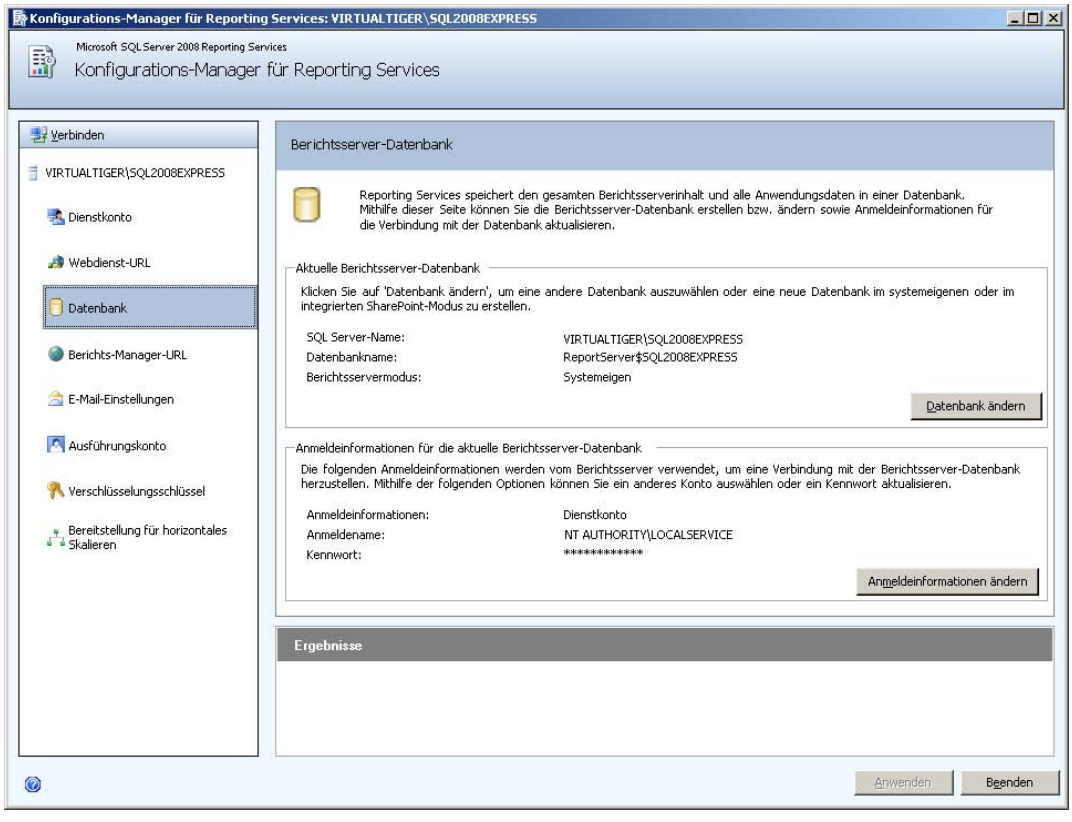


Abbildung 14.3: Konfiguration der Berichtsserver-Datenbank

7. Klicken Sie in der Navigation auf der linken Seite auf *Berichts-Manager-URL*, um das virtuelle Verzeichnis für den Berichts-Manager zu überprüfen bzw. anzupassen.

   Das Verzeichnis sollte auf *Reports_SQL2008EXPRESS* voreingestellt sein. Auch hier ist die entsprechende URL (z.B. *http://virtualtiger:8080/Reports_SQL2008EXPRESS/Pages/Folder.aspx*) angezeigt, über die Sie den Berichts-Manager später erreichen können.

8. Die folgenden Seiten des Konfigurations-Managers sind momentan weniger relevant, sodass wir die Konfiguration nun mit einem Klick auf *Beenden* abschließen können.

Nachdem nun sichergestellt ist, dass die Reporting Services richtig konfiguriert sind, können wir uns mit dem Erstellen von Berichten befassen.

14.3 Erstellen eines Reports mit dem Report-Designer

Zum komfortablen Erstellen eines Reports wird der Report-Designer verwendet. Dieser ist Bestandteil des Business Intelligence Development Studio, das zusammen mit SQL Server installiert wurde.

Hintergrundinfo: Business Intelligence Development Studio

Beim Business Intelligence Development Studio (oder kurz: BIDS) handelt es sich eigentlich um die Entwicklungsumgebung Microsoft Visual Studio, die aber – anstelle von Projekttypen für VB.NET oder C#-Anwendungen – Vorlagen für Projekte aus dem Umfeld von Reporting Services (SSRS) sowie – sofern eine größere Edition von SQL Server verwendet wird – Integration Services (SSIS) und Analysis Services (SSAS) bereithält. Wenn Sie also bereits mit Visual Studio gearbeitet haben, werden Sie sich auch im Business Intelligence Development Studio schnell zurechtfinden.

Sollten Sie sowohl das Business Intelligence Development Studio als auch Visual Studio installiert haben, sind die Projekttypen beider Tools unter einer Oberfläche verfügbar.

Erstellen wir nun einen einfachen Bericht, der die Daten aus den Tabellen *dbo.CD* und *dbo.CDTrack* zusammenfasst:

1. Rufen Sie das Business Intelligence Development Studio auf, das im Windows-Startmenü unter *Alle Programme/Microsoft SQL Server 2008* zu finden ist. (Alternativ können Sie auch den Eintrag *Alle Programme/Microsoft Visual Studio 2008/Microsoft Visual Studio 2008* nutzen, da es sich hier wirklich um dasselbe Programm handelt, bei dem lediglich die verfügbaren Projekttypen je nach installierter Variante variieren.)
2. Klicken Sie auf den Menüpunkt *Datei/Neu/Projekt,* um ein neues Projekt zu erstellen.

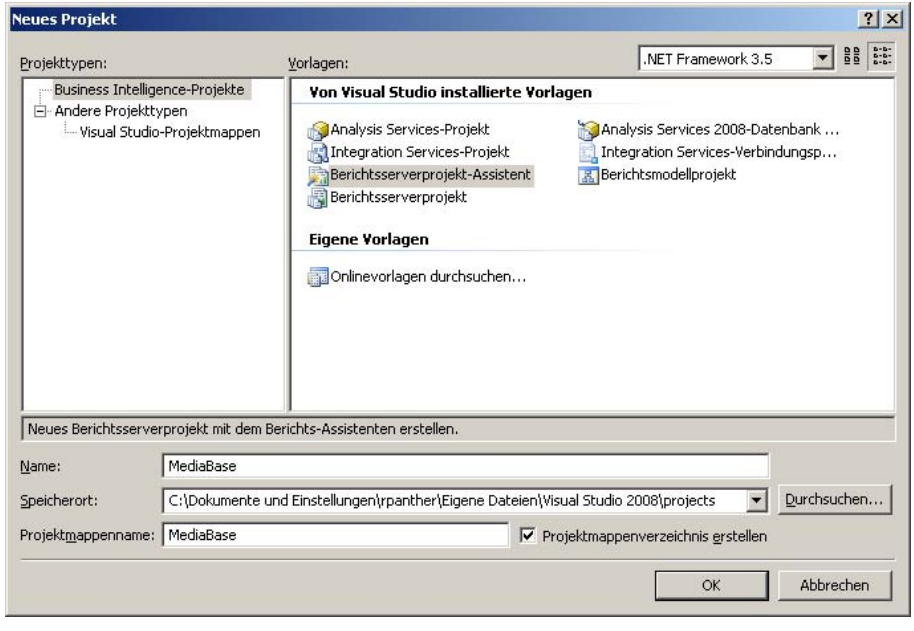


Abbildung 14.4: Das Dialogfeld zum Erstellen eines neuen Projekts

3. Wenn Sie vorher Visual Studio installiert hatten, erscheinen neben den in Abbildung 14.4 dargestellten Projekttypen noch einige mehr zur Auswahl. Sollten Sie aber lediglich das Business Intelligence Development Studio (das zusammen mit SQL Server kommt) installiert haben, entsprechen die zur Verfügung stehenden Projekttypen der Abbildung oben. Wählen Sie in jedem Fall *Business Intelligence-Projekte* als Projekttyp aus und es werden dann genau die *Vorlagen* angezeigt, die zum Business Intelligence Development Studio gehören. Für SQL Server Express mit Advanced Services sind hiervon lediglich die beiden Vorlagen für Berichtsserverprojekte und die Vorlage für Berichtsmodellprojekte relevant. Da die Dienste für Analysis Services und Integration Services für die Express Edition nicht verfügbar sind, machen auch die entsprechenden Projektvorlagen hier wenig Sinn. Berichtsmodellprojekte werden einerseits dazu verwendet, Berichtsmodelle für Ad-hoc-Berichte zu erstellen. Letztere sind allerdings nur für größere Editionen von SQL Server verfügbar. Andererseits können Sie die Berichtsmodelle auch als Datenquelle für ein Berichtsserverprojekt nutzen. So kompliziert wollen wir es im Moment aber nicht machen und konzentrieren uns daher auf die Vorlagen für Berichtsserverprojekte.
4. Da wir möglichst wenig manuell erledigen wollen, wählen Sie die Vorlage *Berichtsserverprojekt-Assistent* und geben dem Projekt den Namen **MediaBase**. Nach einem Klick auf die Schaltfläche *OK* wird ein leeres Projekt angelegt und der Assistent führt Sie Schritt für Schritt durch die Erstellung eines Berichts.
5. Zuerst werden Sie nach der Datenquelle gefragt. Geben Sie auch dieser den Namen **MediaBase** und wählen Sie den Typ *Microsoft SQL Server*. Durch einen Klick auf die Schaltfläche *Bearbeiten* öffnet sich ein Dialogfeld, in dem Sie die Verbindungsdaten eingeben können. Setzen Sie hier den Servernamen auf **.\SQL2008EXPRESS** (damit auch die richtige Serverinstanz verwendet wird), wählen Sie dann *Windows-Authentifizierung* und als Datenbank *MediaBase,* bevor Sie das Dialogfeld durch einen Klick auf *OK* wieder schließen.

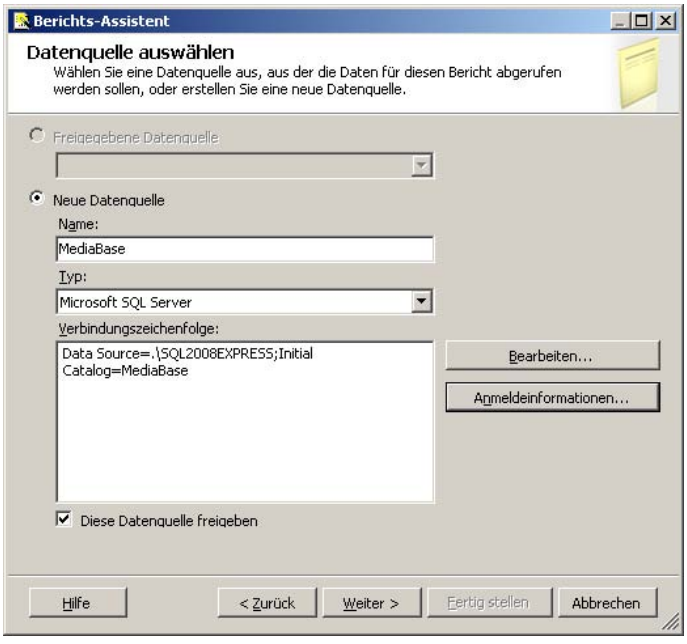


Abbildung 14.5: Das Dialogfeld zum Auswählen einer Datenquelle

6. Aktivieren Sie das Kontrollkästchen *Diese Datenquelle freigeben*, damit die Datenquelle später auch für andere Berichte verwendet werden kann, ohne alle Verbindungsdaten noch einmal eingeben zu müssen. Anschließend klicken Sie auf *Weiter*, damit der Assistent fortfahren kann.
7. Es folgt ein Schritt, in dem Sie eine SQL-Abfrage, auf der später der Bericht basiert, eingeben können. Da dies natürlich recht unkomfortabel ist, klicken Sie auf die Schaltfläche *Abfrage-Generator* und es wird der Abfrage-Designer geöffnet, in dem Sie – ähnlich wie im SQL Server Management Studio – komfortabel eine Abfrage entwerfen können. Klicken Sie hier in der Symbolleiste das Symbol ganz rechts an, um der Abfrage Tabellen hinzuzufügen, und wählen Sie anschließend die Tabellen *CD* und *CDTrack* aus, die Sie über die Schaltfläche *Hinzufügen* im Abfrage-Designer erscheinen lassen.
8. Markieren Sie anschließend in der Tabelle *CD* die Spalten *Titel* und *Kategorie* sowie in der Tabelle *CDTrack* die Spalten *CDNr*, *TrackNr*, *Titel* und *Interpret*. Da nun zwei Spalten mit dem Namen *Titel* auftauchen, wurde für den *Titel* aus der Tabelle *CDTrack* automatisch das Alias *Expr1* vergeben. Um die Bezeichnungen etwas sprechender zu gestalten, ändern Sie dieses Alias in *TrackTitel* und schließen dann den Designer über die *OK*-Schaltfläche.

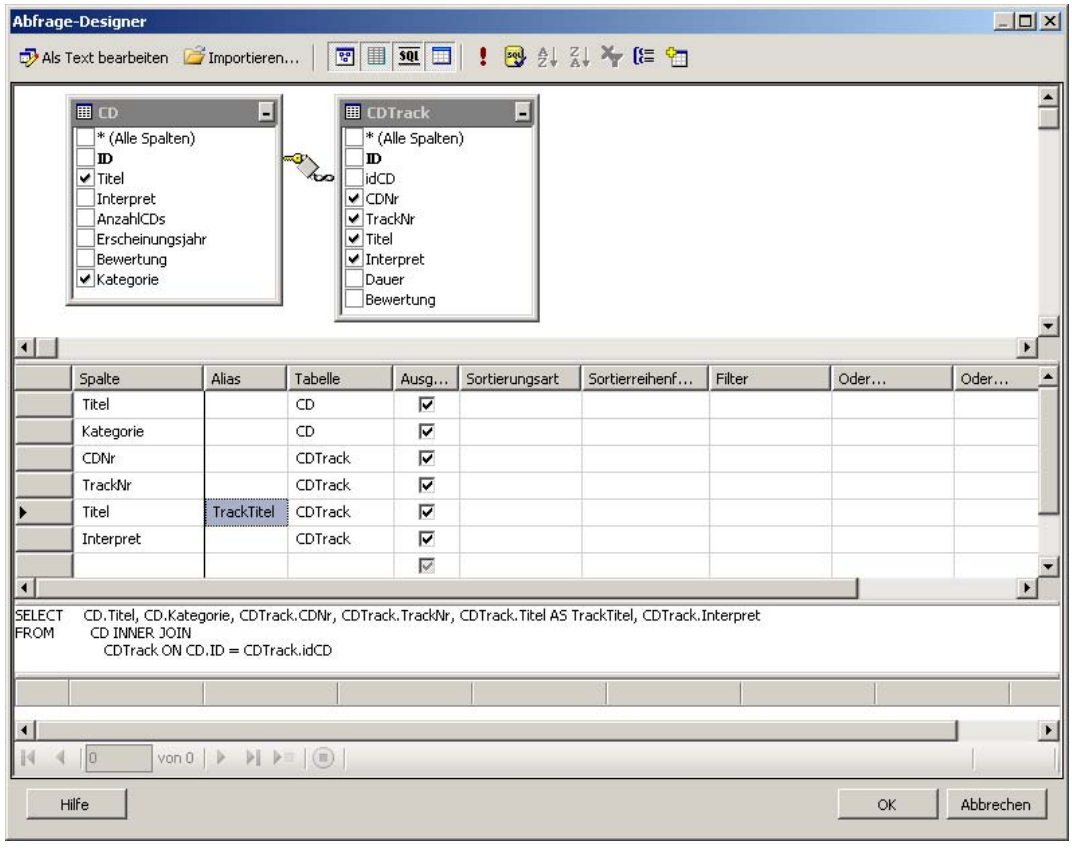


Abbildung 14.6: Der Abfrage-Designer

9. Im nächsten Schritt des Berichts-Assistenten werden Sie nach dem Typ des Berichts gefragt. Behalten Sie hier die Voreinstellung *Tabellarisch* bei und klicken Sie auf *Weiter*.

10. Beim folgenden Schritt, der mit *Tabelle entwerfen* benannt ist, geht es darum zu entscheiden, welche Spalten für Seitenköpfe, Gruppierungen und Detaildarstellungen verwendet werden. Wählen Sie hier die *Kategorie* als Seitenkopf, den *Titel* als Gruppierung und die restlichen Felder als Detailinfos, wie in Abbildung 14.7 dargestellt.

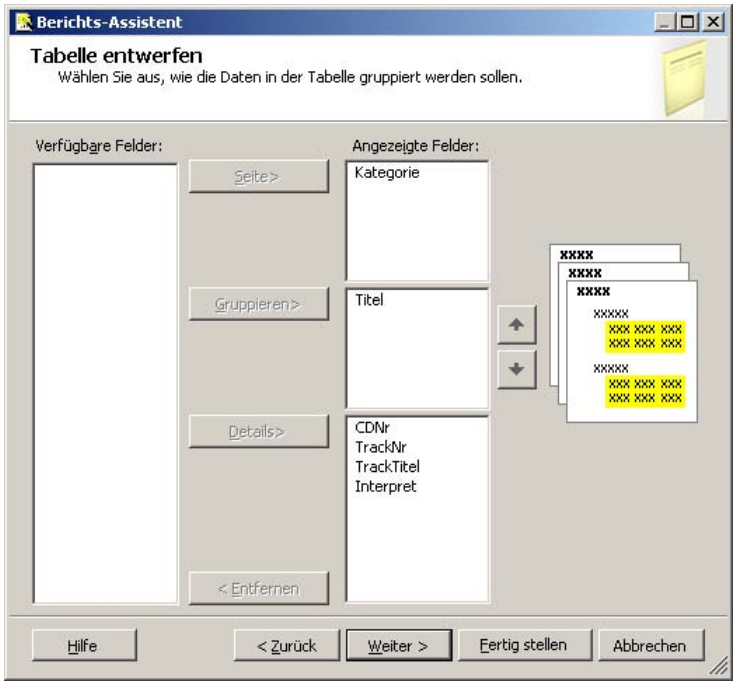


Abbildung 14.7: Das Dialogfeld zum Festlegen der Tabellengestaltung

11. Wählen Sie im nächsten Schritt das Tabellenlayout *Abgestuft* und aktivieren Sie das Kontrollkästchen *Drilldown aktivieren*, damit später gruppierte Bereiche per Mausklick auf- und zugeklappt werden können.
12. Im nächsten Schritt können Sie ein Tabellenformat wählen, das Ihnen zusagt (mein persönlicher Favorit ist *Geschäftlich*, aber das ist sicherlich Geschmackssache).
13. Bei dem folgenden Schritt zur Angabe des Bereitstellungsspeicherorts geben Sie als Berichtsserver **http://localhost/ReportServer_SQL2008EXPRESS** an. Als Bereitstellungsordner kann die Vorgabe *MediaBase* übernommen werden.
14. Nach einem letzten Klick auf *Weiter* erscheint noch eine Berichtszusammenfassung und Sie können einen Namen für den Bericht eingeben (z.B. **CDsMitTracks**) sowie über ein Kontrollkästchen die *Berichtsvorschau* aktivieren.

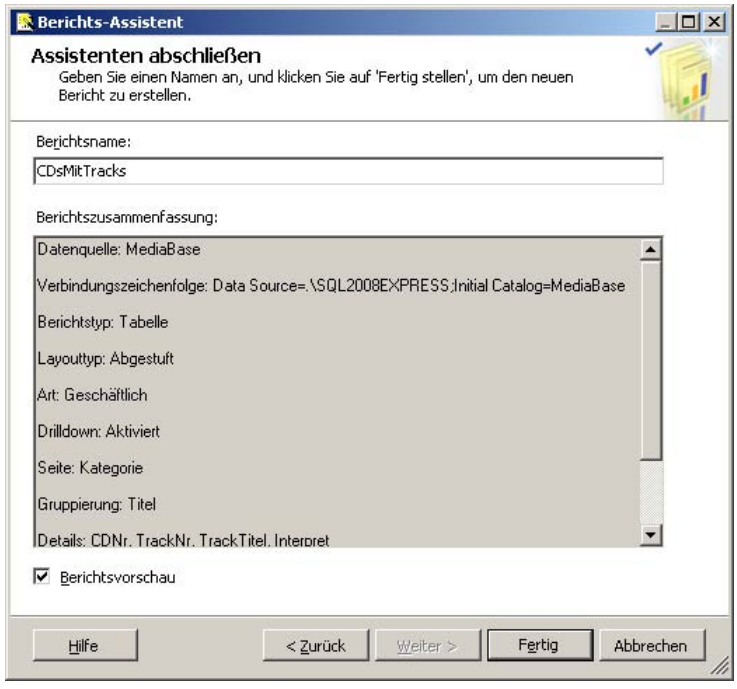


Abbildung 14.8: Die Berichtszusammenfassung

15. Nachdem Sie auf die Schaltfläche *Fertig* geklickt haben, dauert es einige Zeit, bis der Bericht nach den Vorgaben des Assistenten generiert wurde. Anschließend erscheint direkt die Berichtsvorschau (sofern Sie diese Option markiert haben) oder Sie gelangen in den Entwurfsmodus des Berichts. Zwischen beiden Modi können Sie über die entsprechenden Registerkarten jederzeit hin und her wechseln. In der Berichtsvorschau können Sie nun zwischen den einzelnen Seiten (und damit CD-Kategorien) blättern und zu jeder CD die entsprechenden Tracks auf- und zuklappen.

16. Als letzten Schritt dieser Praxisübung wollen wir den Bericht exportieren. Klicken Sie dazu – während Sie sich im Vorschaumodus befinden – auf das kleine Diskettensymbol in der Symbolleiste des Berichts. Es klappt eine Liste herunter, in der Sie das Zielformat auswählen können. Zur Auswahl stehen die Formate XML, CSV, TIFF, PDF, MHTML, Excel und Word. Klicken Sie auf das Format Ihrer Wahl und nach kurzer Zeit erscheint ein Dialogfeld, in dem Sie den Namen und das Verzeichnis der Exportdatei angeben können. Geben Sie hier einen passenden Namen ein und klicken Sie dann auf *Speichern*, um die Datei zu erzeugen.

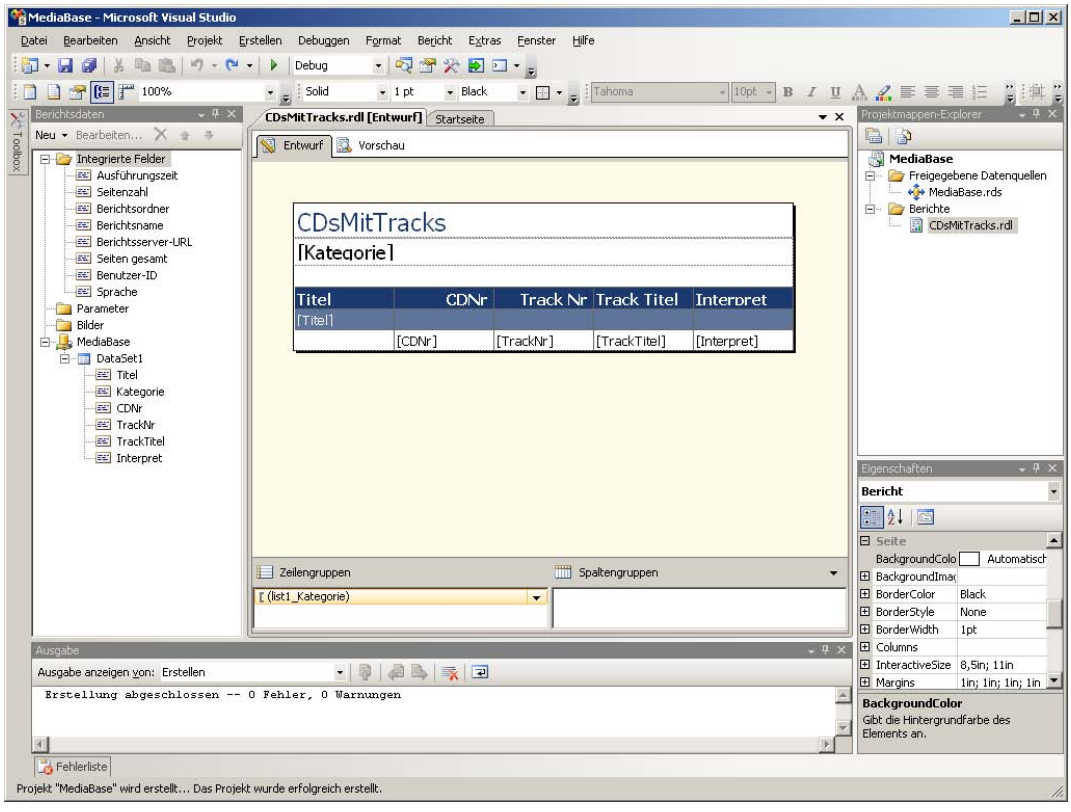


Abbildung 14.9: Der Berichtsentwurfsmodus

Wenn Sie sich die Entwicklungsoberfläche einmal ansehen, stechen vor allem drei Bereiche ins Auge:

- In der Mitte sind die bereits erwähnten Registerkarten zu sehen, mit denen Sie einen Bericht entwerfen oder in der Vorschau betrachten können.
- Auf der linken Seite sind die Berichtsdaten aufgelistet, mit denen Sie einen Bericht gestalten können. Neben den Spalten der Datenquelle sind hier auch einige *Integrierte Felder* zu finden, über die Sie Angaben wie beispielsweise die Seitenzahl in Berichte integrieren können. Sowohl integrierte Felder als auch Datenspalten lassen sich einfach mit der Maus auf den Bericht ziehen, um diese Elemente dort zu platzieren.
- Auf der rechten Seite ist der Projektmappen-Explorer angezeigt, in dem die einzelnen Dateien des Berichtsserverprojekts zu sehen sind. Neben dem Bericht selbst, der in der Datei *CDsMitTracks.rdl* gespeichert ist, ist hier auch die Datenquelle *MediaBase.rds* zu finden, die gespeichert wurde, als Sie bei der Wahl der Datenquelle die Option *Diese Datenquelle freigeben* aktiviert haben.

Hintergrundinfo: Report Definition Language

Die Berichte werden in der Report Definition Language (RDL) abgelegt. Dabei handelt es sich um einen von Microsoft initiierten Standard, der Berichte in Form eines XML-Dokuments beschreibt. Inzwischen unterstützen auch einige Drittanbieter das RDL-Format.

14.4 Übungen zu diesem Kapitel

In diesem Abschnitt finden Sie einige Übungen zu diesem Kapitel. Die richtigen Antworten und Lösungen finden Sie wie immer auf der Website *www.vsxpress.de.*

Übung 14.1

Erweitern Sie den Bericht *CDsMitTracks* so, dass in der Titelleiste auch der Zeitpunkt der Ausführung angezeigt wird.

Übung 14.2

Erstellen Sie einen zweiten Bericht im selben Berichtsprojekt, der alle Bücher nach Autoren gruppiert auflistet.

Tipp: Neuen Bericht im bestehenden Projekt erstellen

Um einen weiteren Bericht in einem existierenden Berichtsserverprojekt zu erstellen, klicken Sie im Projektmappen-Explorer mit der rechten Maustaste auf *Berichte* und wählen dann die Option *Neuen Bericht hinzufügen*. Dadurch startet wieder der Berichts-Assistent, bei dem Sie auch freigegebene Datenquellen nutzen können.

14.5 Zusammenfassung

Nach einem kurzen Überblick über die wichtigsten Features der SQL Server Reporting Services haben Sie gesehen, wie diese zu konfigurieren sind. Anschließend haben Sie mithilfe des Business Intelligence Development Studio ein Berichtsserverprojekt und darin einen Bericht erstellt.

Natürlich gehen die Möglichkeiten der Reporting Services weit über das hinaus, was in diesem Kapitel mithilfe des Berichtserstellungs-Assistenten erzeugt wurde. Insbesondere die Themen Verteilen von Berichten und Nutzen von Berichtsmodellen, aber auch die Parametrisierung von Reports und viele interessante Teilaspekte mehr konnten hier leider nicht behandelt werden.

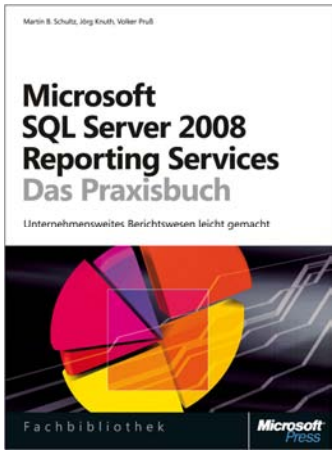
Jedoch sollte diese kurze Einführung in die Reporting Services ausreichen, um Ihnen einen kleinen Einblick in die Arbeit mit den SQL Server Reporting Services zu geben. Wenn Sie tiefer in die Materie einsteigen möchten, so gibt es hier zahlreiche Bücher, die sich ausschließlich mit den SQL Server Reporting Services beschäftigen.

Internet-Link: Weitere Infos zu SQL Server 2008 Reporting Services

Wenn Sie tiefer in das Thema einsteigen möchten, kann ich Ihnen das Buch *Microsoft SQL Server 2008 Reporting Services – Das Praxisbuch* von *Martin B. Schultz*, *Jörg Knuth* und *Sven Bayer* empfehlen. Genauere Informationen zu diesem Buch erhalten Sie über den Softlink **sql1401**.

Als Leseprobe aus diesem Buch steht Ihnen Kapitel 10, *Berichtselemente* über den Softlink **sql1402** zur Verfügung.

Um die Softlink zu nutzen, geben Sie auf der Startseite von *http://www.vsxpress.de* die angegebene Link-ID ein und klicken dann auf *Link öffnen*. Alternativ können Sie auch direkt in der Adressleiste Ihres Browsers die folgenden URL eingeben:
http://go.vsxpress.de/?linkid=sql1401 bzw.
http://go.vsxpress.de/?linkid=sql1402



Kapitel 15

Zusammenarbeit mit anderen SQL Server-Instanzen und -Editionen

In diesem Kapitel lernen Sie

- welche Möglichkeiten es gibt, mehrere SQL Server gemeinsam zu nutzen
- was es mit der Replikation von Datenbanken auf sich hat
- wie Sie SQL Server Express in Verbindung mit der SQL Server Compact Edition nutzen können
- wie Sie ein Upgrade von SQL Server Express auf eine größere Edition durchführen können

15.1 Verbindung zu anderen Servern

SQL Server Express kann auch mit anderen Servern zusammenarbeiten (selbst wenn diese eine größere Edition von SQL Server verwenden).

Die einfachste Form, Server gemeinsam zu nutzen, ist durch die Einrichtung von Verbindungsservern (engl. linked server). Dadurch wird einem Server bekannt gemacht, wie – und mit welchen Anmeldungen – er auf einen zweiten Server zugreifen kann, um serverübergreifende Abfragen durchführen zu können.

Zum Ansprechen einer Tabelle auf einem entfernten Server wird dem Datenbanknamen (wiederum durch einen Punkt getrennt) einfach der Name des Servers vorangestellt. Handelt es sich um eine benannte Instanz auf dem entfernten Server, muss auch der Instanzname angegeben werden (wie üblich durch einen umgekehrten Schrägstrich vom Servernamen getrennt). In diesem Fall muss die Kombination aus Server- und Instanznamen allerdings in eckige Klammern eingeschlossen werden. Um beispielsweise die Tabelle *dbo.CD* aus der Datenbank *MediaBase* von einem Server mit Namen *ZweitServer* und der Instanz *SQL2008Express* abzufragen, ist die folgende Abfrage zu verwenden:

```
SELECT * FROM [ZweitServer\SQL2008].MediaBase.dbo.CD
```

Sofern der entfernte Server allerdings noch nicht als Verbindungsserver eingerichtet ist, erhalten Sie eine Fehlermeldung der folgenden Art:

Meldung 7202, Ebene 11, Status 2, Zeile 1

Server 'ZweitServer\SQL2008' konnte in 'sys.servers' nicht gefunden werden. Stellen Sie sicher, dass der richtige Servername angegeben wurde. Führen Sie ggf. die gespeicherte Prozedur 'sp_addlinkedserver' aus, um den Server in 'sys.servers' hinzuzufügen.

Um dies nachzuprüfen, können Sie mit der folgenden Abfrage einen Blick in die entsprechende Systemtabelle werfen:

```
SELECT * FROM sys.servers
```

Als Ergebnis sollte nur eine Zeile, die den lokalen Server beschreibt, auftauchen.

Um nun einen Verbindungsserver einzurichten, sind folgende Schritte durchzuführen:

1. Stellen Sie im SQL Server Management Studio eine Verbindung zur lokalen Serverinstanz *SQL2008Express* her.
2. Klicken Sie im Objekt-Explorer mit der rechten Maustaste auf *Serverobjekte/Verbindungsserver* und wählen Sie die Option *Neuer Verbindungsserver*. Es erscheint das Dialogfeld zum Einrichten eines neuen Verbindungsservers.
3. Geben Sie auf der Seite *Allgemein* den Namen der Instanz des Verbindungsservers ein (z.B. **Zweit-Server\SQL2008**) und wählen Sie darunter *SQL Server* als *Servertyp*.
4. Wechseln Sie anschließend zur Seite *Sicherheit*, um dort anzugeben, mit welcher Anmeldung die Verbindung hergestellt werden soll.

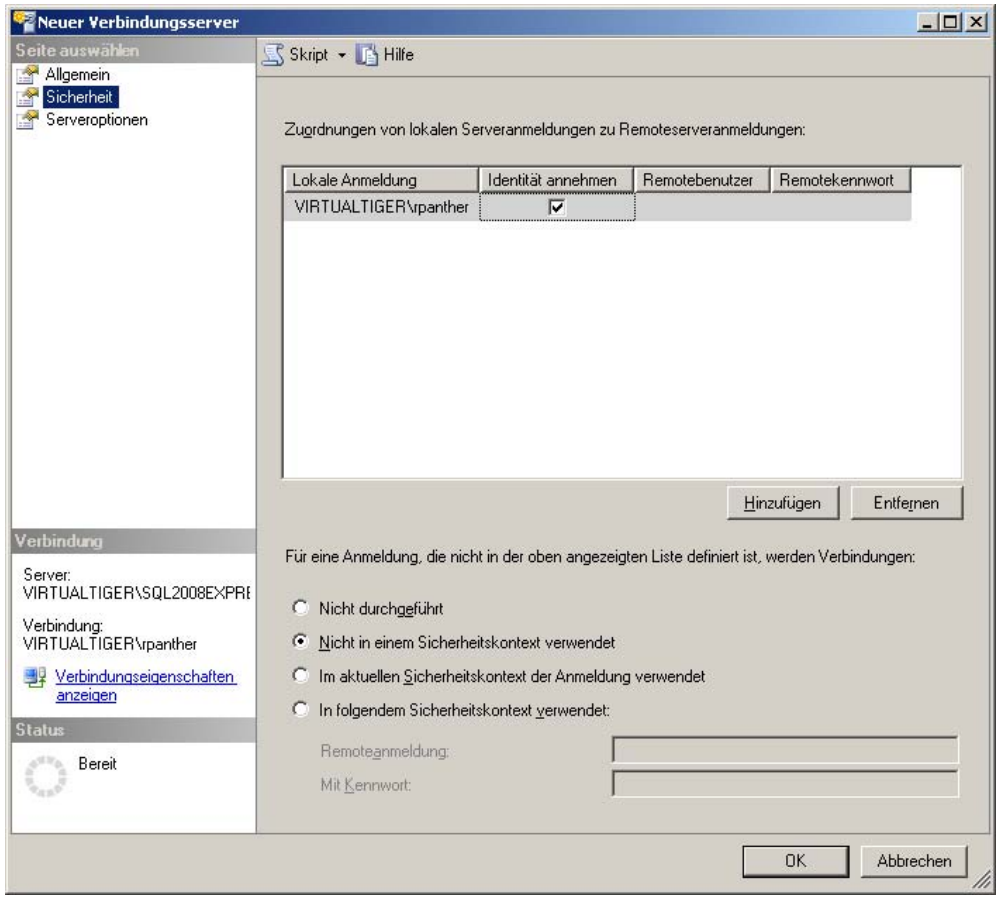


Abbildung 15.1: Zuordnung von Anmeldungen für neue Verbindungsserver

5. Klicken Sie auf *Hinzufügen*, um der Liste eine neue Anmeldungszuordnung hinzuzufügen. Hier wählen Sie dann auf der linken Seite die für den lokalen Server gültige Anmeldung (z.B. *VIRTUALTIGER\rpanther*). Wenn Sie nun das Kästchen *Identität annehmen* anklicken, versucht der lokale Server auch über die zuvor angegebene Anmeldung auf den entfernten Server zuzugreifen. Wenn Sie stattdessen das Kästchen nicht anklicken, können Sie Benutzer und Kennwort für den entfernten Server explizit angeben, die verwendet werden sollen, sobald versucht wird, über die lokale Anmeldung eine Verbindung aufzubauen. Über die Auswahl im unteren Bereich können Sie definieren, wie verfahren werden soll, wenn die Verbindung über keinen der explizit angegebenen Anmeldungen aufgebaut wird.
6. Klicken Sie nun auf *OK* und der Verbindungsserver wird eingerichtet.

Alternativ können Sie auch mit folgender Anweisung den Verbindungsserver hinzufügen und anschließend die Zuordnung von lokaler Anmeldung zur Anmeldung für den entfernten Server erstellen:

```
USE [master]
GO
EXEC master.dbo.sp_addlinkedserver @server = N'ZWEITSERVER\SQL2008', @srvproduct=N'SQL Server'
GO
EXEC master.dbo.sp_addlinkedsrvlogin @rmtsrvname = N'ZWEITSERVER\SQL2008', @locallogin = N'VIRTUALTIGER\rpanther',
@useself = N'True'
GO
```

Sobald der Verbindungsserver eingerichtet ist, können Sie im Objekt-Explorer unter *Serverobjekte/Verbindungsserver* sowohl die verbundene Serverinstanz sehen als auch dessen Datenbanken einblenden (sofern die verwendete Anmeldung die notwendigen Rechte besitzt).

Nun sollte auch die Abfrage auf den verbundenen Server funktionieren. Wenn Sie die Systemtabelle *sys.servers* erneut abfragen, erscheint dort nun auch eine zweite Zeile, in der die Infos zum Verbindungsserver zu sehen sind.

Während Sie mithilfe von Verbindungsservern Daten von verschiedenen Servern explizit gemeinsam nutzen können, bietet SQL Server auch Möglichkeiten an, um Daten zwischen Servern automatisch abzugleichen.

15.2 Replikation

Das Thema der Replikation ist sehr komplex. Dazu kommt die Tatsache, dass eine SQL Server Express-Instanz immer nur als Replikations-Client fungieren kann, der eine Instanz einer größeren SQL Server Edition benötigt, von der er die Daten beziehen kann. Aus diesem Grund möchte ich an dieser Stelle nur einen kurzen Überblick geben, worum es bei der Replikation geht und welche Rolle SQL Server Express in einem Replikationsszenario spielen kann.

Überblick über die SQL Server-Replikation

Die Replikation dient zum automatischen Austausch von Daten zwischen verschiedenen Datenbanken und – meist auch – Servern. Zur Bezeichnung der verschiedenen Komponenten eines Replikationsszenarios werden Begriffe aus dem Verlagswesen verwendet:

- Ein *Verleger* (engl. Publisher) stellt Publikationen bereit, die aus einem oder mehreren Artikeln bestehen können. Dabei entsprechen die Artikel einzelnen Tabellen oder Sichten (bzw. Teilen davon, wenn nicht alle Zeilen oder Spalten ausgewählt werden).
- Ein *Verteiler* (engl. Distributor) nimmt die Publikationen entgegen und stellt diese für Replikations-Clients (oder auch Abonnenten genannt) zur Verfügung.
- Ein *Abonnent* (engl. Subscriber) abonniert beliebige Artikel einer Publikation und erhält darüber automatisch die Daten und Änderungen an den Tabellen, die Bestandteil des Artikels sind.

Der Abonnent verbindet sich also nicht direkt mit dem Verleger, sondern immer nur mit dem Verteiler. Allerdings kann eine SQL Server-Instanz gleichzeitig als Verleger und Verteiler fungieren, was für viele kleinere Replikationsszenarios völlig ausreichend ist.

Es gibt verschiedene Arten einer Replikation, die über die Art und Häufigkeit der Datenabgleiche entscheiden:

- *Snapshot-Replikation* – Die Daten des Abonnements werden komplett übernommen. Dabei werden auch auf dem Abonnenten fehlende Tabellen notfalls mit angelegt. Die Snapshot-Replikation wird oft auch als erster Schritt für die Einrichtung der anderen Replikationstypen verwendet.
- *Transaktions-Replikation* – Datenänderungen aus einzelnen Transaktionen werden regelmäßig an die Abonnenten übertragen.
- *Merge-Replikation* – Datenänderungen werden sowohl auf dem Verleger als auch auf dem Abonnenten gesammelt und an die andere Seite der Replikation übertragen. Diese Variante der Replikation wird auch von mobilen Devices unterstützt.

Hintergrundinfo: *Peer-to-Peer-Replikation*

Seit SQL Server 2005 gibt es mit der Peer-to-Peer-Replikation noch eine weitere Variante, die auf der Transaktionsreplikation basiert, sich aber dadurch von dieser darin unterscheidet, dass hierbei jeder Knoten (engl. Node), der an der Replikation beteiligt ist, sowohl die Rolle des Verlegers als auch die Rolle des Abonnenten übernimmt, während ein Verteiler hierbei gar nicht benötigt wird. Stattdessen tauschen die Knoten alle Datenänderungen untereinander aus, es wird also jede Änderung an alle anderen Knoten weitergegeben. Die Peer-to-Peer-Replikation ist allerdings nur für die Enterprise (bzw. Developer) Edition von SQL Server verfügbar, sodass die Express Edition hiervon nicht betroffen ist.

Welche Rolle spielt SQL Server Express bei der Replikation?

Wie bereits erwähnt, kann SQL Server Express lediglich als Replikationsclient – also als Abonnent – verwendet werden. Sowohl für den Verleger als auch für den Verteiler muss eine Standard oder gar Enterprise (bzw. Developer) Edition von SQL Server verwendet werden.

Um ein Abonnement auf einer Express Edition-Instanz einzurichten, klicken Sie im Objekt-Explorer mit der rechten Maustaste auf den Eintrag *Replikation/Lokale Abonnements* und wählen dann die Option *Neue Abonnements*. Dadurch wird der *Assistent für neue Abonnements* aufgerufen, der Sie durch die Erstellung eines neuen Abonnements leitet.

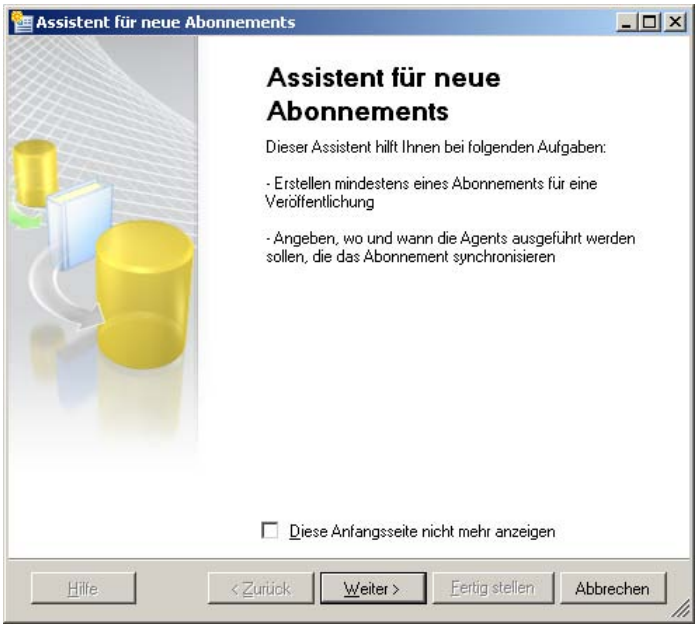


Abbildung 15.2: Der Assistent für neue Abonnements

Sobald ein Abonnement eingerichtet ist, können Sie den Replikationsmonitor aufrufen. Dieser stellt eine grafische Benutzeroberfläche zur Verfügung, mit der Sie prüfen können, ob die Replikation fehlerfrei funktioniert und außerdem einige Informationen zur Performance der Replikation erhalten (z.B. welche Abonnements langsam sind).

Der Replikationsmonitor wird aufgerufen, indem Sie im Objekt-Explorer mit der rechten Maustaste auf *Replikation* klicken und anschließend *Replikationsmonitor starten* auswählen.

15.3 Die SQL Server Compact Edition

Während alle Editionen von SQL Server ab der Express Edition auf derselben Basistechnologie aufbauen, hat die SQL Server Compact Edition hier eine Sonderstellung inne.

Kurzvorstellung SQL Server Compact Edition

Eigentlich ist das »SQL Server« in der Produktbezeichnung auch irreführend, denn genau genommen handelt es sich bei der Compact Edition nicht um einen Datenbankserver, der mit einem eigenen Serverdienst den Zugriff auf Daten bereitstellt, sondern lediglich um ein prozessintern verwendetes Datenbankmodul, das primär für die lokale Speicherung von Daten ausgelegt ist. Vereinfacht dargestellt kann man sagen, dass die SQL Server Compact Edition eigentlich nur aus einem Datenbankdateiformat und einer Programmierschnittstelle, über die man das Dateiformat bearbeiten kann, besteht.

Der Grund hierfür ist in der Herkunft der Compact Edition zu finden. Die SQL Server Compact Edition hat ihren Ursprung in der eher für die Verwendung auf mobilen Geräten (wie Pocket PCs und

Smartphones) ausgelegten SQL Server Mobile Edition. Dies war aber ursprünglich nur auf mobilen Geräten nutzbar, sodass man einen Datenabgleich zu einem großen SQL Server entweder explizit programmieren oder per Replikation lösen musste (SQL Server Mobile kann – wie auch die Express Edition – als Replikations-Client genutzt werden). Mit der Überarbeitung von SQL Server Mobile zu SQL Server Compact Edition wurde dieses Datenbankformat auch für den Desktop-PC nutzbar, sodass man Datenbanken der Compact Edition nun einfach auf einem Desktop-PC erstellen und dann auf ein mobiles Gerät kopieren kann. Aber auch auf dem Desktop-PC sind diese Datenbanken natürlich nutzbar und werden hier meist zum Speichern von lokalen Konfigurationsdaten verwendet. Da kein Datenbankdienst für die Nutzung einer SQL Server Compact-Datenbank erforderlich ist, eignet sich dieses Format auch hervorragend, um Datenbanken zusammen mit selbst entwickelten Anwendungen auszuliefern. Die notwendigen Zugriffsbibliotheken sind dann in Form von .NET-Assemblys in die Anwendung integriert.

Die Kehrseite dieser Einfachheit und Kompaktheit ist, dass dadurch natürlich der Funktionsumfang der Compact Edition stark eingeschränkt ist. So kennt die Compact Edition beispielsweise keine Logins, Datenbankbenutzer oder Rechte. Stattdessen kann die komplette Datenbankdatei mit einem Kennwort geschützt werden. Das macht die Nutzung in Mehrbenutzerszenarios praktisch unmöglich.

Möglichkeiten der Programmierbarkeit wie gespeicherte Prozeduren, Funktionen oder gar Trigger sind nicht gegeben und auch die Anzahl der Datentypen ist eingeschränkt. So gibt es keinen XML-Datentyp und insbesondere die mit SQL Server 2008 neu hinzugekommenen Datentypen (*date*, *time*, *hierarchyid*, *geometry* & *geography*) sind gänzlich unbekannt. Des Weiteren sind interessanterweise ausschließlich Unicode Zeichentypen verfügbar, was wahrscheinlich der engen Kopplung mit der .NET-Technologie zu verdanken ist, die intern auch generell mit Unicode arbeitet.

Tabelle 15.1: Die wichtigsten Unterschiede zwischen SQL Server Compact Edition und Express Edition

	SQL Server Express	SQL Server Compact Edition
nutzbare CPU-Kerne	max. 1 CPU	unbegrenzt
nutzbarer Hauptspeicher	max. 1 GB	unbegrenzt
Arbeitsweise	eigener Serverdienst	läuft prozessintern (als Bestandteil der Anwendung)
Speicherbedarf moderat		sehr gering
	■ 256 MB im Hauptspeicher	■ 5 MB im Hauptspeicher
	■ 1 GB auf Datenträger	■ 2 MB auf Datenträger
SQL-Sprachumfang	volles T-SQL	stark begrenzt
Datentypen	alle von T-SQL (sowie benutzerdefinierte)	nur einige
Datenbankschemas	ja nein	
Programmierbarkeit (gespeicherte Prozeduren, Funktionen, Trigger)	ja	nein
Sicherheitsfeatures	Anmeldungen, Benutzer, Rollen und Rechte	Passwortschutz für Datenbankdateien
Sonstiges		läuft auch auf Windows Mobile

Aber es gibt auch eine ganze Reihe von Gemeinsamkeiten zwischen SQL Server Express und der Compact Edition:

- Beide Produkte sind absolut kostenfrei und per Download über die Microsoft-Website erhältlich
- Beide Produkte sind über SQL Server Management Studio komfortabel zu verwalten
- Die Größe der Datenbankdatei ist bei SQL Server 2008 Express und Compact Edition auf 4 GB pro Datenbank begrenzt; bei SQL Server 2008 R2 Express wurde die Obergrenze auf 10 GB erhöht

Die Compact Edition von SQL Server wird auch automatisch mit SQL Server Express mitinstalliert, sodass wir diese auch ausprobieren können.

Datenbanken mit der Compact Edition erstellen

Das Erstellen einer Datenbank für die Compact Edition ist sehr einfach. Da die Compact Edition ja keinen eigenen Serverdienst verwendet, verbindet man sich im SQL Server Management Studio nicht mit einer SQL Server-Instanz, sondern mit einer Datenbankdatei, welche die Endung *.sdf* trägt.

Erstellen wir nun eine Light-Variante der *MediaBase*-Datenbank für die Compact Edition von SQL Server:

1. Klicken Sie in der Symbolleiste des Objekt-Explorers auf die Schaltfläche *Verbinden*. Dadurch klappt eine kleine Liste auf, welche die Einträge *Datenbankmodul* und *SQL Server Compact* beinhaltet. Da sich der erste Eintrag auf die Verbindung mit einem SQL Server-Dienst bezieht, wählen Sie den zweiten Eintrag und es erscheint das Dialogfeld *Verbindung mit Server herstellen*, das sich nun aber etwas anders präsentiert, als man es von den »richtigen« Datenbanken gewohnt ist.

Abbildung 15.3: Das Dialogfeld zum Herstellen einer Verbindung mit SQL Server Compact

Die Felder *Authentifizierung* und *Anmeldename* sind inaktiv, da die Compact Edition als einziges Sicherheitskonzept den Kennwortschutz für eine Datenbankdatei vorsieht. Anstelle des Servernamens ist der Name der Datenbankdatei auszuwählen. Die voreingestellte Auswahl *(local)* macht im Moment wenig Sinn; wenn Sie die Liste herunterklappen, erscheinen noch die beiden Alternativen *<Neue Datenbank...>* und *<Suche fortsetzen...>*. Über die letztgenannte Option erscheint ein Datei-Suchdialog, mit dem Sie nach einer bestehenden Datenbankdatei suchen können.

2. Klicken Sie nun aber auf die Auswahl <*Neue Datenbank...*>, da bisher ja keine für die Compact Edition gültige Datenbankdatei vorhanden ist. Es erscheint das Dialogfeld zum Erstellen einer neuen Datenbank, das Sie mit den folgenden Werten füllen:

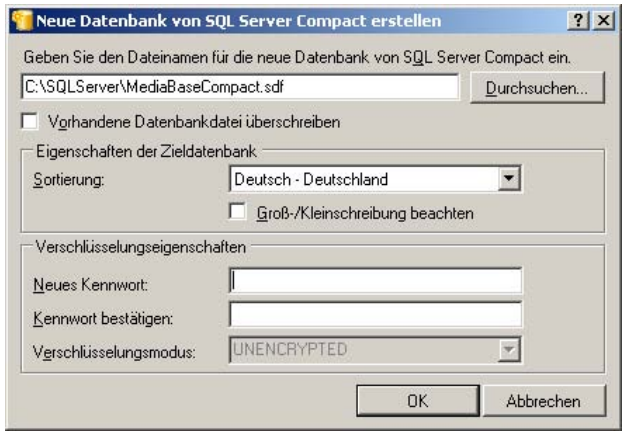


Abbildung 15.4: Erstellen einer neuen SQL Server Compact-Datenbank

Hinweis: SQL Server Compact-Datenbanken schützen

Wenn Sie möchten, können Sie auch ein K ennwort angeben, um die Datenbank zu sichern. Sobald Sie ein Kennwort eingeben, kann auch der Verschlüsselungsmodus ausgewählt werden, wobei Sie z wischen *Platform Default*, *Engine Default* und *PPC2003 Compatibility* wählen können. Die letzte Auswahl ist dann nötig, wenn Si e die Datenbankdatei auch auf älteren Po cket PCs der 2003er-Generation (entspricht Windows CE 4.20) nutzen möchten.

3. Klicken Sie nun auf *OK*, um die Datenbankdatei zu erstellen. Falls Sie kein Kennwort festgelegt haben, erscheint noch eine Sicherheitsabfrage, ob dies wirklich so gewünscht ist, die dann natürlich über die Schaltfläche *Ja* zu bestätigen ist.
4. Nachdem die Datenbankdatei angelegt ist, gelangen Sie wieder in den Anmeldedialog. Dort ist nun aber die gerade erstellte Datenbankdatei bereits ausgewählt und muss nur noch mit einem Klick auf *Verbinden* (sofern Sie ein Kennwort vergeben haben, müssen Sie dieses zuvor noch eingeben) bestätigt werden.
5. Im Objekt-Explorer ist nun auch die gerade erstellte Compact Edition-Datenbank zu sehen. Wenn man die verfügbaren Untereinträge mit denen einer Datenbank aus der Express Edition vergleicht, kann man schnell erahnen, wie sehr der Funktionsumfang der Compact Edition eingeschränkt ist. So sind beispielsweise unter *Programmierbarkeit/Typen* die verfügbaren Datentypen zu sehen, die nur eine Teilmenge der Datentypen ausmachen, die unter SQL Server Express zur Verfügung stehen.

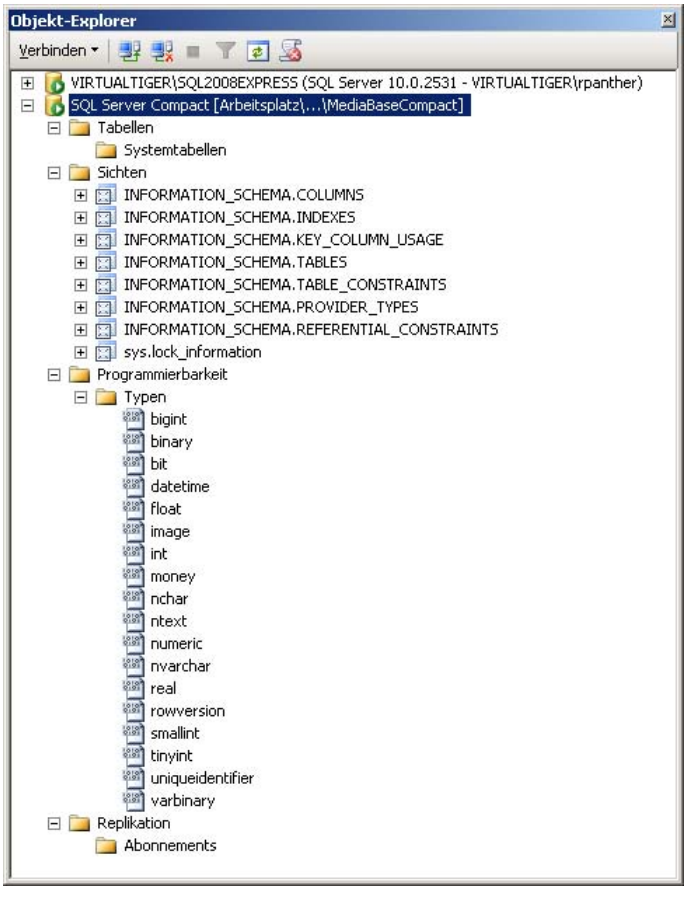


Abbildung 15.5: Die SQL Server Compact Edition-Datenbank im Objekt Explorer

6. Klicken Sie mit der rechten Maustaste auf *Tabellen* und wählen Sie anschließend die Option *Neue Tabelle*. Es erscheint das Dialogfeld zum Erstellen einer neuen Tabelle, das wesentlich überschaubarer ist als das entsprechende Dialogfeld der Express Edition.
7. Geben Sie nun die in Abbildung 15.6 dargestellten Daten ein, um die Tabelle *CD* zu erstellen, und achten Sie dabei auch darauf, für das Feld *ID* die Eigenschaft *Identität* auf *True* zu setzen.

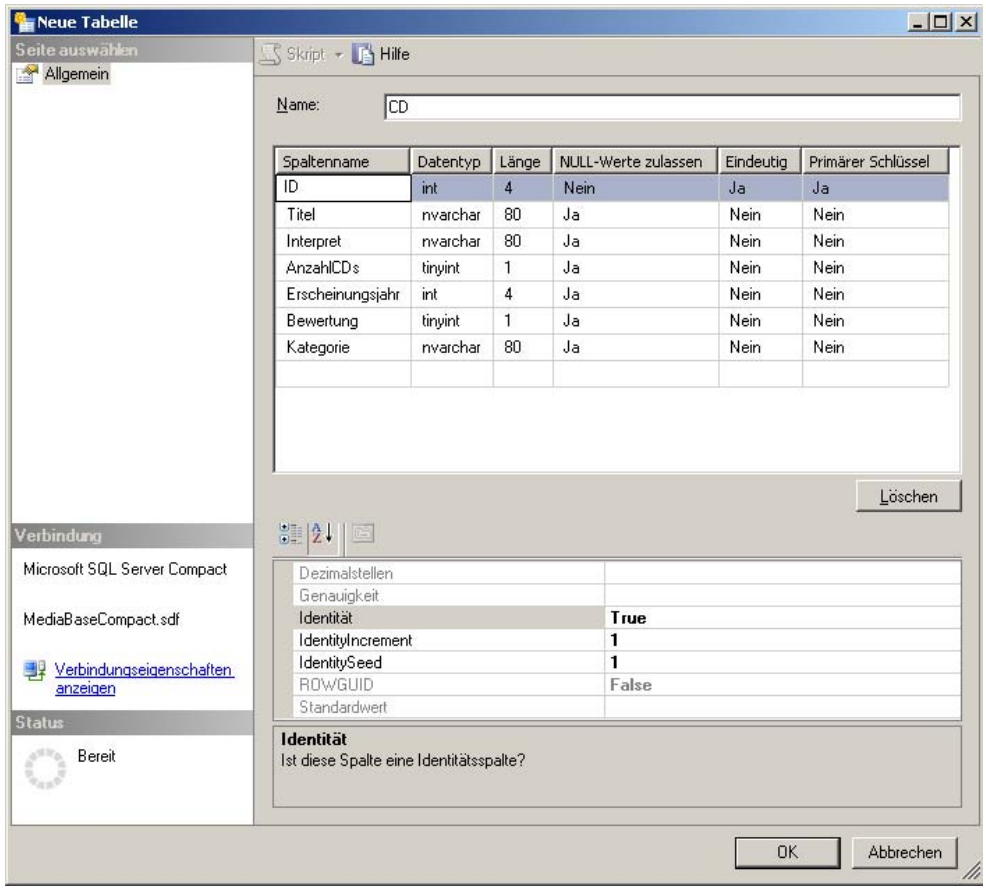


Abbildung 15.6: Das Dialogfeld zum Erstellen einer neuen Tabelle

Wichtig: Keine Schemas verwenden!

Versuchen Sie nicht, für den Tabellennamen ei n Schema a nzugeben. Sie können die Tabelle zwar als *dbo.CD* speichern, werden dann aber beim Versuch, auf diese später zuzugreifen, eine Fehlermeldung erhalten (*Der Tabellenname ist ungültig*).

Dies ist insbesondere wichtig zu wissen, da sich die Tabellennamen bei der Compact Edition nicht ohne Weiteres im Nachhinein wieder ändern lassen.

8. Klicken Sie abschließend auf *OK* und die Tabelle wird erstellt.

Leider bietet die Compact Edition keine Möglichkeit, die Tabellendaten komfortabel anzuzeigen oder gar zu bearbeiten. Stattdessen bleibt nur das Abfragefenster, in dem man die gerade erzeugte Tabelle mit SQL-Anweisungen füllen und anzeigen kann.

Daten zwischen Compact Edition und SQL Server Express austauschen

Wenn Sie eine Anwendung auf Basis von SQL Server Compact entwickelt haben und die Anforderungen an die Datenbank mittlerweile so gewachsen sind, dass ein Umstieg auf eine größere Edition notwendig ist, wäre die Verwendung der Express Edition sicherlich der erste Schritt (zumal der Weg zu den größeren Editionen dann immer noch offen steht).

Problematisch wird es allerdings, die Daten zu übernehmen, da die Compact Edition relativ wenige Schnittstellen nach außen bietet. Für eine Replikation können sowohl Compact Edition als auch Express Edition lediglich als Clients fungieren, sodass auch dieser Ansatz wenig Aussicht auf Erfolg verspricht.

Stattdessen gibt es hier zwei sinnvolle Alternativen, dies zu bewerkstelligen:

- Sie schreiben eine eigene .NET-Anwendung, die sich sowohl mit der Compact Edition-Datenbankdatei als auch mit der Datenbank der Express Edition verbindet und die Daten überträgt. Dies ist sogar generisch machbar, indem Sie in der Compact Edition-Datenbankdatei die Systemsichten *INFORMATION_SCHEMA.TABLES* und *INFORMATION_SCHEMA.COLUMNS* durchlaufen und darüber alle Tabellen der Datenbank kopieren.
- Sie lassen die Dateninhalte aus der Compact Edition über eine SQL SELECT-Abfrage anzeigen, klicken dann mit der rechten Maustaste in den Ergebnisbereich und wählen die Option *Ergebnisse speichern unter* aus. Auf diesem Weg können Sie die Abfrageergebnisse als CSV-Datei speichern, die dann in der Express Edition per `BULK INSERT` wieder eingelesen werden kann.

In beiden Varianten ist zu beachten, dass auf diesem Weg natürlich nur die Rohdaten mitgenommen werden und vorher sowohl die Tabellenstrukturen selbst als auch Indizes etc. manuell erstellt werden müssen.

Schwieriger wird es, wenn Sie Daten von der Express Edition in eine Datenbankdatei der Compact Edition übertragen müssen. Hier bietet sich – neben der oben beschriebenen Variante einer selbst programmierten .NET-Anwendung – die Möglichkeit, die Daten der Express Edition-Datenbank aus einem Abfrageergebnis über die Zwischenablage in ein Excel-Sheet zu kopieren. Dort können sie dann mit etwas Geschick in `INSERT`-Anweisungen verpackt werden, die dann auf der Datenbankdatei der Compact Edition ausgeführt werden können.

15.4 SQL Azure

SQL Azure ist Bestandteil der Microsoft Cloud Computing-Plattform, deren primäres Ziel darin liegt, eine skalierbare, zentral verwaltete und global verfügbare Basis für Anwendungen und Daten zur Verfügung zu stellen. Dazu können Serverdienste inklusive Wartung quasi gemietet werden, die auf verschiedenen Servern in sogenannten Data Centern gehostet werden. Dabei werden alle Daten redundant auf verschiedenen Servern vorgehalten, um somit eine maximale Verfügbarkeit zu gewährleisten. Fällt ein Server aus, so wird dessen Aufgabe automatisch von einem anderen übernommen (Automatic Failover), ohne dass eine Downtime spürbar wäre. Für den Kunden ist diese Lösung damit nahezu wartungsfrei und erfordert nach erfolgreicher Einrichtung keinen weiteren administrativen Aufwand. Werden kurzfristig mehr Ressourcen benötigt, so lassen sich diese dynamisch dazubuchen und anschließend wieder freigeben, sodass nur für das genutzte Volumen gezahlt werden muss. Somit ist auch eine hervorragende Skalierbarkeit gegeben.

Im Falle von SQL Azure ist diese jedoch etwas eingeschränkt, da sich Datenbanken nur in bestimmten Größen buchen lassen (anfangs 1 GB oder 10 GB, inzwischen sind auch 50 GB möglich). Wird mehr Volumen benötigt, so müssen die Daten auf verschiedene Datenbanken aufgeteilt werden, was wiederum Anpassungen in den Anwendungen, die diese Daten nutzen, erforderlich macht. Neben den monatlichen Fixkosten für das verfügbare Datenbankvolumen entstehen noch zusätzliche Datentransferkosten, die von der Menge der übertragenen Daten abhängen.

Ähnlich wie die Compact Edition von SQL Server basiert auch SQL Azure auf denselben Programmiermodellen und Zugriffsprotokollen wie ein »normaler« SQL Server, hat aber an verschiedenen Stellen einige Einschränkungen, die zu beachten sind (die bereits erwähnte maximale Datenbankgröße ist eine davon). Insgesamt ist der Leistungsumfang von SQL Azure jedoch deutlich näher am SQL Server als die SQL Server Compact Edition.

Wenn keine der von SQL Azure nicht unterstützten Features verwendet werden, ist der Aufwand, eine Anwendung auf SQL Azure umzustellen, minimal. Es muss lediglich der Connection String zur Datenbank angepasst werden.

Zusammenspiel zwischen SQL Azure und SQL Server 2008 R2

Auch wenn SQL Azure kein direkter Bestandteil von SQL Server 2008 R2 ist, so wurde es doch etwa zeitgleich eingeführt. Daher verwundert es auch nicht, dass das Management Studio von SQL Server 2008 R2 eine bessere Unterstützung von SQL Azure bietet. Es war zwar auch mit älteren Versionen von Management Studio bereits möglich, sich mit einer SQL Azure-Datenbank zu verbinden, allerdings nur um SQL-Abfragen absetzen zu können.

Mit R2 kann eine SQL Azure-Datenbank nun auch im Objekt-Explorer eingebunden und damit komfortabel verwalten werden. Dabei erfolgen die Angaben zur Verbindung genau wie bei einem normalen Server, wobei lediglich zu beachten ist, dass anstelle von Windows-Authentifizierung nun definitiv SQL Server-Authentifizierung zu verwenden ist, da sich der SQL Azure Server natürlich nicht in einer Domäne befindet, die mit der Ihrer Umgebung vertraut ist.

Im Objekt-Explorer sind nun die folgenden drei Unterbereiche zu sehen:

- *Datenbanken* – In diesem Bereich sind alle eingerichteten Datenbanken zu sehen. Über die Kontextmenüs dieses und der jeweiligen Unterpunkte lassen sich neue Datenbanken einrichten, Datenbankobjekte (wie beispielsweise Tabellen, Sichten, gespeicherte Prozeduren, Funktionen und Datenbankbenutzer) anlegen und verwalten. Dabei gibt es bisher allerdings noch keine Unterstützung durch Assistenten oder spezielle Dialogfelder. Stattdessen werden vorgefertigte Skript-Templates aufgerufen.
- *Sicherheit* – Hier findet sich lediglich der Unterpunkt *Anmeldungen,* mit dem Sie neue Logins zum Zugriff auf Ihre SQL Azure-Datenbank erstellen (oder natürlich auch bestehende anpassen) können.
- Verwaltung – Auch in diesem Bereich gibt es lediglich einen Unterpunkt, nämlich die *Datenebenenanwendungen,* die im folgenden Kapitel ausführlicher behandelt werden.

Um Datenstrukturen in eine SQL Azure-Datenbank zu übertragen, gibt es zwei alternative Wege:

Die klassische Methode liegt darin, per SQL Server Management Studio ein Gesamtskript für die Datenstrukturen zu erstellen und dieses dann auf der SQL Azure-Datenbank auszuführen.

Die modernere Methode wäre die Nutzung einer Datenebenenanwendung (wie ausführlich in *Kapitel 16 – Datenebenenanwendungen* beschrieben). Dazu wird anstelle eines SQL-Skripts ein DAC-Paket erstellt, das die Strukturen der Datenbank beinhaltet. Dieses kann anschließend per SQL Server Management Studio in die SQL Azure-Datenbank eingespielt werden. Der wesentliche Vorteil gegenüber der Skriptvariante liegt darin, dass sich auf diesem Weg auch Aktualisierungen an den Datenstrukturen durchführen lassen. Der Nachteil liegt darin, dass die Quelldatenbank von einem SQL Server 2008 R2 kommen muss, da sich mit älteren Versionen keine DAC-Pakete erstellen lassen.

Sind die Datenstrukturen angelegt, sind eventuell noch Dateninhalte zu übertragen. Auch hierfür lassen sich natürlich SQL-Skripts generieren, die dann einfach auf der SQL Azure-Datenbank ausgeführt werden. Da dies jedoch – je nach Datenvolumen – schnell unhandlich wird, sind hier andere Varianten vorzuziehen. Die (beispielsweise in Kombination mit SQL Server Compact verwendbare) Replikation von Daten wird für SQL Azure leider nicht unterstützt. Stattdessen kann man aber auf das Microsoft Sync Framework zurückgreifen, um Daten zwischen einem klassischen SQL Server und der Cloud auszutauschen.

Als weitere Alternativen gibt es natürlich noch die Möglichkeit, eine kleine .NET-Anwendung zum Übertragen der Daten zu entwickeln, die sich mit beiden Servern verbindet und die Daten von einer Seite liest und auf die andere Seite schreibt.

Komfortabler und eleganter geht dies mit den SQL Server Integration Services oder dem darauf aufbauenden SQL Server Import/Export-Assistenten, der ja bereits für die Express Edition von SQL Server verfügbar ist. Dieser lässt sich dann sogar nutzen, um neben den Daten auch die Datenstrukturen auf die SQL Azure-Datenbank zu bringen. Damit dies funktioniert, sind allerdings ein paar Dinge zu berücksichtigen:

- Wählen Sie als Art des Datenziels den *.Net Framework Data Provider for SqlServer*.
- Bei den Eigenschaften des Datenziels sind folgende Werte zu setzen:
  - *Erweitert/Network Library*: TCP/IP
  - *Quelle/Data Source*: Name des Servers
  - *Quelle/Initial Catalog:* Name der Datenbank
  - *Sicherheit/Password:* Kennwort des SQL Azure Logins
  - *Sicherheit/User ID:* Name des SQL Azure Logins
- Die Zieltabelle muss über einen gruppierten Index verfügen. Sofern die Tabelle in der SQL Azure-Datenbank nicht bereits existierte, müssen Sie die vom Import/Export-Assistenten generierte `CREATE TABLE`-Anweisung manuell anpassen, damit auch der gruppierte Index erstellt wird.

Best Practices: Datenebenenanwendungen und Import-/Export-Assistent

Wenn Sie sich das manuelle Nachbearbeiten der CREATE TABLE-Anweisung sparen wollen, lassen sich die erwähnten Verfahren auch kombinieren, indem Sie per Datenebenenanwendung die benötigten Datenstrukturen anlegen und anschließend per SQL Server-Import/Export-Assistent die Dateninhalte übertragen.

15.5 Umstieg auf eine größere Edition

Im Laufe der Zeit wachsen sowohl die Datenmenge als auch die allgemeinen Anforderungen, sodass es irgendwann sinnvoll sein kann, auf eine größere Edition von SQL Server umzusteigen. Während der Schritt von der Compact Edition auf SQL Server Express noch recht schwierig ist, gestaltet sich ab der Express Edition der Umstieg auf eine größere Edition von SQL Server extrem einfach. Da diese völlig abwärtskompatibel sind, reicht es aus, die Datenbank zu sichern und auf einer größeren Edition wieder einzuspielen. Alternativ zum Sichern und Wiederherstellen können Sie die Datenbank auch trennen und auf dem neuen Server wieder verbinden (wie in Abschnitt *12.1 - Sichern von Datenbankdateien* beschrieben). An den Anwendungen, die auf die Datenbank zugreifen, muss keine Zeile Code geändert werden. Lediglich die Verbindungszeichenfolge (Connection String) muss angepasst werden, sofern Sie einen neuen Server oder eine neue Serverinstanz nutzen. Ob dies nötig ist, hängt davon ab, wo und wie Sie die größere SQL Server Edition installieren.

Hinweis: Developer Edition

Wenn Sie SQL Server für Entwicklungszwecke nutzen, ist als nächstgrößere Edition die Developer Edition zu empfehlen. Diese beinhaltet den maximalen Funktionsumfang (identisch mit der Enterprise Edition), allerdings zum günstigsten Preis (wenn man von den kostenfreien Editionen absieht). Die Lizenz ist allerdings wirklich nur für Entwicklungs- und Testzwecke gültig. Für den Produktiveinsatz wird eine der anderen (teureren) Editionen benötigt.

Wie bei einem Update von einer alten auf eine neuere SQL Server-Version gibt es auch beim Upgrade auf eine der größeren SQL Server-Editionen zwei wesentliche Upgrade-Szenarien, die jeweils andere Vor- und Nachteile bieten:

»Side by Side«-Installation

Genauso wie Sie verschiedene Versionen von SQL Server auf dem Rechner installieren können, die dann über verschiedene Instanznamen ansprechbar sind, lassen sich auch verschiedene Editionen von SQL Server auf einem Rechner installieren. Auch hier erfolgt die Unterscheidung über den jeweiligen Instanznamen, den Sie bei der Installation angeben.

Dadurch, dass dann beide Editionen nicht nur gleichzeitig installiert sind, sondern sogar gleichzeitig laufen, können Sie in Ruhe die Datenbanken, Logins etc. von einer in die andere Instanz übertragen und schließlich bei Bedarf die kleinere Edition deaktivieren oder sogar deinstallieren.

Hinweis: Ein Management Studio für zwei SQL Server

Für das Übertragen der Anmeldungen bietet es sich an, das SQL Server Management Studio der größeren Edition zu verwenden und hier beide SQL Server-Instanzen einzubinden. So haben Sie beide Server im Überblick, ohne ständig zwischen zwei Anwendungen wechseln zu müssen.

Folgende Schritte sollten Sie dabei durchführen:

1. Installation der größeren SQL Server Edition. Es empfiehlt sich, dabei als Instanznamen eine Bezeichnung zu verwenden, aus der auch die Edition hervorgeht (z.B. *SQL2008Standard*).

2. Übertragen der Anmeldungen auf die neue Serverinstanz.
3. Für das Übertragen der Datenbanken haben Sie zwei Möglichkeiten: Entweder Sie sichern die Datenbanken von der kleineren Edition und stellen diese auf der größeren Edition wieder her, oder Sie trennen die Datenbanken und verbinden sie neu mit der größeren Edition. (Beide Varianten werden in *Kapitel 12* ausführlich behandelt.) Die zweite Variante spart natürlich Speicherplatz, da Sie die Datenbanken nicht doppelt auf der Platte haben, dafür sind diese dann mit der kleineren Edition nicht mehr ansprechbar.
4. Zuweisen der Anmeldungen auf die entsprechenden Datenbankbenutzer.
5. Abschalten der SQL Server Express Edition. Dazu können Sie entweder über den SQL Server-Konfigurations-Manager die entsprechenden Dienste beenden und deaktivieren, oder aber Sie deinstallieren die komplette Express Edition. In letzterem Fall sollten Sie generell, falls eine Rückfrage auftaucht, ob evtl. gemeinsam genutzte Dateien entfernt werden dürfen, mit Nein antworten, da die größere SQL Server Edition diese wahrscheinlich noch benötigt.
6. Anpassen der Verbindungszeichenfolgen in Anwendungen, welche die Datenbank benutzen, auf den Namen der neuen Serverinstanz.

Best Practices: Zweistufiges Vorgehen

Wenn man sichergehen will, bietet sich folgendes Vorgehen an, um die Express Edition gefahrlos zu entfernen:

Man deaktiviert die entsprechenden Dienste und wartet einige Zeit, ob dadurch Fehler auftreten. Wenn die größere Edition von SQL Server erwiesenermaßen problemlos funktioniert und sichergestellt ist, dass alle Datenbankobjekte komplett übernommen wurden, kann die Express Edition deinstalliert werden.

»In Place«-Installation

Das zweite Upgrade Szenario ersetzt die kleinere SQL Server Edition (SQL Server Express) komplett mit der größeren. Vorteil dabei ist, dass Sie dann die meisten der oben beschriebenen Schritte sparen können, da hierdurch auch Anmeldungen, verbundene Datenbanken etc. mit in die große Edition übernommen werden. Sogar der Instanzname bleibt derselbe, wodurch in den Anwendungen, die auf die Datenbanken zugreifen, nicht einmal die Verbindungszeichenfolgen angepasst werden müssen.

Nachteilig ist dagegen, dass es dann keinen Weg mehr zurück gibt, da die kleinere SQL Server Edition ja komplett durch die größere ersetzt wurde.

Welchen Weg zum Umstieg auf eine größere SQL Server Edition Sie wählen, bleibt ganz Ihnen überlassen. Als kleine Entscheidungshilfe kann man sagen, dass die »Side by Side«-Installation die bessere Wahl ist, wenn Sie eine geringe Ausfallzeit der Datenbank und eine Möglichkeit zum Zurückgehen auf die Express Edition benötigen. Die »In Place«-Installation ist günstiger, wenn der verfügbare Plattenplatz auf dem Server nicht so groß ist und Sie den Instanznamen beibehalten möchten (damit keine Verbindungszeichenfolgen angepasst werden müssen).

15.6 Übungen zu diesem Kapitel

In diesem Abschnitt finden Sie einige Übungen zu diesem Kapitel. Die richtigen Antworten, Lösungen und den Programmcode finden Sie wie immer auf der Website *www.vsxpress.de.*

Übung 15.1

Erstellen Sie in der Datenbankdatei *MediaBaseCompact.sdf*, die mit der Compact Edition erzeugt wurde, die Tabellen *Buch*, *CDTrack* und *DVD* analog zu ihrer Definition in der SQL Server-Instanz *SQL2008Express*.

Übung 15.2

Übertragen Sie die Daten der Tabelle *dbo.Buch* aus der SQL Server-Instanz *SQL2008Express* in die Datenbankdatei *MediaBaseCompact.sdf*, indem Sie mithilfe von Microsoft Excel entsprechende `INSERT`-Anweisungen generieren.

15.7 Zusammenfassung

In diesem Kapitel haben Sie verschiedene Varianten kennengelernt, wie verschiedene SQL Server miteinander verwendet werden können. Dabei wurde auf die Einrichtung von Verbindungsservern eingegangen, mit denen Sie innerhalb von SQL-Abfragen auf entfernte Server zugreifen können.

Anschließend wurden die verschiedenen Möglichkeiten der Datenbank-Replikation beschrieben, bei denen die Express Edition von SQL Server (ebenso wie die Compact Edition) als Client fungieren kann. Auf diesem Weg können Publikationen, die aus Tabellen (oder Teilen davon) bestehen und von einem Replikationsverteiler zur Verfügung gestellt werden, abonniert werden, sodass Datenänderungen daran automatisch weitergegeben werden.

Der größte Teil des Kapitels widmete sich der Compact Edition, die – durch Beschränkung auf ein notwendiges Minimum an Funktionalität – extrem niedrigen Speicherbedarf erreicht. Die Datenbanken – oder genauer gesagt Datenbankdateien – der Compact Edition können sowohl auf dem Desktop-PC als auch auf einem mobilen Gerät auf Basis von Windows Mobile genutzt werden.

Neu hinzugekommen und vor allem im Zusammenspiel mit SQL Server 2008 R2 komfortabel einsetzbar ist die Möglichkeit, Daten mit SQL Azure in der Cloud global und skalierbar verfügbar zu machen. Dabei kann man zum Verteilen der Datenstrukturen die neu eingeführten Datenebenenanwendungen verwenden, während sich zum Übertragen der Dateninhalte die SQL Server Integration Services bzw. der SQL Server Import/Export-Assistent anbieten.

Zum Schluss wurde das Upgrade auf größere Editionen von SQL Server beschrieben, das nötig ist, falls die Leistungsfähigkeit der Express Edition einmal nicht mehr ausreichen sollte. Hier wurde auf die beiden gängigen Vorgehensweisen für die Installation der neuen Edition eingegangen:

- »Side by Side«-Installation: Die größere SQL Server-Edition wird als zusätzliche Serverinstanz in ein anderes Verzeichnis installiert.
- »In Place«-Installation: Die größere SQL Server-Edition überschreibt die bestehende SQL Server Express-Instanz.

Kapitel 16

Datenebenenanwendungen

In diesem Kapitel lernen Sie

- was Datenebenenanwendungen sind und wie man sie einsetzen kann
- wie Sie neue Datenebenenanwendungen erstellen
- wie Datenebenenanwendungen verteilt werden

16.1 Überblick über Datenebenenanwendungen

Eine der wichtigsten Neuerungen für SQL Server 2008 R2 Express ist die Einführung von sogenannten *Datenebenenanwendungen*. Der etwas holprige Begriff entsprang aus einer Übersetzung der Originalbezeichnung *Data Tier Applications*, womit eigentlich nichts anderes als Datenbankanwendungen gemeint sind. Genau genommen geht es allerdings weniger um die Anwendung selbst, als mehr um die dafür benötigten Datenstrukturen. Dabei soll mithilfe von Datenebenenanwendungen die Verwaltung und insbesondere das Verteilen dieser Datenstrukturen (und von Änderungen daran) erleichtert werden.

In der Database Edition von Visual Studio 2008 gab es bereits einen vergleichbaren Ansatz, indem Datenstrukturen in entsprechenden Datenbankprojekten verwaltet werden konnten. Dabei wurde für jedes einzelne Datenbankobjekt (egal ob Tabelle, Sicht, Index oder gespeicherte Prozedur) eine separate SQL-Skriptdatei vorgehalten, die auch nachträglich bearbeitet werden kann. Selbst eigene SQL-Skripts können dem Projekt hinzugefügt werden. Und auch die Datenbank- und Server-Konfigurationseinstellungen ließen sich in den Projekten unterbringen, wobei dazu XML-Dateien genutzt wurden.

Um die so verwalteten Datenbankstrukturen auf echte Datenbanken zu verteilen, wurden verschiedene Möglichkeiten angeboten. Angefangen von Schemavergleichen zwischen Projekt und Datenbank bis hin zu automatischer Bereitstellung beim »Übersetzen« des Projekts. In beiden Fällen wurden Differenzskripte erzeugt, mit denen die geänderten und fehlenden Datenbankobjekte auf der Zieldatenbank angepasst wurden. Aufseiten von SQL Server wurden also nur SQL-Skripte ausgeführt, eine spezielle Unterstützung für diese Datenbankprojekte war nicht erforderlich.

Die Datenebenenanwendungen gehen hier noch einen Schritt weiter. Aufseiten von Visual Studio hat sich an Struktur und Aufbau der Datenbankprojekte zwar wenig geändert, aber nun werden die Datenebenenanwendungen auch aufseiten von SQL Server als solche registriert und sind damit noch besser verwaltbar. Mit dem SQL Server Management Studio lassen sich ab SQL Server 2008 R2 sogar Datenebenenanwendungen auf Basis einer existierenden Datenbank erstellen, ohne dass dafür Visual Studio benötigt wird. Somit kann man eine Datenbank direkt mit den gewohnten Datenbanktools in Ruhe vorbereiten und anschließend in Form einer Datenebenenanwendung auf andere Server verteilen.

Im Gegensatz zu den Datenbankprojekten der Visual Studio Database Edition, deren kompilierte Form aus XML-Dateien mit der Endung *.dbschema* bestand, nutzen die Datenebenenanwendung sogenannte Pakete mit der Endung *.dacpac*, die eigentlich nichts anderes als im ZIP-Format komprimierte Dateien sind. Diese Dateien werden auch als *DAC-Paket* bezeichnet, wobei DAC eine etwas gewöhnungsbedürftige Abkürzung für **D**ata Tier **A**ppli**c**ation darstellt.

16.2 Erstellen von Datenebenenanwendungen

Zum Erstellen einer Datenebenenanwendung gibt es verschiedene Wege. So kann man sie mit Visual Studio durch Auswahl des passenden Projekttyps erstellen. In diesem Projekt sind dann die einzelnen Datenbankobjekte in Form von separaten Skriptdateien enthalten. Wenn man diese Skriptdateien nicht selbst erstellen will, kann man entweder einen Schemavergleich mit einer bestehenden Datenbank durchführen oder aber eine Skriptdatei importieren, die CREATE-Anweisungen für die gewünschten Datenbankobjekte enthält. Dieses Skript lässt sich wiederum recht einfach mit dem SQL Server Management Studio auf Basis einer bestehenden Datenbank erstellen.

Hinweis: Datenebenenanwendungen und Visual Studio 2010

Um mit Datenebenenanwendungen zu arbeiten, ist Visual Studio 2010 Premium oder Ultimate erforderlich. Mit Einschränkun gen (kein Schemave rgleich) kann auch die Professional Edit ion verwendet werden. Da selb st diese Edition nicht ganz günstig ist und sicherlich nicht jeder Leser dieses Bu ches über eine entsprechende Vi sual Studio -Installation verfügt, beschränken sich die folgenden Beschreibungen auf die Möglichkeiten, Datenebenen anwendungen mit SQL Server-Bordmitteln zu nutzen.

Extrahieren von Datenebenenanwendungen

Wenn die Grundlage für eine Datenebenenanwendung ohnehin eine bestehende Datenbank ist, bietet es sich natürlich auch an, diese direkt im SQL Server Management Studio erstellen zu können. Dies ist mit SQL Server 2008 R2 nun möglich und kann anhand der folgenden Anleitung ausprobiert werden.

Um einen Verbindungsserver einzurichten, sind folgende Schritte durchzuführen:

1. Stellen Sie im SQL Server Management Studio eine Verbindung zur lokalen Serverinstanz *SQL2008Express* her.
2. Klicken Sie im Objekt-Explorer mit der rechten Maustaste auf *Datenbanken/MediaBase* und wählen Sie die Option *Tasks/Datenebenenanwendung extrahieren*. Es erscheint der Assistent zum Extrahieren von Datenebenenanwendungen.

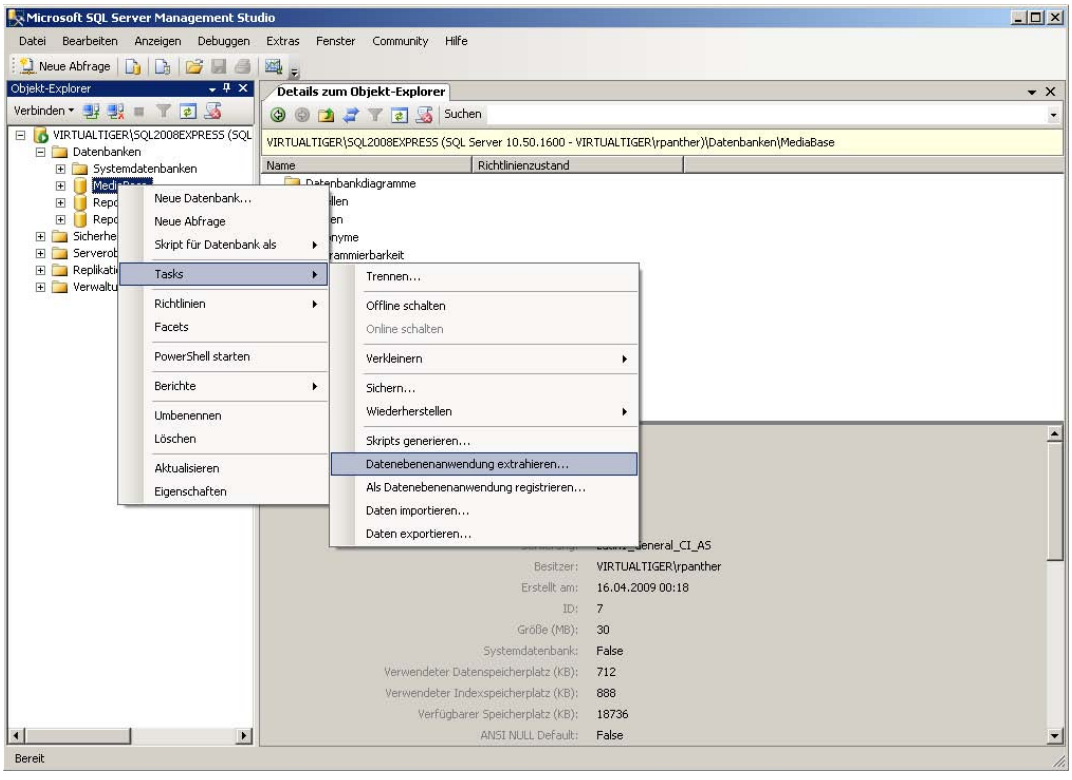


Abbildung 16.1: Aufruf des Assistenten zum Extrahieren von Datenebenenanwendungen

3. Im Schritt *Eigenschaften festlegen* können Sie einen Namen und eine Versionsnummer für die Datenebenenanwendung definieren sowie einen Kommentar angeben, der die Anwendung beschreibt. Dazu ist der Pfad anzugeben, in dem die Datei mit der Endung *.dacpac* abgelegt werden soll.

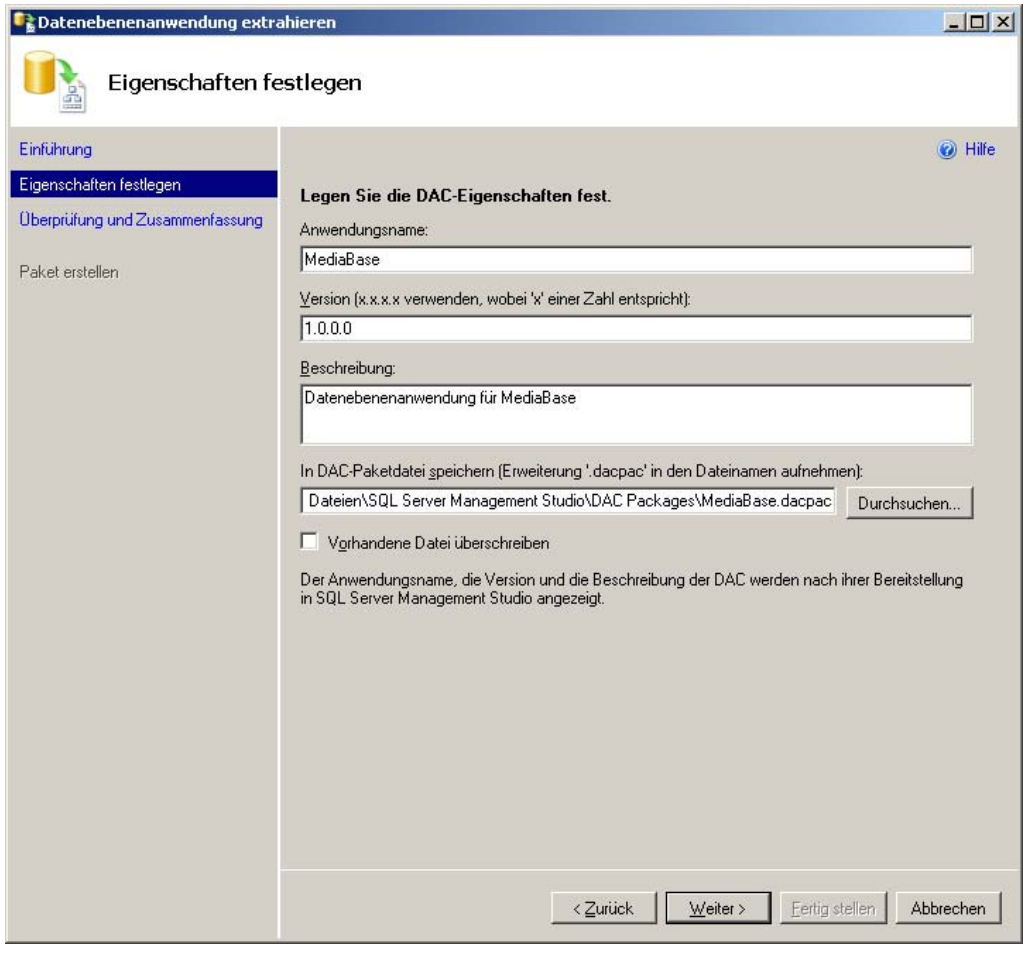


Abbildung 16.2: Definieren der Eigenschaften für eine Datenebenenanwendung

4. Im nächsten Schritt folgt eine Zusammenfassung der vorgenommenen Einstellungen. Dabei wird auch überprüft, ob alle in der Datenbank vorhandenen Objekte von Datenebenenanwendungen unterstützt werden.

Hinweis: Nicht unterstützte Datenbankobjekttypen

Leider werden von Datenebenenanwendungen noch nicht alle Objekttypen unterstützt. Zu den nicht unterstützten Objekttypen gehören vor allem DDL-Trigger, Service Broker und Volltextkataloge. Sollten Sie versuchen, eine Datenebenenanwendung für eine Datenbank zu erstellen, die nicht unterstützte Objekttypen beinhaltet, werden Sie vom Assistenten zum Extrahieren von Datenebenenanwendungen darauf hingewiesen.

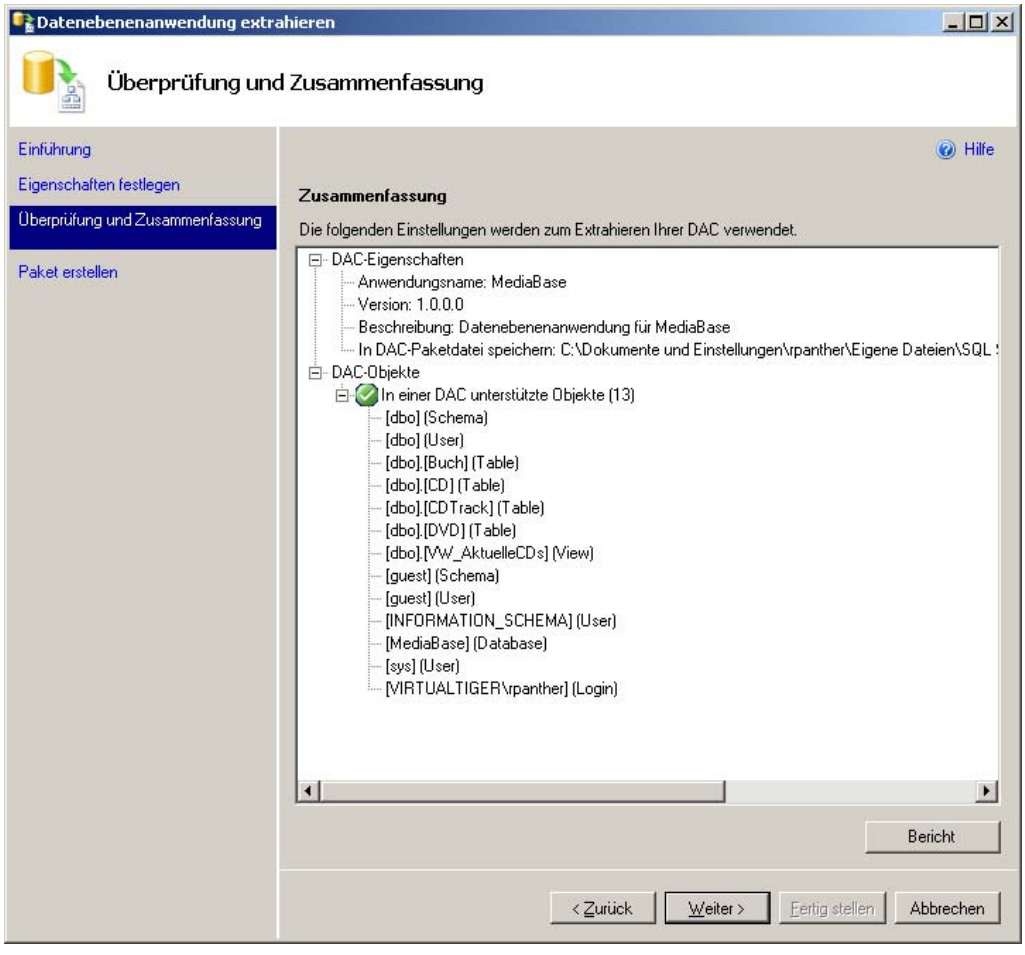


Abbildung 16.3: Zusammenfassung und Überprüfung der verwendeten Objekte

5. Sofern es keine Probleme bei den verwendeten Datenbankobjekten gibt, können Sie auf die Schaltfläche *Weiter* klicken, wodurch das DAC-Paket erstellt wird – und fertig ist Ihre erste Datenebenenanwendung.

Registrieren von Datenebenenanwendungen

Mit dem gerade beschriebenen Verfahren haben Sie eine Datenebenenanwendung auf Basis einer Datenbank erstellt, diese aber noch nicht im Server registriert. Unter *Verwaltung/Datenebenenanwendungen* finden Sie im Objekt-Explorer eine Auflistung der registrierten Datenebenenanwendungen, die jetzt noch leer sein dürfte. Um dies zu ändern, können Sie die Datenebenenanwendung entweder – wie weiter unten beschrieben – importieren oder aber eine bereits vorhandene Datenbank als Datenebenenanwendung registrieren – was in diesem Fall mehr Sinn macht, weil die Datenstrukturen in der vorhandenen Datenbank bereits auf dem aktuellen Stand sind.

Und so registrieren Sie eine vorhandene Datenbank als Datenebenenanwendung:

1. Stellen Sie im SQL Server Management Studio eine Verbindung zur lokalen Serverinstanz *SQL2008Express* her.
2. Klicken Sie im Objekt-Explorer mit der rechten Maustaste auf *Datenbanken/MediaBase* und wählen Sie die Option *Tasks/Datenebenenanwendung registrieren*. Der Assistent zum Registrieren von Datenebenenanwendungen wird geöffnet.
3. Auch hier sind Anwendungsname, Version und Beschreibung anzugeben, bevor die Datenebenenanwendung nach einer Zusammenfassung der Einstellungen registriert wird.

Wenn Sie nun noch einmal unter dem Knoten *Verwaltung* nachsehen, sollte dort auch die neue Datenebenenanwendung erscheinen.

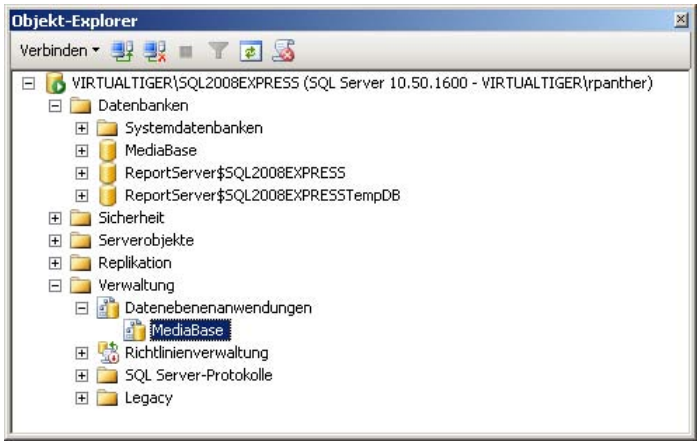


Abbildung 16.4: Anzeige der registrierten Datenebenenanwendungen

16.3 Verteilen von Datenebenenanwendungen

Der hauptsächliche Nutzen von Datenebenenanwendungen liegt natürlich nicht darin, sie auf einem Server zu verwalten, sondern auf andere Zielserver ausrollen zu können. In der Praxis kann dies verwendet werden, um die Datenstrukturen einer Anwendung auf verschiedene Server zu verteilen oder aber um aktualisierte Stände dieser Strukturen von einer Entwicklungsumgebung auf eine Testumgebung oder später auch Produktionsumgebung zu übertragen.

Dazu müssen Sie lediglich das DAC-Paket (also die Datei mit der Endung *.dacpac*) auf dem entsprechenden Zielserver einspielen.

Importieren von Datenebenenanwendungen

Zum Importieren von Datenebenenanwendungen gibt es verschiedene Wege. Im einfachsten Fall kopieren Sie die Datei auf den entsprechenden Zielserver und rufen sie im Windows-Explorer mit einem Doppelklick auf. Daraufhin wird der entsprechende Assistent zum Importieren von Datenebenenanwendungen geöffnet.

Alternativ können Sie dies auch direkt aus dem SQL Server Management Studio erreichen:

1. Stellen Sie im SQL Server Management Studio eine Verbindung zur lokalen Serverinstanz *SQL2008Express* her.
2. Klicken Sie im Objekt-Explorer mit der rechten Maustaste auf *Verwaltung/Datenebenenanwendungen* und wählen Sie die Option *Datenebenenanwendung bereitstellen* aus. Es erscheint ebenfalls der Assistent zum Bereitstellen von Datenebenenanwendungen.
3. Die einzige Entscheidung, die Sie in diesem Assistenten treffen müssen, ist die Auswahl des richtigen DAC-Pakets, alles andere geschieht automatisch. Anhand der bereits registrierten Datenebenenanwendungen wird sogar entschieden, ob ein neues DAC-Paket installiert oder ein bestehendes Paket aktualisiert werden soll.

Aktualisieren von Datenebenenanwendungen

Ein weiterer großer Vorteil, den Datenebenenanwendungen bieten, ist die Möglichkeit, eine bestehende Datenbank zu aktualisieren, um beispielsweise die Datenstrukturänderungen, die mit einer neuen Version der Anwendung erforderlich werden, zu verteilen.

Um dies sinnvoll ausprobieren zu können, verändern wir nun die bestehende Datenbank, indem eine Sicht sowie zwei Spalten aus einer Tabelle entfernt werden:

1. Stellen Sie im SQL Server Management Studio eine Verbindung zur lokalen Serverinstanz *SQL2008Express* her.
2. Löschen sie mit dem SQL Server Management Studio die Sicht *VW_AktuelleCDs*.
3. Öffnen Sie die Tabelle *dbo.DVD* im Bearbeitungsmodus und löschen Sie dort die Spalten *Land* und *Laendercode*.

Wenn die Datenbank nun mit der zuvor gespeicherten Datenebenenanwendung aktualisiert wird, werden die gerade gelöschten Datenbankobjekte wieder neu angelegt. Zum Aktualisieren, können Sie entweder wie weiter oben beschrieben vorgehen und das DAC-Paket importieren, wobei automatisch erkannt wird, dass die Datenebenenanwendung bereits registriert ist und daher aktualisiert werden muss. Alternativ dazu können Sie auch direkt die zu aktualisierende Datenebenenanwendung auswählen:

1. Klicken Sie im Objekt-Explorer mit der rechten Maustaste auf *Verwaltung/Datenebenenanwendungen/MediaBase* und wählen Sie die Option *Datenebenenanwendung aktualisieren* aus. Es erscheint der Assistent zum Aktualisieren von Datenebenenanwendungen.
2. Auch hier ist vor allem die zu aktualisierende DAC-Paketdatei auszuwählen.
3. Der Assistent erkennt nun, dass an der zu aktualisierenden Datenbank Änderungen vorgenommen wurden, die nicht von einer Datenebenenanwendung stammen. Daher müssen Sie durch Anklicken eines Kontrollkästchens bestätigen, dass diese Änderungen verloren gehen dürfen, bevor es weitergeht.

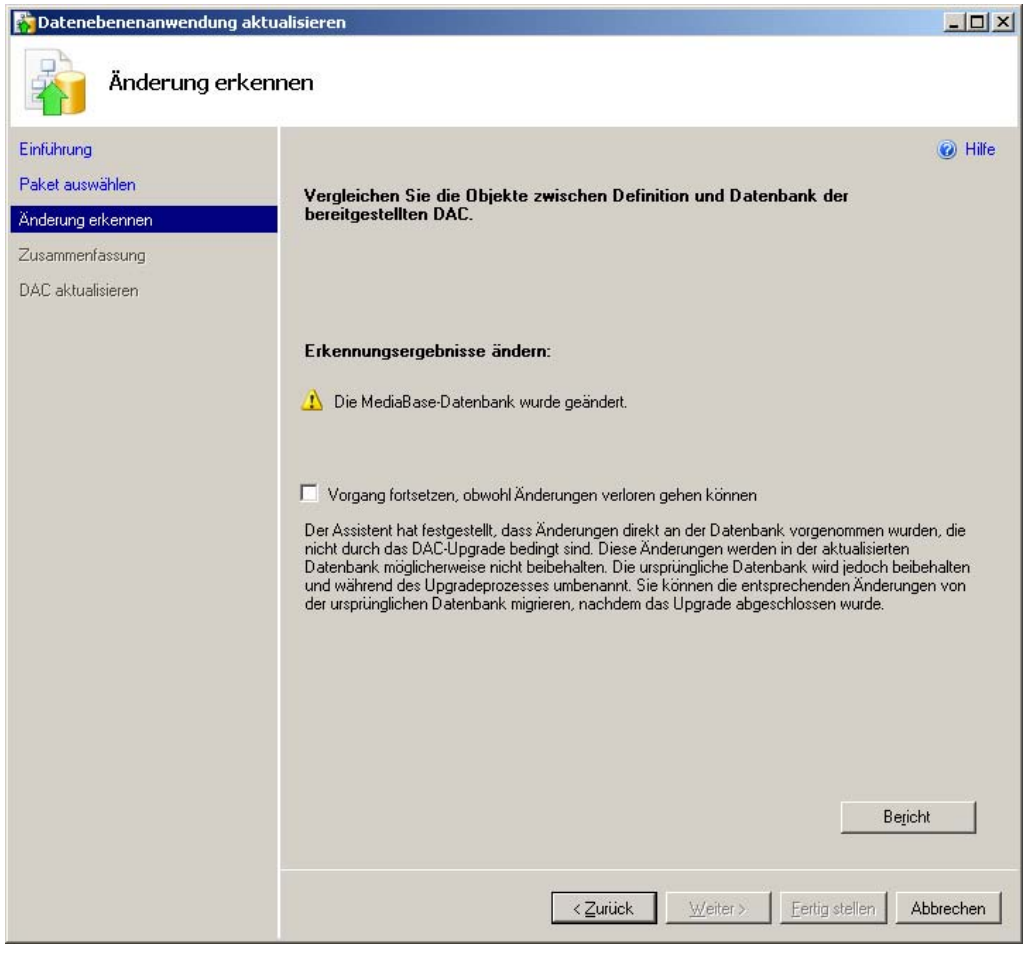


Abbildung 16.5: Der Assistent zum Aktualisieren von Datenebenenanwendungen

4. Zum Schluss erfolgt die übliche Zusammenfassung, bevor die Datenbank aktualisiert wird. Dabei werden existierende Verbindungen zur Datenbank vorher getrennt. Die Daten in den Tabellen bleiben jedoch erhalten, sofern dies möglich ist.

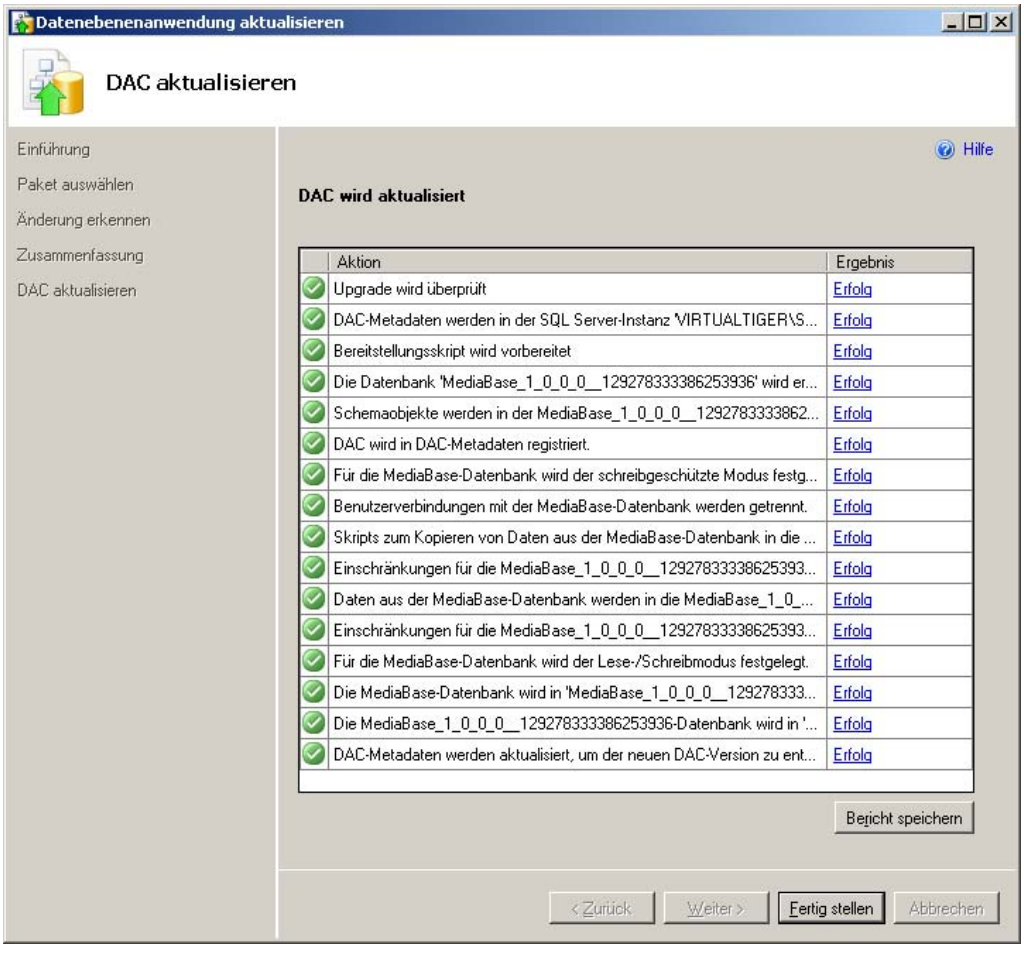


Abbildung 16.6: Die Fortschritts- bzw. Ergebnisanzeige beim Aktualisieren von Datenebenenanwendungen

5. Öffnen Sie nun im Objekt-Explorer den Zweig *MediaBase/Sichten* und vergewissern Sie sich, dass dort die Sicht *VW_AktuelleCDs* wieder aufgetaucht ist.
6. Öffnen Sie den Zweig *MediaBase/Tabellen/dbo.DVD/Spalten* und vergewissern Sie sich, dass dort die Spalten *Land* und *Laendercode* wieder aufgetaucht sind.

Hinweis: Datenbank vor/nach einer Aktualisierung

Beim Aktualisieren der Datenebenenanwendung wird allerdings nicht die bestehende Datenbank geändert, sondern lediglich eine Kopie davon. Die alte Datenbank ist mit den ursprünglichen Datenbankdateien, aber einem neuen Datenbanknamen (z.B. *MediaBase_1_0_0_0__129278333394265456*) weiterhin verfügbar, während die geänderte Datenbank zwar den alten Datenbanknamen (also *MediaBase*) trägt, aber dabei modifizierte Dateinamen (z.B. *MediaBase_1_0_0_0__129278333386253936.mdf* und *MediaBase_1_0_0_0__129278333386253936_log.ldf*) verwendet.

Löschen von Datenebenenanwendungen

In demselben Kontextmenü, mit dem Sie eine Datenebenenanwendung aktualisieren können, finden Sie auch einen Eintrag zum Löschen von Datenebenenanwendungen. Damit wird jedoch nicht die Datenbank selbst gelöscht, sondern lediglich die Registrierung der Datenbank als Datenebenenanwendung.

16.4 Übungen zu diesem Kapitel

In diesem Abschnitt finden Sie einige Übungen zu diesem Kapitel. Die richtigen Antworten, Lösungen und den Programmcode finden Sie wie immer auf der Website *www.vsxpress.de.*

Übung 16.1

Erstellen Sie eine Datenebenenanwendung auf Basis einer beliebigen Datenbank.

Übung 16.2

Importieren Sie die Datenebenenanwendung auf einem anderen Server (sofern verfügbar) und prüfen Sie anschließend im Bereich *Verwaltung,* ob die Datenebenenanwendung korrekt registriert wurde.

Übung 16.3

Löschen Sie die Datenebenenanwendung und vergewissern Sie sich anschließend, dass die Datenebenenanwendung nicht mehr registriert ist, die Datenbank aber noch vorhanden ist.

16.5 Zusammenfassung

Die mit SQL Server 2008 R2 eingeführten Datenebenenanwendungen sind bestens geeignet, um Datenstrukturen einer Anwendung zu verwalten und auf verschiedene Server zu verteilen. Dabei werden Import, Export, Registrierung und Aktualisierung von Datenebenenanwendungen im SQL Server Management Studio durch entsprechende Assistenten unterstützt.

In Kombination mit Visual Studio 2010 (ab Professional) lassen sich Datenebenenanwendungen noch effektiver einsetzen, da sie hier in Form von Datenbankprojekten verwaltet und damit auch in Quellcodeverwaltungssysteme wie beispielsweise Team Foundation Server eingecheckt werden können.

Damit sind wir – abgesehen von den folgenden Anhängen – am Ende des Buchtextes angelangt. Ich hoffe, Sie konnten einige interessante Informationen zur Express Edition von SQL Server mitnehmen und wünsche Ihnen viel Freude und Erfolg bei der Arbeit mit diesem interessanten Produkt.

Robert Panther

Anhang A

Kleine SQL-Referenz

In diesem Kapitel finden Sie eine kleine Referenz der wichtigsten SQL-Kommandos. Im Gegensatz zu vielen anderen Sprachreferenzen will ich hier aber keine unübersichtlichen Syntaxdiagramme mit allen noch so selten benötigten Optionen auflisten. Diese Informationen können Sie bei Bedarf leicht über die Online-Hilfe von SQL Server erhalten.

Auf der anderen Seite sind in dieser Referenz auch keine ausführlichen Erklärungen zu finden, denn dafür ist ja der eigentliche Buchtext gedacht.

Stattdessen finden Sie in dieser Referenz für die gängigsten SQL-Kommandos die gängigsten Varianten jeweils in einer vereinfachten Syntaxdarstellung sowie in Form eines konkreten Beispiels aufgelistet.

Bei der Syntaxdarstellung sind die von eckigen Klammern (z.B. `[Beispiel]`) umfassten Begriffe jeweils durch den mit dem Begriff beschriebenen Inhalt zu ersetzen.

A.1 SELECT

Einfache Abfragen

Grundform

Syntax:

```
SELECT [Feldliste]
FROM [Tabelle]
```

Beispiel:

```
SELECT *
FROM Buch
```

Einschränkungen mit WHERE

Syntax:

```
SELECT [Feldliste]
FROM [Tabelle]
WHERE [Filterbedingungen]
```

Beispiel:

```
SELECT  *
FROM dbo.CD
WHERE Erscheinungsjahr = 2009 AND Interpret LIKE 'Best Of%'
```

Sortierung mit ORDER BY

Syntax:

```
SELECT [Feldliste]
FROM [Tabellen]
ORDER BY [Feldliste]
```

Beispiel:

```
SELECT  *
FROM dbo.Buch
ORDER BY Autor ASC, Erscheinungsjahr DESC
```

Gruppierung mit GROUP BY

Syntax:

```
SELECT [Feldliste]
FROM [Tabelle]
GROUP BY [Feldliste]
HAVING [Filterbedingungen]
```

Beispiel:

```
SELECT COUNT(*) as Anzahl, Erscheinungsjahr
FROM Buch
GROUP BY Erscheinungsjahr
HAVING count(*)>2
```

Komplexere Abfragen

Fallunterscheidung mit CASE

Syntax:

```
SELECT [Feldliste] , CASE WHEN [Bedingung] THEN [Wert1] ELSE [Wert1] END AS [Alias]
FROM [Tabelle]
```

Beispiel:

```
SELECT *, CASE
                    WHEN Hardcover=1 THEN 'Hardcover'
  ELSE                   'Softcover'
         END AS Cover
FROM dbo.Buch
```

Aggregierungsfunktionen mit COMPUTE

Syntax:

```
SELECT [Feldliste]
FROM [Tabelle]
COMPUTE [Aggregierungen]
```

Beispiel:

```
SELECT *
FROM dbo.CD
COMPUTE count(ID)
```

Abfragen auf mehreren Tabellen

Verknüpfen mit JOIN

Syntax:

```
SELECT [Feldliste]
FROM [Tabelle] INNER|LEFT|RIGHT JOIN [Tabelle] ON [Verknüpfungsbedingungen]
```

Beispiel:

```
SELECT CD.Titel, CDTrack.*
FROM CD INNER JOIN CDTrack ON CD.ID = CDTrack.idCD
```

Verbinden mit UNION SELECT

Syntax:

```
SELECT [Feldliste]
FROM [Tabelle1]
UNION
SELECT [Feldliste]
FROM [Tabelle2]
```

Beispiel:

```
SELECT Titel, Interpret, Erscheinungsjahr FROM CD
UNION SELECT Titel, Autor, Erscheinungsjahr FROM Buch
UNION SELECT Titel, Produzent, Erscheinungsjahr FROM DVD
```

Unterabfragen

Unterabfragen mit IN

Syntax:

```
SELECT [Feldliste]
FROM [Tabelle]
WHERE [Feld] IN ([SELECT-Abfrage])
```

Beispiel:

```
SELECT Titel, Interpret
FROM dbo.CD
WHERE ID IN
 (SEL         ECT idCD
 FROM          dbo.CDTrack
          GROUP BY idCD
 HAVI         NG COUNT(*)>13)
```

Unterabfragen mit EXISTS

Syntax:

```
SELECT [Feldliste]
FROM [Tabelle]
WHERE EXISTS ([SELECT-Abfrage])
```

Beispiel:

```
SELECT Titel, Interpret
FROM dbo.CD
WHERE EXISTS
 (SEL         ECT idCD
 FROM          dbo.CDTrack
          WHERE idCD = dbo.CD.ID
          GROUP BY idCD
 HAVI         NG COUNT(*)>13)
```

Unterabfragen als berechnetes Feld

Syntax:

```
SELECT [Feldliste], [SELECT-Abfrage]
FROM [Tabelle]
```

Beispiel:

```
SELECT Titel, Interpret,
 (SEL         ECT count(*)
 FROM          dbo.CDTrack
          WHERE idCD = dbo.CD.ID) AS Anzahl
FROM dbo.CD
```

A.2 Data Manipulation Language (DML)

UPDATE

Einfaches UPDATE

Syntax:

```
UPDATE Tabelle
SET [Zuweisung]
WHERE [Bedingung]
```

Beispiel:

```
UPDATE MeineBuecher
SET Verlag = '(unbekannt)'
WHERE Verlag IS NULL
```

UPDATE auf Basis von mehreren Tabellen

Syntax:

```
UPDATE [UpdateTabelle]
SET [Zuweisungen]
FROM [UpdateTabelle] INNER JOIN [WeitereTabelle] ON [JOIN-Bedingung]
WHERE [Filterbedingungen]
```

Beispiel:

```
UPDATE dbo.CDTrack
SET CDTrack.Bewertung=CD.Bewertung
FROM dbo.CD
          INNER JOIN dbo.CDTrack ON CD.ID=CDTrack.idCD
WHERE CDTrack.Bewertung IS NULL
```

INSERT/SELECT INTO

INSERT mit konkreten Werten

Syntax:

```
INSERT INTO [Tabelle] ([Felder])
VALUES ([Daten])
```

Beispiel:

```
INSERT INTO Buch (Autor, Titel, Hardcover)
VALUES ('Robert Panther', 'Programmieren mit dem .NET Compact Framework', 1)
```

INSERT auf einer Abfrage basierend

Syntax:

```
INSERT INTO [Tabelle] ([Felder])
[SELECT-Abfrage]
```

Beispiel:

```
INSERT INTO CD (Titel)
SELECT Titel FROM DVD
```

SELECT INTO

Syntax:

```
SELECT [Felder]
INTO [Zieltabelle]
FROM [Quelltabelle]
WHERE [Bedingung]
```

Beispiel:

```
SELECT *
INTO MeineBuecher
FROM Buch
WHERE Autor = 'Robert Panther'
```

DELETE/TRUNCATE TABLE

DELETE mit Bedingung

Syntax:

```
DELETE FROM [Tabelle]
```

```
WHERE [Bedingung]
```

Beispiel:

```
DELETE FROM MeineBuecher
WHERE Hardcover=1
```

TRUNCATE TABLE

Syntax:

```
TRUNCATE TABLE [Tabelle]
```

Beispiel:

```
TRUNCATE TABLE MeineBuecher
```

DELETE auf Basis von mehreren Tabellen

Syntax:

```
DELETE [LoeschTabelle]
FROM [LoeschTabelle] INNER JOIN [WeitereTabelle] ON [JOIN-Bedingung]
WHERE [Filterbedingungen]
```

Beispiel:

```
DELETE dbo.CD2
FROM dbo.CD2 LEFT JOIN dbo.CDTrack ON CD2.ID = CDTrack.idCD
WHERE CDTrack.idCD IS NULL
```

MERGE

Syntax:

```
MERGE INTO [Zieltabelle]
USING [Quelltabelle] ON [JOIN-Bedingung]
WHEN MATCHED
          THEN UPDATE SET [Zuweisungen]
WHEN NOT MATCHED
          THEN INSERT ([Feldliste])
 VALU        ES ([Werte])
WHEN NOT MATCHED BY SOURCE THEN
 DELE        TE;
```

Beispiel:

```
MERGE INTO dbo.Buch
USING BuchImport ON Buch.Titel=BuchImport.Titel
WHEN MATCHED
          THEN UPDATE SET Buch.Autor = BuchImport.Autor,
  Buch                  .Titel = BuchImport.Titel,
  Buch                  .Kategorie = BuchImport.Kategorie
WHEN NOT MATCHED
          THEN INSERT (Autor, Titel, Kategorie)
          VALUES (BuchImport.Autor, BuchImport.Titel, BuchImport.Kategorie)
WHEN NOT MATCHED BY SOURCE THEN
 DELE        TE;
```

A.3 Data Definition Language (DDL)

Datenbanken erstellen und konfigurieren

Datenbanken erstellen

Syntax:

```
CREATE DATABASE [Datenbankname]
ON PRIMARY (NAME = N'[Logischer Dateiname]', FILENAME = N'[Dateipfad]', SIZE = [Größe],
    FILEGROWTH = [Dateiwachstum])
LOG ON (NAME = N'[Logischer Protokolldateiname]', FILENAME = N'[Protokolldateipfad]',
    SIZE = [Größe], FILEGROWTH = [Dateiwachstum])
```

Beispiel:

```
CREATE DATABASE MediaBase2
ON  PRIMARY ( NAME = N'MediaBase2', FILENAME = N'c:\Programme\Microsoft SQL
Server\MSSQL10.SQL2008EXPRESS\MSSQL\DATA\MediaBase2.mdf' , SIZE = 20480KB , FILEGROWTH = 10%)
LOG ON ( NAME = N'MediaBase2_log', FILENAME = N'c:\Programme\Microsoft SQL
Server\MSSQL10.SQL2008EXPRESS\MSSQL\DATA\MediaBase2_log.ldf' , SIZE = 10240KB , FILEGROWTH = 10%)
```

Datenbanken konfigurieren

Syntax:

```
ALTER DATABASE [Datenbankname]
SET [Einstellung]
```

Beispiele:

```
ALTER DATABASE MediaBase2 SET COMPATIBILITY_LEVEL = 100
ALTER DATABASE MediaBase2 SET  READ_WRITE
ALTER DATABASE MediaBase2 SET RECOVERY SIMPLE
```

Schemas erstellen

Syntax:

```
CREATE SCHEMA [Schemaname]
```

Beispiel:

```
CREATE SCHEMA Auswertungen
```

Tabellen erstellen und ändern

Tabellen erstellen

Syntax:

```
CREATE TABLE [Tabellenname]
(
```

```
        [Feldauflistung],
        CONSTRAINT [Primärschlüsselname] PRIMARY KEY CLUSTERED ([Primärschlüsselfeld] [Sortierung])
)
```

Beispiel:

```
CREATE TABLE dbo.Adresse
        (ID int IDENTITY(1,1) NOT NULL,
        Name varchar(80) NOT NULL,
        Strasse varchar(80) NULL,
        PLZ char(5) NULL,
        Wohnort varchar(80) NULL,
        CONSTRAINT PK_Adresse PRIMARY KEY (ID))
```

Tabellen ändern (Spalte hinzufügen)

Syntax:

```
ALTER TABLE [Tabellenname] ADD [Spaltenname] [Datentyp]
```

Beispiel:

```
ALTER TABLE dbo.Adresse ADD Geburtstag datetime
```

Tabellen ändern (Standardwert definieren)

Syntax:

```
ALTER TABLE [Tabellenname] ADD CONSTRAINT [ConstraintName] DEFAULT ([Standardwert]) FOR [Spalte]
```

Beispiel:

```
ALTER TABLE dbo.Buch ADD  CONSTRAINT DF_Buch_Sprache DEFAULT ('deutsch') FOR Sprache
```

Tabellen ändern (Fremdschlüssel definieren)

Syntax:

```
ALTER TABLE [Tabellenname] WITH CHECK ADD CONSTRAINT [Fremdschlüsselname]
FOREIGN KEY ([Fremdschlüsselspalte])
REFERENCES [referenzierte Tabelle]([Primärschlüssel der referenzierten Tabelle])
```

Beispiel:

```
ALTER TABLE dbo.CDTrack
WITH CHECK ADD  CONSTRAINT FK_CDTrack_CD
FOREIGN KEY(idCD)
REFERENCES dbo.CD(ID)
```

Sichten erstellen und ändern

Syntax:

```
CREATE | ALTER VIEW [Sicht]
AS [SELECT-Abfrage]
```

Beispiele:

```
CREATE VIEW dbo.VW_CDsAb2005 AS
SELECT ID, Titel, Erscheinungsjahr
FROM dbo.CD
WHERE (Erscheinungsjahr > 2004)
```

```
ALTER VIEW dbo.VW_CDsAb2005 AS
SELECT ID, Titel, Interpret, Erscheinungsjahr
FROM dbo.CD
WHERE (Erscheinungsjahr > 2004)
```

Indizes erstellen und aktualisieren

Indizes erstellen

Syntax:

```
CREATE INDEX [Indexname]
ON [Tabelle] ([Feldauflistung])
```

Beispiel:

```
CREATE INDEX IX_VerlagErscheinungsjahrAutorTitel
ON dbo.Buch
          (Verlag, Erscheinungsjahr, Autor, Titel)
```

Indizes aktualisieren

Syntax:

```
ALTER INDEX [Indexname] ON [Tabellenname] REORGANIZE
ALTER INDEX [Indexname] ON [Tabellenname] REBUILD
```

Beispiele:

```
ALTER INDEX IX_VerlagErscheinungsjahrAutorTitel ON dbo.Buch REORGANIZE
ALTER INDEX IX_VerlagErscheinungsjahrAutorTitel ON dbo.Buch REBUILD
```

Gespeicherte Prozeduren erstellen und ändern

Gespeicherte Prozeduren erstellen und ändern

Syntax:

```
CREATE | ALTER PROCEDURE [Prozedurname]
 [Par  ameterauflistung]
AS
BEGIN
 [SQL  -Anweisungen]
END
```

Beispiele:

```
CREATE PROCEDURE dbo.BuecherVonAutor
 @Aut  or varchar(80)
AS
BEGIN
   SELECT * FROM dbo.Buch WHERE Autor = @Autor
END

ALTER PROCEDURE dbo.BuecherVonAutor
 @Aut  or varchar(80)
AS
```

```
BEGIN
   SELECT * FROM dbo.Buch WHERE Autor = @Autor
END
```

Benutzerdefinierte Funktionen erstellen und ändern

Skalarwertfunktionen erstellen bzw. ändern

Syntax:

```
CREATE | ALTER FUNCTION [Funktionsname]
 ([Pa  rameterauflistung])
RETURNS [Typ des Rückgabewerts]
AS
BEGIN
 [SQL  -Anweisungen]
   RETURN [Rückgabewert]
END
```

Beispiele:

```
CREATE FUNCTION dbo.UFN_JahreSeit
 (@Ja  hreszahl int)
RETURNS int
AS
BEGIN
   RETURN YEAR(GETDATE())-@Jahreszahl
END

ALTER FUNCTION dbo.UFN_JahreSeit
 (@Ja  hreszahl int)
RETURNS int
AS
BEGIN
   RETURN YEAR(GETDATE())-@Jahreszahl
END
```

Tabellenwertfunktionen erstellen bzw. ändern

Syntax:

```
CREATE | ALTER FUNCTION [Funktionsname]
 ([Pa  rameterauflistung])
RETURNS TABLE
AS
RETURN
(
   [SELECT-Abfrage]
)
```

Beispiele:

```
CREATE FUNCTION dbo.UFN_TracksFromCD
 (@id  CD int)
RETURNS TABLE
AS
```

```
RETURN
(
   SELECT * FROM dbo.CDTrack WHERE idCD=@idCD
)

ALTER FUNCTION dbo.UFN_TracksFromCD
 (@id  CD int)
RETURNS TABLE
AS
RETURN
(
   SELECT * FROM dbo.CDTrack WHERE idCD=@idCD
)
```

Trigger erstellen und ändern

DML-Trigger erstellen und ändern

Syntax:

```
CREATE | ALTER TRIGGER [TriggerName]
   ON  [Tabelle]
   AFTER [Aktion]
AS
BEGIN
 [SQL  -Anweisungen]
END
```

Beispiele:

```
CREATE TRIGGER dbo.TRG_ChangeLog
   ON  dbo.Buch
   AFTER UPDATE
AS
BEGIN
 SET   NOCOUNT ON;
   INSERT INTO AenderungsLog (Aenderung) VALUES ('Zeile in dbo.Buch geändert!')
END
ALTER TRIGGER dbo.TRG_ChangeLog
   ON  dbo.Buch
   AFTER UPDATE, INSERT, DELETE
AS
BEGIN
 SET   NOCOUNT ON;

   INSERT INTO AenderungsLog (Aenderung)
   VALUES ('Datenänderung in Tabelle dbo.Buch!')
END
```

Server-Trigger erstellen und ändern

Syntax:

```
CREATE | ALTER TRIGGER [TriggerName]
   ON  ALL SERVER
   FOR [Ereignisse]
```

```
AS
BEGIN
 [SQL  -Anweisungen]
END
```

Beispiel:

```
CREATE TRIGGER TRG_NewDB
   ON  ALL SERVER
   FOR CREATE_DATABASE
AS
BEGIN
   PRINT 'Achtung! Neue Datenbank! '
END
```

Datenbank-Trigger erstellen und ändern

Syntax:

```
CREATE | ALTER TRIGGER [TriggerName]
   ON  DATABASE
   FOR [Ereignisse]
AS
BEGIN
 [SQL  -Anweisungen]
END
```

Beispiel:

```
CREATE TRIGGER TRG_Datenbank
   ON DATABASE
   FOR DDL_DATABASE_LEVEL_EVENTS
AS
BEGIN
 PRIN  T 'Datenbank-Änderung!'
   INSERT INTO MediaBase.dbo.DDLAenderungsLog (Aenderung) VALUES (EVENTDATA())
END
```

Datenbankobjekte löschen

Syntax:

```
   DROP [Objekttyp] [Objektname]
```

Beispiele:

```
   DROP DATABASE MediaBase
   DROP SCHEMA Auswertungen
   DROP TABLE MeineBuecher
   DROP VIEW dbo.VW_CDsAb2005
   DROP INDEX IX_VerlagErscheinungsjahrAutorTitel
   DROP PROCEDURE dbo.BuecherVonAutor
   DROP FUNCTION dbo.UFN_JahreSeit
   DROP TRIGGER TRG_Datenbank
```

A.4 Data Control Language (DCL)

Anmeldungen und Benutzer anlegen

SQL Server-Anmeldungen anlegen

Syntax:

```
CREATE LOGIN [Anmeldename]
WITH PASSWORD = [Passwort], DEFAULT_DATABASE = [Standarddatenbank],
          CHECK_EXPIRATION = ON | OFF, CHECK_POLICY = ON | OFF
```

Beispiel:

```
CREATE LOGIN MediaBaseReadWrite
WITH PASSWORD=N'mbrw', DEFAULT_DATABASE=master, CHECK_EXPIRATION=OFF, CHECK_POLICY=OFF
```

Windows-Anmeldungen anlegen

Syntax:

```
CREATE LOGIN [Anmeldename] FROM WINDOWS
WITH DEFAULT_DATABASE = [Standarddatenbank]
```

Beispiel:

```
CREATE LOGIN VIRTUALTIGER\mmustermann FROM WINDOWS
WITH DEFAULT_DATABASE=master
```

Datenbankbenutzer anlegen

Syntax:

```
CREATE USER [Benutzername] FOR LOGIN [Anmeldename]
```

Beispiele:

```
CREATE USER [MediaBaseReadOnly] FOR LOGIN [MediaBaseReadOnly]

CREATE USER [VIRTUALTIGER\mmustermann] FOR LOGIN [VIRTUALTIGER\mmustermann]
```

Server- und Datenbankrollen

Anmeldungen einer Serverrolle zuordnen

Syntax:

```
EXEC master..sp_addsrvrolemember @loginname = [Anmeldung], @rolename = [Serverrolle]
```

Beispiel:

```
EXEC master..sp_addsrvrolemember @loginame = N'MediaBaseReadWrite', @rolename = N'diskadmin'
```

Neue Datenbankrolle erstellen

Syntax:

```
CREATE ROLE [Rollenname]
```

Beispiel:

```
CREATE ROLE MediaBaseCDRole
```

Datenbankbenutzer einer Datenbankrolle zuordnen

Syntax:

```
EXEC sp_addrolemember [Rolle], [Datenbankbenutzer]
```

Beispiele:

```
EXEC sp_addrolemember N'db_datareader', N'MediaBaseReadOnly'
EXEC sp_addrolemember N'db_owner', N'VIRTUALTIGER\mmustermann'
```

Server- und Datenbankrechte

Serverrechte für Anmeldungen erteilen, verweigern und entziehen

Syntax:

```
GRANT [Serverrecht] TO [Anmeldung] WITH GRANT OPTION
DENY [Serverrecht] TO [Anmeldung]
REVOKE [Serverrecht] TO | FROM [Anmeldung]
```

Beispiele:

```
GRANT SHUTDOWN TO [VIRTUALTIGER\rpanther] WITH GRANT OPTION
DENY ADMINISTER BULK OPERATIONS TO [VIRTUALTIGER\rpanther]
```

Datenbankrechte für Benutzer und Rollen erteilen, verweigern und entziehen

Syntax:

```
GRANT [Datenbankrecht] TO [Benutzer | Rolle] WITH GRANT OPTION
DENY [Datenbankrecht] TO [Benutzer | Rolle]
REVOKE [Datenbankrecht] TO | FROM [Benutzer | Rolle]
```

Beispiele:

```
GRANT SELECT ON [dbo].[CD] TO [MediaBaseCDRole]
GRANT UPDATE ON [dbo].[CD] TO [MediaBaseCDRole]
```

A.5 SQL Server-Datentypen

SQL Server 2008 bietet eine große Menge an verschiedenen Datentypen. Zur besseren Übersicht werden diese in vier Kategorien unterteilt:

- numerische Datentypen
- alphanumerische Datentypen

- Datums- und Zeittypen
- Sonstige Datentypen

Numerische Datentypen

Tabelle A.1: Numerische Datentypen (ganzzahlig)

Datentyp	Platzbedarf	Bedeutung
bit	1 Byte (kann bis zu 8 Bit-Felder beinhalten)	logischer Wert, gespeichert als 0 oder 1 (entspricht falsch oder wahr)
tinyint	1 Byte	ganzzahliger Wert zwischen 0 und 255
smallint	2 Byte	ganzzahliger Wert zwischen -2^{15} und $2^{15}-1$ (-32.768 bis 32.767)
int	4 Byte	ganzzahliger Wert zwischen -2^{31} und $2^{31}-1$ (-2.147.483.648 bis 2.147.483.647)
bigint	8 Byte	ganzzahliger Wert zwischen -2^{63} und $2^{63}-1$ (-9.223.372.036.854.775.808 bis 9.223.372.036.854.775.807)

Tabelle A.2: Numerische Datentypen (Kommazahlen mit fester Genauigkeit)

Datentyp	Platzbedarf	Bedeutung
decimal(p,s)	5 Byte (für 1≤p≤9) 9 Byte (für 10≤p≤19) 13 Byte (für 20≤p≤28) 17 Byte (für 29≤p≤38)	numerischer Wert mit fester Genauigkeit und fester Anzahl an Dezimalstellen (identisch mit numeric): p = Precision (Genauigkeit): Gesamtanzahl an Stellen (1-36) s = Scale (Dezimalstellen): Stellen rechts vom Komma (0-p)
numeric(p,s)	5 Byte (für 1≤p≤9) 9 Byte (für 10≤p≤19) 13 Byte (für 20≤p≤28) 17 Byte (für 29≤p≤38)	numerischer Wert mit fester Genauigkeit und fester Anzahl an Dezimalstellen (identisch mit decimal): p = Precision (Genauigkeit): Gesamtanzahl an Stellen (1-36, Standardwert ist 18) s = Scale (Dezimalstellen): Stellen rechts vom Komma (0-p)
smallmoney	4 Byte	Kommawert (mit Währung) zwischen -214.748,3648 und 214.748,3647
money	8 Byte	Kommawert (mit Währung) zwischen -922.337.203.685.477,5808 und 922.337.203.685.477,5807

Tabelle A.3: Numerische Datentypen (Gleitkommazahlen mit ungefährer Genauigkeit)

Datentyp	Platzbedarf	Bedeutung
float(n)	4 Byte (für 1≤n≤24) 8 Byte (für 25≤n≤53)	Gleitkommazahl mit ungefährer Genauigkeit - 1,79E+308 bis -2,23E-308, 0 und 2,23E-308 bis 1,79E+308 n hat einen Wert zwischen 1 und 53 und gibt die Anzahl von Bits für die Speicherung der Mantisse an (Standardwert ist 53)
real	4 Byte	Gleitkommazahl mit ungefährer Genauigkeit - 3,40E + 38 bis -1,18E - 38, 0 und 1,18E - 38 bis 3,40E + 38

Alphanumerische Datentypen

Tabelle A.4: Alphanumerische Datentypen (nicht Unicode)

Datentyp	Platzbedarf	Bedeutung
char(n)	1 Byte pro Zeichen	Zeichenkette fester Länge n (1 ≤ n ≤ 8.000)
varchar(n)	1 Byte pro Zeichen + 2 Byte für Länge	Zeichenkette variabler Länge bis maximal n (1 ≤ n ≤ 8.000)
varchar(MAX)	1 Byte pro Zeichen + 2 Byte für Länge	Zeichenkette variabler Länge bis maximal 2^{31}-1 (2.147.483.647)
text	1 Byte pro Zeichen	Zeichenkette variabler Länge bis maximal 2^{31}-1 (2.147.483.647) **Achtung:** Dieser Datentyp wird künftig nicht mehr unterstützt, stattdessen *varchar(max)* verwenden!

Tabelle A.5: Alphanumerische Datentypen (Unicode)

Datentyp	Platzbedarf	Bedeutung
nchar(n)	2 Byte pro Zeichen	Zeichenkette fester Länge n (1 ≤ n ≤ 4.000)
nvarchar(n)	2 Byte pro Zeichen + 2 Byte für Länge	Zeichenkette variabler Länge bis maximal n (1 ≤ n ≤ 4.000)
nvarchar(MAX)	2 Byte pro Zeichen + 2 Byte für Länge	Zeichenkette variabler Länge bis maximal 2^{31}-1
ntext	2 Byte pro Zeichen	Zeichenkette variabler Länge bis maximal 2^{30}-1 (1.073.741.823) **Achtung:** Dieser Datentyp wird künftig nicht mehr unterstützt, stattdessen *nvarchar(max)* verwenden!

Binäre Datentypen

Tabelle A.6: Binäre Datentypen (nicht Unicode)

Datentyp	Platzbedarf	Bedeutung
binary	1 Byte pro Zeichen	binäre Daten fester Länge n (1 ≤ n ≤ 8.000)
varbinary	1 Byte pro Zeichen + 2 Byte für Länge	binäre Daten variabler Länge bis maximal n (1 ≤ n ≤ 8.000)
varbinary(max)	1 Byte pro Zeichen + 2 Byte für Länge	binäre Daten variabler Länge bis maximal 2^{31}-1 (2.147.483.647) (kann auch als Filestream gehandhabt werden)
image	1 Byte pro Zeichen + 2 Byte für Länge	Binärdaten variabler Länge bis maximal 2^{31}-1 (2.147.483.647) **Achtung:** Dieser Datentyp wird künftig nicht mehr unterstützt, stattdessen *varbinary(max)* verwenden!

Zeit- und Datumstypen

Tabelle A.7: Zeit- und Datumstypen

Datentyp	Platzbedarf	Bedeutung
date	3 Byte	Datum ohne Uhrzeit (neu in SQL Server 2008!) 01.01.0001 bis 31.12.9999 (Standardwert ist 01.01.1900)
time(n)	5 Byte	Uhrzeit ohne Datum (neu in SQL Server 2008!) 00:00:00.0000000 bis 23:59:59.9999999 (Standardwert ist 00:00:00) n (0 ≤ n ≤ 7) definiert die Nachkommastellen der Sekunden
smalldatetime	4 Byte	Datum mit Uhrzeit (ohne Sekunden) 01.01.1900 00:00:00 bis 06.06.2079 23:59:59
datetime	8 Byte	Datum mit Uhrzeit 01.01.1753 00:00:00.000 bis 31.12.9999 23:59:59.997 (Standardwert ist 01.01.1900 00:00:00) (Genauigkeit gerundet auf 0.000, 0.003 und 0.007 Sekunden)
datetime2(n)	6 Byte (für 0≤n≤2) 7 Byte (für 3≤n≤4) 8 Byte (für 5≤n≤7)	Datum mit Uhrzeit 01.01.0000 00:00:00.0000000 bis 31.12.9999 23:59:59.9999999 (Standardwert ist 01.01.1900 00:00:00) n (0 ≤ n ≤ 7) definiert die Nachkommastellen der Sekunden
datetimeoffset(n)	10 Byte	Datum mit Uhrzeit und Zeitzonenoffset (maximal ± 14:00) 01.01.0000 00:00:00.0000000 bis 31.12.9999 23:59:59.9999999 (Standardwert ist 01.01.1900 00:00:00) n (0 ≤ n ≤ 7) definiert die Nachkommastellen der Sekunden

Sonstige Datentypen

Tabelle A.8: Sonstige Systemdatentypen

Datentyp	Platzbedarf	Bedeutung
hierarchyid	1-892 Byte	Systemdatentyp variabler Länge, speichert Position in einer Hierarchie
sql_variant		Datentyp, der Werte verschiedener SQL Server-gestützter Datentypen speichert, maximale Größe 8.000 Byte, ausgenommen *varchar(max), varbinary(max), nvarchar(max), xml, text, ntext, image, timestamp* und *sql_variant, hierarchyid* sowie benutzerdefinierte Typen
rowversion	8 Byte	automatisch generierte eindeutige, binäre Zahlen (inkrementelle Werte)
timestamp	8 Byte	automatisch generierte eindeutige, binäre Zahlen (inkrementelle Werte) **Achtung:** Dieser Datentyp wird künftig nicht mehr unterstützt, stattdessen *rowversion* verwenden!
uniqueidentifier	16 Byte	global eindeutige Werte (GUID) in hexadezimaler Darstellung
xml	max. 2 GB	Datentyp zur Speicherung von SQL-Daten, erfordert wohlgeformte XML-Fragmente (CONTENT) oder XML-Dokumente (DOCUMENT)

Tabelle A.9: .NET CLR-Datentypen

Datentyp	Platzbedarf	Bedeutung
geography	(variabel)	räumliche Daten basierend auf einem Erdkugel-Koordinatensystem (z.B. GPS-Daten)
geometry	(variabel)	räumliche Daten basierend auf einem flachen (euklidischen) Koordinatensystem

Mithilfe der CLR-Integration können auch eigene Datentypen in .NET implementiert werden.

Tabelle A.10: Datentypen, die nicht in Tabellen verwendet werden können

Datentyp	Platzbedarf	Bedeutung
cursor	(variabel)	Datentyp für Variablen oder für OUTPUT-Parameter von gespeicherten Prozeduren, die einen Verweis auf einen Cursor enthalten
table	(variabel)	Datentyp zur Speicherung von Resultsets von Tabellenwertfunktionen (wird als lokale Variable genutzt)

A.6 Systemobjekte

Es gibt eine große Anzahl an vordefinierten Systemobjekten, mit denen man unter Umständen viel Arbeit sparen kann. Aufgrund der großen Menge wird im Folgenden nur eine Auswahl der wichtigsten Systemobjekte dargestellt (lediglich die Liste der Systemvariablen ist vollständig).

Systemsichten

Die Systemsichten geben eine Sicht auf die intern verwendeten Systemtabellen. Da die Sichten auch in jeder Benutzerdatenbank verwendbar sind, beziehen sich deren Ergebnisse in der Regel auch auf die Objekte der jeweiligen Datenbank.

Tabelle A.11: Überblick über die wichtigsten Systemsichten

Systemsicht	Bedeutung
INFORMATION_SCHEMA.COLUMNS	listet alle Tabellenspalten mit deren Datentyp und Größe auf
INFORMATION_SCHEMA.ROUTINES	listet alle benutzerdefinierten gespeicherten Prozeduren und Funktionen auf
INFORMATION_SCHEMA.TABLES	listet alle Tabellen auf
INFORMATION_SCHEMA.VIEWS	listet alle Sichten auf
sys.databases	listet alle Datenbanken des aktuellen Servers auf

Systemfunktionen

Tabelle A.12: Überblick über die wichtigsten Zeichenkettenfunktionen

Systemfunktion	Bedeutung
LEFT(String, Anzahl)	liefert eine bestimmte Anzahl Zeichen von der linken Seite ausgehend
RIGHT(String, Anzahl)	liefert eine bestimmte Anzahl Zeichen von der rechten Seite ausgehend
SUBSTRING(String, Start, Anzahl)	liefert eine bestimmte Anzahl Zeichen ab der angegebenen Position
LTRIM(String)	entfernt führende Leerzeichen
RTRIM(String)	entfernt folgende Leerzeichen
UPPER(String)	Umwandlung in Großbuchstaben
LOWER(String)	Umwandlung in Kleinbuchstaben

Tabelle A.13: Überblick über die wichtigsten Datumsfunktionen

Systemfunktion	Bedeutung
GETDATE()	das aktuelle Datum
YEAR(Datum)	das Jahr des angegebenen Datums
MONTH(Datum)	der Monat des angegebenen Datums
DAY(Datum)	der Tag des angegebenen Datums
DATEPART(Einheit, Datum)	der zur Einheit passende Bestandteil des Datums
DATEADD(Einheit, Anzahl, Datum)	addiert eine bestimmte Menge an übergebenen Einheiten zu einem Datum
DATEDIFF(Einheit, Datum1, Datum2)	Differenz zwischen zwei Datumswerten in der angegebenen Einheit

Tabelle A.14: Überblick über die wichtigsten mathematischen Funktionen

Systemfunktion	Bedeutung
ABS(Zahl)	liefert den absoluten Wert (also die Zahl ohne Vorzeichen)
SIN(Zahl)	liefert den Sinus des angegebenen Wertes zurück
COS(Zahl)	liefert den Cosinus des angegebenen Wertes zurück
LOG(Zahl)	liefert den natürlichen Logarithmus des angegebenen Wertes zurück
EXP(Zahl)	liefert den exponentiellen Wert der angegebenen Zahl zurück
PI()	die Zahl Pi (3.14159265358979)

Tabelle A.15: Überblick über einige weitere Funktionen

Systemfunktion	Bedeutung
USER_NAME()	liefert den Datenbank-Benutzernamen
HOST_NAME()	liefert den Namen der Arbeitsstation zurück
APP_NAME()	liefert den Namen der aktuellen Anwendung zurück
NEWID()	liefert eine neue GUID zurück

Systemprozeduren

Tabelle A.16: Überblick über die wichtigsten Systemprozeduren

Systemprozedur	Bedeutung
master..sp_addrolemember	fügt einer Datenbankrolle einen Benutzer hinzu
master..sp_addsrvrolemember	fügt einer Serverrolle eine Anmeldung hinzu
sys.sp_renamedb	dient zur Umbenennung von Datenbanken
sys.sp_sqlexec	führt eine übergebene SQL-Anweisung aus
sys.sp_who	zeigt eine Liste aller Serverprozesse
sys.sp_who2	zeigt eine Liste aller Serverprozesse (ausführlichere Version)

Systemvariablen

Tabelle A.17: Gesamtüberblick der Systemvariablen

Systemvariable	Typ	Bedeutung
@@Connections int		Anzahl der (erfolgreichen und erfolglosen) Verbindungen seit dem letzten Start von SQL Server
@@CPU_BUSY int		CPU-Zeit seit dem letzten Start von SQL Server (Angabe in Timeticks)
@@CURSOR_ROWS int		Anzahl der Zeilen im aktuellen Cursor (negativ bei einem asynchron gefüllten oder dynamischen Cursor)
@@DATEFIRST tinyint		erster Tag der Woche (1 = Montag, 7 = Sonntag)
@@DBTS varbinary		letzter verwendeter Timestamp
@@DEF_SORTORDER_ID tinyint		Standardwert für die ID der Sortierung (wird künftig eventuell nicht mehr unterstützt, da nicht dokumentiert)
@@DEFAULT_LANGID smallint		Standardwert für die Sprach-ID (@@LANGID) (wird künftig eventuell nicht mehr unterstützt, da nicht dokumentiert)
@@ERROR int		Fehlernummer zur letzten SQL-Anweisung (0 = kein Fehler)
@@FETCH_STATUS int		Status der letzten Cursor-FETCH-Anweisung (0 = erfolgreich)
@@IDENTITY numeric		zuletzt eingefügter Identitätswert
@@IDLE int		Leerlaufzeit seit dem letzten Start von SQL Server (Angabe in Timeticks)
@@IO_BUSY int		I/O-Zeit seit dem letzten Start von SQL Server (Angabe in Timeticks)
@@LANGID smallint		ID der aktuell eingestellten Sprache
@@LANGUAGE nvarchar		Sprache des SQL Servers
@@LOCK_TIMEOUT int		Sperrtimeout für die aktuelle Sitzung in Millisekunden
@@MAX_CONNECTIONS int		maximal mögliche Anzahl von Verbindungen
@@MAX_PRECISION tinyint		Genauigkeitsgrad, der von den Datentypen *decimal* und *numeric* verwendet wird
@@MICROSOFTVERSION int		Version des SQL Servers (binär kodiert) Die folgende Abfrage zeigt die Hauptversionsnummer an: `SELECT @@MicrosoftVersion / 0x01000000` (wird künftig eventuell nicht mehr unterstützt, da nicht dokumentiert)
@@NESTLEVEL i	*nt*	aktuelle Schachtelungsebene innerhalb von gespeicherten Prozeduren
@@OPTIONS int		aktuelle SET-Optionen (binär kodiert)
@@PACK_RECEIVED int		Anzahl der empfangenen Netzwerkpakete seit dem letzten Start von SQL Server

Systemvariable	Typ	Bedeutung
@@PACK_SENT int		Anzahl der gesendeten Netzwerkpakete seit dem letzten Start von SQL Server
@@PACKET_ERRORS int		Anzahl der Netzwerkpaket-Fehler seit dem letzten Start von SQL Server
@@PROCID int		ID des aktuellen T-SQL-Moduls (gespeicherte Prozedur, Funktion oder Trigger)
@@REMSERVER nvarchar(128)		Name des Remote-Datenbankservers, von dem aus ein Zugriff erfolgt ist (wird künftig nicht mehr unterstützt!)
@@ROWCOUNT int		Anzahl Zeilen, auf die sich die letzte SQL-Anweisung ausgewirkt hat
@@SERVERNAME nvarchar		Name (und Instanz) des SQL Servers
@@SERVICENAME nvarchar		Name des SQL Server-Dienstes (entspricht dem Instanznamen)
@@SPID smallint		SQL-interne Prozess-ID
@@TEXTSIZE int		Anzahl Zeichen, die beim `SELECT` eines Feldes mit dem Datentypen *varchar(max), nvarchar(max), varbinary(max), text, ntext* und *image* ausgegeben werden
@@TIMETICKS int		Anzahl Mikrosekunden pro Timetick
@@TOTAL_ERRORS int		Anzahl der Schreibfehler seit dem letzten Start von SQL Server
@@TOTAL_READ int		Anzahl Lesezugriffe auf Datenträger (nicht auf Cache) seit dem letzten Start von SQL Server
@@TOTAL_WRITE int		Anzahl Schreibzugriffe auf Datenträger seit dem letzten Start von SQL Server
@@TRANCOUNT int		Zahl der offenen Transaktionen
@@VERSION nvarchar		Name und Version des SQL Servers

Anhang B

Inhalt der Buch-DVD

Auf der Buch-DVD befinden sich SQL Server 2008 Express sowie SQL Server 2008 R2 Express jeweils in verschiedenen Varianten sowie Updates, Service Packs und weitere Tools, die zur Installation des SQL Server 2008 erforderlich sind. Falls Sie dieses Buch als ebook erworben haben, können Sie die Begleitdateien unter *www.microsoft-press.de/support.asp?s110=218* oder unter *msp.oreilly.de/support/9783866452183/612* herunterladen.

Die Buch-DVD ist wie folgt gegliedert:

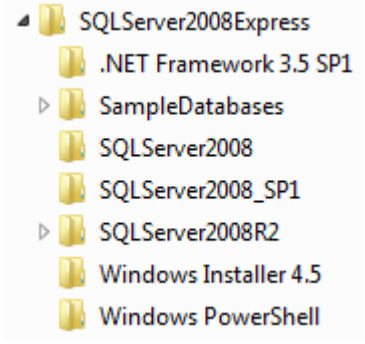


Abbildung B.1: Die Ordnerstruktur der Buch-DVD

Hinweis: Lizenzbedingungen beachten

Bitte beachten Sie, dass die gültigen Lizenzbestimmungen der jeweiligen Hersteller (Microsoft) gelten. Die Lizenzbestimmungen werden bei der Installation der entsprechenden Tools angezeigt. Für SQL Server 2008 R2 Express sind diese außerdem im Ordner *\SQLServer2008R2\Lizenzinformationen* der Buch-DVD zu finden.

B.1 SQL Server 2008/2008 R2 Express

Die Installationsdateien zu SQL Server 2008 Express sowie SQL Server 2008 R2 Express sind in den jeweiligen Unterverzeichnissen *SQLServer2008* und *SQLServer2008R2* der DVD zu finden. Dabei sind aus Platzgründen nur die 32-Bit-Varianten der Software dort zu finden, während die 64-Bit-Varianten bei Bedarf aus dem Internet heruntergeladen werden können.

Tabelle B.1: Die Dateien im Ordner *SQLServer2008*

Datei	Inhalt
SQLEXPR_x86_DEU.exe	SQL Server 2008 Express Edition (deutsch)
SQLEXPRADV_x86_DEU.exe	SQL Server 2008 Express Edition mit Advanced Services (deutsch)
SQLEXPRWT_x86_DEU.exe SQL	Server 2008 Express Edition mit Tools (deutsch)
SQLManagementStudio_x86_DEU.exe SQL	Server 2008 Management Studio (deutsch)
SQLServer2008_BOL_Jun2010_DEU.msi	SQL Server 2008 Books Online – Juni 2010 (deutsch)

Tabelle B.2: Die Dateien im Ordner *SQLServer2008R2*

Datei	Inhalt
SQLEXPR_x86_DEU.exe	SQL Server 2008 R2 Express Edition (deutsch)
SQLEXPRADV_x86_DEU.exe	SQL Server 2008 R2 Express Edition mit Advanced Services (deutsch)
SQLEXPRWT_x86_DEU.exe	SQL Server 2008 R2 Express Edition mit Tools (deutsch)
SQLManagementStudio_x86_DEU.exe SQL	Server 2008 R2 Management Studio (deutsch)
SQLServer2008R2_BOL_DEU.msi	SQL Server 2008 R2 Books Online (deutsch)

Dabei ist zu beachten, dass die Namen der Installationsdateien für SQL Server 2008 und SQL Server 2008 R2 identisch sind.

B.2 Updates und Service Packs

Bisher gibt es nur die Grundvariante der Express Edition als komplettes Setup, das bereits das Service Pack 1 beinhaltet. Für alle anderen Varianten (*SQL Server 2008 Express mit Tools* bzw. *SQL Server 2008 Express mit Advanced Services*) muss das Service Pack 1 separat installiert werden.

Allerdings ist das Service Pack 1 lediglich für SQL Server 2008 Express relevant, da die R2 Edition später erschienen ist und daher die Korrekturen des Service Packs bereits beinhaltet.

Tabelle B.3: Die Dateien im Ordner *SQLServer2008_SP1*

Datei	Inhalt
SQLEXPR_x86_DEU.exe S	QL Server 2008 Express Edition mit SP1 (deutsch) (Version für 32- und 64-Bit-Systeme)
SQLEXPR32_x86_DEU.exe	SQL Server 2008 Express Edition mit SP1 (deutsch) (Version für 32-Bit-Systeme)
SQLServer2008SP1-KB968369-x86-DEU.exe	SQL Server 2008 Service Pack 1 (deutsch) (für alle Editionen von SQL Server 2008)

B.3 Zusatztools und sonstige Dateien

In den restlichen Ordnern sind vor allem Dateien zu finden, die als Installationsvoraussetzung für SQL Server 2008 Express bzw. SQL Server 2008 R2 Express erforderlich sind.

Tabelle B.4: Die Dateien im Ordner *.NET Framework 3.5 SP1*

Datei	Inhalt
dotnetfx35.exe .NET	Framework 3.5 SP1

Tabelle B.5: Die Dateien im Ordner *Windows Installer 4.5*

Datei	Inhalt
redist.txt	Hinweise zur Verteilung
Windows6.0-KB942288-v2-ia64.msu	Windows Installer 4.5 für Windows Server 2008 & Windows Vista (64-Bit, Intel Itanium)
Windows6.0-KB942288-v2-x64.msu W	indows Installer 4.5 für Windows Server 2008 & Windows Vista (64-Bit)
Windows6.0-KB942288-v2-x86.msu W	indows Installer 4.5 für Windows Server 2008 & Windows Vista (32-Bit)
WindowsServer2003-KB942288-v4-ia64.exe	Windows Installer 4.5 für Windows Server 2003 (64-Bit, Intel Itanium)
WindowsServer2003-KB942288-v4-x64.exe	Windows Installer 4.5 für Windows Server 2003 (64-Bit)
WindowsServer2003-KB942288-v4-x86.exe	Windows Installer 4.5 für Windows Server 2003 (32-Bit)
WindowsXP-KB942288-v3-x86.exe	Windows Installer 4.5 für Windows XP (32-Bit)

Tabelle B.6: Die Dateien im Ordner *Windows PowerShell*

Datei	Inhalt
Inhalt.txt	Übersicht der Dateien in diesem Ordner
Windows6.0-KB928439-x86.msu W	indows PowerShell 1.0 – Installationspaket für Windows Vista (KB928439)
WindowsServer2003-KB926139-v2-x86-ENU.exe W	indows PowerShell 1.0 – English-Language Installation Package for Windows XP (KB926139)
WindowsServer2003-KB926140-v5-x86-DEU.exe W	indows PowerShell 1.0 – Lokalisiertes Installationspaket für Windows XP (KB926140)
WindowsServer2003-KB926141-x86-ENU.exe	Windows PowerShell 1.0 – Multilingual User Interface Package for Windows XP (KB926141)
WindowsXP-KB926139-v2-x86-ENU.exe	Windows PowerShell 1.0 English-Language Installation Package for Windows Server 2003 (KB926139)
WindowsXP-KB926140-v5-x86-DEU.exe	Windows PowerShell 1.0 – Lokalisiertes Installationspaket für Windows Server 2003 (KB926140)
WindowsXP-KB926141-v2-x86-ENU.exe	Windows PowerShell 1.0 Multilingual User Interface Package for Windows Server 2003 (KB926141)

Dazu kommen einige Beispieldatenbanken, die normalerweise über *CodePlex* erhältlich sind.

Tabelle B.7: Die Dateien im Ordner *SampleDatabases*

Datei	Inhalt
AdventureWorks2008R2AZ.zip	SQL Azure-Version der AdventureWorks-Beispieldatenbank
AdventureWorks2008R2_RTM.exe	Verschiedene Varianten der AdventureWorks-Datenbank für SQL Server 2008 R2 mit Installationsoberfläche
AdventureWorks2008_SR4.exe Verschiedene	Varianten der AdventureWorks-Datenbank für SQL Server 2008 mit Installationsoberfläche
AdventureWorksDB.msi S	QL Server 2005-Beispieldatenbanken
SQL2000SampleDb.msi	Die klassischen Beispieldatenbanken von SQL Server 2000 (Northwind & Pubs)

Anhang C

Weiterführende Infos im Web

Damit Sie sich die Tipparbeit beim Ausprobieren der unten aufgeführten Links etwas erleichtern können, sind in den folgenden Linklisten in der Spalte *Link-ID* auch sogenannte Softlinks angegeben. Um diese zu nutzen, geben Sie auf der Startseite von *http://www.vsxpress.de* die angegebene Link-ID ein und klicken dann auf *Link öffnen*. Alternativ können Sie auch direkt in der Adressleiste Ihres Browsers die folgende URL eingeben und dabei *id* durch die angegebene Link-ID ersetzen:

*http://go.vsxpress.de/?linkid=**id***

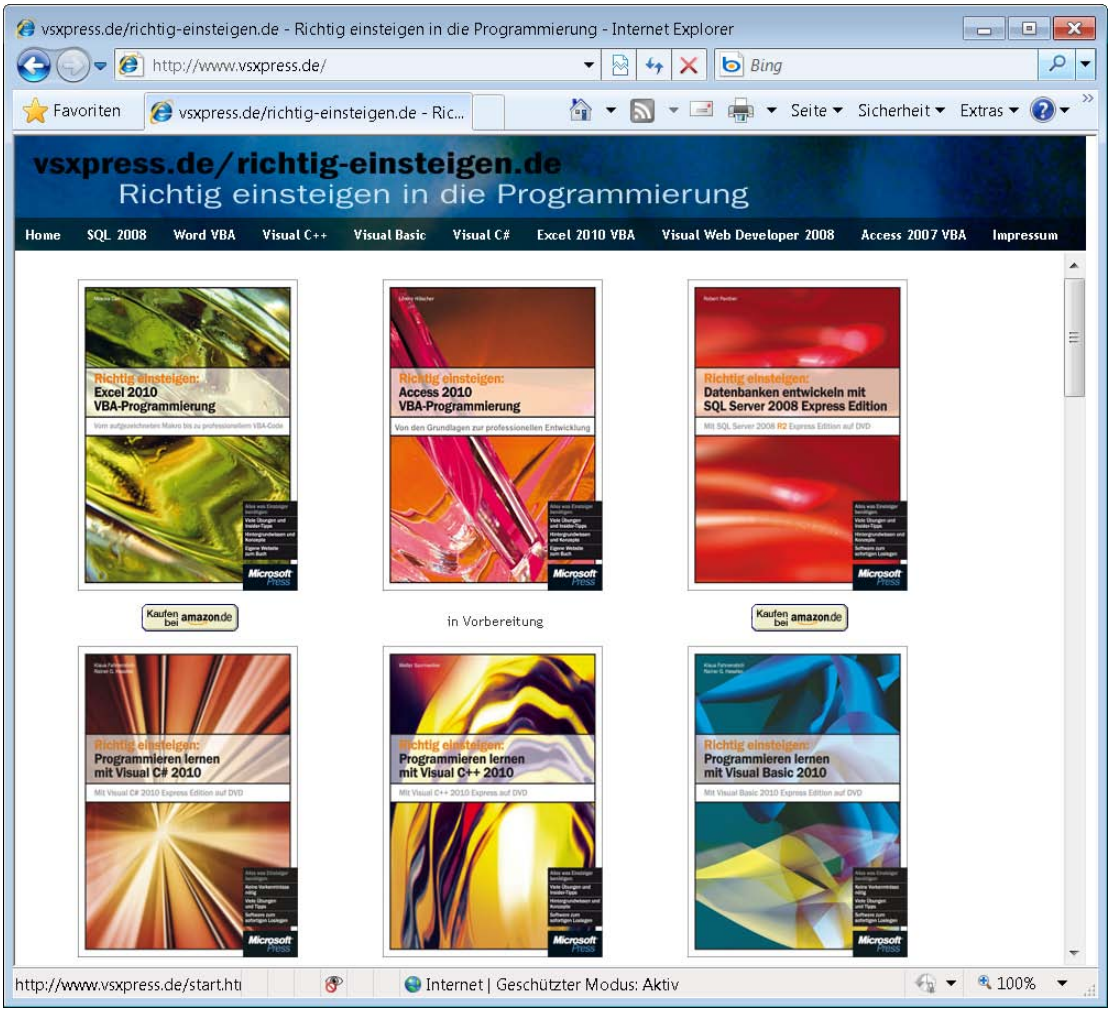


Abbildung C.1: Die Website zur Buchreihe *Richtig einsteigen*

C.1 Die Website zu Buchreihe, Verlag und Autor

Tabelle C.1: Die Links zur Buchreihe, zum Microsoft-Press Webshop und zur Website des Autors

Bedeutung	Hyperlink	Link-ID
Website der »Richtig einsteigen«-Buchreihe	www.vsxpress.de	sqlc01
Website des Microsoft Press-Webshops	www.microsoft-press.de	sqlc02
Website des Autors	www.panthercomputing.de	sqlc03

C.2 Microsoft-Websites zu SQL Server

Tabelle C.2: Microsoft SQL Server Express-Produktseiten

Bedeutung	Hyperlink	Link-ID
Deutsche SQL Server 2008-Seite	www.microsoft.com/germany/sql/2008	sqlc04
Englische SQL Server 2008-Seite	www.microsoft.com/sql/2008/default.mspx	sqlc05
SQL Server 2008 Express Edition	www.microsoft.com/germany/sql/2008/express.mspx	sqlc06
Visual Studio Express Editions (SQL Server 2008 Express)	www.microsoft.com/germany/express/products/database.aspx	sqlc07
SQL Server 2008 R2 (englisch)	www.microsoft.com/sqlserver/2008/en/us/R2.aspx	sqlc31

Tabelle C.3: MSDN & TechNet

Bedeutung	Hyperlink	Link-ID
MSDN SQL Server Developer Center (deutsch)	msdn.microsoft.com/de-de/sqlserver/default.aspx	sqlc08
MSDN SQL Server Developer Center (englisch)	msdn.microsoft.com/en-us/sqlserver/default.aspx	sqlc09
MSDN-Library (SQL Server 2008)	msdn.microsoft.com/de-de/library/ms130214(v=sql.100).aspx	sqlc10
Microsoft TechNet (SQL Server TechCenter)	technet.microsoft.com/de-de/sqlserver/default.aspx	sqlc11
MSDN-Library (SQL Server 2008 R2)	msdn.microsoft.com/de-de/library/ms130214.aspx	sqlc32

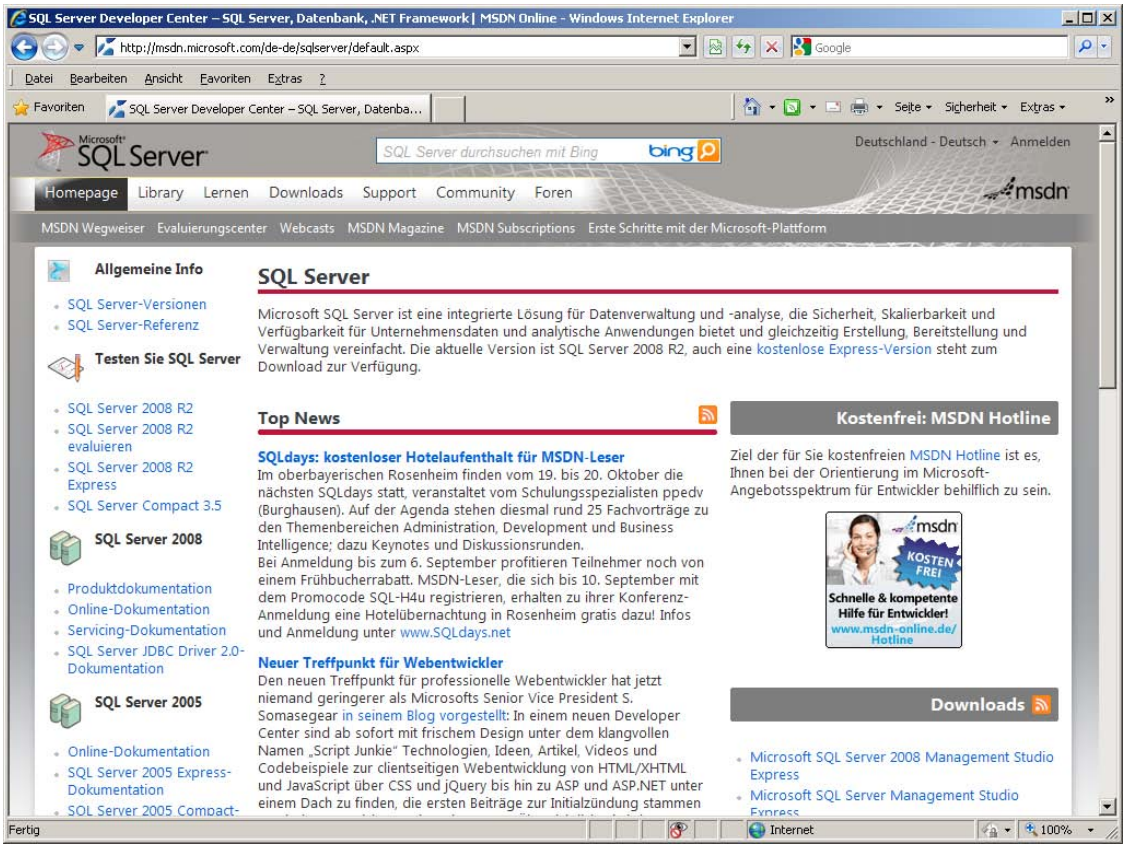


Abbildung C.2: Das SQL Server Developer Center-Ausgangspunkt für viele Informationen

Tabelle C.4: SQL Server 2008-Launch-Event

Bedeutung	Hyperlink	Link-ID
Microsoft Launch Event 2008	www.microsoft.com/germany/msdn/launch2008/default.mspx sqlc1	2
SQL Server 2008 Launch	www.microsoft.com/germany/msdn/launch2008/sql/default.mspx	sqlc13
SQL Server 2008 R2 Launch	www.sqlpass.org/summit/eu2010/Agenda/R2Launch.aspx s	qlc33

C.3 Sonstige Websites zu SQL Server

Die Zahl der Websites, die sich mit SQL Server befassen, wächst von Tag zu Tag. Daher habe ich hier eine Auswahl meiner persönlichen Favoriten zusammengestellt:

Tabelle C.5: Allgemeine Websites zu SQL Server

Bedeutung	Hyperlink	Link-ID
Inside-SQL (deutschsprachiges SQL Server-Portal)	www.insidesql.org sqlc1	4
SQL Server Central (englisch)	www.sqlservercentral.com	sqlc15
SQL Server Pedia (englisch)	www.sqlserverpedia.com	sqlc16
SQL Server Performance (englisch)	www.sql-server-performance.com sqlc1	7
CodePlex (SQL Server Area)	www.codeplex.com/site/search?TagName=SQL%20Server sqlc1	8

Tabelle C.6: SQL Server-Konferenzen

Bedeutung	Hyperlink	Link-ID
Europäische PASS-Konferenz	www.european-pass-conference.com sqlc1	9
SQLCON! (im Rahmen der BASTA!)	www.sqlcon.net	sqlc20
SQL-Konferenz	www.sql-konferenz.de sqlc21	

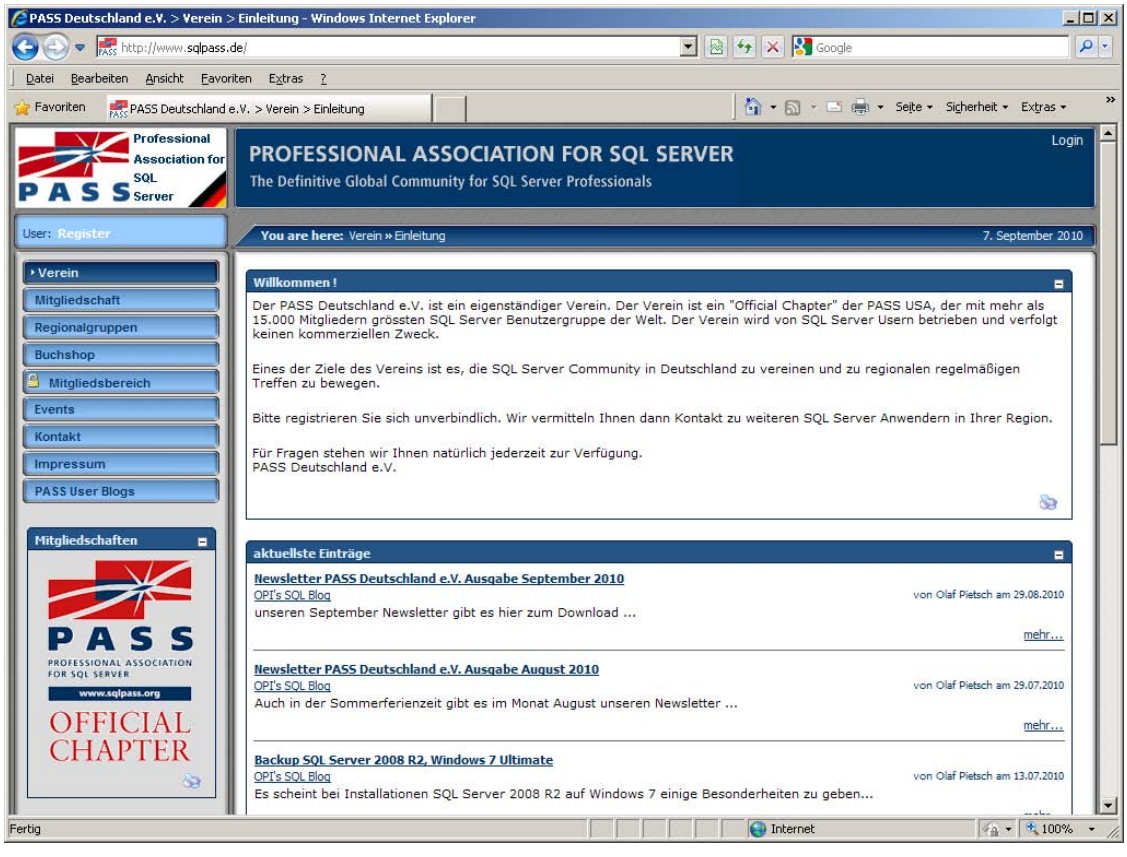


Abbildung C.3: Die Website der deutschen PASS (Untergruppe der weltweit agierenden SQL Server Community)

Tabelle C.7: PASS – Professional Association for SQL Server

Bedeutung	Hyperlink	Link-ID
PASS International (englisch)	www.sqlpass.org	sqlc22
PASS Deutschland (deutsch)	www.sqlpass.de	sqlc23

C.4 SQL Server-Foren und -Newsgroups

Tabelle C.8: MSDN-Foren

Bedeutung	Hyperlink	Link-ID
SQL Server (deutsch)	social.msdn.microsoft.com/Forums/de-DE/category/sqlserver sqlc2	4
SQL Server (englisch)	social.msdn.microsoft.com/forums/en-US/category/sqlserver sqlc2	5
SQL Server Express (deutsch)	social.msdn.microsoft.com/Forums/de-DE/sqlserverexpressde/threads sqlc2	6
SQL Server Express (englisch)	social.msdn.microsoft.com/forums/en-US/sqlexpress/threads sqlc2	7

Tabelle C.9: Microsoft SQL Server-Newsgroups

Bedeutung	Hyperlink	Link-ID
Microsoft Allgemein	www.microsoft.com/communities/newsgroups/en-us/default.aspx sqlc2	8
SQL Server (deutsch)	www.microsoft.com/communities/newsgroups/en-us/default.aspx?dg=microsoft.public.de.sqlserver	sqlc29
SQL Server (englisch)	www.microsoft.com/communities/newsgroups/en-us/default.aspx?dg=microsoft.public.sqlserver.server	sqlc30

Anhang D

Glossar

Im Umfeld von Microsoft SQL Server wird man mit einer großen Menge an Spezialbegriffen und Abkürzungen konfrontiert. Dieses Glossar soll daher die wichtigsten Begriffe erläutern.

ADO (ActiveX Data Objects) Programmierschnittstelle zum Zugriff auf verschiedene Datenbanken (auch unterschiedlicher Anbieter).

ADO.NET Nachfolgetechnologie von ADO, die mit dem .NET Framework eingeführt wurde. Dabei handelt es sich um eine Sammlung von Klassen zum Zugriff auf relationale Datenbanken, aber auch auf XML-Strukturen.

ADO.NET Entity Framework O/R-Mapper von Microsoft, der seit dem Service Pack 1 des .NET Framework 3.5 Bestandteil des Frameworks ist.

ANSI SQL Standardisierte Form der SQL-Sprache.

Assembly Vorkompilierte .NET-Komponente, die normalerweise in Form einer DLL-Datei gespeichert wird.

Authentifizierung Vorgang, mit dem ein Benutzer identifiziert wird.

Autorisierung Vorgang, mit dem bereits identifizierte Benutzer bestimmte Berechtigungen erhalten.

BIDS (Business Intelligence Development Studio) Auf der Entwicklungsumgebung Visual Studio basierende grafische Oberfläche, um mit den Projekttypen für die SQL Server Analysis Services (SSAS), Integration Services (SSIS) und Reporting Services (SSRS) arbeiten zu können.

CLR (Common Language Runtime) Laufzeitumgebung von .NET, die vorkompilierten Code unmittelbar vor dessen Ausführung in eine ausführbare Form übersetzt.

Compact Edition siehe *SQL Server Compact Edition*

CRUD (CRreate Update Delete) Sammelbegriff für die grundlegenden Datenoperationen, mit denen Anwender mit einer Datenbank arbeiten.

CSL (Conceptual Schema Definition Language) Schemadefinitionssprache, mit der das konzeptionelle Modell für das ADO.NET Entity Framework definiert wird.

CTP (Community Technology Preview) Vorabversion, die bereits den endgültigen Funktionsumfang beinhaltet, aber der Benutzer-Community schon zur Verfügung gestellt wird, noch bevor sie zur eigentlichen Produktion freigegeben ist.

DAC-Paket Datei, in der eine Datenebenenanwendung abgelegt ist. Die Datei ist eigentlich ZIP-komprimiert, verwendet aber .dacpac als Dateiendung.

Datenebenenanwendung Datenbankanwendung bzw. die Datenstrukturen dazu, die ab SQL Server 2008 R2 speziell im SQL Server verwaltet werden und somit einfacher auf andere Serverinstanzen verteilbar sind.

DB (Datenbank) Sammlung von Daten bestehend aus Tabellen, die wiederum Zeilen und Spalten enthalten, sowie anderen Datenbankobjekten wie beispielsweise Sichten, gespeicherten Prozeduren etc. Umgangssprachlich wird der Begriff Datenbank fälschlicherweise auch oft verwendet, wenn eigentlich ein DBMS gemeint ist.

DBMS (Datenbank Management System) Software zum Bereitstellen und Verwalten einer Datenbank (z.B. Microsoft SQL Server), oft wird auch die Kurzform Datenbanksystem verwendet.

DCL (Data Control Language) Befehle der SQL-Sprache, die zur Verwaltung von Benutzern, Rollen und Berechtigungen dienen.

DDL (Data Definition Language) Befehle der SQL-Sprache, die zum Erstellen, Ändern und Löschen von Datenstrukturen dienen.

DLL (Dynamic Link Library) Bibliotheken, die Prozeduren, Klassen, Objekte und Konstanten zur Verfügung stellen können.

DML (Data Manipulation Language) Befehle der SQL-Sprache, die zum Einfügen, Ändern und Löschen von Daten dienen.

DQL (Data Query Language) Befehle der SQL-Sprache, die zum Abfragen von Daten dienen. Da sich dies vor allem auf die SELECT-Anweisung bezieht, wird die Abkürzung DQL nur selten verwendet.

DTS (Data Transformation Services) ETL-Technologie von Microsoft, die bei SQL Server 2000 verwendet wurde, inzwischen (seit SQL Server 2005) durch SSIS abgelöst. ETL-Abläufe wurden bei DTS in sogenannten DTS-Paketen definiert und gespeichert.

DWH (Data Warehouse) Meist denormalisierte Speicherung von Daten in einer relationalen Datenbank, um diese schneller auswerten zu können.

Dynamic SQL Zur Programmlaufzeit dynamisch generierte SQL-Anweisungen.

Embedded SQL SQL-Anweisungen, die in den Quelltext von Programmen eingebunden sind.

Entity Framework siehe *ADO.NET Entity Framework*

ETL (Extract, Transform, Load) Standardvorgehen, um Daten aus verschiedenen Quellen zu lesen, zu transformieren und anschließend wieder zu schreiben.

Hauptspeicher siehe *RAM*

Hotfix Update, das kleinere Fehler behebt und – insbesondere dann, wenn es sich um sicherheitsrelevante Fehler handelt – sehr zeitnah nach deren Behebung erscheint.

IIS (Internet Information Server) Webserverprodukt von Microsoft, das unter anderem genutzt werden kann, um für die Reporting Services Berichte zur Verfügung zu stellen.

Inline-SQL siehe *Embedded SQL*

IntelliSense Automatische Vervollständigung bei der Eingabe von Programmcode.

Kumulatives Update Zusammengefasstes Update, das alle Hotfixes und Einzel-Updates beinhaltet, die seit dem letzten Service Pack veröffentlicht wurden.

LINQ (Language Integrated Query) Abfragesprache, die es ermöglicht, über verwalteten Code auf verschiedene Objekte (je nach LINQ-Implementierung) typsicher zuzugreifen.

LINQ to Entities LINQ-Implementierung zum Zugriff auf Entities, die vom ADO.NET Entity Framework zur Verfügung gestellt werden.

LINQ to SQL LINQ- Implementierung zum Zugriff auf SQL-Datenbanken.

MMC (Microsoft Management Console) Mit dem Windows-Betriebssystem ausgeliefertes Administrationstool, mit dem verschiedene Serveranwendungen – wie beispielsweise SQL Server, aber auch der Internet Information Server – verwaltet werden können.

MSL (Mapping Specification Language) Schemadefinitionssprache, mit der die Zuordnung von Datenbankelementen auf Objektklassen für das Entity Framework definiert wird.

Named Pipes Netzwerkprotokoll, das auf benannten FIFO-Datenströmen basiert und auch bidirektional zwischen verschiedenen Rechnern genutzt werden kann. Pipes werden durch die folgende Zeichenfolge angesprochen: *\\ServerName\pipe\PipeName*

Namespace In .NET verwendetes Konstrukt zur Gruppierung von Basisklassen und weiteren Namespaces.

.NET von Microsoft entwickelte Plattform zum Entwickeln von Anwendungen auf Basis des .NET Frameworks.

.NET Framework Laufzeitumgebung für .NET-Anwendungen.

O/R-Mapper Entwicklertools, die Klassen generieren, um objektorientiert auf relationale Datenbanken zugreifen zu können.

ODBC (Open Database Connectivity) Programmierschnittstelle von Microsoft, um verschiedene Datenquellen über ein einheitliches Vorgehen ansprechen zu können; mittlerweile durch OLEDB abgelöst.

OLAP (Online Analytical Processing) Beschreibt die Möglichkeit, komplexe Datenauswertungen auf Basis eines Data Warehouses oder OLAP-Würfels durchzuführen.

OLAP-Würfel (engl. OLAP cube) Repräsentiert die multidimensionale Speicherung von Daten. Auch wenn ein richtiger Würfel nur drei Dimensionen umfasst, kann ein OLAP-Würfel auch mehr Dimensionen umfassen.

OLEDB Object Linking and Embedding Database (teilweise auch in der Schreibweise OLE DB zu finden), Programmierschnittstelle von Microsoft, um verschiedene Datenquellen über ein einheitliches Vorgehen ansprechen zu können; ersetzt die veraltete ODBC-Schnittstelle.

OLTP (Online Transaction Processing) Beschreibt die transaktionelle Verarbeitung von Daten, wie sie normalerweise in relationalen Datenbanksystemen stattfindet.

Pascal Casing In der Softwareentwicklung weit verbreitete Schreibweise, bei der das erste Zeichen jedes Wortes großgeschrieben wird (z.B. DatenLesen).

PASS (Professional Association for SQL Server) Weltweite SQL Server Community mit nationalen und regionalen Untergruppen (die Mitgliedschaft ist kostenlos und jedem zu empfehlen, der sich intensiver mit Microsoft SQL Server auseinandersetzt).

PowerPivot Mit SQL Server 2008 R2 eingeführtes Add-On für Excel und SharePoint, das schnelle und einfache Auswertungen auf OLAP Cubes ermöglicht, ohne dafür die gewohnten Tools verlassen zu müssen.

RAM (Random Access Memory) Flüchtiger Speicher eines Computers; der Zugriff erfolgt deutlich schneller, als auf nicht flüchtige Speicher (wie beispielsweise Festplatten), dafür muss eine konstante Stromversorgung gegeben sein.

RDBMS (Relationales Datenbank-Management-System) Datenbank-Management-System, das auf Tabellen basiert, die mit so genannten Relationen verknüpft werden können. Fast alle aktuellen Datenbanksysteme arbeiten nach diesem Verfahren.

RDL (Report Definition Language) Von Microsoft entwickelte Sprache, um Reportdefinitionen zu speichern (wird inzwischen auch von anderen Anbietern eingesetzt).

Replikation Technologie zur automatisierten Verteilung von Datenänderungen zwischen verschiedenen Datenbankservern.

RTM (Ready To Manufacturing) Endgültige Version des Produkts, die auch in derselben Form in Produktion geht.

Schema Dieser Begriff wird in folgenden Bedeutungen verwendet:

- Ein Datenbankschema beschreibt die Struktur der Datenbank (also deren Tabellendefinitionen etc.).
- Ein SQL-Schema bildet eine logische Gruppierung von Datenbankobjekten (Tabellen, Sichten, gespeicherte Prozeduren etc.), um auf dieser Ebene leichter Berechtigungen erteilen zu können.

SEQUEL (Structured English Query Language) Vorgänger der Sprache SQL, die 1975 von IBM entwickelt wurde.

Service Pack Größeres Update eines Produkts, das nicht selten auch wesentliche Erweiterungen des Funktionsumfangs beinhaltet. Ein Service Pack beinhaltet normalerweise alle bis dahin erschienen Updates sowie vorherige Service Packs.

Shared Memory Interprozess-Kommunikationsprotokoll, bei dem zwei Prozesse einen gemeinsamen Teil des Hauptspeichers nutzen, wodurch dieses Protokoll nicht zur Kommunikation zwischen verschiedenen Rechner verwendet werden kann.

SMO (SQL Server Management Objects) Sammlung von Objekten, die von Entwicklern genutzt werden können, um SQL Server zu verwalten.

SQL (Structured Query Language) Standardisierte Datenbank-Abfragesprache, die von den meisten aktuellen Datenbanksystemen verwendet wird.

SQL Azure Microsofts Cloud-basierte Variante von SQL Server.

SQL Data Services Ursprünglich geplante Variante der Cloud-basierten SQL-Variante, inzwischen durch SQL Azure ersetzt.

SQL Server Dieser Begriff wird in folgenden Bedeutungen verwendet:

- Relationales Datenbanksystem, das auf der Sprache SQL basiert
- Relationales, SQL-basiertes Datenbanksystem der Firma Microsoft (genauer: Microsoft SQL Server)

SQL Server Compact Edition Kompakte Variante von Microsoft SQL Server, die sowohl auf mobilen Geräten (Pocket PCs und Smartphones auf Basis von Windows Mobile) als auch auf vollwertigen PCs einsetzbar ist.

SQL Server Management Studio Grafische Benutzeroberfläche zum Verwalten von einem oder mehreren SQL-Servern.

SQL Server Mobile Mobile Variante von Microsoft SQL Server, die mittlerweile durch die Compact Edition abgelöst wurde.

SQLCMD Kommandozeilentool zum Ausführen von einzelnen SQL-Befehlen oder ganzen SQL-Skripts.

SSAS (SQL Server Analysis Services) Serveranwendung von Microsoft, mit deren Hilfe Daten mehrdimensional modelliert und abgelegt werden können, um darüber effizientere Auswertungen erstellen zu können.

SSCE SQL Server für Windows CE (wurde mittlerweile durch die Compact Edition abgelöst).

SSDL (Storage Schema Definition Language) Schemadefinitionssprache, mit der das logische Modell (Datenmodell) für das Entity Framework definiert wird.

SSIS (SQL Server Integration Services) Aktuelles ETL-Tool von Microsoft, das seit SQL Server 2005 verfügbar ist.

SSMO siehe *SMO*

SSRS (SQL Server Reporting Services) Serveranwendung von Microsoft, mit der Berichte auf Basis von verschiedenen Datenquellen erstellt und für Anwender bereitgestellt werden können.

T-SQL (Transact-SQL) Microsofts Variante der Datenbanksprache SQL.

TCP/IP (Transmission Control Protocol/Internet Protocol) Betriebssystemunabhängige Familie von Netzwerkprotokollen, auf der auch das Internet basiert; Identifizierung von Rechnern erfolgt durch IP-Adressen.

Transaktion Gruppe von Operationen (z.B. SQL-Befehlen), die immer gemeinsam oder gar nicht ausgeführt werden. Tritt innerhalb einer Transaktion ein Fehler auf, wird automatisch ein Rollback ausgeführt, der alle Änderungen der Transaktion wieder rückgängig macht.

Trigger Variante einer gespeicherten Prozedur, die automatisch beim Auftreten gewisser Ereignisse gestartet wird.

URL (Unified Ressource Locator) Standard, der meist im Internet verwendet wird und beschreibt, wie eine Datei oder ein Server zusammen mit dessen Protokoll, Standort und Verzeichnis angegeben werden kann.

VIA (Virtual Interface Architecture) Modernes, von Microsoft, Intel und Compaq entwickeltes Netzwerkprotokoll, das auf geringe Prozessorlast und optimale Netzwerkauslastung ausgelegt ist.

Visual Basic .NET .NET-Variante der Programmiersprache Basic bzw. Visual Basic.

Visual C# .NET-Programmiersprache, die auf C++ basiert, allerdings verwalteten Code nutzt. Daher sind mit C# auch keine direkten Speicherzugriffe üblich.

Visual Studio Entwicklungsoberfläche von Microsoft, die verschiedene Programmiersprachen unterstützt.

Webservice Funktionalität, die über das Internet zur Verfügung gestellt wird und über eine fest definierte Schnittstelle nutzbar ist. Der Aufruf eines Webservices erfolgt – sofern die entsprechenden Referenzen hinterlegt sind – analog zu dem einer Prozedur oder Funktion.

XML (Extensible Markup Language) Herstellerübergreifendes Standardformat, mit dem Daten strukturiert abgelegt werden können.

Stichwortverzeichnis

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z