16 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 Abfrage einem Formular, Bericht oder einem Steuerelement zuzuweisen. Die dazu notwendigen Techniken lernen Sie in diesem Kapitel kennen.

Die Beispiele zu diesem Kapitel finden Sie in der Datenbank *AEMA_SQL.accdb* im Modul *mdlBeispiele_VBADAO*****.

16.1 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 Tabelle im Datenbank-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 16.1).

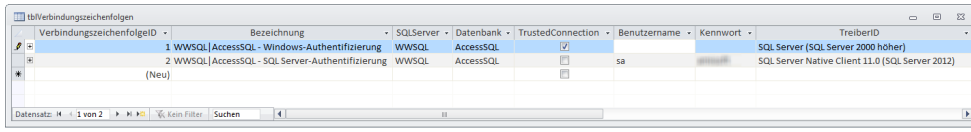

Abbildung 16.1: Die Tabelle speichert die Informationen zu den Verbindungszeichenfolgen

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

```
Private Function Verbindungszeichenfolge(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
        strTemp = strTemp & "UID=" & Nz(Me!txtBenutzername, "[Benutzername]") _
            & ";PWD=" & Nz(Me!txtKennwort, "[Kennwort]")
    End If
    Verbindungszeichenfolge = strTemp
End Function
```

Die Prozedur liest den Datensatz der Tabelle *tblVerbindungszeichenfolgen* ein, dessen Primärschlüsselwert dem per Parameter übergebenen Wert entspricht. Dann setzt sie die einzelnen Elemente zusammen. Die einzige Variante ist die Authentifizierungsart: Bei der Windows-Authentifizierung enthält die Verbindungszeichenfolge das Name-Wert-Paar *Trusted_Connection=Yes*, sonst die Benutzerdaten in der Form *UID=<Benutzername>;PWD=<Kennwort>*. An dieser Stelle weisen wir nochmals darauf hin, dass Sie nach Möglichkeit die Windows-Authentifizierung nutzen sollten – das Speichern von Benutzernamen und Kennwörtern ist nicht zu empfehlen.

16.2 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.

In den folgenden Absätzen schauen wir uns die zweite und die dritte Variante an.

16.2.1 Datensatz löschen per SQL

Bei der ersten Variante legen Sie eine Pass-Through-Abfrage mit der auszuführenden *DELETE*-Anweisung an (siehe Abbildung 16.2). 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

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

» Eintragen der *DELETE*-Anweisung

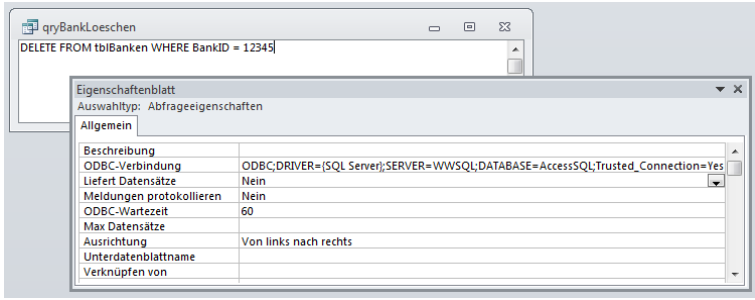

Abbildung 16.2: Statisches Löschen eines Datensatzes per Pass-Through-Abfrage

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

```
CurrentDb.QueryDefs("qryBankLoeschen").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 erwartet den Primärschlüsselwert des zu löschenden Datensatzes als Parameter:

```
Public Sub BankLoeschenSQLPassthrough(lngBankID As Long)
    Dim db As DAO.Database
    Dim qdf As DAO.QueryDef
    Set db = CurrentDb
    Set qdf = db.QueryDefs("qryBankLoeschen")
    qdf.SQL = "DELETE FROM 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:

```
BankLoeschenSQLPassthrough 12345
```

Diese Variante hat noch zwei Nachteile:

» Die an den SQL Server übergebene SQL-Anweisung wird dynamisch zusammengesetzt. Der Ausführungsplan für diese Anweisung muss somit jedesmal neu ermittelt werden.

» Die Verbindungszeichenfolge ist in der Abfrage gespeichert. Wenn sich diese ändert, muss sie an jeder Stelle angepasst werden.

» Wir erfahren nicht, ob die Aktion erfolgreich war und wieviele Datensätze gelöscht wurden.

In den folgenden beiden Abschnitten kümmern wir uns um diese Nachteile.

16.2.2 Datensatz löschen per gespeicherter Prozedur

Als Erstes sorgen wir dafür, dass der SQL Server den Ausführungsplan für die Abfrage speichern kann und erstellen eine gespeicherte Prozedur. Dies erledigen wir mit folgendem SQL-Skript:

```
CREATE PROCEDURE [dbo].[spBankLoeschen] (@BankID int)
AS
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 spBankLoeschen 123456
```

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 22.3 fest.

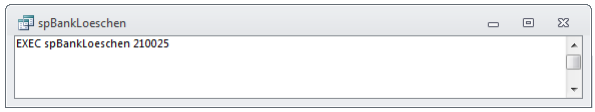

Abbildung 16.3: Aufruf der gespeicherten Prozedur von einer Access-Abfrage aus

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 BankLoeschenSPPassthrough(lngBankID As Long)
    Dim db As DAO.Database
    Dim qdf As DAO.QueryDef
    Set db = CurrentDb
    Set qdf = db.QueryDefs("spBankLoeschen")
```

```
    qdf.SQL = "EXEC spBankLoeschen " & lngBankID
    qdf.Execute
    Set qdf = Nothing
    Set db = Nothing
End Sub
```

Der Aufruf sieht beispielsweise wie folgt aus:

```
BankLoeschenSPPassthrough 123456
```

Dies ändert zunächst den SQL-Ausdruck der Abfrage *spBankLoeschen* wie folgt:

```
EXEC spBankLoeschen 123456
```

Dieser Aufruf wird direkt an den SQL Server gesendet, der dann die gespeicherte Prozedur *spBankLoeschen* mit dem angegebenen Parameter ausführt und den entsprechenden Datensatz löscht.

16.2.3 Pass-Through-Abfrage mit dynamischer Verbindungszeichenfolge

Nun soll noch die Verbindungszeichenfolge direkt aus der Tabelle *tblVerbindungszeichenfolgen* bezogen werden. Dazu übergeben Sie der Prozedur noch die ID der Verbindungszeichenfolge als weiteren Parameter. Dieser Parameter wird an die weiter oben erläuterte Funktion *VerbindungszeichenfolgeNachID* übergeben, die dann die Verbindungszeichenfolge zurückliefert. Das Ergebnis landet direkt in der Eigenschaft *Connect* des *QueryDef*-Objekts, was dem Zuweisen der Verbindungszeichenfolge zur Eigenschaft *ODBC-Verbindung* entspricht. Hier ein Auszug der Prozedur (die übrigen Zeilen entsprechen dem vorherigen Beispiel):

```
Public Sub BankLoeschenSPPassthroughMitVerbindungszeichenfolge(lngBankID As Long, _
        lngVerbindungszeichenfolgeID As Long)
    ...
    qdf.Connect = VerbindungszeichenfolgeNachID(1)
    ...
End Sub
```

Auch hier noch ein Beispielaufruf:

```
BankLoeschenSPPassthroughMitVerbindungszeichenfolge 210028, 1
```

Dies löscht den Datensatz mit dem Wert *210028* im Feld *BankID* und verwendet die Verbindungszeichenfolge mit dem Wert *1* im Feld *VerbindungszeichenfolgeID* der Tabelle *tblVerbindungszeichenfolgen*.

16.2.4 Löschen mit Bestätigung

Schließlich möchten Sie vielleicht noch wissen, ob der Löschvorgang überhaupt erfolgreich war beziehungsweise wieviele Datensätze von der Aktionsabfrage betroffen waren.

T-SQL bietet mit der Funktion *@@ROWCOUNT* ein Mittel, um die Anzahl der von der zuletzt ausgeführten Abfrage betroffenen Datensätze zu ermitteln. Dies bezieht sich auf die Aktionsabfragen der aktuellen Verbindung.

Die folgende gespeicherte Prozedur löscht wie in den obigen Beispielen einen Datensatz mit dem übergebenen Wert für das Feld *BankID*, gibt aber als Ergebnis die Anzahl der betroffenen Datensätze zurück:

```
CREATE PROCEDURE [dbo].[spBankLoeschenMitErgebnis]
@BankID INT
AS
DELETE FROM tblBanken WHERE BankID = @BankID
SELECT @@ROWCOUNT AS RecordsAffected;
```

Wenn Sie diese gespeicherte Prozedur im Abfragefenster im *SQL Server Management Studio* aufrufen, sieht das Ergebnis wie in Abbildung 16.4 aus.

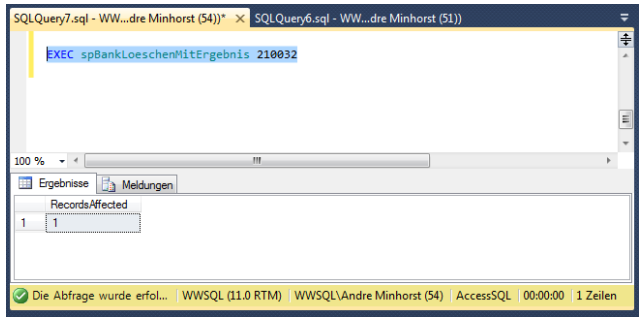

Abbildung 16.4: Ausgabe der Anzahl der gelöschten Datensätze

Um dieses Ergebnis von Access aus zu nutzen, ist eine kleine Änderung am Entwurf der ODBC-Abfrage nötig. Wir haben die Abfrage von oben unter dem Namen *spBankLoeschenMitErgebnis* kopiert und die Eigenschaft *Liefert Datensätze* auf den Wert *Ja* eingestellt (siehe Abbildung 16.5). Anderenfalls liefert die Abfrage das Ergebnis der *SELECT*-Abfrage mit der Anzahl der betroffenen Datensätze nicht zurück!

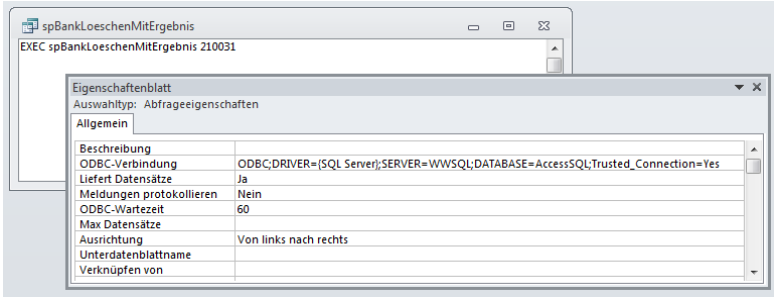

Abbildung 16.5: Einstellungen für eine Aktionsabfrage mit Rückgabewert

Führen Sie diese Abfrage direkt aus, liefert sie das Ergebnis aus Abbildung 16.6.

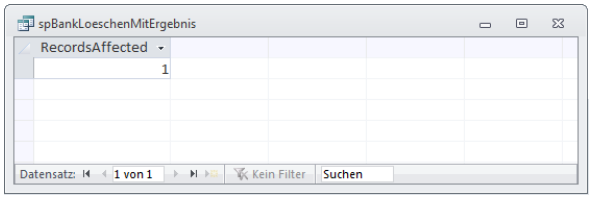

Abbildung 16.6: Ergebnis der gespeicherten Prozedur zum Löschen eines Datensatzes

Dies ist ein Ergebnis, mit dem wir auch unter VBA arbeiten können. Die folgende Prozedur verwendet wieder die *BankID* und die ID der Verbindungszeichenfolge als Parameter:

```
Public Sub BankLoeschenSPPassthroughMitErgebnis(lngBankID As Long, _
        lngVerbindungszeichenfolgeID As Long)
    Dim db As DAO.Database
    Dim qdf As DAO.QueryDef
    Dim rst As DAO.Recordset
    Dim lngAnzahl As Long
    Set db = CurrentDb
    Set qdf = db.QueryDefs("spBankLoeschenMitErgebnis")
    qdf.Connect = VerbindungszeichenfolgeNachID(1)
    qdf.SQL = "EXEC spBankLoeschenMitErgebnis " & lngBankID
    Set rst = qdf.OpenRecordset(dbOpenSnapshot)
    lngAnzahl = rst!RecordsAffected
    MsgBox "Es wurden " & lngAnzahl & " Datensätze gelöscht."
    Set rst = Nothing
    Set qdf = Nothing
    Set db = Nothing
End Sub
```

Sie erzeugt wie gewohnt ein *QueryDef*-Objekt auf Basis der gespeicherten Abfrage *spBankLoeschenMitErgebnis* und ermittelt die gewünschte Verbindungszeichenfolge. Dann weist sie wie zuvor den neuen SQL-Ausdruck zu, führt die Abfrage aber diesmal nicht mit *Execute* aus. Stattdessen erstellt sie ein neues *Recordset*-Objekt und füllt es über die *OpenRecordset*-Methode mit dem Ergebnis der gespeicherten Prozedur. Dies erzeugt ein herkömmliches *Recordset*-Objekt, dass nur einen Datensatz mit einem Feld enthält – und dieses wird mit *rst!RecordsetAffected* ausgelesen und in einem Meldungsfenster ausgegeben.

16.2.5 Dynamische Aktionsabfrage ohne Rückgabewert

Die bisherigen Ansätze gingen davon aus, dass die Access-Datenbank eine gespeicherte Abfrage mit den wichtigsten Eigenschaften zum Ausführen der gespeicherten Prozedur per Pass-Through-Abfrage enthält. Je mehr solcher Abfragen Sie verwenden, desto unübersichtlicher wird es im Navigationsbereich. Und davon abgesehen ändern wir ohnehin zumindest den SQL-Code jeder Pass-Through-Abfrage, die eine gespeicherte Prozedur mit Paramtern ausführt. Dann können wir diese auch gleich neu anlegen – der Performance-Unterschied dürfte sich in Grenzen halten.

Was benötigen wir also im Vergleich zur vorherigen Variante? Eigentlich müssen wir nur den Namen der zu verwendenden gespeicherten Prozedur zusätzlich übergeben, die Verbindungszeichenfolge und die Parameter werden ja bereits verarbeitet. Die erweiterte Prozedur sieht nun so aus:

```
Public Sub TemporaereSPMitParameterAusfuehren(strStoredProcedure As String, _
        lngVerbindungszeichenfolgeID As Long, _
        ParamArray varParameter() As Variant)
    Dim db As DAO.Database
    Dim qdf As DAO.QueryDef
    Dim strSQL As String
    Dim strParameter As String
    Dim var As Variant
    Set db = CurrentDb
    On Error Resume Next
    db.QueryDefs.Delete strStoredProcedure
    On Error GoTo 0
    Set qdf = db.CreateQueryDef(strStoredProcedure)
    For Each var In varParameter
        strParameter = strParameter & ", " & var
    Next var
    If Len(strParameter) > 0 Then
        strParameter = Mid(strParameter, 3)
    End If
    With qdf
```

```
        .Connect = VerbindungszeichenfolgeNachID(lngVerbindungszeichenfolgeID)
        .ReturnsRecords = False
        .SQL = "EXEC " & strStoredProcedure & " " & strParameter
        .Execute
    End With
    Set db = Nothing
End Sub
```

Die Prozedur löscht zunächst wieder eine eventuell bereits vorhandene Abfrage mit dem Namen der zu verwendenden und per Parameter übergebenen gespeicherten Prozedur. Diese wird dann mit der *CreateQueryDef*-Methode neu erstellt. Die folgenden Zeilen stellen die Verbindungszeichenfolge ein, legen für *ReturnRecords* den Wert *False* fest und fügen das Schlüsselwort *EXEC*, den Namen der gespeicherten Prozedur und die Parameterliste zusammen.

Die *Execute*-Methode führt die Abfrage schließlich durch.

16.2.6 Dynamische Aktionsabfrage mit Rückgabewert

Weiter oben haben wir gesehen, dass es sinnvoll sein kann, die Anzahl der von einer Aktionsabfrage betroffenen Datensätze zu ermitteln. Im Beispiel oben haben wir mit der folgenden Zeile beim Erstellen der gespeicherten Prozedur dafür gesorgt, dass diese ein Recordset mit der Anzahl der von der Aktionsabfrage betroffenen Datensätze zurückliefert:

```
SELECT @@ROWCOUNT AS RecordsAffected;
```

Wir können jedoch nicht voraussetzen, dass jede Aktionsabfrage, die in einer gespeicherten Prozedur steckt, das Ergebnis auf diese Weise liefert. Kein Problem: In der folgenden Funktion können Sie angeben, ob diese Zeile in der gespeicherten Prozedur enthalten ist oder nicht. Ist dies nicht der Fall, fügt die Funktion diese Zeile noch hinzu.

Die Funktion erwartet dementsprechend vier Parameter:

- *strStoredProcedure*: Name der zu verwendenden gespeicherten Prozedur
- *lngVerbindungszeichenfolgeID*: ID der Verbindungszeichenfolge aus der Tabelle tblVerbindungszeichenfolgen
- *bolRueckgabeImplementiert*: Wert, der angibt, ob die Rückgabe der Anzahl der von der Aktionsabfrage betroffenen Datensätze in der gespeicherten Prozedur implementiert ist
- *varParameter*: Parameter-Array mit den zu übergebenden Parametern

Die Funktion sieht wie folgt aus:

```
Public Function SPAktionsabfrageMitErgebnis(strStoredProcedure As String, _
        lngVerbindungszeichenfolgeID As Long, _
        bolRueckgabeImplementiert As Boolean, _
```

```
        ParamArray varParameter() As Variant) As Long
    Dim db As DAO.Database
    Dim qdf As DAO.QueryDef
    Dim rst As DAO.Recordset
    Dim strSQL As String
    Dim strParameter As String
    Dim var As Variant
    Set db = CurrentDb
    On Error Resume Next
    db.QueryDefs.Delete strStoredProcedure
    On Error GoTo 0
    Set qdf = db.CreateQueryDef(strStoredProcedure)
    For Each var In varParameter
        strParameter = strParameter & ", " & var
    Next var
    If Len(strParameter) > 0 Then
        strParameter = Mid(strParameter, 3)
    End If
    With qdf
        .Connect = VerbindungszeichenfolgeNachID(lngVerbindungszeichenfolgeID)
        .ReturnsRecords = True
        .SQL = "EXEC " & strStoredProcedure & " " & strParameter
        If Not bolRueckgabeImplementiert Then
            .SQL = .SQL & vbCrLf & "SELECT @@ROWCOUNT AS RecordsAffected;"
        End If
        Set rst = .OpenRecordset
    End With
    SPAktionsabfrageMitErgebnis = rst!RecordsAffected
    Set db = Nothing
End Function
```

Die Funktion arbeitet zunächst wie die zuvor beschriebene Prozedur. Sie ist jedoch als Function deklariert und liefert einen *Long*-Wert als Ergebnis. Außerdem verarbeitet Sie beim Definieren der Pass-Through-Abfrage den Parameter *bolRueckgabeImplementiert*. Hat dieser den Wert *False*, geht die Funktion davon aus, dass die gespeicherte Prozedur kein Recordset mit der Anzahl der von der Aktionsabfrage betroffenen Datensätze zurückliefert. In diesem Fall ergänzt sie den SQL-Ausdruck selbstständig um die entsprechende Zeile. Der SQL-Ausdruck sieht dann beispielsweise wie folgt aus:

```
EXEC spBankLoeschen 123456
SELECT @@ROWCOUNT AS RecordsAffected
```