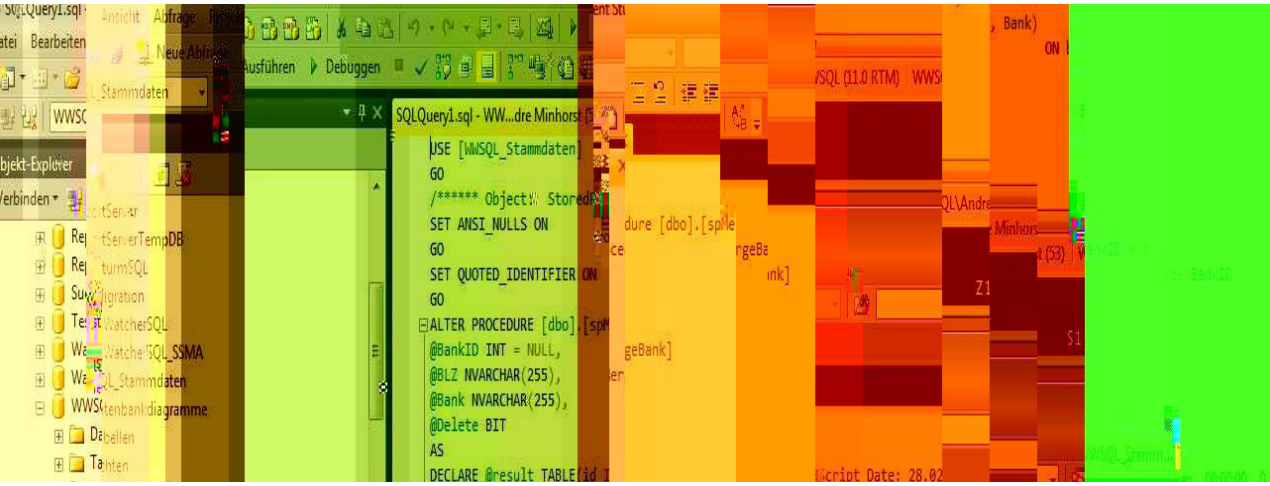
ACCESS UND SQL SERVER



MIGRATION UND ERSTELLUNG VON MEHRBENUTZERANWENDUNGEN

BERND JUNGBLUTH
ANDRÉ MINHORST

André Minhorst Verlag

Bernd Jungbluth
André Minhorst

Access und SQL Server

Migration und Erstellung von Mehrbenutzeranwendungen mit Access 2007-2013 und SQL Server 2012

Bernd Jungbluth, André Minhorst – Anwendungen entwickeln mit Access

ISBN 978-3-944216-01-3

Borkhofer Straße 17, 47137 Duisburg/Deutschland

1. Auflage 2013

Lektorat André Minhorst
Korrektur Rita Klingenstein
Cover/Titelbild André Minhorst
Typographie, Layout und Satz André Minhorst
Herstellung André Minhorst
Druck und Bindung Kösel, Krugzell (www.koeselbuch.de)

Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliographie. Detaillierte bibliografische Daten finden Sie im Internet unter http://dnb.d-nb.de.

Kein Anfang.
Kein Ende.

Inhalt

1 FAQ

Wer sich erstmalig mit der Migration einer reinen Access-Lösung hin zu einer Lösung bestehend aus einem Access-Frontend und einem SQL Server-Backend beschäftigt, steht vor einer Reihe wichtiger Entscheidungen. Dieses Buch wird Ihnen helfen, diese Entscheidungen nach dem aktuellen Stand der Technik richtig zu treffen. Während wir in den folgenden Kapiteln auf alle wichtigen Themen detailliert eingehen, möchten wir im vorliegenden Kapitel eine Zusammenfassung der Fragen liefern, mit denen Sie sich vor dem Start der Migration beschäftigen müssen.

1.1 Wann ist es Zeit, zu migrieren?

Es gibt verschiedene Gründe für den Wechsel von einem Access-Backend zu einem SQL Server-Backend:

» Zu viele Daten: Eine Access-Datenbank kann maximal 2 GB groß sein. Wenn Sie mehr Platz benötigen, bieten beispielsweise die Datenbanken unter der *SQL Server 2012 Express-Edition* bis zu 10 GB Platz. Mit den kostenpflichtigen Vollversionen erhalten Sie sogar bis zu 524 TB, also 10^{12} oder 1.000.000.000.000 Byte. Pro Datenbank wohlgemerkt – eine SQL Server-Instanz kann bis zu 32.767 Datenbanken verwalten und auf Ihrem Rechner können Sie maximal 50 Instanzen installieren.

» Zu viele Benutzer/Zugriffe: Access-Datenbanken sind nicht optimal auf den Mehrbenutzerbetrieb ausgelegt. Ab einer bestimmten Anzahl von Benutzern und Zugriffen nimmt die Performance und die Stabilität merklich ab. Konkrete Zahlen gibt es hierzu nicht; Sie werden feststellen, wann es soweit ist.

» Sicherheit: Enthält die Datenbank sensible Daten, ist ein Access-Backend definitiv der falsche Speicherort. Selbst mit dem mittlerweile (nicht ohne Grund) ausgelaufenen Sicherheitssystem von Access konnten Daten nicht zuverlässig geschützt werden.

» Flexibilität: Eine Verlagerung der Programmlogik in das SQL Server-Backend ist nicht nur ein Grund für eine bessere Performance, sondern es gestaltet auch die zukünftige Frontend-Entwicklung flexibler. Wo auch immer die Entwicklung im Frontend hingehen mag, das Backend ist selten von solchen Veränderungen betroffen. Ob Sie nun Ihre Applikation mit anderen Technologien ergänzen oder es komplett mit neuen Techniken ersetzen möchten, die Erweiterung oder die neue Technik verwenden die Komponenten vom SQL Server-Backend und somit die dort implementierte Logik. Aktuell stehen in der Access-Welt die ADP-Programmierer vor der Herausforderung, die implementierte Logik ihrer ADP (Access Data Project) in eine MDB zu portieren. Die Umstellung von ADP-Projekten, bei denen die Programmlogik im Access-Frontend realisiert wurde, ist weitaus aufwendiger, als bei ADP-Projekten, deren Programmlogik bereits im SQL Server liegt.

1.2 Wie migriere ich die Daten zum SQL Server?

Für eine Migration der Tabellen und Daten einer Access-Datenbank zu einer SQL Server-Datenbank gibt es mehrere Möglichkeiten. Die bekannteste dürfte der *Upsizing-Assistent* sein, der seit Access 2000 in Access als Menüpunkt zur Verfügung steht. Alternativ bietet Microsoft als kostenfreien Download den *SQL Server Migrations-Assistent* für Access an.

Eine weitere Variante ist die manuelle Migration. Hierbei legen Sie mit dem *SQL Server Management Studio* eine neue Datenbank an und definieren dort die Tabellen mitsamt Spalten und Datentypen, Einschränkungen, Standardwerten und referenzieller Integrität

Zum Abschluss importieren Sie die Daten aus der Access-Datenbank in die SQL Server-Datenbank mittels dem *SQL Server Import-/Export-Assistenten*. Diesen starten Sie entweder über den Eintrag *Daten importieren und exportieren* in der Programmgruppe *Microsoft SQL Server* oder im SQL Server Management Studio über das Kontextmenü der neuen Datenbank mit dem Eintrag *Tasks|Daten importieren*.

Mehr zu den jeweiligen Möglichkeiten einer Migration lesen Sie im Kapitel »Tabellen migrieren«, Seite 45.

1.3 .adp oder .mdb/.accdb?

Mit Access 2000 hat Microsoft die Access Data Projects (ADP) eingeführt. Hierbei handelt es sich um eine Access-Datenbankdatei, die die Tabellen, Sichten, gespeicherten Prozeduren und Funktionen einer SQL Server-Datenbank direkt per OLE DB nutzt.

Weitere Ausführungen zu den Vor- und Nachteilen dieser Technologie sind gar nicht nötig, denn: Microsoft hat schon vor einiger Zeit erklärt, dass die Verwendung einer *.mdb*-Datei mit entsprechenden ODBC-Verknüpfungen auf die Tabellen einer SQL Server-Datenbank die aktuell empfohlene Vorgehensweise ist.

Um dies zu unterstützen, wurden die Access-Projekte mit Access 2013 endgültig zu den Akten gelegt; ein Erstellen neuer oder das Öffnen bestehender *.adp*-Dateien ist nicht mehr möglich.

1.4 ADO oder DAO?

Die Antwort auf diese Frage lautet: DAO. Der Grund liegt nicht nur in der oben erwähnten Empfehlung von Microsoft, mit der auch DAO vor ADO propagiert wird, sondern insbesondere in der Abkündigung des Providers *Microsoft OLE DB Provider for SQL Server*.

Genau diesen verwendet ADO für den Zugriff auf SQL Server. Laut Angaben von Microsoft wird dieser Provider mit dem Ende des Supports von SQL Server 2012 im Sommer 2017 nicht mehr

unterstützt. Sie können zwar ADO auch über den Provider *Microsoft OLE DB Provider for ODBC Drivers* verwenden, wie der Name aber schon sagt, nutzen Sie hier letztendlich ODBC. Dafür bietet Ihnen dieser Provider jedoch die Möglichkeit, Ihren ADO-Code weiterhin zu nutzen. Für neuen Code jedoch ist DAO die bessere Variante.

1.5 Welche SQL Server-Version?

Aufgrund der kurzen Produktzyklen und den damit verbundenen Supportzeiten empfiehlt es sich, die jeweils aktuellste Version des SQL Servers zu verwenden – in diesem Fall SQL Server 2012. Die meisten der in diesem Buch beschriebenen Techniken funktionieren auch mit älteren Versionen. Sollte eine Vorgehensweise nicht mit älteren Versionen zusammenarbeiten, weisen wir darauf hin.

1.6 Welche SQL Server-Edition?

Den SQL Server gibt es in verschiedenen Editionen. Nachfolgend eine kurze Übersicht einiger aktueller Editionen und ihrer Eigenschaften:

» *SQL Server 2012 Express*: Kann beim Endkunden eingesetzt werden, unterliegt aber gewissen Einschränkungen. So ist sie auf vier Prozessorkerne und 1 GB Arbeitsspeicher begrenzt und erlaubt Datenbanken mit einer Größe von maximal 10 GB. Diese Version ist kostenlos.

» *SQL Server 2012 Express With Advanced Services*: Gleiche Einschränkungen wie *SQL Server Express*, allerdings unter anderem mit dem SQL Server Management Studio und Reporting Services ausgestattet. Wer seine Berichte also mit den Reporting Services generieren oder gar ein zentrales Berichtswesen etablieren möchte, sollte diese Version verwenden. Diese Version ist ebenfalls kostenlos.

» *SQL Server 2012 Enterprise/Business Intelligence/Standard/Web*: Nutzt maximal 16 Prozessorkerne und bis zu 64 GB Arbeitsspeicher; die *Enterprise Edition* sogar so viele Prozessorkerne und Arbeitsspeicher, wie das Betriebssystem hergibt. Jede Edition verwaltet Datenbanken bis zu einer Größe von 524 TB, was für die meisten zu migrierenden Access-Anwendungen ausreichen dürfte. Kostet eine Menge Geld – mehr Informationen liefert der Softwarehändler Ihres Vertrauens.

» *SQL Server 2012 Developer Edition*: Enthält alle Features der Vollversionen. Darf nur zu Entwicklungszwecken eingesetzt werden, kostet dafür aber weniger als 100 Euro.

Fazit: Der Kunde wird die Version einsetzen, die zu seinen Anforderungen und zu seinem Geldbeutel passt. Sie verwenden die SQL Server 2012 Developer Edition oder, falls ein entsprechendes Microsoft-Abonnement wie MSDN oder Technet vorhanden ist, eine der Vollversionen. Warum genau soll ich als Entwickler bis zu 100 Euro für die Developer Edition ausgeben, statt

eine der Express-Versionen zu verwenden? Weil die Developer Edition alle Funktionen der Enterprise Edition beinhaltet und Sie somit in den Möglichkeiten nicht eingeschränkt sind. Die Einschränkungen der Express Edition werden Sie spätestens dann ärgern, wenn Sie eine Datenbank für einen Kunden erstellen sollen, der eine Standard Edition oder eine noch höhere Edition im Einsatz hat.

Weitere Informationen finden Sie bei Microsoft auf einer Webseite mit dem Titel *Features Supported by the Editions of SQL Server 2012* (da sich die Links von Zeit zu Zeit ändern, geben wir diese hier nicht an).

1.7 Was ist der SQL Server?

Ein Windows-Dienst – nicht mehr und nicht weniger. Selbst das Administrationstool *SQL Server Management Studio* ist lediglich eine Client-Anwendung für den SQL Server.

Der Windows-Dienst *SQL Server* verarbeitet die Anforderungen der Clients an den Datenbankserver. Dieser Dienst empfängt also SELECT-Anweisungen vom Client, ermittelt die Daten und gibt das Ergebnis an den Client zurück. Oder er erhält den Auftrag Daten anzulegen, bestehende Daten zu ändern oder zu löschen.

Alle Aktionen erledigt SQL Server transaktional, weshalb seine Definition auch korrekt *transaktionaler Datenbankserver* (OLTP, Online Transactional Processing) lautet. Insgesamt kann der Windows-Dienst *SQL Server* 32.767 Zugriffe gleichzeitig verarbeiten.

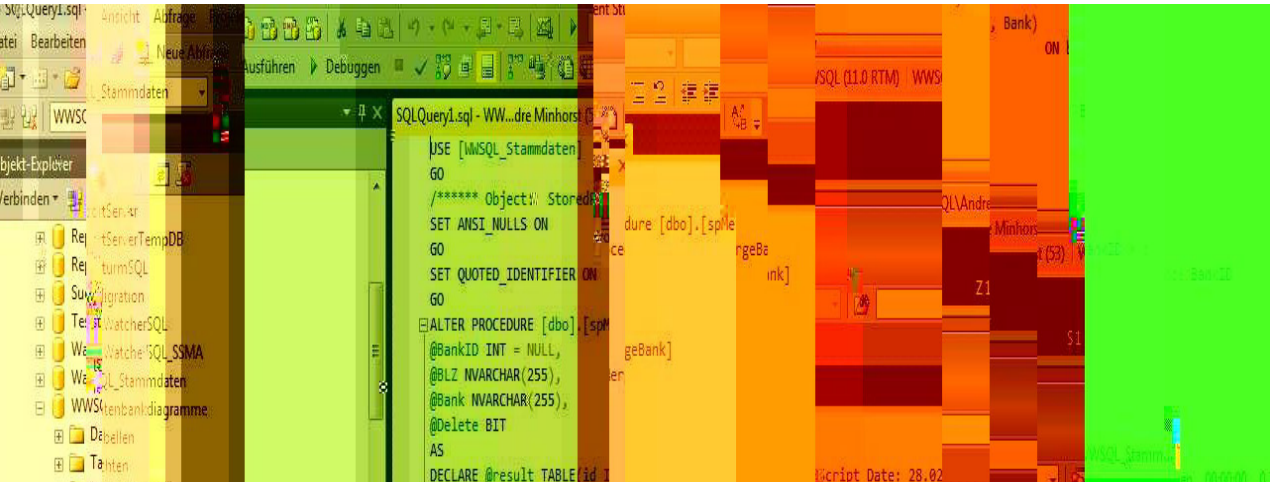
Übrigens ist *SQL Server* der technische Begriff für einen Datenbankserver. Oracle, MySQL und andere sind also ebenfalls SQL Server. Nur mit dem Unterschied, dass Microsoft sich den Namen als Produktnamen hat schützen lassen. Das Produkt *Microsoft SQL Server* umfasst jedoch inzwischen weitaus mehr als nur einen Datenbankserver. Im Lieferumfang ist unter anderem auch enthalten:

» *Reporting Services (SSRS):* Weit mehr als ein Berichtsgenerator. Mit SSRS lässt sich in Unternehmen ein zentrales Berichtswesen etablieren. Die Informationen stehen dabei in Form von interaktiven Berichten zur Verfügung.

» *Integration Services (SSIS):* Das ETL-Tool von Microsoft. Mit SSIS können Sie Daten automatisiert importieren, exportieren, transformieren, konsolidieren und integrieren. Nicht nur für den Betrieb von Datawarehouse-Systemen interessant, sondern auch für jegliche Art von Datenimporten und -exporten.

» *Analysis Services (SSAS):* Nicht OLTP sondern OLAP (Online Analytical Processing). Mit SSAS erstellen und verwalten Sie mehrdimensionale Datenbanken zur schnelleren Auswertung und Analyse von Unternehmensdaten. Ebenfalls in SSAS enthalten ist *Data Mining*, mit dem Sie Daten auf Muster, Beziehungen untereinander und statistische Auffälligkeiten untersuchen können.

... DIE ÜBRIGEN SEITEN GIBT'S GEDRUCKT UNTER SHOP.MINHORST.COM



BERND JUNGBLUTH
ANDRÉ MINHORST

André Minhorst Verlag

2 Installation

Als Entwickler installieren Sie den SQL Server am einfachsten direkt auf den Rechner, mit dem Sie auch entwickeln und auf dem sich auch die Access-Installation befindet. Alternativ können Sie diesen natürlich auch auf einem anderen Rechner im Netzwerk installieren oder auf einer virtuellen Maschine.

Es gibt lediglich eine Einschränkung, die sich auf die Entwicklung auswirken kann: Wenn Sie bei der Entwicklung mit der Windows-Authentifizierung arbeiten möchten, müssen Sie sich entweder direkt am Rechner mit dem SQL Server anmelden oder von einem Rechner, welcher der gleichen Domäne wie der SQL Server angehört. Viele Entwickler richten für sich selbst gar nicht erst eine Domäne ein, sondern verwenden eine Arbeisgruppe oder eine Heimnetzgruppe zur Vernetzung ihrer Rechner.

Sollten Sie den SQL Server auf einem anderen Rechner als auf Ihrem Arbeitsrechner installieren wollen, können Sie sich nur bei Vorhandensein einer Domäne mit der Windows-Authentifizierung beim SQL Server anmelden. Ansonsten bleibt nur die SQL Server-Authentifizierung.

Voraussetzungen

Die für die Installation von SQL Server 2012 notwendigen Voraussetzungen erfahren Sie im Internet in einem Artikel mit dem Titel *Hardware- und Softwareanforderungen für die Installation von SQL Server 2012*. Sollten Sie eine virtuelle Maschine verwenden, sei angemerkt, dass diese normalerweise mit einem kryptischen Rechnernamen daherkommen. Da die Standardinstanz des SQL Servers, also die erste (und gegebenenfalls einzige) Instanz über den Namen des Rechners angesprochen wird, sollten Sie den Rechnernamen entsprechend anpassen. Alle weiteren Instanzen beinhalten ebenfalls den Servernamen nach dem Schema *<MeinServer>\<MeineInstanz>*. Es ist also empfehlenswert, den Servernamen grundsätzlich anzupassen.

Damit weisen wir gleich an dieser Stelle darauf hin, dass Sie mehrere Instanzen des SQL Servers installieren können, die unabhängig voneinander arbeiten – dies kann beispielsweise sinnvoll sein, wenn Sie eine Produktiv- und eine Testinstanz betreiben möchten.

2.1 Start der Installation

Vermutlich liegt Ihnen entweder eine DVD mit den Installationsdateien vor oder Sie haben eine *.iso*-Datei mit dem Inhalt der DVD auf der Festplatte liegen. Die *.iso*-Datei brennen Sie entweder auf eine DVD oder Sie mounten die *.iso*-Datei. Das bedeutet, dass Sie mithilfe eines entsprechenden Tools ein virtuelles DVD-Laufwerk mit einem eigenen Laufwerkbuchstaben anlegen und so auf die Inhalte der *.iso*-Datei zugreifen können. Bei aktuelleren Windows-Versionen reicht ein Doppelklick auf die *.iso*-Datei, um den Inhalt anzuzeigen.

In jedem Falle klicken Sie nun doppelt auf die auf der DVD beziehungsweise in der *.iso*-Datei enthaltene Datei *Setup.exe*. Dies öffnet den Dialog aus Abbildung 2.1. Im hier markieren Bereich *Planung* können Sie zum Beispiel prüfen, ob das Zielsystem alle für die Installation nötigen Voraussetzungen erfüllt. Interessant ist hier der Eintrag *Systemkonfigurationsprüfung* – allerdings wird diese Funktion ohnehin ausgeführt, wenn Sie den SQL Server installieren.

Unbedingt empfehlenswert ist die Lektüre des Eintrags Sicherheitsdokumentation. Hier erhalten Sie einige Hinweise, wie Sie Sicherheitsrisiken rund um die SQL Server-Installation von vornherein reduzieren können.

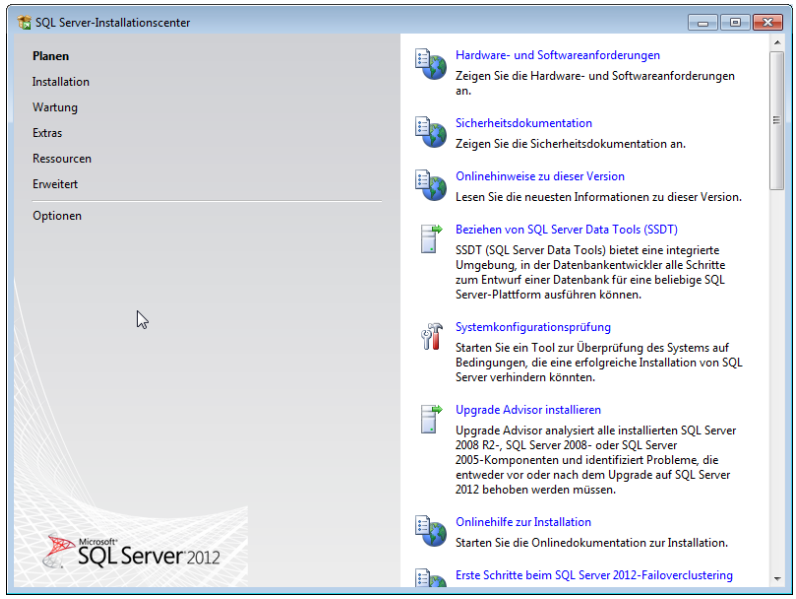


Abbildung 2.1: Das Installationscenter für den SQL Server, hier der Bereich *Planung*.

Vorab noch der Hinweis, dass der Setup-Assistent sich hier und da etwas Zeit lässt und man vermuten könnte, das nichts passiert – dies ist aber normalerweise nicht der Fall. Sind die Voraussetzungen erfüllt, klicken Sie hier auf den Eintrag *Installation*. Damit öffnen Sie den Dialog aus Abbildung 2.2, der wiederum einige Optionen anbietet. Wir wollen eine neue Installation anlegen, also wählen Sie hier den Eintrag *Neue eigenständige SQL Server-Installation oder Hinzufügen von Funktionen zu einer vorhandenen Installation* aus. Wie bereits angekündigt, prüft das Setup vor dem Start der eigentlichen Installation, ob der Zielrechner alle Voraussetzungen für die Installation des SQL Servers erfüllt. Es könnte sein, dass *Computer neu starten* als erforderlich markiert ist. Das kann auch nach einem Neustart wieder vorkommen. Eine Lösung hierzu gibt es unter *http://sqlfaq.de/blog/?p=217*.

Anschließend erscheint der Dialog aus Abbildung 2.3 und ermittelt den Product Key. Der folgende Dialog fragt wie üblich die Zustimmung der Lizenzbedingungen ab. Diese müssen Sie ange-

ben, damit die Installation fortgesetzt werden kann. Ob Sie Microsoft Daten zur Nutzung des SQL Servers schicken möchten, müssen Sie selbst entscheiden.

Abbildung 2.2: Installationsoptionen für den SQL Server

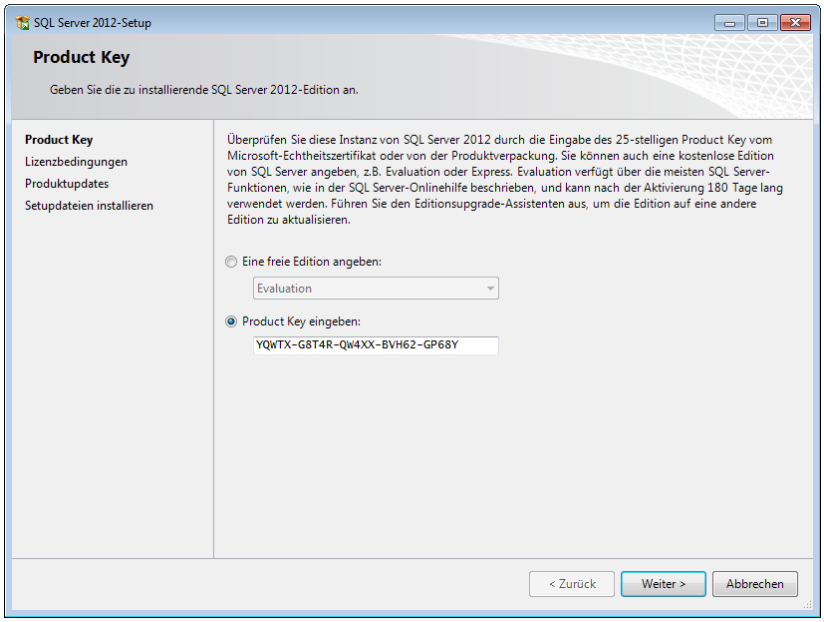


Abbildung 2.3: Abfrage des Product Keys

Im folgenden Schritt fragt der Assistent gegebenenfalls, ob Aktualisierungen, die seit Veröffentlichung der auf dem Installationsmedium enthaltenen Version erschienen sind, berücksichtigt werden sollen. Dies kann nicht schaden, daher stimmen Sie den Updates zu (siehe Abbildung 2.4).

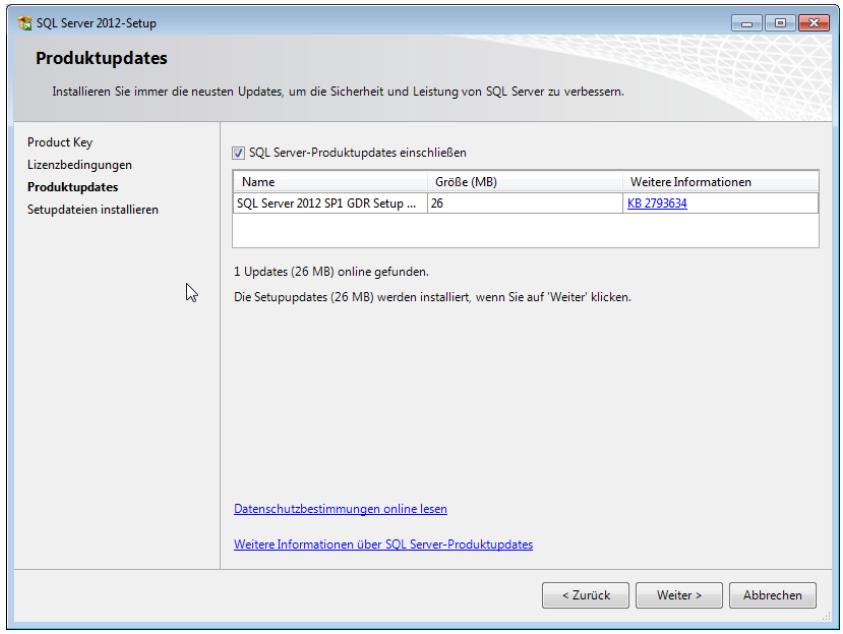


Abbildung 2.4: Nachfrage, ob Aktualisierungen berücksichtigt werden sollen

Anschließend installiert der Setup-Assistent die eigentlichen Setup-Dateien auf dem Zielrechner. Gegebenenfalls wird hier ein Update der Setup-Dateien installiert (siehe Abbildung 2.5).

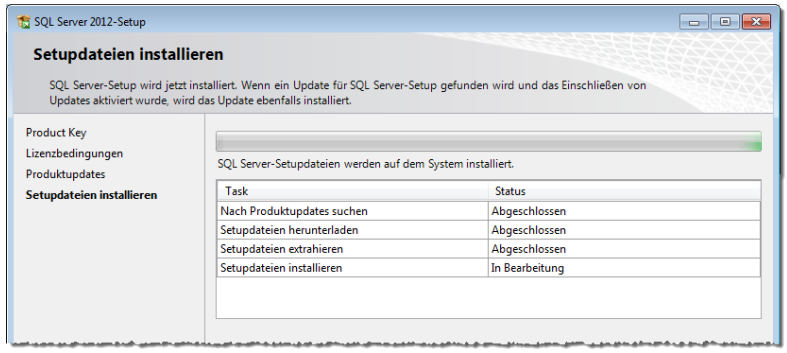


Abbildung 2.5: Installieren der Setup-Dateien

Nun folgen weitere vorbereitende Prüfungen – siehe Abbildung 2.6. Hier kommt es in der Regel vor, dass die aktivierte Windows Firewall angemeckert wird. Ein Klick auf den Link *Warnung* rechts neben dem Dialog zeigt eine Meldung an, die Einzelheiten liefert (siehe Abbildung 2.7). Diese Warnung verhindert in der Tat eine erfolgreiche Installation. Um einen sauberen Ablauf der Installation zu gewährleisten, deaktivieren Sie für den entsprechenden Zeitraum die Firewall.

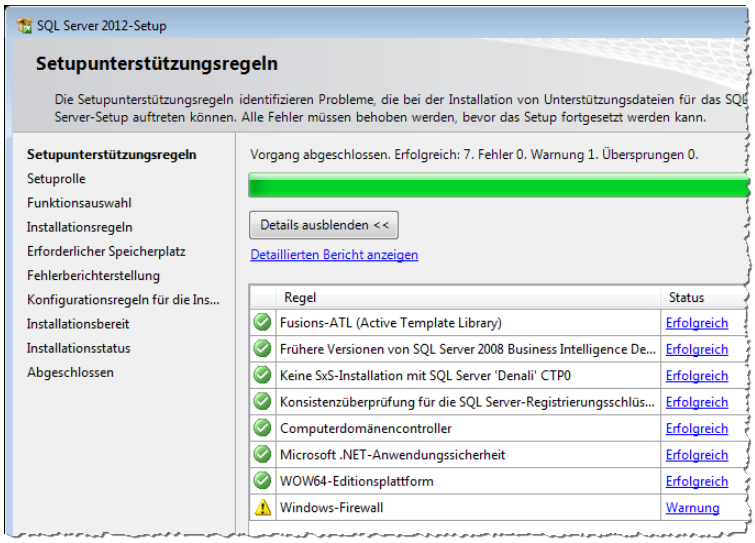


Abbildung 2.6: Es gibt Probleme mit der Windows Firewall.

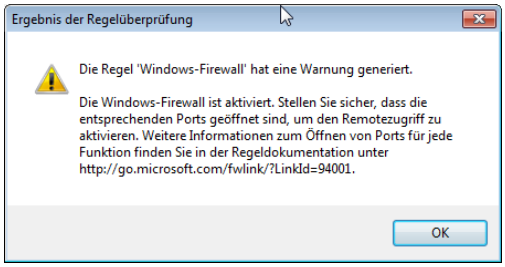


Abbildung 2.7: Die aktivierte Firewall von Windows könnte den Remotezugriff blockieren.

Um die Firewall zu deaktivieren, gehen Sie wie folgt vor:

» Öffnen der Systemsteuerung von Windows

» Auswählen von *System und Sicherheit*, dort auf *Windows-Firewall* klicken

» Im nun erscheinenden Dialog auf *Windows-Firewall ein- oder ausschalten* klicken.

» Dort unter *Standorteinstellungen für das öffentliche Netzwerk* die Option *Windows-Firewall deaktivieren (nicht empfohlen)* aktivieren (siehe Abbildung 2.8)

Im SQL Server-Setup können Sie die Prüfungen mit einem Klick auf Erneut ausführen wiederholen. Die Warnung bezüglich der Firewall erscheint nun nicht mehr.

Im nächsten Schritt erscheint der Dialog aus Abbildung 2.9. Hier wählen Sie den Eintrag *SQL Server-Funktionsinstallation* aus.

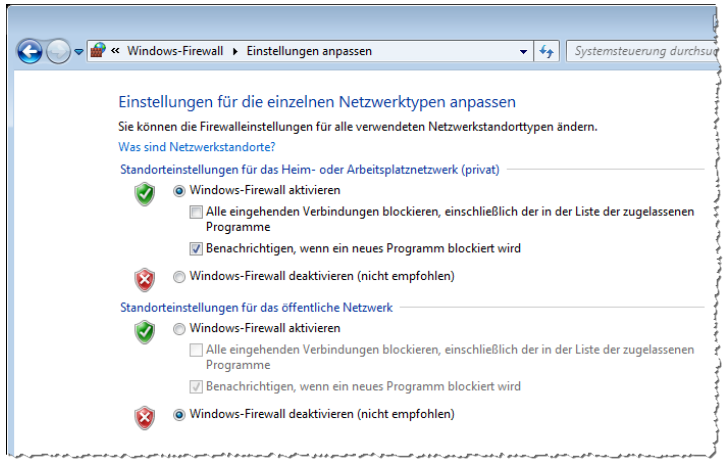


Abbildung 2.8: Deaktivieren der Windows Firewall, speziell der für das öffentliche Netzwerk

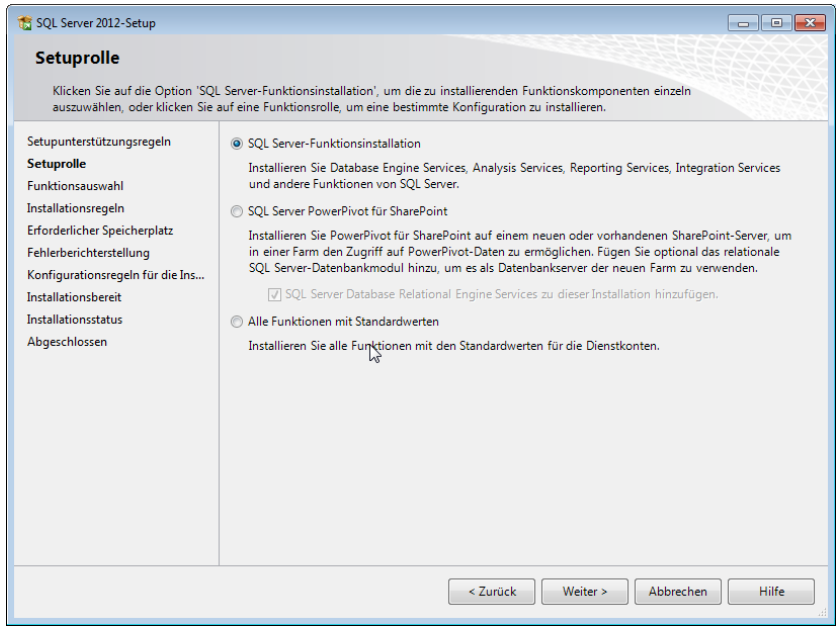


Abbildung 2.9: Auswahl zur Installation des SQL Servers

Einen Dialog weiter steht dann die detaillierte Auswahl der gewünschten Komponenten an. Hier wählen Sie folgende Einträge aus (ein Mausklick auf die Komponenten liefert weitere Informationen):

» Database Engine Services

» Reporting Services – Systemeigen (falls Sie Berichte mit den Reporting Services erstellen möchten – in diesem Buch werden Reporting Services aus Platzgründen nicht beschrieben)

» Konnektivität der Clienttools

» Abwärtskompatibilität der Clienttools

» Dokumentationskomponenten

» Verwaltungstools – Einfach (enthält unter anderem das SQL Server Management Studio)

» Verwaltungstools – Vollständig (enthält unter anderem Management Studio-Unterstützung für Reporting Services, SQL Server Profiler)

Neben den Komponenten wählen Sie hier auch noch den Zielort für die Installation aus. Hier können Sie die Standardeinstellung beibehalten (siehe Abbildung 2.10).

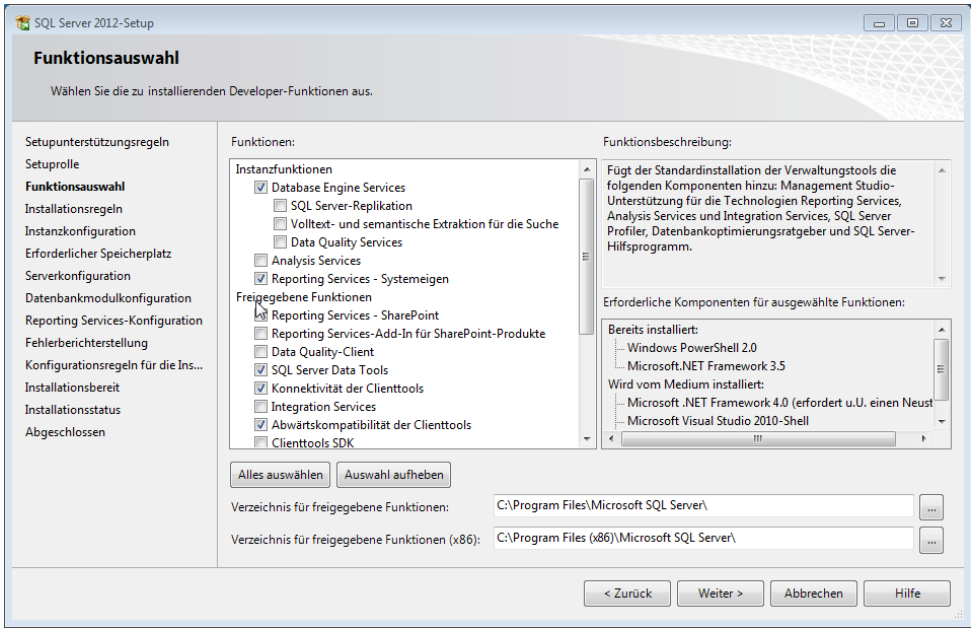


Abbildung 2.10: Festlegen der zu installierenden Komponenten und des Zielverzeichnisses

Nach dem Prüfen weiterer Voraussetzungen fragt der nächste Dialog ab, ob Sie eine Standardinstanz oder eine benannte Instanz installieren möchten. Letzteres ist sinnvoll, wenn Sie mehrere Instanzen des SQL Servers auf einem Rechner laufen lassen möchten. Anderenfalls belassen Sie die Einstellung bei der Standardinstanz (siehe Abbildung 2.11). Der Setup-Assistent stellt dann noch einmal die zuvor erfassten Informationen zusammen und lässt sich diese bestätigen. Danach legen Sie im Dialog aus Abbildung 2.12 fest, welcher Dienst auf welche Weise gestartet werden soll.

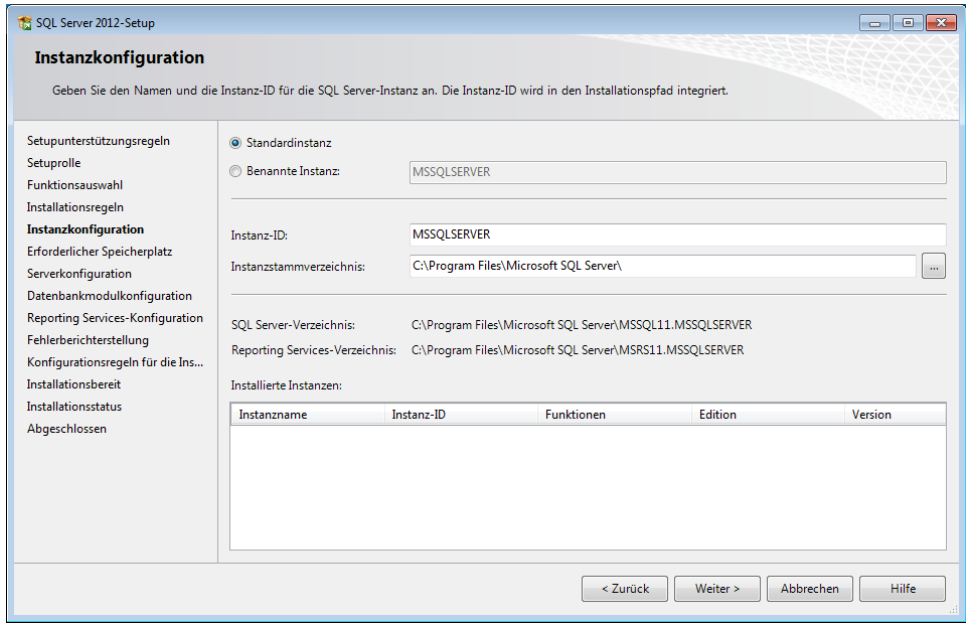


Abbildung 2.11: Konfiguration der SQL Server-Instanz

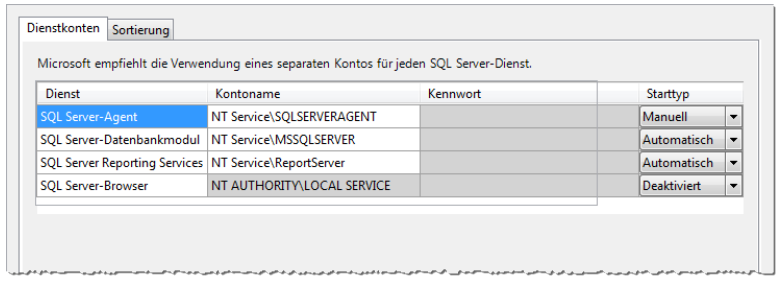


Abbildung 2.12: Einstellen des Starttyps der einzelnen Dienste

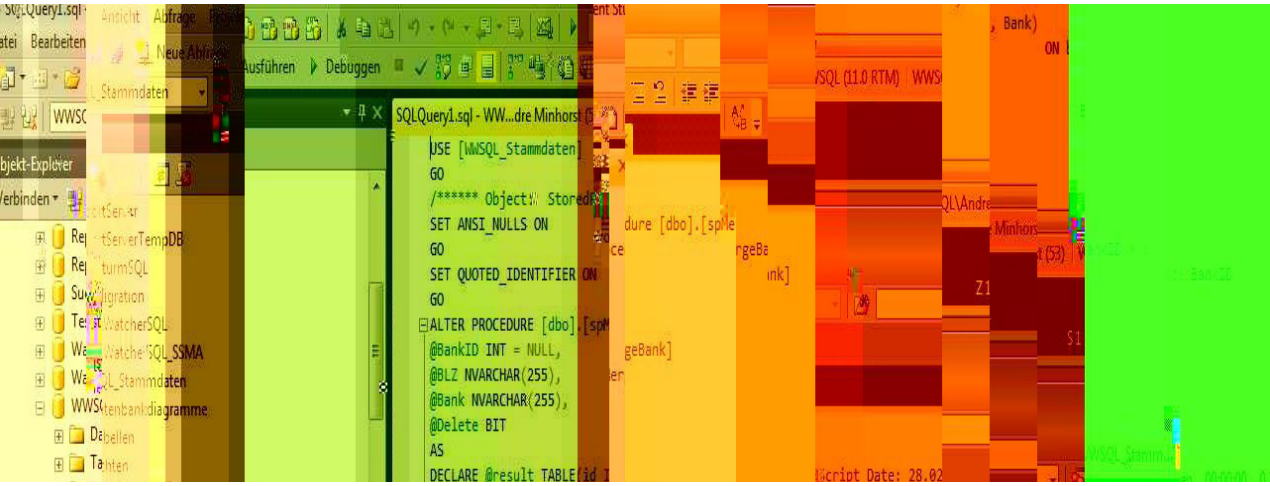
Die Dienste dieses Dialogs haben die folgenden Aufgaben:

» *SQL Server-Agent*: Ist eine Art Taskplaner des SQL Servers. Damit lassen sich beispielsweise Backups terminieren.

» *SQL Server-Datenbankmodul*: Dieser Dienst sorgt für das Speichern, Verarbeiten und Sichern der Daten, den Zugriff und die Transaktionsverarbeitung.

» *SQL Server Reporting Services*: Stellt ein zentrales Berichtswesen zur Ausgabe von unternehmensweiten Informationen in Form von Berichten zur Verfügung (wird nur angezeigt, sofern die Reporting Services weiteren oben zur Installation ausgewählt wurden).

... DIE ÜBRIGEN SEITEN GIBT'S GEDRUCKT UNTER SHOP.MINHORST.COM



BERND JUNGBLUTH
ANDRÉ MINHORST

André Minhorst Verlag

3 Tabellen migrieren

Die Migration eines Datenbank-Backends von einer Access-Datenbank zu einer SQL Server-Datenbank erfordert zunächst den Transfer der Tabellen aus der Access-Datenbank in die SQL Server-Datenbank. Anschließend wird der Rest der Anwendung angepasst – zumindest sollte dies der Fall sein.

Entgegen der weitläufigen Ansicht besteht eine Migration eben nicht nur aus dem Übertragen der Tabellen von Access nach SQL Server und dem anschließenden Einbinden der SQL Server-Tabellen ins Access-Frontend. Dieser Irrtum hat wohl seinen Ursprung in der einfachen Handhabung des Upsizing-Assistenten, der in den Access-Versionen 2000 bis 2010 integriert war.

Zwar dürfte die neue Access/SQL Server-Anwendung basierend auf den eingebundenen SQL Server-Tabellen bis auf wenige Ausnahmen genauso funktionieren wie vorher die reine Access-Anwendung, in den meisten Fällen aber mit einer schlechteren Performance. Dies ist mit ein Grund, warum die eigentliche Migration erst nach der Übertragung der Tabellen beginnt.

Um die Performance und um die Möglichkeit der Anbindung der Frontend-Anwendung an die Tabellen im Backend kümmern wir uns jedoch in späteren Kapiteln. Zunächst einmal schauen wir uns in diesem Kapitel die Migration der Tabellen an und im Anschluss daran im Kapitel *»Datenbanken und Tabellen erstellen«, Seite 155,* das Neuerstellen von Tabellen.

3.1 Sinn und Zweck der Migration

Access-Anwendungen entstehen aus den verschiedensten Gründen. Manchmal bemerkt ein Mitarbeiter, dass die Daten zu komplex sind, um sie in Excel-Tabellen zu verwalten. Oder er stellt fest, dass es zum Verknüpfen der Daten zweier Tabellen noch andere Möglichkeiten außer dem *SVERWEIS* geben muss. Dann hört er sich bei Kollegen um oder googelt im Web, um schließlich auf Access zu stoßen.

Dank Assistenten und Vorlagen hat Microsoft die Entwicklung von Datenbankanwendungen ja so einfach gemacht! Der Rest geht meist ganz schnell und die Excel-Daten fristen von nun an in einer Access-Datenbank ihr Dasein.

Manchmal bemerkt ein Mitarbeiter auch direkt, dass Excel nicht der richtige Ort zur Aufbewahrung seiner Daten ist und greift sofort zur Wunderwaffe unter den Datenbankanwendungen. Schnell ein paar Tabellen erstellt, Abfragen zur Datenauswahl und Formulare zur Dateneingabe gebaut und los geht's.

Doch eine Datenbankanwendung ist selten wirklich fertiggestellt. Zur Projektverwaltung kommt noch eine Zeiterfassung, diese zieht noch eine Mitarbeiterverwaltung nach sich und Rechnungen möchte man schließlich auch noch mit der Anwendung erstellen.

Kann wirklich nur ein Benutzer damit arbeiten? Nein! Die Anwendung wird auf den Server gelegt und ab jetzt greifen alle Mitarbeiter darauf zu und verwalten ihre Arbeitszeiten und Projekte damit.

Mit etwas Glück kennt der Entwickler, der die Datenbank ins Leben gerufen hat, sich etwas aus und teilt die Datenbank auf, sodass ein Backend auf dem Server liegt und jeder Benutzer ein eigenes Frontend auf der Festplatte hat.

Es greifen mehr und mehr Benutzer auf die Datenbank zu und es werden neue Features und somit neue Tabellen, Abfragen, Formulare, Berichte und VBA-Module hinzugefügt. Auch die Datenmenge wächst stetig. Es ist ein immerwährendes Wechselspiel zwischen dem steigendem Informationsbedarf an der Datenbank und dem steigendem Informationsgehalt in der Datenbank.

Schließlich ist die Anwendung ein wichtiges Instrument im Unternehmen geworden, ohne die die Bewältigung des Tagesgeschäfts undenkbar ist! Jede wichtige Information landet in der Datenbank, dementsprechend gelingt kaum ein Arbeitsschritt ohne Datenbankzugriff.

So weit so gut, denn gegen den Einsatz von Access-Datenbanken im Alltagsgeschäft gibt es nichts einzuwenden. Wenn denn die Access-Datenbank mit wachsender Anzahl von Benutzern und steigender Datenmenge nicht allmählich instabil werden würde. Der Grund für die Instabilität liegt in der Architektur einer Access-Datenbank.

3.1.1 Dateiserver vs. Client/Server

Access ist eine Desktop-Datenbank. Ob die Daten nun lokal von einem Benutzer oder im Netzwerk von vielen Benutzern geändert werden, jeder Benutzer ändert die Daten direkt in der Datei. In einem Mehrbenutzerbetrieb arbeiten also alle Benutzer gleichzeitig mit ein und derselben Datei – in unserem Fall mit ein und derselben Datenbank-Datei.

Aus diesem Grund ist mit Access lediglich ein Dateiserver-Betrieb möglich. Daran ändert auch das Aufteilen der Access-Datenbank in ein Front- und Backend nichts. Es mag zwar nach einer solchen Aufteilung jeder Benutzer auf seinem *Client* mit seinem Frontend arbeiten und alle Frontends greifen auf ein und dasselbe Backend zu, dass in der Regel auf einem *Server* gespeichert ist. Um einen *Client/Server-Betrieb* handelt es sich hierbei dennoch nicht. Auch diese Konstellation ist ein Dateiserver-Betrieb.

Bei einem tatsächlichen Client/Server-Betrieb werden die Anforderungen zur Datenermittlung oder Datenänderung vom Client an einen Serverdienst übergeben, der diese im Datenbank-Server umsetzt.

Tatsächlich ist auch der Microsoft SQL Server nichts anderes als ein Windows-Dienst. Dieser Windows-Dienst übernimmt die Anforderung – beispielsweise die SQL-Anweisung einer Abfrage – und führt diese aus. Bei einem Client/Server-Betrieb arbeiten also nicht mehrere Benutzer gleichzeitig mit der Datenbank-Datei, sondern lediglich nur ein einziger Windows-Dienst.

3.1.2 Performance

Der Unterschied zwischen einem Dateiserver-Betrieb und einem Client-/Server-Betrieb wirkt sich auf die Performance aus.

Bei einem Access-Frontend mit einem Access-Backend werden bei einer Abfrage, die auf mehreren Tabellen basiert, alle Daten dieser Tabellen als Kopie vom Backend über das Netzwerk an das Frontend übertragen.

Dort ermittelt Access das Ergebnis der Abfrage und zeigt die Daten an. Ändert der Benutzer die Daten, werden diese wieder über das Netzwerk in die Datenbank-Datei geschrieben.

Für eine solche Verarbeitung ist die Performance des Clients und des Netzwerks maßgebend. Das Netzwerk benötigt die entsprechende Bandbreite um die angeforderten Daten an die Clients zu übertragen und die Clients die entsprechende Rechenleistung zur Verarbeitung der Daten. Sie können also einen noch so schnellen Server verwenden – auf die Geschwindigkeit beim Zugriff auf die Daten im Access-Backend hat dies kaum einen Einfluss.

Arbeitet das Access-Frontend mit einer SQL Server-Datenbank, werden die SQL-Anweisungen vom Client an den SQL Server-Dienst übergeben. Je nach SQL-Anweisung ändert dieser die Daten direkt in der Datenbank oder er ermittelt die Daten und überträgt nur das Ergebnis an den Client.

Natürlich spielt auch in diesem Fall die Übertragungsgeschwindigkeit im Netzwerk immer noch eine Rolle, aber nun ist die Performance des Servers maßgebend. Der SQL Server ändert bzw. ermittelt die Daten und überträgt nur das Ergebnis an das Frontend. Dadurch übernimmt der Server die tatsächliche Arbeit und die Netzwerklast wird erheblich geringer. Die Nutzung der Server-Ressourcen und die Auslastung des Netzwerks werden verbessert.

3.1.3 Stabilität

Ein weiterer Unterschied zwischen einem Dateiserver-Betrieb und einem Client-/Server-Betrieb ist die Stabilität des Systems.

Da in einem Dateiserver-Betrieb mehrere Benutzer mit derselben Datenbank-Datei arbeiten, wird mit jedem weiteren Benutzer der Abgleich der Daten in der Access-Datenbank schwieriger. Ein Umstand, der das Risiko eines Datenverlustes oder einer Dateninkonsistenz erhöht. Dazu kommt, dass die Änderung der Daten in der Datenbank-Datei immer über das Netzwerk erfolgt. Gibt es beim Schreiben der Daten einen Fehler, kann dies ebenfalls ein Grund für eine Dateninkonsistenz sein. In einem solchen Fall hilft es oft nur, die Access-Datenbank zu reparieren und zu komprimieren – und zu hoffen.

Fehlerhafte Netzwerkübertragungen kann es natürlich auch bei einem Client-/Server-Betrieb geben. Doch hier ist die Datenkonsistenz nicht gefährdert. Zum einen erhält der SQL Server-Dienst keine Daten, sondern lediglich die entsprechenden SQL-Anweisungen zum Ändern der

Daten und zum anderen arbeitet der SQL Server mit Netz und doppeltem Boden – mit dem Transaktionsprotokoll. Jede Aktion wird im Transaktionsprotokoll aufgezeichnet und erst nach der Ausführung abgeschlossen. Sollte es zu einem Netzwerkfehler oder einer anderen Störung kommen, wird die aktuelle Aktion nicht abgeschlossen und die Datenbank anhand des Transaktionsprotokolls in Zustand vor der Aktion gesetzt. Mehr zum Transaktionsprotokoll lesen Sie im Kapitel *»Sichern und Wiederherstellen«, Seite 499*.

3.1.4 Ausfallsicherheit

Eine inkonsistente oder gar defekte Access-Datenbank lässt sich durch *Reparieren und Komprimieren* eventuell wieder in einen funktionsfähigen Zustand versetzen. Kann der Schaden hiermit jedoch nicht behoben werden, bleibt nur die Wiederherstellung der letzten Sicherung. Dies bedeutet in den meisten Fällen einen Datenverlust vom Zeitpunkt der letzten Sicherung bis zum Zeitpunkt des Problems.

Wie jede andere Datei eines Dateiservers wird auch die Access-Datenbank von einer entsprechenden Sicherungsoftware gesichert – und diese Sicherungen erfolgen in der Regel nachts. Eine Sicherung der Access-Datenbank tagsüber ist eher selten der Fall. Alleine schon aus dem Grund, da sich eine Access-Datenbank im laufenden Betrieb nicht sichern lässt.

Dies ist beim SQL Server anders. Er lässt nicht nur Sicherungen im laufenden Betrieb zu, er bietet direkt auch eigene Funktionen zur Ausfallsicherheit an. Ein SQL Server ist darauf ausgelegt, dass die Datenbanken permanent zur Verfügung stehen. Die Anforderungen an einen – wie es in SQL Server 2012 heißt – *AlwaysOn*-Betrieb werden erfüllt.

Hierfür stellt der SQL Server nicht nur eigene Funktionen zur Datenbanksicherung zur Verfügung, sondern auch andere Methoden zur Ausfallsicherheit wie Datenbankspiegelung, Transaktionsprotokollversand und Failover-Cluster.

Zur reinen Datenbanksicherung bietet SQL Server drei verschiedene kombinierbare Varianten: die Vollsicherung, die differenzielle Sicherung und die Transaktionsprotokollsicherung. Diese Varianten sind Inhalt des Kapitels *»Sichern und Wiederherstellen«, Seite 499*. An dieser Stelle sei nur kurz erwähnt, dass sich eine SQL Server-Datenbank je nach Sicherungskonzept auch tagsüber mit so gut wie keinem Datenverlust wiederherstellen lässt.

3.1.5 Zugriffsschutz

Ist Ihre Access-Datenbank vor fremden Zugriff sicher? Verwenden Sie ein Datenbank-Kennwort oder eine Arbeitsgruppen-Informationsdatei? Letztere lässt sich eh nur beim Datenbankformat MDB verwenden. ACCDB unterstützt nur noch das Datenbank-Kennwort.

Mit Access 2007 wurden also die sowieso schon spärlichen Möglichkeiten eines Zugriffsschutzes um 50% reduziert. Für die verbleibenden 50% – das Datenbank-Kennwort – kursieren bereits seit Jahren entsprechende Tools im Internet, um auch diesen Zugriffschutz zu umgehen.

SQL Server bietet eine mehrstufige Sicherheitsarchitektur, die zwischen Authentifizierung und Autorisierung unterscheidet. Für jeden Zugriff auf die Daten einer Datenbank findet zunächst eine Authentifizierung des Benutzers am SQL Server statt.

Erst wenn diese bestanden ist, folgt die Autorisierung des Benutzers an der Datenbank mit den ihm dort zugewiesenen Rechten. Mehr zur Sicherheitsarchitektur und wie Sie Ihre Daten vor fremden Zugriff schützen, lesen Sie im Kapitel *»Sicherheit und Benutzerverwaltung«, Seite 395*.

3.1.6 Skalierbarkeit

Die maximale Größe einer Access-Datenbank beträgt 2 Gigabyte. Diese Grenze ist beim heutigen Umgang mit Daten schnell erreicht. Dazu kommt, dass die Anzahl der Benutzer einer Access-Datenbank theoretisch auf 255 Benutzer begrenzt ist. Wobei die Obergrenze für eine einigermaßen performante Access-Datenbank eher bei 20 Benutzern liegen dürfte.

Die Grenzen beim SQL Server sehen da schon etwas anders aus. Ein SQL Server kann maximal 32.767 Benutzerverbindungen verwalten. In der kostenlosen Variante – der SQL Server Express Edition – sind bis zur Version 2008 maximal 5 Gigabyte Daten möglich, seit der Version 2008 R2 bis zu 10 Gigabyte. Die Grenze bei allen kostenpflichtigen Versionen liegt bei 542.272 Terrabyte – pro Datenbank wohlgemerkt. Eine SQL Server-Instanz kann 32.767 Datenbanken verwalten und auf einem Server können Sie wiederum 50 SQL Server-Instanzen installieren. Der Unterschied zu einer Access-Datenbank ist alleine durch diese Zahlen deutlich erkennbar.

3.1.7 Flexibilität

Der letzte Grund für eine Migration ist kein technischer, sondern vielmehr ein konzeptioneller.

In Access realisieren Sie die Programmlogik zur Verwaltung der Daten mit Formularen, Berichten, Abfragen, Makros und Modulen. Diese Objekte und somit auch die Logik stehen jedoch nur in Access zur Verfügung. Weder die Formulare, noch die Berichte, Abfragen, Makros oder Module sind eigenständig und können in anderen Applikationen wiederverwendet werden – mühsames Kopieren und Einfügen von VBA-Code mal ausgenommen.

Dieser Umstand schränkt die Flexibilität zur Verarbeitung der Daten stark ein. Sollen bspw. die Daten der Access-Datenbank parallel mit einer Web-Applikation gepflegt werden, müssen Sie die Programmlogik in der Web-Applikation erneut umsetzen. Dies betrifft auch zukünftige Änderungen beider Applikationen – jede Änderung und Erweiterung müssen Sie zweimal programmieren. Eine redundante Programmierarbeit, die nicht nur zeitintensiv, sondern auch fehleranfällig ist.

Verlagern Sie jedoch die Programmlogik zum SQL Server und bilden diese dort mit den Datenbankobjekten Sichten, Gespeicherten Prozeduren, Funktionen und Trigger ab, sind Sie bei der Verwendung der Frontends flexibler. Die Frontends nutzen lediglich die jeweiligen Datenbankobjekte zum Ermitteln, Ändern, Löschen oder Hinzufügen von Daten. Sollte sich die Programm-

logik ändern, sind nur die entsprechenden Datenbankobjekte anzupassen. Die Frontends selbst bleiben von diesen Änderungen weitesgehend unberührt.

Durch einen konsequenten Einsatz der Datenbankobjekte gehen alle Datenermittlungen und Datenänderungen immer den gemäß der Geschäftsregeln vorgeschriebenen Weg – unabhängig vom jeweiligen Frontend.

Mit einer stringenten Berechtigungsvergabe können Sie sogar direkte Datenänderungen in den Tabellen sperren, so dass Datenermittlungen und Datenänderungen nur über die in den Datenbankobjekten definierte Programmlogik und somit entsprechend der vorgeschriebenen Geschäftsregeln möglich sind.

3.1.8 Gründe für eine Migration

Fassen wir also die Gründe für eine Migration nochmals zusammen:

» Performance

» Stabilität

» Ausfallsicherheit

» Zugriffschutz

» Skalierbarkeit

» Flexibilität

Einige gute Gründe für eine Migration. Wie nun aber die Tabellen von Access zum SQL Server übertragen werden können, zeigen die nächsten Abschnitte.

3.2 Die Migration

In den nachfolgenden Abschnitten erfahren Sie, wie Sie die Beispieldatenbank per Assistent migrieren und wie Sie das Ergebnis prüfen. Dazu müssen Sie an dieser Stelle nichts weiter vorbereiten.

In der Realität jedoch sollten Sie eine Migration nicht von jetzt auf gleich durchführen. Selbstverständlich planen Sie eine Migration sorgfältig und prüfen die verschiedenen Assistenten oder gar die Variante einer manuellen Migration. Abhängig vom Ergebnis dieser Prüfung legen Sie die Vorgehensweise der Migration fest und führen diese mit einer Kopie Ihrer Access-Datenbank testweise durch.

Schauen Sie sich das Ergebnis der Testmigration genau an. Prüfen Sie, ob alle Tabellen und auch alle Daten migriert wurden, sowie welche Fehler bei der Migration aufgetreten sind oder was der jeweilige Assistent schlicht und ergreifend nicht migrieren konnte.

Anschließend folgt der Test des Abfrageverhaltens Ihrer Access/SQL Server-Applikation – und das unter realistischen Bedingungen.

Dies bedeutet, dass Sie den Test nicht alleine durchführen, sondern mit mehreren Benutzern, um eine realistisches Mehrbenutzer-Szenario zu erhalten. Erst hierdurch werden Sie das tatsächliche Abfrageverhalten einschätzen können.

Sind Sie mit der Testmigration und dem Abfrageverhalten zufrieden, folgt die tatsächliche Migration Ihrer Access-Datenbank.

3.3 Beispieltabellen für die Migration

Migriert wird das Backend *AEMA_BE.accdb* der Beispieldatenbank zum Buch *Anwendungen entwickeln mit Access*, dessen Datenmodell Sie in Abbildung 3.1 finden.

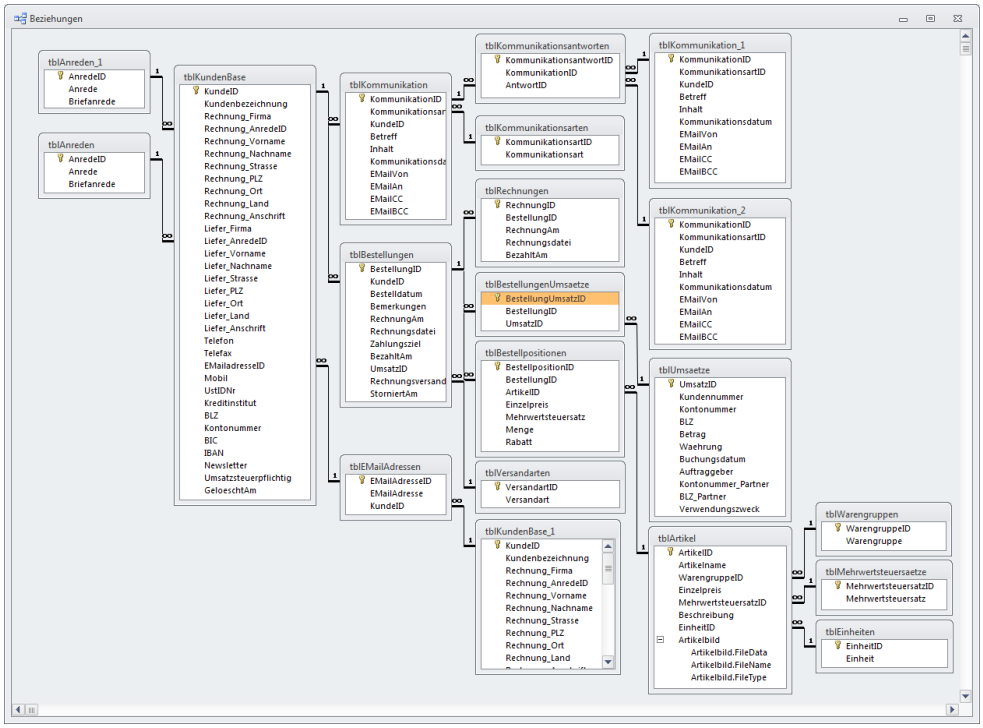


Abbildung 3.1: Datenmodell der Beispieldatenbank

Eine Erklärung des Datenmodells sparen wir uns an dieser Stelle. Das es sich um ein komplexes Datenmodell handelt, ist auch ohne Erklärung direkt ersichtlich. Für die Migration selbst ist das Datenbank-Design aber ohnehin nicht relevant.

3.4 Upsizing-Assistent

Die bekannteste Methode zur Migration einer Access-Datenbank zu einer SQL Server-Datenbank ist der in Access integrierte *Upsizing-Assistent*. Diesen gibt es bereits seit Access 97, damals noch als eigenständige Applikation. In Access integriert ist der Upsizing-Assistent seit Access 2000.

Mit Access 2000 wurden auch die *Access Data Projects* – kurz *ADP* – eingeführt. ADP ist bzw. war ein eigenes Datenbankformat in Access zur direkten Verwaltung der Daten einer SQL Server-Datenbank – ohne eingebundene Tabellen und sogar ohne Jet-Engine.

Seit Access 2013 ist das ADP-Format nicht mehr in Access enthalten. Ebenso gibt es dort auch den Upsizing-Assistenten nicht mehr. Der Upsizing-Assistent ist also Geschichte. Aus diesem Grund werden wir nicht weiter auf den Upsizing-Assistenten eingehen, sondern uns auf den seit 2005 verfügbaren *SQL Server Migrations-Assistenten für Access* konzentrieren.

3.5 SQL Server Migrations-Assistent für Access

Der *SQL Server Migrations-Assistent für Access* – kurz *SSMA-A* – ist eine eigenständige Applikation, die Microsoft als kostenfreien Download zur Verfügung stellt. Vor der ersten Verwendung ist zwar eine Lizenzierung notwendig, aber auch diese ist kostenfrei.

Mit dem SQL Server Migrations-Assistenten können verschiedene Datenbanken zu SQL Server migriert werden. Es gibt eine Variante für die Migration von MySQL-Datenbanken, von Sybase-Datenbanken, von Oracle-Datenbanken und natürlich auch von Access-Datenbanken.

Diese Vielfältigkeit ist ein Grund für die Architektur vom SQL Server Migrations-Assistenten. Denn er ist weitaus mehr als nur ein Assistent. Mit SSMA-A erstellen Sie ein Migrationsprojekt, dass Sie in drei Schritten durchführen können.

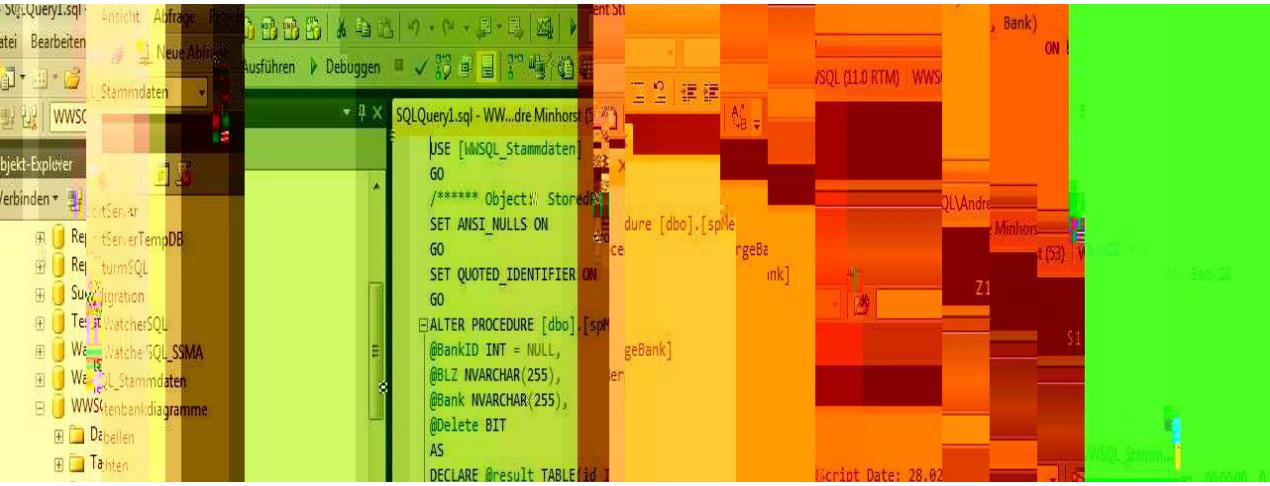
» Schritt 1 erstellt anhand der Access-Datenbank ein SQL-Skript zur Erstellung von SQL Server-Tabellen und Sichten.

» Schritt 2 führt dieses Skript in der SQL Server-Datenbank aus und legt die Tabellen und Sichten an.

» Schritt 3 migriert die Daten aus den Tabellen der Access-Datenbank in Tabellen der SQL Server-Datenbank.

Sie können nach jedem Schritt das Projekt speichern und mit der Migration zu einem späteren Zeitpunkt fortfahren. Ebenso ist es möglich, nur einen Teil der Schritte durchzuführen und den Rest über einen anderen Weg zu realisieren. So wäre es denkbar, mit Schritt 1 lediglich das SQL-Skript zu erstellen. Anschließend passen Sie das SQL-Skript an Ihre Anforderungen an und führen es manuell in der entsprechenden SQL Server-Datenbank aus, um darauf mittels Schritt 3 die Daten zu migrieren.

... DIE ÜBRIGEN SEITEN GIBT'S GEDRUCKT UNTER SHOP.MINHORST.COM



BERND JUNGBLUTH
ANDRÉ MINHORST

André Minhorst Verlag

4 Tabellen verknüpfen

Der Betrieb einer Access-Anwendung auf Basis einer SQL Server-Datenbank steht und fällt mit dem Zugriff auf die Daten der SQL Server-Datenbank. Dieses Kapitel zeigt verschiedene Möglichkeiten, von Access aus über die Objekttypen Tabelle oder Abfrage auf die Daten in den Tabellen des SQL Servers zuzugreifen.

Beispieldatenbank

Die Beispieldatenbank zu diesem Kapitel heißt *ASQL_Verknuepfungen.accdb*.

4.1 Tabellen manuell verknüpfen

Die wohl bekannteste Möglichkeit ist der Zugriff auf eine SQL Server-Tabelle mittels einer ODBC-Verknüpfung. Unter Access 2010 sehen die dazu nötigen Schritte wie folgt aus – unter anderen aktuellen Access-Versionen geschieht dies auf ähnliche Weise. Beim Verknüpfen mit dem entsprechenden Assistenten von Access gibt es im Wesentlichen zwei Varianten:

» Verknüpfung mit Datei-DSN

» Verknüpfung mit System-DSN

Wir erläutern zunächst die Vorgehensweise zum Erstellen einer Verknüpfung über eine Datei-DSN. Anschließend gehen wir auf die System-DSN und weitere Methoden ein. Starten Sie den entsprechenden Assistenten, und zwar über den Ribbon-Eintrag *Externe Daten|Importieren und Verknüpfen|ODBC-Datenbank* (siehe Abbildung 4.1).

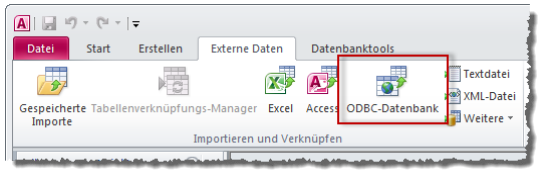


Abbildung 4.1: Aufruf des Assistenten zum Verknüpfen von ODBC-Datenbanken

Im folgenden Dialog geben Sie an, dass Sie eine Verknüpfung erstellen möchten (siehe Abbildung 4.2). Schließlich wollen wir jederzeit die aktuellen Daten vorfinden. Gelegentlich benötigen Sie vielleicht auch einmal einen Import vom SQL Server, aber in diesem Fall nicht.

Als nächstes legen Sie fest, aus welcher Datenquelle die Daten stammen. Der nun erscheinende Dialog verwirrt beim ersten Einsatz mehr als das er hilft (siehe Abbildung 4.3). Als erstes stellt sich die Frage, ob Sie eine Dateidatenquelle oder eine Computerdatenquelle benötigen.

Der Unterschied ist folgender: Eine Dateidatenquelle enthält die für die Herstellung der Verbindung nötigen Informationen in einer Datei. Eine Computerdatenquelle hingegen speichert die Verbindungsdaten in der Registry. Um eine Computerdatenquelle zu erstellen, müssen Sie Administratorrechte besitzen.

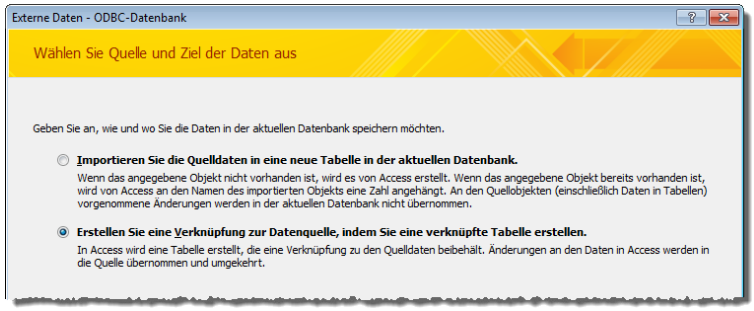


Abbildung 4.2: Importieren oder verknüpfen?

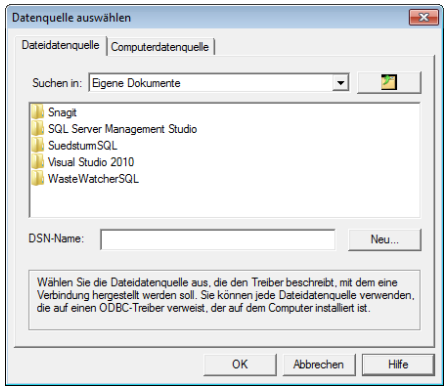


Abbildung 4.3: Auswählen oder Erstellen einer Datenquelle

4.1.1 Verknüpfung mit Dateidatenquelle erstellen (Datei-DSN)

Wir erstellen zunächst eine Verknüpfung mithilfe einer Dateidatenquelle. Bleiben Sie auf der Registerkarte *Dateidatenquelle* und klicken Sie auf die Schaltfläche *Neu*.

Treiber auswählen

Nach der Entscheidung, eine Dateidatenquelle zu erstellen, wählen Sie im nächsten Schritt den gewünschten Treiber aus. Wenn Sie im Dialog aus Abbildung 4.4 ganz nach unten scrollen, finden Sie dort zwei passende Einträge – *SQL Server* und *SQL Server Native Client x.0*, wobei x für die jeweils aktuelle Version des SQL Servers steht. Der Eintrag *SQL Server* wird von Windows hinzugefügt, der Eintrag *SQL Server Native Client x.0* vom SQL Server selbst. Der Unterschied ist,

dass der *SQL Server Native Client x.0* sich auf die jeweils installierte Version des SQL Servers bezieht und somit gegebenenfalls Features unterstützt, die der einfache *SQL Server*-Treiber nicht bietet. Wählen Sie also bei Verfügbarkeit den *SQL Server Native Client x.0*.

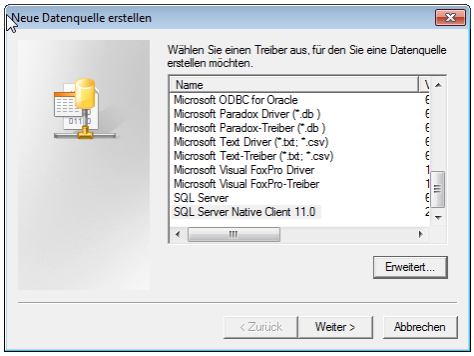


Abbildung 4.4: Auswahl eines Treibers

Sollten Sie von einem anderen Rechner auf den SQL Server zugreifen, ist der *SQL Server Native Client x.0* möglicherweise nicht installiert. Sie können diesen bei Microsoft herunterladen und nachinstallieren. Im nächsten Schritt legen Sie dann den Namen der Datenquelle fest – der Einfachheit halber wählen wir den Namen, den wir auch für die SQL Server-Datenbank vergeben haben (siehe Abbildung 4.5).

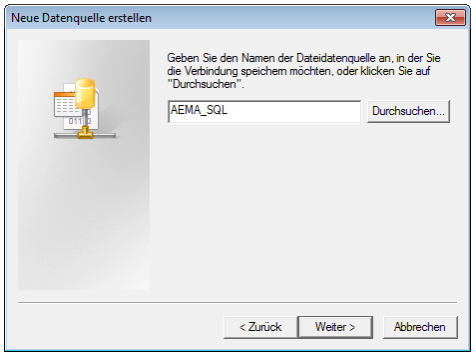


Abbildung 4.5: Name der Datenquelle festlegen

Im folgenden Schritt schließen Sie die Erstellung der Dateidatenquelle bereits ab – zumindest was diesen Teil des Assistenten angeht.

Nun geht es mit dem datenquellenspezifischen Teil weiter. Der Dialog aus Abbildung 4.6 fragt die SQL Server-Instanz ab, welche die betroffene Datenbank enthält. Das Aufklappen des Kombinationsfeldes dauert etwas, da der Assistent das Netzwerk nach SQL Server-Instanzen durchsucht. Sie tun gut daran, den Namen des SQL Servers zu kennen und diesen direkt einzutippen –

das spart auf Dauer einiges an Zeit. Befindet sich der SQL Server auf dem gleichen Rechner, wählen Sie *(local)* aus, sonst geben Sie den Namen des Server an (in diesem Beispiel *ASQL*).

Sollten Sie mehrere Instanzen des SQL Servers auf dem Zielserver betreiben, geben Sie den Server nach dem Schema *<Servername>\<Instanzname>* an.

Bei der Express Edition verwenden Sie immer den Instanznamen *<Servername>\SQLEXPRESS*.

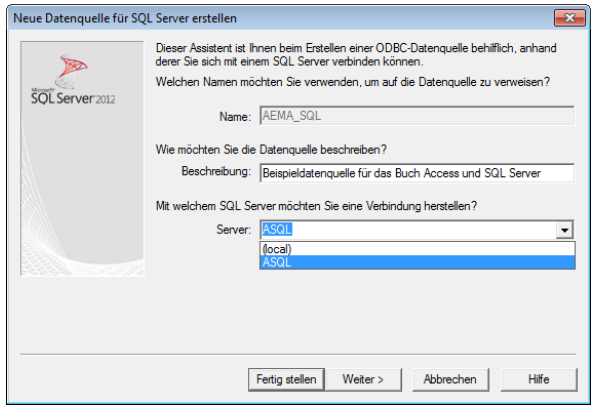


Abbildung 4.6: SQL Server auswählen

Der folgende Dialog gibt Ihnen dann die Möglichkeit, die Authentifizierungsmethode festzulegen (siehe Abbildung 4.7). Mehr zu den Authentifizierungsmethoden erfahren Sie unter *»Anmeldung mit SQL Server-Authentifizierung«, Seite 436*.

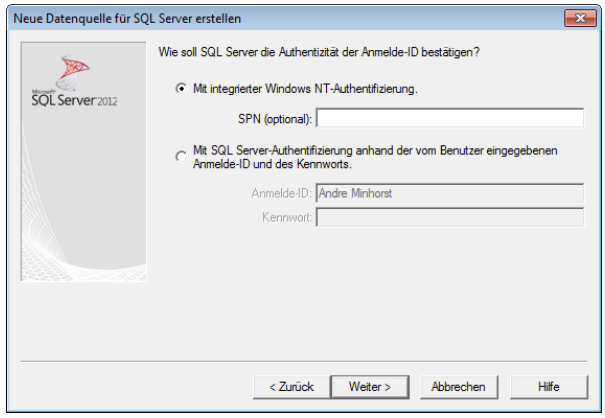


Abbildung 4.7: Art der Authentifizierung

Anschließend legen Sie die Standarddatenbank fest – das ist die Datenbank, deren Tabellen in ein paar Schritten zum Verknüpfen angeboten werden (siehe Abbildung 4.8).

Die Einstellungen des folgenden Dialogs können Sie beibehalten. Schließlich können Sie die Verbindung testen (siehe Abbildung 4.9).

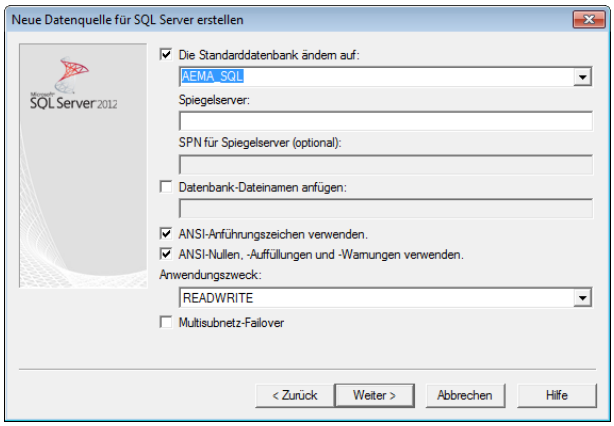


Abbildung 4.8: Ändern der Standarddatenbank

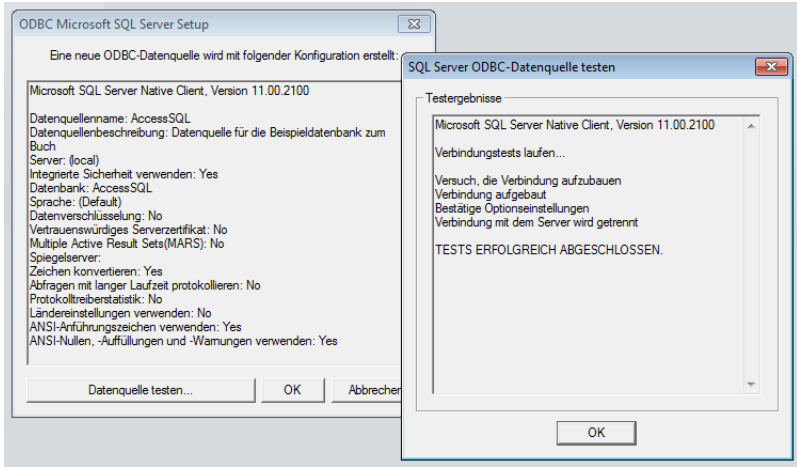


Abbildung 4.9: Testen der Datenquelle

Klicken Sie dann auf *OK*. Der zu Beginn gestartete Assistent zeigt nun die neue Dateidatenquelle in der Liste an (siehe Abbildung 4.10). Wählen Sie diese aus und klicken Sie auf *OK*.

Damit können Sie nun endlich auf die Tabellen und weitere Objekte der Quelldatenbank zugreifen und die zu verknüpfenden Objekte auswählen (siehe Abbildung 4.11).

Nach der Auswahl etwa der Tabelle *tblBanken* zeigt Access diese mit einem speziellen Symbol versehen im Navigationsbereich an. Sie können nun bereits direkt auf die enthaltenen Daten zugreifen (siehe Abbildung 4.12).

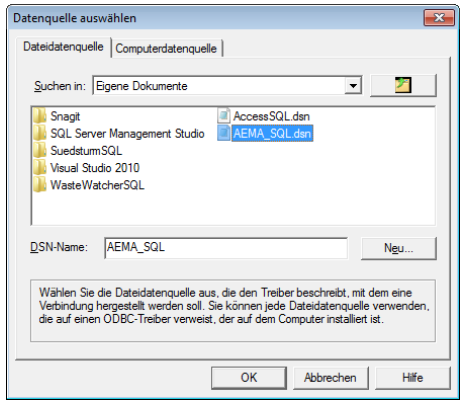


Abbildung 4.10: Auswahl der soeben erstellten Dateidatenquelle

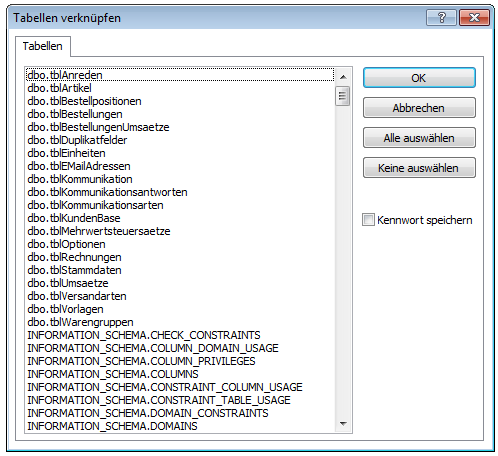


Abbildung 4.11: Auswahl der zu verknüpfenden Objekte

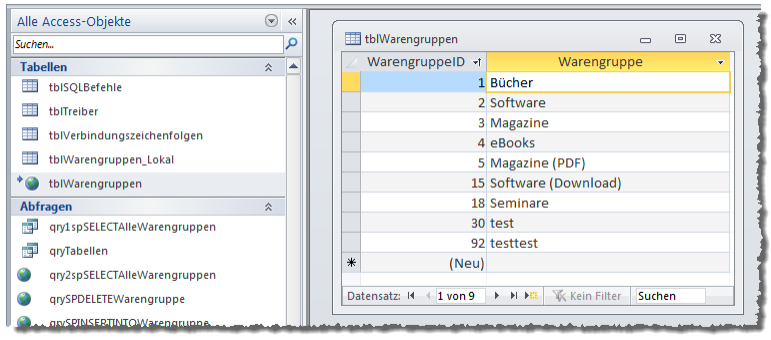


Abbildung 4.12: Eine per ODBC verknüpfte Tabelle

DSN-Datei ansehen

Was für Daten wurden nun in der DSN-Datei gespeichert? Im Prinzip handelt es sich dabei um eine schlichte Zeichenkette. Wenn es Sie interessiert, wie nun die erzeugte Dateidatenquelle aussieht, werfen Sie einen Blick auf die am gewünschten Speicherort (unter Windows 7 standardmäßig beispielsweise *C:\Users\<Benutzername>\Documents*) angelegte Textdatei *Access-SQL.dsn* (siehe Abbildung 4.13). Diese enthält nun die Verbindungszeichenfolge beziehungsweise den Connectionstring der Verbindung.

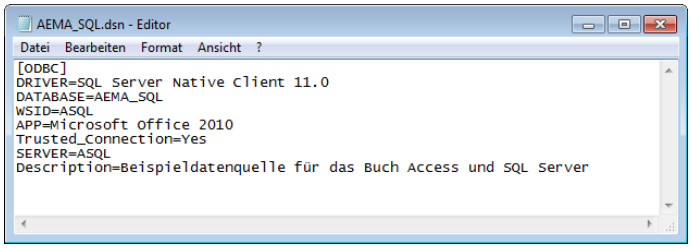


Abbildung 4.13: Dateidatenquelle im Editor

4.1.2 Erstellen einer Computerdatenquelle

Alternativ zur Dateidatenquelle können Sie sich gleich zu Beginn für eine Computerdatenquelle entscheiden. Dazu müssen Sie mit Administratorrechten arbeiten. Dazu navigieren Sie zunächst im Windows Explorer zum Verzeichnis *C:\Program Files (x86)\Microsoft Office\Office14*. Dort klicken Sie mit der rechten Maustaste auf die Datei *MSAccess.exe* und wählen aus dem Kontextmenü den Eintrag *Als Administrator ausführen* aus (siehe Abbildung 4.14).

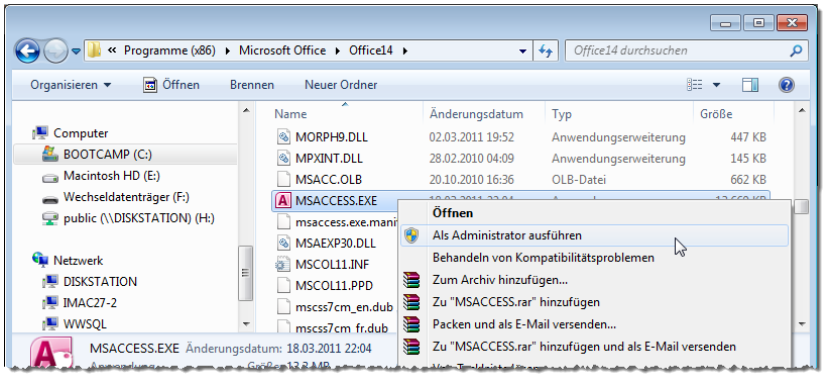


Abbildung 4.14: Access mit Administratorrechten öffnen

In diesem Fall steht eine weitere Entscheidung an: Möchten Sie eine Benutzerdatenquelle oder eine Systemdatenquelle erstellen? Der Unterschied besteht darin, dass eine Benutzerdatenquelle

nur vom aktuell angemeldeten Benutzer verwendet werden kann, die Systemdatenquelle jedoch von allen Benutzern (siehe Abbildung 4.15).

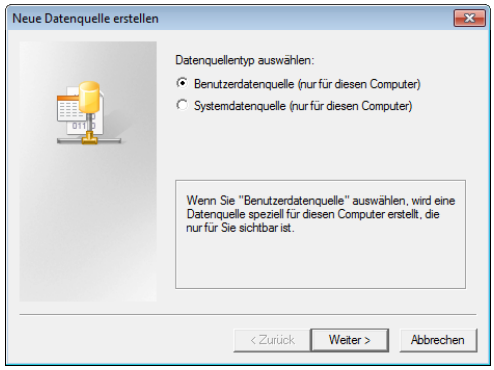


Abbildung 4.15: Benutzerdatenquelle oder Computerdatenquelle?

Um die Unterschiede zu untersuchen, haben wir zwei Computerdatenquellen erstellt – eine Benutzerdatenquelle und eine Systemdatenquelle (siehe Abbildung 4.16). Mit diesen Datenquellen arbeiten Sie nicht viel anders als mit den Dateidatenquellen – Sie wählen diese aus oder legen sie neu an und greifen dann auf die Liste der Tabellen zu, um die gewünschten Verknüpfungen zu erstellen. Interessant ist noch der Ort, an dem die Datenquellen in der Registry gespeichert werden. So finden Sie die Benutzerdatenquellen in der Registry unter folgendem Zweig (siehe Abbildung 4.17):

```
HKEY_CURRENT_USER\Software\ODBC\ODBC.INI\<Name der Datenquelle>
```

Die Systemdatenquellen sind hier zu finden:

```
HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI\<Name der Datenquelle>
```

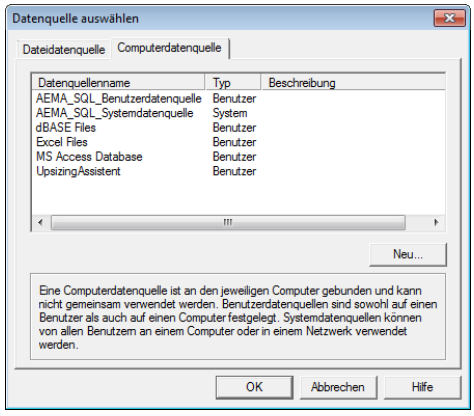


Abbildung 4.16: Zwei neu erstellte Computerdatenquellen

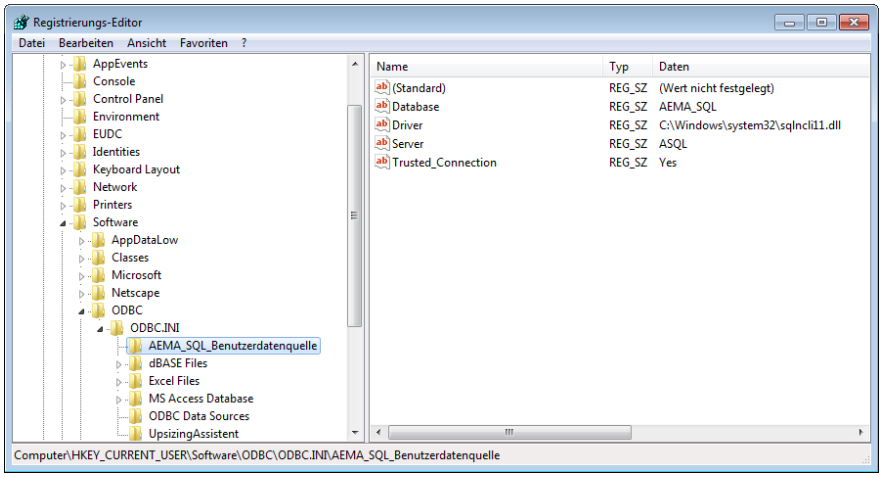


Abbildung 4.17: Eintrag für die Benutzerdatenquelle in der Registry

4.1.3 Verbinden von Tabellen ohne Primärschlüssel

Wenn die zu verknüpfende Tabelle keinen Primärschlüssel aufweist, können Sie selbst einen auswählen. In diesem Fall erscheint der Dialog aus Abbildung 4.18.

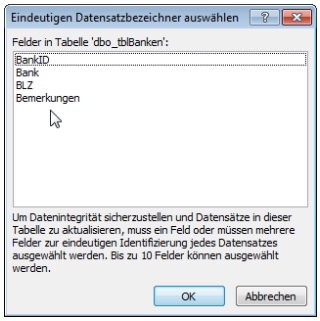


Abbildung 4.18: Aufforderung, ein ID-Feld festzulegen

Wenn Sie hier kein Feld auswählen, erstellt Access eine Verknüpfung, deren Datensätze nicht geändert werden können (siehe Abbildung 4.19).

Wenn Sie nun im Navigationsbereich auf die frisch verknüpfte Tabelle klicken, öffnet Access diese in der Datenblattansicht. Sie können ohne Probleme auf die enthaltenen Daten zugreifen. Sollten Sie jedoch die Access-Datenbank schließen und erneut öffnen, erscheint der Dialog aus Abbildung 4.20 – zumindest, wenn Sie die SQL Server-Authentifizierung verwenden. Die bei der Erstellung der Verknüpfung angegebenen Benutzerdaten werden also offensichtlich nicht gespeichert.

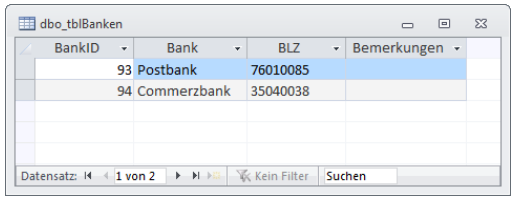


Abbildung 4.19: Verknüpfungen ohne Angabe eines eindeutigen Feldes sind nicht aktualisierbar.

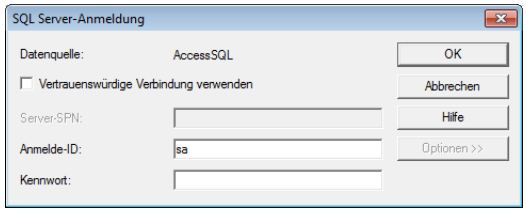


Abbildung 4.20: Die Benutzerdaten werden nicht gespeichert

Dies erhöht die Sicherheit gegenüber älteren Kombinationen aus Access und SQL Server entscheidend – dort konnten Sie bei der Erstellung der Verknüpfung noch angeben, dass Benutzername und Kennwort gespeichert werden.

Was ist passiert?

Access hat eine Verknüpfung zu einer Tabelle des SQL Servers hergestellt. Im Detail wurde dabei ein Datensatz in der Tabelle *MSysObjects* angelegt, der wie in Abbildung 4.21 aussieht.

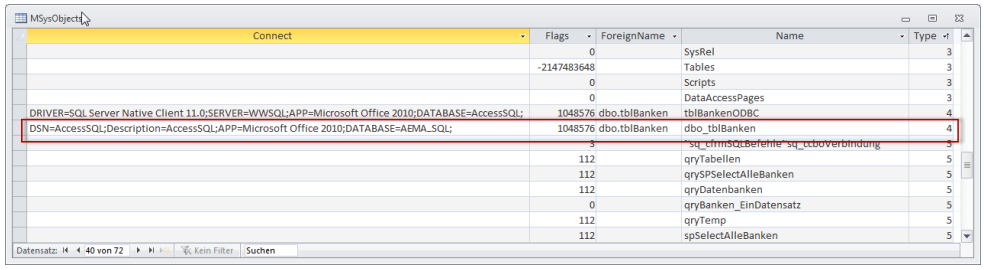
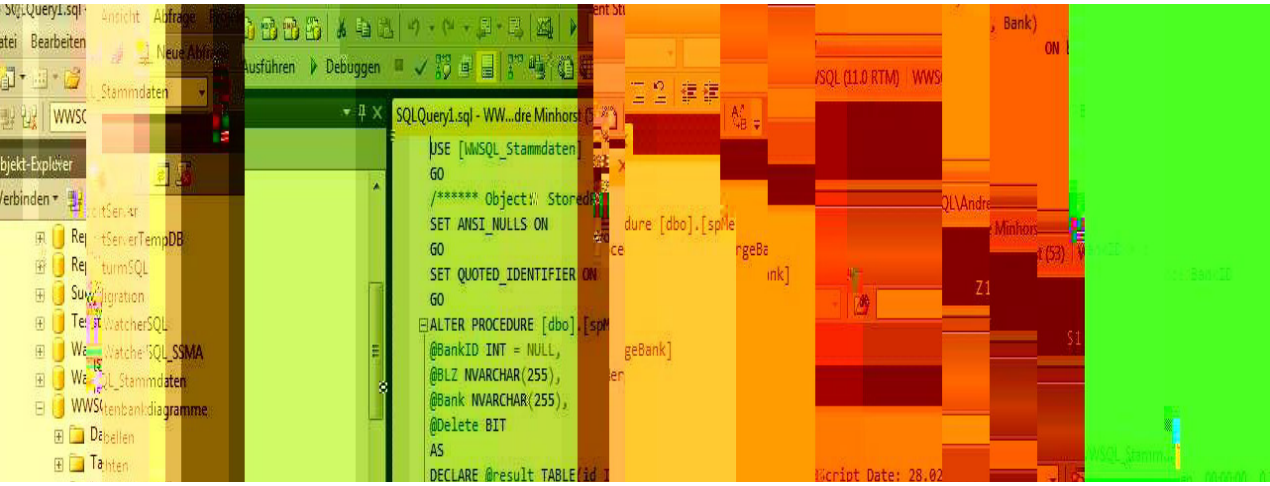


Abbildung 4.21: Der Eintrag für die neue Verknüpfung in der Systemtabelle *MSysObjects*

Sollte die Tabelle nicht im Navigationsbereich erscheinen, aktivieren Sie die Anzeige der Systemobjekte. Dazu klicken Sie mit der rechten Maustaste auf den Titel des Navigationsbereichs und wählen aus dem nun erscheinenden Kontextmenü den Eintrag *Navigationsoptionen...* aus. Dort markieren Sie im unteren Bereich die Option *Systemobjekte anzeigen* (siehe Abbildung 4.22).

Die Tabelle *MSysObjects* enthält für uns folgende interessante Felder:

... DIE ÜBRIGEN SEITEN GIBT'S GEDRUCKT UNTER SHOP.MINHORST.COM



BERND JUNGBLUTH
ANDRÉ MINHORST

André Minhorst Verlag

5 Performance analysieren

Wenn Sie eine Access-Applikation erstellen, die ihre Daten aus den Tabellen einer SQL Server-Datenbank bezieht, sollten Sie von Beginn an auf Performance bedacht sein. Die Performance verschiedener Zugriffe auf die Daten des SQL Servers können Sie vom Frontend aus messen. Sie können auch prüfen, wie sich verschiedene Zugriffsarten auf die Performance auswirken.

Das Angebot des SQL Servers stellt für diesen Zweck zwei tolle Werkzeuge zur Verfügung: *SQL Server Profiler* und die *Erweiterten Ereignisse*.

Beide zeichnen jegliche Aktionen im SQL Server auf. Dazu gehören neben systeminternen Befehlen auch die *SELECT*-Abfragen und Aktionsabfragen, die das Access-Frontend an den SQL Server übergibt, sowie die Ausführung von gespeicherten Prozeduren, Funktionen und Triggern. Die Aufzeichnungen werden als Trace oder Ablaufverfolgung bezeichnet und basieren auf der Protokollierung von Ereignissen. In diesem Kapitel lernen Sie beide Tools kennen.

Direkte Zugriffe statt Umwege

In diesem Kapitel geht es nicht primär darum, eine SQL Server-Anwendung und die enthaltenen Objekte brutal auf Performance zu trimmen. Hauptsächlich ist es wichtig, die Werkzeuge kennenzulernen, mit denen sich die Zugriffe auf die Daten im SQL Server protokollieren und analysieren lassen. Dies vor allem mit dem Hintergrund, den direkten Zugriff von Access aus auf die Daten zu fördern – also beispielsweise den Einsatz von Pass-Through-Abfragen zur Ermittlung von Daten und zum Ausführen von Aktionsabfragen.

Zugriffe etwa über per ODBC eingebundene Tabellen erkennen Sie mithilfe des SQL Server Profilers und der Erweiterten Ereignisse leicht. Und je nachdem, ob diese sich als Performance-Bremsen erweisen, können Sie diese dann durch direkte Zugriffe auf den SQL Server ersetzen.

5.1 SQL Server Profiler

Den *SQL Server Profiler* starten Sie über den Startmenü-Eintrag *Start|Alle Programme|Microsoft SQL Server 2012|Leistungstools|SQL Server Profiler* oder über den Menüpunkt *Extras|SQL Server Profiler* im *SQL Server Management Studio*.

Als erstes erscheint ein Anmeldedialog. Hierüber erstellen Sie eine Verbindung zu der SQL Server-Instanz, deren Ereignisse Sie protokollieren möchten – im aktuellen Fall also zu der SQL Server-Instanz, die Ihre migrierte Datenbank verwaltet. Nach der Anmeldung sehen Sie den Dialog *Ablaufverfolgungseigenschaften* (siehe Abbildung 5.1).

Sollte wider Erwarten weder der Anmeldedialog noch der Dialog *Ablaufverfolgungseigenschaften* zu sehen sein, können Sie auch eine neue Ablaufverfolgung über das Menü anlegen. Dies

erledigen Sie über den Menübefehl *Datei|Neue Ablaufverfolgung...* oder mit der Tastenkombination *Strg + N* (siehe Abbildung 5.2). Im Dialog *Ablaufverfolgungseigenschaften* tragen Sie zunächst einen Namen zur Ablaufverfolgung ein. Wenn Sie die Ergebnisse der Ablaufverfolgung speichern möchten, haben Sie zwei Möglichkeiten: in einer Tabelle oder in einer Datei. In den meisten Fällen reicht es, sich die Ergebnisse direkt in der Ausgabe des Profilers anzusehen.

Für tiefergehende Analysen ist die direkte Ausgabe im Profiler jedoch nicht geeignet. Das Ergebnis lässt sich weder filtern noch sortieren oder gruppieren. Was zum Beispiel die Ermittlung gleichartiger Zeilen, die auf mehrfachen Aufruf des gleichen SQL-Befehls hindeuten, erschwert. In einem solchen Fall ist es sinnvoll, das Ergebnis in einer SQL Server-Tabelle zu speichern und anschließend mittels T-SQL zu sortieren und/oder zu gruppieren.

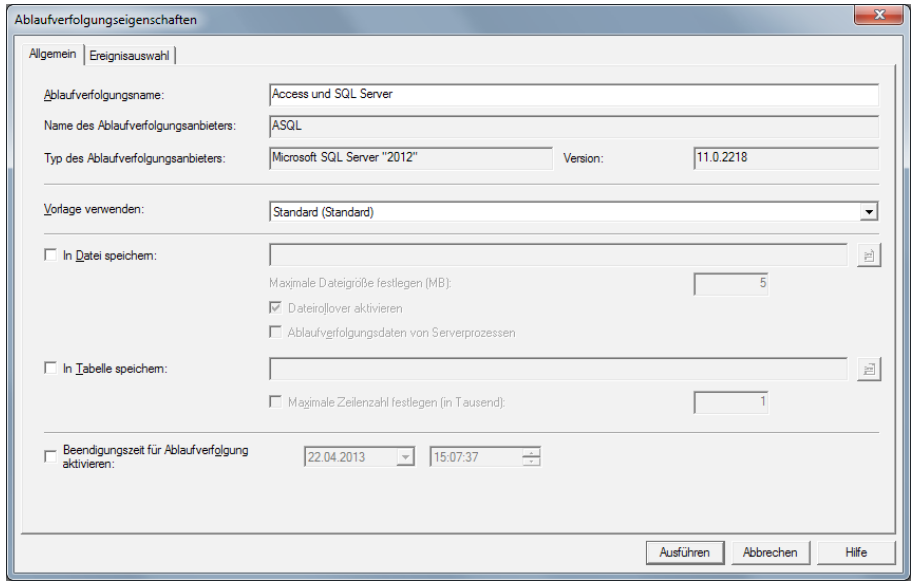


Abbildung 5.1: Der Dialog *Ablaufverfolgungseigenschaften*

Um die Profiler-Ergebnisse in einer Tabelle zu speichern, klicken Sie zunächst auf die Option *In Tabelle speichern*. Der Profiler liefert Ihnen nun erneut einen Anmeldedialog. Dieser Dialog hilft beim Herstellen einer Verbindung zu der SQL Server-Instanz, in der sich die Datenbank befindet, die Sie zum Speichern der Profiler-Ergebnisse nutzen möchten.

Die Tabelle zum Speichern könnte sich auch in der zu untersuchenden Datenbank befinden, da die zum Speichern notwendigen Aktionen nicht protokolliert werden. Aus Gründen der Performance ist es jedoch empfehlenswert, die Ergebnisse einer Ablaufverfolgung in eine Datenbank einer anderen SQL Server-Instanz zu schreiben.

Nach der Anmeldung erscheint ein Dialog, mit dem Sie den Namen der Zieltabelle angeben oder eine vorhandene Zieltabelle auswählen (siehe Abbildung 5.3).

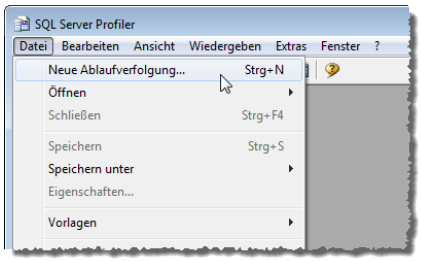


Abbildung 5.2: Anlegen einer neuen Ablaufverfolgung

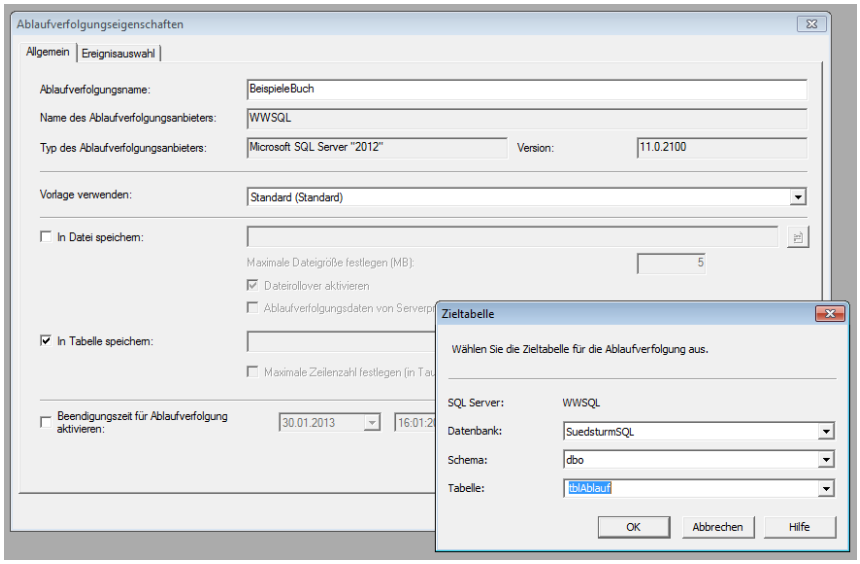


Abbildung 5.3: Festlegen einer Zieltabelle zum Speichern der durch den *SQL Server Profiler* ermittelten Daten

Für die nun folgenden Beispiele reicht die direkte Ausgabe im Profiler absolut aus. Sie können natürlich die Ausgabe in einer Tabelle auch beibehalten und sich das Ergebnis später mit T-SQL anschauen. Auf der zweiten Registerkarte des Dialogs *Ablaufverfolgungseigenschaften* legen Sie fest, welche Informationen zu welchen Ereignissen der Profiler speichern soll. Zu Beginn zeigt der Dialog einige Standardinformationen in sogenannten *Ereignisspalten* für die wichtigsten Ereignisse an. Mit den beiden Optionen *Alle Ereignisse anzeigen* und *Alle Spalten anzeigen* blenden Sie weitere Ereignisse und Ereignisspalten ein (siehe Abbildung 5.5).

Der Profiler zeichnet die Ereignisse aller Datenbanken der referenzierten SQL Server-Instanz auf. Wenn Sie also nur ermitteln möchten, welche Ereignisse die zu untersuchende Datenbank verursacht, müssen Sie noch einen entsprechenden Filter setzen. Dieser soll sich auf den Datenbanknamen beziehungsweise die Datenbank-ID beziehen.

Beide Informationen zeichnet der Profiler standardmäßig nicht auf. Wenn Sie sich einmal ansehen möchten, welche Datenbanken überhaupt Ereignisse auslösen, müssen Sie zunächst noch die beiden Ereignisspalten *DatabaseID* und *DatabaseName* markieren (siehe Abbildung 5.4). Die Reihenfolge der gewählten Ereignisspalten lässt sich über die Schaltfläche *Spalten organisieren* ändern.

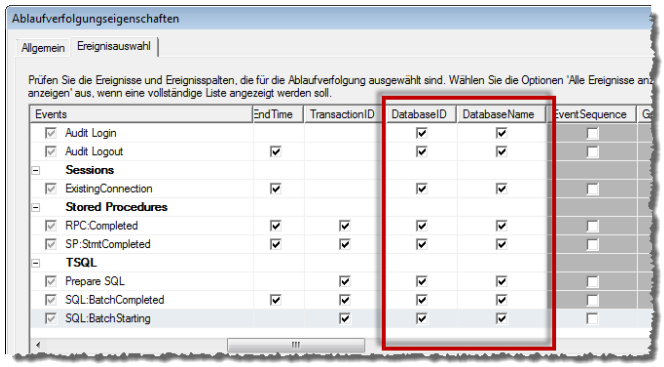


Abbildung 5.4: Aktivieren der Eigenschaften *DatabaseID* und *DatabaseName*

Fehlt noch der eigentliche Filter. Dazu öffnen Sie den entsprechenden Dialog mit der Schaltfläche *Spaltenfilter* (unten rechts im Dialog *Ablaufverfolgungseigenschaften*). Klicken Sie auf *DatabaseName*, erweitern Sie dann den Eintrag *Wie* des Baums rechts und tragen Sie dort den Namen der zu untersuchenden Datenbank ein (siehe Abbildung 5.6). Performanter ist allerdings die Verwendung der Ereignisspalte *DatabaseID*. Im Gegensatz zum Datenbanknamen kennen Sie die *DatabaseID* normalerweise nicht. Das ist aber kein Problem: Öffnen Sie das SQL Server Management Studio, klicken Sie im Objekt-Explorer mit der rechten Maustaste auf den Namen der zu untersuchenden Datenbank und wählen Sie den Eintrag *Neue Abfrage* aus. Geben Sie nun die folgende Anweisung in das neue, leere Abfragefenster ein und betätigen Sie die Taste *F5*, um die Abfrage auszuführen:

```
SELECT DB_ID();
```

Dies liefert die *DatabaseID* der aktuellen Datenbank zurück, die Sie als Vergleichswert für ein Filterkriterium auf Basis der gleichnamigen Ereignisspalte im Profiler verwenden können. Die Datenbank-ID ist nicht der einzige mögliche Filter. Sie können für fast alle Ereignisspalten einen Filter festlegen. Betrachtet man die Ereignisspalten *Hostname*, *Loginname* und *Applicationname,* wird deutlich, wie detailliert sich eine Ablaufverfolgung definieren lässt.

Vorlage verwenden

Zur Analyse des Zusammenspiels von Access und SQL Server verwenden wir die Ereignisse *RPC:Completed*, *SQL:StmtStarting* und *SQL:StmtCompleted* mit ihren Standardinformationen zuzüglich der Ereignisspalten *DatabaseID*, *DatabaseName*, *Hostname* und *RowCounts*. Nach der

Auswahl dieser Ereignisspalten können Sie die Ablaufverfolgung mit einem Klick auf die Schaltfläche *Ausführen* starten.

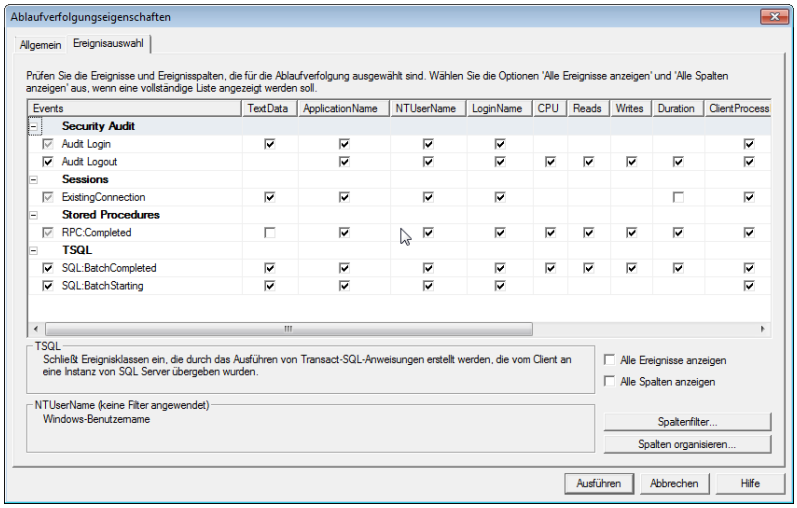


Abbildung 5.5: Ereignisse und ihre Ereignisspalten

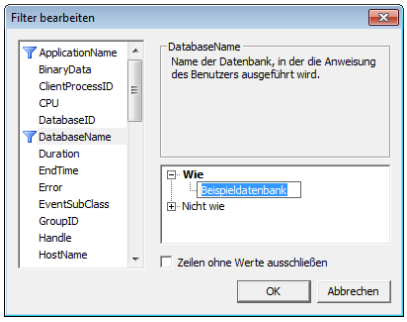


Abbildung 5.6: Filtern nach Ereignissen, die durch eine bestimmte Datenbank ausgelöst werden

Sie können aber auch die Vorlage aus dem Download zu diesem Buch verwenden. Dazu rufen Sie den Menübefehl *Datei|Vorlagen|Vorlage importieren...* auf (siehe Abbildung 5.7). Wählen Sie dann im *Datei öffnen*-Dialog die Datei *AccessSQL.tdf* aus.

5.2 Ablaufverfolgung starten

Eine Ablaufverfolgung starten Sie mit dem Menübefehl *Datei|Neue Ablaufverfolgung*. Dies öffnet den Dialog zum Herstellen der Verbindung mit der SQL Server-Instanz. Wählen Sie dort die gewünschte SQL Server-Instanz aus und klicken Sie auf *Verbinden*.

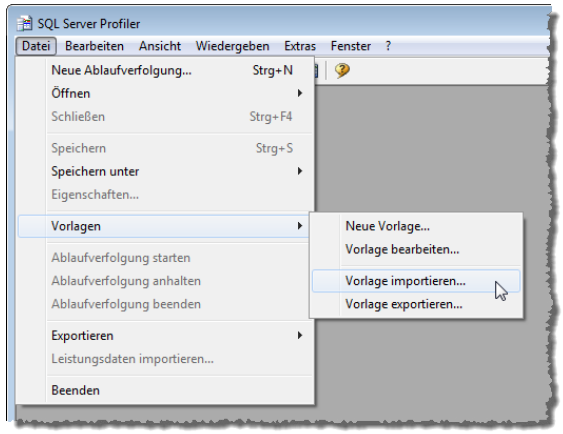


Abbildung 5.7: Vorlage für eine Ablaufverfolgung auswählen

Anschließend erscheint wieder der Dialog mit den Ablaufverfolgungseigenschaften. Hier wählen Sie unter *Vorlage verwenden* die soeben importierte Vorlage aus (siehe Abbildung 5.8).

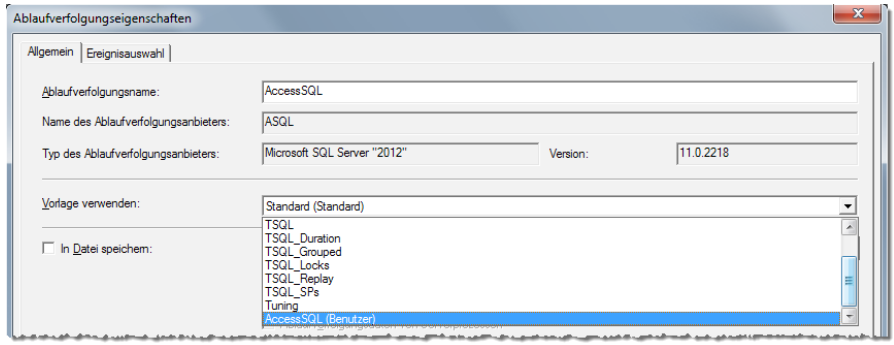


Abbildung 5.8: Auswählen der zuvor importierten Vorlage

Ein Wechsel zur Registerkarte *Ereignisauswahl* liefert die in dieser Vorlage aktivierten Ereignisse und Ereignisspalten (siehe Abbildung 5.9).

Bevor Sie die Ablaufverfolgung starten, müssen Sie noch den Filter einstellen, damit der Profiler auch nur die Ereignisse der beobachteten Datenbank ausgibt. Zwar lässt sich auch der Filter in einer Vorlage speichern, bei einem Filter auf der Datenbank-ID ist dies jedoch nicht sinnvoll.

Die Datenbank-ID wird in jeder SQL Server-Instanz sequenziell vergeben und beinhaltet deshalb nicht immer denselben Wert.

Ein Klick auf die Schaltfläche *Ausführen* startet die Ablaufverfolgung. Der Profiler zeichnet nun alle gewünschten Ereignisse auf.

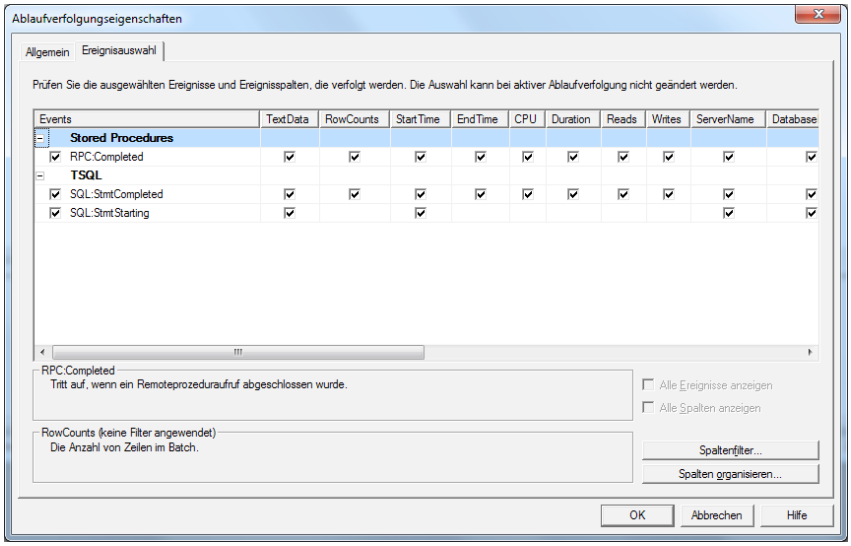


Abbildung 5.9: Diese Ereignisse reichen zur Untersuchung der Performance einer Access -SQL Server-Lösung in der Regel aus.

5.3 Profiler verwenden

Nachdem Sie die Ablaufverfolgung eingerichtet haben, sollten Sie sich ein wenig mit dem Profiler vertraut machen. Damit die Menge der ausgegebenen Informationen Sie nicht erschlägt, beginnen Sie einfach einmal mit dem Ausführen einer simplen SQL-Anweisung im *SQL Server Management Studio*. Klicken Sie mit der rechten Maustaste auf die Beispieldatenbank (hier *AccessSQL*) und wählen Sie den Kontextmenüeintrag *Neue Abfrage...* aus. Bevor Sie eine neue Abfrage ausführen, betätigen Sie im Profiler den Menübefehl *Bearbeiten|Ablaufverfolgungsfenster löschen* beziehungsweise die Tastenkombination *Strg + Umschalt + Entf*. Durch diesen Befehl löschen Sie alle bisherigen Einträge und sehen im Anschluss nur die Einträge der neuen Abfrage. Geben Sie dann die folgende Abfrage im SQL Server Management Studio ein und betätigen Sie die Taste *F5*:

```
SELECT * FROM dbo.tblBestellpositionen WHERE Einzelpreis > 10;
```

Im Profiler schlägt sich dies wie in Abbildung 5.10 nieder. Es gibt zwei Einträge: einen für den Beginn und einen weiteren für das Beenden der SQL-Anweisung. Beides keine großen Ereignisse. Der Profiler notiert sich die Abfrage und einige weitere Informationen wie beispielsweise die Dauer in der Spalte *Duration* und die Anzahl der ermittelten Datensätze in der Spalte *RowCounts*. Der Inhalt der Spalte *Textdata* der aktuellen Zeile wird unter der Liste komplett angezeigt. Das ist hilfreich, wenn der Text zu lang ist, um in einer Zeile der Spalte dargestellt zu werden.

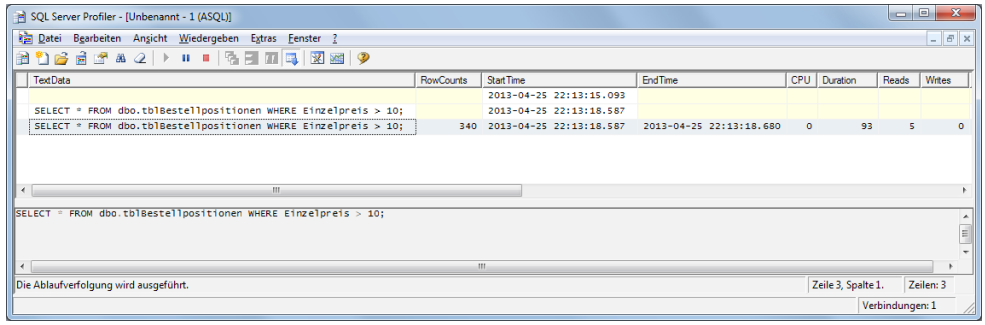


Abbildung 5.10: Protokollieren einer ersten Beispielabfrage im Profiler

Dauer einer Aktion

Die Dauer einer Aktion, also einer einzigen Zeile im Protokoll der Ablaufverfolgung, wird in die Spalte *Duration* eingetragen. Hierbei gibt es einen kleinen Unterschied zwischen der Darstellung in der Ausgabe des Profilers und in dem in einer Tabelle gespeicherten Protokolls: Im Profiler wird die Zeit in Millisekunden ausgegeben, in der Tabelle hingegen in Mikrosekunden.

Zugriff per ODBC-Verknüpfung

Nun schauen wir uns an, wie sich die vermeintlich gleiche Aktion auswirkt, wenn wir den Zugriff auf die Daten in einer Access-Datenbank über eine per ODBC eingebundene Tabelle durchführen. Dabei erstellen wir eine Access-Abfrage mit folgendem SQL-Ausdruck:

```
SELECT * FROM tblBestellpositionen WHERE Einzelpreis > 10;
```

Bevor Sie die Abfrage ausführen, löschen Sie im Profiler das bisherige Ergebnis mit der Tastenkombination *Strg + Umschalt + Entf*. Das Ergebnis der Abfrage sehen Sie in Abbildung 5.11.

Die ersten beiden Einträge im Profiler zeigen mit den Ereignissen *SQL:StmtStarting* und *SQL:StmtCompleted* den Start und das Ende der folgenden SQL-Anweisung:

```
SELECT "dbo"."tblBestellpositionen"."BestellpositionID"
FROM "dbo"."tblBestellpositionen"
WHERE ("Einzelpreis" > 10 )
```

Dabei handelt es sich jedoch nicht um die ursprüngliche SQL-Anweisung. Anstelle der Daten aller Datenspalten sollen hier nur die Daten der Spalte *BestellpositionID* ermittelt werden. Der Grund für diese SQL-Anweisung liegt im Zusammenspiel von Access und SQL Server. Bei der Datenermittlung werden zunächst nur die Primärschlüssel der Ergebnisdaten übermittelt. Erst im zweiten Schritt folgt die Übertragung der restlichen Datenspalten.

Diese Datenermittlung erfolgt über *Remote Procedure Calls*, deren Ereignisse *RPC:Completed* in den folgenden Einträgen im Profiler zu sehen sind. Die Vorbereitung der *Remote Procedure Calls* sind die nächsten beiden Zeilen im Profiler.

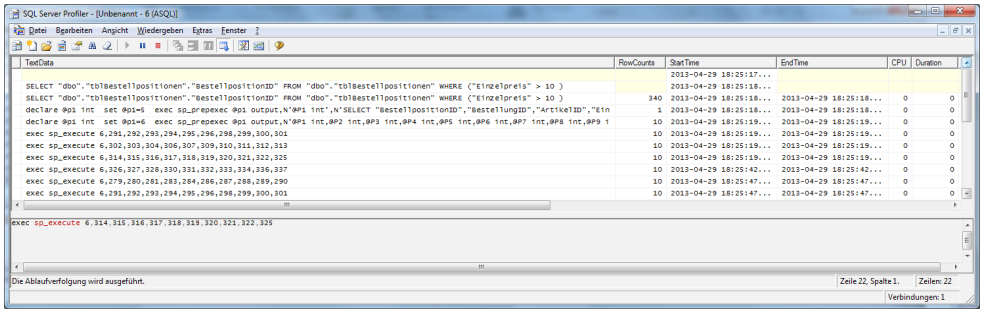


Abbildung 5.11: Protokoll beim Abrufen eines einzigen Datensatzes per ODBC

Dabei ermittelt die erste SQL-Anweisung bereits die Datenspalten des ersten Datensatzes.

```
declare @p1 int
set @p1=15
exec sp_prepexec @p1 output,N'@P1 int',N'SELECT "BestellpositionID",
"BestellungID","ArtikelID","Einzelpreis","Mehrwertsteuersatz","Menge","Rabatt"
FROM "dbo"."tblBestellpositionen"  WHERE "BestellpositionID" = @P1',279
select @p1
```

Die zweite SQL-Anweisung bereitet den *Remote Procedure Call* zur Ermittlung mehrerer Datensätze vor und liefert die Datenspalten der nächsten zehn Datensätze.

```
declare @p1 int
set @p1=16
exec sp_prepexec @p1 output,N'@P1 int,@P2 int,@P3 int,@P4 int,@P5 int,@P6 int,@P7 int,
@P8 int,@P9 int,@P10 int',N'SELECT "BestellpositionID","BestellungID","ArtikelID",
"Einzelpreis","Mehrwertsteuersatz","Menge","Rabatt"  FROM "dbo"."tblBestellpositionen"
WHERE "BestellpositionID" = @P1 OR "BestellpositionID" = @P2
OR "BestellpositionID" = @P3 OR "BestellpositionID" = @P4 OR "BestellpositionID" = @P5
OR "BestellpositionID" = @P6 OR "BestellpositionID" = @P7 OR "BestellpositionID" = @P8
OR "BestellpositionID" = @P9 OR "BestellpositionID" = @P10',279,280,281,283,284,286,
287,288,289,290
select @p1
```

Im Anschluss daran folgt die Datenermittlung der restlichen Daten - und das immer im gleichen Schema: Es werden lediglich zehn Datensätze ermittelt.

```
exec sp_execute 16,291,292,293,294,295,296,298,299,300,301
```

Diese Art der Datenermittlung wird solange wiederholt, bis alle im Abfragefenster sichtbaren Datensätze übertragen wurden. Hinzu kommen zehn weitere Datensätze, für den Fall, dass der Benutzer im Ergebnis blättert. Zeigt Access in der Datenblattansicht 20 Datensätze, werden insgesamt 30 Datensätze übermittelt. Mit den 30 Datensätzen ist die eigentliche Abfrage aber noch nicht beendet, liefert das Gesamtergebnis doch weitaus mehr Datensätze.

Dennoch werden keine weiteren Datensätze eingelesen. Wenn Sie das Abfragefenster unberührt lassen, gibt es auch keine zusätzlichen Datenermittlungen mehr. Es wird tatsächlich gewartet,

ob noch weitere Daten in der Ausgabe angezeigt werden sollen. Springen Sie nun zum letzten Datensatz, beginnt die Datenübertragung aufs Neue. Dieses Mal jedoch ohne Vorbereitung der *Remote Procedure Calls*, sondern direkt mit der Anweisung zur Ermittlung des letzten Datensatzes.

```
exec sp_execute 15,705
```

Im Anschluss daran folgt sofort die Ermittlung der weiteren Datensätze, die im Abfrageergebnis zu sehen sind.

```
exec sp_execute 16,650,652,654,656,657,658,659,660,661,662
```

Die Vorgehensweise der Datenermittlung ist immer dieselbe, auch wenn es keine zehn Datensätze mehr zu ermitteln gibt. In diesem Fall wird der entsprechende Wert einfach mehrmals über die vorbereitete SQL-Anweisung ermittelt:

```
exec sp_execute 16,692,697,698,704,704,704,704,704,704,704
```

Nun mag es im ersten Moment vorteilhaft erscheinen, dass nur die Daten übertragen werden, die in Access zu sehen sind. Wäre da nicht die automatische Aktualisierung der angezeigten Daten, die jedes Mal eine erneute Übertragung der bereits ermittelten Datensätze auslöst. Dieser Automatismus ist abhängig von der Eigenschaft *ODBC-Anzeigeaktualisierungsintervall (s)*, die Sie in den Optionen unter *Clienteinstellungen* finden. Hier definieren Sie den Intervall, in dem die Daten aktualisiert werden sollen. Ändern Sie den Intervall in 15 Sekunden und Sie werden im Profiler sehen, wie die bereits angezeigten Daten alle 15 Sekunden aufs Neue übertragen werden. Das ODBC-Anzeigeaktualisierungsintervall ist eine wichtige Komponente in Ihrer Access/SQL Server-Anwendung. Er hat Auswirkungen auf das Sperrverhalten der SQL Server-Datenbank und somit auf die Gesamtperformance Ihrer Applikation. Doch dazu später mehr.

Dynaset und Snapshot

Das hier gezeigte Verhalten der Datenermittlung ist typisch für alle Access-Abfragen, deren Daten im Abfrageergebnis geändert werden können und für eingebundene Tabellen mit Primärschlüsseln. In beiden Fällen bezeichnet man die Art der Ergebnismenge als *Dynaset*. Der Vorteil ist die mögliche Datenänderung, der Nachteil die Art und Weise der Datenermittlung und der damit verbundenen Aktualisierung vom Abfrageergebnis.

Binden Sie eine Tabelle ohne Primärschlüssel ein, spricht man bei der Ergebnismenge von einem *Snapshot*. Hier sind die Daten nicht nur schreibgeschützt, sondern es ändert sich auch das Abfrageverhalten.

Um dies zu verdeutlichen, erstellen Sie im SQL Server Management Studio zunächst eine Sicht. Dazu starten Sie eine neue Abfrage über den Kontextmenübefehl *Neue Abfrage* der Beispieldatenbank *AccessSQL*. Die Sicht erstellen Sie mit der folgenden SQL-Anweisung:

```
CREATE VIEW dbo.vBestellpositionen
AS
SELECT * FROM dbo.tblBestellpositionen;
```

Anschließend binden Sie die Sicht in Access wie eine Tabelle ein. Dabei klicken Sie beim Dialog *Eindeutigen Datensatzbezeichner auswählen* auf die Schaltfläche *Abbrechen*. Die Sicht steht nun als Tabelle in Access zur Verfügung - jedoch ohne Primärschlüssel. Jetzt löschen Sie im Profiler zunächst wieder die Anzeige mittels *Strg + Umschalt + Entf* und führen dann in einer Access-Abfrage die folgende SQL-Anweisung aus:

```
SELECT * FROM dbo_vBestellpositionen WHERE Einzelpreis > 10;
```

Das Ergebnis ist identisch mit der Abfrage aus unserem ersten Beispiel. Im Profiler ist das Ereignis *SQL:StmtStarting* zu sehen und zwar jetzt auch mit der SQL-Anweisung, die Sie eingegeben haben. Na gut, einen kleinen Unterschied zur ursprünglichen SQL-Anweisung gibt es dann doch: Das "*" durch wurde durch die tatsächlichen Spalten ersetzt.

```
SELECT "BestellpositionID", "BestellungID", "ArtikelID", "Einzelpreis",
"Mehrwertsteuersatz", "Menge", "Rabatt"
FROM "dbo"."vBestellpositionen" WHERE ("Einzelpreis" > 10 )
```

Im Gegensatz zum Dynaset bleibt es zunächst bei diesem einen Eintrag im Profiler. Die Ereignisse zu *RPC:Completed* bleiben in diesem Fall aus, da bei einem Snapshot alle Daten auf einmal nach Access übertragen werden.

Warten Sie einen Moment und Sie werden im Profiler auch den Eintrag zum Ereignis *SQL:StmtCompleted* sehen. Wenn Sie nicht so lange warten möchten, können Sie dem Ganzen auch etwas nachhelfen. Springen Sie in der Datenblattansicht der Access-Abfrage einfach zum letzten Datensatz.

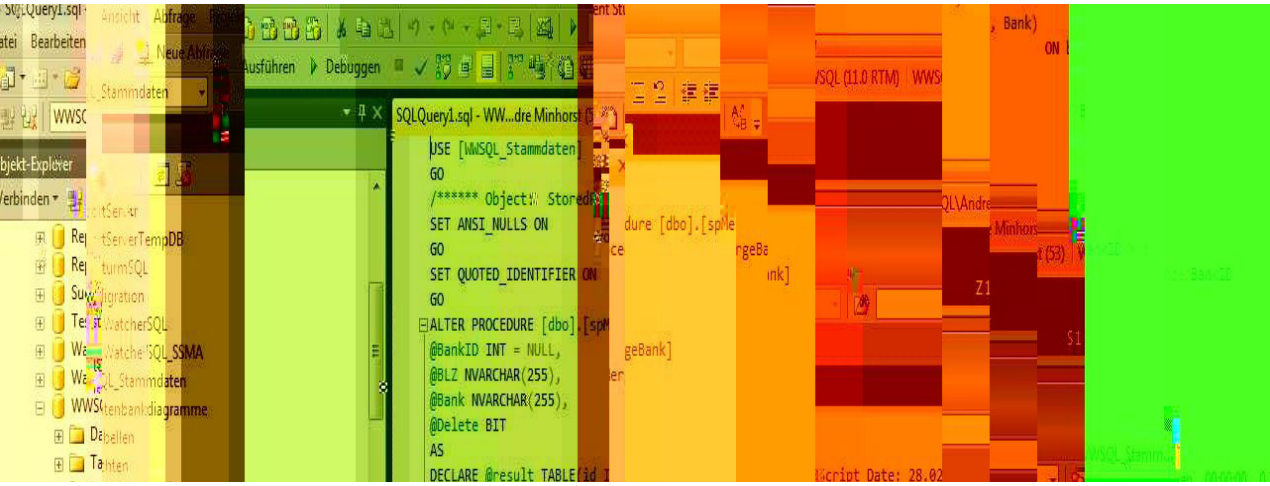
Analyse von Access-Abfragen

Die Art und Weise der Datenübertragung bei eingebundenen Tabellen ist nicht die einzige Besonderheit beim Zusammenspiel von Access und SQL Server. Access interpretiert eingebundene Tabellen zunächst als eigene Tabellen. Erst beim Datenzugriff erkennt Access, dass es sich um eine externe und per ODBC eingebundene Tabelle handelt. Leider ist das auch das Einzige, was erkannt wird: Es handelt sich um eine externe Tabelle.

Access erkennt nicht, von welcher externen Datenbank die eingebundene Tabelle stammt. Ein Umstand, durch den der Access-Abfrageoptimierer gezwungen ist, die SQL-Anweisungen der Access-Abfragen derart zu ändern, dass so gut wie jede mögliche externe Datenbank damit arbeiten kann. So kommt es selten vor, dass die SQL-Anweisung einer Access-Abfrage 1:1 an den SQL Server übergeben werden kann. Auch dieses Verhalten lässt sich mit dem Profiler aufzeichnen. Bereinigen Sie die aktuelle Ablaufverfolgung mittels *Strg + Umschalt + Entf* und führen Sie in einer Access-Abfrage die folgende SQL-Anweisung aus:

```
SELECT TOP 3 tblArtikel.Artikelname, Sum(tblBestellpositionen.Menge) AS SummevonMenge
FROM tblBestellpositionen INNER JOIN tblArtikel ON tblBestellpositionen.ArtikelID =
tblArtikel.ArtikelID
GROUP BY tblArtikel.Artikelname
ORDER BY Sum(tblBestellpositionen.Menge) DESC;
```

... DIE ÜBRIGEN SEITEN GIBT'S GEDRUCKT UNTER SHOP.MINHORST.COM



BERND JUNGBLUTH
ANDRÉ MINHORST

André Minhorst Verlag

6 SQL Server Management Studio

Mit SQL Server werden einige hilfreiche Tools ausgeliefert. Eines davon ist das SQL Server Management Studio. Hiermit verwalten Sie Ihre SQL Server-Instanzen und die darin enthaltenen Datenbanken. Dieses Kapitel liefert keine umfassende Beschreibung dieses Tools, sondern zeigt Ihnen einige seiner interessantesten Funktionen und Möglichkeiten.

6.1 Start und Anmeldung

Das SQL Server Management Studio ist im Startmenü unter *Alle Programme|Microsoft SQL Server 2012|SQL Server Management Studio* zu finden. Nach dem Start zeigt es zunächst den Dialog aus Abbildung 6.1 an, in dem Sie die Instanz des SQL Servers auswählen, die Sie verwalten möchten. Danach wählen Sie die Art der Authentifizierung.

Sie haben die Wahl zwischen der Windows-Authentifizierung und der SQL Server-Authentifizierung. Bei letzterer müssen Sie den SQL Server-Benutzername mitsamt Kennwort in die beiden Eingabefelder *Benutzername* und *Kennwort* eingeben – mehr zum Thema Authentifizierung lesen Sie im Kapitel »Sicherheit und Benutzerverwaltung«, Seite 395. Mit einem Klick auf die Schaltfläche *Verbinden* wird die Verbindung zur SQL Server-Instanz hergestellt.

Abbildung 6.1: Verbindung mit dem SQL Server herstellen

Nach dem Verbinden zeigt der Objekt-Explorer eine Übersicht der verfügbaren Objekte an (siehe Abbildung 6.2).

Sie können sich auch mit mehr als einer Instanz verbinden. Dazu wählen Sie im Objekt-Explorer in der Auswahlliste *Verbinden* den Eintrag *Datenbankmodul*. Der Anmeldedialog aus Abbildung 6.1 wird erneut geöffnet. Hier geben Sie nun in *Servername* eine andere Instanz an, wählen wieder die Authentifizierungsmethode und melden sich mit einem Klick auf *Verbinden* an. Nach der Anmeldung listet der Objekt-Explorer beide Instanzen auf.

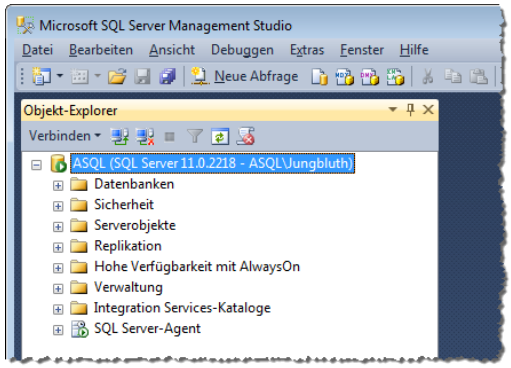


Abbildung 6.2: Objekt-Explorer des SQL Server Management Studios

Ebenso ist es möglich, sich mit ein und derselben Instanz mehrmals zu verbinden und dabei jeweils eine andere Anmeldung zu verwenden. Eine einfache Methode, um das Berechtigungskonzept zu testen.

6.2 Verwalten einer SQL Server-Instanz

Nun könnten wir durchaus alle im Objekt-Explorer verfügbaren Objekte und die gesamten Features des SQL Server Management Studios beschreiben. Der Umfang dieser Beschreibung würde allerdings ein eigenes Buch füllen.

Da begrenzen wir uns doch lieber auf die Möglichkeiten, die während und nach der Migration einer Access-Datenbank relevant sind.

Viele dieser Möglichkeiten werden in diesem Buch in eigenen Kapiteln behandelt. Es folgt eine kurze Übersicht der Objekte, deren Verwendung und Verwaltung in späteren Kapiteln beschrieben sind.

» *Datenbanken*: Erstellen und verwalten von Datenbanken sowie Sichern und Wiederherstellen von Datenbanken – siehe Kapitel »Datenbanken und Tabellen erstellen«, Seite 155, und Kapitel »Sichern und Wiederherstellen«, Seite 499

» *Sicherheit*: Erstellen und verwalten von Anmeldungen und den zugehörigen Berechtigungen – siehe Kapitel *»Sicherheit und Benutzerverwaltung«, Seite 395*

» *Verwaltung | Erweiterte Ereignisse*: Erstellen und Verwalten von erweiterten Ereignissen um beispielsweise das Abfrageverhalten der SQL Server-Instanz zu beobachten – siehe Kapitel »Performance analysieren«, Seite 105

In den folgenden Abschnitten zeigen wir Ihnen noch weitere interessante Möglichkeiten des SQL Server Management Studios.

6.2.1 SQL Server-Protokolle

SQL Server protokolliert wichtige Informationen in Protokollen. Diese Protokolle stehen unter *Verwaltung|SQL Server-Protokolle* zur Verfügung. Inhalt der Protokolle sind unter anderem Meldungen über den Start vom SQL Server-Dienst, Ausführungen von Sicherungen sowie Hinweise zu fehlerhaften Anmeldungen.

Nach Markieren einer Meldung sehen Sie im unteren Bereich weitere Detailinformationen (siehe Abbildung 6.3).

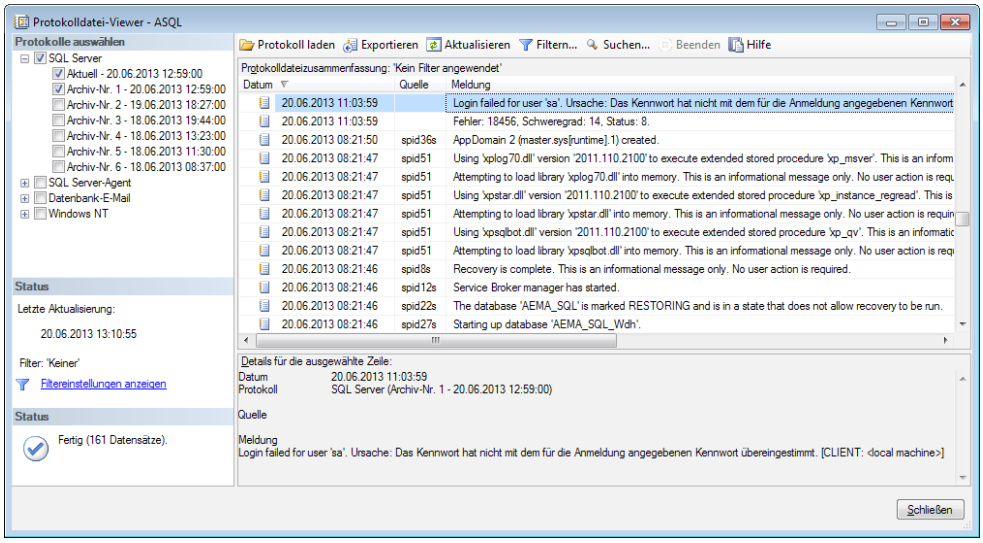


Abbildung 6.3: Die Protokolle einer SQL Server-Instanz

6.2.2 SQL Server-Agent

Der SQL Server-Agent ist der Taskplaner des SQL Servers. Hier definieren Sie Aufgaben, die vom SQL Server automatisch nach den von Ihnen festgelegten Zeitplänen ausgeführt werden.

Dazu muss der SQL Server-Agent allerdings auch gestartet sein. Ob dies der Fall ist, sehen Sie am Symbol vom SQL Server-Agent, den Sie am Ende des Objekt-Explorers finden.

Enthält dieses Symbol einen grünen Pfeil, ist der SQL Server-Agent aktiv; ist es jedoch ein nach unten zeigender roter Pfeil, wird der SQL Server-Agent nicht ausgeführt (siehe Abbildung 6.4).

Sollte der SQL Server-Agent nicht aktiv sein, können Sie ihn bei Bedarf manuell starten. Dazu wählen Sie in dessen Kontextmenü den Eintrag *Starten*. Sie erhalten die Meldung aus Abbildung 6.5.

Wie an der Meldung zu erkennen ist, handelt es sich beim SQL Server-Agent um einen Windows-Dienst. Bekanntlicherweise lassen sich Windows-Dienste auch derart konfigurieren, dass sie

beim Start des Rechners automatisch gestartet werden. Dies ist ebenso mit dem Dienst des SQL Server-Agent möglich – was für einen Taskplaner auch durchaus sinnvoll ist.

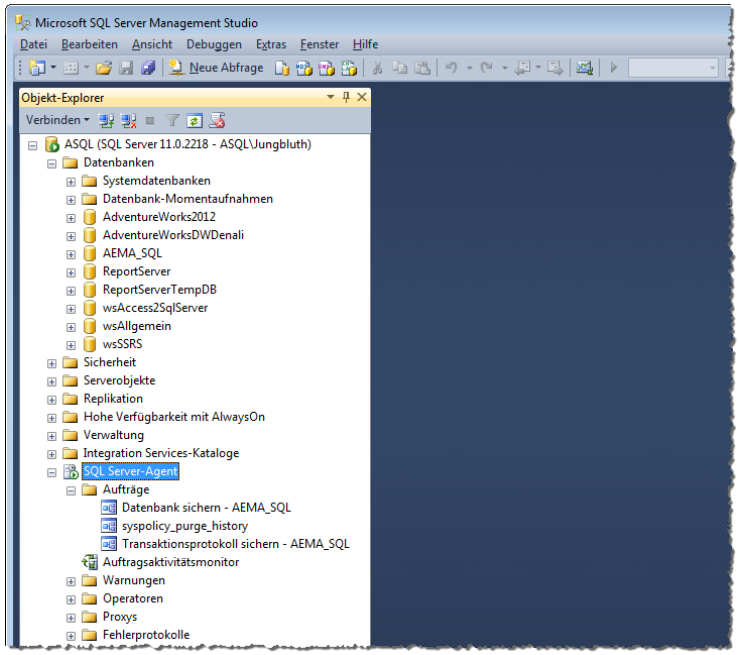


Abbildung 6.4: Die SQL Server-Agent im Objekt-Explorer

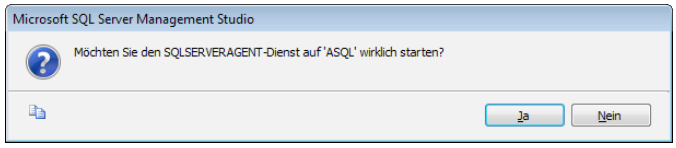


Abbildung 6.5: Starten des SQL Server-Agent

Die grundsätzlichen Startoptionen des SQL Server-Agent – und die der anderen Windows-Dienste vom SQL Server – verwalten Sie am besten mit dem *SQL Server-Konfigurations-Manager*. Sie finden dieses Tool in der Programmgruppe des SQL Servers im Ordner *Konfigurationstools*.

Im *SQL Server-Konfigurations-Manager* öffnen Sie die Gruppe *SQL Server-Dienste* und markieren den Eintrag *SQL Server-Agent (MSSQLSERVER)*. Wobei der in den Klammern angegebene Name abweichen kann, wenn Sie eine benannte Instanz oder die SQL Server Express Edition verwenden.

Ein Doppelklick auf diesen Eintrag zeigt die Einstellungen und bietet die Möglichkeit, den Starttyp festzulegen sowie den Dienst manuell zu starten (sieheAbbildung 6.6).

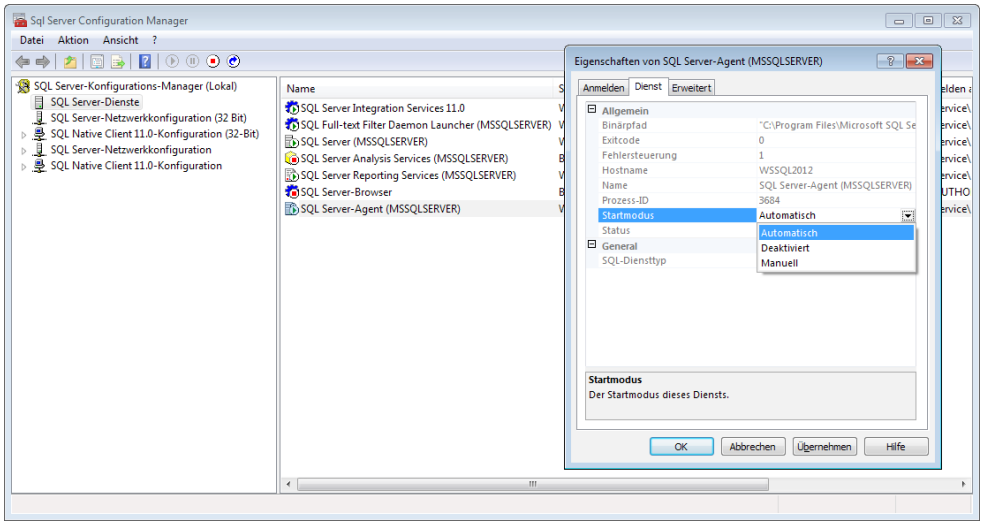


Abbildung 6.6: Prüfen und starten des SQL Server-Agent

Die im SQL Server-Agent geplanten Aufträge sehen Sie im Eintrag *SQL Server-Agent | Aufträge*. Über das Kontextmenü von *Aufträge* lassen sich auch neue Aufträge inklusive ihrer Zeitpläne anlegen.

Eine detaillierte Übersicht über alle Aufträge, deren aktuellen Status, Informationen zur letzten und nächsten Ausführung sowie dem Ergebnis der letzten Ausführung bietet Ihnen der *Auftragsaktivitätsmonitor* (siehe Abbildung 6.7).

Diesen finden Sie ebenfalls im Eintrag *SQL Server-Agent*.

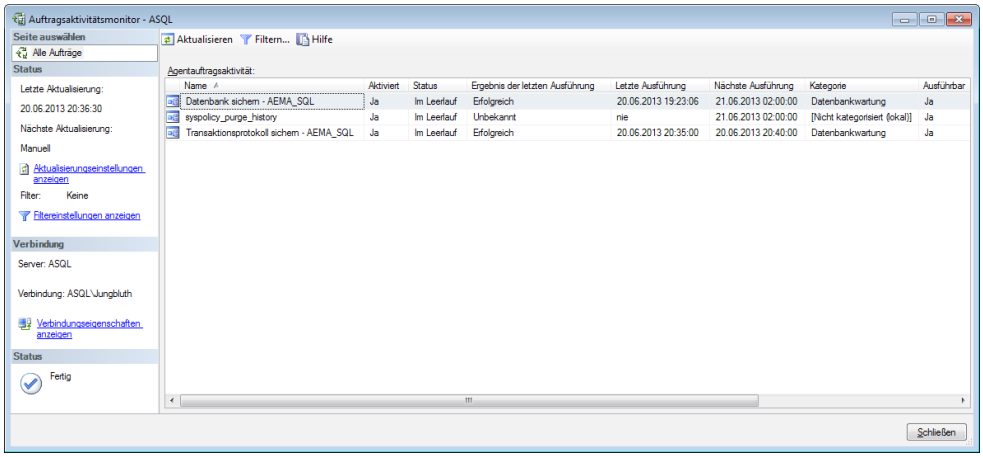


Abbildung 6.7: Der Auftragaktivitätsmonitor

Hier steht Ihnen pro Auftrag ein Kontextmenü zur Verfügung, über das Sie dessen Eigenschaften bearbeiten, ihn aktiveren oder deaktivieren sowie manuell starten können. Auch die Historie der einzelnen Auftragsausführungen lässt Sie über das Kontextmenü einsehen – durch den Eintrag *Verlauf anzeigen*.

Die Auftragshistorie liefert Ihnen Informationen über die Auftragsausführung, ob diese erfolgreich oder fehlerhaft beendet wurde sowie Informationen zu jedem einzelnen Auftragsschritt (siehe Abbildung 6.8). Über die als Links dargestellten Einträge verzweigen Sie zu den entsprechenden Eigenschaften des Auftrags. So führt der in der Abbildung zu sehende Link *1* direkt zum ersten Auftragsschritt des Auftrags *Transaktionsprotokoll sichern – AEMA_SQL*.

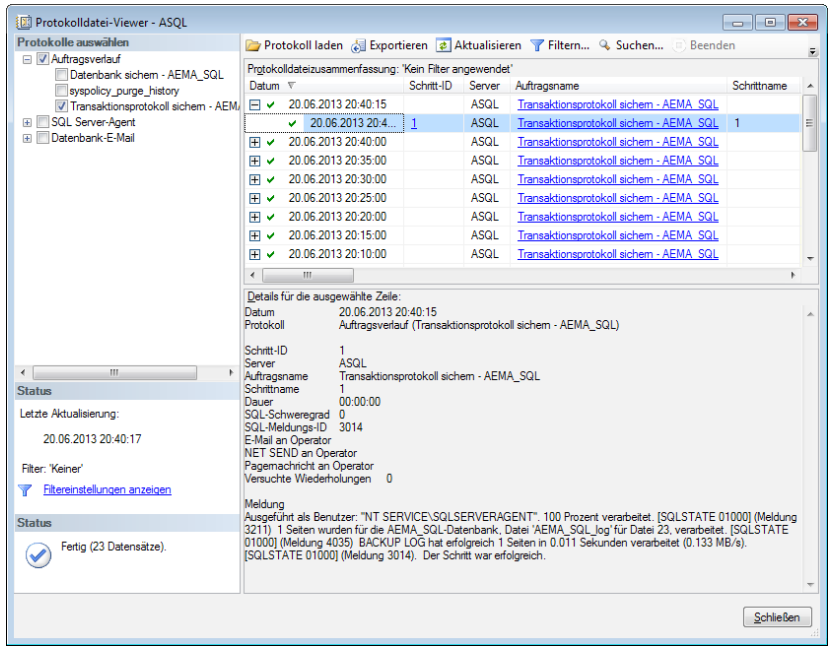


Abbildung 6.8: Die Historie eines Auftrags

Es wird gar nicht so lange dauern, bis Sie die erste Aufgabe erstellen: das regelmäßige Sichern Ihrer Datenbanken. Wie Sie am effektivsten einen solchen Auftrag einrichten, lesen Sie in Kapitel »Sichern und Wiederherstellen«, Seite 499. In der Express Edition von SQL Server steht der SQL Server-Agent leider nicht zur Verfügung.

6.2.3 Auswerten einer SQL Server-Instanz

Informationen zur SQL Server-Instanz und den dort verwalteten Datenbanken liefern der Aktivitätsmonitor und die zahlreichen Standardberichte. Entgegen den Auswertungen mit den *Erweiterten Ereignissen* und dem *SQL Server Profiler* (siehe Kapitel »Performance analysieren«,

Seite 105) sind für diese Informationen keine Definitionen von Sitzungen beziehungsweise Ablaufverfolgungen notwendig, sondern sie stehen direkt zur Verfügung.

Aktivitätsmonitor

Der Aktivitätsmonitor ist die erste Adresse, wenn Sie etwas über die Auslastung Ihres SQL Servers erfahren möchten. Sie starten den Aktivitätsmonitor über die Symbolleiste mit der in Abbildung 6.9 dargestellten Schaltfläche.

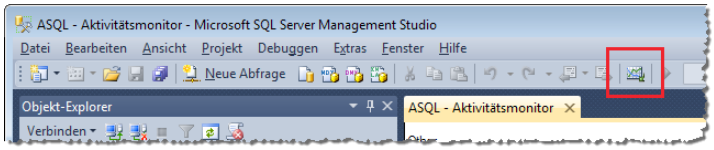


Abbildung 6.9: Die Schaltfläche zum Start des Aktivitätsmonitors Die Schaltfläche zum Start des Aktivitätsmonitors

Der Abschnitt *Übersicht* informiert Sie in vier Diagrammen über die Auslastung des SQL Servers. Hier sehen Sie die aktuelle CPU-Auslastung, die Anzahl der wartenden Tasks, das Eingabe-/Ausgabe-Verhalten in Megabyte pro Sekunde und die Anzahl der Batchanforderungen pro Sekunde. Die weiteren Abschnitte liefern Ihnen Detailinformationen zu den aktuell aktiven Prozessen, eventuellen Wartevorgänge, das Eingabe-/Ausgabeverhalten der Datenbankdateien sowie eine Auflistung der Abfragen, deren Laufzeit nicht eben performant ist. Von diesen Informationen möchten wir Ihnen nun die beiden interessantesten vorstellen.

Da wäre als erstes der Abschnitt *Prozesse*. Hier sehen unter anderem, welche Benutzer aktuell von welchen Clients mit welcher Anwendung auf welche Datenbank zugreifen. Zu jedem Eintrag steht Ihnen ein Kontextmenü zur Verfügung, das den T-SQL-Befehl des Prozesses zeigt (siehe Abbildung 6.10), den Prozess zur weiteren Verfolgung in den Profiler exportiert und mit dem Sie den Prozess sogar abbrechen können.

Möchten Sie nur die Prozesse zu einer bestimmten Datenbank sehen, filtern Sie die Ansicht über die Überschrift *Datenbank*. Diese Filtermöglichkeit steht Ihnen bei jeder Spalte zur Verfügung – ein Filtern nach Client, Anmeldename oder Applikation ist also ebenso möglich.

Etwas irreführend ist die Bezeichnung des Abschnitts *Aktuelle wertvolle Abfragen*. Die hier aufgelisteten Abfragen sind alles andere als wertvoll. Vielmehr enthält dieser Abschnitt eine Hitparade der Abfragen mit dem schlechtesten Abfrageverhalten – ein guter Ausgangspunkt zur Optimierung der Abfragen in Ihrem SQL Server.

Die Analyse dieser Abfragen lässt sich auf eine Datenbank begrenzen. Dazu wählen Sie lediglich in der letzten Spalte namens *Datenbank* die entsprechende Datenbank aus. Die Ansicht wird dadurch auf die Abfragen gefiltert, die in der gewählten Datenbank ausgeführt wurden. Nun können Sie sich die langsamen Abfragen dieser einen Datenbank nach und nach vornehmen.

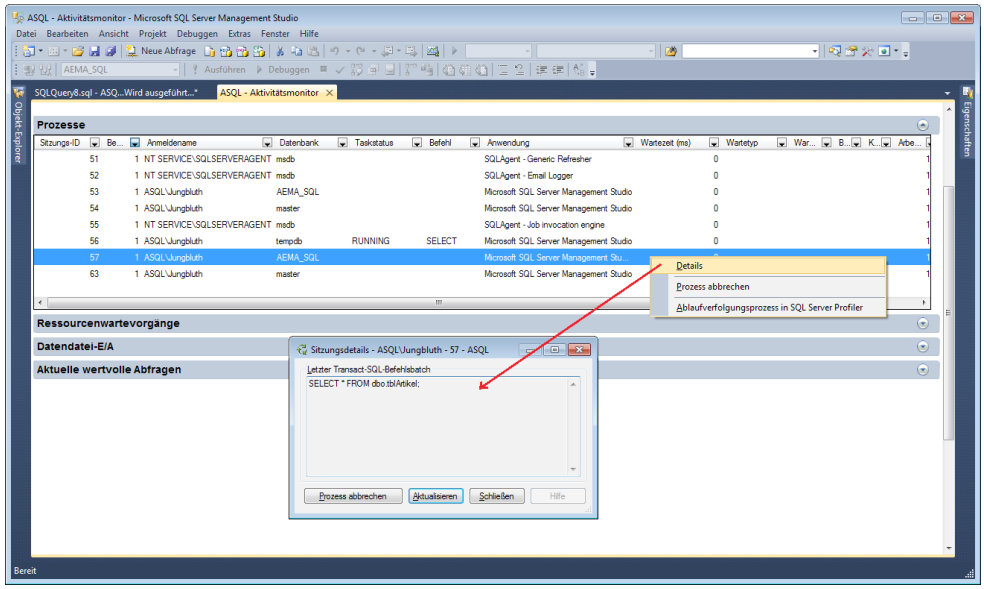


Abbildung 6.10: Die aktiven Prozesse im Aktivitätsmonitor

Auch in dieser Übersicht gibt es zu jedem Eintrag ein Kontextmenü. Hierüber lässt sich der SQL-Befehl der Abfrage zum Bearbeiten in einem Abfragefenster öffnen und der zugehörige Ausführungsplan einsehen (siehe Abbildung 6.11).

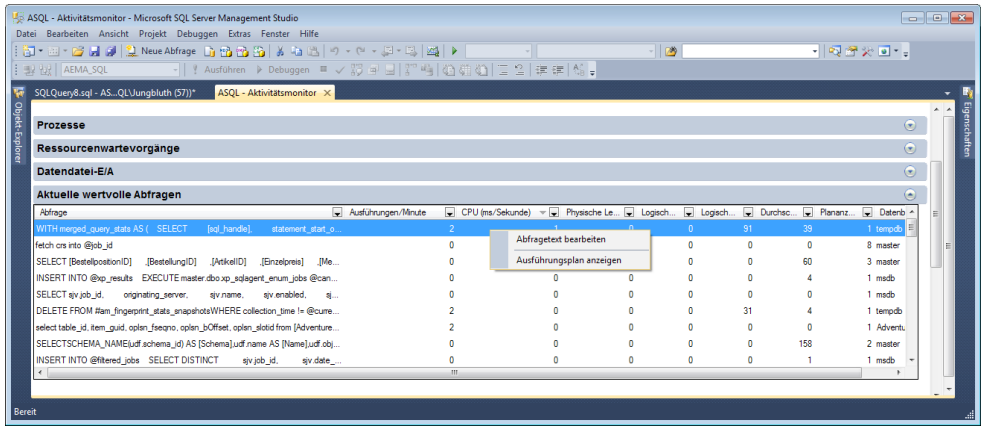


Abbildung 6.11: Die gar nicht mal so wertvollen Abfragen

Berichte

Das SQL Server Management Studio bietet Ihnen zu fast jedem Objekt einen oder mehrere Standardberichte an, die Ihnen detaillierte Informationen zum gewählten Objekt anzeigen. Diese

Standardberichte können Sie – falls vorhanden – über den Kontextmenüeintrag *Berichte|Standardberichte* des jeweiligen Objekts abrufen. Ein Blick in das Angebot der vorhandenen Standardberichte lohnt sich allemal. Ein paar dieser Berichte möchten wir Ihnen nun vorstellen.

Beginnen wir mit den Berichten zur SQL Server-Instanz. Das Kontextmenü des Eintrags zum SQL Server bietet Ihnen einiges an Standardberichten (siehe Abbildung 6.12). Hier eine kurze Aufstellung der interessantesten:

» *Serverdashboard*: Zeigt aktuelle Daten zur Aktivität, CPU-Auslastung sowie das Eingabe-/Ausgabeverhalten plus die Konfigurationsparameter des SQL Servers.

» *Arbeitsspeichernutzung*: Zeigt die aktuelle Speicherauslastung inklusive Aufstellung des Speicherbedarfs nach Komponenten.

» *Aktivität - alle Sitzungen*: Zeigt die aktiven Sitzungen nach Benutzern.

» *Leistung - Objektausführungsstatistik*: Zeigt eine Statistik der ausgeführten Objekte über die gesamte SQL Server-Instanz.

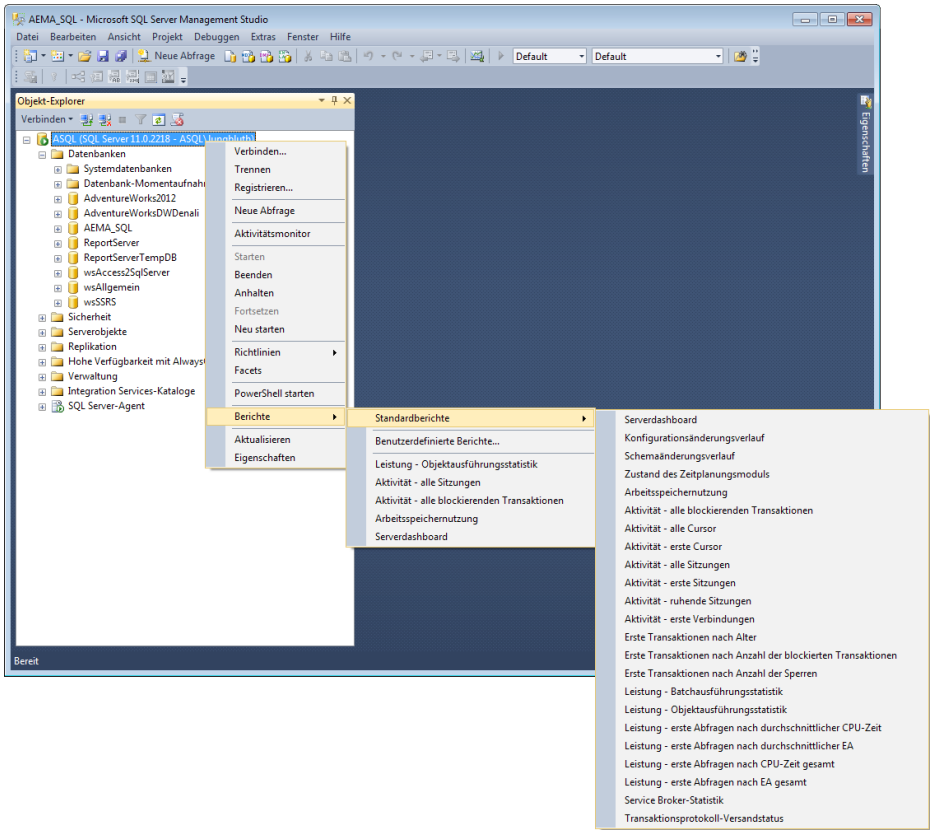


Abbildung 6.12: Die Standardberichte der SQL Server-Instanz

Auch zu den Datenbanken gibt es mehrere Standardberichte:

» *Datenträgerverwendung*: Zeigt die Speicherplatzverwendung der Datenbankdateien.

» *Datenträgerverwendung durch Tabellen*: Zeigt die Speicherplatzverwendung der Tabellen einer Datenbank inklusive Anzahl der Datensätze.

» *Sicherungs- und Wiederherstellungsereignisse*: Zeigt Informationen über die einzelnen Sicherungs- und Wiederherstellungsvorgänge.

» *Objektausführungsstatistik*: Zeigt eine Statistik der ausgeführten Objekte der Datenbank.

» *Statistik zur Indexverwendung*: Zeigt die Verwendung der Indexe durch Benutzer und System.

» *Physische Statistik indizieren*: Zeigt die Speicherplatzverwendung pro Tabelle und Index, inklusive Empfehlungen zur Reorganisation oder Neuerstellung von Indexen.

» *Schemaänderungsverlauf*: Zeigt, wann welche Datenbankobjekte durch welchen Benutzer angelegt oder geändert wurden.

» *Benutzerstatistik*: Zeigt die aktuellen Benutzerverbindungen zur Datenbank.

Schauen Sie sich die einzelnen Berichte mal an – und natürlich auch die hier nicht erwähnten. Sie werden schnell einige zu Ihren Favoriten küren. Da ist es sehr hilfreich, dass bereits aufgerufene Berichte in das Kontextmenü des jeweiligen Objekts aufgenommen werden. So ist der Bericht für den nächsten Aufruf schneller erreichbar.

Alternativ können Sie das Angebot auch mit eigenen Berichten ergänzen. Diese fügen Sie über den Kontextmenüeintrag *Berichte|Benutzerdefinierte Berichte* dem jeweiligen Objekt hinzu.

6.3 Erstellen und verwalten von SQL-Skripten

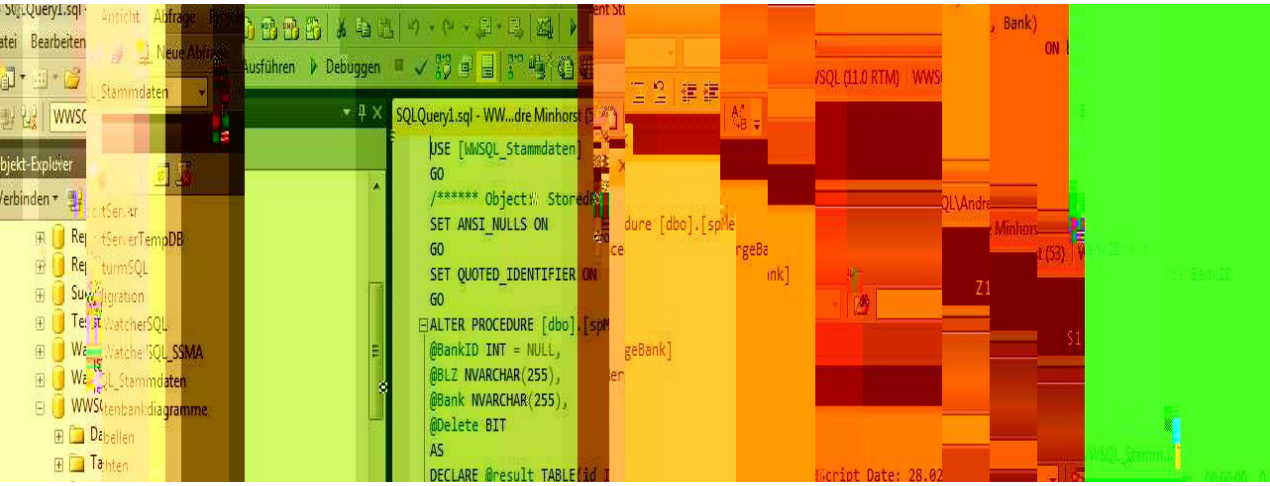
Als Administrator Ihrer SQL Server-Instanz werden Sie hier und da SQL-Anweisungen ausführen. Das SQL Server Management Studio bietet Ihnen nicht nur die Möglichkeit der Ausführung socher SQL-Anweisungen, sondern Sie können diese auch als SQL-Skripte speichern und sogar in Projekten organisieren.

6.3.1 Das Abfragefenster

Zu Beginn werden Sie einfache kleinere SQL-Anweisungen im SQL Server Management Studio ausführen – sei es, um kurz die Daten einer Tabelle auszugeben oder neue Datenbankobjekte anzulegen beziehungsweise bestehende zu ändern.

Diese SQL-Anweisungen geben Sie in Abfragefenstern ein, die Sie über die Symbolleiste mit der Schaltfläche *Neue Abfrage* öffnen. Alternativ ist auch die Tastenkombination *Alt + N* möglich. Dabei wird jedes neue Abfragefenster in einer eigenen Registerkarte geöffnet.

... DIE ÜBRIGEN SEITEN GIBT'S GEDRUCKT UNTER SHOP.MINHORST.COM



BERND JUNGBLUTH
ANDRÉ MINHORST

André Minhorst Verlag

7 Datenbanken und Tabellen erstellen

Bei der Migration werden Sie die Tabellen und Daten der Access-Datenbank vermutlich mit dem einen oder anderen Assistenten zum SQL Server übertragen.

Vielleicht erstellen Sie aber auch eine neue Anwendung auf Basis von Access und dem SQL Server oder Sie fügen einer migrierten Datenbank neue Tabellen hinzu beziehungsweise erstellen neue oder löschen bestehende Spalten.

In jedem Fall kann es nicht schaden, die Methoden vom SQL Server zum Erstellen, Ändern und Löschen von Datenbanken, Tabellen und Spalten zu kennen.

In diesem Kapitel stellen wir Ihnen die Werkzeuge für die Arbeit mit Datenbanken und Tabellen vor. Ergänzend dazu erhalten Sie weitere Informationen – beispielsweise eine Aufstellung der unterschiedlichen Dateitypen einer Datenbank sowie eine Gegenüberstellung der Datentypen vom SQL Server zu den üblicherweise in Access verwendeten Datentypen. Außerdem erfahren Sie, wie Sie mit T-SQL Datenbanken, Tabellen, Spalten und Beziehungen erstellen.

Zur Arbeit mit den Datenbanken und Tabellen verwenden Sie das SQL Server Management Studio, dass in Kapitel »SQL Server Management Studio«, Seite 135, beschrieben ist.

7.1 Begriffsklärung

Nachfolgend finden Sie die Erläuterung einiger wichtiger Begriffe in Zusammenhang mit dem Erstellen und Verwenden von Datenbanken.

7.1.1 Dateiendungen

Beim Erstellen und beim Umgang mit Datenbankdateien werden Sie die folgenden Dateiendungen kennenlernen:

» *.mdf*-Datei: Primäre Datenbankdatei (Master Data File). Enthält alle Informationen der Datenbank wie die Tabellen und die darin enthaltenen Daten, Sichten, Gespeicherten Prozeduren sowie die Benutzerrechte innerhalb der Datenbank.

» *.ndf*-Datei: Sekundäre Datenbankdatei. Wird im Zusammenhang mit größeren Datenbanken verwendet, die auf mehrere Datenträger aufgeteilt werden sollen oder müssen.

» *.ldf*-Datei: Datei zum Speichern der Transaktionsprotokolle. Protokolliert die Änderungen der in der Datenbank gespeicherten Daten.

Weitere Informationen zu den Datenbankdateien finden Sie unter »Wie speichert der SQL Server seine Daten?«, Seite 24.

7.1.2 Systemdatenbanken

Neben den von Ihnen erstellten Datenbanken zeigt der Objekt-Explorer des SQL Server Management Studios auch die Systemdatenbanken einer SQL Server-Instanz (siehe Abbildung 7.1). Hier kurz einige Informationen zu diesen notwendigen Datenbanken:

» *master*: Die zentrale Datenbank der SQL Server-Instanz. Sie enthält alle Informationen über die Instanz und der darin enthaltenen Datenbanken.

» *model*: Die Vorlage zum Erstellen einer neuen Datenbank. Jede neue SQL Server-Datenbank ist eine Kopie dieser Datenbank.

» *msdb*: Die Administrationsdatenbank vom SQL Server. Sie enthält die Aufträge des SQL Server-Agenten, die Backuphistorie, die Wartungspläne und vieles mehr.

» *tempdb*: Die "Auslagerungsdatei" des SQL Servers. Hier speichert der SQL Server Zwischenergebnisse von Abfragen, für die es im Arbeitsspeicher keinen Platz mehr gibt.

Weitere Inforamtionen über die Systemdatenbanken finden Sie unter »Welche Bedeutung haben die Systemdatenbanken?«, Seite 25.

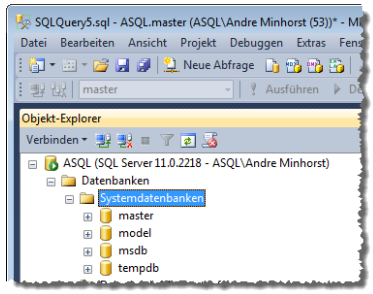


Abbildung 7.1: Systemdatenbanken im Objekt-Explorer des SQL Server Management Studios

7.2 Befehle in Abfragefenster eingeben

Wer sich schon in T-SQL oder in einen anderen SQL-Dialekt eingearbeitet hat, möchte Tabellen, Spalten, Beziehungen et cetera vielleicht per Skript erstellen. Zum Durchführen von Änderungen oder zum Löschen der verschiedenen Elemente gibt es entsprechende T-SQL-Befehle.

T-SQL-Anweisungen setzen Sie normalerweise von einem Abfragefenster des SQL Server Management Studios (nachfolgend kurz *SSMS*) aus ab. Vorher müssen Sie dem SQL Server mitteilen, auf welche Datenbank sich die Anweisungen beziehen. Dazu gibt es zwei Möglichkeiten:

» Sie wählen aus dem Kontextmenü der gewünschten Datenbank im Objekt-Explorer den Eintrag *Neue Abfrage* aus (siehe Abbildung 7.2).

» Oder Sie klicken einfach auf die Schaltfläche *Neue Abfrage* der Symbolleiste und geben dann einen Befehl ein, um die gewünschte Datenbank auszuwählen (siehe Abbildung 7.3).

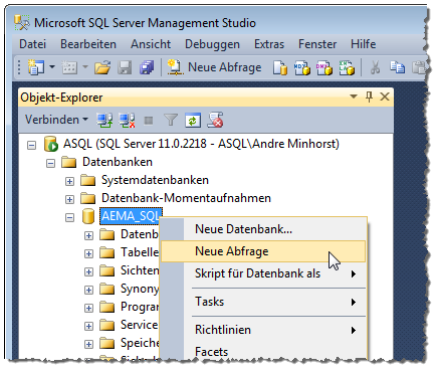


Abbildung 7.2: Öffnen einer neuen Abfrage zum Absetzen von T-SQL-Anweisungen

Der Befehl heißt *USE* und erwartet den Namen der zu verwendenden Datenbank als Parameter – unserem Fall also *AEMA_SQL*:

```
USE AEMA_SQL;
```

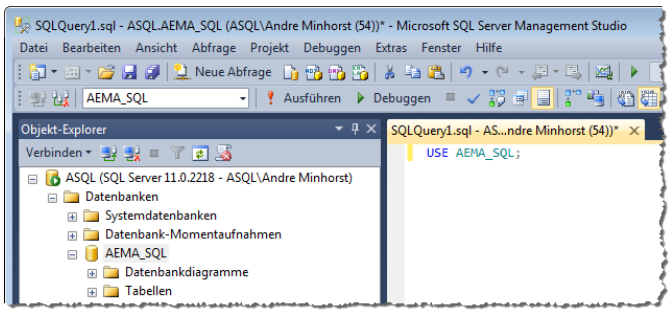


Abbildung 7.3: Anlegen einer neuen Abfrage und anschließendes Auswählen der Zieldatenbank

Um diesen Befehl auszuführen, betätigen Sie einfach die Taste *F5*. Dabei muss sich der Fokus im Abfragefenster befinden. Sollten bereits weitere Befehle im Abfragefenster enthalten sein, können Sie den auszuführenden Befehl markieren und diesen allein per *F5* ausführen.

Der SQL Server zeigt im unteren Bereich des Abfragefensters den Namen der Datenbank an, auf den sich die abgesetzten Anweisungen beziehen (siehe Abbildung 7.4).

Aber wäre es nicht komfortabler, Datenbanken, Tabellen, Spalten und Beziehungen über die entsprechenden Entwurfsansichten anzulegen? Das kommt darauf an, wie gut Sie sich sowohl mit der Benutzeroberfläche als auch mit den SQL-Befehlen auskennen. Während Benutzeroberflächen

von Zeit zu Zeit optischen und strukturellen Änderungen unterzogen werden, dürfte der grundlegende Sprachschatz von SQL sich so schnell nicht ändern.

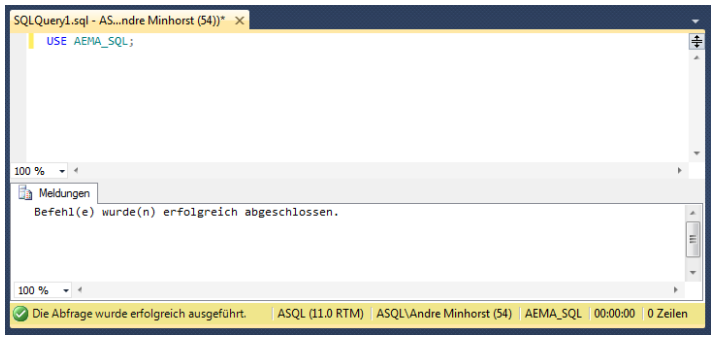


Abbildung 7.4: Anzeige der aktuellen Datenbank im Fußbereich des Abfragefensters

Ein weiterer Grund für den Einsatz von T-SQL ist, dass Sie im Download dieses Buches ein Tool finden, mit dem Sie T-SQL-Befehle auch von Access aus eingeben können. Das ist enorm praktisch, wenn Sie gerade mit der Access-Oberfläche arbeiten und nicht wegen jeder kleinen Änderung zwischen Access und dem SSMS wechseln möchten. Mehr zu diesem Tool erfahren Sie im Kapitel »Access-SQL Server-Tools«, Seite 459.

7.3 Neue Datenbank erstellen

Datenbanken erstellen Sie entweder über die Benutzeroberfläche des SSMS oder per T-SQL. T-SQL-Befehle setzen Sie dabei wiederum im SSMS über ein Abfragefenster ab oder von anderer Stelle – beispielsweise mit einer Pass-Through-Abfrage von Access aus. In den nachfolgenden Abschnitten finden Sie auch einige grundlegende Informationen zu den Begriffen rund um die SQL Server-Datenbank.

7.3.1 Neue Datenbank per SSMS erstellen

Eine neue Datenbank erstellen Sie im SSMS mit dem Kontextmenü-Eintrag *Neue Datenbank...* des *Datenbanken*-Elements im Objekt-Explorer (siehe Abbildung 7.5).

Dies öffnet den Dialog *Neue Datenbank*, mit dem Sie weitere Eigenschaften festlegen – allen voran den Datenbanknamen, zum Beispiel *AccessSQL*. Dieser Name wird auch als Dateiname der beiden standardmäßig angelegten Datenbankdateien verwendet, also etwa *AccessSQL.mdf* und *AccessSQL_log.ldf*. Den Datenbanknamen können Sie später leicht ändern, dies wirkt sich jedoch nicht auf die Namen der Datenbankdateien aus. Im Dialog stellen Sie auch ein Benutzerkonto für den Besitzer der Datenbank ein. Empfehlenswert ist hier die Angabe des SQL Server-Kontos *sa*. (siehe Abbildung 7.6).

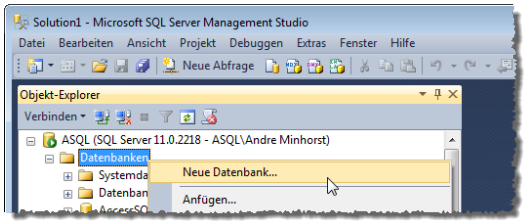


Abbildung 7.5: Erstellen einer Datenbank über den Objekt-Explorer

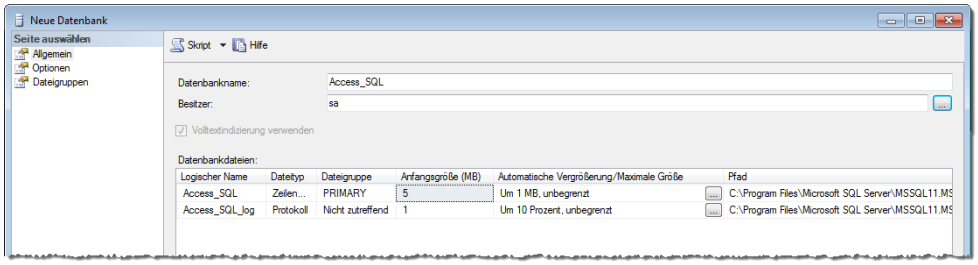


Abbildung 7.6: Festlegen der Eigenschaften der neuen Datenbank

Unter *Datenbankdateien* legen Sie den Speicherort der beiden Datenbankdateien fest und definieren deren Ausgangsgröße. Wenn möglich, speichern Sie die Datenbankdateien nicht auf ein und demselben Laufwerk. Durch das Trennen der Datenbankdateien auf separate Laufwerke erzielen Sie eine bessere Performance. Die Ausgangsgröße der neuen Datenbank sollte zum einen größer sein als die der Vorlagendatenbank *model*, damit die dort enthaltenen Elemente auch in der neuen Datenbank aufgenommen werden können. Zum anderen ist die Größe so zu dimensionieren, dass sie für eine Weile ausreicht. Zwar kann der SQL Server die Datenbank bei Bedarf vergrößern, dies geht jedoch zu Lasten der Performance.

Den Faktor zum Vergrößern der jeweiligen Datenbankdatei bestimmen Sie über die Schaltflächen mit den drei Punkten (...) der Spalte *Automatische Vergrößerung/Maximale Größe*. Dieses können Sie in absoluter oder relativer Form angeben: Wenn die Datenbank schnell größer wird, sollten Sie eine Prozentangabe tätigen, damit die Anzahl der Vergrößerungsvorgänge gering bleibt. Dies gilt jedoch nur, solange die Datenbank eine überschaubare Größe hat und genügend Festplattenplatz vorhanden ist. Schließlich belegt eine 60 GB große Datenbank nach einer Vergrößerung von 10 % direkt 6 GB mehr Plattenplatz. In solchen Fällen ist eine Vergrößerung in angegebenen MB-Schritten sinnvoller.

Auf der zweiten Registerseite wählen Sie das Wiederherstellungsmodell für die Datenbank aus. Mehr Informationen zum Wiederherstellungsmodell finden Sie im Kapitel *»Sichern und Wiederherstellen«, Seite 499*. Zudem können Sie hier die Sortierung der Datenbank angeben, sollte diese von der Standardsortierung der SQL Server-Instanz abweichen. Die Sortierung ist maßgebend für die Reihenfolge, in der später bei Abfragen Sortierungen ausgeführt werden.

Der Kompatibilitätsgrad gilt eher für SQL Server-Datenbanken, die von einer älteren SQL Server-Version auf die aktuelle migriert wurden. Für neue Datenbanken verwenden Sie am besten die aktuellste Version und somit die Standardeinstellung.

Alle anderen Optionen sind für das Anlegen einer einfachen Datenbank zunächst nicht so wichtig. Hier behalten Sie die Standardwerte einfach bei. Dies gilt auch für die Einstellungen der dritten Registerseite *Dateigruppen*. Sie können die neue Datenbank nun also mit einem Klick auf *OK* anlegen.

7.3.2 Neue Datenbank per T-SQL erstellen

Alternativ zur Benutzeroberfläche erstellen Sie eine Datenbank mit dem folgenden Befehl:

```
CREATE DATABASE <Datenbankname>;
```

Die erfolgreiche Ausführung dieses Befehls wird im unteren Bereich des Abfragefensters gemeldet (siehe Abbildung 7.7).

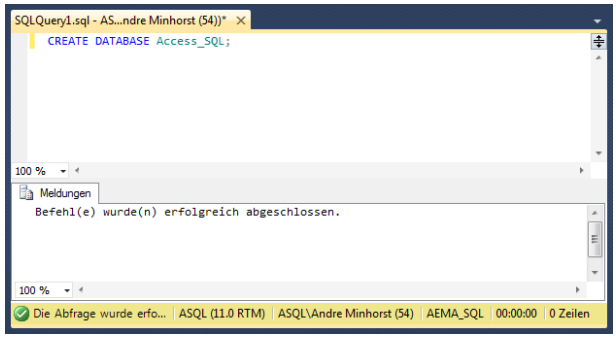


Abbildung 7.7: Erfolgreiche Erstellung einer Datenbank mit T-SQL

Nach dem Aktualisieren mit dem Kontextmenü-Eintrag *Aktualisieren* des Elements *Datenbanken* im Objekt-Explorer vom SSMS zeigt ein neuer Eintrag, dass die Datenbank tatsächlich angelegt wurde (siehe Abbildung 7.8).

Geben Sie bei der *CREATE DATABASE*-Methode keine weiteren Parameter außer dem Datenbanknamen an, legt der SQL Server den Speicherort für die Datenbankdateien selbstständig fest. Dabei verwendet er die Verzeichnisse, die Sie bei der Installation des SQL Servers angegeben haben.

Normalerweise jedoch bestimmen Sie die Speicherorte und andere Informationen beim Anlegen der Datenbank selbst:

```
CREATE DATABASE AccessSQL2
    ON PRIMARY (
        NAME = 'AccessSQL2',
```

```
    FILENAME = 'C:\Datenbanken\MSSQL11.MSSQLSERVER\MSSQL\DATA\AccessSQL2.mdf',
    SIZE = 4160KB,
    MAXSIZE = UNLIMITED,
    FILEGROWTH = 1024KB )
LOG ON (
    NAME = 'AccessSQL2_log',
    FILENAME = 'D:\Datenbanken\MSSQL11.MSSQLSERVER\MSSQL\DATA\AccessSQL2_log.ldf',
    SIZE = 1040KB,
    MAXSIZE = 2048GB,
    FILEGROWTH = 10%)
```

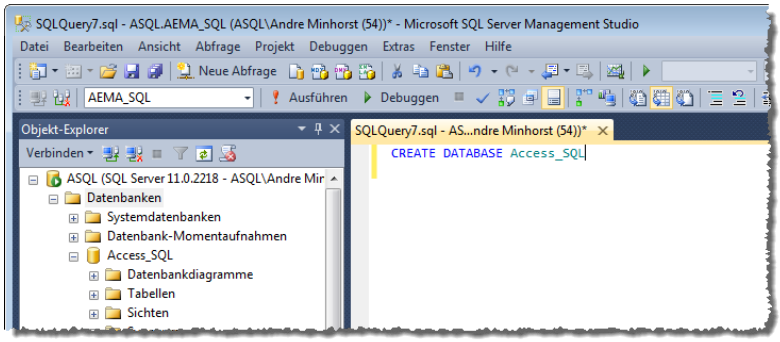


Abbildung 7.8: Die neue Datenbank im Objekt-Explorer

Da eine SQL Server-Datenbank aus mindestens zwei Datenbankdateien besteht, werden hier zweimal fast identische Informationen verwendet. Der erste Teil hinter dem Schlüsselwort *PRIMARY* erstellt die Datenbankdatei mit den eigentlichen Daten. Die zweite, deren Eigenschaften hinter dem Schlüsselwort *LOG* angegeben werden, dient als Transaktionsprotokolldatei.

Es kann noch weitere Datenbankdateien geben, die erste jedoch wird mit *PRIMARY* ausgezeichnet und muss unbedingt vorhanden sein. Dieser ersten Datenbankdatei widmet sich dann auch der obere Abschnitt: Hier wird der logische Name der Datenbankdatei angegeben, der Speicherort und Dateiname, die Größe beim Erstellen, die maximale Größe (hier *Unlimited* für unbegrenzt) und den Faktor zum Vergrößern der Datei. Hierbei gelten natürlich die gleichen Regeln wie beim Anlegen einer Datenbank über die Benutzeroberfläche.

Die Transaktionsprotokolldatei wird mit den gleichen Parametern definiert. Unterschiede gibt hier bei den Parameterwerten, wie bei der Endung der Dateinamen. Die primäre Datenbankdatei endet auf *.mdf*, die Transaktionsprotokolldatei auf *.ldf*.

Die Eigenschaften können Sie später in den Datenbankeigenschaften einsehen. Dazu wählen Sie den Kontextmenü-Eintrag *Eigenschaften* des *Datenbank*-Elements im Objekt-Explorer (siehe Abbildung 7.9). Um diese Eigenschaften ganz schnell zu erhalten, öffnen Sie mit *Strg + N* ein neues Abfragefenster und führen Sie die folgende Anweisung aus:

```
EXECUTE sp_helpdb AEMA_SQL;
```

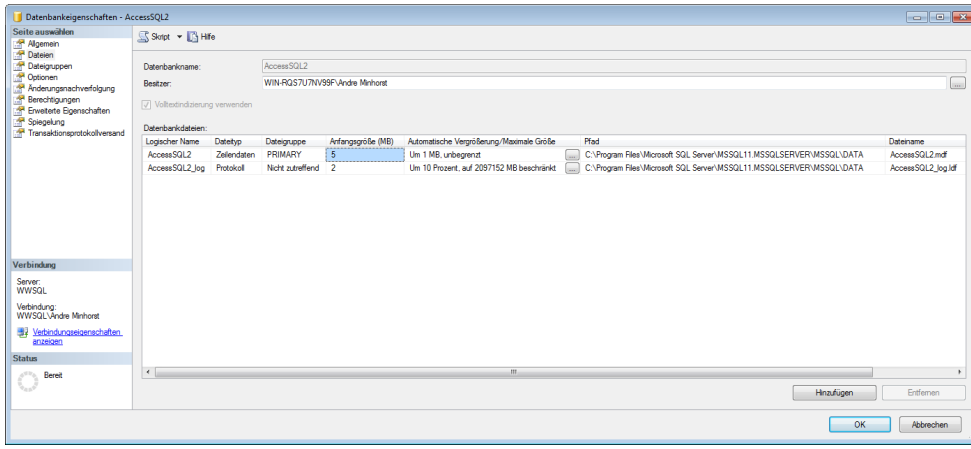


Abbildung 7.9: Eigenschaften der Dateien der Datenbank

7.3.3 Nach dem Anlegen

Wird die neue Datenbank nach dem Anlegen nicht direkt im Objekt-Explorer vom SSMS angezeigt, müssen Sie zunächst eines der übergeordneten Elemente aktualisieren – am einfachsten mit dem entsprechenden Kontextmenü-Eintrag *Aktualisieren* etwa des Elements *Datenbanken*.

Alternativ können Sie auch mit der folgenden Systemprozedur prüfen, ob die Datenbank angelegt wurde:

```
EXEC sys.sp_databases;
```

Diese gibt die Namen aller vorhandenen Datenbanken der aktuellen SQL Server-Instanz im Ergebnisfenster aus. Noch mehr Informationen erhalten Sie über die Abfrage der Sicht *sys.databases*:

```
SELECT name, database_id, create_date FROM sys.databases;
```

7.4 Datenbank löschen

Für das Löschen einer Datenbank ist es wichtig, dass diese gerade nicht verwendet wird. Den Rest erledigen Sie sowohl per SSMS als auch per T-SQL recht schnell.

7.4.1 Datenbank per SSMS löschen

Wählen Sie im Kontextmenü der Datenbank im Objekt-Explorer den Eintrag *Löschen* aus und klicken Sie im folgenden Dialog auf *OK*.

7.4.2 Datenbank per T-SQL löschen

Wenn Sie die Beispieldatenbank soeben angelegt haben, können Sie diese auch gleich einmal testweise löschen. Dazu geben Sie die folgende Anweisung im Abfragefenster ein:

```
DROP DATABASE <Datenbankname>;
```

7.5 Tabellen erstellen

Am Anfang erstellen Sie die Tabellen einer SQL Server-Datenbank höchstwahrscheinlich im Rahmen der Migration einer Access-Datenbank. Früher oder später werden Sie jedoch an den Punkt kommen, an dem Sie in der SQL Server-Datenbank neue Tabellen hinzufügen. Dazu können Sie die Benutzeroberfläche des SSMS oder auch T-SQL verwenden.

7.5.1 Tabelle per SSMS erstellen

Das SSMS bietet im Kontextmenü des Objekt-Explorer-Eintrags *Datenbanken|<Datenbankname>|Tabellen* den Befehl *Neue Tabelle ...* an (siehe Abbildung 7.10). Ein Klick auf diesen Befehl zeigt die Entwurfsansicht zum Erstellen von Tabellen. Hier können Sie, ähnlich wie in der Tabellenentwurfsansicht von Access, Spaltenname und Datentyp eingeben und festlegen, ob die Spalte *NULL*-Werte zulassen soll. Im unteren Bereich namens *Spalteneigenschaften* gibt es weitere Konfigurationsmöglichkeiten zu den einzelnen Spalten, beispielsweise die Angabe eines Standardwerts und die Definition einer Autowert-Funktion in der Gruppe *Identifikationspezifikation*. Mit der Tastenkombination *Strg + S* speichern Sie die Tabelle unter dem angegebenen Namen (siehe Abbildung 7.11).

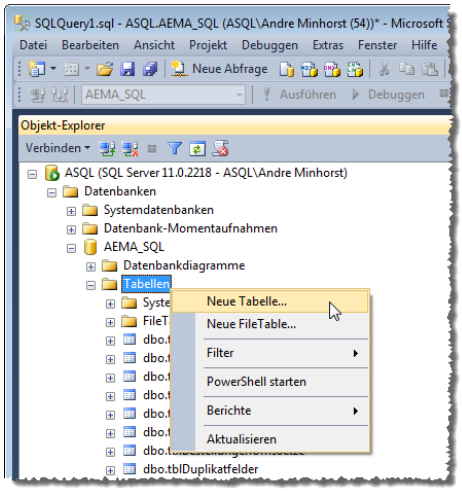


Abbildung 7.10: Erstellen einer neuen Tabelle im SQL Server Management Studio

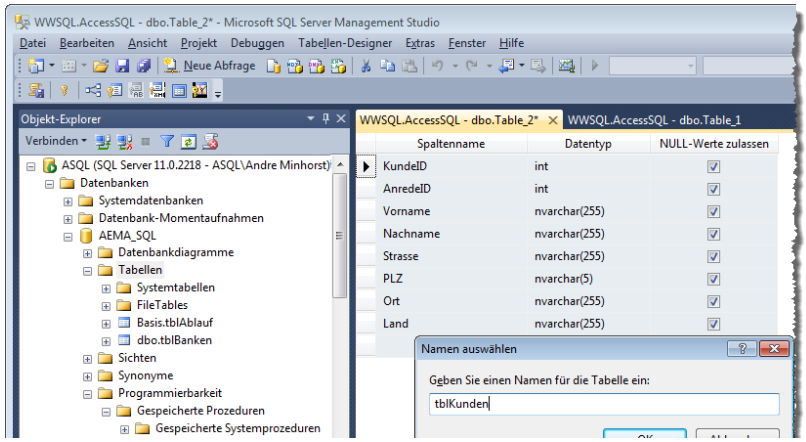


Abbildung 7.11: Anlegen einer Tabelle in der Entwurfsansicht des SQL Servers

7.5.2 Datentypen von Access nach SQL Server

SQL Server arbeitet mit anderen Datentypen als Access. Um Ihnen den Umstieg zu erleichtern, folgt hier eine Liste der SQL Server-Datentypen, die den gängigen Access-Datentypen entsprechen:

» *Autowert* (*Long Integer*): *integer* mit Spaltenoption *IDENTITY*

» *Autowert* (*Replikations-ID*): *uniqueidentifier* mit der Funktion *NEWID()* als Standardwert

» *Zahl (Byte)*: *tinyint*

» *Zahl (Integer)*: *smallint*

» *Zahl (Long Integer)*: *integer*

» *Zahl (Single)*: *real*

» *Zahl (Double)*: *float*

» *Zahl (Dezimal)*: *decimal* oder *numeric*

» *Zahl (Replikations-ID)*: *uniqueidentifier*

» *Währung*: *money*

» *Ja/Nein*: *bit* mit Option *NOT NULL*

» *Text*: *nvarchar(n)* für Unicode und *varchar(n)* für ANSI; *(n)* steht für die maximale Länge

» *Memo*: *ntext* oder *nvarchar(max)* für Unicode und *text* oder *varchar(max)* für ANSI

» *Hyperlink*: *ntext* oder *nvarchar(max)*, jedoch ohne Hyperlink-Funktionalität

» *Datum/Uhrzeit*: *datetime*

» *OLE-Objekt*: *image* oder *varbinary(max)*

» *Anlage*: *ntext* oder *nvarchar(max)*

» *Text* als *RichText*: *ntext* oder *nvarchar(max)* inklusive HTML-Tags

Bei der Vergabe der Datentypen sollten Sie beachten, dass die Datentypen *ntext*, *text* und *image* als veraltet gekennzeichnet sind und in einer der nächsten SQL Server-Versionen nicht mehr unterstützt werden.

7.5.3 Tabelle per T-SQL erstellen

Unter T-SQL erstellen Sie Tabellen mit der Anweisung *CREATE TABLE*. Diese erwartet als ersten Parameter das Schema, dem die Tabelle zugeordnet wird inklusive dem Tabellennamen.

Mehr zur Verwendung von Schemata lesen Sie in Kapitel »Sicherheit und Benutzerverwaltung«, Seite 395. Nach dem Tabellennamen folgen in Klammern die einzelnen Spaltennamen mit den jeweiligen Datentypen. In einem einfachen Fall sieht das so aus:

```
CREATE TABLE dbo.tblAnreden(
     AnredeID int NOT NULL,
     Anrede nvarchar(255) NULL,
     Briefanrede nvarchar(255) NULL);
```

Dies erzeugt schlicht eine Tabelle namens *dbo.tblAnreden* mit den drei Spalten *AnredeID*, *Anrede* und *Briefanrede* in den entsprechenden Datentypen. Das Feld *AnredeID* soll dabei keine *NULL*-Werte erlauben. Um sicherzustellen, dass eine *CREATE TABLE*-Anweisung auch im Kontext der richtigen Datenbank abgesetzt wird, verwenden Sie vorab die folgende Anweisung:

```
USE <Datenbankname>;
```

Vom erfolgreichen Anlegen der Tabelle können Sie sich im SSMS an zwei Stellen überzeugen: links im Objekt-Explorer unter dem Eintrag *Spalten* der angelegten Tabelle sowie auf der rechten Seite, wenn Sie aus dem Kontextmenü des Eintrags für die Tabelle den Befehl *Entwerfen* auswählen (siehe Abbildung 7.12).

Beachten Sie, dass Sie den Kontextmenü-Eintrag *Aktualisieren* eines der übergeordneten Objekte betätigen müssen, damit die neue Tabelle im Objekt-Explorer sichtbar ist.

7.6 Löschen einer Tabelle

Eine Tabelle löschen Sie mit der *DROP TABLE*-Anweisung, wobei Sie den Namen der Tabelle als Parameter angeben:

```
DROP TABLE <Tabellenname>;
```

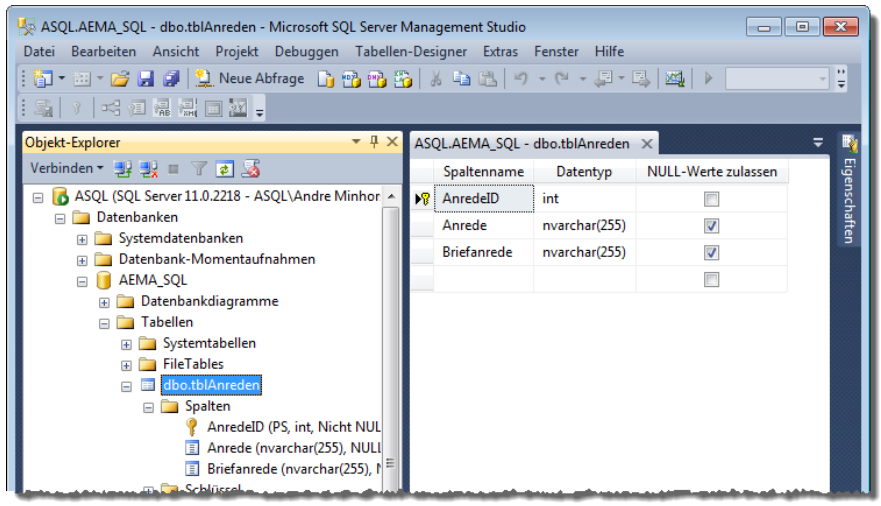


Abbildung 7.12: Die soeben erstellte Tabelle im Objekt-Explorer und in der Entwurfsansicht im SQL Server Management Studio

Im SSMS gelingt dies über den Kontextmenü-Eintrag *Löschen* der entsprechenden Tabelle und Anklicken der Schaltfläche *OK* in dem darauf folgenden Dialog.

Das Löschen einer Tabelle ist nur möglich, wenn aktuell kein Benutzer mit dieser Tabelle arbeitet und die zu löschende Tabelle nicht als primäre Tabelle in einer Fremdschlüsselbeziehung definiert ist.

7.6.1 Löschen und Neuerstellen einer Tabelle

Möchten Sie noch Änderungen am Entwurf einer frisch erstellten Tabelle durchführen, die noch keine Daten enthält, können Sie diese auch schnell löschen und neu erstellen.

Angenommen, die *CREATE TABLE*-Anweisung ist noch im Abfragefenster vorhanden, dann fügen sie einfach eine Existenzprüfung zur Tabelle mit der Systemfunktion *Object_ID* und davon abhängig den Befehl *DROP TABLE* ein.

Dies sieht für unser Beispiel wie folgt aus:

```
USE AccessSQL;
IF Object_ID('dbo.tblAnreden') Is Not Null
BEGIN
   DROP TABLE dbo.tblAnreden;
END;
CREATE TABLE dbo.tblAnreden(
      AnredeID integer NOT NULL IDENTITY(1,1),
      Anrede nvarchar(255) NOT NULL,
      Briefanrede varchar(255) NOT NULL);
```

Haben Sie die Änderungen gegenüber dem vorherigen *CREATE TABLE*-Aufruf bemerkt? Das Schlüsselwort *IDENTITY* legt fest, dass die Spalte *AnredeID* mit einem Autowert gefüllt wird.

Der erste Parameter steht dabei für den Startwert, der zweite für die Schrittweite – hier erhalten beide den Wert *1*. Das Schlüsselwort *NOT NULL* sorgt dafür, dass auch die Spalten *Anrede* und *Briefanrede* gefüllt werden müssen.

7.7 Ändern einer Tabelle

Einer Tabelle können Sie nachträglich Spalten hinzufügen, diese ändern oder bestehende Spalten entfernen. Ebenso ist es möglich, zu einer Tabelle Einschränkungen zu definieren wie Primärschlüssel, eindeutige Indizes, Fremdschlüsselbeziehungen, Standardwerte und Eingabeprüfungen.

7.7.1 Hinzufügen einer Spalte per T-SQL

Das nachträgliche Ergänzen einer Tabelle mit einer weiteren Spalte erledigen Sie mit der *ALTER TABLE*-Anweisung in Verbindung mit dem Schlüsselwort *ADD* – im Beispiel der Tabelle *tblAnreden* sieht dies wie folgt aus:

```
ALTER TABLE dbo.tblAnreden
ADD Adressanrede varchar(255);
```

7.7.2 Ändern einer Spalte per T-SQL

Auch die Eigenschaften einer Spalte lassen sich mit dem *ALTER*-Befehl ändern. Wenn Sie feststellen, dass 255 Zeichen für die Spalte *Adressanrede* doch zu wenig waren und dass diese Spalte keine *NULL*-Werte aufnehmen soll, können Sie dies wie folgt ändern:

```
ALTER TABLE dbo.tblAnreden
ALTER COLUMN Adressanrede varchar(100) NOT NULL;
```

Sie geben hier explizit an, dass Sie eine Spalte ändern möchten, und zwar die Spalte *Postleitzahl*.

Spalten im Produktivbetrieb ändern

Bei der Änderung des Datentyps einer Spalte legt der SQL Server die Tabelle komplett neu an, überträgt die Daten und löscht die alte Tabelle. Während der Entwicklungsphase ist dies unproblematisch, im laufenden Betrieb jedoch kann eine solche Änderung Probleme verursachen. Schließlich wird bei diesem Vorgang die Tabelle komplett gesperrt, weshalb Zugriffe auf die Tabelle gegebenenfalls mit Timeout-Fehlern quittiert werden.

Aus diesem Grund sind solche Änderungen standardmäßig nicht gestattet. Da sich jedoch während der Entwicklungsphase Änderungen an Tabellen kaum vermeiden lassen, können Sie die-

se mit einer speziellen Option im SSMS erlauben. Sie finden die Option im Optionen-Dialog (*Extras|Optionen*) im Bereich *Designer|Tabellen- und Datenbank-Designer* (siehe Abbildung 7.13).

Hier deaktivieren Sie für die Entwicklungsphase die Option *Speichern von Änderungen verhindern, die die Neuerstellung von Tabellen erfordern*. Für den Produktivbetrieb sollten Sie diese Option jedoch wieder aktivieren, um Timeouts zu vermeiden.

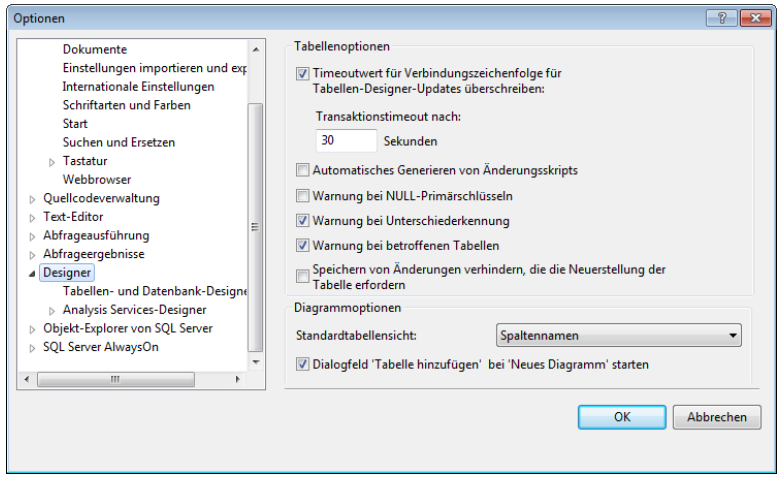


Abbildung 7.13: Einstellen der Option, die das Ändern von Spalten einer Tabelle erlaubt

7.7.3 Ändern eines Spaltennamens per T-SQL

Spaltennamen lassen sich nicht so einfach mit der *ALTER TABLE*-Anweisung ändern. In diesem Fall verwenden Sie die Systemprozedur *sp_rename*.

Wollen Sie etwa das Feld *BLZ* der Tabelle *tblBanken* in *Bankleitzahl* umbenennen, verwenden Sie folgende Anweisung:

```
EXEC sp_rename 'tblAnreden.Adressanrede', 'AnredeAdressblock', 'COLUMN'
```

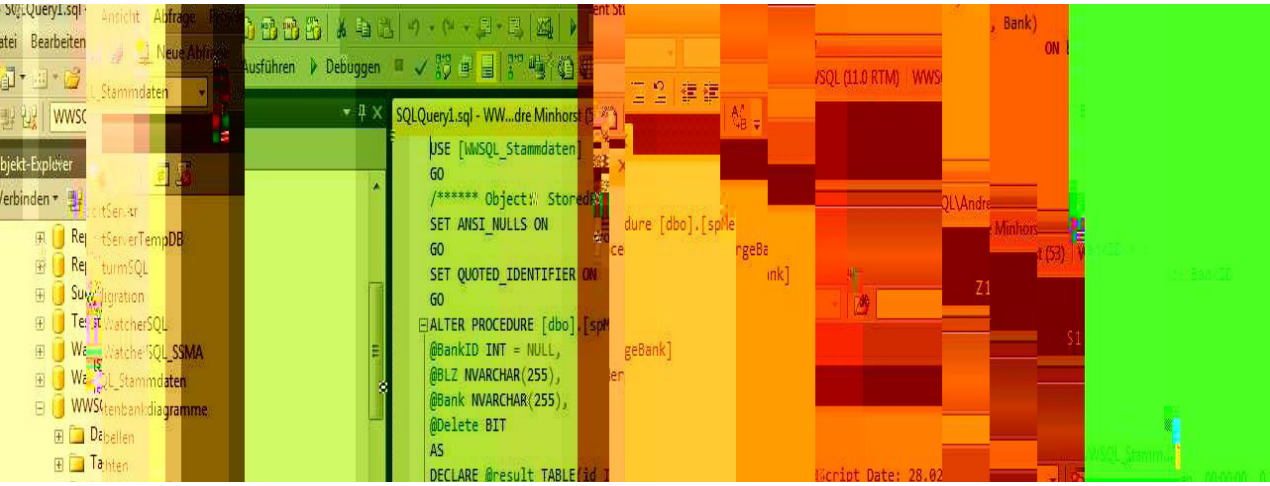
Der letzte Parameter mit dem Wert *COLUMN* teilt der Systemprozedur mit, um welchen Objekttyp es sich handelt.

7.7.4 Löschen einer Spalte per T-SQL

Gelegentlich legt man auch eine Spalte zu viel an. Auch diese werden Sie mit T-SQL leicht wieder los. Dazu verwenden Sie die Schlüsselwörter *DROP COLUMN* innerhalb einer *ALTER TABLE*-Anweisung:

```
ALTER TABLE dbo.tblAnreden DROP COLUMN AnredeAdressblock;
```

... DIE ÜBRIGEN SEITEN GIBT'S GEDRUCKT UNTER SHOP.MINHORST.COM



BERND JUNGBLUTH
ANDRÉ MINHORST

André Minhorst Verlag

8 Abfragen migrieren

Mit der Migration der Tabellen Ihrer Anwendung zu einer SQL Server-Datenbank und dem Verknüpfen der SQL Server-Tabellen mit der Access-Anwendung ist zwar bereits ein kleiner Schritt getan – aber die Hauptarbeit steht noch bevor. Die bestehenden Abfragen, Formulare, Berichte und VBA-Routinen greifen über die Verknüpfung mit den Tabellen im SQL Server-Backend genauso auf die Daten zu wie zuvor – vorausgesetzt, die Verknüpfungen erhalten die gleichen Namen wie die migrierten Tabellen. Allerdings werden alle Zugriffe über die Jet-Engine gesteuert und dann an den SQL Server weitergeleitet. Diese Art des Datenzugriffs beinhaltet weitaus mehr Nachteile als Vorteile. Sie holen viel mehr aus der Kombination Access und SQL Server heraus, wenn Sie direkt auf die Daten im SQL Server zugreifen. Wie dies von Formularen, Steuerelementen, Berichten und VBA-Routinen aus gelingt, zeigen wir später – beispielsweise in den Kapiteln »Tabellen verknüpfen«, Seite 83, »SQL Server-Zugriff per VBA«, Seite 307, und »Formulare und Berichte«, Seite 345.

In diesem Kapitel liefern wir Informationen, warum Sie die vorhandenen Abfragen nicht weiterverwenden sollten und welche Alternativen es dazu im SQL Server gibt.

8.1 Access-Abfragen und SQL Server-Tabellen

Nachdem Sie die Tabellen des SQL Servers per Verknüpfung in Access eingebunden und dieser Verknüpfung den gleichen Namen gegeben haben, den auch die zuvor in Access befindliche Tabelle hatte, können Sie von Abfragen aus genauso auf die in der Tabelle enthaltenen Daten zugreifen wie zuvor. Allein durch das Migrieren der Tabellen zum SQL Server und das Einbinden erhalten Sie jedoch in der Regel keine Performance-Verbesserung. In vielen Fällen verschlechtert sich die Performance sogar.

Um dies zu verstehen, benötigen Sie ein paar Hintergrundinformationen. Was genau geschieht eigentlich bei der Ausführung einer Access-Abfrage die auf verknüpfte Tabellen? Dabei werden die folgenden Schritte durchgeführt:

» Der Abfrage-Optimierer der Jet-Engine erstellt einen Ausführungsplan.

» Anhand des Ausführungsplans entscheidet Access, wie die Abfrage an den SQL Server übergeben wird.

Wie die Jet-Engine die Abfrage an den SQL Server übergibt, können Sie nicht beeinflussen. Es gibt die folgenden Varianten:

» Die Jet-Engine übergibt die komplette Abfrage an den SQL Server, wo diese ausgeführt wird. Das Ergebnis wird dann an Access zurückgeliefert. Dies ist die optimale Variante, an der es nichts mehr zu verbessern gibt.

» Die Abfrage wird in mehrere SQL-Abfragen aufgeteilt. Der SQL Server gibt die Ergebnismengen der einzelnen SQL-Abfragen an den Client zurück. Dort wird die ursprüngliche Abfrage dann lokal ausgeführt. Bei dieser Variante werden in den meisten Fällen mehr Datensätze von SQL Server an Access übertragen als notwendig. Hier ist die Performance wieder abhängig vom Netzwerk und vom Client.

» Die Abfrage wird gar nicht an den SQL Server übergeben. Dann liefert der SQL Server die Daten aller an der Abfrage beteiligten Tabellen an Access, wo die Abfrage lokal ausgeführt wird. Im Grunde genommen haben wir hier dasselbe Verhalten, wie bei einer reinen Access-Anwendung mit Frontend und Backend. Die Performance ist wieder abhängig vom Netzwerk und vom Client.

Wann der Access-Abfrageoptimierer sich für welche Variante entscheidet, kann nicht beeinflusst werden. Auch lassen sich keine Regeln definieren, unter welchen Umständen welche Variante verwendet wird.

Das Zusammenspiel von Access und SQL Server mit dem SQL Server Native Client hat sich seit Access 2010 etwas verbessert. So werden Abfragen mit Verweisen auf Formularfelder nun als entsprechende Parameterabfragen an den SQL Server übergeben. Es gibt aber auch immer noch einfache SQL-Anweisungen, die nicht 1:1 an den SQL Server übertragen werden. Zum Beispiel eine *SELECT TOP*-Anweisung: Hier fordert Access vom SQL Server alle Daten und filtert diese dann lokal auf die tatsächliche Anzahl. Sie können das hier beschriebene Verhalten mit dem SQL Server Profiler bzw. mit den XEvents beobachten. Mehr dazu im Kapitel »Performance analysieren«, Seite 105.

Gegenüber der Vorgehensweise, die etwa beim Einsatz einer reinen Frontend-Backend-Lösung auf Access-Basis eingesetzt wird, ergibt sich also lediglich bei der ersten Variante des Abfrageoptimierers ein Vorteil. Diese kommt aber leider nur selten vor und die beiden anderen Varianten bieten keinen Vorteil gegenüber einer reinen Access-Lösung.

Wie aber vermeiden wir nun, dass der Access-Abfrageoptimierer die SQL-Anweisungen verändert und sorgen dafür, dass der SQL Server nur die tatsächliche SQL-Anweisung ausführt? Wie also nutzen wir wirklich die Fähigkeiten des SQL Servers? Die Kurzform der Antwort lautet: Wir migrieren nach den Tabellen auch noch die Abfragen zum SQL Server.

8.1.1 Vorteile migrierter Abfragen

Die Migration einer Abfrage macht je nach Abfrage eine Menge Arbeit. Was ist nun der Hintergrund der Migration – warum sollte eine Abfrage schneller Ergebnisse liefern, wenn sich der SQL-Code in der SQL Server-Datenbank befindet und nicht mehr in der Access-Datenbank? Ein wichtiger Grund ist die Menge der übertragenen Daten: Wenn Sie dem Abfrageoptimierer die Entscheidung überlassen, in welcher Art und Weise er die Daten einer Abfrage anfordert, werden in der Regel mehr Daten von SQL Server zu Access übertragen. Access selbst ermittelt dann aus diesen Daten das eigentliche Ergebnis. Befindet sich die Logik der Abfragen in Sichten

und gespeicherten Prozeduren auf dem SQL Server, führt dieser die SQL-Anweisung aus und übergibt nur das Ergebnis an Access. Um dies zu erreichen, müssen Sie in Access lediglich auf die Sichten und gespeicherten Prozeduren des SQL Servers zugreifen. Sie entlasten auf diese Weise nicht nur den Client und nutzen die Hardware des Servers, Sie reduzieren auch die Netzwerklast.

Es gibt noch einen weiteren guten Grund, die Abfragen vom SQL Server ausführen zu lassen: Die Ausführungspläne im Abfragecache. SQL Server speichert zu den am häufigsten verwendeten Objekten die zugehörigen Ausführungspläne im Abfragecache. Ein Ausführungsplan enthält den effizientesten Weg zur Datenermittlung. Durch das Vorhalten der Ausführungspläne muss dieser nicht jedes Mal aus Neue ermittelt werden. Dies spart bei der Ausführung Zeit und steigert die Gesamtperformance des Systems. Mehr zu den Ausführungsplänen und dem Abfragecache lesen Sie im Kapitel »Welchen Nutzen haben der Abfragecache und die Ausführungspläne?«, Seite 27.

8.1.2 Abfragetypen im SQL Server

Es gibt im SQL Server kein Pendant zum Access-Objekttyp *Abfrage*. Dort gibt es Sichten und gespeicherte Prozeduren. Welche der beiden Objekttypen Sie für welche Abfrage verwenden, ist abhängig vom Typ der Access-Abfrage.

8.2 Sichten

Sichten sind vergleichbar mit den Auswahl- beziehungsweise *SELECT*-Abfragen unter Access. Sie können damit, genau wie mit einer Access-Abfrage, beliebige Spalten und Zeilen aus einer oder mehreren Tabellen ermitteln. Die Spalten legen Sie durch die Feldliste fest, die Zeilen durch das Kriterium in der *WHERE*-Klausel der Abfrage und evtl. Gruppierungen mit der Klausel *GROUP BY*.

Achtung: Sie können keine Parameter festlegen, um beispielsweise die *WHERE*-Klausel zur Laufzeit anzupassen. Eine Sicht liefert also immer die durch die Kriterien der Sicht festgelegten Daten. Parameter werden von Sichten nicht unterstützt. Kurz und knapp: Eine Sicht besteht aus einer einfachen SELECT-Anweisung, möglicherweise ergänzt mit *WHERE*, *GROUP BY* und *HAVING*. Ein *ORDER BY* ist zwar auch möglich, aber nicht problemlos. Mehr dazu später.

Legen wir also einmal eine einfache Sicht an. Dazu starten Sie das SQL Server Management Studio und öffnen im Objekt-Explorer die migrierte Datenbank *AEMA_SQL*. Eine neue Sicht erstellen Sie über den Eintrag *Neue Sicht...* im Ordner *Sichten*. Im nun erscheinenden Dialog *Tabelle hinzufügen* wählen Sie die in der Sicht zu verwendenden Tabellen aus. Dort können Sie allerdings nicht nur Tabellen, sondern auch bereits erstellte *Sichten*, benutzerdefinierte *Funktionen* und sogenannte *Synonyme* auswählen (siehe Abbildung 8.1). Sichten lernen Sie gerade kennen, Funktionen stellen wir im Kapitel »Funktionen«, Seite 273, vor und *Synonyme* sind benutzerdefinierte Namen für Tabellen und Sichten, die über Verbindungsserver zur Verfügung stehen.

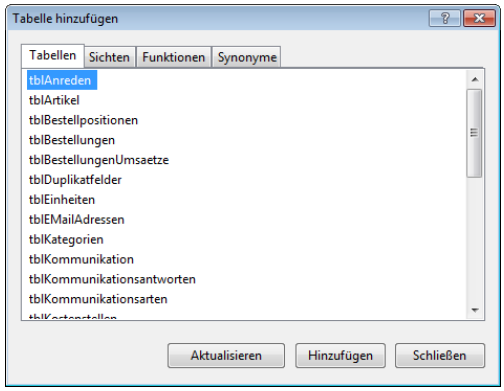


Abbildung 8.1: Auswahl der Datenherkunft der Sicht

Die erste Sicht soll die Artikel der Warengruppe *Magazine* inklusiver der Bezeichnung der Warengruppe liefern. Dazu wählen Sie die Tabellen *tblArtikel* und *tblWarengruppen* aus und übernehmen diese mit einem Klick auf *Hinzufügen* in den *Abfragedesigner*. Hier können Sie auch das Filterkriterium definieren. Der Abfragedesigner zeigt gleichzeitig den aktuellen SQL-Code für die View an. Durch einen Klick auf das rote Ausrufezeichen in der Symbolleiste wird die Ansicht noch um das Ergebnis der Sicht ergänzt (siehe Abbildung 8.2).

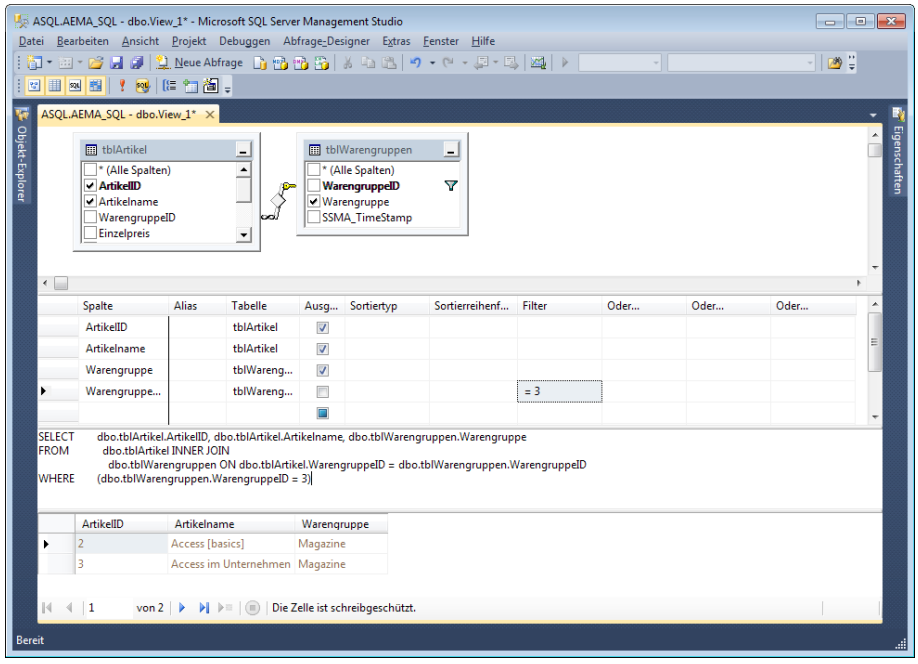


Abbildung 8.2: Erstellen einer View

Die Sicht speichern Sie wie in Access mit der Schaltfläche *Speichern* in der Symbolleiste (siehe Abbildung 8.3) und geben ihr anschließend eine Bezeichnung – beispielsweise *vArtikelWarengruppeMagazine.*

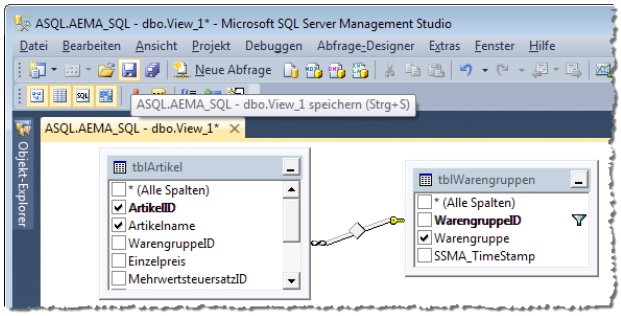


Abbildung 8.3: Speichern einer Sicht

Die neue Sicht sehen Sie nun im Ordner *Sichten*. Sollte dies nicht der Fall sein, müssen Sie die Auflistung des Ordners aktualisieren. Dazu wählen Sie im Ordner *Sichten* den Kontextmenübefehl *Aktualisieren*.

8.2.1 Sichten verwenden

Einen Blick in die Daten der Sicht liefert Ihnen der Eintrag *Oberste 1000 Zeilen auswählen* im Kontextmenü der Sicht. Dadurch wird ein neues Abfragefenster mit dem entsprechenden *SELECT TOP 1000*-Befehl geöffnet und der SQL-Befehl direkt ausgeführt (siehe Abbildung 8.4). Möchten Sie mehr als 1000 Datensätze sehen, entfernen Sie einfach die *TOP 1000*-Klausel und führen den SQL-Befehl erneut aus.

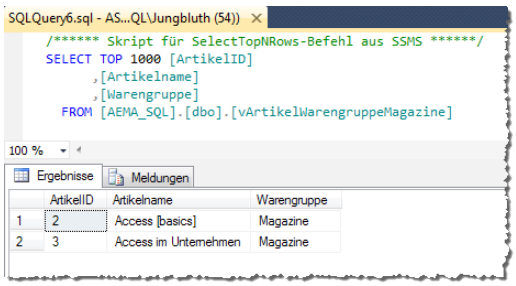


Abbildung 8.4: Das Ergebnis einer Sicht

In T-SQL verwenden Sie eine Sicht wie eine Tabelle. Die folgende Anweisung liefert Ihnen das Ergebnis der Sicht.

```
SELECT * FROM dbo.vArtikelWarengruppeMagazine;
```

Sie können Sichten auch bei Tabellenverknüpfungen mit *INNER JOIN*, *LEFT JOIN*, *RIGHT JOIN* und *CROSS JOIN* verwenden. Da Sie Sichten wie Tabellen ansprechen, ist die Verwendung von Sichten im SQL Server überall möglich – in SQL-Anweisungen, gespeicherten Prozeduren, Funktionen, Trigger und auch in anderen Sichten.

Wo genau Sie die Sicht überall verwenden, zeigt Ihnen die Objektabhängigkeit der Sicht. Diese Übersicht öffnen Sie ebenfalls über das Kontextmenü der Sicht – nun mit dem Eintrag *Abhängigkeiten anzeigen* (siehe Abbildung 8.5).

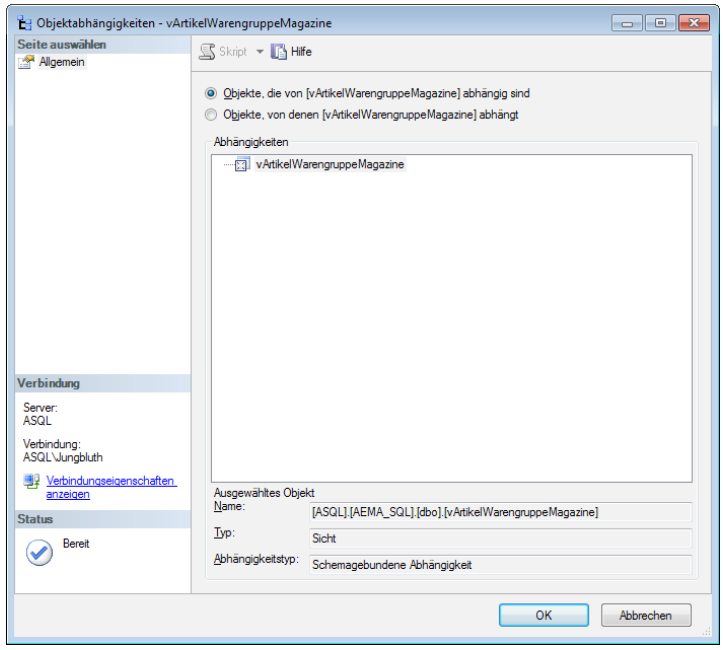


Abbildung 8.5: Die Verwendung der Sicht

8.2.2 Sichten ändern

Natürlich lässt sich eine Sicht auch wieder ändern. Ein Klick auf den Eintrag *Entwerfen* im Kontextmenü der Sicht öffnet diese im *Abfragedesigner*. Hier können Sie nun die Änderungen vornehmen und die Sicht wieder speichern. Allerdings sollten Sie sich vor der Änderung prüfen, in welchen Objekten die Sicht überall verwendet wird und dass die Änderung keine negativen Auswirkungen auf die Ausführung dieser Objekte hat.

8.2.3 Sicht per Skript erstellen

Der Abfragedesigner in allen Ehren, aber früher oder später verwenden Sie zum Anlegen von Sichten T-SQL. Es ist auch recht einfach: Sie erstellen die *SELECT*-Anweisung, prüfen das Ergebnis

und wenn Sie damit zufrieden sind, schreiben Sie vor der *SELECT*-Anweisung den Befehl *CREATE VIEW*, den Namen der Sicht und den Zusatz *AS*. Das war's dann auch schon.

Alles was Sie dazu benötigen, ist ein neues Abfragefenster. Dies öffnen Sie mit dem Kontextmenü-Eintrag *Neue Abfrage* der Datenbank – hier *AEMA_SQL*. Im folgenden Beispiel legen wir eine weitere Sicht an. Dieses Mal sollen die Artikel aller Warengruppen ermittelt werden.

Geben Sie den folgenden Code in das Abfragefenster ein:

```
CREATE VIEW dbo.vArtikelAllerWarengruppen
AS
SELECT dbo.tblArtikel.ArtikelID, dbo.tblArtikel.Artikelname, dbo.tblWarengruppen.
Warengruppe
FROM dbo.tblArtikel INNER JOIN dbo.tblWarengruppen
ON dbo.tblArtikel.WarengruppeID = dbo.tblWarengruppen.WarengruppeID;
```

Wenn Sie nun die Taste *F5* betätigen, führt der SQL Server die Anweisung aus und erstellt die Sicht.

Zum Ändern einer bestehenden Sicht verwenden Sie fast die gleiche Syntax wie oben. Der Unterschied ist, dass Sie statt des Schlüsselworts *CREATE* das Schlüsselwort *ALTER* verwenden:

```
ALTER VIEW dbo.vArtikelAllerWarengruppen
AS
SELECT dbo.tblArtikel.ArtikelID, dbo.tblArtikel.Artikelname, dbo.tblArtikel.Einzelpreis,
dbo.tblWarengruppen.Warengruppe
FROM dbo.tblArtikel INNER JOIN dbo.tblWarengruppen
ON dbo.tblArtikel.WarengruppeID = dbo.tblWarengruppen.WarengruppeID;
```

8.2.4 Sichten und sortierte Ausgaben

Sichten liefern keine sortierte Ausgabe der Ergebnismenge! Auch wenn Sie im Abfragedesigner eine Sortierung definieren können und ein Test dieser Abfrage die Daten dort auch sortiert ausgibt.

Der Abfragedesigner weist Sie beim Speichern der Sicht sogar mit der Meldung aus Abbildung 8.6 auf die fehlende Sortierung hin – wenn auch etwas kryptisch. Übersetzt könnte hier ebenso stehen: „Sie dürfen gerne ein *ORDER BY* in der Sicht definieren. Wenn Sie aber sicher sein wollen, dass Sie ein sortiertes Ergebnis erhalten, führen Sie die Sicht immer mit einem *ORDER BY* aus".

Bleibt die Frage, warum die Sicht mit einer Sortierung definiert werden kann, wenn bei der Verwendung der Sicht immer explizit eine Sortierung mit *ORDER BY* angegeben werden muss. Über die Antwort dieser Frage wird in den Microsoft-Foren schon seit Jahren diskutiert.

Um dies zu verdeutlichen – und um Ihnen einen Workaround vorzustellen – öffnen Sie die Sicht *vArtikelAllerWarengruppen* in der Entwurfsansicht des Abfragedesigners. Wählen Sie zur Ausgabespalte *Warengruppe* die Sortierung *Aufsteigend* in der Auswahlliste *Sortiertyp* aus und klicken Sie anschließend in der Symbolleiste auf die Schaltfläche mit dem roten Ausrufezeichen.

Im unteren Bereich des Abfragedesigners sehen Sie das Ergebnis sortiert nach der Warengruppe (siehe Abbildung 8.7).

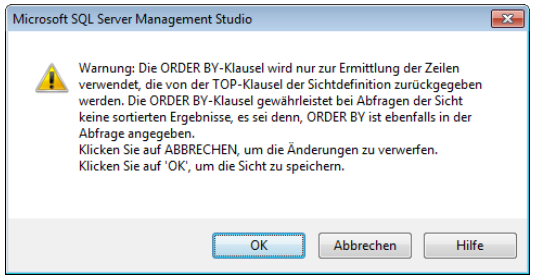


Abbildung 8.6: Warnhinweis beim Versuch, eine Sicht mit einer Sortierung anzulegen

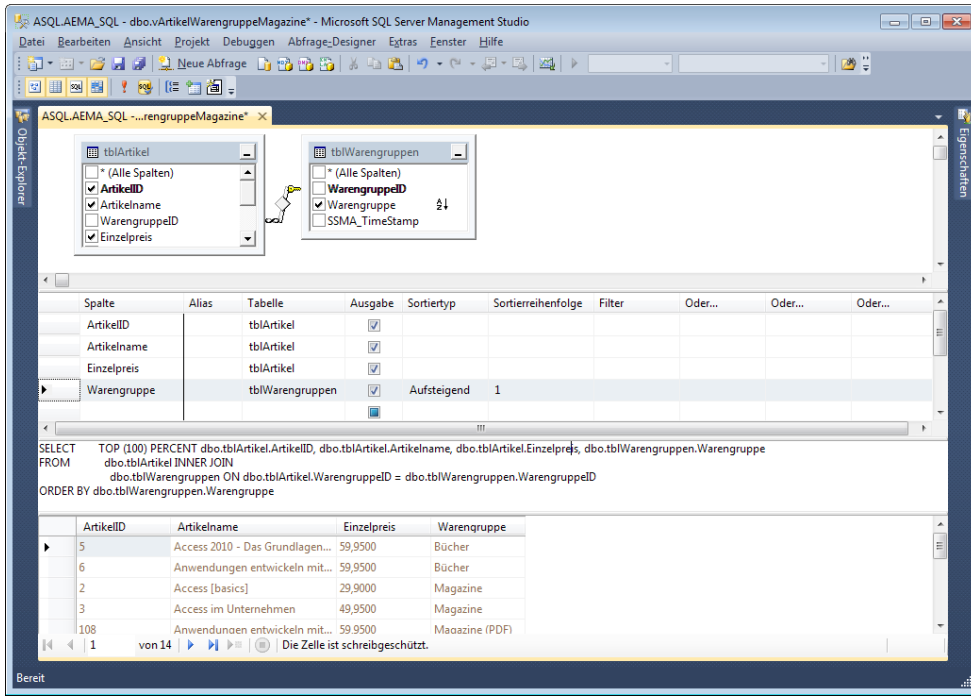
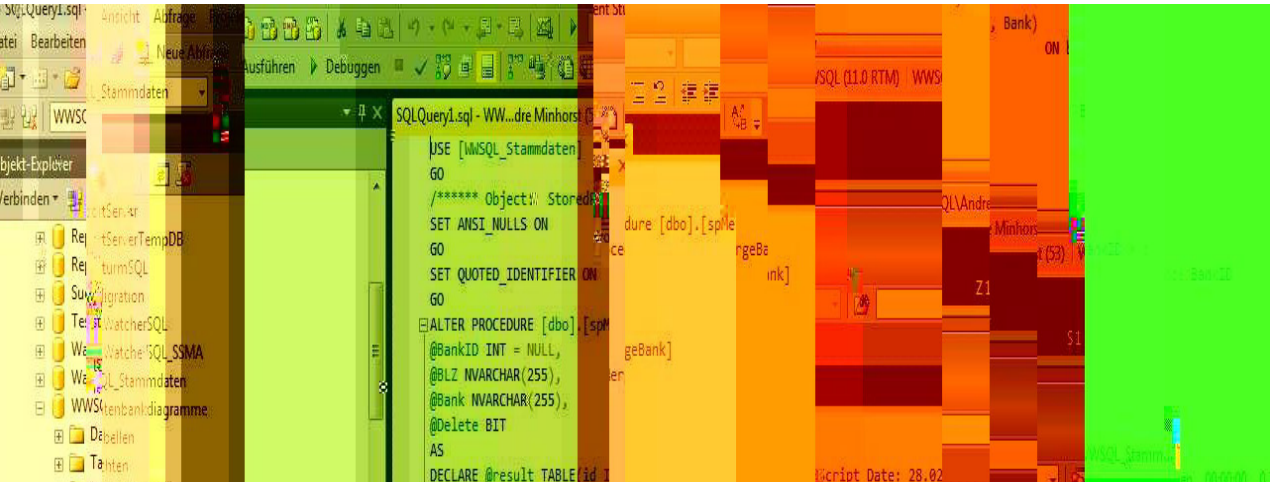


Abbildung 8.7: Die Sortierung in einer Sicht

Bei dieser Änderung wurde die *SELECT*-Anweisung nicht nur durch eine *ORDER BY*-Klausel, sondern auch um *TOP (100) PERCENT* erweitert. Dieser Zusatz definiert die *ORDER BY*-Klausel auf alle Datensätze des Ergebnisses. Beim Speichern der Sicht erhalten Sie den Hinweis aus Abbildung 8.7, die Sie mit einem Klick auf *OK* bestätigen.

Nun öffnen Sie ein neues Abfragefenster und führen dort die folgende *SELECT*-Anweisung aus:

... DIE ÜBRIGEN SEITEN GIBT'S GEDRUCKT UNTER SHOP.MINHORST.COM



BERND JUNGBLUTH
ANDRÉ MINHORST

André Minhorst Verlag

9 T-SQL-Grundlagen

In gespeicherten Prozeduren, Triggern und benutzerdefinierten Funktionen kommen Sie teilweise mit den üblichen SQL-Anweisungen wie *SELECT*, *UPDATE*, *CREATE* et cetera aus. Gelegentlich werden Sie jedoch Geschäftslogik auf den SQL Server auslagern wollen, weil dieser spezielle Aufgaben einfach schneller erledigen kann als wenn Sie dies von Access aus durchführen. Nun beherrscht der SQL Server kein VBA, und so müssen Sie gespeicherte Prozeduren, Trigger und Co. mit T-SQL-Befehlen und -Strukturen programmieren. Dieses Kapitel liefert die notwendigen Grundlagen für die Beispiele der entsprechenden Kapitel. Dabei bewegen wir uns vorerst auf SQL Server-Ebene – die Interaktion mit Access, etwa zum Übergeben von Parametern aus Access heraus an eine gespeicherte Prozedur, besprechen wir in den folgenden Kapiteln.

Einige Möglichkeiten in T-SQL

Die Befehle von T-SQL bieten eine Reihe Möglichkeiten, die wir uns in den folgenden Abschnitten ansehen. Dazu gehören die folgenden:

» Eingabe- und Ausgabeparameter nutzen

» Variablen, temporäre Tabellen und *Table*-Variablen für Zwischenergebnisse verwenden

» Programmfluss steuern

» Anweisungen in Schleifen wiederholt ausführen

» Daten hinzufügen, ändern und löschen

» Systemwerte abfragen

» Fehler behandeln

9.1 Grundlegende Informationen

Zum Einstieg einige wichtige Hinweise zum Umgang mit diesem Kapitel und den enthaltenen Techniken.

9.1.1 T-SQL-Skripte erstellen und testen

Vielleicht möchten Sie die hier abgebildeten Beispiele direkt ausprobieren. Alles was Sie dazu benötigen, ist das SQL Server Management Studio mit seinen Abfragefenstern. Ein neues Abfragefenster erhalten Sie mit der Tastenkombination *ALT + N*. Dabei sollten Sie vorher die Datenbank markieren, in der Sie das T-SQL-Skript ausführen möchten. Durch die Markierung wird das Abfragefenster mit der Verbindung zu dieser Datenbank geöffnet. Sie können diese Zuordnung

natürlich noch nachträglich ändern. Dazu wählen Sie entweder in der Symbolleiste die Datenbank über die Auswahlliste aus (siehe Abbildung 9.1) oder Sie geben am Anfang Ihres T-SQL-Skripts die folgende Anweisung ein:

```
USE <Datenbank>;
GO
```

Die *USE*-Anweisung wechselt zu der angegebenen Datenbank.

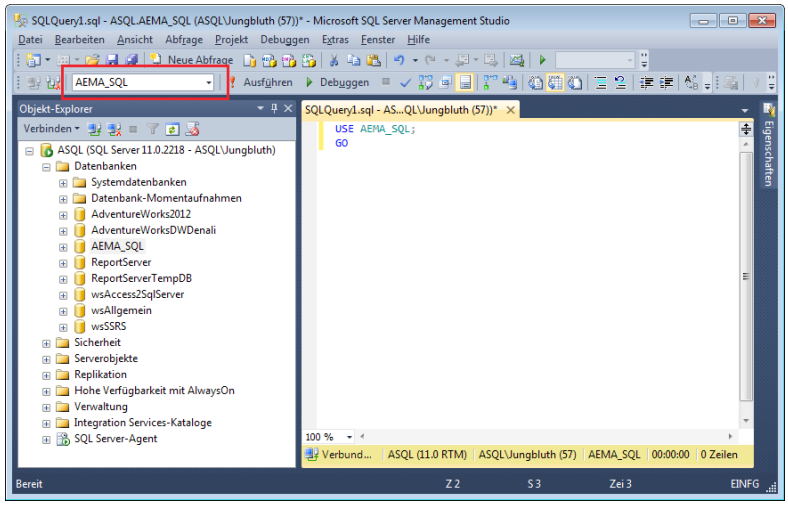


Abbildung 9.1: Die Auswahl der Datenbank für die Abfrage

Nachdem die Datenbank festgelegt ist, geben Sie im Abfragefenster die einzelnen SQL-Anweisungen zu Ihrem T-SQL-Skript ein. Dabei können Sie einzelne Anweisungen wie auch das komplette Skript mit *F5* ausführen. Das Ergebnis wird dann im unteren Bereich des Abfragefensters ausgegeben (siehe Abbildung 9.22). Mehr zu den Möglichkeiten, Abfragen zu erstellen und auszuführen, lesen Sie im Kapitel »SQL Server Management Studio«, Seite 135.

9.1.2 SELECT und PRINT

Mit *SELECT* ermitteln Sie nicht nur Daten aus einer oder mehreren Tabellen. Sie können mit *SELECT* auch Konstanten oder die Inhalte von Variablen ausgeben. Die Ergebnisse einer *SELECT*-Anweisung werden immer an den Client zurückgegeben. Im SQL Server Management Studio landen diese in der Registerkarte *Ergebnisse*. Bei einem Aufruf von Access heraus – beispielsweise über eine *Pass-Through*-Abfrage – werden die ermittelten Daten an Access übergeben.

Der *PRINT*-Befehl gibt lediglich Meldungen aus. Diese sehen Sie nach der Ausführung in der Registerkarte *Meldungen*. Bei einem längeren T-SQL-Skript, dass viele Verarbeitungsschritte durchführt und erst am Ende ein Ergebnis liefert, lassen sich mit *PRINT* Informationen zu ein-

zelnen Verarbeitungsschritten ausgeben – etwa Meldungen über die gerade eben ausgeführte SQL-Anweisung ergänzt mit der Anzahl der verarbeiteten Datensätze.

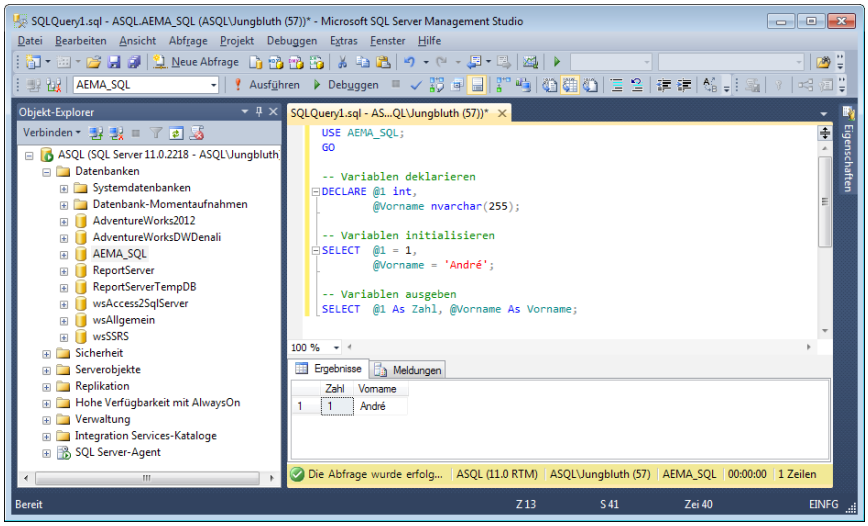


Abbildung 9.2: Ausführen von T-SQL-Skripten im Abfragefenster

SELECT liefert also immer Daten, während *PRINT* lediglich Meldungen ausgibt. Die folgenden beiden Abbildungen zeigen dies deutlich. Obwohl beide Anweisungen denselben Inhalt liefern, handelt es sich in Abbildung 9.3 um Daten und in Abbildung 9.4 lediglich um eine Meldung.

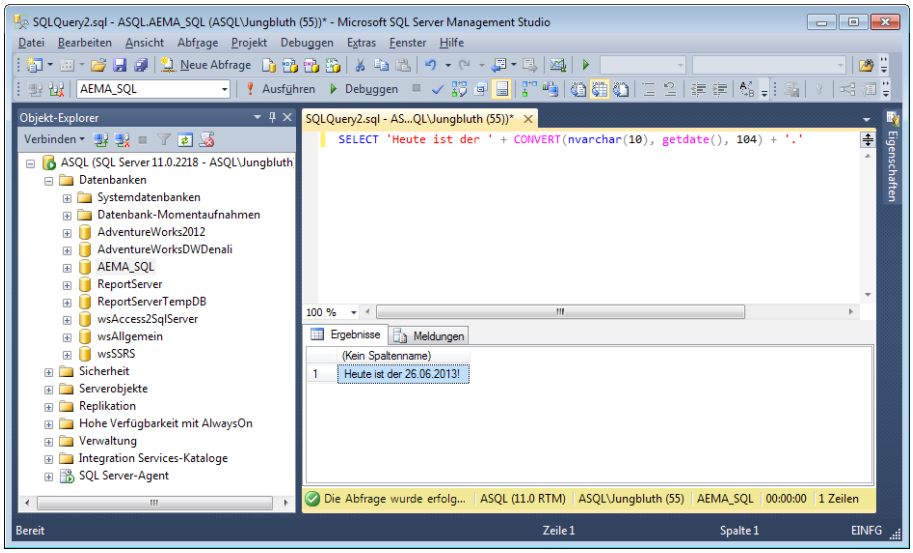


Abbildung 9.3: Ausgabe in das Meldungen-Fenster

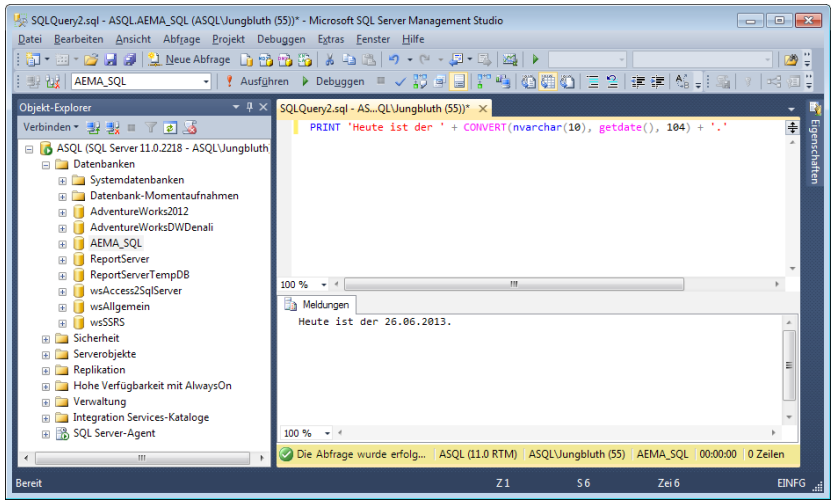


Abbildung 9.4: Ausgabe in das Ergebnisse-Fenster

9.1.3 Zeichenfolgen

Sie haben es vielleicht schon in den beiden vorherigen Abbildungen erkannt: Zeichenfolgen verkettet man unter T-SQL mit dem Plus-Operator (*+*). Und Literale werden ausschließlich in Hochkommata eingefasst, nicht in Anführungszeichen. Achtung: Wenn eines der per *+* verketteten Elemente den Wert *NULL* hat, wird der komplette Ausdruck zu *NULL*. Abhilfe schafft hier der seit SQL Server 2012 verfügbare Befehl *CONCAT*. Dieser ignoriert nur das Element mit dem *NULL*-Wert und verkettet alle anderen. In den Versionen vor 2012 müssen Sie die einzelnen Elemente in einer *CASE*-Anweisung auf mögliche *NULL*-Werte prüfen. (siehe Abbildung 9.5).

Zeichenketten auf mehrere Zeilen aufteilen

Wenn Sie eine Zeichenkette im SQL-Code zur besseren Lesbarkeit aufteilen wollen, fügen Sie einfach ein Backslash-Zeichen am Ende des ersten Teils ein und fahren Sie in der folgenden Zeile mit dem Rest fort:

```
PRINT 'Dieser Zweizeiler wird in \
einer Zeile ausgegeben.'
```

9.1.4 Datum

Beim Arbeiten mit einem Datumswert ist das Eingabeformat des Datums entscheidend – und das ist wiederum abhängig von der Standardsprache der SQL Server-Instanz. Ist diese *Deutsch*, arbeiten Sie mit dem deutschen Datumsformat, also Tag-Monat-Jahr. Steht die Standardsprache auf *Englisch*, werden Datumswerte im englischen Format (Monat-Tag-Jahr) abgelegt. Da kann schnell mal aus dem 06.07.2013 der 07.06.2013 werden.

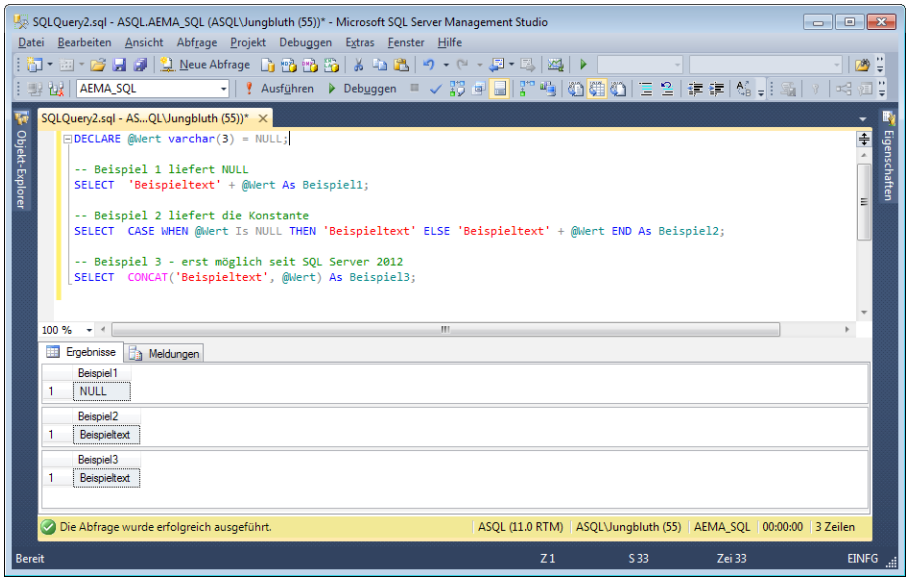


Abbildung 9.5: Verketten von Zeichenfolgen mit *NULL*-Werten

Arbeiten Sie in einer SQL Server-Instanz mit der Standardsprache *Englisch*, müssen Sie entweder bei einer SQL-Anweisung mit Datumswerten das Datum formatieren oder aber Sie ändern das Datumsformat für das gesamte T-SQL-Skript. Letzteres übernimmt folgender T-SQL-Befehl:

```
SET DATEFORMAT dmy;
```

Welche möglichen Datumsformate Ihnen zur Verfügung stehen, sehen Sie nach Ausführen dieser Systemprozedur:

```
EXEC sp_helplanguage;
```

Zum Formatieren eines Datums in das entsprechende Format stehen Ihnen seit SQL Server 2012 zwei Möglichkeiten zur Verfügung: *CONVERT* und *FORMAT*. Bis SQL Server 2012 war eine Formatierung des Datums nur über eine Konvertierung des Werts mit dem T-SQL-Befehl *CONVERT* möglich. Folgende Anweisung konvertiert das Datum des aktuellen Zeitpunkts in das ISO-Format:

```
SELECT CONVERT(nvarchar(10), GETDATE(), 112) As Datum_ISO;
```

Die Art der Formatierung wird durch den Parameter *Style* festgelegt. In diesem Beispiel ist dies der Wert *112*. Um nur die Uhrzeit aus einem Datumswert zu ermitteln, geben Sie den Wert *108* ein.

```
SELECT CONVERT(nvarchar(8), GETDATE(), 108) As Uhrzeit;
```

Die verfügbaren Werte für den Parameter *Style* entnehmen Sie bitte der SQL Server-Hilfe unter dem Stichwort *CONVERT und CAST*.

Mit *FORMAT* definieren Sie das Ausgabeformat ähnlich wie in VBA: Sie übergeben den Wert und das gewünschte Format. Ergänzend können Sie hier noch den Ländercode angeben. Dieser kann ausschlaggebend für das Ergebnis sein. So bleibt der Datumswert *06.07.2013* bei einer Formatierung mit dem Ländercode *de-de* weiterhin der 6. Juli 2013. Ändern Sie den Ländercode in *en-us*, wird das Datum als 7. Juni 2013 interpretiert (siehe Abbildung 9.6).

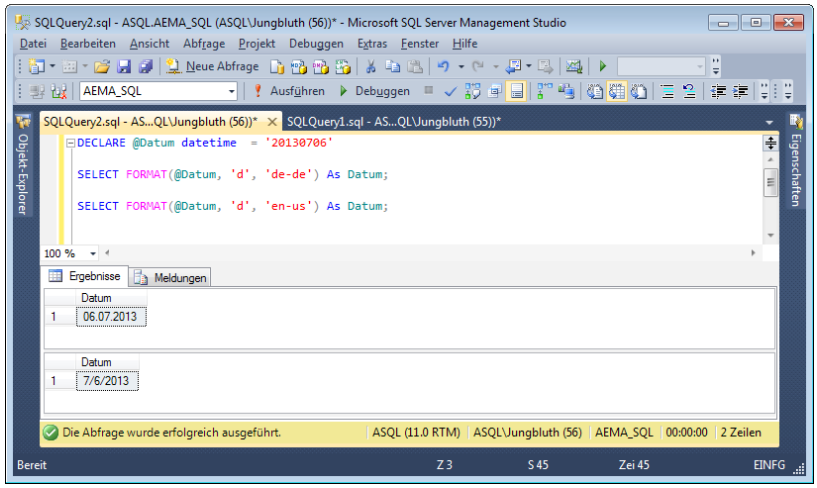


Abbildung 9.6: Formatierungen von Datumswerten

FORMAT und *CONVERT* formatieren Datumswerte nicht nur zur Ausgabe von Daten. Gerade in einer *WHERE*-Bedingung ist das korrekte Format des Datumswert entscheidend für das Ergebnis der *SELECT*-Anweisung. Sie müssen also das Format des Datumswerts in der *WHERE*-Bedingung dem der Datenbank anpassen.

Es geht auch einfacher: Sie verwenden in *WHERE*-Bedingungen das ISO-Format. Mit dem ISO-Format sind Sie auf der sicheren Seite. Folgende *SELECT*-Anweisung wird immer korrekt als der 01.08.2012 interpretiert:

```
SELECT BestellungID, KundeID, Bestelldatum FROM dbo.tblBestellungen
WHERE Bestelldatum = '20120801'
```

9.1.5 Das Semikolon

Anweisungen, die nicht Teil einer Struktur wie *IF...ELSE*, *BEGIN...END* et cetera sind, sollten Sie mit einem Semikolon abschließen. Dies führt zwar bei den meisten T-SQL-Befehlen nicht zu einem Fehler, ist jedoch SQL-Standard.

Bei einigen der neueren T-SQL-Befehle wie *MERGE* und bei der Verwendung einer *Common Table Expression* ist das Semikolon bereits Bestandteil der Syntax. Am besten gewöhnen Sie sich das Semikolon direkt an.

9.1.6 Code kommentieren

Wenn Sie Kommentare in T-SQL-Anweisungen einfügen möchten, können Sie dies auf zwei Arten erledigen:

» Um nur eine Zeile als Kommentar zu kennzeichen, stellen Sie dieser zwei Minuszeichen voran (--). Diese Art des Kommentars können Sie auch innerhalb einer Zeile nutzen, um zu verhindern, dass Bestandteile der SQL-Anweisung (beispielsweise die *WHERE*-Bedingung) ausgeführt werden.

» Möchten Sie mehrere aufeinanderfolgende Zeichen als Kommentar kennzeichen, fügen Sie vor der ersten Zeile die Zeichenfolge */** und hinter der letzten Zeile des Kommentars (am Ende der Zeile oder auch zu Beginn der folgenden Zeile) die Zeichenfolge **/* ein.

Beispiele:

```
-- Dies ist ein einzeiliger Kommentar.
SELECT * FROM dbo.tblArtikel -- WHERE WarengruppeID = 18;
/*
Dies ist ein
mehrzeiliger Kommentar.
*/
```

9.2 Variablen und Parameter

Genau wie jede andere Programmiersprache erlaubt auch T-SQL den Einsatz von Variablen. Im Vergleich etwa zu VBA sind Gültigkeitsbereich und Lebensdauer von Variablen sehr begrenzt:

Eine Variable ist maximal innerhalb der gespeicherten Prozedur, dem Trigger oder der Funktion gültig. Genau genommen kann dies noch weiter eingeschränkt werden – und zwar auf den sogenannten Batch. Darauf kommen wir später zu sprechen. Da der Gültigkeitsbereich einer Variablen derart eingeschränkt ist, nehmen wir in diesen Abschnitt gleich noch die Parameter hinzu. Parameter sind Variablen, die beim Aufruf einer gespeicherten Prozedur oder einer Funktion von der aufrufenden Instanz gefüllt werden können.

Auch hier gibt es eine Einschränkung gegenüber VBA: Einem Parameter können Sie unter T-SQL nur einen Wert zuweisen, aber keinen Verweis auf einen Wert. Das heißt, dass sich Änderungen an einem Parameter innerhalb einer gespeicherten Prozedur oder Funktion keinesfalls auf den Wert der Variablen in der aufrufenden Instanz auswirken.

9.2.1 Variablen deklarieren

Eine Variable muss zunächst deklariert werden. Dies geschieht mit der Anweisung *DECLARE*, die zwei Parameter erwartet: den mit führendem @-Symbol ausgestatteten Variablennamen und den Datentyp.

Der Datentyp entspricht den beim Erstellen von Feldern verwendeten Datentypen, also beispielsweise *int* oder *nvarchar(255)*. Eine Laufvariable deklarieren Sie also etwa wie in folgendem Beispiel:

```
DECLARE @i int;
```

Sie können mit einer *DECLARE*-Anweisung gleich mehrere Variablen deklarieren:

```
DECLARE @i int, @j int;
```

Eine Variable zum Speichern einer Zeichenkette deklarieren Sie so:

```
DECLARE @Vorname nvarchar(255);
```

Die Namen von Variablen und Parametern dürfen keine Leerstellen oder Sonderzeichen enthalten – mit Ausnahme des Unterstrichs.

9.2.2 Variablen füllen

Eine Variable wird mit der *SET*-Anweisung gefüllt. Im Falle der soeben deklarierten Variablen *@i* sieht das so aus:

```
SET @i = 1;
```

Eine Textvariable füllen Sie, indem Sie den Text in Hochkommata einfassen. Anführungszeichen (") sind nicht zulässig!

```
SET @Vorname = 'André';
```

Sie können Variablen in neueren SQL Server-Versionen (ab SQL Server 2008) auch direkt bei der Deklaration mit einem Wert füllen:

```
DECLARE @k int = 10;
```

Möchten Sie mehrere Variablen initialisieren, verwenden Sie anstelle einzelner *SET*-Anweisungen einfach eine *SELECT*-Anweisung:

```
SELECT @i = 1, @Vorname = 'André';
```

Variable aus Abfrage füllen

Variablen lassen sich auch mit dem Ergebnis einer Abfrage füllen. Für die Abfrage gelten die gleichen Regeln wie für eine mit der *IN*-Klausel von SQL verwendete Abfrage: Sie muss in Klammern eingefasst werden und darf nur einen einzigen Datensatz mit einem einzigen Feld zurückliefern. Im folgenden Beispiel liefert die Abfrage den kleinsten Wert für das Feld *BankID* der Tabelle *tblBanken*:

```
DECLARE @BankID int;
SET @BankID = (SELECT TOP 1 BankID FROM dbo.tblBanken ORDER BY BankID);
SELECT @BankID;
```

Die *TOP 1*-Klausel sorgt dafür, dass die Abfrage nur einen Datensatz liefert, und da nur ein Feld als Ergebnismenge angegeben ist, gibt die Abfrage auch nur einen Wert zurück.

Variable in Abfrage füllen

Es gibt noch eine alternative Schreibweise, bei der die Variable direkt in die Abfrage integriert wird:

```
DECLARE @BankID int;
SELECT TOP 1 @BankID = BankID FROM dbo.tblBanken ORDER BY BankID;
SELECT @BankID;
```

Achten Sie darauf, dass die Abfrage nur einen Datensatz liefert. Enthält das Abfrageergebnis mehrere Datensätze, wird der Variablen der Wert des letzten Datensatzes zugewiesen. Ein anderer, viel interessanterer Aspekt bei dieser Variante ist, dass Sie mehrere Werte gleichzeitig in Variablen schreiben können. Die folgende Anweisungsfolge gibt die drei Werte des gefundenen Datensatzes zwar nur im Ergebnisfenster aus, aber natürlich können Sie diese Werte auch auf ganz andere Art und Weise weiterverarbeiten:

```
DECLARE @BankID int;
DECLARE @Bank varchar(255);
DECLARE @BLZ varchar(8);
SELECT TOP 1 @BankID = BankID, @Bank = Bank, @BLZ = BLZ FROM dbo.tblBanken ORDER BY
BankID;
SELECT @BankID, @Bank, @BLZ;
```

Variable mit Funktionswert füllen

T-SQL bietet eine ganze Reihe nützlicher eingebauter Funktionen – zum Beispiel zum Ermitteln des aktuellen Zeitpunkts oder des aktuell angemeldeten Benutzers.

Auch damit können Sie Variablen füllen – hier mit dem aktuellen Zeitpunkt:

```
DECLARE @AktuellesDatum datetime;
SET @AktuellesDatum = GETDATE();
SELECT @AktuellesDatum;
```

9.2.3 Werte von Variablen ausgeben

Während Sie mit T-SQL programmieren, möchten Sie vielleicht zwischenzeitlich den aktuellen Wert einer Variablen ausgeben lassen. Dies erledigen Sie mit der *SELECT*-Anweisung:

```
SELECT @i As i, @Vorname As Vorname;
```

Soll die Ausgabe lediglich eine Information sein, verwenden Sie anstelle *SELECT* die Ihnen bereits bekannte *PRINT*-Anweisung. Der Wert der Variablen sehen Sie dann in der Registerkarte *Meldungen*.

```
PRINT 'I: ' + CAST(@i As nvarchar(5)) + ' Vorname: ' + @Vorname;
```

9.2.4 Variablen ohne Wertzuweisung

Wenn Sie einer Variablen keinen Wert zuweisen, enthält diese den Wert *NULL*. Dies gilt, im Gegensatz zu VBA, für alle Datentypen. Folgende Anweisungen liefern also den Wert *NULL* zurück:

```
DECLARE @j int;
SELECT @j;
```

9.2.5 Gültigkeitsbereich

Weiter oben haben wir bereits erwähnt, dass eine Variable maximal innerhalb einer gespeicherten Prozedur, eines Triggers oder einer Funktion gültig ist. Es gibt eine Einschränkung: Das Schlüsselwort *GO* löscht alle Variablen (siehe Abbildung 9.7).

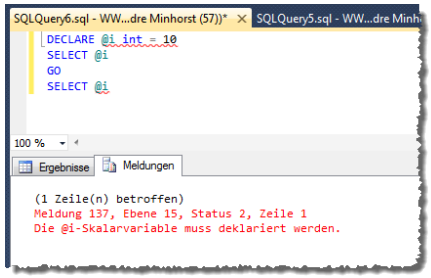


Abbildung 9.7: *GO* leert alle Variablen.

Das ist natürlich nicht der Sinn dieses Schlüsselworts. *GO* ist vielmehr dazu gedacht, den Code in verschiedene Batches zu unterteilen, die einzeln vom SQL Server abgearbeitet werden. Wozu braucht man das? Manche Anweisungen sind nur als erste Anweisung eines Batches zulässig. Dazu gehören die Anweisungen zum Erstellen von gespeicherten Prozeduren oder Sichten wie *CREATE PROC* oder *CREATE VIEW*.

9.2.6 Variablen einsetzen

Variablen können Sie beispielsweise in Auswahlabfragen einsetzen: Sie füllen die Variable mit einem Wert und setzen dann die Variable als Vergleichskriterium in einer *SELECT*-Abfrage ein:

```
DECLARE @BankID int;
SET @BankID = 12;
SELECT * FROM dbo.tblBanken WHERE BankID = @BankID;
```

9.2.7 Parameter

Parameter können Sie in gespeicherten Prozeduren oder in Funktionen einsetzen. Sie werden genau wie Variablen deklariert – mit dem Unterschied, dass die *DECLARE*-Anweisung entfällt.

Um den Kapiteln »Gespeicherte Prozeduren«, Seite 257, und »Funktionen«, Seite 273, vorzugreifen: Beim SQL Server legen Sie gespeicherte Prozeduren und Funktionen nicht einfach in einem Modul an, sondern Sie erstellen diese mit einer *CREATE*-Anweisung, ändern sie mit der *ALTER*-Anweisung und verwenden zum Löschen die DROP-Anweisung.

Übergabeparameter

Die Parameterliste wird in Klammern hinter dem Namen der gespeicherten Prozedur oder der Funktion angegeben. Das folgende Beispiel erstellt eine gespeicherte Prozedur. Die Parameterdefinition sehen Sie in der ersten Zeile:

```
CREATE PROC dbo.spBankNachID(@BankID int)
AS
SELECT * FROM tblBanken WHERE BankID = @BankID;
```

Um die gespeicherte Prozedur mit dem gewünschten Parameter auszuführen, verwenden Sie folgende Anweisung:

```
EXEC dbo.spBankNachID 210023;
```

Dies liefert die Werte aller Felder der Tabelle *tblBanken* für den per Parameter angegebenen Datensatz. Genau wie Variablen können Sie auch Parameter direkt bei der Deklaration vorbelegen:

```
CREATE PROC dbo.spAnrede(@AnredeID int = 1)
AS
SELECT Anrede FROM dbo.tblAnreden WHERE AnredeID = @AnredeID;
```

Rufen Sie die gespeicherte Prozedur ohne Parameter auf, liefert diese den Wert der Tabelle *tblAnreden* mit dem Wert *1* im Feld *AnredeID* zurück:

```
EXEC dbo.spAnrede;
```

Wenn Sie hingegen keinen Standardwert angeben und beim Aufruf keinen Parameter zuweisen, löst dies einen Fehler aus.

Rückgabeparameter

Gespeicherte Prozeduren können auch einzelne Werte zurückgeben. Diese werden in der Parameterliste mit dem Schlüsselwort *OUTPUT* hinter dem Datentyp gekennzeichnet – also beispielsweise wie folgt. Die Prozedur füllt den Parameter *@Rueckgabe* und gibt diesen zurück:

```
CREATE PROC dbo.spRueckgabewert(@Rueckgabe int OUTPUT)
AS
SET @Rueckgabe = 100;
```

Vor dem Aufruf der Prozedur deklarieren Sie eine Variable zum Aufnehmen des Rückgabewertes. Diese Variable übergeben Sie mit der Kennzeichnung *OUTPUT* an die gespeicherte Prozedur. Die *SELECT*-Anweisung gibt schließlich den zurückgegebenen Wert aus:

```
DECLARE @Ergebnis int;
EXEC dbo.spRueckgabewert @Ergebnis OUTPUT;
SELECT @Ergebnis;
```

9.3 Temporäre Tabellen

Manchmal möchten Sie vielleicht komplexere Datenstrukturen in Variablen zwischenspeichern – zum Beispiel einen oder mehrere komplette Datensätze. Hierzu bietet Ihnen SQL Server zwei Möglichkeiten: die temporären Tabellen und die *Table*-Variablen. Eine temporäre Tabelle erstellen Sie wie eine normale Tabelle – mit einem kleinen Unterschied: Der Name der temporären Tabelle beginnt mit dem Raute-Zeichen (#). Im folgenden Beispiel erstellen wir eine temporäre Tabelle mit den gleichen Feldern wie die Tabelle *tblAnreden*:

```
CREATE TABLE #tblAnreden(
    AnredeID int, Anrede nvarchar(10)
);
```

Diese Tabelle wird nicht in der Liste der Tabellen der Datenbank angezeigt. Wo aber erscheint sie dann? Bevor wir dies untersuchen, legen wir noch eine weitere Tabelle an – diesmal mit zwei Raute-Zeichen vor dem Tabellennamen. Dies bedeutet, dass das Objekt als globale temporäre Tabelle angelegt werden soll:

```
CREATE TABLE ##tblAnreden(
    AnredeID int, Anrede nvarchar(10)
);
```

Die beiden temporären Tabellen finden sich in der Sytemdatenbank *tempdb* (siehe Abbildung 9.88). Die Bezeichnung der lokalen temporären Tabelle wird noch durch einige Unterstriche (_) und eine Nummer ergänzt. Diese Maßnahme ergreift der SQL Server, damit jede Verbindung eine eigene lokale temporäre Tabelle dieses Namens anlegen kann. Sie können so beispielsweise eine solche Tabelle direkt im SQL Server Management Studio erzeugen und eine weitere in einer Verbindung, die Sie von einer Access-Datenbank aus erstellt haben. Der Gültigkeitsbereich dieser beiden Tabellenarten lässt sich leicht experimentell prüfen. Führen Sie einmal im SQL Server Management Studio einen Datensatz zur globalen temporären Tabelle hinzu:

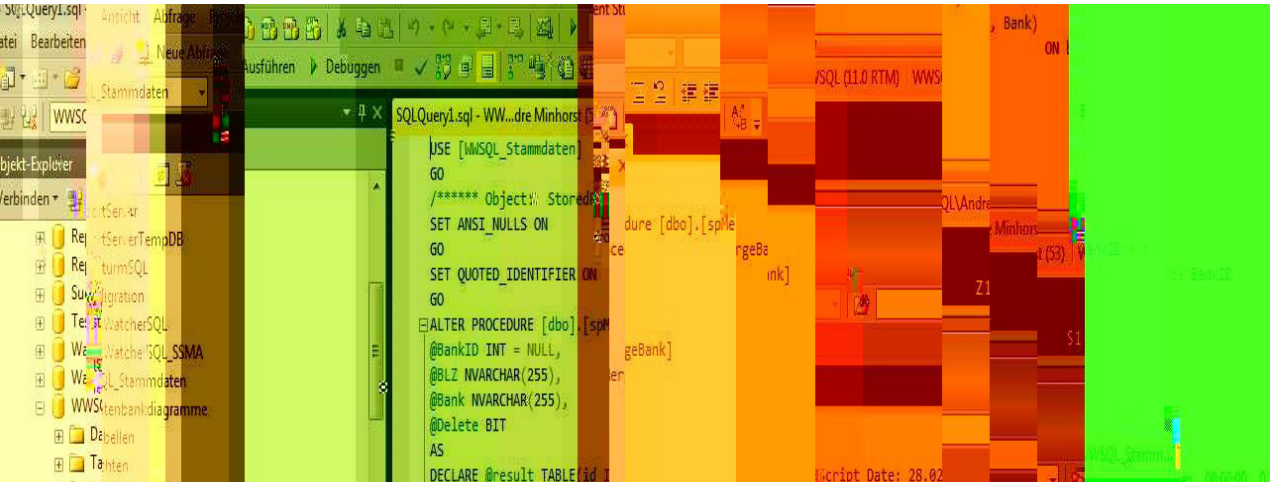
```
INSERT INTO ##tblAnreden(AnredeID, Anrede) VALUES(3,N'Firma');
```

Lesen Sie dann den Inhalt dieser temporären Tabelle in Access mittels einer Pass-Through-Abfrage mit folgender Anweisung aus:

```
SELECT * FROM ##tblAnreden;
```

Das Ergebnis ist der zuvor im SQL Server Management Studio angelegte Datensatz. Wenn Sie die gleichen Schritte für die temporäre lokale Tabelle durchführen, finden Sie die angelegten Datensätze nur in der jeweils gleichen Verbindung vor.

... DIE ÜBRIGEN SEITEN GIBT'S GEDRUCKT UNTER SHOP.MINHORST.COM



BERND JUNGBLUTH
ANDRÉ MINHORST

André Minhorst Verlag

10 Gespeicherte Prozeduren

Neben den Sichten können Sie auch mit Gespeicherten Prozeduren auf die Daten einer SQL Server-Datenbank zugreifen und sich das ermittelte Ergebnis ausgeben lassen. Dabei sind gespeicherte Prozeduren gegenüber Sichten wesentlich flexibler: Sie erlauben zum Beispiel den Einsatz von Parametern und bieten zudem die Möglichkeit, Daten hinzuzufügen, zu ändern und zu löschen. Dabei können gespeicherte Prozeduren eine oder mehrere SQL-Anweisungen oder auch weitere Befehle und sogar Strukturen wie Bedingungen oder Variablen enthalten.

Wie wir bereits weiter vorn im Buch erläutert haben, sind Access-Abfragen, die sich auf Tabellen einer SQL Server-Datenbank beziehen, meist sehr viel langsamer als solche Abfragen, die direkt im SQL Server ausgeführt werden. Deshalb verwenden Sie besser Sichten und gespeicherte Prozeduren statt Access-Abfragen.

Im Gegensatz zu Sichten, die Sie genau wie die Tabellen einer SQL Server-Datenbank mit einer Access-Datenbank verknüpfen können, ist zu gespeicherten Prozeduren keine Verknüpfung möglich – zumindest keine direkte. Wenn Sie eine gespeicherte Prozedur aufrufen und das Ergebnis in Access anzeigen oder weiterverwenden möchten, müssen Sie eine Pass-Through-Abfrage anlegen, welche den Aufruf der gespeicherten Prozedur und eventuell benötigte Parameter enthält.

10.1 Vorteile gespeicherter Prozeduren

Sie können über eingebundene Tabellen auf eine SQL Server-Datenbank zugreifen oder Sie schreiben die gewünschte SQL-Anweisung in eine Pass-Through-Abfrage und senden diese zur Ausführung an den SQL Server. Der beste Weg ist es jedoch, die SQL-Anweisungen in Form einer gespeicherten Prozedur in der SQL Server-Datenbank zu speichern und diese dann über eine Pass-Through-Abfrage aufzurufen. Warum die Abarbeitung direkt im SQL Server schneller erfolgt als bei verknüpften Tabellen, haben wir bereits im Kapitel »Performance analysieren«, Seite 105, besprochen. Warum aber soll man SQL-Anweisungen in gespeicherte Prozeduren schreiben und diese nicht ad-hoc aufrufen? Dies und weitere Vor- wie auch Nachteile von gespeicherten Prozeduren klären die folgenden Abschnitte.

10.1.1 Geschwindigkeit

Grundsätzlich ist es möglich, beim Einsatz einer Access-Anwendung mit SQL Server-Backend komplett auf gespeicherte Prozeduren zu verzichten. Sie könnten die SQL-Abfragen oder auch komplette Abfolgen von SQL-Anweisungen in einer Pass-Through-Abfrage speichern und diese dann zum SQL Server senden, damit dieser die Anweisungen ausführt. Das Ausführen einer gespeicherten Prozedur, die dieselbe Anweisung enthält, ist jedoch wesentlich schneller als die

Ad-hoc-Ausführung von SQL-Anweisungen über eine Pass-Through-Abfrage. Schauen wir uns an, warum dies so ist:

Die Ad-hoc-Ausführung von SQL-Anweisungen unterscheidet sich auf den ersten Blick nicht von der Ausführung einer gespeicherten Prozedur oder den anderen SQL Server-Objekten Sichten, Funktionen und Triggern. Bei allen erfolgt zunächst eine Syntaxprüfung des enthaltenen SQL und eine Existenzprüfung der dort verwendeten Tabellen, Sichten et cetera samt deren Spalten. Anschließend erstellt der Abfrageoptimierer den Ausführungsplan und speichert diesen im Prozedurcache. Zur Erinnerung: der Ausführungsplan legt fest, in welcher Reihenfolge etwa die Tabellen und Felder einer SQL-Anweisung gelesen werden und wie dabei die Indizes zu berücksichtigen sind. Im Anschluss daran folgt die Ad-hoc-Ausführung der SQL-Anweisung beziehungsweise des SQL Server-Objekts anhand des eben erstellten Ausführungsplans.

Ab der zweiten Ausführung entfällt das Erstellen des Ausführungsplans, denn der bereits im Prozedurcache gespeicherte wird einfach wiederverwendet. Da die Arbeit des Abfrageoptimierers in diesem ganzen Vorgang die zeitintensivste ist, wird durch die Wiederverwendung viel Zeit gespart und eine bessere Gesamtperformance erreicht. Die vom Abfrageoptimierer erstellten Ausführungspläne und deren Wiederverwendung haben wir im Kapitel »FAQ«, Seite 19, kurz bereits beschrieben.

Soweit die kurze Wiederholung zur Vorgehensweise der Ausführung einer SQL-Anweisung und/oder eines SQL Server-Objekts. Bis jetzt ist kein Unterschied zwischen einer Ad-hoc-SQL-Anweisung und einer gespeicherten Prozedur zu erkennen. Aber es gibt einen, der in der Performance einen großen Unterschied ausmacht.

Der Abfrageoptimierer erstellt für eine Ad-hoc-SQL-Anweisung einen Ausführungplan, der exakt auf diese SQL-Anweisung zutrifft. Führen Sie zum Beispiel die beiden folgenden SQL-Anweisungen einzeln aus, ergibt dies zwei Ausführungpläne:

```
SELECT Bestelldatum FROM dbo.tblBestellungen WHERE KundeID = 74;
SELECT Bestelldatum FROM dbo.tblBestellungen WHERE KundeID = 96;
```

Hier greift der Vorteil und Nutzen des Ausführungsplans nur, wenn beide Abfragen mehrmals aufgerufen werden. Erstellen Sie eine gespeicherte Prozedur, die Ihnen ebenfalls das Bestelldatum liefert, wobei jedoch die *KundeId* als Parameter übergeben wird, erzeugt der Abfrageoptimierer nur einen Ausführungsplan – für die gespeicherte Prozedur. Dieser wird bei jedem Aufruf wiederverwendet, unabhängig von der übergebenen ID des Kunden.

Bei einer gespeicherten Prozedur ist somit nicht nur die Wiederverwendbarkeit des zugehörigen Ausführungplans höher, der Prozedurcache enthält auch weitaus weniger Ausführungspläne.

10.1.2 Datenkonsistenz und Geschäftsregeln

Sie können Ihre Anwendung auch so konzipieren, dass Sie keine Tabellen einer SQL Server-Datenbank einbindet. An Stelle dessen verwenden Sie gespeicherte Prozeduren für die Datener-

mittlung, wie auch für das Hinzufügen, Ändern und Löschen von Daten. Dies ist ohne Weiteres möglich – *SELECT*-, *UPDATE*-, *INSERT*- und *DELETE*-Anweisungen lassen sich problemlos in gespeicherten Prozeduren abbilden, und dies sogar kombiniert. Nicht nur das: in gespeicherten Prozeduren können Sie richtig programmieren.

Es stehen Ihnen fast alle Möglichkeiten von T-SQL zur Verfügung, wie *IF...ELSE* und *WHILE* sowie die Verwendung von Variablen und Systemfunktionen (siehe Kapitel »T-SQL-Grundlagen«, Seite 221). Die Gewährleistung der Datenkonsistenz und die Einhaltung der Geschäftsregeln lassen sich also auch mit gespeicherten Prozeduren realisieren.

Was aber haben Sie nun davon? Gesetzt den Fall, dass Sie nicht nur mit einem Access-Frontend auf die Daten im SQL Server-Backend zugreifen, sondern auch noch über eine Webanwendung und vielleicht eine .NET-Desktop-Anwendung, müssen Sie die Geschäftsregeln in allen Frontends realisieren. Selbst die kleinste Änderung ist dann in allen Versionen der Benutzeroberfläche anzupassen. Definieren Sie die Geschäftregeln hingegen im SQL Server-Backend in gespeicherten Prozeduren, brauchen Sie die Änderung nur dort durchzuführen und nicht in jedem einzelnen Frontend.

Angenommen, Sie entscheiden sich, in einer Tabelle keine Datensätze mehr zu löschen, sondern stattdessen die Datensätze als inaktiv zu kennzeichnen. Für diese Änderung müssten Sie in jeder Benutzeroberfläche den Löschvorgang durch eine entsprechende *UPDATE*-Anweisung austauschen. Liegt Ihre Logik für den Löschvorgang in einer gespeicherten Prozedur, ersetzen Sie nur dort den *DELETE*-Befehl durch einen *UPDATE*-Befehl.

Die Änderung an einer Stelle bedeutet nicht nur weniger Aufwand, auch die Fehleranfälligkeit ist geringer. Es gibt noch einen weiteren und nicht zu unterschätzenden Vorteil: Nach dem Speichern der gespeicherten Prozedur steht die Änderung direkt für alle Benutzeroberflächen zur Verfügung. Ein erneutes Verteilen der Access-Datenbanken, .NET-Applikationen, oder was auch immer die Benutzerobfläche anbietet, entfällt.

Jetzt werden Sie möglicherweise einwerfen, dass Sie ja ausschließlich mit einem Access-Frontend arbeiten und dies auch in absehbarer Zeit nicht geändert werden soll. Dann erhalten Sie durch die Verlagerung der Geschäftsregeln in gespeicherte Prozeduren immer noch einen Vorteil: Sobald Sie nämlich selbst innerhalb eines einzigen Access-Frontends von mehreren Stellen aus etwa die Daten einer Tabelle ändern, müssen Sie an all diesen Stellen die Geschäftslogik implementieren. Liegt diese hingegen in einer gespeicherten Prozedur, brauchen Sie sich bei der Programmierung des Frontends keine Sorgen darüber zu machen.

Und damit der Benutzer nicht doch irgendeinen Weg findet, eine Tabelle per ODBC einzubinden oder per VBA darauf zuzugreifen, entziehen Sie dem Benutzer einfach die kompletten Rechte an dieser Tabelle!

Wenn ihm als einzige Möglichkeit zum Ändern eines Datensatzes eine entsprechende gespeicherte Prozedur vorliegt, werden die dazu definierten Geschäftsregeln in jedem Fall durchgesetzt. Diese Vorgehensweise lässt sich natürlich auch beim Hinzufügen und Löschen von Datensätzen

und sogar bei der Datenermittlung und Datenausgabe nutzen. Jegliche Aktion mit den Daten erfolgt über gespeicherte Prozeduren und somit über die dort definierte Geschäftslogik.

10.2 Nachteile von gespeicherten Prozeduren

Wenn denn überhaupt von Nachteilen die Rede sein kann, sind lediglich zwei Punkte zu erwähnen. Und beide betreffen die Verwendung von gespeicherten Prozeduren. Gespeicherte Prozeduren werden mit dem Befehl *EXECUTE* ausgeführt. *EXECUTE* wiederum lässt sich in nicht einer *SELECT*-Anweisung nutzen. Aus diesem Grund ist es nicht möglich, eine gespeicherte Prozedur in einer *SELECT*-Anweisung ähnlich einer Tabelle, Sicht oder Funktion anzusprechen.

Der zweite Punkt bezieht sich auf die Verwendung von gespeicherten Prozeduren in Access. Die von einer gespeicherten Prozedur gelieferten Daten können in Access nicht wie bei einer eingebundenen Tabelle direkt geändert werden. Dies liegt jedoch nicht an der gespeicherten Prozedur, sondern vielmehr an der Pass-Through-Abfrage.

Daten, die Access über eine Pass-Through-Abfrage ermittelt, können Sie schlicht und einfach nicht ändern. Dabei ist es egal, ob Sie in der Pass-Through-Abfrage eine gespeicherte Prozedur oder eine SQL-Anweisung ausführen.

10.3 Gespeicherte Prozeduren erstellen

Gespeicherte Prozeduren verwenden die Programmiersprache *T-SQL*. Diese ist selbst verglichen mit VBA relativ übersichtlich. Für die Erstellung gespeicherter Prozeduren liefert diese Programmiersprache die folgenden interessanten Features:

- Definition von Ein- und Ausgabeparametern mit oder ohne Standardwerte
- Deklaration und Verwendung von Variablen
- Abfrage von Systemwerten wie Anzahl geänderter Datensätze
- Verwendung einfacher Strukturen wie *IF...ELSE*
- Programmierung einfacher Schleifen
- Speichern von Zwischenergebnissen in temporären Tabellen oder *TABLE*-Variablen
- Verschachteln von gespeicherten Prozeduren
- Einsatz von Sichten und benutzerdefinierten Funktionen innerhalb gespeicherter Prozeduren
- Implementieren einer Fehlerbehandlung

Eine kleine Einführung in T-SQL finden Sie in Kapitel »T-SQL-Grundlagen«, Seite 221.

10.3.1 Anlegen einer gespeicherten Prozedur mit Vorlage

Das SQL Server Management Studio bietet für den Beginn eine recht gute Hilfe zum Anlegen einer gespeicherten Prozedur.

Dazu wählen Sie im Objekt-Explorer zunächst die entsprechende Datenbank aus (etwa die Beispieldatenbank *AEMA_SQL*) und dort im Kontextmenü des Elements *Programmierbarkeit|Gespeicherte Prozeduren* den Eintrag *Neue gespeicherte Prozedur ...* (siehe Abbildung 10.1).

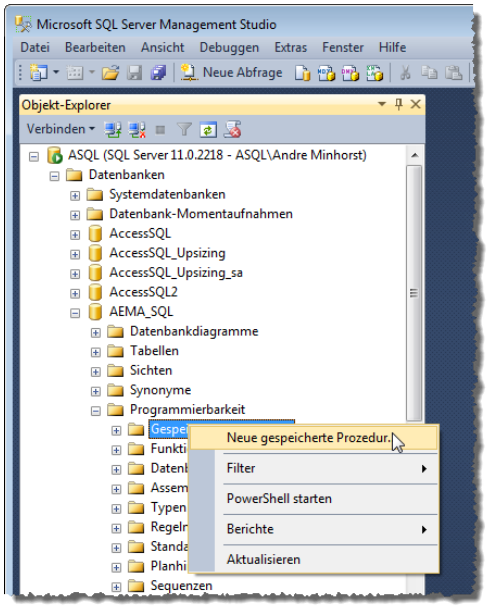


Abbildung 10.1: Anlegen des Grundgerüsts einer gespeicherten Prozedur

Dies öffnet ein neues Abfragefenster und bildet dort das Grundgerüst für eine neue gespeicherte Prozedur ab (siehe Abbildung 10.2). Der Code der Vorlage sieht auf den ersten Blick etwas verwirrend aus, aber wenn Sie erstmal die Kommentare entfernen, ist es nur noch halb so wild.

Hier wird direkt ein wesentlicher Unterschied zum VBA-Editor deutlich: Beim SQL Server legen Sie eine Prozedur nicht einfach in einem Modul an, sondern Sie verwenden dazu einen T-SQL-Befehl, in diesem Fall *CREATE PROCEDURE*.

Es gibt auch nirgends eine sichtbare Fassung der gespeicherten Prozedur – Sie können lediglich eine Anweisung generieren lassen, die den Code zum Erstellen oder Ändern der gespeicherten Prozedur bereitstellt. Der wiederum sieht fast genauso aus wie der, mit dem Sie die gespeicherte Prozedur einst erstellt haben.

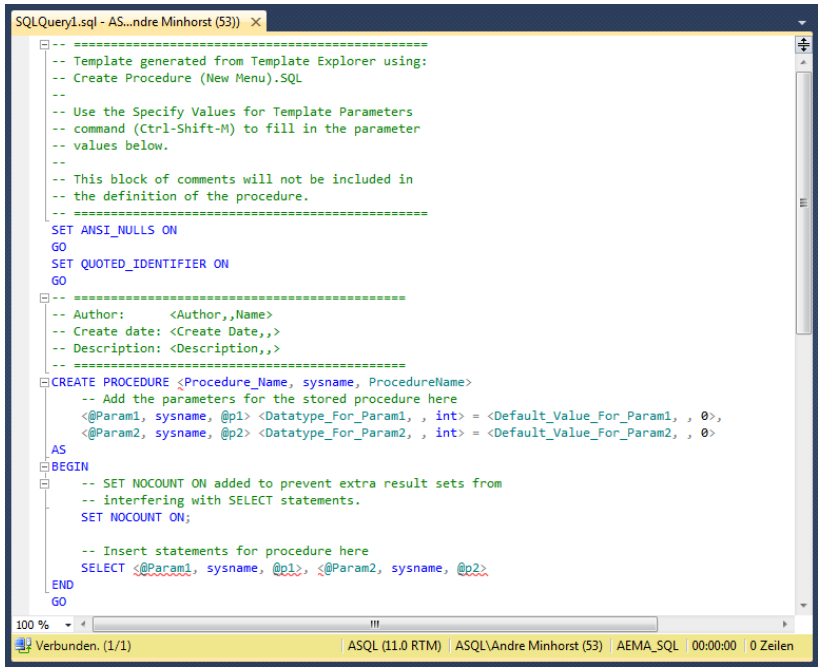


Abbildung 10.2: Vorlage für eine gespeicherte Prozedur

10.3.2 Neue gespeicherte Prozedur

Trotz dieser ungewohnten Vorgehensweise wollen wir uns ans Werk machen. Leeren wir das Abfragefenster und beginnen komplett von vorn. Der erste Teil der Anweisung zum Erstellen einer gespeicherten Prozedur lautet immer wie folgt und wird meist in der ersten Zeile abgebildet:

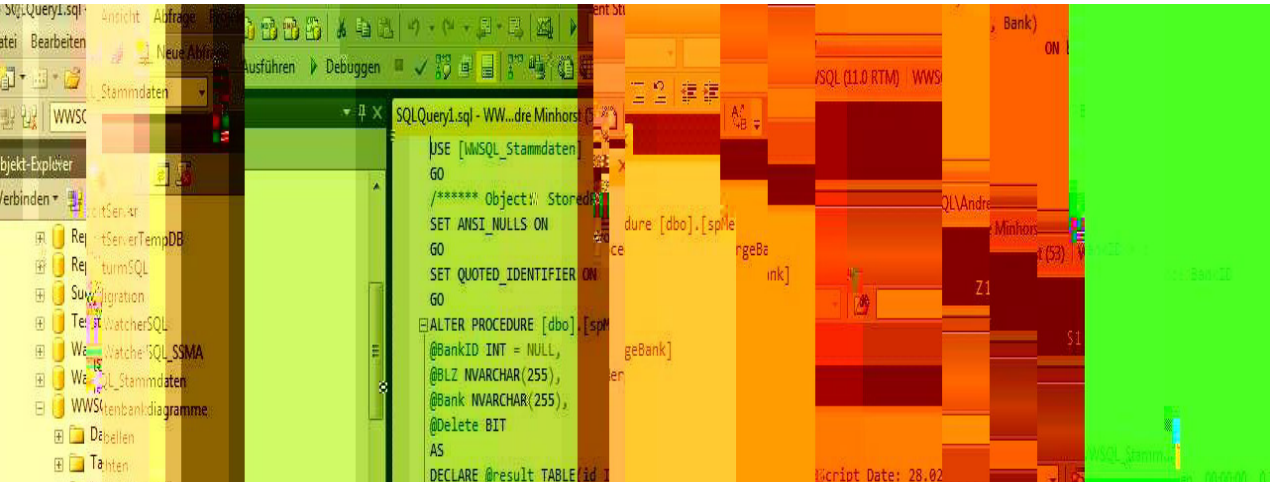
```
CREATE PROCEDURE <Schema>.<Prozedurname>
```

Es ist üblich, die Anweisung zum Erstellen einer gespeicherten Prozedur auf mehrere Zeilen aufzuteilen. Dies dient lediglich der Übersicht – Sie könnten die Anweisung auch in einer Zeile erfassen.

Die erste Zeile legt das Schema und den Namen der gespeicherten Prozedur fest. Der Name beginnt meist mit *sp* und darf nur alphanumerische Zeichen und den Unterstrich enthalten. Sie können natürlich auch ein anderes Präfix als *sp* verwenden, aber nicht *sp_*. Dies ist das Präfix der gespeicherten Prozeduren des Systems.

Das Problem mit diesem Präfix ist, dass der SQL Server beim Aufruf von gespeicherten Prozeduren mit dem Präfix *sp_* diese zunächst in der *master*-Datenbank und anschließend in den Systemobjekten der aktuellen Datenbank sucht, bevor er sie letztendlich bei den benutzerdefinierten Prozeduren findet. Dieser Suchvorgang kostet nur unnötig Zeit.

... DIE ÜBRIGEN SEITEN GIBT'S GEDRUCKT UNTER SHOP.MINHORST.COM



BERND JUNGBLUTH
ANDRÉ MINHORST

André Minhorst Verlag

11 Funktionen

Bestimmt enthält Ihre Access-Applikation einige VBA-Funktionen. Funktionen, in denen Sie wiederkehrende Funktionalität, wie Berechnungen, Datenermittlungen und Programmlogiken, ausgelagert haben, um diese dann an mehreren Stellen im VBA-Code oder in Access-Abfragen wiederzuverwenden. Auch SQL Server bietet Ihnen die Möglichkeit dieser modularen Entwicklungsmethode.

Mit den Funktionen im SQL Server lassen sich ebenfalls wiederkehrende Berechnungen, Programmlogiken und Datenermittlungen kapseln und in gespeicherten Prozeduren, Sichten und Triggern wiederverwenden. Funktionen werden wie die anderen SQL Server-Objekte in T-SQL programmiert. Sie unterstützen Parameter und liefern je nach Typ einen Wert oder eine Ergebnismenge. Insgesamt gibt es drei verschiedene Typen, die Sie in diesem Kapitel kennen lernen.

In Access lassen sich die Funktionen vom SQL Server nicht direkt verwenden. Zwar liefern zwei der drei Funktionstypen Ergebnismengen, dennoch können Sie diese Funktionen nicht wie Tabellen oder Sichten in Access einbinden. Es bleibt Ihnen eigentlich nur der Weg über eine Pass-Through-Abfrage; dieser ist aber nicht empfehlenswert.

Funktionen lassen sich nur in Verbindung mit SQL-Anweisungen nutzen – und wie Sie im Kapitel »Gespeicherte Prozeduren«, Seite 257, gelernt haben, sollten Sie aus Gründen der Performance in Pass-Through-Abfragen keine SQL-Anweisungen verwenden, sondern gespeicherte Prozeduren. Folglich nutzen Sie die Funktionen lediglich in Ihren gespeicherten Prozeduren, Sichten und Triggern und somit nur indirekt in Access.

11.1 Vorteile von Funktionen

Ein großer Vorteil der Funktionen ist die Möglichkeit der modularen Entwicklung. Wiederkehrende Berechnungen zum Beispiel müssen Sie nicht in jeder SQL-Anweisung angeben, in der Sie diese benötigen, sondern Sie definieren diese nur ein einziges Mal in einer Funktion. Diese Funktion verwenden Sie dann an den jeweiligen Stellen, an denen Sie sonst die Berechnung angegeben hätten. Ein gutes Beispiel für eine wiederkehrende Berechnung ist die Positionssumme einer Bestellung, die sich aus Einzelpreis, Menge und Rabatt ergibt. Oder Sie erstellen eine Funktion, die Ihnen aus dem übergebenen Nachnamen und Vornamen eine Zeichenfolge mit Vor- und Nachnamen erstellt. Eine solche Funktion lässt sich dann gleich für mehrere Zwecke verwenden, beispielsweise für die Namen der Mitarbeiter und für die der Ansprechpartner von Lieferanten und Kunden. Diese Art der Funktionen sind typische Beispiele für eine Skalarfunktion.

Nicht nur Berechnungen, sondern auch Datenermittlungen können von Funktionen übernommen werden. Die Tabellenwertfunktionen liefern Ergebnismengen und lassen sich dabei wie Sichten und Tabellen ansprechen. Benötigen Sie zum Beispiel eine Auflistung mit Kunden, die

einen Newsletter beziehen möchten und eine weitere mit den Kunden, die den Newsletter nicht erhalten wollen, können Sie beide Ausgaben mit einer Tabellenwertfunktion realisieren. Diese liefert abhängig vom übergebenen Parameter die Kunden mit Newsletteranmeldung oder die Kunden ohne Newsletteranmeldung.

Die weiteren Vorteile von Funktionen sind schnell beschrieben: Funktionen bieten die gleichen Vorteile wie Gespeicherte Prozeduren. Sie können mit Funktionen die Geschäftslogik im SQL Server abbilden und erhalten dabei eine bessere Performance, denn auch für die Funktionen werden Ausführungspläne erstellt, im Prozedurcache gespeichert und bei jedem Einsatz der Funktion wiederverwendet. Mehr zu den Ausführungplänen lesen Sie im Kapitel »FAQ«, Seite 19.

11.2 Skalarfunktion

Die Skalarfunktion liefert – nomen es omen – einen einzelnen Wert. Bei diesem Wert kann es sich um eine Konstante handeln, einem Ergebnis einer einfachen *SELECT*-Anweisung, einer Berechnung anhand der übergebenen Parameter oder auch um das Ergebnis einer komplexen Logik, realisiert mit mehreren SQL-Anweisungen, Variablen und Programmstrukturen wie *IF... ELSE* und *WHILE*.

Nicht nur der Inhalt des Werts, auch dessen Datentyp wird über die Skalarfunktion definiert. Dabei können Sie fast alle Datentypen nutzen, die Ihnen auch bei der Definition einer Tabelle zur Verfügung stehen. Lediglich die veralteten Datentypen für binäre Werte (*Image*, *nText* und *Text*) wie auch *Table*-Variablen werden nicht unterstützt. Skalarfunktionen sind vielseitig einsetzbar:

- Als Spalten in *SELECT*-Anweisungen
- Als Filterkriterien in *WHERE*-Bedingungen
- Als Werte in *INSERT*- und *UPDATE*-Anweisungen
- Als Standardwerte für Spalten von Tabellen
- Zur Prüfung von Einschränkungen von Spalten und Tabellen
- Zur Initialisierung von Variablen
- Zur Steuerung des Programmflusses in *IF*- und *WHILE*-Anweisungen

11.2.1 Skalarfunktion anlegen

Zum Erstellen einer Skalarfunktion können Sie wie bei den Gespeicherten Prozeduren eine Vorlage verwenden. Sie erhalten diese Vorlage, indem Sie im Objekt-Explorer den Kontextmenüeintrag *Neue Skalarfunktion* des Elements *Programmierbarkeit|Funktionen|Skalarfunktionen* wählen. Leider ist diese Vorlage wie bei den Gespeicherten Prozeduren eher verwirrend als hilfreich.

Am besten legen Sie eine Skalarfunktion in einem neuen Abfragefenster an. Dazu markieren Sie im Objekt-Explorer die Datenbank, in der die Skalarfunktion gespeichert werden soll, und wählen dort im Kontextmenü den Eintrag *Neue Abfrage* aus.

Im neuen Abfragefenster geben Sie dann den entsprechenden *CREATE FUNCTION*-Befehl an, ergänzt mit dem Schemanamen und dem Namen der Skalarfunktion. Den Typ der Skalarfunktion müssen Sie dabei nicht angeben. Dieser ergibt sich aus der weiteren Syntax.

Im folgenden Beispiel möchten wir eine Skalarfunktion erstellen, die das aktuelle Tagesdatum ausgibt. Dazu wird der Rückgabewert der Systemfunktion *GETDATE()*, die den aktuellen Zeitpunkt liefert, in den Datentyp *Date* konvertiert. Die Skalarfunktion nennen wir *sfAktuellesDatum*. Die erste Zeile unserer neuen Skalarfunktion lautet somit wie folgt:

```
CREATE FUNCTION dbo.sfAktuellesDatum
```

Um beim späteren Programmieren die Skalarfunktion als solche zu erkennen, empfiehlt sich bei der Namensvergabe die Verwendung eines entsprechenden Präfix – zum Beispiel *sf*.

Als nächstes ist die Definition der Eingabeparameter an der Reihe. Die Angabe der Parameter erfolgt immer in runden Klammern. Dies gilt auch für Skalarfunktionen ohne Parameter. Ob Sie also in Ihrer Skalarfunktion Parameter verwenden oder nicht, die runden Klammern sind Pflicht. In unserem Beispiel verwenden wir keine Parameter.

Nach den Klammern folgt die Anweisung *RETURNS*, mit der Sie den Datentyp des Skalarwerts – des Rückgabewerts der Skalarfunktion – festlegen. Das Datum des aktuellen Zeitpunkts liefern wir im Datentyp *Date*:

```
RETURNS date
```

Nun kommt noch das Schlüsselwort *AS*, das Sie bereits von den Gespeicherten Prozeduren kennen, und anschließend die Programmlogik zur Ermittlung des Skalarwerts innerhalb eines *BEGIN...END*-Blocks. In unserem Beispiel besteht die Logik lediglich aus der Konvertierung des Rückgabewerts der Systemfunktion *GETDATE()* in den Datentyp *Date*.

Das Ergebnis einer Skalarfunktion wird unabhängig vom Datentyp über *RETURN* ausgegeben. Da in unserem Beispiel die Programmlogik lediglich aus der Rückgabe eines konvertierten Werts besteht, verbinden wir die Konvertierung direkt mit der *RETURN*-Anweisung:

```
CREATE FUNCTION dbo.sfAktuellesDatum ()
RETURNS date
AS
BEGIN
       RETURN CAST(GETDATE() As Date);
END
```

Diese recht überschaubare Skalarfunktion legen Sie nun mit der Taste *F5* an. Anschließend ist sie im Element *Programmierbarkeit|Funktionen|Skalarfunktionen* zu sehen. Sollte dies nicht der

Fall sein, aktualisieren Sie die Ansicht über den Kontextmenüeintrag *Aktualisieren* des Elements *Skalarfunktionen*.

Die einfachste Verwendung dieser Skalarfunktion ist ein simpler Aufruf per *SELECT*:

```
SELECT dbo.sfAktuellesDatum() As Tagesdatum;
```

Beachten Sie, dass Sie beim Aufruf einer Skalarfunktion immer das Schema und die Klammern der Parameter angeben. Beide sind zwingend erforderlich, auch wenn Sie nur das Standardschema *dbo* verwenden beziehungsweise keine Parameter definiert sind. Führen Sie nun die Skalarfunktion unter Berücksichtigung dieser beiden Syntaxvorschriften aus, erhalten Sie als Ergebnis das Datum von heute im ISO-Format (siehe Abbildung 11.1).

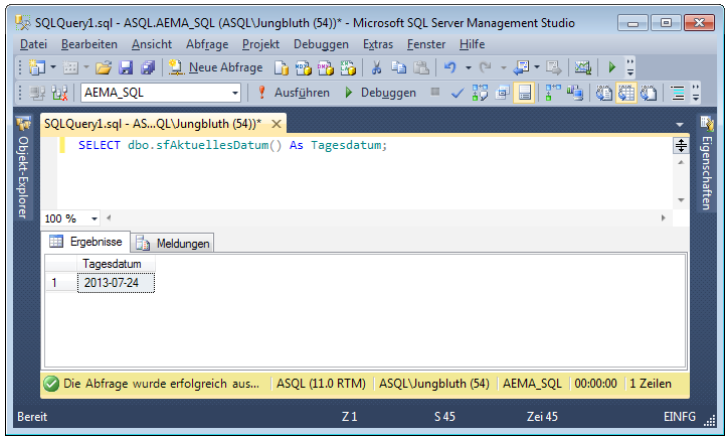


Abbildung 11.1: Eine Skalarfunktion im Einsatz

Die Skalarfunktion können Sie ebenso in einer *WHERE*-Bedingung verwenden. Folgende Abfrage liefert Ihnen die Bestellungen von heute:

```
SELECT * FROM dbo.tblBestellungen WHERE Bestelldatum = dbo.sfAktuellesDatum();
```

11.2.2 Skalarfunktion mit Parametern

Die Parameter einer Skalarfunktion werden direkt nach dem Funktionsnamen in den bereits erwähnten runden Klammern angegeben. Dabei wird jeder Parameter mit einem Datentyp deklariert, sein Name beginnt immer mit einem @-Zeichen und darf keine Sonderzeichen mit Ausnahme des Unterstrichs enthalten und mehrere Parameter sind mit einem Komma voneinander zu trennen.

Im nächsten Beispiel soll eine Skalarfunktion die Summe einer Bestellposition ermitteln. Hierzu erhält die Skalarfunktion über Parameter die Werte *Einzelpreis*, *Menge* und *Rabatt*, berechnet die Positionssumme und gibt das Ergebnis im Datentyp *Money* aus:

```
CREATE FUNCTION dbo.sfPositionssumme (
@Einzelpreis money,
@Menge int,
@Rabatt money)
RETURNS money
AS
BEGIN
     RETURN (@Einzelpreis * @Menge) - @Rabatt;
END
```

Diese Skalarfunktion liefert Ihnen nun zu jeder Bestellposition die Positionssumme, wie in Abbildung 10.2 zu sehen ist.

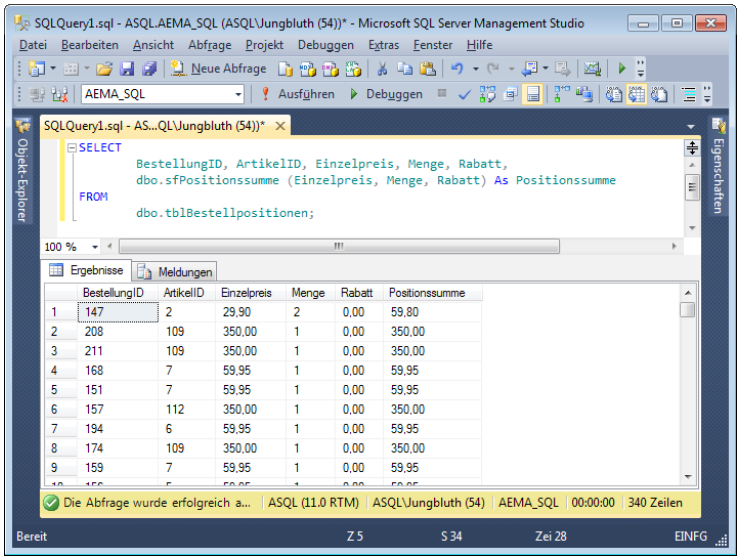


Abbildung 11.2: Die Berechnung der Positionssumme durch eine Skalarfunktion

11.2.3 Skalarfunktion mit mehreren Anweisungen

Die bisher gezeigten Beispiele ermitteln den Skalarwert anhand einer einzigen Anweisung. Nun kann eine Skalarfunktion, wie oben bereits beschrieben, auch mehr als eine Anweisung enthalten. Auch diesen Fall möchten wir an einem Beispiel zeigen. Dieses Mal ermittelt die Skalarfunktion den Namen der Warengruppe zu einer übergebenen *WarengruppeID*:

```
CREATE FUNCTION dbo.sfWarengruppe (
@WarengruppeID int)
RETURNS nvarchar(255)
AS
BEGIN
    DECLARE @Warengruppe nvarchar(255);
```

```
    SELECT @Warengruppe = Warengruppe FROM dbo.tblWarengruppen
    WHERE WarengruppeID = @WarengruppeID;
    RETURN @Warengruppe;
END
```

Die Skalarfunktion *sfWarengruppe* empfängt über den Parameter *@WarengruppeID* einen Wert vom Datentyp *Integer* und liefert nach der Verarbeitung den Skalarwert im Datentyp *nVarChar(255)*.

Die im *BEGIN...END*-Block enthaltene Programmlogik startet mit der Definition der Variablen *@Warengruppe*. Da sowohl die Ermittlung wie auch die Ausgabe des Skalarwerts über diese Variable erfolgt, erhält sie den gleichen Datentyp wie die Skalarfunktion selbst – in diesem Fall *nVarChar(255)*. Dann wird mit der im Parameter *@WarengruppeID* enthaltenen ID der entsprechende Datensatz in der Tabelle *tblWarengruppen* ermittelt und dabei der Wert der Spalte *Warengruppe* der Variablen *@Warengruppe* zugewiesen. Abschließend folgt die Ausgabe über die *RETURN*-Anweisung mit dem Wert der Variablen.

Verwenden Sie diese Skalarfunktion in der folgenden Abfrage, erhalten Sie neben den Informationen zum Artikel auch die Bezeichnung der Warengruppe.

```
SELECT ArtikelID, Artikelname, dbo.sfWarengruppe(WarengruppeID) As Warengruppe
FROM dbo.tblArtikel;
```

11.2.4 Skalarfunktion ändern

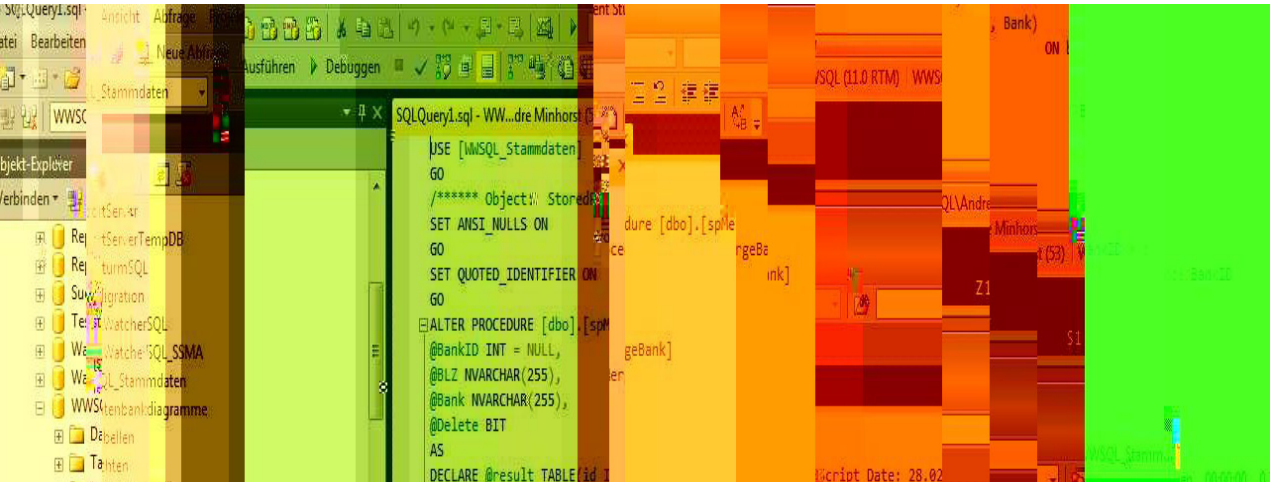
Den Quellcode einer Skalarfunktion ändern Sie in derselben Art und Weise, wie Sie ihn erstellt haben – mit einem T-SQL-Skript in einem Abfragefenster. Sie erhalten den Quellcode über den Eintrag *Ändern* aus dem Kontextmenü der Skalarfunktion.

Das Abfragefenster zeigt fast den gleichen Code, den Sie beim Erstellen der Funktion eingegeben haben. Da Sie aber über diesen Weg keine Funktion erstellen, sondern eine bestehende ändern möchten, enthält der Quellcode nun anstelle der Anweisung *CREATE FUNCTION* die Anweisung *ALTER FUNCTION*. Nachdem Sie den Quellcode der Skalarfunktion an die neuen Anforderungen angepasst haben, führen Sie das T-SQL-Skript mit der Taste *F5* aus, worauf die Skalarfunktion neu erstellt und gespeichert wird. Sie kennen diese Vorgehensweise bereits von den Gespeicherten Prozeduren.

11.3 Tabellenwertfunktionen

Im Gegensatz zu Skalarfunktionen liefern Tabellenwertfunktionen keinen einzelnen Wert, sondern einen oder mehrere Datensätze. SQL Server stellt Ihnen zwei Typen von Tabellenwertfunktionen zur Verfügung. Beide sind mit einer Sicht vergleichbar: sie liefern auf Basis einer *SELECT*-Anweisung Daten einer oder mehrerer Tabellen, werden in SQL-Anweisungen wie eine Tabelle oder Sicht angesprochen und lassen sich dabei auch mit anderen Tabellen verknüpfen.

... DIE ÜBRIGEN SEITEN GIBT'S GEDRUCKT UNTER SHOP.MINHORST.COM



BERND JUNGBLUTH
ANDRÉ MINHORST

André Minhorst Verlag

12 Trigger

Trigger lassen sich mit gespeicherten Prozeduren vergleichen. Sie werden auf ähnliche Weise angelegt und geändert und können eine oder mehrere SQL-Anweisungen kombiniert mit T-SQL-Befehlen und Strukturen wie Bedingungen oder Variablen enthalten. Ebenso wie bei den gespeicherten Prozeduren erstellt der SQL Server bei der ersten Ausführung eines Triggers einen Ausführungsplan und speichert diesen im Prozedurcache, sodass bei nachfolgenden Aufrufen der gespeicherte Ausführungsplan wiederverwendet wird und die Ausführung des Triggers einen Performancevorteil erhält.

Damit enden die Gemeinsamkeiten aber auch schon. Denn im Gegensatz zu gespeicherten Prozeduren sind Trigger keine eigenständigen Objekte und lassen sich dementsprechend auch nicht manuell aufrufen. Ein Trigger ist vielmehr Bestandteil der Definition einer Tabelle und legt fest, welche Aktionen bei den verschiedenen Datenoperationen ausgelöst werden.

12.1 Funktionsweise von Triggern

Wer bislang nur mit Access programmiert hat, erhält ein besseres Verständnis von der Funktionsweise von Triggern, wenn er diese mit den Ereignisprozeduren von Formularen oder Steuerelementen vergleicht. In Access können Sie Ereignisprozeduren hinterlegen, die durch verschiedene Ereignisse eines Formulars oder Steuerelements ausgelöst werden – etwa beim Laden oder Schließen des Formulars, beim Anzeigen oder Löschen eines neuen Datensatzes oder beim Betätigen einer Schaltfläche. Ähnlich ist es mit den Tabellen einer SQL Server-Datenbank. Auch hier lassen sich Prozeduren definieren, die durch verschiedene Ereignisse ausgelöst werden.

Welche Ereignisse können dies sein? Die folgende Übersicht fasst die möglichen Trigger zusammen:

» *Aktualisieren eines Datensatzes:* Wird ein Datensatz einer Tabelle geändert, löst dies den sogenannten Update-Trigger aus.

» *Löschen eines Datensatzes:* Wird ein Datensatz einer Tabelle gelöscht, löst dies den Delete-Trigger aus.

» *Anlegen eines Datensatzes:* Auch beim Anlegen eines Datensatzes wird ein Trigger ausgelöst, in diesem Fall der Insert-Trigger.

Dabei ist wichtig, dass es keine Rolle spielt, auf welchem Weg ein Datensatz der Tabelle hinzugefügt, geändert oder gelöscht wird – ob im Frontend, mittels einer gespeicherten Prozedur oder durch die direkte Eingabe einer entsprechenden Anweisung im SQL Server Management Studio. Existiert an der Tabelle für diese Aktionen ein Trigger, wird dieser auch ausgelöst.

12.2 Pro und Contra

Trigger sind wegen ihrer Unbestechlichkeit nahezu perfekt zur Gewährleistung der Datenkonsistenz und zur Einhaltung von Geschäftsregeln. Nachfolgend finden Sie einige praktische Einsatzzwecke:

» *Durchsetzen referenzieller Integrität:* Referenzielle Integrität inklusive Lösch- und Aktualisierungsweitergabe sind fester Bestandteil des Funktionsumfangs vom SQL Server. Dennoch kann es sein, dass Sie zusätzliche Anforderungen an die Prüfung referenzieller Integrität oder an die Lösch- und Aktualisierungsweitergabe haben. Diese lassen sich mit Triggern realisieren.

» *Definition von Standardwerten:* SQL Server bietet die Möglichkeit, Standardwerte für Felder auf Basis von Konstanten sowie einfacher T-SQL-Funktionen und eigener Skalarfunktionen zu definieren. Alternativ wäre dies auch mit Triggern möglich.

» *Definition von Einschränkungen:* Um eine Datenmanipulation zu verhindern, die gegen definierte Regeln verstößt, bieten Tabellen die *Check Constraints* (zu deutsch Einschränkungen). Ähnlich der Standardwerte können Sie zur Prüfung der Eingaben Konstanten sowie einfache T-SQL-Funktionen und eigene Skalarfunktionen nutzen. Die Prüfung der Eingaben kann aber auch von Triggern übernommen werden.

» *Protokollieren von Datenänderungen:* Jede Änderung an den Daten einer Tabelle lassen sich über Trigger protokollieren. Auf diese Weise wäre nachvollziehbar, wer wann welchen Datensatz hinzugefügt, geändert oder gelöscht hat.

» *Redundante Datenhaltung:* Redundante Daten werden gerne zur Performance-Steigerung verwendet. Um zum Beispiel die Rechnungssumme nicht für jede Auswertung erneut aus den Summen der Rechnungspositionen ermitteln zu müssen, könnte diese ebenso gut im Rechnungskopf gespeichert werden. Es muss jedoch gewährleistet sein, dass bei jeder Änderung der Positionen die Summe im Rechnungskopf angepasst wird. Wenn schon Redundanz, dann sollte diese auch korrekt und konsistent sein. Diese Konsistenz könnte ein Trigger sicherstellen. Egal ob eine neue Position hinzugefügt, eine bestehende geändert oder gelöscht wird, der Trigger übernimmt die Übertragung der neuen Summe in den Rechnungskopf.

Dies sind nur ein paar Beispiele für die Verwendung von Triggern. Beispiele, die auch gleichzeitig gute Argumente für den Einsatz von Triggern liefern. Nun gibt es aber auch durchaus einige Argumente gegen die Verwendung von Triggern.

Nicht umsonst wird oft über das Für und Wider von Triggern diskutiert. Wir möchten Ihnen die Entscheidung überlassen und führen nun auch die Gegenargumente auf:

» *Keine Transparenz:* Die Trigger einer Datenbank und deren Auswirkungen sind nicht einfach zu überblicken. Trigger sind fest mit einer Tabelle verbunden und deshalb auch nur dort zu

finden. Um die Trigger einer Datenbank zu sehen, müssen Sie sich also mühsam durch alle Tabellen klicken oder aber Sie fragen die entsprechenden Systemtabellen der Datenbank ab und erstellen sich so eine Aufstellung aller Trigger. Aber auch mit einer Übersicht aller Trigger bleiben deren Auswirkungen weiterhin im Verborgenen. Schließlich könnte das Hinzufügen eines Datensatzes in Tabelle A durch einen Trigger eine Datenänderung an Tabelle B auslösen, die wiederum einen Trigger auslöst, der in Tabelle C und in Tabelle D einen Wert ändert. Es wäre sogar möglich, dass eine dieser Änderungen einen anderen Datensatz in Tabelle A löscht, worauf dort erneut ein Trigger aktiv wird. Solche rekursiven Aufrufe sind bis zu einer Tiefe von 32 Ebenen möglich, zum Glück nicht ohne vorherige Aktivierung.

» *Schlechte Performance:* Abhängig von der Programmierung eines Triggers führt dieser seine Verarbeitung zu jedem Datensatz aus, der gerade hinzugefügt, geändert oder gelöscht wird. Je aufwendiger die Verarbeitung des Triggers ist, umso schlechter wird die Performance der Datenverarbeitung. Dies macht sich gerade dann bemerkbar, wenn Sie mit einer Anweisung gleich mehrere Datensätze verarbeiten.

» *Komplexe Programmierung:* Zwar verwenden Sie zur Programmierung von Triggern ebenfalls T-SQL, jedoch ist die Vorgehensweise und Logik hierbei etwas anders. Mehr dazu später im Abschnitt »Trigger erstellen«, Seite 292.

» *Für das Frontend unsichtbar:* Eine Datenänderung über eine SQL-Anweisung im VBA oder direkt in einer eingebundenen Tabelle weist Sie in keiner Weise auf einen möglichen Trigger hin. Dies bedeutet, dass Sie im Quellcode des Frontends von den Aktionen eines Triggers nichts sehen. Bilden Sie hingegen die Logik des Triggers mit gespeicherten Prozeduren ab, sehen Sie im Quellcode den Aufruf der jeweiligen gespeicherten Prozeduren.

» *Gespeicherte Prozeduren:* Jegliche Aktion, die ein Trigger vor oder nach einer Datenänderung ausführen soll, können Sie auch in einer gespeicherten Prozedur realisieren. Eingabeprüfungen zum Beispiel führen Sie in einer gespeicherten Prozedur in entsprechenden *IF*-Anweisungen aus, worauf in Abhängigkeit vom Ergebnis die Aktion ausgeführt wird oder eben nicht. Oder soll beispielsweise beim Ändern einer Rechnungsposition die Rechnungssumme im Kopf angepasst werden, sind der *INSERT*- und *UPDATE*-Befehl lediglich in einer Transaktion auszuführen. Gespeicherte Prozeduren bieten Triggern gegenüber einen entscheidenden Vorteil: Anstelle der verborgenen Aktionen eines Triggers, die zudem noch eine wilde Verkettung weiterer Trigger zur Folge haben kann, sehen Sie in einer gespeicherten Prozedur alle notwendigen SQL-Anweisungen – lesbar und einfach nachvollziehbar.

Natürlich kommt es bei der Entscheidung für den Einsatz von Triggern nicht nur auf die oben genannten Argumente an. Viel entscheidender sind die Umstände des Projekts. In einer bestehenden und bereits installierten Client-/Server-Applikation werden Sie sich nicht unbedingt für die Variante mit gespeicherten Prozeduren entscheiden. Dies würde entscheidende Änderungen am Frontend nach sich ziehen, müssten Sie dort doch die jeweiligen Aufrufe der gespeicherten Prozeduren programmieren. In einem solchen Fall lässt sich die Logik schneller mit Triggern abbilden, da Sie hierbei das Frontend nicht zu ändern brauchen. Nur als Tipp: Dokumentieren

Sie Ihre Trigger und deren Auswirkungen auf andere Tabellen und deren Trigger von Anfang an ausführlich. Das erspart Ihnen später Zeit bei einer Fehlersuche. Sind Sie jedoch noch am Anfang des Projekts, sollten Sie den gespeicherten Prozeduren den Vorzug geben und auf Trigger verzichten – und genau dies ist der Fall, wenn Sie nach der Migration einer Access-Datenbank zum SQL Server auch die Logik von Access nach SQL Server verlagern. Technisch spricht nichts dagegen, Trigger und gespeicherte Prozeduren gemeinsam zu verwenden. Sie sollten jedoch gerade bei dieser Konstellation den Tipp zur ausführlichen Dokumentation berücksichtigen.

12.3 Zwei Arten von Triggern

SQL Server bietet Ihnen zwei verschiedene Trigger an. Beide reagieren auf die Aktionen *INSERT*, *UPDATE* und *DELETE*, jedoch zu unterschiedlichen Zeitpunkten:

» *INSTEAD OF-Trigger*: wird vor der Datenmanipulation ausgelöst und ersetzt die eigentliche Aktion.

» *AFTER-Trigger*: wird nach der Datenmanipulation ausgelöst.

Um zu verstehen, wann die beiden Trigger ausgeführt werden, schauen wir uns einmal den gesamten Prozess einer Datenmanipulation am Beispiel einer *INSERT*-Anweisung an:

» Erstellen der virtuellen Tabellen *inserted* und *deleted*: Mehr dazu im Anschluss.

» Auslösen des *INSTEAD OF*-Triggers: Die Logik des Triggers wird verarbeitet. Beinhaltet diese beispielsweise eine Prüfung der Eingabe, könnte der Trigger bei einem negativen Ergebnis die *INSERT*-Anweisung bereits an dieser Stelle abbrechen.

» Prüfen der Einschränkungen (Check Contraints) und der referenziellen Integrität: Schlägt eine der Prüfungen fehl, wird die Verarbeitung der *INSERT*-Anweisung abgebrochen.

» Erzeugen von Standardwerten: Zu jeder Spalte, die nicht durch die *INSERT*-Anweisung einen Wert erhält, wird gemäß der Spaltendefinition der Standardwert erstellt.

» Manipulation der Daten: Der Datensatz wird der Datenseite und somit der Tabelle hinzugefügt.

» Auslösen des *AFTER*-Triggers: Die im Trigger enthaltene Logik wird verarbeitet. Gibt es hierbei einen Fehler, wird der komplette *INSERT*-Vorgang abgebrochen und die Tabelle auf den Zustand vor der Aktion zurückgesetzt – es wird ein *ROLLBACK* ausgeführt.

» Bestätigen der Datenmanipulation: Die *INSERT*-Anweisung wird mit einem *COMMIT* bestätigt.

Interessant bei diesem Prozess ist der *AFTER*-Trigger. Obwohl dieser erst nach der eigentlichen Aktion ausgeführt wird, ist er dennoch ein fester Bestandteil dieser Aktion. Ein Fehler im *AFTER*-Trigger würde in unserem Beispiel die *INSERT*-Anweisung verhindern, auch wenn es beim eigentlichen Hinzufügen des Datensatzes kein Problem gibt. Ein Umstand, den es bei der

Programmierung von *AFTER*-Triggern zu beachten gilt. Dort sollten keine komplexen und fehleranfälligen Verarbeitungen erfolgen.

Die virtuellen Tabellen *inserted* und *deleted*

Kommen wir auf die virtuellen Tabellen *inserted* und *deleted* zurück. Beide Tabellen werden bei jeder Datenmanipulation erstellt und enthalten die davon betroffenen Daten.

Abhängig von der Aktion sind die Daten wie folgt auf die beiden Tabellen verteilt:

» *INSERT:* Die Tabelle *inserted* beinhaltet die neuen Datensätze, die Tabelle *deleted* ist leer.

» *UPDATE:* Die Tabelle *inserted* beinhaltet die neuen Werte der Datensätze, die Tabelle *deleted* die Werte vor der Änderung.

» *DELETE:* Die Tabelle *inserted* ist leer und die Tabelle *deleted* beinhaltet die gelöschten Datensätze.

Im Trigger können Sie nun mit den Daten dieser beiden Tabellen arbeiten. Die Daten lassen sich per *SELECT* auswerten und dabei mit anderen Tabellen verknüpfen sowie aggregieren, sortieren und filtern. Nur das Ändern der Daten ist nicht möglich.

Ob Sie die Daten der beiden virtuellen Tabellen im Trigger überhaupt benötigen, ist abhängig von der dort abgebildeten Logik. Ein Trigger reagiert auf die Aktion, für die er erstellt wurde und nicht unbedingt auf die dabei geänderten Daten. Sie könnten zum Beispiel mit einem AFTER-Trigger lediglich das Löschen von Datensätzen protokollieren.

Dabei wird nur der Zeitpunkt und der Benutzer in einem Protokoll gespeichert, nicht aber die gelöschten Daten. Möchten Sie hingegen die entfernten Datensätze ebenfalls protokollieren, müssen Sie im Trigger die Daten der virtuellen Tabelle *deleted* auswerten und diese im Protokoll speichern. Wie Sie die Daten der virtuellen Tabellen im Trigger verwenden, sehen Sie in den kommenden Beispielen.

Mehrere Trigger an einer Tabelle

Eine Tabelle kann maximal einen *INSTEAD OF*-Trigger enthalten, jedoch mehrere *AFTER*-Trigger. Es wäre also beispielsweise möglich, einen *AFTER*-Trigger für die Aktionen *INSERT*, *UPDATE* und *DELETE* zur Protokollierung von Datenänderungen zu definieren sowie einen weiteren *AFTER*-Trigger für einen besonderen und eher seltenen Fall einer *UPDATE*-Anweisung. In einem solchen Fall würden beim Ändern eines Datensatzes beide Trigger ausgelöst.

Grundsätzlich gibt es für die Ausführung mehrerer *AFTER*-Trigger keine Reihenfolge, es sei denn Sie legen diese mit der Systemprozedur *sp_SetTriggerOrder* fest. Allerdings definieren Sie hiermit lediglich die beiden *AFTER*-Trigger, die als erstes und als letztes ausgeführt werden.

Die Ausführungsreihenfolge für alle anderen *AFTER*-Trigger lässt sich nicht beeinflussen. Haben Sie also an einer Tabelle vier *AFTER*-Trigger, ist nur die Ausführungsreihenfolge für den als ersten

und den als letzten definierten Trigger festgelegt. Die Ausführung der beiden anderen Trigger erfolgt willkürlich.

Grundsätzlich sollten Sie sich gut überlegen, ob Sie die Logik eines *AFTER*-Triggers in einem Trigger abbilden oder ob Sie diese auf mehrere Trigger aufteilen. Wie so oft, gilt auch hier der Tipp: *KISS* – Keep It Simple And Stupid.

12.4 Trigger erstellen

Trigger werden wie die anderen SQL Server-Objekte mit der *CREATE*-Anweisung angelegt – hier mit *CREATE TRIGGER*. Das SQL Server Management Studio bietet zudem wie bei gespeicherten Prozeduren und Funktionen eine Vorlage mit der Syntax zum Anlegen eines Triggers.

Um diese Vorlage zu erhalten, navigieren Sie im Objekt-Explorer zu der Tabelle, an der der Trigger angelegt werden soll. Dort wählen Sie im Kontextmenü des Elements *Trigger* den Eintrag *Neuer Trigger*. Die Vorlage wird dann in einem neuen Abfragefenster geöffnet (siehe Abbildung 12.1).

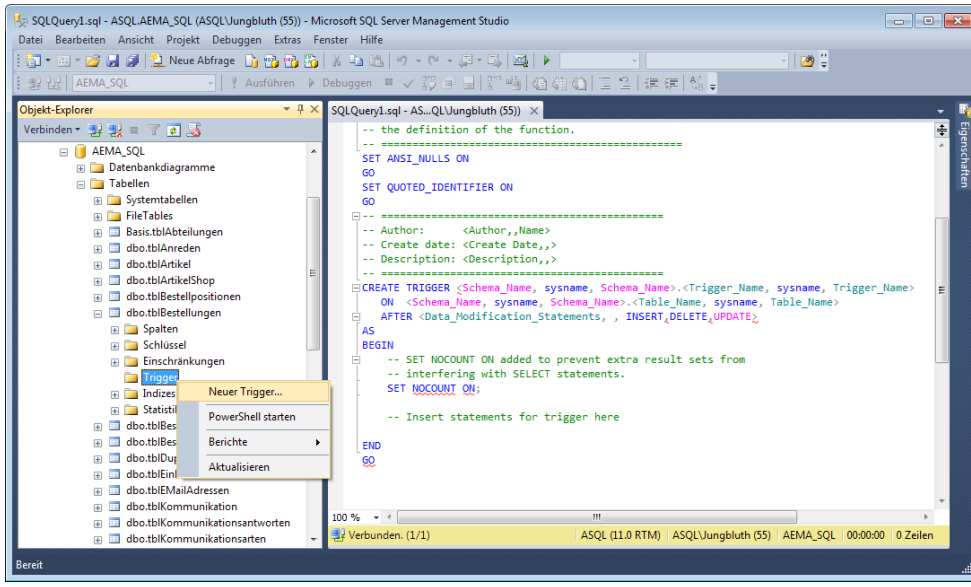


Abbildung 12.1: Einen neuen Trigger anlegen

Leider ist auch diese Vorlage nicht sonderlich hilfreich. Deshalb zeigen wir Ihnen in den folgenden Beispielen, wie Sie Trigger ohne die Vorlage anlegen.

Ein Trigger ist in seiner Struktur vielschichtiger als eine gespeicherte Prozedur, eine Sicht oder eine Funktion. Infolgedessen ist auch seine Syntax etwas komplexer. Bevor wir also zu den ei-

gentlichen Beispielen kommen, möchten wir Ihnen zunächst die Grundstruktur eines Triggers vorstellen.

Los geht es mit der *CREATE TRIGGER*-Anweisung. Es folgt der Name des Schemas, dem Sie den Trigger zuordnen möchten. Anschließend geben Sie dem Trigger einen Namen. Bei der Namensvergabe empfiehlt sich wieder die Verwendung eines Präfix, etwa *tr_*.

Ebenso empfehlenswert ist es, bereits im Triggernamen darauf hinzuweisen, zu welcher Tabelle der Trigger gehört, um welchen Typ es sich handelt und bei welchen Aktionen er ausgelöst wird. Hier ein Beispiel für den Namen eines Triggers, der als *AFTER*-Trigger auf *UPDATE*-Anweisungen an der Tabelle *tblBestellungen* reagiert:

```
dbo.tr_tblBestellungenAfterUpdate
```

Nach dem Schema und der Bezeichnung folgt mit dem Schlüsselwort *ON* die Zuweisung zur Tabelle. Anschließend wird die Art des Triggers mit *AFTER* oder *INSTEAD OF* angegeben und die Aktionen aufgelistet, bei denen der Trigger ausgelöst werden soll.

```
CREATE TRIGGER dbo.tr_tblBestellungenAfterUpdate
ON dbo.tblBestellungen AFTER Update
```

Mit dem Schlüsselwort *AS* leiten Sie dann den eigentlichen Programmcode ein, der auch hier wieder in einem *BEGIN...END*-Block enthalten ist.

```
CREATE TRIGGER dbo.tr_tblBestellungenAfterUpdate
ON dbo.tblBestellungen AFTER Update
AS
BEGIN
  SET NOCOUNT ON;
  -- Ihr Programmcode
END
```

Im obigen Skript sehen Sie die Anweisung *SET NOCOUNT ON*. Diese geben Sie in jedem Trigger aus den gleichen Gründen wie bei den gespeicherten Prozeduren an.

Innerhalb des *BEGIN...END*-Blocks können Sie mit T-SQL, Variablen, *IF*-Anweisungen und Schleifen arbeiten. Dabei sollten Sie jedoch nie die Performance außer Acht lassen. Bedenken Sie, dass die hier abgebildete Logik bei jeder Aktion ausgelöst wird, für die der Trigger definiert ist.

Die Trigger-Programmierung bietet eine besondere Spezialität: die Funktion *UPDATE()*. Hiermit prüfen Sie, ob sich bei der aktuellen Datenverarbeitung der Wert einer bestimmten Spalte geändert hat. Zur Prüfung geben Sie dazu den Spaltennamen als Parameter an:

```
IF UPDATE(<spaltenname>)
```

Wurde der Inhalt der Spalte geändert, liefert die Funktion den Wert *True*, ansonsten *False*. Auf diese Weise können Sie die weitere Verarbeitung des Triggers abhängig von der Änderung einer bestimmten Spalte steuern. Im Abschnitt »INSTEAD OF-Trigger erstellen«, Seite 294, erstellen sehen Sie den Einsatz dieser Funktion.

Neben den gespeicherten Prozeduren sind Trigger die einzigen SQL Server-Objekte, mit denen Sie Daten in Tabellen hinzufügen, ändern und löschen können. Jedoch sollten Sie in einem Trigger die SQL-Befehle *INSERT*, *UPDATE* und *DELETE* vorsichtig einsetzen und vorab prüfen, ob diese Aktionen nicht weitere Trigger an den betroffenen Tabellen auslösen.

Dies gilt auch für gespeicherte Prozeduren, die Sie in Triggern verwenden. Auch hier sind vorher die Auswirkungen zu prüfen.

12.4.1 INSTEAD OF-Trigger erstellen

Ein *INSTEAD OF*-Trigger wird vor der eigentlichen Datenmanipulation ausgeführt. Er ist also prädestiniert für Prüfungen. Bei einem negativen Ergebnis der Prüfung kann im Trigger die eigentliche Aktion abgebrochen oder auch korrigiert werden.

Liefert die Prüfung ein positives Ergebnis, bedeutet dies aber nicht, dass der Trigger die tatsächliche Aktion ausführt. *INSTEAD OF* steht nun mal für *Anstelle dessen*, weshalb auch die im Trigger enthaltene Logik anstelle der eigentlichen SQL-Anweisung ausgeführt wird. Soll also im positiven Fall der Prüfung die tatsächliche Aktion verarbeitet werden, muss der Quellcode des Triggers die SQL-Anweisung der tatsächlichen Aktion auch enthalten.

Ein *INSTEAD OF*-Trigger kann die eigentliche SQL-Anweisung ebenso komplett mit einer anderen Verarbeitung ersetzen. So wäre es beispielsweise möglich, dass ein Trigger anstelle einer ursprünglichen *DELETE*-Anweisung eine *UPDATE*-Anweisung ausführt. Auf diese Weise könnten Sie das Löschen von Datensätzen vermeiden und stattdessen diese als Inaktiv kennzeichnen.

Im Gegensatz zum *AFTER*-Trigger erweitert ein *INSTEAD OF*-Trigger also nicht den Vorgang der eigentlichen Datenverarbeitung, sondern ersetzt diesen durch seine eigene Logik. Für die Verarbeitung der Daten ist dann nicht mehr die ursprüngliche Aktion maßgebend, sondern die SQL-Anweisungen des *INSTEAD OF*-Triggers.

In unserem ersten Beispiel wollen wir dies nutzen und mit einem *INSTEAD OF*-Trigger das Löschen von Datensätzen in der Tabelle *tblRechnungen* vermeiden. Da hier eine bestimmte Datenverarbeitung kategorisch ausgeschlossen werden soll, ist dies ein klassicher Fall für einen *INSTEAD OF*-Trigger.

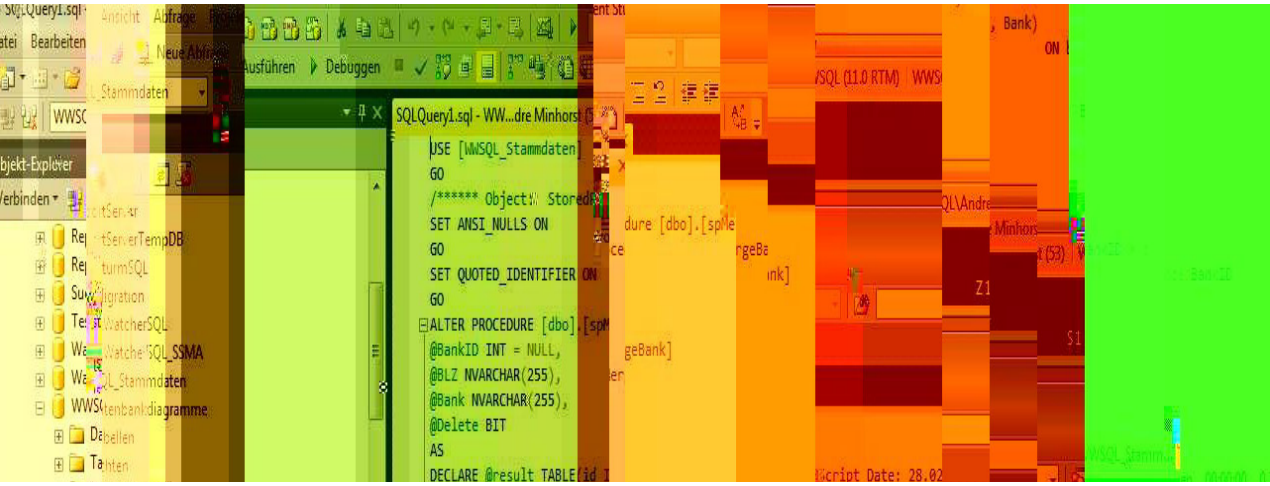
Für den neuen Trigger öffnen Sie zunächst ein neues Abfragefenster über den Eintrag *Neue Abfrage* aus dem Kontextmenü der Beispieldatenbank *AEMA_SQL*. Dort geben Sie nun die *CREATE TRIGGER*-Anweisung ein, gefolgt vom Schema und dem Namen für den Trigger.

```
CREATE TRIGGER dbo.tr_tblRechnungenInsteadOfDelete
```

Das es sich um einen Trigger an der Tabelle *tblRechnungen* handelt, der zudem auf die Aktion *DELETE* reagieren soll, definieren Sie mit den folgenden Parametern.

```
ON dbo.tblRechnungen INSTEAD OF Delete
```

... DIE ÜBRIGEN SEITEN GIBT'S GEDRUCKT UNTER SHOP.MINHORST.COM



BERND JUNGBLUTH
ANDRÉ MINHORST

André Minhorst Verlag

13 SQL Server-Zugriff per VBA

Sie werden an verschiedenen Stellen per VBA auf die Tabellen der SQL Server-Datenbank zugreifen müssen – sei es, um eine ODBC-Verknüpfung herzustellen oder zu aktualisieren, das Ergebnis einer gespeicherten Abfrage abzurufen oder eine solche auszuführen oder auch um ein Recordset auf Basis einer gespeicherten Prozedur einem Formular oder einem Steuerelement zuzuweisen. Die dazu notwendigen Techniken lernen Sie in diesem Kapitel kennen.

Für den Zugriff auf die Daten der SQL Server-Datenbank per VBA stehen mit DAO und ADO gleich zwei Möglichkeiten zur Verfügung. Während Sie mit DAO die Daten einer SQL Server-Datenbank nur indirekt über eingebundene Tabellen oder Pass-Through-Abfragen verarbeiten können, bietet ADO den direkten Zugriff auf die SQL Server-Datenbank. Leider wird der für ADO notwendige Provider Microsoft *OLE DB Provider for SQL Server* ab Sommer 2017 nicht mehr unterstützt. Mehr dazu lesen Sie im Abschnitt »ADO oder DAO?«, Seite 20. Diese Abkündigung ist auch der Grund, warum wir in diesem Kapitel ausnahmslos den Datenzugriff per DAO beschreiben.

Beispiele

Die Beispiele zu diesem Kapitel finden Sie in der Datenbank *13_SQLServerZugriffPerVBA.accdb*.

Als SQL Server-Datenbank verwenden Sie die Datenbank *AEMA_SQL* aus dem Download.

Die T-SQL-Anweisungen zum Erstellen der in diesem Kapitel verwendeten gespeicherten Prozeduren finden Sie jeweils unter dem entsprechenden Code in der Access-Beispieldatenbank.

13.1 Verbindungszeichenfolgen

Als erstes benötigen Sie natürlich Zugriff auf die SQL Server-Datenbank. Voraussetzung dafür ist eine geeignete Verbindungszeichenfolge. Bei den Verbindungszeichenfolgen gibt es verschiedene Ansätze.

Sie sind relativ flexibel, wenn Sie die notwendigen Informationen in einer lokalen Tabelle im Access-Frontend speichern und von Access aus per VBA darauf zugreifen, um Verbindungszeichenfolgen daraus zu erstellen. Alternativ können Sie Verbindungszeichenfolgen in einer DSN-Datei oder als System-DSN in der Registry speichern – wir gehen an dieser Stelle jedoch auf die Variante der tabellenbasierten Verbindungszeichenfolge ein. Die notwendigen Daten speichern wir in den Beispieldatenbanken in einer Tabelle namens *tblVerbindungszeichenfolgen* (siehe Abbildung 13.1).

Außerdem benötigen wir eine Tabelle namens *tblTreiber*, welche die Daten der gängigen Treiber enthält. Beide Tabellen werden später in Kapitel *»Access-SQL Server-Tools«, Seite 459* erläutert.

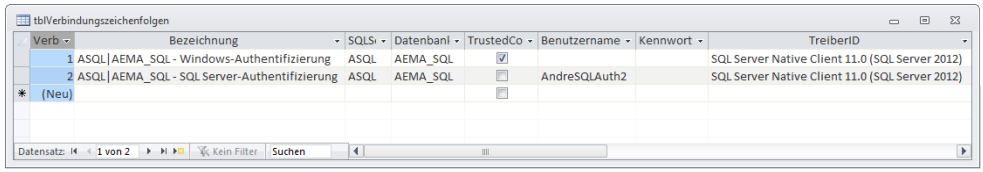


Abbildung 13.1: Die Tabelle speichert die Informationen zu den Verbindungszeichenfolgen

In der Beispielanwendung verwenden wir die folgende Prozedur, um eine Verbindungszeichenfolge aus den Daten der Tabelle *tblVerbindungszeichenfolgen* zu ermitteln (siehe Modul *mdlToolsSQLServer*):

```
Public Function VerbindungszeichenfolgeNachID(lngVerbindungszeichenfolgeID As Long) _
                                                                        As String
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Dim strTemp As String
    Dim strTreiber As String
    Dim strServer As String
    Dim strDatenbank As String
    Set db = CurrentDb
    Set rst = db.OpenRecordset("SELECT * FROM tblVerbindungszeichenfolgen WHERE _
                        VerbindungszeichenfolgeID = " & lngVerbindungszeichenfolgeID)
    strTreiber = DLookup("Treiber", "tblTreiber", "TreiberID = " & rst!TreiberID)
    strServer = rst!SQLServer
    strDatenbank = rst!Datenbank
    strTemp = "ODBC;DRIVER={" & strTreiber & "};" & "SERVER=" & strServer & ";" _
        & "DATABASE=" & strDatenbank & ";"
    If rst!TrustedConnection = True Then
        strTemp = strTemp & "Trusted_Connection=Yes"
    Else
        If Len(strBenutzername) = 0 Then
            strBenutzername = Nz(rst!Benutzername, "")
        End If
        If Len(strKennwort) = 0 Then
            strKennwort = Nz(rst!Kennwort, "")
        End If
        If Len(strBenutzername) > 0 Then
            strTemp = strTemp & "UID=" & strBenutzername & ";"
            If Len(strKennwort) > 0 Then
                strTemp = strTemp & "PWD=" & strKennwort
            End If
        End If
    End If
    VerbindungszeichenfolgeNachID = strTemp
End Function
```

Die Prozedur liest den Datensatz der Tabelle *tblVerbindungszeichenfolgen* ein, dessen Primärschlüsselwert dem per Parameter übergebenen Wert entspricht. Die beiden weiteren, optionalen Parameter *strBenutzername* und *strKennwort* sind speziell für den Einsatz mit der SQL Server-

Authentifizierung vorgesehen. In diesem Fall muss die Access-Anwendung die Benutzerdaten bereitstellen, um eine Verbindung herzustellen. Diese sollen aber nicht in der Datenbank gespeichert werden, da die Daten sonst für jedermann zugänglich wären. Die Parameter haben den Zweck, dass Sie diese zuvor per Formular vom Benutzer einmalig pro Sitzung abfragen und in entsprechenden Variablen speichern, die gleichnamig sind und wie folgt deklariert werden:

```
Public strBenutzername As String
Public strKennwort As String
```

Deren Inhalt kann dann bei Bedarf an die Funktion *VerbindungszeichenfolgeNachID* übergeben werden. Diese Funktion setzt die einzelnen Elemente dann zu einer Verbindungszeichenfolge zusammen. Abhängig davon, ob Windows-Authentifizierung oder SQL Server-Authentifizierung gewählt wurde, erhält die Verbindungszeichenfolge das Name-Wert-Paar *Trusted_Connection =Yes*, sonst die Benutzerdaten in der Form *UID=<Benutzername>;PWD=<Kennwort>*. Dabei prüft die Funktion zunächst, ob *strBenutzername* beziehungsweise *strKennwort* einen Wert enthalten. Falls nicht, verwendet sie die Werte aus der Tabelle *tblVerbindungszeichenfolgen*.

An dieser Stelle weisen wir nochmals darauf hin, dass Sie nach Möglichkeit die Windows-Authentifizierung nutzen sollten.

13.1.1 Standardverbindungszeichenfolge

Diese Variante der Funktion erlaubt die flexible Auswahl einer der in der Tabelle *tblVerbindungszeichenfolgen* gespeicherten Verbindungen per ID. Das ist während der Entwicklung sehr praktisch, aber wenn die Datenbank einmal beim Anwender gelandet ist, reicht wohl eine einzige Verbindungszeichenfoglge aus. Ist dies der Fall, erledigen Sie zwei Schritte:

» Hinzufügen eines Feldes namens *Aktiv* mit dem Datentyp *Ja/Nein* zur Tabelle *tblVerbindungszeichenfolge* und Aktivieren der Option für die standardmäßig zu verwendende Verbindungszeichenfolge sowie

» Erstellen einer Funktion, welche die Routine *VerbindungszeichenfolgeNachID* für die als Standard festgelegte Verbindungszeichenfolge aufruft.

Die dazu verwendete Funktion sieht wie folgt aus:

```
Public Function Standardverbindungszeichenfolge() As String
    Dim lngAktivID As Long
    If Not lngAktivID = 0 Then
        Standardverbindungszeichenfolge = VerbindungszeichenfolgeNachID(lngAktivID)
    Else
        MsgBox "Achtung: Es ist keine Verbindungszeichenfolge als aktiv gekennzeichnet."
    End If
End Function
```

Die Funktion ermittelt per *DLookup*-Funktion den ersten Eintrag der Tabelle *tblVerbindungszeichenfolgen*, dessen Feld *Aktiv* den Wert *True* aufweist. Dieser wird dann der Funktion *Verbin-*

dungszeichenfolgeNachID übergeben, um die entsprechende Verbindungszeichenfolge zu ermitteln. Sollte keine Verbindungszeichenfolge als aktiv markiert sein, erscheint eine entsprechende Meldung.

13.1.2 ID der aktiven Verbindungszeichenfolge ermitteln

Andere Routinen erwarten gegebenenfalls die ID der zu verwendenden Verbindungszeichenfolge. Wenn Sie nicht explizit einen Wert des Feldes *VerbindungszeichenfolgeID* der Tabelle *tblVerbindungszeichenfolgen* übergeben möchten, können Sie folgender Funktion den Wert für den aktiven Datensatz ermitteln:

```
Public Function AktiveVerbindungszeichenfolgeID() As Long
    Dim lngAktivID As Long
    lngAktivID = Nz(DLookup("VerbindungszeichenfolgeID", _
        "tblVerbindungszeichenfolgen", "Aktiv = True"), 0)
    If lngAktivID = 0 Then
        MsgBox "Achtung: Es ist keine Verbindungszeichenfolge als aktiv gekennzeichnet."
    End If
    AktiveVerbindungszeichenfolgeID = lngAktivID
End Function
```

Auch diese Funktion liefert eine Meldung, wenn keine aktive Verbindungszeichenfolge gefunden werden konnte.

13.2 Verbindung und Zugriffsdaten prüfen

Aufbauend auf den vorherigen Funktionen wollen wir nun eine weitere Funktion einführen, die prüft, ob eine Verbindungszeichenfolge funktioniert. Diese heißt *VerbindungTesten* und befindet sich ebenfalls im Modul *mdlToolsSQLServer*. Die Funktion erledigt gleich zwei Aufgaben:

» Sie testet die Verbindung und liefert einen *Boolean*-Wert zurück, der über das Ergebnis Auskunft gibt.

» Wenn die Verbindung SQL Server-Authentifizierung nutzt, soll zumindest das Kennwort nicht in der Verbindungszeichenfolge und auch nicht in der Tabelle *tblVerbindungszeichenfolgen* enthalten sein. Die Funktion prüft die Verbindung für die verfügbaren Werte von Benutzername und Kennwort und zeigt ein Formular zur erneuten Eingabe oder Korrektur der Werte an, wenn damit keine Verbindung hergestellt werden konnte.

Die Funktion erwartet folgende Parameter:

» *lngVerbindungszeichenfolgeID*: Primärschlüsselwert der zu verwendenden Verbindung aus der Tabelle *tblVerbindungszeichenfolgen*

» *strVerbindungszeichenfolge*: Rückgabeparameter, der die geprüfte Verbindungszeichenfolge zurückgibt

Nun hat diese Funktion nicht nur die Aufgabe, einfach nur Verbindungen zu testen, sondern speziell im Falle von Verbindungen auf Basis der SQL Server-Authentifizierung gleich noch eventuell fehlende Daten wie *Benutzername* oder *Kennwort* abzufragen.

Diese sollen, sofern ermittelt, natürlich nicht etwa in die Tabelle *tblVerbindungszeichenfolgen* eingetragen werden, wo sie leicht abzugreifen wären, sondern nur in globalen Variablen. Diese deklarieren Sie wie folgt ebenfalls im Modul *mdlToolsSQLServer*:

```
Public strBenutzername As String
Public strKennwort As String
```

13.2.1 Funktion zum Testen der Verbindung

Die Funktion deklariert zunächst zwei Variable. *bolTrustedConnection* ermittelt, ob eine Verbindung mit Windows-Authentifizierung aufgebaut werden soll, *bolVerbindungHerstellt* nimmt das Ergebnis einer Funktion zum Herstellen der Verbindung auf:

```
Public Function VerbindungTesten(lngVerbindungszeichenfolgeID As Long, _
        strVerbindungszeichenfolge As String) As Boolean
    Dim bolTrustedConnection As Boolean
    Dim bolVerbindungHergestellt As Boolean
```

bolTrustedConnection wird gleich zu Beginn mit dem entsprechenden Wert der Tabelle *tblVerbindungszeichenfolgen* gefüllt. Anschließend prüft die Funktion, ob *strBenutzername* und/oder *strKennwort* leer sind.

In diesem Fall ruft sie eine weitere Funktion auf, die einen Dialog zum Eingeben der fehlenden Daten anzeigt – mehr dazu weiter unten. Liefert dieser Dialog keine Anmeldedaten, weil der Benutzer diesen abbricht, wird die Funktion beendet und liefert den Wert *False* zurück:

```
    bolTrustedConnection = DLookup("TrustedConnection", "tblVerbindungszeichenfolgen", _
        "VerbindungszeichenfolgeID = " & lngVerbindungszeichenfolgeID)
    If (Len(strBenutzername) * Len(strKennwort) = 0) And Not bolTrustedConnection Then
        If LogindatenErmitteln(lngVerbindungszeichenfolgeID) = False Then
            Exit Function
        End If
    End If
```

Nun ermittelt die Funktion *VerbindungszeichenfolgeNachID* eine Verbindungszeichenfolge, und zwar auf Basis der ID der zu verwendenden Verbindung aus der Tabelle *tblVerbindungszeichenfolgen* und der Werte aus *strBenutzername* und *strKennwort*:

```
    strVerbindungszeichenfolge = VerbindungszeichenfolgeNachID( _
        lngVerbindungszeichenfolgeID)
```

Die Funktion *VerbindungHerstellen* prüft, ob mit dieser Verbindungszeichenfolge ein Zugriff auf die Datenbank möglich ist, das Ergebnis landet in der Variablen *bolVerbindungHergestellt*:

```
    bolVerbindungHergestellt = VerbindungHerstellen(strVerbindungszeichenfolge)
```

Die folgende *Do While*-Schleife wird überhaupt nur durchlaufen, wenn noch keine Verbindung mit den erfassten Daten hergestellt werden konnte – anderenfalls ist die Abbruchbedingung direkt erfüllt und die Funktion wird mit dem Rückgabewert *True* beendet.

Sollte der Benutzer jedoch noch nicht die korrekten Daten eingegeben haben, beginnt der Durchlauf der *Do While*-Schleife. Diese gibt zunächst eine Meldung aus, dass der erste Versuch nicht gelungen ist. Dann prüft die Funktion, ob es sich um Windows-Authentifizierung oder SQL Server-Authentifizierung handelt. Im ersteren Fall lässt sich das Problem an dieser Stelle nicht beheben, die Funktion endet mit dem Rückgabewert *False*. Bei SQL Server-Authentifizierung erscheint ebenfalls die Meldung, der Benutzer kann jedoch den Benutzernamen korrigieren:

```
    Do While Not bolVerbindungHergestellt
        MsgBox "Die Verbindung konnte nicht hergestellt werden."
        On Error GoTo 0
        If Not bolTrustedConnection Then
            If LogindatenErmitteln(lngVerbindungszeichenfolgeID) = True Then
                strVerbindungszeichenfolge = VerbindungszeichenfolgeNachID( _
                    lngVerbindungszeichenfolgeID)
                bolVerbindungHergestellt = VerbindungHerstellen( _
                    strVerbindungszeichenfolge)
            Else
                Exit Function
            End If
        Else
            Exit Function
        End If
    Loop
    VerbindungTesten = True
End Function
```

Beim Verbinden können beispielsweise die folgenden Probleme auftreten:

» Der Name des SQL Servers ist falsch beziehungsweise der SQL Server nicht verfügbar. Es dauert eine Weile, bis dieser Fehler auftritt.

» Der Name der Datenbank ist falsch beziehungsweise die Datenbank ist nicht vorhanden.

» Bei der Windows-Authentifizierung greift ein Benutzer ohne ausreichende Berechtigungen/Benutzerkonto auf die Datenbank zu. Dies löst einen Fehler aus, bei dem im Anschluss der eingebaute Verbindungsdialog zur Korrektur der Verbindungsdaten angezeigt wird.

» Bei der SQL Server-Authentifizierung stimmen die Anmeldedaten (Benutzername und Kennwort) nicht. Dies löst einen Fehler aus, der abgefangen werden kann. Wichtig: Dies gelingt nur, wenn die Parameter *USR* und *PWD* in der Verbindungszeichenfolge vorhanden sind (*...;USR=;PWD=* reicht aus). Fehlen die Daten, erscheint wiederum der eingebaute Anmeldedialog.

In den meisten Fällen liefert der Treiber jedoch nur einen wagen Hinweis auf den Fehler. Eine genaue Ermittlung des Problems würde hier den Rahmen sprengen.

13.2.2 Login-Formular

Die Funktion *LoginDatenErmitteln* öffnet das Formular *frmLogin* und übergibt als Öffnungsargument die ID der zu verwendenden Verbindungszeichenfolge. Der Code dieser Funktion wird nach dem Öffnen des Formulars solange angehalten, bis der Benutzer das Formular mit der Schaltfläche *OK* unsichtbar macht oder es mit *Abbrechen* schließt. Danach prüft die Funktion, ob das Formular noch geöffnet, aber unsichtbar ist und liest in diesem Fall die eingegebenen Anmeldedaten in die beiden Variablen *strBenutzername* und *strKennwort* ein. Schließlich schließt sie das Formular endgültig und liefert den Wert *True* als Rückgabewert:

```
Public Function LogindatenErmitteln(lngVerbindungszeichenfolgeID) As Boolean
    DoCmd.OpenForm "frmLogin", WindowMode:=acDialog, OpenArgs:=lngVerbindungszeichenfolgeID
    If IstFormularGeoeffnet("frmLogin") Then
        strBenutzername = Forms!frmLogin!txtBenutzername
        strKennwort = Forms!frmLogin!txtKennwort
        DoCmd.Close acForm, "frmLogin"
        LogindatenErmitteln = True
    End If
End Function
```

Das Formular sieht im Entwurf wie in Abbildung 13.2 aus.

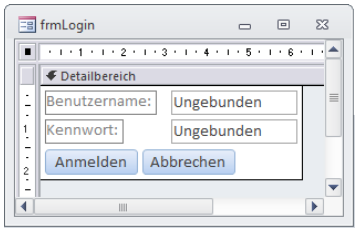


Abbildung 13.2: Formular zum Abfragen von Benutzername und Kennwort

Damit das Formular beim Öffnen gleich die in der Tabelle *tblVerbindungszeichenfolgen* enthaltenen Anmeldedaten in den beiden Textfeldern anzeigt, löst es die folgende Ereignisprozedur aus:

```
Private Sub Form_Open(Cancel As Integer)
    Dim strVerbindungszeichenfolge As String
    lngVerbindungszeichenfolgeID = Nz(Me.OpenArgs)
    If lngVerbindungszeichenfolgeID = 0 Then
        lngVerbindungszeichenfolgeID = AktiveVerbindungszeichenfolgeID
    End If
    Me!txtBenutzername = Nz(DLookup("Benutzername", "tblVerbindungszeichenfolgen", _
        "VerbindungszeichenfolgeID = " & lngVerbindungszeichenfolgeID), strBenutzername)
    Me!txtKennwort = Nz(DLookup("Kennwort", "tblVerbindungszeichenfolgen", _
        "VerbindungszeichenfolgeID = " & lngVerbindungszeichenfolgeID), strKennwort)
End Sub
```

Die Prozedur schreibt die per Öffnungsargument übergebene *VerbindungszeichenfolgeID* in eine entsprechende Variable, die nachfolgend im Kriterium zweier *DLookup*-Aufrufe eingesetzt wird,

um Benutzername und Kennwort des entsprechenden Eintrags der Tabelle *tblVerbindungszeichenfolgen* zu ermitteln.

13.2.3 Herstellen einer Verbindung

Die Funktion *VerbindungHerstellen* erwartet die Verbindungszeichenfolge als Parameter. Sie erstellt ein neues temporäres *QueryDef*-Objekt und weist diesem die Verbindungszeichenfolge sowie eine Dummy-Abfrage (hier *SELECT 1 AS Test*) zu.

Dann öffnet sie nach vorheriger Deaktivierung der Fehlerbehandlung ein auf dem *QueryDef* basierendes Recordset. Gelingt dies, war der Verbindungsaufbau erfolgreich und die Funktion liefert den Wert *True* zurück:

```
Public Function VerbindungHerstellen(strVerbindungszeichenfolge As String) As Long
    Dim db As DAO.Database
    Dim qdf As DAO.QueryDef
    Set db = CurrentDb
    Set qdf = db.CreateQueryDef("")
    With qdf
        .Connect = strVerbindungszeichenfolge
        .ReturnsRecords = True
        .SQL = "SELECT 1 AS Test"
        On Error Resume Next
        .OpenRecordset
        If Err.Number = 0 Then
            VerbindungHerstellen = True
        End If
        On Error GoTo 0
    End With
End Function
```

13.2.4 Verwendung dieser Funktionen

Die Funktion *VerbindungTesten* prüft nicht nur, ob der Zugriff auf die Tabellen der SQL Server-Datenbank gelingt, sondern liefert auch gleich noch die korrekte Verbindungszeichenfolge zurück. Mit dieser Zeichenfolge im Gepäck können Sie gleich einige wichtige Schritte durchführen, die Probleme mit dem Zugriff auf eingebundene Objekte wie per ODCB verknüpfte Tabellen oder Pass-Through-Abfragen verhindern.

Im Falle der verknüpften Tabellen treten Probleme auf, wenn Sie diese unter Windows-Authentifizierung eingebunden haben und der Kunde dann per SQL Server-Authentifizierung darauf zugreifen will – dies zeigt wiederum den eingebauten Anmeldedialog an. Andersherum verhält es sich ebenso.

Steht also ein Wechsel zwischen den beiden Authentifizierungsarten an, ist ein erneutes Einbinden aller Tabellen nötig. Haben Sie die Tabellen unter SQL Server-Authentifizierung eingebunden und melden sich dann unter einem anderen SQL Server-Account an, greifen Sie automatisch

unter diesem Account auf die Tabellen zu. Den Unterschied machen die in der Systemtabelle *MSysObjects* gespeicherten Verbindungseigenschaften.

Enthalten diese den Ausdruck *Trusted_Connection=Yes*, können Sie mit Windows-Authentifizierung darauf zugreifen, sonst nur mit SQL Server-Authentifizierung. Alternativ müssen Sie die Verknüpfungen aktualisieren. Dies ist dank der Funktion *TabelleVerknuepfen* aus »Tabellen per VBA verknüpfen«, Seite 97 zwar auch kein Problem, je nach Anzahl der eingebundenen Tabellen kann dieser Vorgang jedoch entsprechend Zeit beanspruchen.

Pass-Through-Abfragen sollten Sie grundsätzlich beim Starten der Anwendung mit der aktuellen Verbindungszeichenfolge ausstatten, bei SQL Server-Authentifizierung allerdings ohne Angabe des Kennwortes – dieses könnte sonst einfach ausgelesen werden. Dies erledigt die nachfolgend vorgestellte Funktion.

13.2.5 Pass-Through-Abfragen aktualisieren

Die Funktion *PTAbfrageAktualisieren* erwartet den Namen der Pass-Through-Abfrage und die Verbindungszeichenfolge als Parameter. Da diese den in diesem Buch vorgestellten Techniken zufolge mit der Funktion *VerbindungszeichenfolgeNachID* zusammengestellt wird, enthält sie bei Benutzung der SQL Server-Authentifizierung gegebenenfalls das Kennwort des aktuellen Benutzers.

Daher enthält die Funktion einige Zeilen Code, welche die Verbindungszeichenfolge auf das Auftreten des Ausdrucks *;PWD=* untersuchen. Ist dieser vorhanden, wird eine neue Verbindungszeichenfolge namens *strVerbindungOhnePWD* zusammengestellt, die alle Zeichen bis zum Auftreten von *;PWD=* enthält und gegebenenfalls alle Zeichen hinter dem nachfolgenden Semikolon:

```
Public Sub PTAbfrageAktualisieren(strPTAbfrage As String, _
        strVerbindungszeichenfolge As String)
    Dim db As DAO.Database
    Dim qdf As DAO.QueryDef
    Dim intStart As Integer
    Dim intEnde As Integer
    Dim strVerbindungOhnePWD As String
    intStart = InStr(1, strVerbindungszeichenfolge, ";PWD")
    If intStart > 0 Then
        strVerbindungOhnePWD = Left(strVerbindungszeichenfolge, intStart)
        intEnde = InStr(intStart + 1, strVerbindungszeichenfolge, ";")
        If Not intEnde = 0 Then
            strVerbindungOhnePWD = strVerbindungOhnePWD & _
                Mid(strVerbindungszeichenfolge, intEnde + 1)
        End If
    Else
        strVerbindungOhnePWD = strVerbindungszeichenfolge
    End If
    Set db = CurrentDb
```

```
    Set qdf = db.QueryDefs(strPTAbfrage)
    qdf.Connect = strVerbindungOhnePWD
End Sub
```

Wenn Sie mit dieser Prozedur alle Pass-Through-Abfragen aktualisieren wollen, durchlaufen Sie in einer weiteren Prozedur alle Einträge der Tabelle *MSysObjects*, deren Feld *Connect* einen Wert enthält, der mit *ODBC=* beginnt und rufen von dort für jeden Eintrag die Prozedur *PTAbfrageAktualisieren* auf:

```
Public Sub AlleAbfragenAktualisieren()
    Dim db As DAO.Database
    Dim qdf As DAO.QueryDef
    Set db = CurrentDb
    For Each qdf In db.QueryDefs
        If InStr(Nz(qdf.Connect,""), "ODBC;") > 0 Then
            PTAbfrageAktualisieren qdf.Name, Standardverbindungszeichenfolge
        End If
    Next qdf
End Sub
```

13.3 Aktionsabfrage ausführen

Aktionsabfragen sind Abfragen, die Daten ändern – also Lösch-, Aktualisierungs- und Anfügeabfragen. Diese können Sie auf verschiedene Arten ausführen:

» Erstellen einer Aktionsabfrage in Access, die sich auf die Daten einer per ODBC verknüpften Tabelle bezieht

» Erstellen einer Pass-Through-Abfrage, welche die Aktionsabfrage enthält und diese direkt an den SQL Server übermittelt

» Erstellen einer gespeicherten Prozedur, welche die Aktionsabfrage enthält und die notwendigen Parameter entgegen nimmt – also beispielsweise die ID eines zu löschenden Datensatzes –, und die über eine Pass-Through-Abfrage aufgerufen wird.

Die Auflistung stellt auch die Reihenfolge bezüglich der Performance dieser Möglichkeiten dar: Der erste Punkt liefert die schlechteste Performance, beim zweiten Punkt wird diese schon etwas besser und beim dritten Punkt ist sie optimal.

Aus diesem Grund schauen wir uns nachfolgend lediglich die zweite und die dritte Variante an.

13.3.1 Datensatz löschen per SQL

Bei der ersten Variante legen Sie eine Pass-Through-Abfrage mit der auszuführenden *DELETE*-Anweisung an (siehe Abbildung 13.3). Dazu sind drei Schritte nötig:

» Erstellen einer neuen, leeren Abfrage und Schließen des Dialogs *Tabelle anzeigen*

» Wechseln des Abfragetyps auf *Pass-Through*

» Einstellen der Eigenschaft *ODBC-Verbindung* auf die gewünschte Verbindungszeichenfolge (hier *ODBC;DRIVER={SQL Server Native Client 11.0};SERVER=ASQL;DATABASE=AEMA_SQL; Trusted_Connection=Yes*)

» Einstellen der Eigenschaft *Liefert Datensätze* auf *Nein*

» Eintragen der *DELETE*-Anweisung

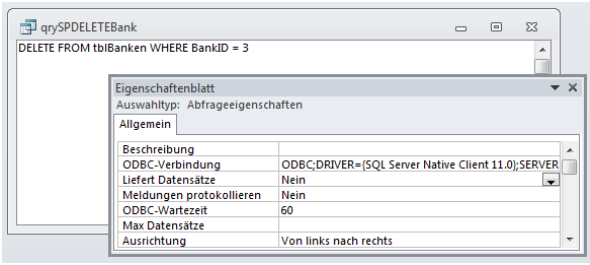


Abbildung 13.3: Statisches Löschen eines Datensatzes per Pass-Through-Abfrage

Die Abfrage können Sie dann per VBA mit einer einzigen Anweisung ausführen:

```
CurrentDb.QueryDefs("qrySPDELETEBank").Execute
```

Damit haben Sie allerdings noch nicht viel gewonnen: Die Anweisung löscht ja nur genau den Datensatz, dessen ID Sie als Kriterium angegeben haben. Immerhin haben wir aber bereits eine Abfrage erstellt, die den richtigen Typ aufweist, die Verbindungszeichenfolge enthält und deren Eigenschaft *Liefert Datensätze* auf *Nein* eingestellt ist. Diese nutzen wir nun, um gezielt einen bestimmten Datensatz zu löschen.

Die folgende Prozedur (wie auch die weiteren Beispiele im Modul *mdlBeispiele*) erwartet den Primärschlüsselwert des zu löschenden Datensatzes als Parameter:

```
Public Sub BankLoeschen_PT(lngBankID As Long)
    Dim db As DAO.Database
    Dim qdf As DAO.QueryDef
    Set db = CurrentDb
    Set qdf = db.QueryDefs("qrySPDELETEBank")
    qdf.SQL = "DELETE FROM dbo.tblBanken WHERE BankID = " & lngBankID
    qdf.Execute
    Set qdf = Nothing
    Set db = Nothing
End Sub
```

Sie referenziert die soeben erstellte Abfrage und ändert die enthaltene SQL-Anweisung so, dass diese als Kriterium den per Parameter übergebenen Primärschlüsselwert enthält und führt die geänderte Abfrage mit der *Execute*-Anweisung aus. Der Aufruf dieser Prozedur sieht etwa so aus:

```
BankLoeschen_PT 2
```

Diese Variante hat noch folgende Nachteile:

» Die an den SQL Server übergebene SQL-Anweisung wird dynamisch zusammengesetzt. Wenn sich die SQL-Anweisung dabei von einer bereits verwendeten unterscheidet, also etwa ein anderer Parameterwert zum Einsatz kommt, muss der Ausführungsplan neu erstellt werden.

» Die Verbindungszeichenfolge ist in der Abfrage gespeichert. Wenn sich diese ändert, muss sie in jeder Abfrage angepasst werden.

» Wir erfahren nicht, ob die Aktion erfolgreich war und wie viele Datensätze gelöscht wurden.

In den folgenden beiden Abschnitten kümmern wir uns um diese Nachteile.

13.3.2 Datensatz löschen per gespeicherter Prozedur

Als Erstes sorgen wir dafür, dass der SQL Server unabhängig vom übergebenen Parameter nur einen Ausführungsplan für die Abfrage erstellt, speichert und bei weiteren Aufrufen wiederverwendet. Dazu erstellen wir eine gespeicherte Prozedur, und zwar mit folgendem SQL-Skript:

```
CREATE PROCEDURE dbo.spDELETEBankNachID (@BankID int)
AS
SET NOCOUNT ON;
DELETE FROM tblBanken WHERE BankID = @BankID;
```

Die dadurch erzeugte gespeicherte Prozedur erwartet den Primärschlüsselwert des zu löschenden Datensatzes als Parameter. Wenn Sie die gespeicherte Prozedur direkt vom Abfragefenster des SQL Servers aus ausführen wollten, würden Sie dies mit folgender Anweisung erledigen:

```
EXEC dbo.spDELETEBankNachID 7
```

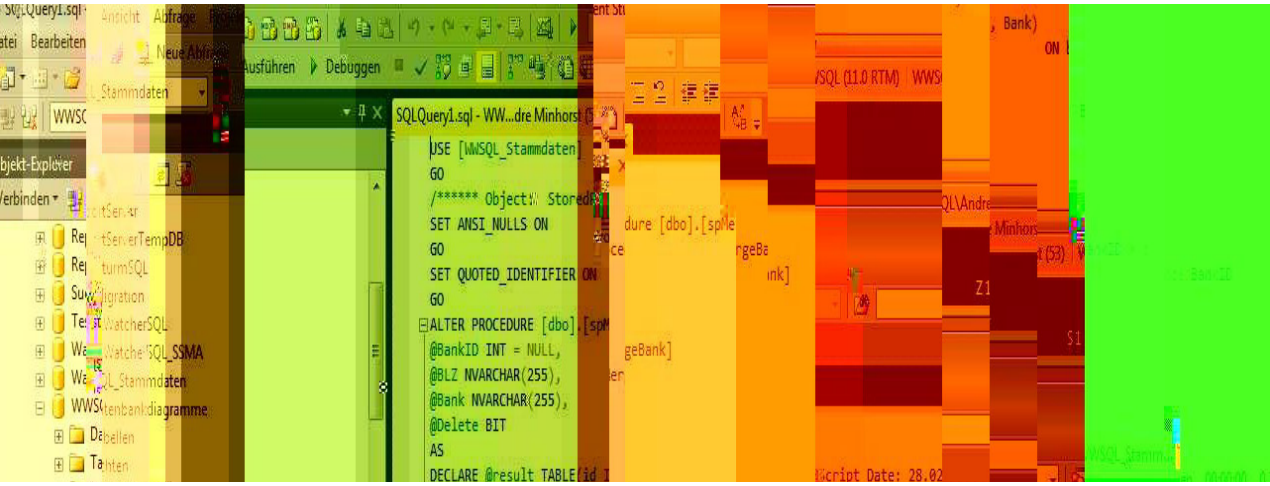
Wir wollen dies allerdings von Access aus erledigen. Also erstellen Sie zunächst eine neue Abfrage, wandeln diese in eine Pass-Through-Abfrage um und legen den SQL-Ausdruck aus Abbildung 13.4 fest.

Abbildung 13.4: Aufruf der gespeicherten Prozedur von einer Access-Abfrage aus

In dieser Abfrage müssen Sie nun natürlich ebenfalls den Primärschlüsselwert des zu löschenden Datensatzes als Parameter angeben. Dies erledigen Sie ähnlich wie oben:

```
Public Sub BankLoeschenNachID_PT(lngBankID As Long)
    Dim db As DAO.Database
    Dim qdf As DAO.QueryDef
```

... DIE ÜBRIGEN SEITEN GIBT'S GEDRUCKT UNTER SHOP.MINHORST.COM



BERND JUNGBLUTH
ANDRÉ MINHORST

André Minhorst Verlag

14 Formulare und Berichte

Formulare sind das A und O bei der Bearbeitung von Daten in einer Datenbankanwendung. Sie zeigen die Daten an und stellen Möglichkeiten zum Erstellen, Bearbeiten und Löschen bereit. Wie erfolgt der Zugriff in einer Kombination aus Access-Frontend und SQL Server-Backend? Wie füllen Sie Formulare, Unterformulare und Steuerelemente wie das Kombinationsfeld oder das Listenfeld? Und funktioniert dies alles auch bei Berichten? All diese Fragen beantwortet das vorliegende Kapitel.

14.1 Daten von Access und vom SQL Server

Unter Access ist es so einfach: Sie legen für ein Formular eine *Datenherkunft* fest und legen Steuerelemente an, die an die Felder der Datenherkunft gebunden sind. Genauso einfach ist es bei Kombinations- und Listenfeldern – hier verwenden Sie die Eigenschaft *Datensatzherkunft*, um die auszuwählenden Daten einzubinden. Fehlen noch die Unterformulare, bei denen Sie wie im Hauptformular einfach die Datenherkunft auf die gewünschte Tabelle oder Abfrage einstellen. Fehlt noch das i-Tüpfelchen: Zeigen Haupt- und Unterformular Daten aus zwei per 1:n-Beziehung verknüpften Tabellen an, können Sie diese ganz einfach synchronisieren. Dazu stellen Sie einfach die beiden Eigenschaften *Verknüpfen von* und *Verknüpfen nach* auf das Fremdschlüsselfeld im Unterformular sowie das Primärschlüsselfeld im Hauptformular ein.

Wenn Ihr Datenbank-Frontend seine Daten aus den Tabellen einer SQL Server-Datenbank beziehen soll, können Sie dies grundsätzlich ebenso handhaben. Sie benötigen nur entsprechende ODBC-Verknüpfungen zu den Tabellen und tragen diese dann als Datenherkunft oder Datensatzherkunft von Formularen und Steuerelementen ein. Auf diese Weise sind die Daten dann sogar wie bei einer reinen Access-Datenbank aktualisierbar (außer natürlich, Sie verwenden eine nicht aktualisierbare Abfrage als Datenherkunft).

Allerdings ist Access beim Einlesen von Daten aus eigenen oder verknüpften Tabellen aus anderen Access-Datenbanken wenig wählerisch: Es holt sich erst mal alle Daten und filtert oder sortiert diese dann auf dem Client-Rechner nach den angegebenen Kriterien.

Beziehen Sie die Daten vom SQL Server, werden ebenfalls die Daten an den Client-Rechner übertragen. Der Umfang ist abhängig von der verwendeten Datenherkunft. Bei einer per ODBC eingebundenen Tabelle sind dies alle in der Ausgabe angezeigten Daten zuzüglich weiterer Datensätze, um auf ein eventuelles Blättern in den Daten vorbereitet zu sein. Bei einer Access-Abfrage ist es abhängig vom Access-Abfrageoptimierer. Dieser optimiert die Access-Abfrage für den Zugriff auf eine ODBC-Datenbank, was wiederum drei Möglichkeiten als Ergebnis haben kann:

» Die SQL-Anweisung der Access-Abfrage wird 1:1 an den SQL Server übergeben. Dort werden die Daten ermittelt und das Ergebnis an den Client-Rechner übertragen.

» Der Access-Abfrageoptimierer erstellt aus der SQL-Anweisung der Access-Abfrage mehrere SQL-Anweisungen, die an den SQL Server übergeben werden. Dort werden die Daten anhand dieser SQL-Anweisungen ermittelt und die Ergebnisse an den Client-Rechner übertragen. In Access werden dann die Ergebnisse miteinander verknüpft und gegebenenfalls gefiltert, gruppiert und sortiert.

» Der Access-Abfrageoptimierer fordert beim SQL Server alle Daten der in der Abfrage enthaltenen Tabellen beim SQL Server an. Dieser überträgt die Daten an den Client-Rechner, wo dann die eigentliche Abfrage ausgeführt wird.

Für die Ermittlung der Daten sperrt der SQL Server nicht nur die einzelnen Datensätze, sondern immer die gesamte Datenseite, auf der sich die Datensätze befinden. Dabei handelt es sich um eine Lesesperre, die andere Lesevorgänge erlaubt, aber keine Schreibzugriffe zulässt. Die Datenseiten sind zwar nur zur Ermittlung der Daten und somit sehr kurz gesperrt, aber Access aktualisiert den Lesevorgang automatisch alle 1.500 Sekunden — in älteren Versionen gar alle 60 Sekunden. Dies betrifft alle in einem Formular verwendeten Datenquellen – die Datenherkunft des Formulars wie auch die Datensatzherkunft der darin enthaltenen Kombinations- und Listenfeldern. In einem Mehrbenutzersystem kann dies die Performance erheblich beeinträchtigen. Mehr zum Thema Datenzugriff und den dabei gesetzten Sperren lesen Sie im Kapitel »Performance analysieren«, Seite 105. Dieses Kapitel wird Ihnen einige Techniken zeigen, wie Sie möglichst performant, bequem und dennoch ohne unnötige Sperren auf die Daten der Tabellen des SQL Servers zugreifen.

14.2 Formulardaten in reinen Access-Datenbanken

Möchten Sie ein Formular einer Access-Datenbank gleich beim Öffnen an eine Datenherkunft binden, die ihre Daten allein aus Access-Tabellen bezieht, haben Sie folgende Möglichkeiten:

» Füllen der Eigenschaft Datenherkunft mit dem Namen einer Tabelle (*tblKunden*),

» mit dem Namen einer Abfrage (*qryKunden*) oder

» mit einem SQL-Ausdruck (*SELECT * FROM tblKunden*).

In der Abfrage und im SQL-Ausdruck können Sie außerdem noch Filterkriterien unterbringen.

Sollten Sie keine Filterkriterien oder Informationen zu dem zuerst anzuzeigenden Datensatz angegeben haben, können Sie diese per VBA nachträglich festlegen – zum Beispiel auf die folgenden Arten:

» durch Zuweisen des Kriteriums an die Eigenschaft *Filter* und Aktivieren der Eigenschaft *FilterOn* mit dem Wert *True* (beispielsweise in den Ereignisprozeduren, die durch die Ereignisse *Beim Öffnen*, *Beim Laden* oder *Beim Anzeigen* ausgelöst werden) oder

» durch Einstellen der *FindFirst*-Methode des *Recordset*-Objekts des Formulars.

Gelegentlich weist man einem Formular auch erst beim Öffnen oder zu verschiedenen anderen Gelegenheiten seine Datenherkunft zu. Wenn Sie etwa ein Suchformular verwenden, das im Formularkopf die Steuerelemente zum Eingeben der Suchkriterien enthält, wird der Inhalt des Unterformulars zur Anzeige der Ergebnisse beim Starten der Suche aktualisiert.

Die Bindung des Formulars und seiner Steuerelemente an die Datenherkunft sorgt dafür, dass Sie die Daten in der Regel nicht nur anzeigen, sondern auch bearbeiten können. Formulare greifen aber auch noch auf andere Weise auf die Daten von Tabellen zu: nämlich zur Bereitstellung der mit Kombinations- und Listenfeldern auszuwählenden Daten. Dazu legen Sie meist gleich beim Entwurf des Formulars eine Tabelle, eine Abfrage oder einen SQL-Ausdruck fest, der die Daten für das Kombinations- oder Listenfeld liefert.

Gerade Kombinationsfelder sollen oft erst im weiteren Verlauf mit Daten gefüllt werden – beispielsweise, wenn ein Kombinationsfeld Daten in Abhängigkeit vom Wert eines weiteren Kombinationsfeldes anzeigen soll. Auch bei Listenfeldern kann dies geschehen: etwa wenn zwei Listenfelder die zugeordneten und die nicht zugeordneten Daten einer m:n-Beziehung abbilden (etwa zwischen Fahrzeugen und Ausstattungsmerkmalen). Hier ist die Datensatzherkunft der beiden Listenfelder nach jeder Änderung neu zu füllen, was mithilfe entsprechender Abfragen geschieht.

Eine weitere Besonderheit ist der einfache Umgang unter Access mit Haupt- und Unterformularen. Wenn das Unterformular Daten aus einer Tabelle enthält, die per Fremdschlüsselfeld mit dem Primärschlüsselfeld der Datenherkunft des Hauptformulars verknüpft ist, erkennt Access dies beim Einfügen eines Unterformulars mit entsprechender Datenherkunft automatisch. Es füllt dann die beiden Eigenschaften *Verknüpfen von* und *Verknüpfen nach* des Unterformular-Steuerelements automatisch mit dem Namen des Fremdschlüsselfeldes und des Primärschlüsselfeldes der Beziehung.

Bei Zugriff auf den SQL Server verwenden Sie eine Reihe anderer Techniken, die Sie in den folgenden Abschnitten vorfinden.

14.3 Formulardaten aus SQL Server-Tabellen

Die wichtigste Überlegung beim Erstellen oder Migrieren einer Access-Datenbank mit einem SQL Server-Backend ist diese: Der Wechsel zum SQL Server geschieht in der Regel, weil die reine Access-Lösung die Anzahl der Zugriffe oder Benutzer nicht mehr verarbeiten konnte. Das heißt, dass zu erwarten ist, dass mehrere Benutzer auf die Daten zugreifen und gegebenenfalls auch einmal die gleichen Daten betrachten oder bearbeiten.

Was ist beim Betrachten und Bearbeiten zu beachten? Beim Betrachten brauchen Sie nur dafür zu sorgen, dass der Benutzer jeweils die aktuelle Version der Daten zu Gesicht bekommt. Dabei sollten die anzuzeigenden Daten direkt vom SQL Server ermittelt und an das Access-Frontend übertragen werden. Da die Daten eh nur angezeigt werden, bietet sich hierfür eine Pass-Through-

Abfrage mit einer gespeicherten Prozedur an. Die Daten stehen dann im Gegensatz zu einer eingebundenen Tabelle oder einer Access-Abfrage lediglich schreibgeschützt zur Verfügung, dafür aber entfällt das automatische Aktualisieren der Daten und somit das wiederholte Setzen von Lesesperren.

Interessant wird es beim Bearbeiten von Daten. Auch diese Daten müssen zunächst angezeigt werden, sollen dann aber änderbar sein. Eine Pass-Through-Abfrage als Datenherkunft kann hierfür nicht so einfach verwendet werden, da diese die Daten schreibgeschützt liefert. Da ist es doch naheliegend, als Datenherkunft eine eingebundene Tabelle zu verwenden. Doch diese Datenherkunft wird alle 1.500 Sekunden aktualisiert, was wiederum Lesesperren auf dem SQL Server auslöst.

Sie sehen, es geht auch hierbei um die eingangs erwähnten Lesesperren. Die Sperre für den eigentlichen Änderungsvorgang ist an dieser Stelle marginal. Der entsprechende Datensatz erhält vor der Änderung eine exklusive Sperre und wird dann geändert. Aber gerade das Setzen dieser exklusiven Sperre ist nur möglich, wenn der Datensatz frei von anderen Sperren ist — und dies gilt auch für Lesesperren. Um unnötige Lesesperren zu vermeiden und dennoch die Daten ändern zu können, gibt es verschiedene Strategien, die wir wie folgt abstufen möchten:

» Das Formular zur Bearbeitung ist an eine mit ODBC verknüpfte Tabelle gebunden. Die Daten können genau wie bei einer lokalen Tabelle bearbeitet und gespeichert werden. Alle angezeigten Daten (auch die der Kombinations- und Listenfelder) werden alle 1.500 Sekunden aktualisiert, was wiederum entsprechende Lesesperren auf den Datenseiten im SQL Server auslöst, auf denen die angezeigten Datensätze gespeichert sind.

» Das Formular ist zum Betrachten der Daten an eine Pass-Through-Abfrage mit einer gespeicherten Prozedur gebunden, die ein Recordset mit den Daten aus den Tabellen des SQL Servers liefert. Erst wenn der Benutzer mit der Bearbeitung des Datensatzes beginnt, wird nur der betroffene Datensatz in den Bearbeitungsmodus versetzt. Dazu ersetzt das Formular die Pass-Through-Abfrage als Datenherkunft durch eine SQL-Anweisung auf eine entsprechende, per ODBC verknüpfte Tabelle, die exakt nur den zu ändernden Datensatz liefert. Der Datensatz kann dann wie unter Access gewohnt bearbeitet und gespeichert werden. Zwar wird auch bei dieser Variante eine Lesesperre im SQL Server gesetzt, hier aber nur bei der Datenseite, die den angezeigten Datensatz enthält. Dies gilt auch für die automatische Aktualisierung von Access in dem angegebenen Turnus. Dabei wird immer nur die betroffene Datenseite gesperrt.

» Das Formular verwendet ebenfalls eine Pass-Through-Abfrage mit einer gespeicherten Prozedur als Datenlieferant. Möchte der Benutzer einen Datensatz bearbeiten, löst das Formular jedoch die Bindung an die Datenherkunft. Auch die einzelnen Steuerelemente werden von der Bindung an die einzelnen Felder der Datenherkunft befreit. Dafür schreibt das Formular jedoch die im zu bearbeitenden Datensatz enthaltenen Daten in die Steuerelemente des Formulars. Durch die fehlende Bindung an den Datensatz kann der Benutzer diesen nun bearbeiten, ohne das im SQL Server entsprechende Lesesperren ausgelöst werden. Beim

Speichern des Datensatzes liest das Formular die Werte der einzelnen Felder ein und übergibt diese an eine gespeicherte Prozedur, welche den betroffenen Datensatz direkt auf dem SQL Server ändert.

Wie dies nun gelingt, lässt sich am einfachsten an drei entsprechenden Beispielen erläutern.

Hinweis zu den Access-SQL Server-Tools

In Kapitel *»Access-SQL Server-Tools«, Seite 459* stellen wir einige Tools etwa zum Verwalten von Verbindungszeichenfolgen, zum Verknüpfen mit Tabellen et cetera vor. Um diese wie im vorliegenden Kapitel zu nutzen, kopieren Sie einfach alle Objekte der Beispieldatenbank *17_AccessSQLTools* in die Datenbank, mit der Sie die Beispiele ausprobieren möchten.

14.3.1 Formular mit ODBC-Tabelle als Datenherkunft

Für das erste Beispiel wählen wir eine einfache Tabelle etwa zum Speichern von Warengruppen. Diese Tabelle heißt *tblWarengruppen* und enthält die Spalten *WarengruppeID* und *Warengruppe* sowie eine *timestamp*-Spalte.

Für das erste Beispiel benötigen Sie eine Verknüpfung zu dieser auf dem SQL Server liegenden Tabelle als ODBC-Verknüpfung (siehe Abbildung 14.1), die Sie über das Formular *frmTabellen-Verknuepfen* hinzufügen können. Für die folgenden Beispiele ändern Sie den automatisch generierten Namen der Tabelle von *dbo_tblWarengruppen* in *tblWarengruppen*.

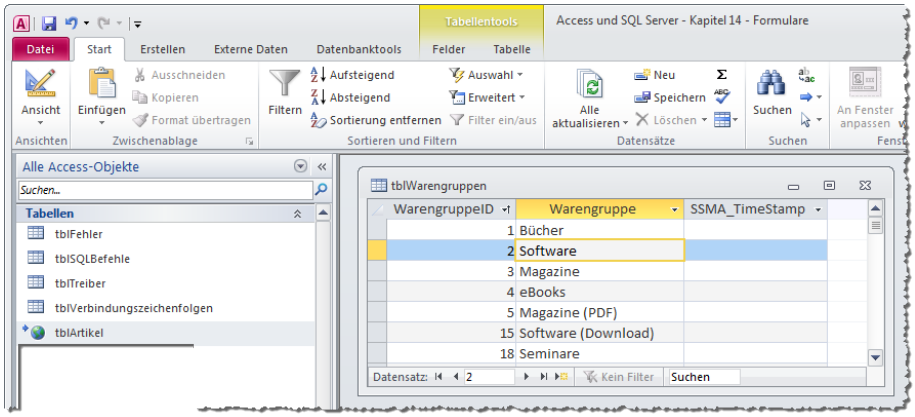


Abbildung 14.1: Die per ODBC verknüpfte Tabelle *tblWarengruppen*

Nun erstellen Sie ein Formular, in dem Sie die Tabelle *tblWarengruppen* als Datenherkunft verwenden (siehe Formular *frmWarengruppenODBC* in der Beispieldatenbank *14_Formulare.accdb*). Ziehen Sie dann die zwei Felder *WarengruppeID* und *Warengruppe* aus der Feldliste in den Entwurf des Formulars. Die Felder sind nun auf herkömmliche Weise an die Felder der Datenherkunft gebunden (siehe Abbildung 14.2).

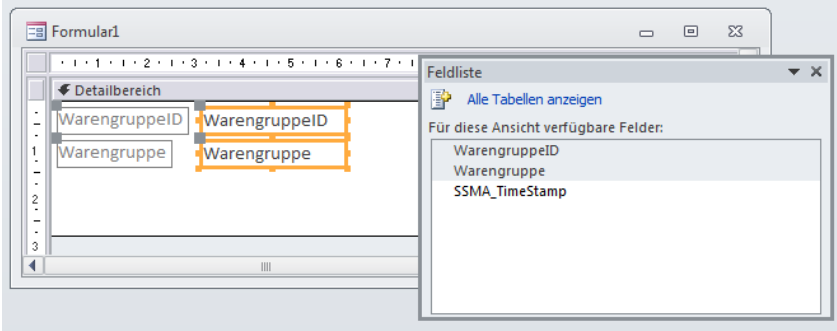


Abbildung 14.2: Formular mit einfacher Verknüpfung zur eingebundenen ODBC-Tabelle

Wenn Sie nun in die Formularansicht wechseln, zeigt das Formular die Daten der Tabelle *tblWarengruppen* an und erlaubt das Blättern zwischen den Datensätzen und auch das Bearbeiten wie beim Zugriff auf eine lokale Tabelle. Nachteile, wie bereits erwähnt: Der aktuell angezeigte Datensatz und eventuell auf der gleichen Datenseite befindliche Datensätze werden alle 1.500 Sekunden neu ermittelt, wodurch entsprechende Lesesperren gesetzt werden.

Achtung! Wenn Sie in einem Formular auf eine per ODBC eingebundene Tabelle zugreifen, sollten Sie immer dafür Sorge tragen, dass die *timestamp*-Spalte der Tabelle in der Datenherkunft enthalten ist. Eine Datenherkunft ohne *timestamp*-Spalte verschlechtert die Performance bei Datenänderungen. Dabei ist es nicht notwendig, die *timestamp*-Spalte im Formular anzuzeigen. Mehr dazu lesen Sie im Abschnitt »Timestamp-Spalte – ja oder nein?«, Seite 69.

14.3.2 Formular mit gespeicherter Prozedur und ODBC-Tabelle zum Bearbeiten

Im zweiten Beispiel soll das Formular zunächst mit den Daten einer gespeicherten Prozedur gefüllt werden. Diese Prozedur erstellen Sie im *SQL Server Management Studio* wie folgt:

```
CREATE PROCEDURE dbo.spSelectAlleWarengruppen
AS
SET NOCOUNT ON;
SELECT WarengruppeID, Warengruppe FROM dbo.tblWarengruppen ORDER BY WarengruppeID;
```

In der Access-Datenbank erstellen Sie eine neue Abfrage als Pass-Through-Abfrage, welche die soeben erstellte gespeicherte Prozedur mit der *EXEC*-Methode ausführt und das Ergebnis zurückliefert (siehe Abbildung 14.3). Diese Abfrage speichern Sie unter dem Namen *qrySPSELECTAlleWarengruppen*. Als Formular können Sie das Formular des vorherigen Beispiels kopieren und unter dem Namen *frmWarengruppenSPODBC* speichern. Es sind kaum Änderungen nötig – Sie müssen lediglich die Datenherkunft auf den Namen der Pass-Through-Abfrage *qrySPSELECTAlleWarengruppen* einstellen.

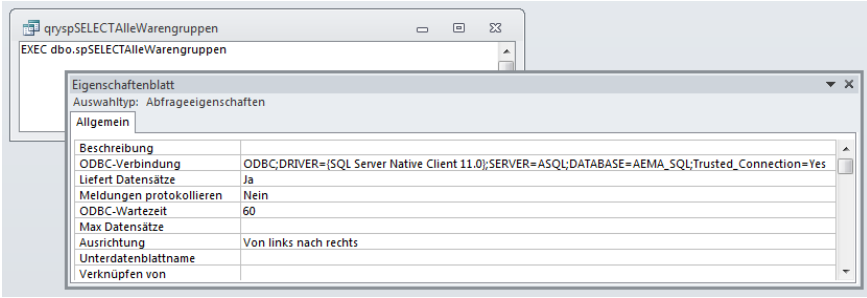


Abbildung 14.3: ODBC-Abfrage mit gespeicherter Prozedur

Wenn Sie nun in die Formularansicht wechseln, sieht dieses zunächst genauso aus wie das Formular mit der per ODBC verknüpften Tabelle als Datenherkunft. Allerdings können Sie weder die angezeigten Daten bearbeiten noch Datensätze löschen oder hinzufügen – siehe Statuszeile in Abbildung 14.4.

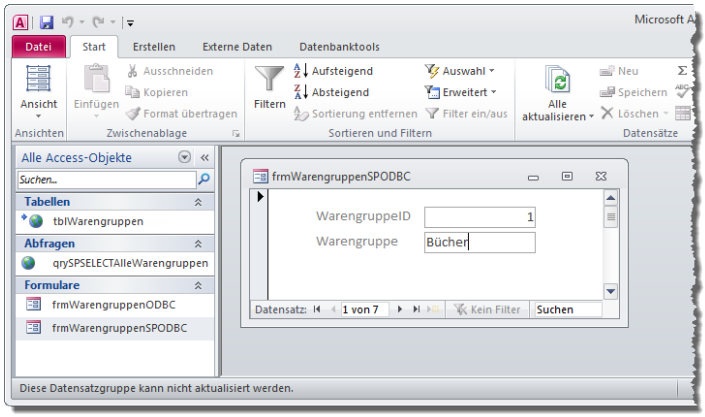


Abbildung 14.4: Dieses Formular ist über eine Pass-Through-Abfrage an eine gespeicherte Prozedur gebunden.

Alternativ können Sie, wie im Beispielformular *frmWarengruppenSPODBC* gezeigt, die Datenherkunft beim Laden des Formulars dynamisch zuweisen. Dazu legen Sie eine Ereignisprozedur an, die durch das Ereignis *Beim Laden* ausgelöst wird und die wie folgt aussieht:

```
Private Sub Form_Load()
    Me.RecordSource = SPRecordsource("dbo.spSELECTAlleWarengruppen", _
        Standardverbindungszeichenfolge)
End Sub
```

Die Prozedur verwendet die Funktion *SPRecordsource* aus dem Modul *mdlToolsSQLServer* (siehe Abschnitt *»Recordsource aus gespeicherter Prozedur ohne Parameter«, Seite 338*), um eine

neue Pass-Through-Abfrage auf Basis der gespeicherten Prozedur *spSELECTAlleWarengruppen* zu erstellen. Dabei wird die durch die Funktion *Standardverbindungszeichenfolge* ermittelte Verbindungszeichenfolge übergeben. Die Funktion legt die Pass-Through-Abfrage an und liefert den Namen der angelegten Abfrage zurück. Diese weist die Ereignisprozedur dann direkt der Eigenschaft *RecordSource* (also Datenherkunft) des Formulars zu. Dies bringt zum Beispiel den Vorteil mit sich, dass Sie die zu verwendende Verbindung jederzeit ändern können, ohne jedes Mal die Pass-Through-Abfragen anpassen zu müssen. Den Wert der Eigenschaft *Datenherkunft* können Sie in diesem Fall löschen.

Noch besser ist allerdings die folgende Variante, bei der wir die Funktion *SPRecordset* verwenden, um dem Formular direkt ein Recordset mit den anzuzeigenden Daten zuzuweisen:

```
Private Sub Form_Load()
    Set Me.Recordset = SPRecordset("dbo.spSELECTAlleWarengruppen", _
        Standardverbindungszeichenfolge)
End Sub
```

Der Vorteil ist, dass hier noch nicht einmal mehr eine Abfrage in der Datenbank gespeichert werden muss.

14.3.3 Anlegen, bearbeiten und löschen

Wie aber wollen wir nun dem Benutzer die Möglichkeit bieten, einen neuen Datensatz anzulegen oder einen vorhandenen Datensatz zu bearbeiten oder zu löschen? Dazu können wir geeignete Schaltflächen anlegen – mit den Beschriftungen *Neu*, *Löschen*, *Bearbeiten* und schließlich auch noch *Speichern*. Das Formular sieht dann wie in Abbildung 14.5 aus.

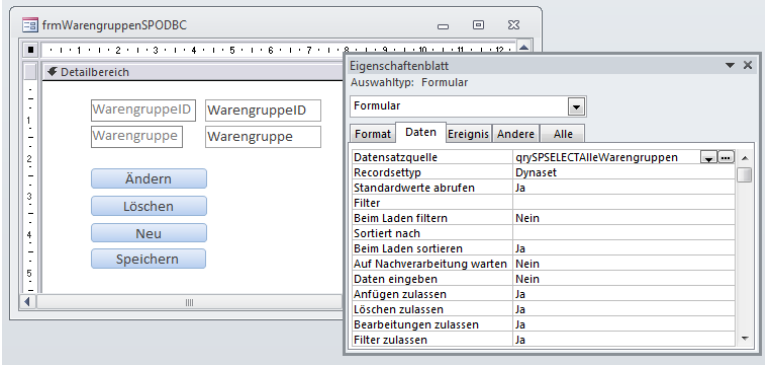


Abbildung 14.5: Das Formular *frmWarengruppenSPODBC*

Was geschieht, wenn der Benutzer auf die verschiedenen Schaltflächen klickt?

» *Neu*: Das Formular erhält als Datenherkunft eine *SELECT*-Abfrage an die per ODBC-verknüpfte Tabelle *tblWarengruppen*, die jedoch keinen Datensatz liefert.

» *Bearbeiten*: Das Formular wird per *SELECT*-Anweisung an die ODBC-verknüpfte Tabelle *tblWarengruppen* gebunden, wobei diese nur den zu bearbeitenden Datensatz liefert.

» *Löschen*: Der aktuelle Datensatz wird per gespeicherter Prozedur gelöscht, die Datenherkunft des Formulars aktualisiert und der Datensatzzeiger auf den Datensatz verschoben, der auch beim normalen Löschen in einer gebundenen Tabelle markiert würde.

» *Speichern*: Nach dem Erstellen eines neuen oder dem Bearbeiten eines existierenden Datensatzes muss der Benutzer dem Formular mitteilen, dass der Datensatz gespeichert werden soll. Das Formular merkt sich den Primärschlüsselwert des erzeugten beziehungsweise bearbeiteten Datensatzes, tauscht die SQL-Anweisung gegen die Pass-Through-Abfrage mit der gespeicherten Prozedur als Datenherkunft aus und verschiebt den Datensatzzeiger auf den erzeugten beziehungsweise bearbeiteten Datensatz.

Anlegen eines neuen Datensatzes

Zum Anlegen eines neuen Datensatzes soll der Benutzer auf die Schaltfläche *cmdNeu* klicken. Dies löst die folgende Prozedur aus, die zunächst die Datenherkunft auf eine neue Abfrage einstellt. Diese liefert keinen Datensatz der Tabelle *tblWarengruppen* – dafür sorgt das Kriterium *1=2*. Da diese per ODBC verknüpfte Tabelle aktualisierbar ist, zeigt das Formular direkt einen neuen, leeren Datensatz an (siehe Abbildung 14.6):

```
Private Sub cmdNeu_Click()
    Me.RecordSource = "SELECT * FROM tblWarengruppen WHERE 1=2"
    Me!cmdLoeschen.Enabled = False
End Sub
```

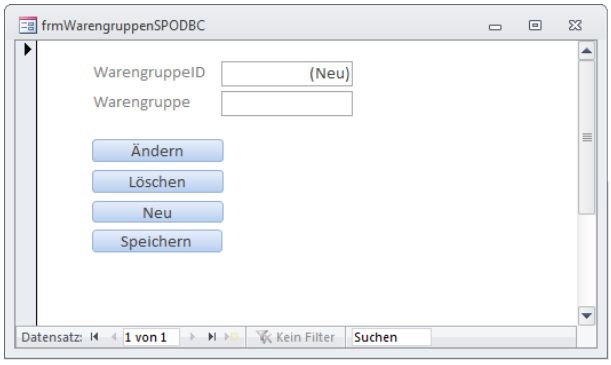


Abbildung 14.6: Anlegen eines neuen Datensatzes per ODBC-verknüpfter Tabelle

Außerdem wird die Schaltfläche *cmdLoeschen* deaktiviert, da das Löschen eines Datensatzes, für den noch kein Primärschlüsselwert erzeugt wurde, nicht möglich ist.

Wenn der Benutzer nun einen neuen Datensatz anlegt und dann zum nächsten neuen Datensatz wechselt, wird dieser gespeichert. Dies wäre der Zeitpunkt, an dem die gespeicherte Prozedur

wieder als Datenherkunft eingestellt werden sollte, damit wieder alle Datensätze verfügbar sind. Praktisch wäre es, wenn der Datensatzzeiger direkt beim neu angelegten Datensatz landet.

Dazu sind ein paar Zeilen Code nötig. Durch das Speichern des neuen Datensatzes werden einige Ereignisse ausgelöst, beispielsweise *Vor Aktualisierung* und *Nach Aktualisierung*. Im ersten dieser beiden Ereignisse ist der vom SQL Server vergebene Primärschlüsselwert noch nicht bekannt, daher können wir die gewünschten Aktionen erst in der Prozedur durchführen, die durch das Ereignis *Nach Aktualisierung* des Formulars ausgelöst wird. Diese füllen Sie wie folgt:

```
Private Sub Form_AfterUpdate()
    Dim lngWarengruppeID As Long
    lngWarengruppeID = Me!WarengruppeID
    Me.Painting = False
    Set Me.Recordset = SPRecordset("dbo.spSELECTAlleWarengruppen", _
        Standardverbindungszeichenfolge)
    Me.Recordset.FindFirst "WarengruppeID = " & lngWarengruppeID
    Me!cmdLoeschen.Enabled = True
    Me.Painting = True
End Sub
```

Die Prozedur speichert zunächst den Primärschlüsselwert des neuen Datensatzes. Dann stellt Sie die Bildschirmaktualisierung aus, um unerwünschtes Flackern zu vermeiden. Das Formular bekommt wieder das Ergebnis der Funktion *SPRecordset* als Recordset zugewiesen. Der Datensatzzeiger wird auf den Datensatz eingestellt, dessen Feld *WarengruppeID* dem neuen Primärschlüsselwert entspricht. Die beim Anlegen des Datensatzes gesperrte Schaltfläche *cmdLoeschen* wird wieder aktiviert.

Schließlich wird die Bildschirmaktualisierung wieder aktiviert, wodurch das Formular den neuen Datensatz anzeigt – diesmal allerdings nicht als Datensatz der per ODBC verknüpften Tabelle *tblWarengruppen*, sondern der gespeicherten Prozedur *spSELECTAlleWarengruppen*. Dadurch kann der Benutzer nun auch wieder durch alle Datensätze dieser Tabelle navigieren.

Ändern eines Datensatzes

Das Ändern eines Datensatzes geschieht fast auf identische Weise. Der einzige Unterschied ist, dass die durch den Klick auf die Schaltfläche *cmdAendern* ausgelöste Prozedur keine leere Datenherkunft einstellt, sondern eine mit dem zu ändernden Datensatz – und auch hier wird die Schaltfläche *cmdLoeschen* deaktiviert:

```
Private Sub cmdAendern_Click()
    Me.RecordSource = "SELECT * FROM tblWarengruppen WHERE WarengruppeID = " _
        & Me!WarengruppeID
    Me!cmdLoeschen.Enabled = False
End Sub
```

Falls das Formular nur bestimmte Spalten des Datensatzes anbieten soll, sollte die *SELECT*-Anweisung auch nur diese Spalten plus der *timestamp*-Spalte der Tabelle enthalten. Der Benutzer kann den zu ändernden Datensatz nun bearbeiten und mit den von Access bekannten

Mitteln speichern. Dies löst wiederum das Ereignis *Nach Aktualisierung* und die weiter oben beschriebene Ereignisprozedur aus.

Änderung speichern

Wenn Sie eine Schaltfläche etwa namens *cmdSpeichern* verwenden möchten, um das Speichern des Datensatzes zu ermöglichen, legen Sie dafür die folgende Ereignisprozedur an. Das Einstellen von *Me.Dirty* auf den Wert *False* speichert bereits durchgeführte Änderungen in der Datenherkunft:

```
Private Sub cmdSpeichern_Click()
    Me.Dirty = False
End Sub
```

Löschen eines Datensatzes

Wenn wir an dieser Stelle das begonnene Paradigma fortsetzen wollten, müssten wir das Formular wieder an die ODBC-verknüpfte Tabelle binden, den aktuellen Datensatz aufrufen und diesen dann mit der entsprechenden Methode löschen.

Diesen Umweg gehen wir an dieser Stelle nicht. Stattdessen führen wir eine gespeicherte Prozedur aus, die das Formular über eine Pass-Through-Abfrage aufruft. Die gespeicherte Prozedur legen Sie wie folgt im SQL Server an:

```
CREATE PROCEDURE dbo.spDELETEWarengruppeNachID @WarengruppeID int
AS
SET NOCOUNT ON;
DELETE FROM dbo.tblWarengruppen WHERE WarengruppeID = @WarengruppeID;
```

Die gespeicherte Prozedur *spDELETEWarengruppeNachID* verwendet einen Parameter namens *@WarengruppeID*. Um die gespeicherte Prozedur mit dem Parameter zu bestücken und auszuführen, erstellen Sie folgende Routine, die durch das Ereignis *Beim Klicken* der Schaltfläche *cmdLoeschen* ausgelöst wird:

```
Private Sub cmdLoeschen_Click()
    Dim db As DAO.Database
    Dim qdf As DAO.QueryDef
    Set db = CurrentDb
    Set qdf = db.CreateQueryDef("")
    With qdf
        .Connect = "ODBC;DRIVER={SQL Server};SERVER=ASQL;DATABASE=AEMA_SQL;
                                                        Trusted_Connection=Yes"
        .ReturnsRecords = False
        .SQL = "EXEC dbo.spDELETEWarengruppeNachID " & Me!WarengruppeID
        .Execute
    End With
End Sub
```

Dies löscht zuverlässig den aktuellen Datensatz, aktualisiert allerdings noch nicht die Daten des Formulars – mehr dazu gleich. Wer das Kapitel »SQL Server-Zugriff per VBA«, Seite 307 gele-

sen hat, weiß auch, dass die hier verwendeten Anweisungen dort schon in ähnlicher Form in parametrisierte Prozeduren beziehungsweise Funktionen ausgelagert wurden. Wenn Sie dem Benutzer den Erfolg des Löschvorgangs bestätigen und das Formular aktualisieren möchten, verwenden Sie einfach die folgende Variante der Prozedur *cmdLoeschen_Click*:

```
Private Sub cmdLoeschen_Click()
    Dim lngAnzahl As Long
    Dim lngAbsolutePosition As Long
    lngAnzahl = SPAktionsabfrageMitErgebnis("dbo.spDELETEWarengruppeNachID", _
        Standardverbindungszeichenfolge, False, "", Me!WarengruppeID)
    MsgBox "Anzahl gelöschter Datensätze: " & lngAnzahl
    lngAbsolutePosition = Me.Recordset.AbsolutePosition
    Set Me.Recordset = SPRecordset("dbo.spSELECTAlleWarengruppen", _
        Standardverbindungszeichenfolge)
    Me.Recordset.AbsolutePosition = lngAbsolutePosition
End Sub
```

Die Prozedur nutzt die Funktion *SPAktionsabfrageMitErgebnis*, um die gespeicherte Prozedur aufzurufen und das Ergebnis in der Variablen *lngAnzahl* zu speichern. Außerdem aktualisiert sie die Datenherkunft des Formulars, indem sie die Funktion *SPRecordset* erneut aufruft und das Ergebnis der Recordset-Eigenschaft des Formulars zuweist. Dabei merkt sie sich mit der Variablen *lngAbsolutePosition* vorher die aktuelle Position des Datensatzzeigers und stellt diese nach dem Aktualisieren wieder her.

Wenn Sie die Funktion *SPAktionsabfrageMitErgebnis* einsetzen, wird der Code im Klassenmodul des Formulars wesentlich überschaubarer. Außerdem werden Sie ähnlichen Code sehr oft verwenden und erhalten somit eine wesentlich besser wartbare Anwendung. Abbildung 14.7 zeigt, wie sich der Code auswirkt.

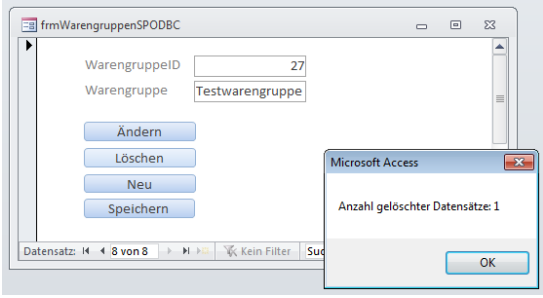


Abbildung 14.7: Meldung eines erfolgreichen Löschvorgangs

14.3.4 Formular mit gespeicherter Prozedur und ungebundener Bearbeitung

Einen Schritt weiter geht die Variante, bei der das Formular zum Bearbeiten komplett von der Datenherkunft entbunden wird. Die folgenden Abschnitte zeigen, wie dies funktioniert, die

Beispiele finden Sie im Formular *frmWarengruppenSPUngebunden* der Beispieldatenbank. Das Formular ist über die Pass-Through-Abfrage *qrySPSELECTAlleWarengruppen* an die gespeicherte Prozedur *spSELECTAlleWarengruppen* gebunden. Diese Bindung erreichen Sie wiederum entweder durch direkte Zuweisung der Pass-Through-Abfrage an die Eigenschaft *Datenherkunft* oder durch die folgende Prozedur:

```
Private Sub Form_Load()
    Set Me.Recordset = SPRecordset("dbo.spSELECTAlleWarengruppen", _
       Standardverbindungszeichenfolge)
End Sub
```

Ungebundenes Löschen eines Datensatzes

Die einfachste Aktion ist das Löschen eines Datensatzes. Dies gelingt nämlich genau wie im vorherigen Beispiel – Änderungen an den Datenfeldern sind ja nicht nötig. Wir kommen jedoch später noch im Rahmen einer Variante auf dieses Thema zurück (siehe Abschnitt *»Formular mit der Merge-Anweisung«, Seite 361*)

Ungebundenes Ändern eines Datensatzes

Interessanter wird es, wenn wir einen vorhandenen Datensatz ändern möchten, ohne das Formular an eine per ODBC eingebundene Tabelle zu binden. Im Detail heben wir die Bindung sogar komplett auf – der Benutzer kann somit die Daten einfach in das Formular eingeben. Das Aufheben der Bindung erfolgt in zwei Schritten:

» Entfernen der Bindung der Steuerelemente

» Entfernen der Bindung des Formulars

Die beiden Aktionen stößt wiederum ein Klick auf die Schaltfläche *cmdAendern* an, die folgende Ereignisprozedur auslöst:

```
Private Sub cmdAendern_Click()
    SteuerelementeEntbinden False
    Me.RecordSource = ""
    Me!cmdLoeschen.Enabled = False
End Sub
```

Die erste Anweisung ruft eine Prozedur namens *SteuerelementeEntbinden* auf, welche die Eigenschaft *Steuerelementinhalt* der Steuerelemente entleert. Die zweite stellt schließlich die Eigenschaft *Datenherkunft* des Formulars auf eine leere Zeichenkette ein. Damit zeigt das Formular nunmehr nur noch einfache Textfelder und sonstige Steuerelemente an.

Steuerelemente entbinden

Die folgende Version der Prozedur *SteuerelementeEntbinden* kümmert sich lediglich um Steuerelemente des Typs *Textbox*. Sie soll jedoch je nach der anstehenden Aktion, also Anlegen eines neuen Datensatzes oder Bearbeiten eines bestehenden Datensatzes, leicht unterschied-

lich agieren. In jedem Fall durchläuft sie alle Steuerelemente des Formulars, die Access über die Auflistung *Controls* des Formulars verfügbar macht. In der *For Each*-Schleife wird jedes Steuerelement zunächst mit der Objektvariablen *ctl* referenziert. Danach schreibt die Prozedur den Wert der Eigenschaft *ControlSource* des aktuellen Steuerelements in die Variable *strControlSource*. Dies kann einen Fehler auslösen, wenn das Steuerelement nicht an ein Feld gebunden werden kann und die Eigenschaft *ControlSource* gar nicht aufweist. Das ist aber Teil des Plans: Wir fassen die Zuweisung in die beiden Anweisungen *On Error Resume Next* und *On Error Goto 0* ein, um die Fehlerbehandlung kurzzeitig auszuschalten. Auf diese Weise wird *strControlSource* nur gefüllt, wenn es sich um ein gebundenes Steuerelement handelt.

Eine *If...Then*-Bedingung prüft dann, ob *strControlSource* einen Wert enthält und führt nur in diesem Fall die folgenden Schritte aus. Diese bestehen darin, den Wert der Eigenschaft *Controlsource* in der Eigenschaft *Tag* zwischenzuspeichern. Die Variable *varValue* nimmt derweil den Wert des an das Steuerelement gebundenen Feldes auf. Warum? Damit wir anschließend das Steuerelement durch Einstellen von *ControlSource* auf eine leere Zeichenkette „entbinden" und anschließend das ungebundene Steuerelement mit dem entsprechenden Wert füllen können.

So machen wir aus dem gebundenen Steuerelement, das den Wert des betroffenen Feldes anzeigt, ein ungebundenes Steuerelement, dass den Wert dieses Feldes enthält. Das Füllen des jeweiligen Steuerelements soll jedoch nur erfolgen, wenn der Parameter *bolNeu* den Wert *False* enthält, also ein vorhandener Datensatz bearbeitet wird. Anderenfalls werden die vormals gebundenen Steuerelemente einfach geleert:

```
Private Sub SteuerelementeEntbinden(bolNeu As Boolean)
    Dim ctl As Control
    Dim strControlSource As String
    Dim varValue As Variant
    For Each ctl In Me.Controls
        strControlSource = ""
        On Error Resume Next
        strControlSource = ctl.ControlSource
        On Error GoTo 0
        If Len(strControlSource) > 0 Then
            ctl.Tag = ctl.ControlSource
            varValue = ctl.Value
            ctl.ControlSource = ""
            If Not bolNeu Then
                ctl.Value = varValue
            Else
                ctl.Value = Null
            End If
        End If
    Next ctl
End Sub
```

Nun kann der Benutzer die Daten im Formular anpassen. Danach müssen die geänderten Daten gespeichert werden. Dies erledigt der Benutzer mit einem Klick auf die Schaltfläche *cmdSpeichern*, was folgende Ereignisprozedur auslöst:

```
Private Sub cmdSpeichern_Click()
    Speichern
    SteuerelementeBinden
End Sub
```

Diese Prozedur ruft wiederum zwei weitere Routinen auf. Die erste kümmert sich um den eigentlichen Speichervorgang, die zweite bindet das Formular und die Steuerelemente wieder an den zuvor geänderten Datensatz. Schauen wir uns zunächst die Prozedur *Speichern* an:

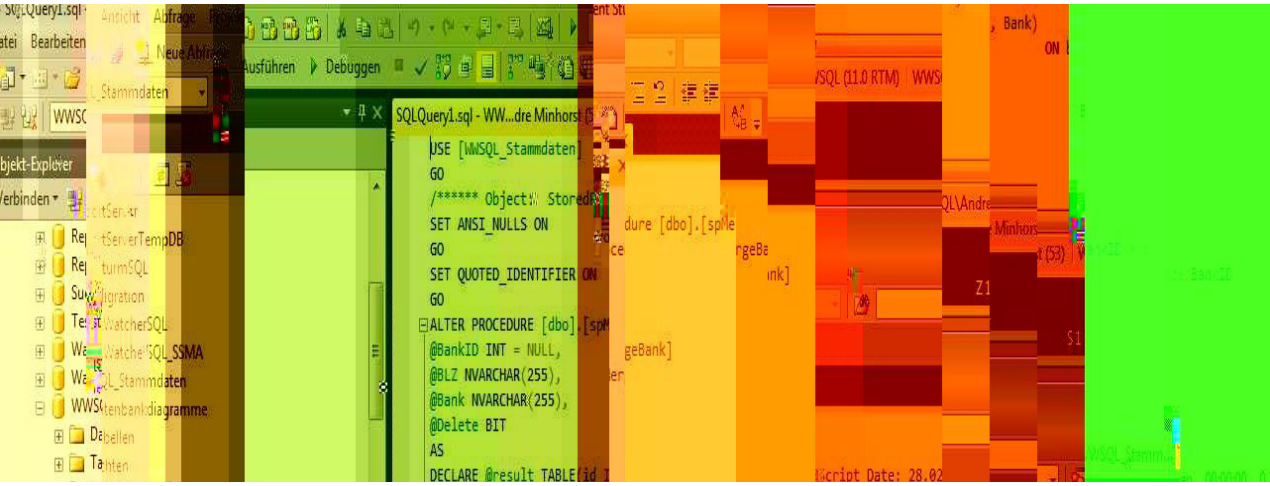
```
Private Sub Speichern()
    Dim varWarengruppeID As Variant
    If IsNull(Me!WarengruppeID) Then
        varWarengruppeID = SPAktionsabfrageMitErgebnis("dbo.spINSERTINTOWarengruppeMitID", _
            Standardverbindungszeichenfolge, True, "", Me!Warengruppe)
    Else
        varWarengruppeID = Nz(Me!WarengruppeID, "NULL")
        SPAktionsabfrageMitErgebnis "dbo.spALTERWarengruppe", _
            Standardverbindungszeichenfolge, False, "", varWarengruppeID, Me!Warengruppe
    End If
    Set Me.RecordSet = SPRecordset("dbo.spSELECTAlleWarengruppen", _
        Standardverbindungszeichenfolge)
    Me.Recordset.FindFirst "WarengruppeID = " & varWarengruppeID
    Me!cmdLoeschen.Enabled = True
End Sub
```

Die Prozedur prüft zunächst den Wert des Primärschlüsselfeldes *WarengruppeID*. Enthält das Feld den Wert *Null*, hat der Benutzer zuvor einen neuen Datensatz angelegt, der nun auch in der Tabelle gespeichert werden soll (siehe weiter unten). Anderenfalls soll ein vorhandener Datensatz mit den neuen Werten überschrieben werden. Beim Ändern eines vorhandenen Datensatzes verwendet die Prozedur die Routine *SPAktionsabfrageMitErgebnis*, um die gespeicherte Prozedur *spALTERWarengruppe* aufzurufen. Dabei soll die mit der Funktion *Standardverbindungszeichenfolge* ermittelte Verbindungszeichenfolge verwendet werden (siehe Abschnitt »ID der aktiven Verbindungszeichenfolge ermitteln«, Seite 310). Als weitere Parameter werden *False* für den Parameter *bolRueckgabeInSPImplementiert*, der Primärschlüsselwert aus *varWarengruppeID* und der neue Wert für das Feld *Warengruppe* übergeben. Bevor wir uns die gespeicherte Prozedur ansehen, betrachten wir noch die restlichen Zeilen der aktuellen Prozedur. Diese stellt die Datenherkunft wieder auf das Ergebnis der gespeicherten Prozedur *spSELECTAlleWarengruppen* ein und aktiviert mit der *FindFirst*-Methode wieder den zuvor bearbeiteten Datensatz.

Die danach aufgerufene Prozedur *SteuerelementeBinden* durchläuft alle Steuerelemente des Formulars und stellt die Eigenschaft *Steuerelementinhalt* (*ControlSource*) wieder auf den in der Eigenschaft *Marke* (*Tag*) gespeicherten Feldnamen ein – dies allerdings nur für solche Steuerelemente, deren *Tag*-Eigenschaft auch einen Wert enthält:

```
Private Sub SteuerelementeBinden()
    Dim ctl As Control
    For Each ctl In Me.Controls
```

... DIE ÜBRIGEN SEITEN GIBT'S GEDRUCKT UNTER SHOP.MINHORST.COM



BERND JUNGBLUTH
ANDRÉ MINHORST

André Minhorst Verlag

15 Sicherheit und Benutzerverwaltung

Einer der Hauptgründe, warum Sie Access-Anwendungen mit einer SQL Server-Datenbank als Backend verwenden, ist die Sicherheit. In reinen Access-Anwendungen können Sie schlicht und einfach nicht sicherstellen, dass jeder Benutzer nur die für ihn vorgesehenen Daten einsehen oder bearbeiten darf. Darüber hinaus können Sie noch nicht einmal verhindern, dass Unbefugte sich Zugriff zu den in den Tabellen enthaltenen Daten verschaffen. Es gibt zwar noch die Möglichkeit, vollständige Datenbanken mit einem Kennwort vor dem Zugriff Unbefugter zu schützen, aber hier ist keine Vergabe differenzierter Berechtigungen möglich.

SQL Server hingegen arbeitet mit einem mehrstufigen Sicherheitssystem. Nach der Authentifizierung am SQL Server selbst erfolgt die Autorisierung an den jeweiligen Datenbanken. Der Benutzer verschafft sich dabei mit seinem Windows-Benutzerkonto oder einem SQL Server-Benutzerkonto Zugang zum SQL Server und erhält dort die vorgesehenen Berechtigungen für die einzelnen Datenbanken. Diese wiederum können sehr fein eingestellt werden. Doch eines nach dem anderen – in den nächsten Abschnitten lernen Sie die Grundlagen des Sicherheitssystems unter SQL Server kennen und erfahren, wie Sie dieses einrichten.

15.1 Sicherheit auf Server- und Datenbankebene

Um mit den Daten einer Datenbank arbeiten zu können, müssen Sie durch zwei Türen. Die erste Tür ist die des SQL Servers und die zweite die der jeweiligen Datenbank. Der Vorgang ist vergleichbar mit den Türen eines Unternehmens.

Stellen Sie sich vor, Sie besuchen einen Kunden. Der Besuch beginnt in der Regel am Empfang. Dort melden Sie sich an. Die freundliche Empfangsdame prüft Ihr Anliegen, indem Sie bei dem entsprechenden Mitarbeiter nachfragt, ob er Sie denn erwartet. Bejaht der Mitarbeiter diese Frage, werden Sie in der Regel höflich gebeten, kurz auf ihn zu warten. Nun haben Sie zwar Zutritt zum Unternehmen, aber noch keinen richtigen Zugang zu einer Abteilung. Im Vergleich mit der mehrstufigen Anmeldung beim SQL Server haben Sie Ihre Authentifizierung am SQL Server erfolgreich bestanden, eine Autorisierung an einer Datenbank steht aber noch aus.

Die Autorisierung folgt direkt im Anschluss. In unserem Beispiel durch den Mitarbeiter des Unternehmens, der Sie mit in die Abteilung *Einkauf* nimmt. Dort sollen Sie Daten erfassen, ändern und löschen. In der Abteilung angekommen, haben Sie nun auch die Autorisierung bestanden. Die Autorisierung für die Abteilung *Einkauf*, in der Sie Daten ändern dürfen – oder im Fall des SQL Servers: die Autorisierung für die Datenbank *Einkauf*, in der Sie lesenden und schreibenden Zugriff auf die Daten haben.

Nach der Mittagspause wechseln Sie zur Abteilung *Personal*. Dort erstellen Sie neue Auswertungen. Sie besitzen also noch eine weitere Autorisierung zu einer anderen Abteilung, dort aller-

dings nur zum Lesen von Daten. Im Sinne vom SQL Server gibt es für Ihre Authentifizierung auch eine Autorisierung an der Datenbank *Personal*, hier jedoch nur mit lesendem Zugriff.

Das ist das Prinzip der SQL Server-Sicherheit. Mit einem Benutzerkonto erhalten Sie Zugang zum SQL Server – die sogenannte *Anmeldung*. Dieses Benutzerkonto kann einer oder mehreren Datenbanken zugeordnet sein – der *Benutzer* einer Datenbank. Eine Anmeldung kann als Benutzer in den jeweilis zugeordneten Datenbanken unterschiedliche Rechte haben. Wobei sich die Rechte sehr detailliert vergeben lassen: Sie können Daten lesen, ändern und löschen, Objekte erstellen, ändern und löschen sowie Objekte ausführen. Abbildung 15.1 zeigt dies schematisch.

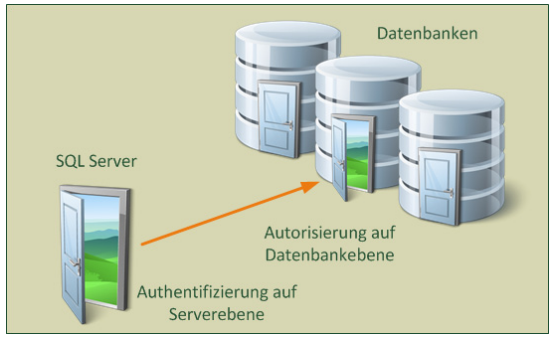


Abbildung 15.1: Das zweistufige Sicherheitskonzept beim SQL Server

15.2 Authentifizierung am SQL Server

Zur Authentifizierung bietet SQL Server zwei Varianten an.

» *Windows-Authentifizierung:* Bei der Windows-Authentifizierung prüft SQL Server, ob es für den Benutzer, der vom lokalen oder einem entfernten Rechner auf den SQL Server zugreift, ein Benutzerkonto auf dem SQL Server gibt. Dabei handelt es sich um das Domänen- oder Arbeitsgruppen-Benutzerkonto, das dem SQL Server direkt oder auch indirekt über eine lokale Gruppe oder eine Domänengruppe zugeordnet ist. Die Windows-Authentifizierung erfordert also keine eigene Anmeldung am SQL Server. Die Anmeldung erfolgt direkt mit der Windows-Anmeldung.

» *SQL Server-Authentifizierung:* Bei der SQL Server-Authentifizierung prüft SQL Server den Zugriff ebenfalls über ein Benutzerkonto. Nur handelt es sich hierbei um ein Benutzerkonto, dass im SQL Server selbst verwaltet wird. Dazu muss das Benutzerkonto samt Kennwort im SQL Server anlegt werden. Die SQL Server-Authentifizierung erfordert immer eine Anmeldung am SQL Server – mit dem Benutzernamen und dem Kennwort. Auch das Benutzerkonto *sa*, das immer vorhanden ist (und tunlichst mit einem starken Kennwort versehen werden will), ist ein Benutzerkonto der SQL Server-Authentifizierung.

Die Windows-Authentifizierung ist die Standard-Authentifizierungsmethode. Diese kann bei der Installation oder auch nachträglich in den Eigenschaften der SQL Server-Instanz mit der SQL Server-Authentifizierung erweitert werden.

15.3 Benutzer und Benutzergruppen

Wenn Sie eine Access/SQL Server-Anwendung installieren, müssen Sie die Berechtigungen der Benutzer oder Gruppen für die Objekte der jeweiligen Datenbanken festlegen.

Je nach Größe des Unternehmens des Kunden finden Sie in dessen Netzwerk bereits Gruppen- und Benutzerkonten vor. Das heißt, dass für jeden Mitarbeiter ein unternehmensweites Windows-Benutzerkonto existiert. Außerdem gibt es verschiedene Windows-Benutzergruppen, welche die Benutzer enthalten und für die bestimmte Berechtigungen festgelegt wurden.

Die Berechtigungen werden in der Regel auf Gruppenbasis vergeben, damit beim Ausscheiden oder Hinzukommen von Benutzern keine neuen Regeln für den jeweiligen Benutzer entfernt oder festgelegt werden müssen – der Benutzer wird einfach der Benutzergruppe entnommen beziehungsweise zu einer Benutzergruppe hinzugefügt, welche die entsprechenden Berechtigungen aufweist.

Genauso sollten Sie auch beim SQL Server vorgehen – vorausgesetzt, dies ist beim Kunden nicht bereits so eingerichtet. Falls Sie das System neu aufsetzen (manchmal landet eine Anwendung ja auch auf einer eigenen SQL Server-Instanz), verwenden Sie die entsprechenden Windows-Benutzergruppen zur Authentifizierung – und keine einzelnen Windows-Benutzerkonten.

Der Vorteil liegt auf der Hand. Sie ordnen die Windows-Benutzergruppen den einzelnen Datenbanken zu und vergeben dort die Berechtigungen. Dies spart nicht nur eine Menge Verwaltungsaufwand, denn in der Regel gibt es weniger Gruppen als Benutzer.

Sie sparen auch eine doppelte Rechteverwaltung. Verwalten Sie die Windows-Benutzer einzeln, müssen Sie bei jeder personellen Änderung die Zuordnung des Benutzers zu den Datenbanken und den damit verbundenen Rechten anpassen.

Eine personelle Änderung wird aber bereits auf der Systemebene durch den Systemadministrator verwaltet, indem dieser den Benutzer einer anderen Windows-Benutzergruppe zuordnet. Verwenden Sie im SQL Server lediglich die Windows-Benutzergruppe, wirken sich die Änderungen des Systemadministrators direkt auf die Rechte des Benutzers im SQL Server und den Datenbanken aus.

Wechselt also ein Mitarbeiter von der Abteilung *Einkauf* in die Abteilung *Personal*, verschiebt der Systemadministrator das Windows-Benutzerkonto des Mitarbeiters in die Windows-Benutzergruppe *Personal*. Da diese Windows-Benutzergruppe im SQL Server der Datenbank *Personal* mit entsprechenden Rechten zugeordnet ist, hat der Mitarbeiter direkt auch alle notwendigen Rechte an der Datenbank *Personal*.

Domäne oder keine Domäne

Möchten Sie per Windows-Authentifizierung auf den SQL Server zugreifen, setzt dies eine Windows-Domäne voraus. Sie können sich nicht mit einem lokalen Windows-Benutzer oder einer Windows-Benutzergruppe an einen SQL Server auf einem anderen Rechner anmelden. Dies gelingt nur – wie in den folgenden Beispielen beschrieben – wenn der Benutzer am gleichen Rechner angemeldet ist, auf dem auch der SQL Server läuft.

Verwendet der Kunde keine Domäne, bleibt Ihnen nur die SQL Server-Authentifizierung. Was zieht dies für die folgenden Beispiele nach sich? Lediglich den Hinweis, dass Sie, wenn das Testnetzwerk keine Domäne enthält, nur vom lokalen Rechner auf den SQL Server zugreifen können. Möchten Sie beispielsweise in Ihrem privaten Netzwerk von einem Rechner auf den SQL Server auf einem anderen Rechner zugreifen, müssen Sie die SQL Server-Authentifizierung verwenden.

15.4 Beispiel zum Einrichten der Sicherheit mit der Windows-Authentifizierung

In den folgenden Abschnitten nehmen wir eine Beispielkonfiguration vor. Dabei erledigen wir folgende Schritte:

» Anlegen zweier Benutzerkonten unter Windows

» Anlegen zweier Windows-Benutzergruppen, denen je eines der neuen Benutzerkonten hinzugefügt wird

» Zuordnen der Windows-Benutzergruppen als Anmeldungen im SQL Server

» Zuordnen dieser neuen Anmeldungen zur Beispieldatenbank *AEMA_SQL*

15.5 Benutzer unter Windows anlegen

Wenn Sie mit den nachfolgenden Beispielen experimentieren möchten, legen Sie zwei Benutzer auf Ihrem Windows-System an. Dazu gehen Sie wie folgt vor:

» Klicken Sie im Startmenü mit der rechten Maustaste auf den Eintrag *Computer* und wählen Sie den Eintrag *Verwalten* aus dem Kontextmenü aus.

» Öffnen Sie im nun erscheinenden Dialog den Ordner *System|Lokale Benutzer und Gruppen*.

» Wählen Sie Kontextmenü des Ordners *Benutzer* den Eintrag *Neuer Benutzer...* aus (siehe Abbildung 15.2).

» Geben Sie die Benutzerdaten für den neuen Benutzer an (siehe Abbildung 15.3).

» Schließen Sie die Eingabe mit einem Klick auf *Erstellen* ab. Der Benutzer wird gespeichert und der Dialog *Neuer Benutzer* wird geleert, damit Sie weitere Benutzer anlegen können. Legen Sie gleich noch einen weiteren Benutzer namens *Alfred Neumann* an und schließen Sie den Dialog danach mit einem Klick auf *Schließen*.

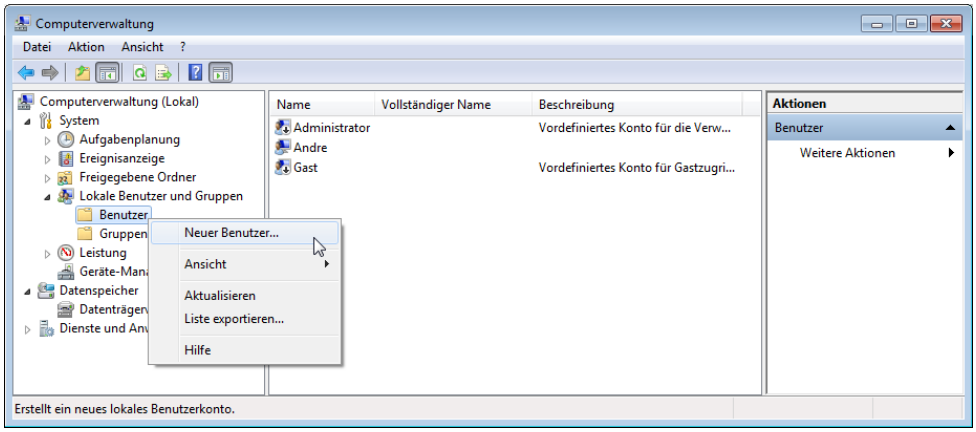


Abbildung 15.2: Anlegen eines neuen Windows-Benutzers

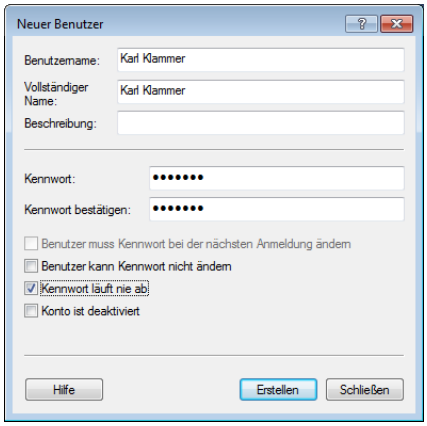


Abbildung 15.3: Eintragen der Daten des neuen Benutzers

Die beiden neu angelegten Benutzer werden nun in der Liste der Benutzer angezeigt (siehe Abbildung 15.4).

Nun fügen Sie zwei Gruppen hinzu, denen Sie die beiden Benutzer zuweisen. Dazu wählen Sie den Kontextmenüeintrag *Neue Gruppe* im Ordner *Gruppen* in der *Computerverwaltung* aus.

Im nun erscheinenden Dialog legen Sie eine neue Gruppe namens *Basis* an (siehe Abbildung 15.5).

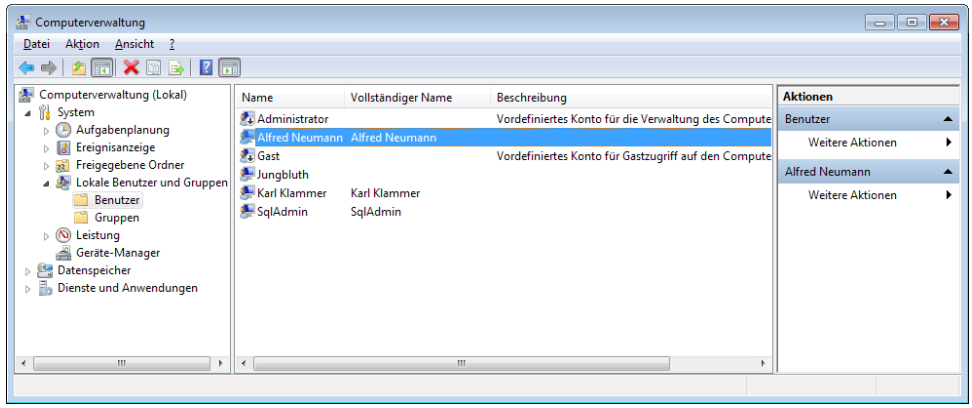


Abbildung 15.4: Liste mit neuen Windows-Benutzern

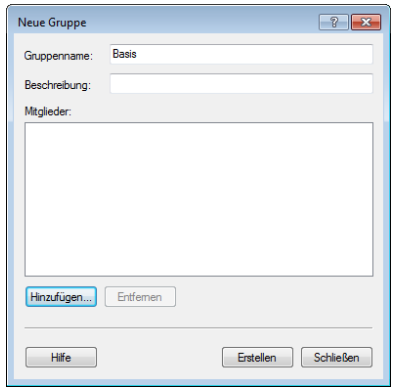


Abbildung 15.5: Anlegen einer neuen Gruppe namens *Basis*

Dieser Gruppe weisen Sie nun noch einen Benutzer zu, beispielsweise den zuvor angelegten *Karl Klammer*. Klicken Sie dazu auf *Hinzufügen*. Im nun erscheinenden Dialog können Sie die anzulegenden Elemente einfach eintragen (siehe Abbildung 15.6).

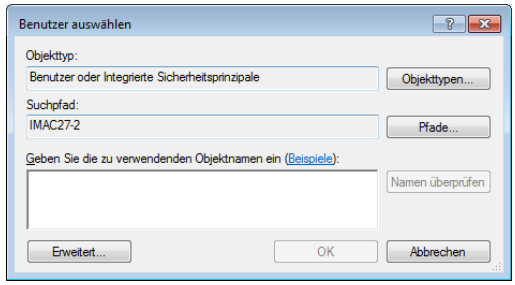


Abbildung 15.6: Auswählen des hinzuzufügenden Benutzers

Wenn Sie hier *Karl Klammer* eintragen und auf *Namen überprüfen* klicken, ergänzt der Dialog noch den Rechnernamen (etwa *ASQL\Karl Klammer*). Haben Sie den Benutzernamen nicht im Kopf, klicken Sie auf *Erweitert...* und zeigen so den Dialog aus Abbildung 15.7 an. Hier finden Sie nach Betätigung der Schaltfläche *Jetzt suchen* unter anderem auch die eben angelegten Benutzer.

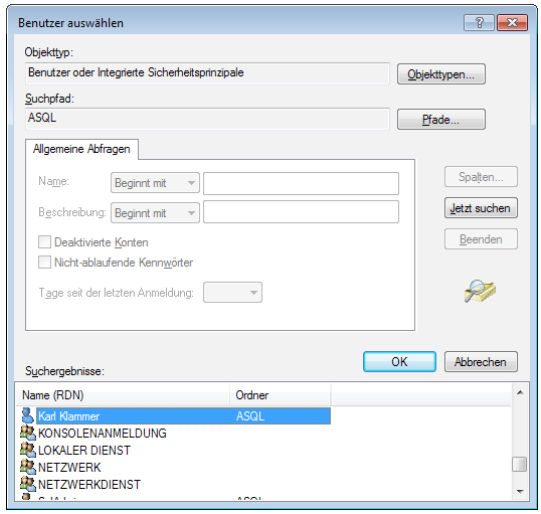


Abbildung 15.7: Auswählen hinzuzufügender Benutzer

Sollte der gesuchte Benutzer dort nicht auftauchen, stellen Sie mit einem Klick auf die Schaltfläche *Objekttypen...* sicher, dass die Benutzer auch angezeigt werden. Gegebenenfalls müssen Sie dort noch den Eintrag *Benutzer* aktivieren (siehe Abbildung 15.8).

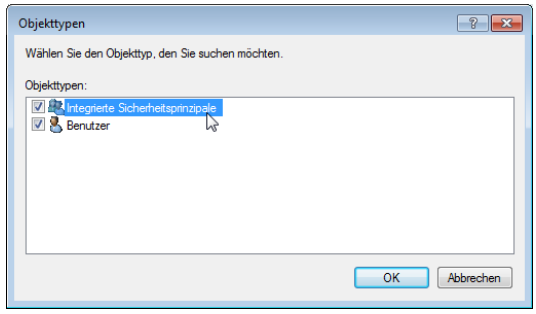


Abbildung 15.8: Aktivieren der Anzeige der Benutzer in der Auswahlliste

Schließlich landet der neue Benutzer in der neuen Gruppe (siehe Abbildung 15.9). Auf die gleiche Weise legen Sie eine weitere Gruppe namens *Erweitert* an und weisen dieser den Benutzer *Alfred Neumann* zu.

Die beiden frisch angelegten Gruppen erscheinen schließlich in der Liste der eingebauten und der benutzerdefinierten Gruppen in der Computerverwaltung (siehe Abbildung 15.10).

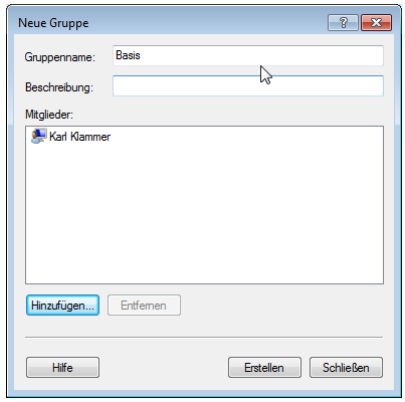


Abbildung 15.9: Neue Benutzergruppe mit dem ersten Benutzer

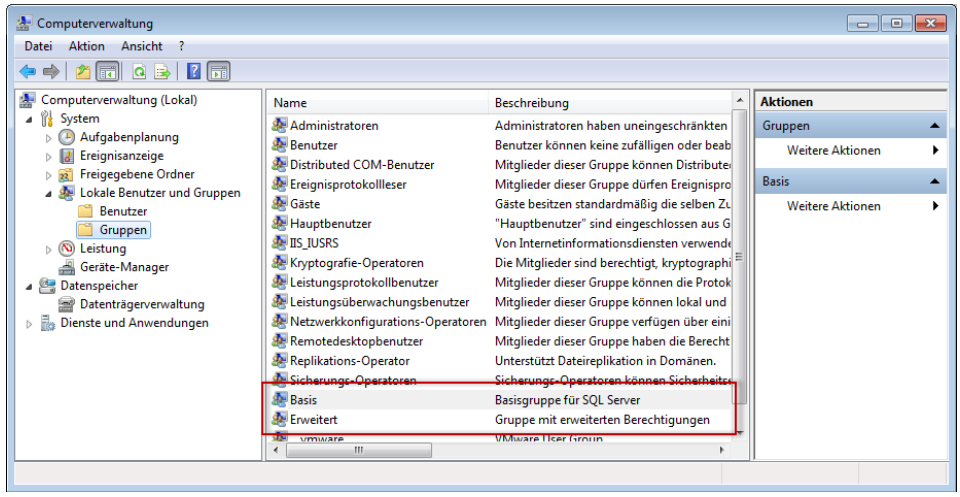


Abbildung 15.10: Neue Gruppen in der Computerverwaltung

15.5.1 Windows-Benutzergruppen im SQL Server

Wie bereits erwähnt, wollen wir im SQL Server gar nicht erst beginnen, einzelne Windows-Benutzer zu verwalten, sondern gleich auf Basis der Windows-Benutzergruppen arbeiten. Hier ist zu beachten, dass wir nun zunächst *Anmeldungen* zur Authentifizierung am SQL Server erstellen, diesen die Windows-Benutzergruppen zuordnen und anschließend die neuen Anmeldungen als *Benutzer* in der Zieldatenbank anlegen. *Anmeldungen* und *Benutzer* legen Sie im SQL Server über das *SQL Server Management Studio* an. Hier klicken Sie mit der rechten Maustaste auf den

Eintrag *Sicherheit* und wählen den Befehl *Neu|Anmeldung* aus dem Kontextmenü aus (siehe Abbildung 15.11).

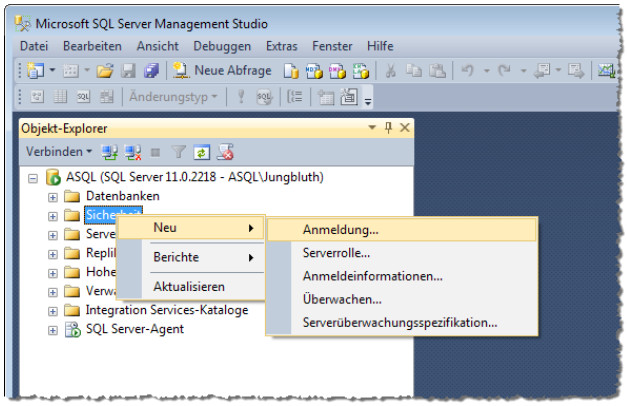


Abbildung 15.11: Hinzufügen einer SQL Server-Anmeldung

Im nun erscheinenden Dialog klicken Sie der Einfachheit halber auf die Schaltfläche *Suchen* (siehe Abbildung 15.12).

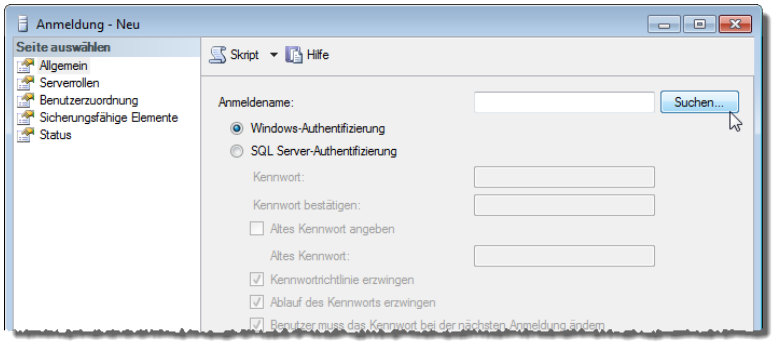


Abbildung 15.12: Hinzufügen der neuen Anmeldungen über die *Suchen*-Funktion

Dies öffnet den Dialog *Benutzer oder Gruppen auswählen* aus Abbildung 15.13. Hier können Sie den Objektnamen in der Form *<Rechnername>\<Gruppenname>* eintragen und diesen mit einem Klick auf *Namen überprüfen* verifizieren.

Versuchen wir dies mit *<Rechnername>\Basis*, also eine der beiden soeben unter Windows angelegten Benutzergruppen, findet der Dialog die Benutzergruppe möglicherweise nicht.

Das Problem lässt sich in der Regel leicht lösen: Klicken Sie einmal auf die Schaltfläche *Objekttypen...*, erscheint der Dialog aus Abbildung 15.14. Hier dürfte die Option *Gruppen* deaktiviert sein. Aktivieren Sie diese und schließen Sie den Dialog wieder.

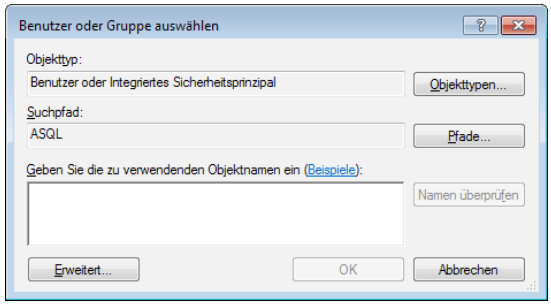


Abbildung 15.13: Dialog zum Auswählen von Benutzern oder Gruppen

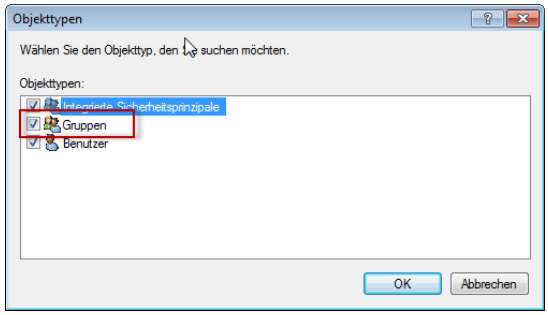


Abbildung 15.14: Objektart *Gruppe* aktivieren

Klicken Sie nun im Dialog *Benutzer oder Gruppen auswählen* auf die Schaltfläche *Namen überprüfen*, erkennt der SQL Server die gewünschte Benutzergruppe.

Sollten Sie die Benutzergruppe immer noch nicht finden, klicken Sie im Dialog *Benutzer oder Gruppen auswählen* auf die Schaltfläche *Erweitert*. Dies öffnet den Dialog aus Abbildung 15.15. Eventuell müssen Sie auch hier unter *Objekttypen* den Eintrag *Gruppen* aktivieren, bevor Sie auf die Schaltfläche *Suchen* klicken. Die Liste im unteren Bereich sollte nun den gewünschten Eintrag anzeigen, den Sie durch Markieren und Schließen des Dialogs übernehmen.

Standarddatenbank einstellen

Im Dialog *Anmeldung – Neu* legen Sie noch die Standarddatenbank fest – in diesem Fall die Beispieldatenbank *AEMA_SQL*. Auf diese Weise können Sie später beim Erstellen der Verbindungszeichenfolgen direkt auf die Standarddatenbank zugreifen.

Serverrolle festlegen

Nun wechseln Sie zur Seite *Serverrollen*. Hier legen Sie fest, welche administrativen Rechte die neue Anmeldung im SQL Server hat. Standardmäßig sind alle Anmeldungen der Serverrolle *public* zugeordnet (siehe Abbildung 15.16).

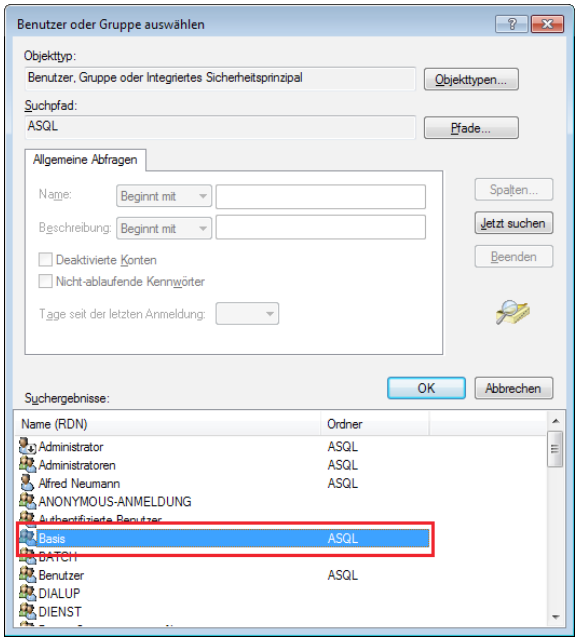


Abbildung 15.15: Suchen nach Windows-Benutzern oder Windows-Benutzergruppen

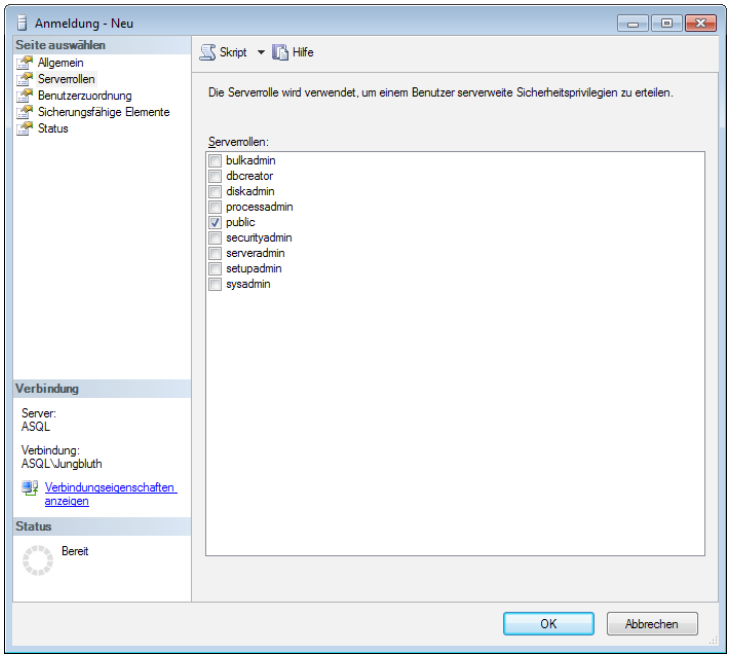


Abbildung 15.16: Die Serverrollen vom SQL Server

Die Serverrolle *public* stellt den kleinsten gemeinsamen Nenner der Berechtigungen zur Verwaltung des SQL Servers dar. Durch Auswahl einer oder mehrerer anderer Serverrollen erweitern Sie die Rechte einer Anmeldung.

» *sysadmin*: Ausführen aller Aktivitäten im SQL Server

» *serveradmin*: Ändern der SQL Server-Konfiguration, sowie Beenden vom SQL Server-Dienst

» *securityadmin*: Verwalten von Anmeldungen

» *processadmin*: Beenden von Prozessen

» *setupadmin*: Hinzufügen und entfernen von Verbindungsservern

» *bulkadmin*: Ausführen von Massenimporten

» *diskadmin*: Verwalten von Datensicherungsmedien

» *dbcreator*: Anlegen, ändern, löschen und wiederherstellen von Datenbanken

» *Eigene Serverrollen*: Seit SQL Server 2012 können eigene Serverrollen erstellt werden.

Die jeweiligen Rechte der einzelnen Serverrollen können Sie sich mit der Systemprozedur *sp_srvrolepermission <Serverrolle>* anschauen.

Je nach Zuordnung der Anmeldung zu den Serverrollen vergeben Sie also Rechte zur Administration des SQL Servers. Beinhaltet die Anmeldung wie in unserem Fall eine Windows-Benutzergruppe, stehen allen Benutzer dieser Gruppe die jeweiligen administrativen Rechte der zugeordneten Serverrolle zur Verfügung. Seien Sie also vorsichtig mit der Zuordnung der Serverrollen. Insbesondere bei der Serverrolle *sysadmin* wird gerne übersehen, dass Benutzer dieser Serverrolle vollen Zugriff auf alle Datenbanken und den darin gespeicherten Daten haben.

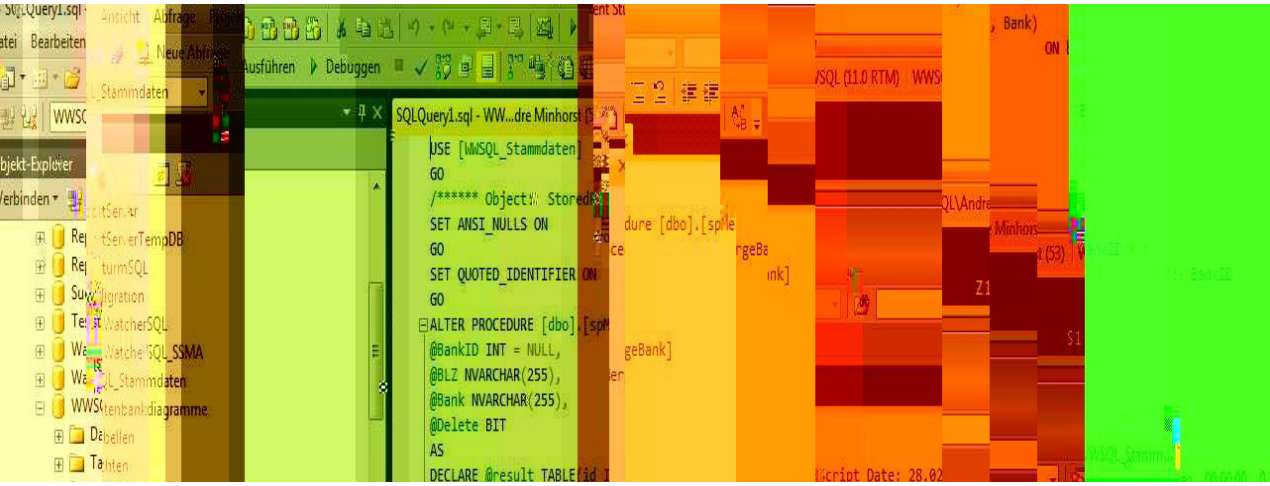
In unserem Beispiel belassen wir es bei der Standardeinstellung und ordnen der Anmeldung keine weiteren Serverrollen zu. Damit sind alle notwendigen Schritte der Authentifizierung erledigt.

15.5.2 Von der Anmeldung zum Benutzer

Nun müssen Sie noch festlegen, auf welche Datenbanken die Anmeldung (die Mitglieder der Windows-Benutzergruppe) zugreifen darf. Dies entspricht der zweiten Stufe des SQL Server-Sicherheitskonzepts – der Autorisierung. Dazu wechseln Sie zur Seite *Benutzerzuordnung*. Dort sehen Sie alle Datenbanken des SQL Servers, die sie per Mausklick der Anmeldung zuordnen können. Diese Zuordnung erstellt in der jeweiligen Datenbank den *Benutzer* für die Anmeldung.

Wählen Sie die Beispieldatenbank *AEMA_SQL* aus, um die Anmeldung *ASQL/Basis* als Benutzer in der Datenbank *AEMA_SQL* anzulegen. Der Benutzername lässt sich in der Spalte *Benutzer* ändern. In unserem Beispiel steht die Anmeldung *ASQL/Basis* als Benutzer *Basis* in der Datenbank *AEMA_SQL* zur Verfügung (siehe Abbildung 15.17).

... DIE ÜBRIGEN SEITEN GIBT'S GEDRUCKT UNTER SHOP.MINHORST.COM



BERND JUNGBLUTH
ANDRÉ MINHORST

André Minhorst Verlag

16 Bilder und Dateien im SQL Server

Eines der großen Probleme von Access-Datenbanken ist der vergleichsweise geringe Speicherplatz. Sicher, wenn Sie nur reine Textinformationen oder Zahlen verwalten, kommen Sie damit eine Weile aus. Soll die Datenbank aber auch noch Elemente wie Bilder oder Dateien aufnehmen, geraten Sie schnell ans Limit. Unter Access gibt es dabei die Möglichkeit, nur den Pfad zur jeweiligen Datei zu speichern und diese im Dateisystem zu belassen, aber damit sind einige Einschränkungen verbunden – zum Beispiel müssen Sie die Pfadangaben ändern, wenn die Datenbank samt referenzierter Dateien einmal verschoben werden soll und Sie müssen sich selbst darum kümmern, dass die Daten immer an Ort und Stelle bleiben und gesichert werden.

16.1 FILESTREAM

Mit SQL Server 2008 hat Microsoft ein neues Feature namens *FILESTREAM* eingeführt, mit dem Sie Dateien im SQL Server speichern können, ohne dass die Datenbank unnötig anwächst oder Sie sich selbst um die Verwaltung der Dateien in einem Verzeichnis kümmern müssen. *FILESTREAM* bietet das Beste der beiden Varianten: Sie speichern die Dateien direkt in der Tabelle und der SQL Server legt diese in einem Verzeichnis ab und verwaltet sie dort.

In SQL Server 2012 wurde dieses Feature noch ausgebaut – um die *FileTable*, die wir weiter unten betrachten. Wir schauen uns in diesem Kapitel zunächst an, wie *FILESTREAM* funktioniert und wie Sie Dateien über Access in einer SQL Server-Datenbank speichern und diese in Access wieder auslesen. Dabei bauen wir auf dem Stand von SQL Server 2012 auf.

16.1.1 FILESTREAM aktivieren

Bevor Sie *FILESTREAM* nutzen können, müssen Sie das Feature zunächst aktivieren. Dazu starten Sie den *SQL Server-Konfigurations-Manager* (Startmenü, *Alle Programme|SQL Server 2012|Konfigurationstools|SQL Server-Konfigurations-Manager*). Klicken Sie doppelt auf den Eintrag *SQL Server-Dienst* und dann mit der rechten Maustaste rechts auf *SQL Server (MSSQLSERVER)* — wobei der in Klammern angegebene Name der Instanzname ist und auch abweichen kann. Wählen Sie den Eintrag *Eigenschaften* aus dem Kontextmenü aus.

Im nun erscheinenden Dialog wechseln Sie zur Registerseite *FILESTREAM* (siehe Abbildung 16.1). Dort finden Sie folgende Optionen:

» *FILESTREAM für Transact-SQL-Zugriff aktivieren*: Aktiviert den *FILESTREAM* auf SQL Server-Basis.

» *FILESTREAM für E/A-Datenzugriff aktivieren*: Aktiviert FILESTREAM für den Zugriff über das Dateisystem.

» *Windows-Freigabename*: Gibt den Namen des Verzeichnisses an, in dem die Dateien gespeichert werden.

» *Zugriff von Remoteclients auf FILESTREAM-Daten zulassen*: Erlaubt den Zugriff von anderen Rechnern im Netzwerk.

Die hier vorgenommenen Einstellungen können Sie auch über das SQL Server Management Studio vornehmen. Dazu zeigen Sie die Eigenschaften des Eintrags für den SQL Server an und wechseln dort zur Seite *Erweitert*. Wenn Sie *FILESTREAM* für den Zugriff über das Dateisystem aktiviert haben, finden Sie im Windows Explorer eine entsprechende Freigabe vor (siehe Abbildung 16.2). Wozu diese benötigt wird, erfahren Sie weiter unten.

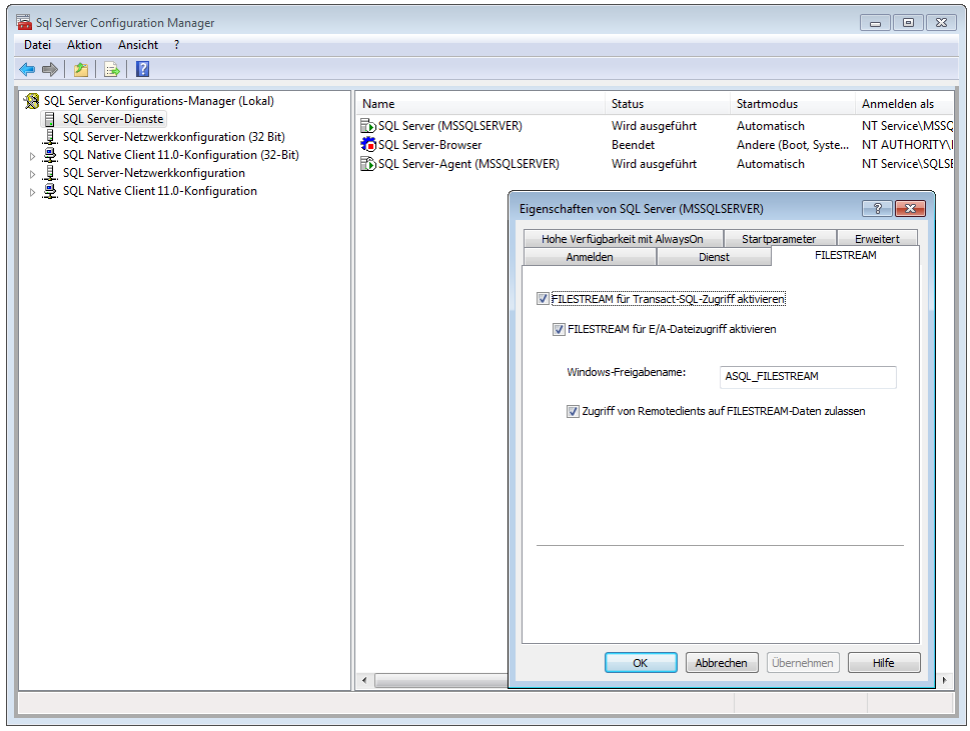


Abbildung 16.1: Aktivieren von *FILESTREAM*

16.1.2 Datenbank für FILESTREAM erstellen

Nun schauen wir uns die notwendigen Arbeiten auf Seite der Datenbank an. Dazu legen wir zunächst eine neue Datenbank namens *ASQL_FILESERVER* an. Dies erledigen wir diesmal direkt per T-SQL, und zwar wie folgt (die Pfade müsse Sie gegebenenfalls anpassen):

```
CREATE DATABASE ASQL_FILESTREAM
ON
```

```
PRIMARY (NAME=ASQL_FILESTREAM_DB,
FILENAME='C:\Program Files\Microsoft SQL Server\MSSQL11.ASQL\MSSQL\DATA\FILESTREAM_DB.mdf'),
FILEGROUP ASQL_FILESTREAM_GROUP
CONTAINS FILESTREAM(NAME=FILESTREAM_FS,
FILENAME='C:\Program Files\Microsoft SQL Server\MSSQL11.ASQL\MSSQL\DATA\ASQL_FILESTREAMS')
LOG ON (NAME=ASQL_FILESTREAM_LOG,
FILENAME='C:\Program Files\Microsoft SQL Server\MSSQL11.ASQL\MSSQL\DATA\FILESTREAM_DB.ldf')
```

Der erste Teil der Anweisung erstellt die Datenbank-Datei und legt einen logischen Namen für die Dateigruppe fest. *CONTAINS FILESTREAM* legt fest, dass die Datenbank in Tabellen gespeicherte Dateien letztendlich im Dateisystem speichert. Das dahinter mit FILENAME angegebene Verzeichnis wird ebenfalls erstellt.

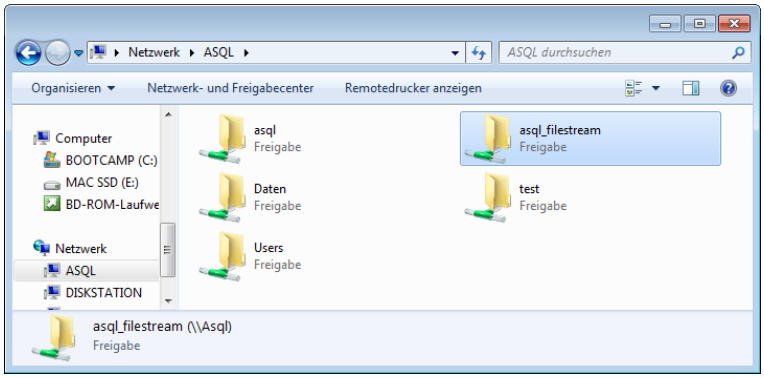


Abbildung 16.2: Freigabe für die über den SQL Server gespeicherten Dateien

Die neue Datenbank kommt im SQL Server Management Studio wie jede andere daher – kein Wunder: Die Möglichkeiten für den Umgang mit diesem Feature beschränken sich auch auf T-SQL, wie die folgenden Abschnitte zeigen werden.

16.1.3 Tabelle mit FILESTREAM-Spalte erstellen

Eine Tabelle, die Dateien in einer *FILESTREAM*-Spalte speichern soll, weist zwei Besonderheiten gegenüber herkömmlichen Tabellen auf:

» Die Spalte für die *FILESTREAM*-Daten muss den Datentyp *varbinary(max)* aufweisen und mit dem Schlüsselwort *FILESTREAM* gekennzeichnet werden.

» Die Tabelle muss eine Spalte mit eindeutigen GUIDs enthalten. Dazu fügen Sie der Tabelle eine Spalte mit dem Datentyp *uniqueidentifier* hinzu, ergänzen diese mit dem Parameter *ROWGUIDCOL* und dem DEFAULT-Wert *NewId()* (diese Funktion erstellt eine neue GUID) sowie der Definition eines eindeutigen Schlüssels. Auch wenn es naheliegend ist, diese Spalte direkt als Primärschlüssel zu verwenden, sollten Sie es auch Gründen besserer Performance nicht tun.

Die folgende Anweisung erstellt eine Tabelle namens *tblBilder* mit den Spalten *BildID* (Primärschlüsselspalte), *Bildbezeichnung*, *Bild* (als *FILESTREAM*-Spalte) und *DateiID* (als eindeutige *GUID*-Spalte mit *NewId()* als Standardwert):

```
CREATE TABLE dbo.tblBilder (
    BildID int IDENTITY(1,1) CONSTRAINT PK_tblBilder_BildID PRIMARY KEY,
    Bildbezeichnung varchar(255),
    Bild varbinary(max) FILESTREAM,
    DateiID UNIQUEIDENTIFIER ROWGUIDCOL DEFAULT newid()
    NOT NULL CONSTRAINT UK_tblBilder_DateiID UNIQUE);
```

Wenn Sie sich die Tabelle im Entwurf ansehen, finden Sie keinerlei Hinweis darauf, dass es sich bei der Spalte *Bild* um eine *FILESTREAM*-Spalte handelt (siehe Abbildung 16.3).

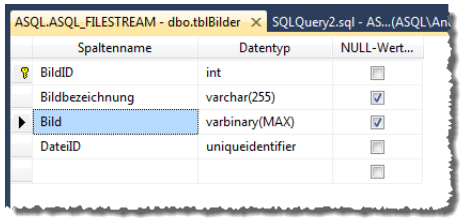


Abbildung 16.3: Tabelle mit *FILESTREAM*-Spalte

16.1.4 FILESTREAM-Spalte unter Access

Nach dem Einbinden der Tabelle in eine Access-Datenbank liefert diese ebenfalls keine besonderen Eigenschaften – die *FILESTREAM*-Spalte wird als herkömmliche Spalte mit dem Datentyp *OLE-Objekt* angezeigt (siehe Abbildung 16.4).

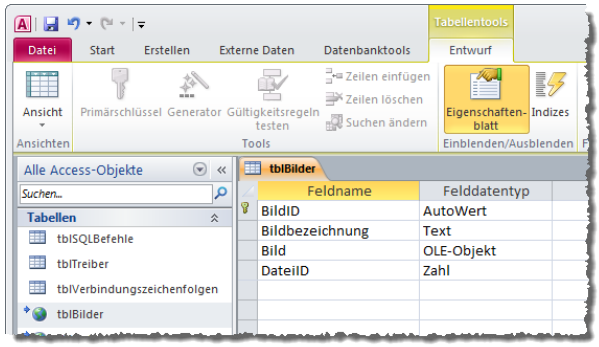


Abbildung 16.4: Unter Access eingebundene Tabelle mit *FILESTREAM*-Feld

Verwenden Sie die Tabelle als Datenherkunft eines Formulars, können Sie Dateien ganz einfach per Drag and Drop vom Windows Explorer in das OLE-Feld ziehen (siehe Abbildung 16.5). Das

Bild wird dann allerdings als Symbol angezeigt und nicht als Bild. Ein Doppelklick auf das OLE-Feld öffnet die Datei dann auch mit der dafür vorgesehenen Anwendung.

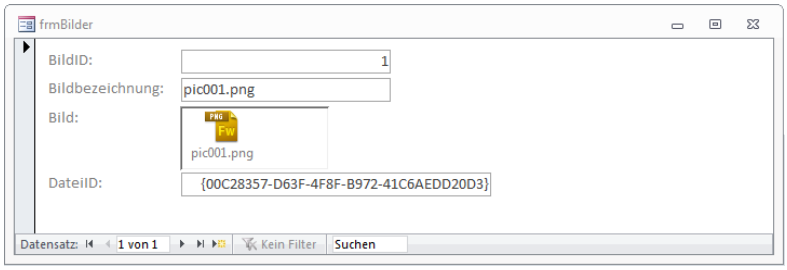


Abbildung 16.5: Formular mit OLE-Feld

Nun interessiert uns aber zunächst einmal, was mit der im Feld *Bild* gespeicherten Bilddatei geschehen ist. Die Abfrage des entsprechenden Datensatzes im Abfragefenster des SQL Server Management Studios liefert das Ergebnis aus Abbildung 16.6, also die binäre Darstellung der Datei.

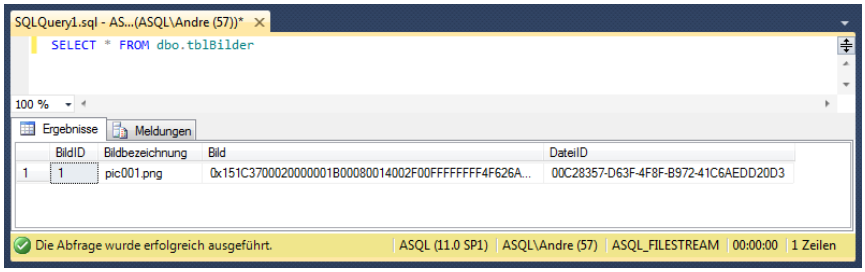


Abbildung 16.6: Bild als BLOB im SQL Server

Ein Blick in das für die *FILESTREAM*-Dateien angelegte Verzeichnis liefert auch nicht viel Erhellendes (siehe Abbildung 16.7). Dort befinden sich zwar einige Ordner, von denen einer auch eine Datei enthält, deren Speichergröße ungefähr mit der des gespeicherten Bildes übereinstimmt, aber öffnen lässt sich die Datei nicht – auch nicht nach dem Anfügen der Original-Dateiendung. Die Verwaltung dieser Ordner und Dateien obliegt alleine dem SQL Server-Dienst.

Löschen Sie nun den Datensatz mit dem eben eingefügten Objekt, wird auch die zugehörige Datei im Verzeichnis entfernt. Jedoch nicht direkt, da dies ein eigener Prozess — der *Garbage Collector* — übernimmt.

16.2 Tabellen mit FileTable

Die *FileTable*-Technik basiert auf *FILESTREAM*. Damit können Sie sowohl Dateien über die Datenbank speichern und diese dann über das Dateisystem öffnen als auch Dateien in dem für

die Datenbank vorgesehenen Verzeichnis speichern und diese damit unter den Zugriff der SQL Server-Datenbank stellen.

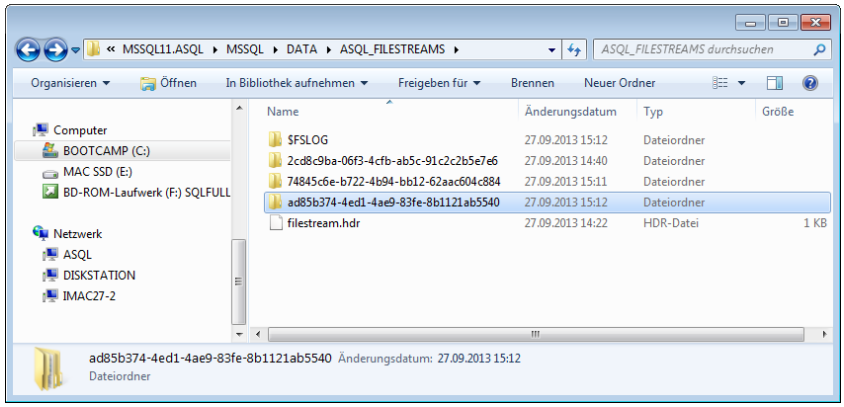


Abbildung 16.7: Verzeichnisse im *FILESTREAM*-Ordner

16.2.1 FILESTREAM-Beispieldatenbank erstellen

Zu Beispielzwecken erstellen wir zunächst eine neue Datenbank, die mit normalem *FILESTREAM* ausgestattet ist:

```
CREATE DATABASE ASQL_FileTable
ON
PRIMARY (NAME=ASQL_FileTable_DB,
FILENAME='C:\Program Files\Microsoft SQL Server\MSSQL11.ASQL\MSSQL\DATA\FileTable_DB.mdf'),
FILEGROUP ASQL_FileTable_GROUP CONTAINS FILESTREAM(NAME=FileTable_FS,
FILENAME='C:\Program Files\Microsoft SQL Server\MSSQL11.ASQL\MSSQL\DATA\ASQL_FileTable')
LOG ON (NAME=ASQL_FileTable_LOG,
FILENAME='C:\Program Files\Microsoft SQL Server\MSSQL11.ASQL\MSSQL\DATA\FileTable_DB.ldf')
```

16.2.2 FILESTREAM-Verzeichnis festlegen

Weiter oben haben wir bei der Anpassung des SQL Servers für die Verwendung von *FILESTREAM* und *FileTable* bereits einen Windows-Freigabenamen festgelegt, unter dem die vom SQL Server verwalteten Dateien aufzufinden sein sollen. Diese Freigabe wird von der kompletten SQL Server-Instanz genutzt, was bedeutet, dass nicht nur eine, sondern auch mehrere Datenbanken darauf zugreifen. Deshalb müssen Sie pro Datenbank noch ein Unterverzeichnis erstellen. Dieses legen Sie in den Eigenschaften der Datenbank auf der Seite *Optionen* unter *FILESTREAM* fest.

Stellen Sie für die Eigenschaft *FILESTREAM-Verzeichnisname* einen entsprechenden Wert wie etwa *ASQL_FileTable* ein und legen Sie mit der Eigenschaft *Nicht transaktionsgebundener FILESTREAM-Zugriff* fest, wieweit der Zugriff von außerhalb des SQL Servers zugelassen werden soll – also beispielsweise über den Windows Explorer (siehe Abbildung 16.8).

Das Verzeichnis können Sie auch mit folgender Anweisung festlegen:

```
ALTER DATABASE ASQL_FILETABLE SET FILESTREAM(DIRECTORY_NAME='ASQL_FILETABLE');
```

Und diese Anweisung stellt die Zugriffsart auf *FULL*, *READONLY* oder *OFF* ein:

```
ALTER DATABASE ASQL_FILETABLE SET FILESTREAM(NON_TRANSACTED_ACCESS=FULL);
```

Wenn Sie nun noch einmal den Windows Explorer bemühen, werden Sie unter der bereits vorhandenen Freigabe einen neuen Ordner vorfinden, der den angegebenen Namen trägt (siehe Abbildung 16.9).

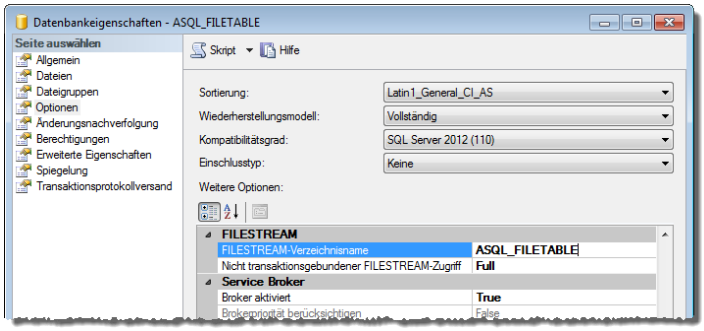


Abbildung 16.8: *FILESTREAM*-Einstellungen für eine Datenbank

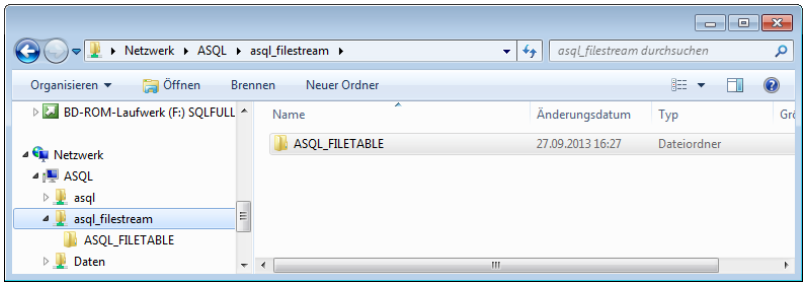


Abbildung 16.9: Ein neues Verzeichnis unterhalb der *FILESTREAM*-Freigabe

16.2.3 FileTable erstellen

Zum aktuellen Zeitpunkt können Sie allerdings noch keine Dateien über das Dateisystem in diesem Verzeichnis ablegen. Um dies zu ändern, müssen Sie eine Tabelle zum Speichern der Dateien erstellen. Für *FileTables* sieht der SQL Server einen eigenen Bereich im Objekt-Explorer vor, und zwar unterhalb des Eintrags *Tabellen*. Dort finden Sie im Kontextmenü auch die Möglichkeit, den grundlegenden Code zum Erstellen einer neuen *FileTable* in einer neuen Abfrage abzubilden (siehe Abbildung 16.10). Im Gegensatz zur *FILESTREAM*-Spalte, das Sie einfach mit anderen Spalten in einer herkömmlichen Tabelle unterbringen können, ist die *FileTable* prinzipiell ein

eigener Tabellentyp, deren Struktur bestimmten Vorgaben unterliegt. Eine einfache Anweisung zum Erstellen einer *FileTable* ist die folgende:

```
CREATE TABLE dbo.tblFileTable AS FILETABLE
  WITH
  (FILETABLE_DIRECTORY = 'tblFileTable', FILETABLE_COLLATE_FILENAME = database_default)
```

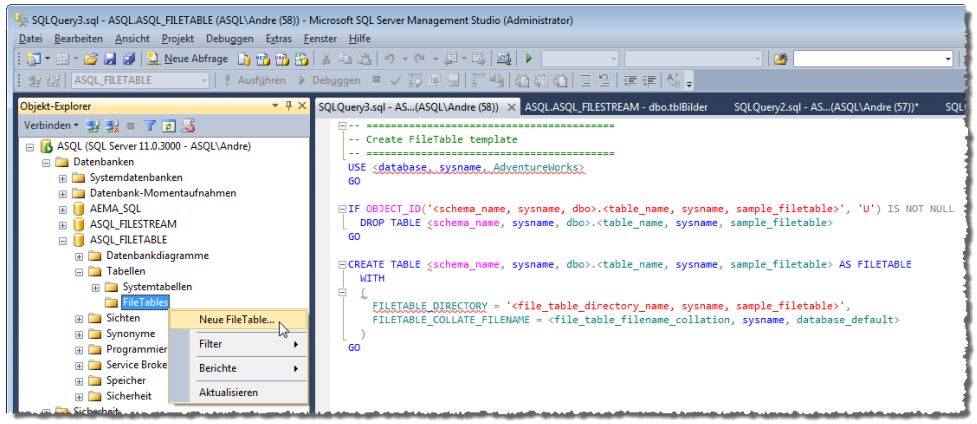


Abbildung 16.10: Erstellen einer neuen *FileTable*

Dabei geben Sie keine Spaltennamen an, sondern nur ein paar Eigenschaften. Die erste hier abgebildete heißt *FILETABLE_DIRECTORY* und gibt an, in welchem Verzeichnis unterhalb des Verzeichnisses für die aktuelle Datenbank die Dateien dieser Tabelle gespeichert werden sollen. Die zweite heißt *FILETABLE_COLLATE_FILENAME* und legt die Sortierreihenfolge für die Tabelle fest.

Nach dem Anlegen erscheint die Tabelle in dem für *FileTables* vorgesehenen Bereich im Objekt-Explorer (siehe Abbildung 16.10). Wie Sie dem Kontextmenü für diesen Eintrag entnehmen können, sehen die möglichen Aktionen etwas anders aus als für herkömmliche Tabelle – zum Beispiel fehlt die Möglichkeit zum Editieren der Tabelle.

Dies hat seinen Grund: Die Tabelle hat immer das gleiche Schema und Sie können keine Spalten entfernen, umbenennen oder hinzufügen. Interessant ist hingegen der Eintrag *FileTable-Verzeichnis durchsuchen*: Dieser Befehl öffnet den Windows Explorer mit dem Verzeichnis, das die Dateien dieser *FileTable* aufnimmt.

Verzeichnisstruktur

Noch ein Hinweis zur Verzeichnisstruktur: Diese beginnt immer mit der Freigabe für die Instanz des SQL Servers, die Sie in den Freigaben für den Rechner mit der entsprechenden Instanz finden (in diesem Fall *ASQL*). Die nächste Ebene trägt den auf Ebene der SQL Server-Instanz angegebenen Namen, hier also etwa *ASQL_FILESTREAM*. Die zweite Verzeichnisebene erhält den Namen des Verzeichnisses, das für die aktuelle Datenbank für die Eigenschaft *FILESTREAM-*

17 Access-SQL Server-Tools

Der Betrieb einer Access-Anwendung auf Basis einer SQL Server-Datenbank steht und fällt mit dem Zugriff auf die Daten der SQL Server-Datenbank. Dieses Kapitel zeigt verschiedene Möglichkeiten, von Access aus über die Objekttypen Tabelle oder Abfrage auf die Daten in den Tabellen einer SQL Server-Datenbank zuzugreifen.

Beispiele

Die Beispielformulare zum Verwalten von Verbindungen und zum Ausführen von SQL-Befehlen von Access aus sowie zum formulargesteuerten Herstellen von Verknüpfungen finden Sie in der Beispieldatenbank *17_AccessSQLTools.accdb*.

Das Add-In zum Erstellen von gespeicherten Prozeduren von Access aus kommt in Form der Datenbank *amvStoredProcedures.accda*.

17.1 Verbindungen per Formular verwalten

Wenn Sie häufiger per VBA oder per Pass-Through-Abfrage auf die Daten des SQL Servers zugreifen, ist es sinnvoll, die Verbindungsdaten in einer Tabelle zu speichern. Als Erstes legen Sie dazu zwei Tabellen an. Die erste speichert die Namen der verschiedenen Datenbanktreiber und heißt *tblTreiber*. Sie sieht im Entwurf wie in Abbildung 17.1 aus.

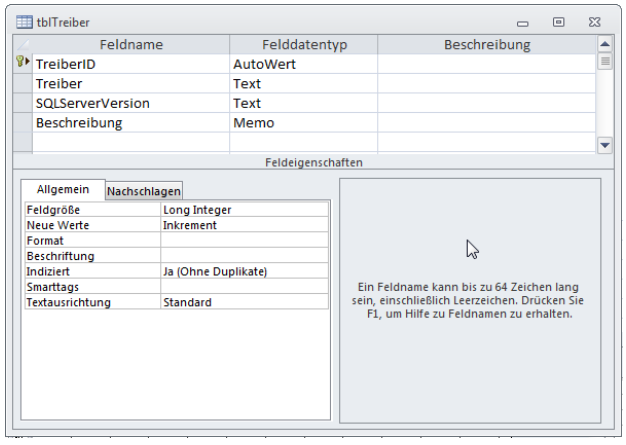
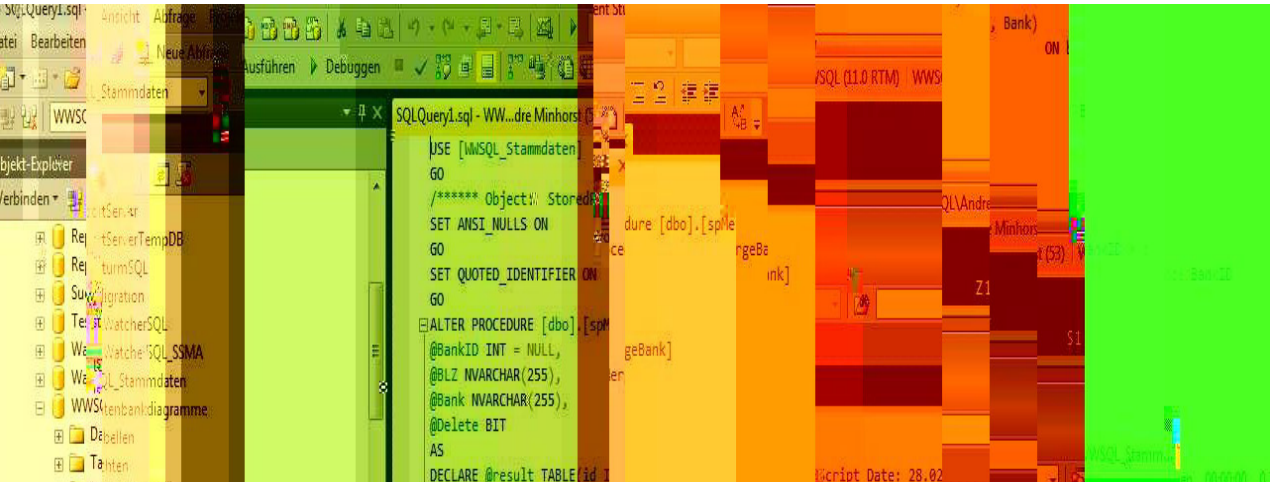


Abbildung 17.1: Tabelle zum Speichern der SQL Server-Versionen

Nach einem Wechsel in die Datenblattansicht tragen Sie dort die aktuellen Treiber ein (siehe Abbildung 17.2). Diese können Sie dann später komfortabel auswählen.

... DIE ÜBRIGEN SEITEN GIBT'S GEDRUCKT UNTER SHOP.MINHORST.COM



BERND JUNGBLUTH
ANDRÉ MINHORST

André Minhorst Verlag

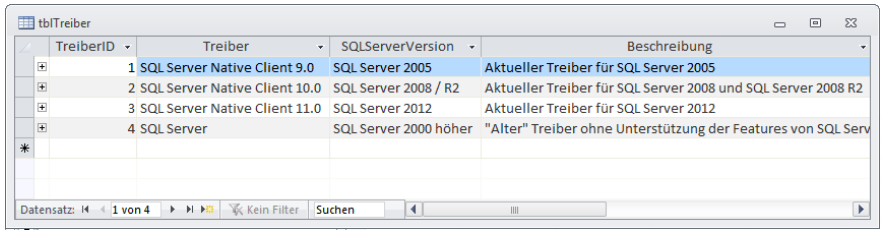


Abbildung 17.2: Aktuell verfügbare Treiber

Die eigentlichen Verbindungsdaten landen in der Tabelle *tblVerbindungszeichenfolgen*. Diese sieht im Entwurf wie in Abbildung 17.3 aus.

Das Feld *TreiberID* ist ein Fremdschlüsselfeld und dient zum Nachschlagen der Datensätze der Tabelle *tblTreiber*. Dabei zeigt das entsprechende Nachschlagefeld den Treiber an sowie in Klammern die SQL Server-Version.

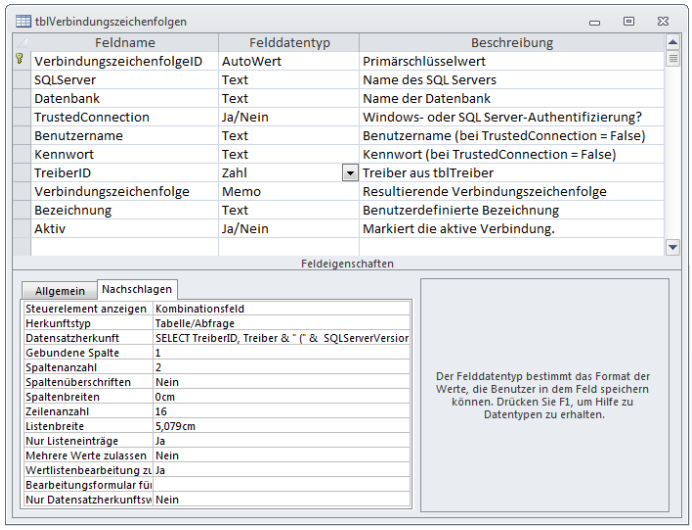


Abbildung 17.3: Tabelle zum Speichern der Verbindungszeichenfolgen und weiterer Informationen

Um die Daten dieser Tabelle komfortabel verwalten zu können, erstellen Sie ein Formular namens *frmVerbindungszeichenfolgen*. Dieses verwendet die Tabelle *tblVerbindungszeichenfolgen* als Datenherkunft.

Ziehen Sie alle Felder dieser Tabelle außer *ConnectionID* in den Detailbereich des Formulars. Fügen Sie dann noch ein Kombinationfeld zum Formularkopf hinzu. Dieses erhält die folgende Abfrage als Datensatzherkunft:

```
SELECT VerbindungszeichenfolgeID, Bezeichnung FROM tblVerbindungszeichenfolgen;
```

Wenn Sie die Eigenschaften *Spaltenanzahl* und *Spaltenbreiten* dieses Kombinationsfeldes auf die Werte *2* und *0cm* einstellen, blendet es die *ConnectionID* aus und zeigt den Namen des SQL Servers und der Datenbank der vorhandenen Verbindungszeichenfolgen an.

Wenn Sie noch eine Schaltfläche namens *cmdOK* zum Schließen des Formulars hinzufügen, sieht dieses wie in Abbildung 17.4 aus. Außerdem enthält das Formular noch eine Schaltfläche namens *cmdTesten*, mit der die aktuelle Verbindung getestet werden kann – mehr dazu weiter unten.

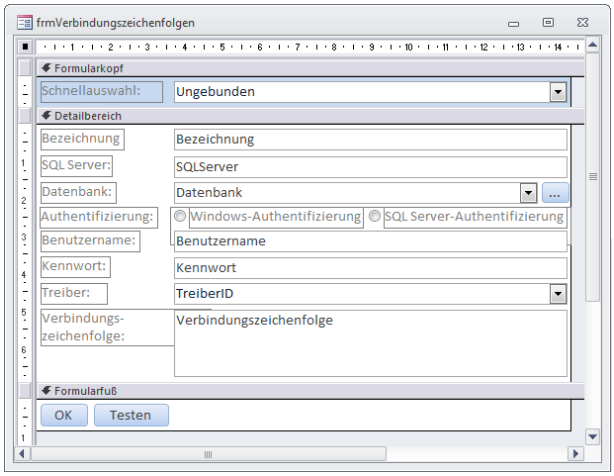


Abbildung 17.4: Formular zur Anzeige der Verbindungszeichenfolgen

Nun lassen sich mit diesem Formular gerade einmal ein paar Felder einer Tabelle verwalten. Etwas Komfort darf es dann doch bieten. Zunächst einmal wollen wir das Feld *TrustedConnection* etwas aussagekräftiger gestalten. Hat *TrustedConnection* den Wert *True*, soll die Windows-Authentifizierung verwendet werden, sonst die SQL Server-Authentifizierung. Also ersetzen wir das Kontrollkästchen durch eine kleine Optionsgruppe namens *ogrTrustedConnection*, stellen für die Eigenschaft *Steuerelementinhalt* den Wert *TrustedConnection* ein und fügen der Optionsgruppe zwei Optionsfelder mit den Bezeichnungen *Windows-Authentifizierung* (Optionswert *-1*) und *SQL Server-Authentifizierung* (Optionswert *0*) hinzu. Die Rahmenart der Optionsgruppe legen wir auf *Transparent* fest (siehe Abbildung 17.5).

Die beiden Textfelder *Benutzername* und *Kennwort* sollen nur aktiviert sein, wenn das Feld *TrustedConnection* den Wert *False* hat, also wenn die SQL Server-Authentifizierung verwendet werden soll. Deshalb legen Sie für das Ereignis *Nach Aktualisierung* der Optionsgruppe die folgende Ereignisprozedur an:

```
Private Sub ogrTrustedConnection_AfterUpdate()
    Me!txtBenutzername.Enabled = Not Me!ogrTrustedConnection
    Me!txtKennwort.Enabled = Not Me!ogrTrustedConnection
    If Me.Dirty Then
        Me!txtVerbindungszeichenfolge = VerbindungszeichenfolgeErmitteln( _
```

```
            Nz(Me!cboDatenbank, "[Datenbankname]"))
    End If
End Sub
```

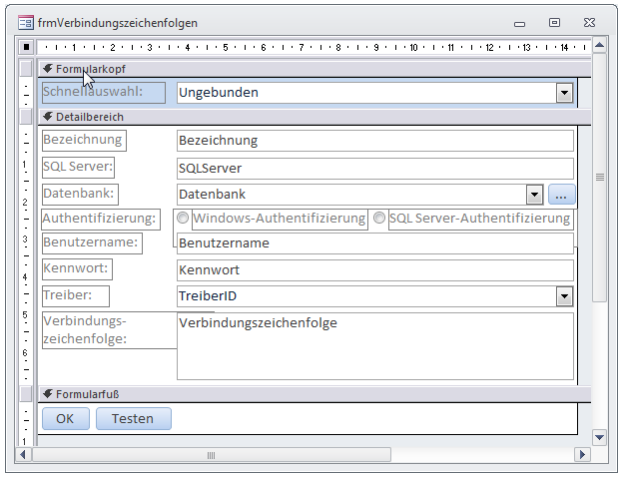


Abbildung 17.5: Festlegen der Authentifizierungsmethode per Optionsgruppe

Neben der Aktivierung beziehungsweise Deaktivierung der beiden Textfelder soll die Prozedur auch noch die Verbindungszeichenfolge aktualisieren. Dazu ruft sie die Funktion *Verbindungszeichenfolge* auf und weist die damit ermittelte Verbindungszeichenfolge dem Textfeld *txtVerbindungszeichenfolge* zu (diese Funktion schauen wir uns weiter unten an). Dies allerdings nur, wenn der Benutzer die Ereignisprozedur *ogrTrustedConnection_AfterUpdate* ausgelöst hat und der aktuelle Datensatz somit den Status *Dirty* besitzt. Die Prozedur wird nämlich auch noch zu einer weiteren Gelegenheit ausgelöst – nämlich dann, wenn das Formular den Datensatz anzeigt (also etwa beim Öffnen des Formulars oder beim Wechseln des Datensatzes). Dieser Aufruf erfolgt in der Ereignisprozedur, die durch das Ereignis *Beim Anzeigen* des Formulars ausgelöst wird:

```
Private Sub Form_Current()
    ogrTrustedConnection_AfterUpdate
    strSQLServer = Nz(Me!txtSQLServer)
    strBenutzername = Nz(Me!txtBenutzername)
    strKennwort = Nz(Me!txtKennwort)
End Sub
```

Die Prozedur speichert außerdem den Wert aus *txtSQLServer* in der Variablen *strSQLServer* sowie die Werte der Felder *txtBenutzername* und *txtKennwort* in den Variablen *strBenutzername* und *strKennwort*. Diese Variablen werden übrigens wie folgt im Kopf des Klassenmoduls *Form_frmVerbindungszeichenfolgen* deklariert:

```
Dim strSQLServer As String
Dim strKennwort As String
Dim strBenutzername As String
```

Noch etwas mehr Komfort würde das Formular bieten, wenn es alle verfügbaren SQL Server-Instanzen in einem Kombinationsfeld auflisten würde. Die Programmierung einer solchen Funktion ist jedoch sehr aufwendig und steht in keinem Verhältnis dazu, mal eben den Namen der gewünschten SQL Server-Instanz in ein Textfeld einzutragen. Wenn Sie den Namen des SQL Servers nicht kennen, geben Sie einfach im SQL Server Management Studio die folgende Abfrage ein:

```
SELECT @@SERVERNAME
```

Interessanter ist es da schon, bei bekannter SQL Server-Instanz die darin enthaltenen Datenbanken auszulesen und in einem Kombinationsfeld zur Auswahl bereitzustellen.

Die Datenbanken sollen erst eingelesen werden, wenn der Benutzer die neben dem Kombinationsfeld *cboDatenbank* befindliche Schaltfläche mit den drei Punkten (...) anklickt. Dies löst die folgende Prozedur aus:

```
Private Sub cmdDatenbankenEinlesen_Click()
    If Len(Nz(Me!txtSQLServer)) = 0 Then
        MsgBox "Bitte geben Sie den Namen der SQL Server-Instanz ein."
        Me!txtSQLServer.SetFocus
        Exit Sub
    End If
    If Len(Nz(Me!cboTreiberID)) = 0 Then
        MsgBox "Bitte wählen Sie einen Treiber aus."
        Me!cboTreiberID.SetFocus
        Exit Sub
    End If
    If Me!ogrTrustedConnection = False Then
        If Len(Nz(Me!txtBenutzername)) = 0 Then
            MsgBox "Bitte geben Sie einen Benutzernamen ein."
            Me!txtBenutzername.SetFocus
            Exit Sub
        End If
        If Len(Nz(Me!txtKennwort)) = 0 Then
            MsgBox "Bitte geben Sie ein Kennwort ein."
            Me!txtKennwort.SetFocus
            Exit Sub
        End If
    End If
    Set Me!cboDatenbank.Recordset = DatenbankenEinlesen
End Sub
```

Die Prozedur prüft zunächst, ob der Benutzer überhaupt den Namen der zu untersuchenden SQL Server-Instanz eingetragen hat und weist gegebenenfalls darauf hin. Außerdem prüft sie in dem Fall, dass die SQL Server-Authentifizierung aktiviert ist, ob Benutzername und Kennwort in die entsprechenden Textfelder eingetragen wurden.

Erst danach ruft sie eine Funktion namens *DatenbankenEinlesen* auf, die als Ergebnis ein Recordset zurückgibt. Dieses landet dann direkt in der Eigenschaft *Recordset* des Kombinationfeldes.

17.1.1 Datenbanken einer SQL Server-Instanz einlesen

Die folgende Funktion liefert die Namen aller Datenbanken des im Textfeld *txtSQLServer* angegebenen SQL Servers in einem Recordset zurück. Dazu ermittelt Sie zunächst die aktuelle Verbindungszeichenfolge, allerdings nicht für die im Kombinationsfeld *cboDatenbank* angegegebene Datenbank, sondern für die Datenbank *master*.

Dies setzt natürlich entsprechende Berechtigungen voraus. Die auszuführende Abfrage heißt *SELECT Name FROM sys.databases* und wird in einem neu erstellten *QueryDef*-Objekt unter Verwendung der zuvor ermittelten Verbindungszeichenfolge erstellt. Die Erstellung des *QueryDef*-Objekts übernimmt die Funktion *QueryDefErstellen* (siehe weiter unten). Nach dem Deaktivieren der Fehlerbehandlung ermittelt die Funktion durch Ausführen der *OpenRecordset*-Methode des *QueryDef*-Objekts das Ergebnis des QueryDefs. Dieses wird als Funktionswert von *DatenbankenEinlesen* an die aufrufende Routine zurückgegeben. Sollte ein Fehler beim Öffnen des Recordsets auftreten, erscheint eine entsprechende Meldung mit Angabe des Fehlers:

```
Private Function DatenbankenEinlesen() As Recordset
    Dim strSQL As String
    Dim qdf As DAO.QueryDef
    Dim strVerbindungszeichenfolge As String
    strVerbindungszeichenfolge = VerbindungszeichenfolgeErmitteln("master")
    strSQL = "SELECT Name FROM sys.databases;"
    Set qdf = QueryDefErstellen(strSQL, strVerbindungszeichenfolge)
    On Error Resume Next
    DoCmd.Hourglass True
    Set DatenbankenEinlesen = qdf.OpenRecordset
    DoCmd.Hourglass False
    If Not Err.Number = 0 Then
        MsgBox "Die Verbindung konnte nicht hergestellt werden:" & vbCrLf _
            & Err.Number & " " & Err.Description
    End If
End Function
```

17.1.2 Zusammenstellen der Verbindungszeichenfolge

Die Funktion *VerbindungszeichenfolgeErmitteln* stellt die Verbindungszeichenfolge für die aktuell in den Steuerelementen des Formulars angegebenen Informationen zusammen. Ein Element der Verbindungszeichenfolge wird jedoch per Parameter übergeben: der Name der zu verwenden den Datenbank. Das hat allein den Grund, dass die Funktion einmal zum Zusammenstellen der Verbindungszeichenfolge für die in den Formularfeldern angegebene Datenbank eingesetzt wird, aber auch zum Ermitteln der Zeichenfolge für die *master*-Datenbank (zum Einlesen der Datenbanken des aktuell ausgewählten SQL Servers). Die Funktion erstellt eine Zeichenkette, die zwei verschiedene Formate haben kann:

» erstens die für die Windows-Authentifizierung und

» zweitens die für die SQL Server-Authentifizierung.

In beiden Fällen stellt die Funktion zunächst eine Zeichenkette nach folgendem Schema zusammen:

```
ODBC;DRIVER={<Treiber>};SERVER=<Server>;DATABASE=<Datenbank>;
```

Diesem hängt die Funktion je nach Authentifizierungsmethode einen der folgenden Ausdrücke an:

» *Trusted_Connection=Yes* oder

» *UID=<Benutzername>;PWD=<Kennwort>*.

Im Falle der SQL Server-Authentifizierung muss der Benutzername auf jeden Fall angegeben werden, das Kennwort kann man freilassen. Es wird dann vom Treiber in einem weiteren Dialog dynamisch abgefragt. Sollte in diesem Fall der Benutzername fehlen, liefert die Funktion *VerbindungszeichenfolgeErmitteln* eine leere Zeichenkette zurück.

Das Ergebnis gibt die Funktion schließlich an die aufrufende Routine zurück:

```
Private Function VerbindungszeichenfolgeErmitteln(strDatenbank As String) As String
    Dim strTemp As String
    strTemp = "ODBC;DRIVER={" & Me!cboTreiberID.Column(1) & "};" _
        & "SERVER=" & Me!txtSQLServer & ";" & "DATABASE=" & strDatenbank & ";"
    If Me!ogrTrustedConnection = True Then
        strTemp = strTemp & "Trusted_Connection=Yes"
    Else
        If Len(Nz(Me!txtBenutzername)) > 0 Then
            strTemp = strTemp & "UID=" & Me!txtBenutzername & ";"
        Else
            MsgBox "Kein Benutzername angegeben."
            strTemp = ""
            Exit Function
        End If
        If Len(Nz(Me!txtKennwort)) > 0 Then
            strTemp = strTemp & "PWD=" & Me!txtKennwort
        End If
    End If
    VerbindungszeichenfolgeErmitteln = strTemp
End Function
```

17.1.3 Erstellen des QueryDef-Objekts

Die Funktion *QueryDefErstellen* erwartet eine SQL-Anweisung und eine Verbindungszeichenfolge als Parameter und liefert ein *QueryDef* zurück, das eine Pass-Through-Abfrage für die angegebenen Parameter enthält. Dabei füllt die Funktion zunächst die Variable *db* mit einem Verweis auf die aktuelle Datenbank-Sitzung. Danach erzeugt die sie ein neues *QueryDef*-Objekt und weist der Eigenschaft *Connect* die Verbindungszeichenfolge sowie die SQL-Anweisung zu. Die Abfrage soll Datensätze vom SQL Server zurückliefern, also stellt die Funktion die Eigenschaft *ReturnsRecords* auf den Wert *True* ein. Den Verweis auf das fertig konfigurierte *QueryDef*-Objekt liefert die Funktion als Funktionswert zurück:

```
Private Function QueryDefErstellen(strSQL As String, strVerbindungszeichenfolge _
        As String) As DAO.QueryDef
    Dim db As DAO.Database
    Dim qdf As DAO.QueryDef
    Set db = CurrentDb
    Set qdf = db.CreateQueryDef("")
    With qdf
        .Connect = strVerbindungszeichenfolge
        .SQL = strSQL
        .ReturnsRecords = True
    End With
    Set QueryDefErstellen = qdf
End Function
```

17.1.4 Treiber auswählen

Das Kombinationsfeld *cboTreiberID* stellt alle in der Tabelle *tblTreiber* gespeicherten Treiber zur Auswahl zur Verfügung (siehe Abbildung 17.6). Dazu verwendet das Kombinationsfeld die folgende Datensatzherkunft:

```
SELECT tblTreiber.TreiberID, tblTreiber.Treiber, Treiber & " (" & SQLServerVersion & ")"
AS Treibername FROM tblTreiber;
```

Das erste Feld ist der Primärschlüssel, das zweite enthält den Treiber-Ausdruck, wie er in der Verbindungszeichenfolge erscheinen soll und das dritte den Wert zur Anzeige im Kombinationsfeld.

Damit die ersten beiden Spalten nicht im Kombinationfeld erscheinen, stellen Sie die beiden Eigenschaften *Spaltenanzahl* und *Spaltenbreiten* auf die Werte *3* und *0cm;0cm* ein. Wenn der Benutzer einen Treiber auswählt, soll die Verbindungszeichenfolge aktualisiert und angezeigt werden. Dazu löst das Kombinationsfeld das Ereignis *Nach Aktualisierung* aus, was folgende Prozedur nach sich zieht:

```
Private Sub cboTreiberID_AfterUpdate()
    Me!txtVerbindungszeichenfolge = VerbindungszeichenfolgeErmitteln(Me!cboDatenbank)
End Sub
```

Dies ruft wiederum die Funktion *VerbindungszeichenfolgeErmitteln* auf den Plan, die Sie ja bereits weiter oben kennengelernt haben.

17.1.5 Verbindungszeichenfolge testen

Fehlt noch der Test der ermittelten Verbindungszeichenfolge. Diesen führen Sie mit einem Mausklick auf die Schaltfläche *cmdTesten* aus. Die dazugehörige Ereignisprozedur startet mit der folgenden Deklarationszeile:

```
Private Sub cmdTesten_Click()
    Dim strVerbindungszeichenfolge As String
```

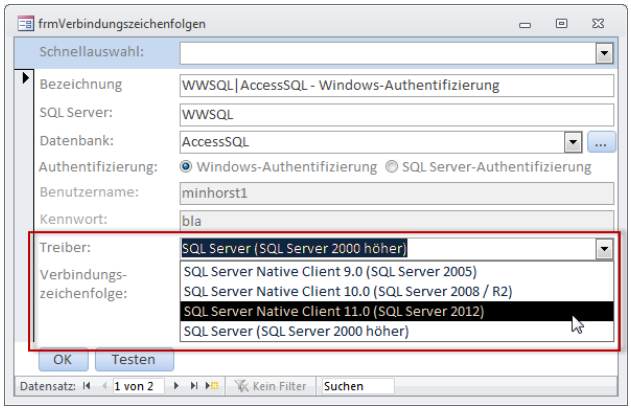


Abbildung 17.6: Auswahl des Treibers für die Verbindungszeichenfolge

Dann prüft sie, ob der Benutzer eine Datenbank ausgewählt hat und bricht die Prozedur gegebenenfalls mit einer entsprechenden Meldung ab:

```
If Len(Me!cboDatenbank) = 0 Then
    MsgBox "Keine Datenbank ausgewählt."
    Me!cboDatenbank.SetFocus
    Exit Sub
End If
```

Ist eine Datenbank vorhanden, speichert die Prozedur den aktuellen Datensatz und ruft die Funktion *VerbindungTesten* auf (siehe Abschnitt *»Verbindung und Zugriffsdaten prüfen«, Seite 310*). Dieser übergibt sie die ID der zu testenden Verbindung und die Variable *strVerbindungszeichenfolge* als Rückgabeparameter. Je nach Ergebnis erscheint eine entsprechende Meldung:

```
Me.Dirty = False
If VerbindungTesten(Me!VerbindungszeichenfolgeID, strVerbindungszeichenfolge) = _
                                                                    True Then
    MsgBox "Die Verbindung wurde erfolgreich hergestellt."
Else
    MsgBox "Die Verbindung konnte nicht hergestellt werden."
End If
```

Die folgende Ereignisprozedur wird vor dem Aktualisieren des Formulars ausgelöst und aktualisiert jeweils die Verbindungszeichenfolge auf Basis der aktuell gespeicherten Daten:

```
Private Sub Form_BeforeUpdate(Cancel As Integer)
    Me!txtVerbindungszeichenfolge = _
        VerbindungszeichenfolgeErmitteln(Nz(Me!cboDatenbank, "[Datenbankname]"))
End Sub
```

Ändert der Benutzer den Inhalt des Feldes *txtSQLServer*, soll sich diese Änderung immer direkt auf die im Feld *txtVerbindungszeichenfolge* angezeigte Verbindungszeichenfolge auswirken. Dazu speichert die Prozedur den aktuellen Inhalt des Textfeldes (ermittelt mit der Eigenschaft

Text – nicht zu verwechseln mit *Value*, dem gespeicherten Wert) in der Variablen *strSQLServer* und ruft die Prozedur *VerbindungszeichenfolgeErstellen* erneut auf, um diese mit dem neuen SQL Server zu aktualisieren:

```
Private Sub txtSQLServer_Change()
    strSQLServer = Me!txtSQLServer.Text
    strBenutzername = Nz(Me!txtBenutzername)
    strKennwort = Nz(Me!txtKennwort)
    Me!txtVerbindungszeichenfolge = _
        VerbindungszeichenfolgeErmitteln(Nz(Me!cboDatenbank, "[Datenbankname]"))
End Sub
```

Parallel schreibt die Prozedur auch die aktuellen Werte der beiden Textfelder *txtBenutzername* und *txtKennwort* in die entsprechenden Variablen, damit diese vor dem Ermitteln der Verbindungszeichenfolge aktualisiert werden.

Die beiden Textfelder lösen übrigens ähnliche Ereignisprozeduren aus, wenn das Ereignis *Bei Änderungen* eintritt. So ist der im Textfeld txtVerbindungszeichenfolge angezeigt Ausdruck jederzeit auf dem aktuellen Stand.

Schließlich fehlt noch das Kombinationsfeld *cboSchnellauswahl*, das nach dem Auswählen eines neuen Eintrags die entsprechenden Verbindungsdaten im Formular anzeigen soll:

```
Private Sub cboSchnellauswahl_AfterUpdate()
    Me.Recordset.FindFirst "VerbindungszeichenfolgeID = " & Me!cboSchnellauswahl
End Sub
```

Die Schaltfläche *cmdOK* schließt den Dialog.

17.2 Befehle per Formular ausführen

Wenn Sie eine Access-Anwendung mit SQL Server-Backend entwickeln, wird es Ihnen mit der Zeit unpraktisch erscheinen, immer zwischen den beiden Anwendungen hin- und herwechseln zu müssen – erst recht, wenn sich beide nicht auf dem gleichen Rechner befinden (gut – virtuelle Maschinen und Remote-Zugriff sorgen zumindest dafür, dass Sie nicht zwischen den verschiedenen Rechnern hin- und herlaufen müssen).

Wie wäre es also, wenn Sie den größten Teil der Arbeit innerhalb nur einer der beiden Anwendungen durchführen könnten? Da das Editieren von Formularen, Berichten und VBA-Code über das SQL Server Management Studio nicht möglich ist, wählen wir Access als Plattform für diesen Weg. Welche Arbeitsschritte genau sollen wir von Access aus durchführen? Nun, da gibt es eine ganze Reihe:

» Durchführen einer Ad-Hoc-Abfrage auf Basis einer der im SQL Server gespeicherten Tabellen oder anderer Objekte wie gespeicherter Prozeduren

» Ändern der Daten in den Tabellen einer SQL Server-Datenbank

» Erstellen, Bearbeiten und Löschen von Tabellen, Feldern und Beziehungen

» Durchführen anderer Abfragen, beispielsweise um alle Tabellen der aktuellen Datenbank zu ermitteln

Sie sehen: Da kommen bereits einige Anforderungen zusammen. Und damit Sie diese Aufgaben auch alle erledigen können, finden Sie in den vorherigen Kapiteln die Grundlagen zum Abfragen und Ändern von Daten sowie zum Ändern des Datenmodells.

Nun aber zunächst zum notwendigen Werkzeug. Dieses sieht wie in Abbildung 17.7 aus und ermöglicht die Eingabe und das Ausführen von SQL-Anweisungen. In diesem Beispiel wird die in das Feld *SQL* eingetragene und markierte *SELECT*-Anweisung ausgeführt, das Unterformular im unteren Bereich zeigt das Ergebnis der Abfrage an.

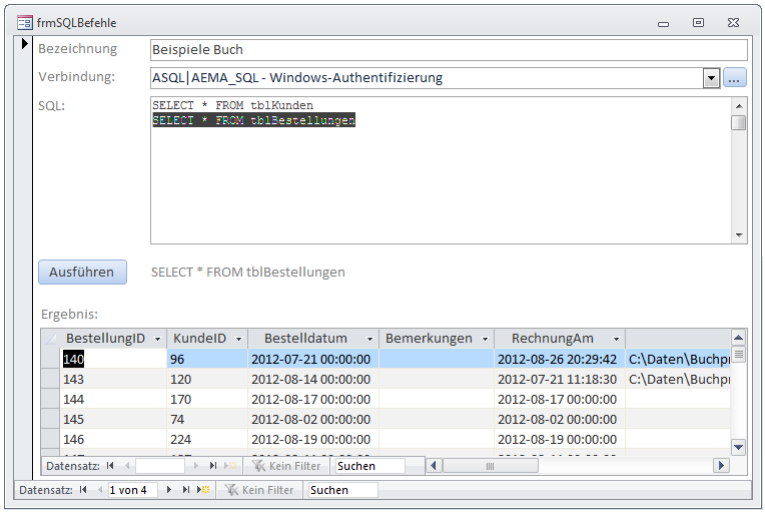


Abbildung 17.7: Formular für den direkten Zugriff auf den SQL Server

Wie arbeiten Sie mit diesem Formular? Wie Sie der Abbildung entnehmen können, bietet das Formular ein Feld namens *Bezeichnung*. Dies und die typischen Elemente gebundener Formulare wie der Datensatzmarkierer sowie der Navigationsbereich deuten darauf hin, dass Sie die in die Steuerelemente des Formulars eingegebenen Daten speichern und wiederverwenden können.

Im Wesentlichen handelt es sich dabei um drei Datenfelder: eine Bezeichnung des aktuellen Satzes von SQL-Anweisungen, die Verbindung zur SQL Server-Datenbank, in der sich die betroffenen Tabellen und weitere Datenbankobjekte befinden sowie die auszuführenden SQL-Anweisungen.

Um einen Server und eine Datenbank beziehungsweise die passende Verbindungszeichenfolge festzulegen, verwenden Sie das weiter oben erstellte Formular *frmVerbindungszeichenfolgen*.

Klicken Sie auf die Schaltfläche rechts neben dem Kombinationsfeld, öffnet sich dieses Formular. Die dort zuletzt bearbeitete Verbindungszeichenfolge wird beim Schließen des Formulars in das Kombinationsfeld mit der Beschriftung *Verbindung* eingetragen (siehe Abbildung 17.8).

Neben einfachen *SELECT*-Anweisungen können Sie auch Aktionsabfragen mit *INSERT INTO*, *UPDATE* und *DELETE* einsetzen. Tragen Sie diese einfach in das Textfeld ein, markieren Sie die auszuführende Abfrage und betätigen Sie die Taste *F5* oder die Schaltfläche mit der Beschriftung *Ausführen*. In diesem Fall zeigt das Unterformular die Anzahl der von der Änderung betroffenen Datensätze an (siehe Abbildung 17.9).

Und wenn Sie einmal eine fehlerhafte SQL-Anweisung eingeben wie in Abbildung 17.10, dann liefert das Unterformular sogar die Fehlermeldungen.

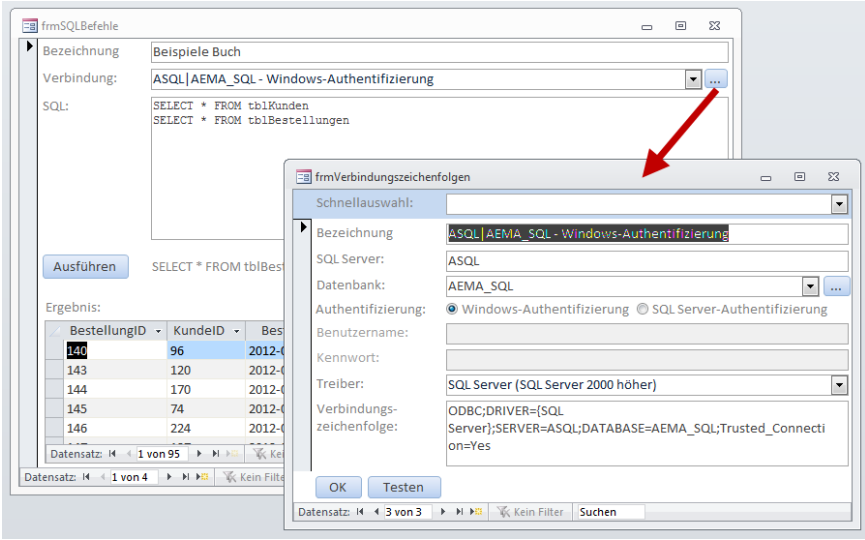


Abbildung 17.8: Die Verbindungszeichenfolge stellen Sie mit dem zuvor erstellten Formular zusammen.

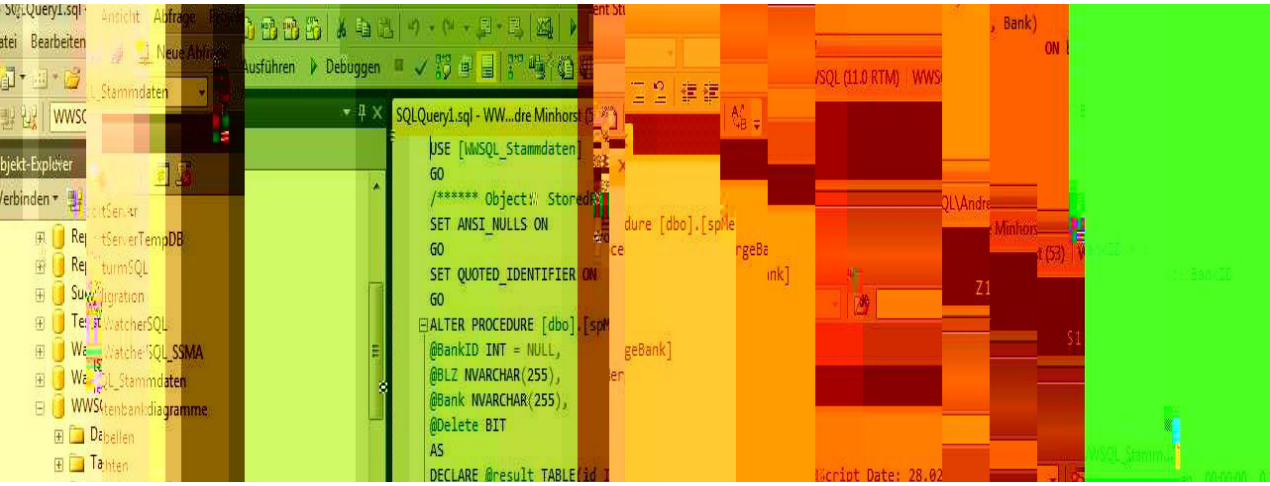
Wichtig ist, dass Sie die auszuführende Anweisung jeweils mit der Maus markieren und diese erst dann mit *F5* oder der Schaltfläche *Ausführen* starten.

17.2.1 Aufbau des Formulars

Das Formular sieht in der Entwurfsansicht wie in Abbildung 17.11 aus. Es verwendet die Tabelle *tblSQLBefehle* als Datenherkunft. Diese Tabelle ist wie in Abbildung 17.12 aufgebaut. Das Feld *SQLBefehl* soll mehr als 255 Zeichen aufnehmen können und wird deshalb als Memofeld ausgelegt.

Das Feld *Verbindung* ist ein Nachschlagefeld, das die Verbindungszeichenfolgen aus der Tabelle *tblVerbindungszeichenfolgen* zur Auswahl anbietet, und zwar in der Form *<SQL Server>|<Daten-*

... DIE ÜBRIGEN SEITEN GIBT'S GEDRUCKT UNTER SHOP.MINHORST.COM



BERND JUNGBLUTH
ANDRÉ MINHORST

André Minhorst Verlag

18 Sichern und Wiederherstellen

Die Sicherungsstrategie einer reinen Access-Lösung ist schnell besprochen: Bei einer einzigen Datenbank erstellen Sie regelmäßig Sicherungskopien der kompletten Datenbank und bei Frontend-Backend-Lösungen sichern Sie das auf dem Server liegende Backend in einem wiederkehrenden Turnus. Man könnte sich noch differenzierte Methoden ausdenken, aber letztlich bleiben nicht allzuviele Möglichkeiten. Bei einem SQL Server-Backend sieht dies ganz anders aus. Die Dateien einer Datenbank einfach wegspeichern? Geht nicht. Der Zugriff auf diese Dateien ist eingeschränkt, da der SQL Server permanent darauf zugreift.

Dafür aber bietet SQL Server eigene Funktionen zum Erstellen von Datenbanksicherungen – und die lassen sich auch im laufenden Betrieb nutzen. Natürlich gibt es ebenso entsprechende Funktionen zur Wiederherstellung gesicherter Datenbanken.

Dieses Kapitel beleuchtet beide Seiten und liefert Beispiele für das Anlegen von Sicherungen sowie für die anschließende Wiederherstellung.

18.1 Sicherungstypen

Der SQL Server bietet zwei Sicherungstypen an:

» Die Vollsicherung sichert immer die komplette Datenbank. Dies umfasst alle Datendateien mit den Dateierweiterungen *mdf* und *ndf*, sowie die Transaktionsprotokolldateien mit der Dateierweiterung *ldf*.

» Die differenzielle Sicherung sichert nur die Änderungen seit der letzten Vollsicherung beziehungsweise der letzten differenziellen Sicherung.

Je nach Wiederherstellungsmodell einer Datenbank können die beiden Sicherungstypen um eine Transaktionsprotokollsicherung ergänzt werden. Das Transaktionsprotokoll übernimmt im SQL Server eine wichtige Funktion. Jede Änderung an den Daten einer SQL Server-Datenbank wird zunächst im Transaktionsprotokoll gespeichert und erst dann ausgeführt. Nach erfolgreicher Ausführung markiert der SQL Server die Änderung im Transaktionsprotokoll als erledigt. Sollte die Ausführung fehlschlagen, kann der SQL Server aufgrund des Protokolls alle Änderungen der Transaktion rückgängig machen. Mehr zum Transaktionsprotokoll lesen Sie im Kapitel »FAQ«, Seite 19.

Wie Sie sich vorstellen können, wächst die Transaktionprotokolldatei sehr schnell. Um dies zu vermeiden, sollten die Einträge der bereits erledigten Transaktionen regelmäßig aus dem Transaktionsprotokoll entfernt werden. Dies erfolgt über eine Transaktionsprotokollsicherung. Sind die Einträge gesichert, können sie problemlos gelöscht werden. Schließlich lässt sich der Stand der letzten Sicherung jederzeit wiederherstellen.

Womit wir auch direkt beim Vorteil einer Transaktionsprotokollsicherung wären. Da hierbei nur die noch nicht gesicherten Einträge des Transaktionsprotokolls gesichert werden, ist der Umfang der Sicherung gering und somit auch nicht zeitaufwendig. Das Sichern des Transaktionsprotokolls kann also öfter erfolgen als eine Vollsicherung oder eine differenzielle Sicherung – zum Beispiel alle fünf Minuten.

Ergänzen Sie die Vollsicherung oder eine Kombination von Vollsicherung und differenzieller Sicherung um eine Transaktionsprotokollsicherung, verlieren Sie bei einem Crash maximal die Änderungen am Datenbestand des letzten Intervalls zwischen zwei Sicherungen des Transaktionsprotokolls. Bei einer Transaktionsprotokollsicherung alle fünf Minuten wären dies die Datenänderungen von maximal 5 Minuten. Wohingegen der Datenverlust einer Vollsicherung oder auch einer differenziellen Sicherung je nach Zeitplan der Sicherungen und Eintreten des Crashs die Datenänderungen von mehreren Stunden enthalten kann. Ein Volumen, dass sich in den seltensten Fällen rekonstruieren lässt.

Es geht sogar noch weiter, denn gegebenenfalls ist es bei einem Crash sogar möglich, das Transaktionsprotokoll manuell zu sichern – in diesem Fall retten Sie sogar alle Daten bis zum Crash.

Eine Transaktionsprotokollsicherung dient jedoch nicht nur zur Datensicherung. Sie sorgt durch das Entfernen der Einträge erledigter Transaktionen auch dafür, dass das Transaktionsprotokoll nicht unnötig wächst und dabei mehr Festplattenplatz als notwendig belegt. Sie sollten also je nach Wiederherstellungsmodell der Datenbank eine regelmäßige Transaktionsprotokollsicherung durchführen, auch wenn Sie diese zum Wiederherstellen einer Datenbank nicht benötigen. Ohne eine Transaktionsprotokollsicherung werden die Einträge erledigter Transaktionen nicht gelöscht und die Transaktionsprotokolldatei stetig vergrößert – bis es keinen freien Platz mehr auf der Festplatte gibt.

18.2 Wiederherstellungsmodelle

Die Möglichkeit beziehungsweise Notwendigkeit einer Transaktionsprotokollsicherung ist abhängig vom Wiederherstellungsmodell der Datenbank. SQL Server bietet drei Wiederherstellungsmodelle:

» *Vollständig*: Protokolliert alle Transaktionen im Transaktionsprotokoll. Transaktionsprotokollsicherungen sind möglich und somit auch das Wiederherstellen einer Datenbank bis zur letzten Transaktionsprotokollsicherung – wie auch das Wiederherstellen einer Datenbank bis zu einem bestimmten Sicherungszeitpunkt.

» *Einfach*: Protokolliert alle Transaktionen im Transaktionsprotokoll, wobei die Einträge erledigter Transaktionen automatisch entfernt werden. Transaktionsprotokollsicherungen sind nicht möglich. Eine Datenbank lässt sich somit nur mit den Daten der letzten differenziellen Sicherung oder der letzten Vollsicherung wiederherstellen.

» *Massenprotokolliert*: Protokolliert alle Transaktionen im Transaktionsprotokoll, mit Ausnahme von Massenvorgängen wie *CREATE INDEX* und Datenimporte per *BULK INSERT*. Transaktionsprotokollsicherungen sind möglich und somit auch das Wiederherstellen einer Datenbank bis zur letzten Transaktionsprotokollsicherung. Dabei werden die nicht protokollierten Massenvorgänge logischerweise nicht wiederhergestellt, da sie in der Sicherung nicht enthalten sind. Die Massenvorgängee müssen nach der Wiederherstellung manuell wiederholt werden.

Das Wiederherstellungsmodell legen Sie in den Eigenschaften der jeweiligen Datenbank fest (siehe Abbildung 18.1).

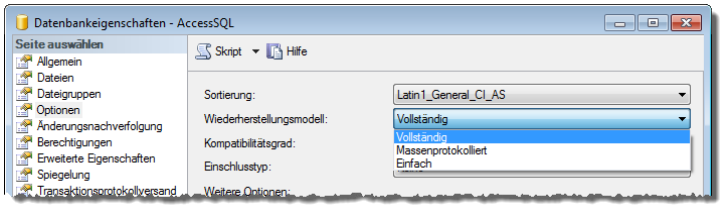


Abbildung 18.1: Einstellen des Wiederherstellungsmodells in den Optionen einer Datenbank

Wann ist nun eine Transaktionsprotokollsicherung erforderlich? Bei Datenbanken mit dem Wiederherstellungsmodell *Vollständig* oder *Massenprotokolliert*. Hier sorgt die Transaktionsprotokollsicherung für das Entfernen der Einträge erledigter Transaktionen und verhindert somit ein unnötiges Anwachsen des Transaktionsprotokolls und der zugehörigen Datei.

18.3 Welcher Sicherungstyp ist der Richtige?

Eine Vollsicherung ist die Grundlage für jegliche Wiederherstellung einer Datenbank. Zusätzlich können Sie die differenzielle Sicherung und/oder die Transaktionprotokollsicherung einsetzen.

Der Unterschied zwischen einer Vollsicherung und einer differenziellen Sicherung besteht darin, dass bei der Vollsicherung der aktuelle Zustand der Datenbank gespeichert wird. Bei der differenziellen Sicherung speichert der SQL Server nur die Änderungen seit der letzten Vollsicherung beziehungsweise der letzten differenziellen Sicherung.

Warum führt man nicht einfach regelmäßig Vollsicherungen durch? Es spricht eigentlich nichts dagegen, solange die Datenbank klein ist und die Sicherung dementsprechend in kurzer Zeit erledigt wird. Bei größeren Datenbanken kostet dies jedoch Performance und verschlechtert die Antwortzeiten der Datenbank.

Aus diesem Grund werden Vollsicherungen meist zu Zeiten geringer Auslastung durchgeführt – etwa nachts. Reicht die Vollsicherung nicht aus, führt man ein- oder mehrmals täglich differenzielle Sicherungen durch. Gegebenenfalls ist eine Datenbank so groß, dass eine Vollsicherung

nicht in das Zeitfenster geringer Auslastung hineinpasst – dann verschiebt man diese beispielsweise auf das Wochenende und führt unter der Woche regelmäßig differenzielle Sicherungen aus.

Andersherum: Warum erstellt man nicht zu Beginn eine Vollsicherung und legt danach nur noch differenzielle Sicherungen an? Ganz einfach: Weil jede differenzielle Sicherung die Änderungen der Daten seit der vorherigen Vollsicherung beziehungsweise differenziellen Sicherung erfasst. Dementsprechend muss man bei der Wiederherstellung einer Datenbank auch alle seit der letzten Vollsicherung durchgeführten differenziellen Sicherungen wiederherstellen, was je nach Anzahl der differenziellen Sicherungen sehr aufwendig sein kann. Und das wirkt sich wiederum auf den Zeitraum aus, in dem die Datenbank nach einem Crash nicht verfügbar ist.

Wer es sich wegen der geringen Größe der Datenbank leisten kann, erstellt also möglichst oft Vollsicherungen und ergänzt diese eventuell mit differenziellen Sicherungen. Je größer die Datenbank wird, desto mehr wird man auf differenzielle Sicherungen bauen – was zugunsten der Performance zur Laufzeit ausfällt, aber zu Ungunsten der für die Wiederherstellung benötigte Zeit.

Bei den Wiederherstellungsmodellen *Vollständig* oder *Massenprotokolliert* lassen sich beide Sicherungstypen mit der Transaktionsprotokollsicherung ergänzen. Wie bereits erwähnt, sichert eine Transaktionsprotokollsicherung lediglich die Einträge im Transaktionsprotokoll, die seit der letzten Vollsicherung oder der letzten differenziellen Sicherung eingetragen wurden.

Der Vorteil der Transaktionsprotokollsicherung liegt in ihrer Größe und somit auch in der guten Performance. Eine Transaktionsprotokollsicherung hat so gut wie keine Auswirkungen auf die Performance der Datenbank, bietet aber im Falle eines Crashs die Möglichkeit, die Datenbank mit geringem Datenverlust wiederherzustellen.

Welche Sicherungstypen Sie für Ihre Datenbank verwenden, hängt letztendlich von der Datenbankgröße und dem erlaubten Datenverlust ab. Folgende Konstellationen sind denkbar:

» *Vollsicherungen in kurzen Abständen:* Zum Beispiel stündliche Vollsicherungen – empfehlenswert nur bei kleinen Datenbanken.

» *Vollsicherungen in längeren Abständen plus mehrere differenzielle Sicherungen in mittleren Abständen:* Zum Beispiel eine Vollsicherung in der Nacht plus eine differenzielle Sicherung am Mittag oder eine Vollsicherung am Wochenende und täglich eine differenzielle Sicherung.

» *Vollsicherungen in längeren Abständen plus mehrere differenzielle Sicherungen in mittleren Abständen ergänzt mit Transaktionsprotokollsicherungen in kurzen Abständen:* Zum Beispiel eine Vollsicherung am Wochenende plus eine tägliche differenzielle Sicherung ergänzt mit Transaktionsprotokollsicherungen im Abstand von fünf Minuten.

» *Vollsicherungen in längeren Abständen ergänzt mit Transaktionsprotokollsicherungen in kurzen Abständen:* Zum Beispiel eine Vollsicherung in der Nacht ergänzt mit einer Transaktionsprotokollsicherung alle 5 Minuten.

18.4 Beispiele zum Erstellen einer Sicherung

In den folgenden Abschnitten konfigurieren wir Vollsicherungen, differenzielle Sicherungen sowie Transaktionsprotokollsicherungen und führen diese aus.

Als Beispiel für die Sicherungen verwenden wir die Datenbank *AEMA_SQL*. Wenn Sie die nachfolgenden Beispiele reproduzieren möchten, prüfen Sie zunächst die Option *Wiederherstellungsmodell* der Datenbank. Diese muss den Wert *Vollständig* enthalten (siehe oben).

18.5 Erstellen einer Vollsicherung

Die Vollsicherung ist die Grundlage für das Sichern und Wiederherstellen. Ohne eine Vollsicherung sind differenzielle Sicherungen und Transaktionsprotokollsicherungen nutzlos. Um eine Vollsicherung durchzuführen, wählen Sie im SQL Server Management Studio aus dem Kontextmenü der Datenbank den Eintrag *Tasks|Sichern...* aus (siehe Abbildung 18.2).

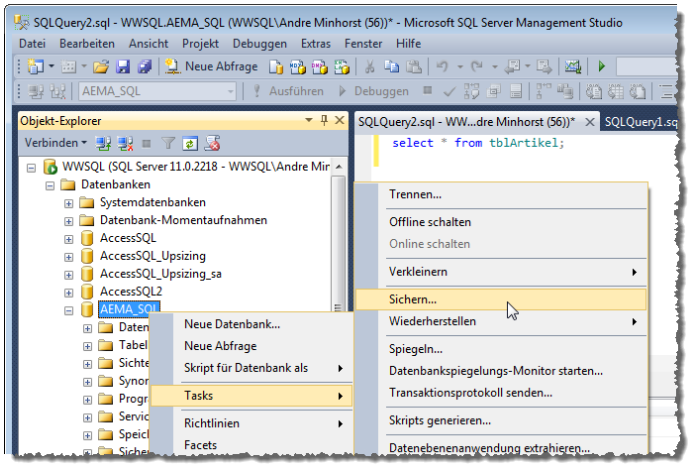


Abbildung 18.2: Öffnen des Dialogs zum Sichern einer Datenbank

Im nun erscheinenden Dialog prüfen Sie, ob die Option *Sicherungstyp* auf *Vollständig* eingestellt ist (siehe Abbildung 18.3). Wir wollen die komplette Datenbank sichern, also behalten wir den Wert *Datenbank* für die Eigenschaft *Sicherungskomponente* bei. Größere Datenbanken bestehen unter Umständen aus mehreren Datenbank- und Transaktionsprotokolldateien. Mit der Option *Dateien und Dateigruppen* lässt sich definieren, welche der Dateien gesichert werden sollen.

Unter *Sicherungssatz* geben Sie einen Namen und eine Beschreibung an. Der Sicherungssatz enthält die eigentliche Sicherung sowie Informationen über die Sicherung selbst. Im Gegensatz

dazu steht der Mediensatz. Hierbei handelt sich in der Regel um die Datei(en), welche einen oder mehrere Sicherungssätze aufnehmen.

Mit der Option *Sicherungssatz läuft ab* definieren Sie, ab wann ein Sicherungssatz überschrieben werden darf. Hier stellen Sie entweder ein Alter in Tagen ein oder ein fixes Datum. Dies macht vor Allem dann Sinn, wenn sie mehrere Sicherheitssätze in einer einzigen Datei speichern – diese Einstellung nehmen Sie auf der Seite *Optionen* des Dialogs *Datenbank sichern* vor.

Ganz unten im Dialog legen Sie schließlich noch fest, wohin die Sicherung gespeichert werden soll. Dabei wählt man zunächst zwischen Festplatte und Band aus (vorausgesetzt, es ist ein Bandlaufwerk vorhanden). Letzteres ist eine schlechte Alternative, da die Speicherung auf Festplatte wesentlich schneller erfolgt.

Sinnvoll wäre es, wenn Sie überhaupt Bandsicherungen durchführen möchten, zunächst Sicherungen in Dateien zu erstellen und diese dann über eine externe Software auf Band zu sichern. Diese Aktion schlägt sich dann nicht direkt auf die Performance des SQL Servers nieder.

Als Ziel der Sicherung verwenden Sie entweder die standardmäßig eingestellte Datei oder definieren eine neue. Selbst wenn Sie nur den Dateinamen ändern möchten, müssen Sie den Eintrag zur aktuellen Zieldatei mit der Schaltfläche *Entfernen* löschen und mit *Hinzufügen...* eine neue auswählen beziehungsweise anlegen.

Grundsätzlich ist es empfehlenswert, die Datei für die Sicherung in ein anderes Laufwerk zu speichern wie die Datenbankdateien – besser noch auf eine externe Festplatte oder einen anderen Rechner. Und dies nicht nur aus Gründen der Ausfallsicherheit, sondern auch aus Gründen der Performance.

Möchten Sie die Sicherung auf einem Netzlaufwerk speichern, geben Sie einfach einen UNC-Pfad in der Form *Rechner**Freigabe**Verzeichnis**Sicherung.bak* an. Bei Sicherungen auf ein Netzlaufwerk sollte die Option *Sicherung nach dem Abschluss überprüfen* auf der Seite *Optionen* aktiviert sein.

Über die Schaltfläche *Hinzufügen ...* können Sie für die Sicherung weitere Ziele angeben. Die Sicherung wird dann gleichmäßig über alle angegebenen Ziele verteilt. Sinn und Zweck dieser Möglichkeit ist wie so oft die Verbesserung der Performance. Durch das Schreiben der Sicherung auf mehrere Dateien wird eine bessere Performance bei der Datenbanksicherung erzielt.

Ob in einer der angegebenen Zieldateien Sicherungssätze enthalten sind, sehen Sie über die Schaltfläche *Inhalt*. Diese öffnet den Dialog aus Abbildung 18.4 mit den Informationen zu den einzelnen Sicherungssätzen der aktuell markierten Datei.

Schließlich legen Sie auf der Seite *Optionen* fest, wie mit den Sicherungssätzen verfahren werden soll (siehe Abbildung 18.5). Sie können jede Sicherung in einer eigenen Datei speichern oder auch mehrere Sicherungen in einer Datei zusammenfassen – die Art der Sicherung (Vollsicherung, differenzielle Sicherung oder Transaktionsprotokollsicherung) ist dabei unerheblich. Ebenso ist es möglich, die Sicherung mehrerer Tage in einer Datei zu speichern.

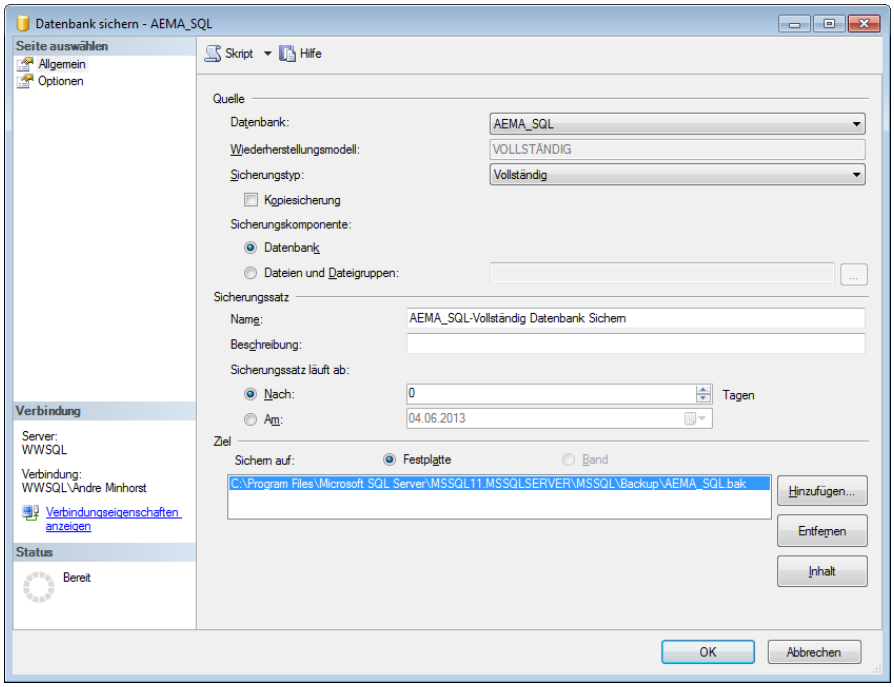


Abbildung 18.3: Einstellen der Eigenschaften für eine Sicherung

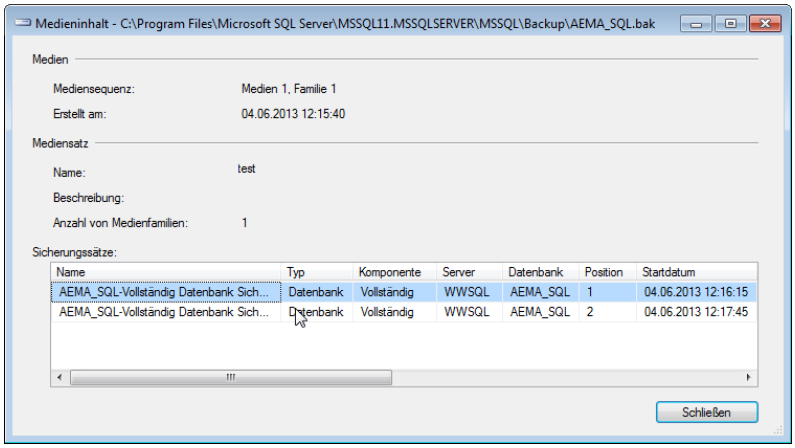


Abbildung 18.4: Inhalt einer Sicherungsdatei

Der Vorteil ist, dass eine Sicherung aus einer Datei schnell wiederhergestellt werden kann. Der Nachteil besteht darin, dass eine Beschädigung einer Datei mit mehreren Sicherungssätzen den kompletten Inhalt unbrauchbar macht – das Gleiche gilt natürlich für das versehentliche Löschen.

Durch die Einstellung *Auf vorhandenen Mediensatz sichern|An vorhandenen Sicherungssatz anfügen* wird jede Sicherung in die gleiche Datei gespeichert.

Um dies zu prüfen, führen Sie einfach mehrmals hintereinander eine Vollsicherung durch und betrachten die Dateigröße der Sicherungsdatei (beispielsweise *C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\Backup\AEMA_SQL.bak*).

Mit der Option *Auf vorhandenen Mediensatz sichern|Alle vorhandenen Sicherungssätze überschreiben* überschreiben Sie die vorhandenen Sicherungen in der Datei.

Die Option *Auf neuen Mediensatz sichern und alle vorhandenen Sicherungssätze löschen* sorgt dafür, dass ebenfalls die unter *Ziele* genannten Datei(en) für die Sicherung genutzt werden. Allerdings leert dies die Zieldatei(en) und vergibt den unter *Name für neuen Mediensatz* angegebenen Mediensatznamen. Dieser ist dann im Dialog aus Abbildung 18.4 sichtbar.

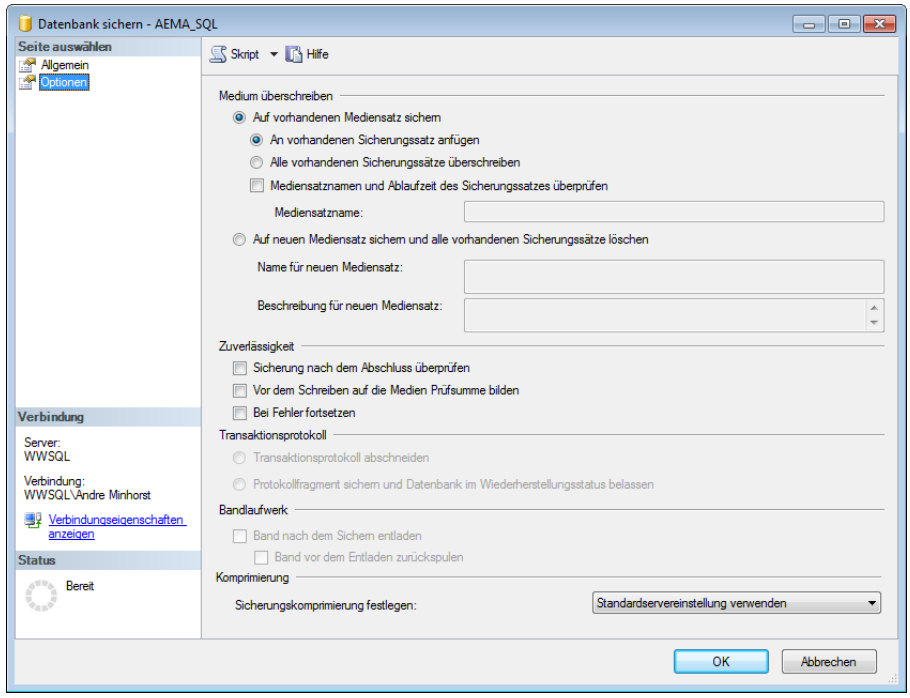


Abbildung 18.5: Zweite Seite der Optionen einer Datenbanksicherung

Was aber bewirkt der Mediensatzname? Sie können in einer weiteren Option namens *Mediensatznamen und Ablaufzeit des Sicherungssatzes überprüfen* festlegen, dass der Mediensatz vor dem Überschreiben auf den richtigen Namen überprüft wird.

Die Sicherung erfolgt dann nur, wenn der Mediensatz den angegebenen Namen aufweist. Hat der angegebene Mediensatz nicht den korrekten Namen, erscheint eine entsprechende Mel-

dung (siehe Abbildung 18.6). Lassen Sie den Namen des Mediensatzes leer, müsste Sie entsprechend auf einen leeren Mediensatznamen prüfen.

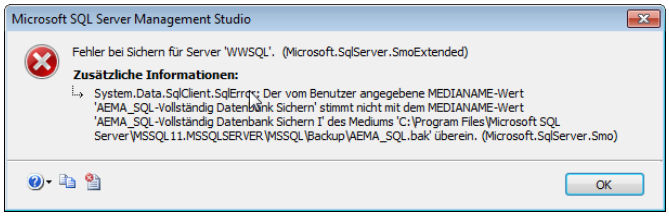


Abbildung 18.6: Fehler beim Abgleich des Mediensatznamens

Sollten Sie bei der Sicherung mit Mediensatznamen arbeiten wollen, aktivieren Sie bei der ersten Sicherung die Option *Auf neuen Mediensatz sichern und alle vorhandenen Sicherungssätze löschen* und vergeben Sie dabei einen Mediensatznamen.

Bei den folgenden Sicherungen aktivieren Sie dann *Auf vorhandenen Mediensatz sichern* und geben den festgelegten Mediensatznamen an. Auf diese Weise verhindern Sie, dass die Sicherung in der falschen Datei landet.

18.5.1 Sicherung prüfen

Auf der Seite *Optionen* gibt es einen weiteren Bereich namens *Zuverlässigkeit*. Hier können Sie die folgenden drei Optionen aktivieren:

» *Sicherung nach dem Abschluss überprüfen*: Prüft, ob der Sicherungssatz vollständig und lesbar ist.

» *Vor dem Schreiben auf die Medien Prüfsumme bilden*: Erstellt beim Sichern für jede Datenseite eine Prüfsumme und vergleicht diese mit der Prüfsumme der gesicherten Datenseite. Diese Funktion ist natürlich aufwendiger als die einfache Prüfung.

» *Bei Fehler fortsetzen*: Legt fest, ob die Sicherung nach dem Auftreten eines Fehlers fortgesetzt werden soll.

18.5.2 Einstellungen für die Transaktionsprotokollsicherung

Im Vorgriff auf die Beschreibung der Transaktionsprotokollsicherung weiter unten schauen wir uns noch die beiden Optionen unter *Transaktionsprotokoll* auf der Seite *Optionen* an:

» *Transaktionprotokoll abschneiden*: Sorgt dafür, dass das Transaktionsprotokoll nach der Sicherung verkleinert wird. Dies betrifft nur das Protokoll, nicht aber die Datei, die es enthält.

» *Protokollfragment sichern und Datenbank im Wiederherstellungsstatus belassen*: Ermöglicht im Falle eines Crashs eine abschließende Transaktionsprotokollsicherung.

18.5.3 Optionen für Bandlaufwerke

Unter *Bandlaufwerk* finden Sie zwei voneinander abhängige Optionen für den Einsatz von Bandlaufwerken:

» *Band nach dem Sichern entladen*

» *Band vor dem Entladen zurückspulen*

Die beiden Optionen sind selbsterklärend.

18.5.4 Komprimierung der Datensicherung

Die Enterprise Edition des SQL Servers stellt eine Komprimierung der Sicherung zur Verfügung. Der Vorgang der Sicherung wird dabei zwar etwas langsamer, dafür aber sind die Sicherungsdateien kleiner.

18.6 Vollsicherung durchführen

Nach dem Einstellen der Optionen auf den beiden Seiten des Dialogs *Datenbank sichern* können Sie die Sicherung mit einem Mausklick auf die Schaltfläche *OK* starten. Der SQL Server führt die Sicherung nun durch und bestätigt den Erfolgsfall mit einer entsprechenden Meldung. Die Sicherung wird dabei in der angegebenen Sicherungsdatei gespeichert.

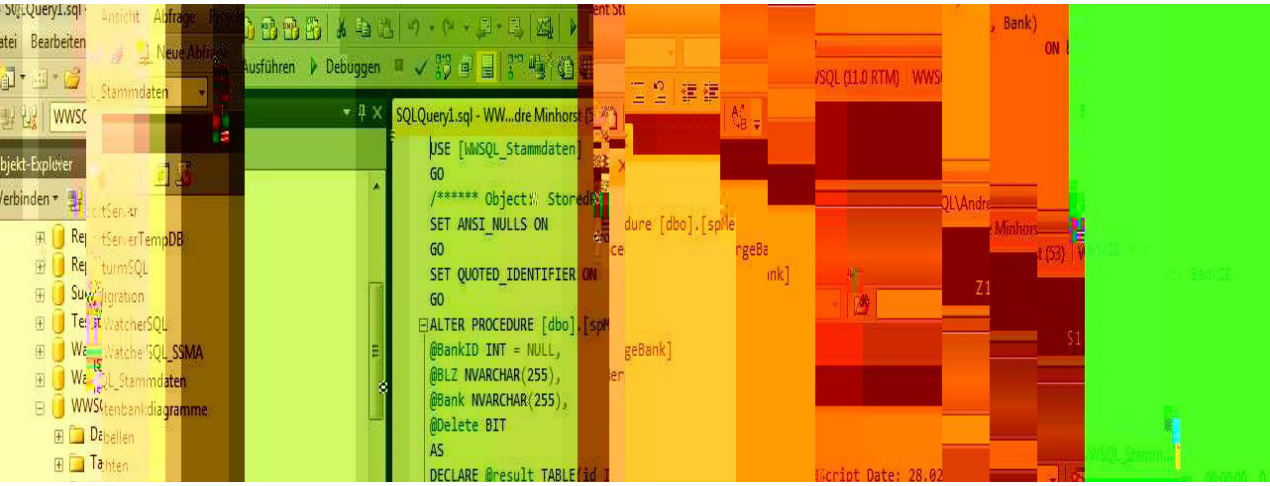
Irgendwie fehlt an dieser Stelle eine Möglichkeit, die Sicherungsparameter zu speichern, um diese später nochmals aufzurufen. Beim erneuten Öffnen des Dialogs zeigt dieser wieder die Standardwerte an. Allerdings haben Sie ja bereits erfahren, dass eigentlich alle Operationen von der Datenbankabfrage bis zur Erstellung von Datenbanken und Tabellen per Skript durchgeführt werden. Dementsprechend bietet auch der Dialog *Datenbank sichern* die Möglichkeit, ein Skript mit den angegebenen Parametern zu erstellen. Dieses sieht dann beispielsweise für die Standardeinstellungen wie folgt aus:

```
BACKUP DATABASE [AEMA_SQL]
TO  DISK = N'C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\Backup\
AEMA_SQL.bak'
WITH NOFORMAT, NOINIT,  NAME = N'AEMA_SQL-Vollständig Datenbank Sichern', SKIP,
NOREWIND, NOUNLOAD,  STATS = 10
```

18.6.1 Sicherung per SQL Server-Agent

Nun hilft dieses Skript allein noch nicht viel weiter, denn wir möchten dieses ja am liebsten zeitgesteuert ausführen. Wenn Sie mindestens die Standard Edition des SQL Server 2012 einsetzen, können Sie dies mithilfe des SQL Server-Agent erledigen. Dazu gehen Sie wie nachfolgend beschrieben vor:

... DIE ÜBRIGEN SEITEN GIBT'S GEDRUCKT UNTER SHOP.MINHORST.COM



BERND JUNGBLUTH
ANDRÉ MINHORST

André Minhorst Verlag