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* oben bereits 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 bereits im Kapitel /****FAQ kurz 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*). 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*.

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).

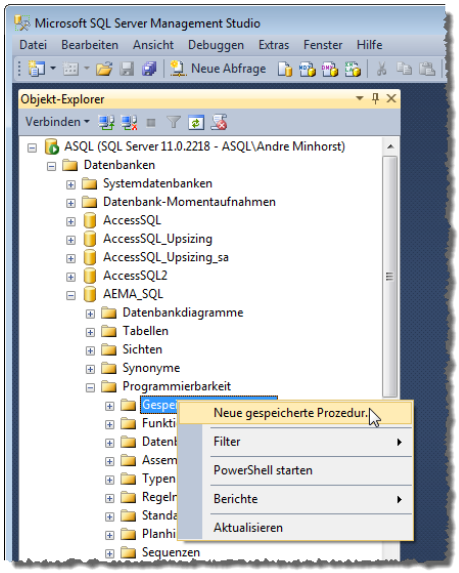

Abbildung 10.1: Anlegen des Grundgerüsts einer gespeicherten Prozedur

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.

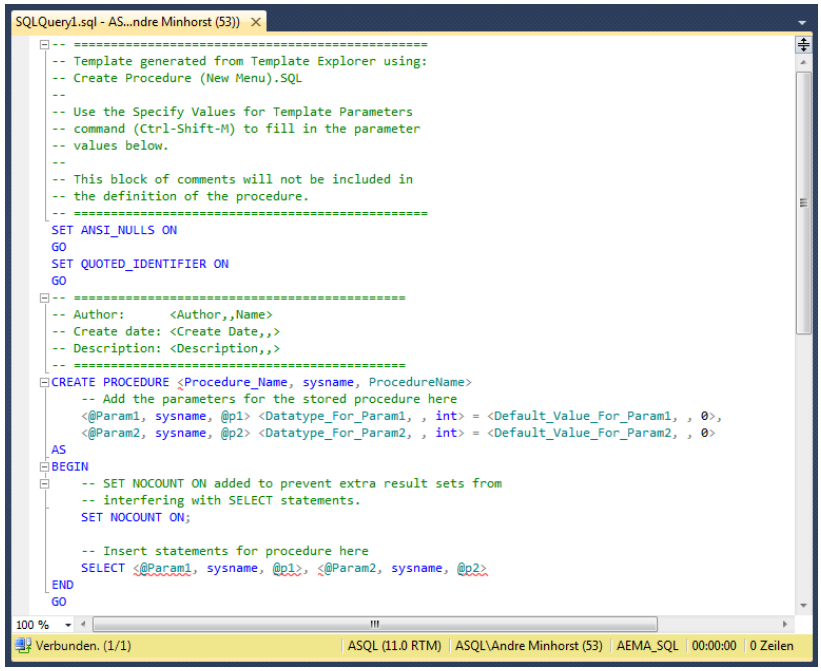

Abbildung 10.2: Vorlage für eine gespeicherte Prozedur

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.

Den Platzhalter *<Schema>* ersetzen Sie mit dem Namen des Schemas, dem die gespeicherte Prozedur angehören soll. Wenn Sie Ihr Berechtigungskonzept nicht auf Schemata aufbauen wollen (siehe Kapitel ****, Sicherheit), geben Sie hier einfach immer *dbo* an. Dies gilt auch für die übrigen Datenbankobjekte. Die erste Zeile lautet dann also:

```
CREATE PROCEDURE dbo.spAlleArtikel
```

Nach der Benennung folgen die Parameter. Die Definition eines Parameters besteht aus den folgenden Elementen:

» Name des Parameters, der immer mit @ beginnt

» Datentyp des Parameters – zum Beispiel *int* oder *nvarchar(255)*. Die Datentypen sind die gleichen, die auch bei der Definition von Tabellen zum Einsatz kommen (siehe Abschnitt ****, *Datentypen von Access nach SQL Server*).

» Schlüsselwort *OUTPUT*, falls es sich um einen Rückgabewert handelt

» Standardwert, der mit einem Gleichheitszeichen zugewiesen wird. Ein Standardwert kennzeichnet den Parameter als optional, wodurch dieser beim Aufruf der gespeicherten Prozedur nicht zwingend angegeben werden muss.

Eine Prozedur kann komplett ohne Parameter auskommen, Sie können aber auch bis zu 1.024 Parameter definieren. Die Parameter werden durch Kommata voneinander getrennt und wiederum der Übersicht halber in jeweils eine eigene Zeile geschrieben.

Nach der Parameterliste folgt das Schlüsselwort *AS* und schließlich die eigentlichen Anweisungen der gespeicherten Prozedur. In einem ganz einfachen Fall sieht dies nun wie folgt aus:

```
CREATE PROCEDURE dbo.spAlleArtikel
AS
SELECT * FROM dbo.tblArtikel;
```

Um die Prozedur zu erstellen, betätigen Sie am einfachsten die Taste *F5*. Das Abfragefenster quittiert dies, sofern keine Fehler auftreten, wie in Abbildung 10.3.

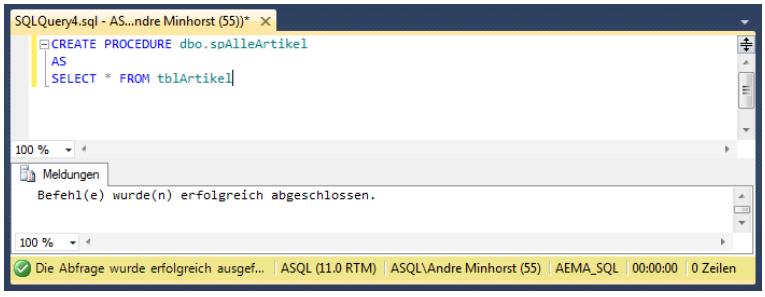

Abbildung 10.3: Erstellen einer ersten gespeicherten Prozedur

Um die soeben erstellte Prozedur auszuführen, tragen Sie den folgenden Befehl in ein neues oder in das aktuelle Abfragefenster ein:

```
EXEC dbo.spAlleArtikel;
```

Wenn Sie die Anweisung in ein Abfragefenster eingeben, das bereits andere Anweisungen enthält, markieren Sie die auszuführende Anweisung zuvor und betätigen Sie dann die Taste *F5*. Das Abfragefenster liefert das Resultat in der Datenblattansicht (siehe Abbildung 10.4).

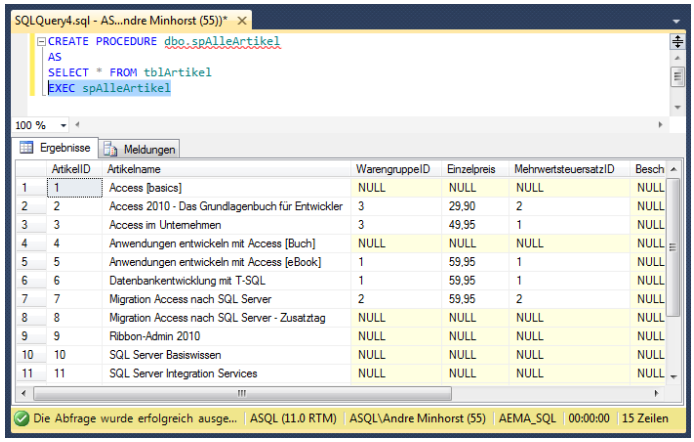

Abbildung 10.4: Aufruf der gespeicherten Prozedur über das Abfragefenster

10.3.3 Gespeicherte Prozedur mit Parametern

Möchten Sie der gespeicherten Prozedur Parameter übergeben, was im Großteil der Fälle geschehen wird, geben Sie diese vor dem *AS*-Schlüsselwort an. Wir erweitern die Logik der eben beschriebenen Prozedur so, dass diese nur einen bestimmten Artikel liefert:

```
CREATE PROC dbo.spArtikelNachID
@ArtikelID int
AS
SELECT * FROM dbo.tblArtikel WHERE ArtikelID = @ArtikelID;
```

Den Parameter übergeben Sie nun wie folgt:

```
EXEC dbo.spArtikelNachID 2;
```

Die gespeicherte Prozedur liefert jetzt nur den gesuchten Artikel zurück.

In diesem Beispiel wird eine Ganzzahl als Parameter übergeben. Erwartet der Parameter jedoch eine Zeichenfolge, müssen Sie diese in Hochkommata angeben. Dies gilt auch für Parameter mit Datumswerten, hier ist die Zeichenfolge das Datum im ISO-Format (*yyyy-mm-dd*). Auch Dezimalzahlen übergeben Sie als Zeichenfolge, wobei Sie als Dezimaltrennzeichen den Punkt