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.

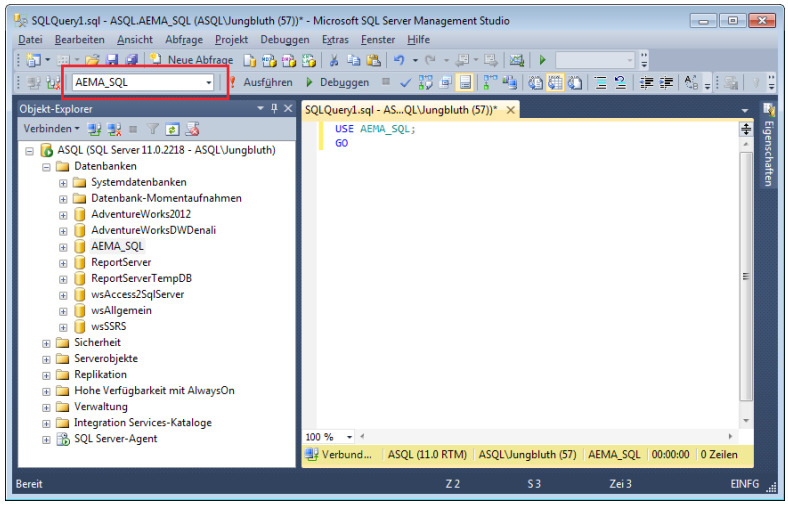

Abbildung 9.1: Die Auswahl der Datenbank für die Abfrage

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.

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.

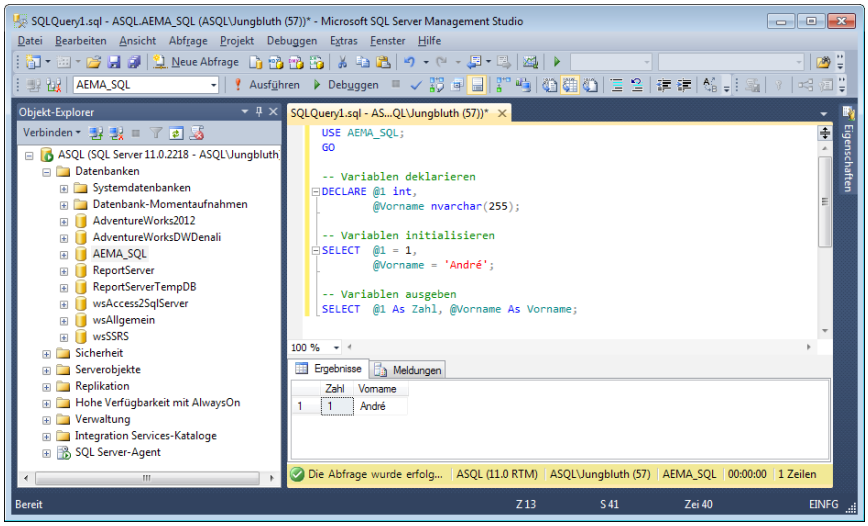

Abbildung 9.2: Ausführen von T-SQL-Skripten im Abfragefenster

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.

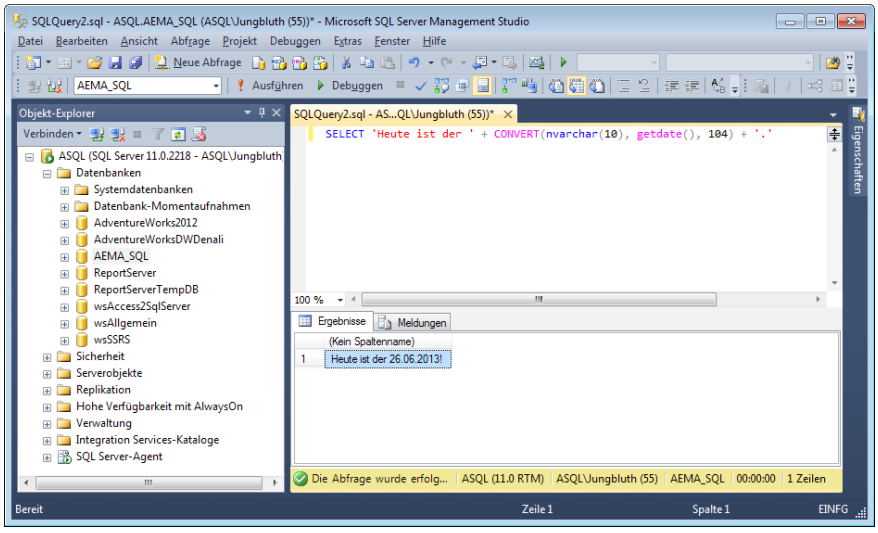

Abbildung 9.3: Ausgabe in das Meldungen-Fenster

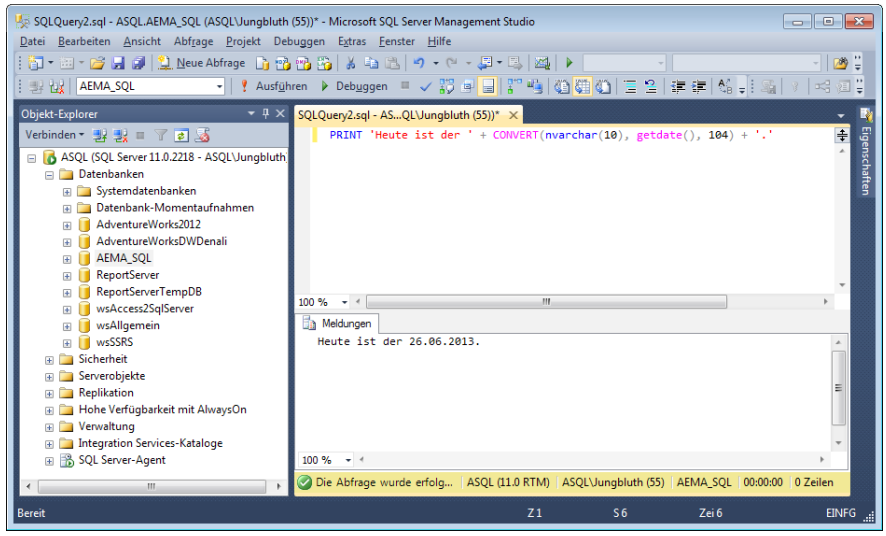

Abbildung 9.4: Ausgabe in das Ergebnisse-Fenster

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.

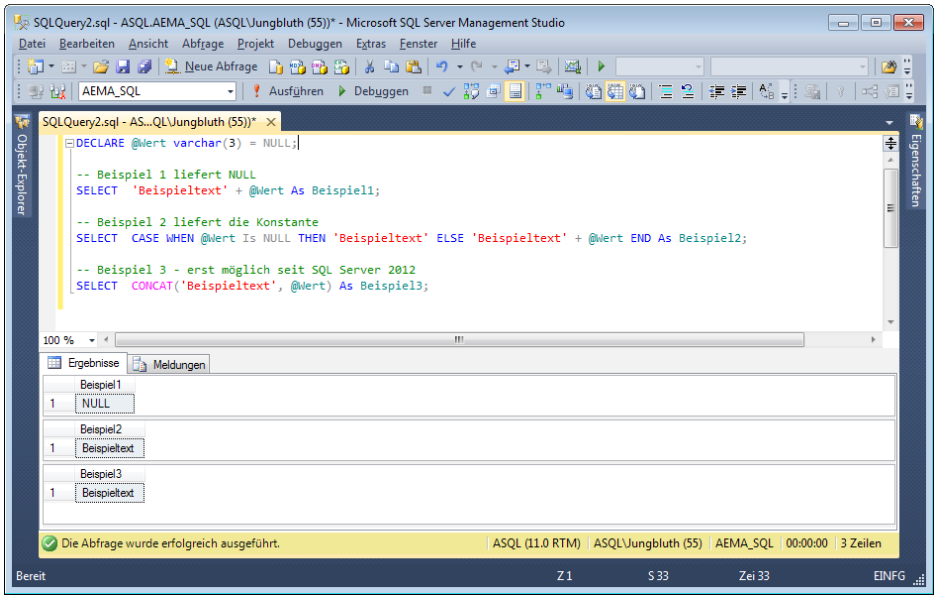

Abbildung 9.5: Verketten von Zeichenfolgen mit *NULL*-Werten

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).

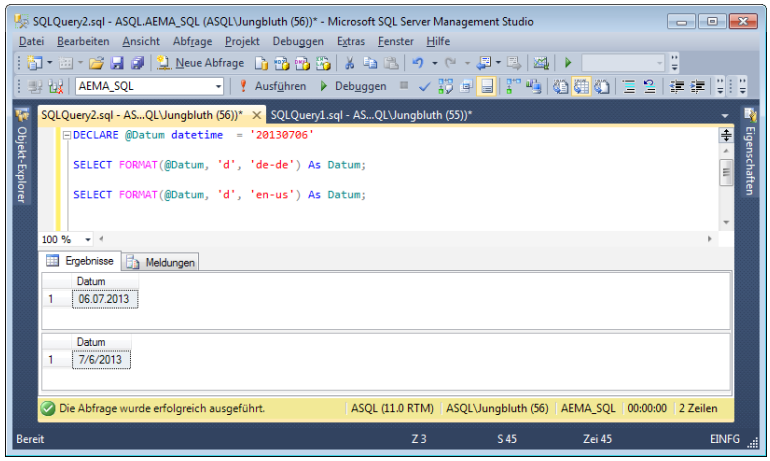

Abbildung 9.6: Formatierungen von Datumswerten

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)*. Wenn Sie beispielsweise eine Laufvariable deklarieren möchten, würden Sie dies so erledigen:

```
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).

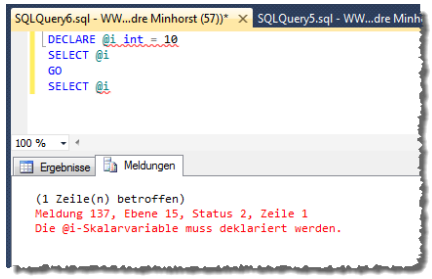

Abbildung 9.7: *GO* leert alle Variablen.

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 lassen sich an verschiedenen Stellen einsetzen. Ein Beispiel sind Auswahlabfragen: 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 und ****Benutzerdefinierte Funktionen 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:

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

Die Prozedur füllt den Parameter *@Rueckgabe* und gibt diesen zurück.

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: