

VBA mit Access

Das umfassende Handbuch

» Hier geht's
direkt
zum Buch

DIE LESEPROBE

Kapitel 4

Ein Streifzug in die Welt der Objekte

In diesem Kapitel werde ich die wichtigsten und interessantesten Objekte von Access und VBA behandeln. Jedes Objekt in Access hat bestimmte Methoden und Eigenschaften, die genau für dieses Objekt ausgeführt werden können.

Unter anderem werde ich in diesem Kapitel folgende Fragen beantworten:

- ▶ Wie zeige ich Informationen zu meiner Datenbank an?
- ▶ Wie beende ich Datenbanken sowie die Applikation?
- ▶ Wie greife ich mit Funktionen auf Tabelle zu?
- ▶ Welche Drucker sind installiert, und wie heißen sie?
- ▶ Welche Tabellen, Berichte und Formulare befinden sich in der Datenbank?
- ▶ Wie rufe ich Berichte und Formulare in Access auf?
- ▶ Wie transferiere ich Daten nach Excel?
- ▶ Welche Möglichkeiten habe ich, auf integrierte Dialoge in Access zuzugreifen?

In der Entwicklungsumgebung sehen Sie die Methoden und Eigenschaften sofort, wenn Sie ein Objekt eingeben und danach einen Punkt setzen. Dann erscheint nämlich prompt ein Kontextmenü, das die zur Verfügung stehenden Methoden und Eigenschaften anzeigt.

Alle Prozeduren und Funktionen aus diesem Kapitel finden Sie in den Materialien zum Buch im Verzeichnis *Kap04* unter dem Namen *Objekte.accdb*.

4.1 Das »Application«-Objekt

Das *Application*-Objekt steht auf oberster Ebene. Es bezieht sich auf die aktive Microsoft-Access-Anwendung und enthält alle darunterliegenden Objekte, wie Formulare, Reports, Drucker und Bildschirm.

4.1.1 Datenbankinformationen erhalten

Anhand des `Application`-Objekts können Sie einiges über Ihre Access-Umgebung erfahren, indem Sie verschiedene Methoden anwenden. Listing 4.1 gibt den Namen der Datenbank wieder.

```
Sub AccessDBErmitteln()  
  
    MsgBox "Die aktuelle Datenbank heißt: " & _  
        Application.CurrentProject.Name  
  
End Sub
```

Listing 4.1 Den Namen der Datenbank ermitteln

Die Eigenschaft `Name` gibt den Namen des Objekts `CurrentProject` bekannt.

Möchten Sie nicht nur den Namen der aktuell geöffneten Datenbank angezeigt bekommen, sondern auch den kompletten Speicherpfad, so starten Sie die Prozedur aus Listing 4.2.

```
Sub AccessDBMitPfadErmitteln()  
  
    MsgBox "Die aktuelle Datenbank heißt: " & _  
        Application.CurrentDb.Name  
  
End Sub
```

Listing 4.2 Namen und Speicherort der Datenbank ermitteln (Variante 1)

Die `CurrentDb`-Methode gibt eine Objektvariable des Typs `Database` zurück, die der Datenbank entspricht, die momentan im Microsoft-Access-Fenster geöffnet ist.

Alternativ zu der letzten Prozedur können Sie Listing 4.3 einsetzen, um den Pfad der aktuell geöffneten Datenbank auszugeben.

```
Sub AccessDBMitPfadErmitteln2()  
  
    MsgBox "Die aktuelle Datenbank heißt: " & _  
        Application.CurrentProject.Path  
  
End Sub
```

Listing 4.3 Namen und Speicherort der Datenbank ermitteln (Variante 2)

Über die Eigenschaft `Path`, die Sie auf das Objekt `CurrentProject` anwenden, ermitteln Sie den Pfad der aktuell geöffneten Datenbank.

4.1.2 Aktuellen Anwendernamen ermitteln

Mit der Methode `CurrentUser` geben Sie den Namen des aktuellen Benutzers der Datenbank zurück.

```
Sub AktuellerUser()

    MsgBox "Der aktuelle Benutzer ist: " & _
        Application.CurrentUser, vbInformation

End Sub
```

Listing 4.4 Den aktuellen Benutzer der Datenbank abfragen

Möchten Sie hingegen den aktuell an Windows angemeldeten User abfragen, dann verwenden Sie folgende Zeile:

```
MsgBox environ("username")
```

4.1.3 Installierte Drucker ermitteln

Um zu ermitteln, welche Drucker Sie einsetzen und an welchem Anschluss sie hängen, können Sie das neue Auflistungsobjekt `Printers` nutzen, das Sie im Zusammenspiel mit dem Objekt `Application` einsetzen. Den dafür notwendigen Code sehen Sie in Listing 4.5.

```
Sub DruckerErmitteln()
    Dim prtDrucker As Printer
    Dim strDrucker As String

    For Each prtDrucker In Application.Printers
        With prtDrucker
            strDrucker = strDrucker & vbCrLf & "Druckername: " & .DeviceName & vbCrLf _
                & "Anschluss: " & .Port
        End With
    Next prtDrucker
    Debug.Print strDrucker

End Sub
```

Listing 4.5 Alle installierten Drucker auslesen

Die Eigenschaft `DeviceName` zeigt den Druckernamen an. Über die Eigenschaft `Port` zeigen Sie den Anschluss an, an dem Ihr Drucker hängt.

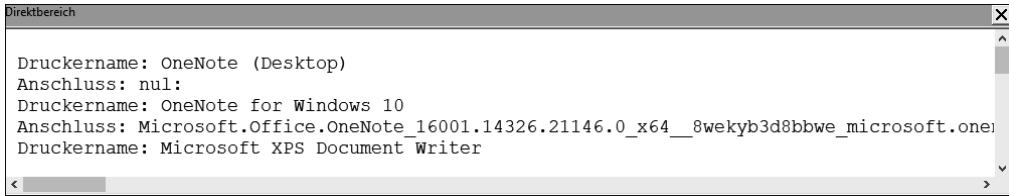


Abbildung 4.1 Alle im System bekannten Drucker werden ermittelt und ausgegeben.

4.1.4 Datenbank schließen

Soll die aktuelle Datenbank geschlossen werden, dann können Sie für diese Aufgabe die Methode `CloseCurrentDatabase` einsetzen.

```
Sub AktuelleDBschließen()

    Application.CloseCurrentDatabase

End Sub
```

Listing 4.6 Access-Datenbank schließen

4.1.5 Access beenden

Die Methode `Quit` dient dem Beenden von Access. Sie können dabei bestimmen, ob Änderungen angenommen oder verworfen werden sollen. Bei der Prozedur aus Listing 4.7 wird Access geschlossen. Dabei werden Sie durch eine Meldung aufgefordert, anzugeben, wie Sie mit den Änderungen umgehen möchten.

```
Sub AccessBeenden()

    Application.Quit acQuitPrompt

End Sub
```

Listing 4.7 Access beenden

Um zu bestimmen, was mit den Änderungen an der Datenbank beim Beenden des Programms geschehen soll, können Sie die `acQuit`-Option einsetzen. Dabei stehen Ihnen folgende Konstanten zur Verfügung:

- ▶ `acQuitPrompt`: Beim Schließen Ihrer Datenbank wird ein Meldungsfenster angezeigt, in dem Sie selbst entscheiden müssen, ob Sie Änderungen an der Datenbank akzeptieren oder verwerfen möchten.

- ▶ `acQuitSaveAll`: Bei dieser Standardeinstellung werden alle geänderten Daten in der Datenbank automatisch gespeichert, ohne dass eine Rückfrage erfolgt.
- ▶ `acQuitSaveNone`: Bei Verwendung dieser Konstanten wird die Access-Datenbank geschlossen, wobei keine Änderungen an den Daten übernommen werden.

4.1.6 Aktuelle Access-Version ausgeben

Über die Eigenschaft `Version` des Objekts `Application` finden Sie heraus, welche Access-Version bei Ihnen oder Ihrem Kunden im Einsatz ist.

Die Prozedur aus Listing 4.8 meldet Ihnen die aktuell installierte Access-Version.

```
Sub AccessVersionAusgeben()
    MsgBox "Sie arbeiten mit der Access-Version: " & Application.Version
End Sub
```

Listing 4.8 Access-Version ermitteln

4.1.7 Formular anlegen

Über das Objekt `Application` und die Methode `CreateForm` können Sie ein Formular anlegen und dabei einen Bezug zu einer in der Datenbank vorhandenen Tabelle herstellen. Des Weiteren ist es dabei möglich, das neue Formular auf Basis eines bereits vorhandenen herzustellen.

```
Sub NeuesFormularAnlegen()
    Dim frm As Form

    Set frm = Application.CreateForm _
        (Application.CurrentDb.Name, "Artikel")

    DoCmd.Restore
    frm.RecordSource = "Artikel"

End Sub
```

Listing 4.9 Ein neues Formular erstellen

Die `CreateForm`-Methode erstellt ein Formular und gibt ein `Form`-Objekt zurück. Dabei lautet die Syntax dieser Methode:

```
CreateForm([Datenbank[, Formularvorlage]])
```

Im Argument `Datenbank` geben Sie den Namen der Datenbank an, die die Formularvorlage enthält, mit der Sie Ihr Formular erstellen wollen. Wenn Sie die aktuelle Datenbank verwenden möchten, geben Sie dieses Argument nicht an bzw. ermitteln Sie über die Anweisung `Application.CurrentDb.Name` den Namen der aktiven Datenbank.

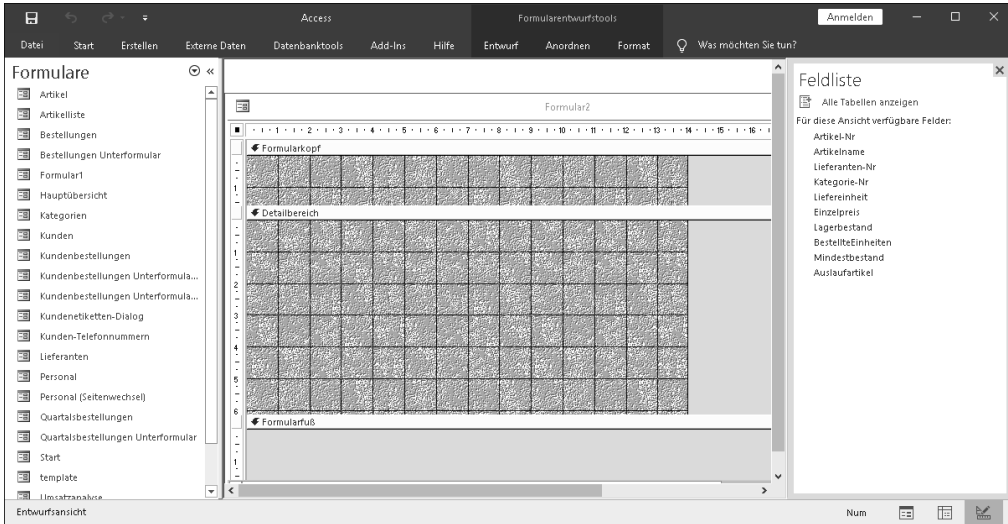


Abbildung 4.2 Ein neues Formular wurde auf Basis einer Vorlage erstellt.

Im Argument `Formularvorlage` geben Sie den Namen des Formulars an, das Sie als Vorlage zum Erstellen eines neuen Formulars verwenden möchten. Wenn Sie dieses Argument nicht angeben, legt Microsoft Access dem neuen Formular die Vorlage zugrunde, die auf der Registerkarte `FORMULARE • BERICHTE` des Dialogfeldes `OPTIONEN` angegeben ist. Sie finden diese Einstellungen bei den `OPTIONEN` unter `OBJECT-DESIGNER` im Abschnitt `ENTWURFSANSICHT FÜR FORMULAR/BERICHTE`.

Mit der Methode `Restore` stellen Sie ein maximiertes oder minimiertes Fenster in seiner vorherigen Größe wieder her. Über die Eigenschaft `RecordSource` geben Sie an, woher das neue Formular die Daten nehmen soll, mit denen es verknüpft wird.

4.1.8 Durchschnitt errechnen

Ebenfalls direkt unterhalb des `Application`-Objekts liegt die Funktion `DAvg`. Mit dieser Funktion ermitteln Sie den Mittelwert beispielsweise aus einer Tabelle. Die Prozedur aus Listing 4.10 berechnet aus allen Frachtkosten der Tabelle `Bestellungen` den Durchschnitt und gibt ihn auf dem Bildschirm aus.

```
Sub MittelwertBerechnen()
```

```
    MsgBox "Die durchschnittlichen Frachtkosten liegen bei" _
```

```
& vbCrLf & Format(DAvg("[Frachtkosten]", _
"Bestellungen"), "0.00"), vbInformation
```

End Sub

Listing 4.10 Den Durchschnitt eines Tabellenfeldes berechnen

Die Funktion `Format` wird hier eingesetzt, um die Ausgabe in das gewünschte Format zu bringen.

4.1.9 Summen ermitteln

Die Funktion `DSum` liegt auch direkt unterhalb des `Application`-Objekts. Mit ihrer Hilfe ermitteln Sie die Summe eines Feldes beispielsweise aus einer Tabelle. Die folgende Prozedur in Listing 4.11 ermittelt aus allen Frachtkosten der Tabelle *Bestellungen* die Summe und gibt diese auf dem Bildschirm aus.

```
Sub SummeBerechnen()

    MsgBox "Die Summe der Frachtkosten beträgt" _
        & vbCrLf & Format(DSum("[Frachtkosten]", _
        "Bestellungen"), "0,000.00"), vbInformation
```

End Sub

Listing 4.11 Die Summe eines Tabellenfeldes berechnen

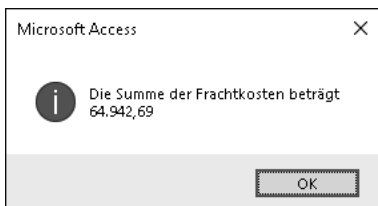


Abbildung 4.3 Die Summe aller Frachtkosten aus einer geschlossenen Tabelle ermitteln

4.1.10 Datensätze zählen

Mit der Funktion `DCount` ermitteln Sie die Anzahl der Datensätze beispielsweise einer Tabelle. Die Prozedur aus Listing 4.12 zählt aus der Tabelle *Bestellungen* die Datensätze und gibt die Anzahl auf dem Bildschirm aus.


```
Sub SätzeZählen()
```

```
    MsgBox "Die Anzahl der Datensätze beträgt" & _  
        & vbCrLf & DCount("*", "Bestellungen"), vbInformation
```

```
End Sub
```

Listing 4.12 Die Anzahl aller Datensätze einer Tabelle ermitteln

Sollen die Sätze nur bedingt gezählt werden, dann starten Sie die Prozedur aus Listing 4.13. Dort werden nur Sätze gezählt, bei denen die Frachtkosten bei mehr als 100 € liegen.

```
Sub SätzeBedingtZählen()
```

```
    MsgBox "Die Anzahl der Datensätze beträgt" & _  
        & vbCrLf & DCount("Frachtkosten", "Bestellungen", "Frachtkosten>100"), _  
        vbInformation
```

```
End Sub
```

Listing 4.13 Ein zusätzliches Kriterium für eine bedingte Zählung angeben

Das dritte Argument entspricht dem Kriterium, auf Basis dessen gezählt werden soll.

4.1.11 Minimal- und Maximalwerte ermitteln

Mit den Funktionen `DMax` und `DMin` ermitteln Sie aus einer Tabelle für ein bestimmtes Feld den größten bzw. den kleinsten Wert. Die Prozedur aus Listing 4.14 findet in allen Frachtkosten in der Tabelle *Bestellungen* den größten bzw. den kleinsten Wert und gibt ihn auf dem Bildschirm aus.

```
Sub MinUndMaxBerechnen()
```

```
    MsgBox "Die höchsten Frachtkosten betragen" & _  
        vbCrLf & Format(DMax("[Frachtkosten]", _  
            "Bestellungen"), _  
            "0,000.00") & vbCrLf & vbCrLf & _  
            "Die niedrigsten Frachtkosten betragen" & _  
            vbCrLf & Format(DMin("[Frachtkosten]", _  
            "Bestellungen"), "0.00"), vbInformation
```

```
End Sub
```

Listing 4.14 Die Ausreißerwerte einer Tabelle ermitteln

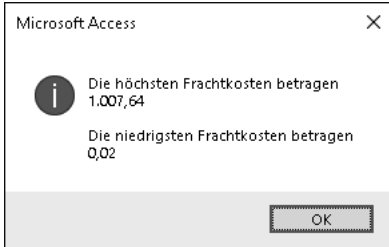


Abbildung 4.4 Die Extremwerte bei den Frachtkosten ermitteln

4.2 Das Objekt »AccessObject«

Mithilfe des Objekts `AccessObject` können Sie auf Auflistungsobjekte zugreifen und diese auswerten. Dabei stehen Ihnen die Auflistungsobjekte aus Tabelle 4.1 zur Verfügung.

| AccessObject | Auflistung | Enthält Informationen über |
|-----------------------|---------------------|---|
| Datenzugriffsseite | AllDataAccessPages | gespeicherte Datenzugriffsseiten (wird seit Access 2007 nicht mehr unterstützt) |
| Datenbankdiagramm | AllDatabaseDiagrams | gespeicherte Datenbankdiagramme |
| Formular | AllForms | gespeicherte Formulare |
| Funktion | AllFunctions | gespeicherte Funktionen |
| Makro | AllMacros | gespeicherte Makros |
| Modul | AllModules | gespeicherte Module |
| Abfrage | AllQueries | gespeicherte Abfragen |
| Bericht | AllReports | gespeicherte Berichte |
| gespeicherte Prozedur | AllStoredProcedures | gespeicherte Prozeduren |
| Tabelle | AllTables | gespeicherte Tabellen |
| Sicht | AllViews | gespeicherte Ansichten |

Tabelle 4.1 Alle »AccessObjects« im Überblick

Sehen Sie nun anhand einiger Beispiele, wie Sie diese Auflistungsobjekte einsetzen können.

Im Beispiel aus Listing 4.15 lesen Sie die Namen aller Module aus, die sich in Ihrer aktuellen Datenbank befinden.

```
Sub AlleModuleAuflisten()  
    Dim obj As AccessObject  
    Dim dbs As Object  
  
    Set dbs = Application.CurrentProject  
    For Each obj In dbs.AllModules  
        Debug.Print obj.Name  
    Next obj  
  
End Sub
```

Listing 4.15 Alle Module in der Datenbank auflisten

Über die Eigenschaft Name lassen Sie sich die Namen der einzelnen Module ausgeben, indem Sie das Auflistungsobjekt AllModules einsetzen.

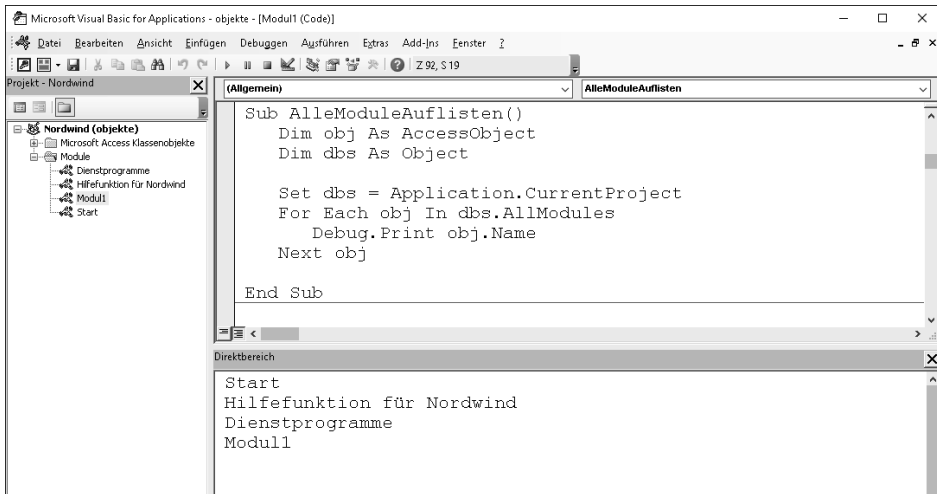


Abbildung 4.5 Alle Module der Datenbank im Direktfenster ausgeben

So können Sie nach und nach jedes einzelne Access-Objekt abfragen. Die Prozedur aus Listing 4.16 listet z. B. alle Tabellen der Datenbank auf.

```
Sub TabellenAuflisten()  
    Dim obj As AccessObject  
    Dim dbs As Object  
  
    Set dbs = Application.CurrentData  
    For Each obj In dbs.AllTables
```

```

    Debug.Print obj.Name
  Next obj

```

```
End Sub
```

Listing 4.16 Alle Tabellen der Datenbank auflisten

Im Auflistungsobjekt `AllTables` sind alle Tabellen der Datenbank verzeichnet. Wollen Sie die Abfrage aus Listing 4.16 auf alle geöffneten Tabellen beschränken, setzen Sie die Prozedur aus Listing 4.17 ein.

```

Sub TabelleGeoeffnet()
  Dim obj As AccessObject
  Dim dbs As Object

  Set dbs = Application.CurrentData

  For Each obj In dbs.AllTables
    If obj.IsLoaded = True Then
      Debug.Print "Name: " & obj.Name
      Debug.Print "Erstellungsdatum: " & obj.DateCreated
      Debug.Print "Änderungsdatum: " & obj.DateModified & vbLf
    End If
  Next obj

End Sub

```

Listing 4.17 Alle geöffneten Tabellen der Datenbank auflisten

Über die Eigenschaft `IsLoaded` ermitteln Sie, ob die Tabelle momentan geladen ist. Über die Eigenschaft `Name` finden Sie den Namen der Tabelle heraus. Die Eigenschaften `DateCreated` sowie `DateModified` geben Auskunft über das Erstellungs- bzw. das letzte Änderungsdatum der Tabelle.

Darüber hinaus können Sie sich mit dem `AccessObject` alle Abfragen der Datenbank anzeigen lassen. Die Prozedur für diesen Zweck zeigt Listing 4.18.

```

Sub AlleAbfragenAuflisten()
  Dim obj As AccessObject
  Dim dbs As Object

  Set dbs = Application.CurrentData

  For Each obj In dbs.AllQueries
    Debug.Print obj.Name
  Next obj

```

```
Next obj
```

```
End Sub
```

Listing 4.18 Alle Abfragen einer Datenbank auflisten

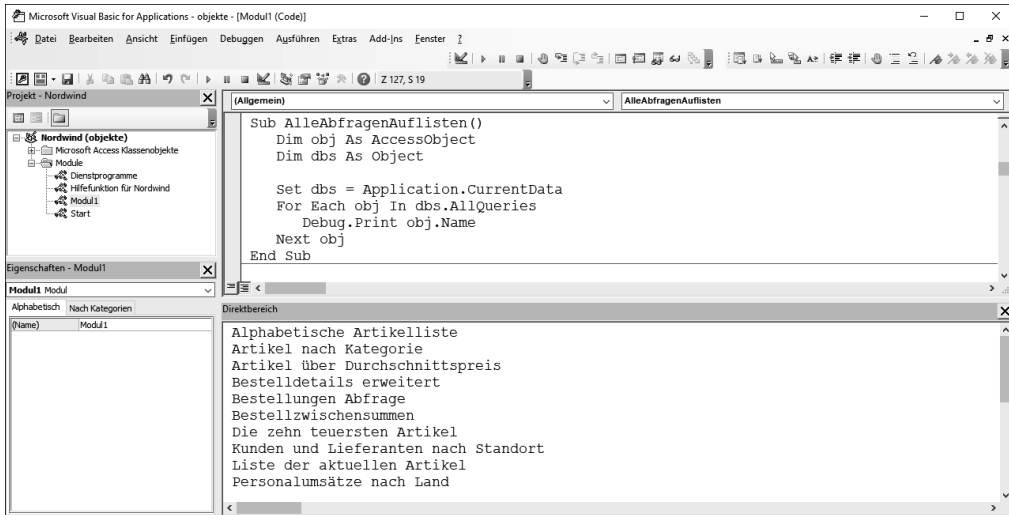


Abbildung 4.6 Den Namen, das Erstellungs- sowie das Änderungsdatum der geöffneten Tabellen herausfinden

Mithilfe des Auflistungsobjekts `AllQueries` ermitteln Sie alle Abfragen der Datenbank.

Der Vollständigkeit halber folgt in Listing 4.19 noch eine Prozedur zum Auflisten aller Formulare einer Datenbank.

```
Sub AlleFormulareAuflisten()  
    Dim obj As AccessObject  
    Dim dbs As Object  
  
    Set dbs = Application.CurrentProject  
    For Each obj In dbs.AllForms  
        Debug.Print obj.Name  
    Next obj  
  
End Sub
```

Listing 4.19 Alle Formulare einer Datenbank auflisten

Mit dem Auflistungsobjekt `AllForms` ermitteln Sie alle Formulare in Ihrer Datenbank. Wie aber gehen Sie vor, wenn Sie prüfen möchten, ob ein bestimmtes Formular in Ihrer Datenbank existiert? Um diese Aufgabe zu lösen, schreiben Sie eine Funktion wie die in Listing 4.20.

```
Function FormularPrüfer(str) As String
    Dim obj As AccessObject

    For Each obj In CurrentProject.AllForms
        If obj.Name = str Then str = obj.Name
    Next
    FormularPrüfer = str
End Function
```

Listing 4.20 Ist ein bestimmtes Formular in der Datenbank vorhanden?

Über das Auflistungsobjekt `AllForms` durchlaufen Sie alle Formulare in Ihrer Datenbank. Prüfen Sie, ob sich das gesuchte Formular innerhalb der `For Each`-Schleife befindet. Wenn ja, rufen Sie wie in Listing 4.21 das Formular über die Methode `OpenForm` auf.

```
Sub FormularDa()
    On Error GoTo Fehler
    If FormularPrüfer("Personal") <> "" _
        Then DoCmd.OpenForm "Personal"
    Exit Sub
Fehler:
    MsgBox "Dieses Formular gibt es nicht!"
End Sub
```

Listing 4.21 Formular öffnen nach Prüfung

Gerade haben Sie das Objekt `DoCmd` verwendet. Sie sehen etwas später in diesem Kapitel noch einige Beispiele für den Einsatz dieses Objekts.

4.3 Das Objekt »CodeData«

Das `CodeData`-Objekt verweist auf Objekte, die innerhalb der von der Serveranwendung (Jet oder SQL) verwalteten Codedatenbank gespeichert sind.

Das `CodeData`-Objekt besitzt mehrere Auflistungen, die bestimmte Objekttypen innerhalb der Codedatenbank enthalten.

Wie schon beim Objekt `AccessObject` gezeigt, können Sie auch mithilfe des Objekts `CodeData` alle Tabellen, seien sie nun geöffnet oder nicht, im Direktbereich der Entwicklungsumgebung ausgeben.

```
Sub TabellenAuflisten2()  
    Dim obj As AccessObject  
  
    For Each obj In Application.CodeData.AllTables  
        Debug.Print obj.Name  
    Next obj  
  
End Sub
```

Listing 4.22 Alle Tabellen auflisten

Neben der Auflistung `AllTables` existieren hier diejenigen Auflistungen, die ich bereits beim Objekt `AccessObject` in Tabelle 4.1 beschrieben habe.

4.4 Das Objekt »DoCmd«

Sie können das Objekt `DoCmd` verwenden, um Microsoft-Access-Aktionen aus Visual Basic heraus auszuführen. Eine Aktion führt Operationen durch, wie etwa das Schließen von Fenstern, das Öffnen von Berichten und Formularen oder das Festlegen der Werte von Steuerelementen.

4.4.1 Berichte aufrufen

In der Prozedur aus Listing 4.23 wird der Bericht *Alphabetische Artikelliste* geöffnet und die Seite 1 zweimal gedruckt. Danach wird der Bericht wieder geschlossen.

```
Sub BerichtÖffnenUndDrucken()  
    DoCmd.Echo False  
    DoCmd.OpenReport _  
        "Alphabetische Artikelliste", acViewPreview  
    DoCmd.PrintOut acPages, 1, 1, , 2  
    DoCmd.Close  
    DoCmd.Echo True  
End Sub
```

Listing 4.23 Bericht öffnen, drucken und schließen

Beim Ausführen einer Prozedur in Microsoft Access werden durch Bildschirmaktualisierungen oft Informationen angezeigt, die für die Funktionalität der Prozedur

ohne Bedeutung sind. Wenn Sie das Argument `Echo` auf `False` setzen, wird die Prozedur ohne Bildschirmaktualisierung ausgeführt. Beim Beenden der Prozedur schaltet Microsoft Access automatisch `Echo` wieder ein und aktualisiert das Fenster. Während der Laufzeit der Prozedur ändert sich demnach die Ansicht nicht.

Mit der Methode `OpenReport` öffnen Sie den angegebenen Bericht. Dabei können Sie auswählen, in welcher Ansicht der Bericht geöffnet werden soll. Zu diesem Zweck stehen Ihnen einige Konstanten zur Verfügung:

- ▶ Die Konstante `acViewPreview` ruft die Seitenansicht des Berichts auf.
- ▶ Über die Konstante `acViewDesign` wechseln Sie in den Entwurfsmodus des Berichts.
- ▶ Die Konstante `acViewNormal` gibt den Bericht sofort auf dem Drucker aus.

Die Methode `PrintOut` führt den Drucken-Befehl aus und hat folgende Syntax:

```
PrintOut(Druckbereich, Von, Bis, Druckqualität, Exemplare, ExemplareSortieren)
```

Im Argument `Druckbereich` geben Sie an, was Sie genau drucken möchten. Dabei stehen Ihnen folgende Konstanten zur Verfügung:

- ▶ Mit der Konstante `acPages` legen Sie fest, welche Seiten des Berichts Sie drucken möchten. Wenn Sie diese Konstante verwenden, müssen Sie die Argumente `Von` und `Bis` angeben.
- ▶ Mit der Konstante `acPrintAll` wird der gesamte Bericht gedruckt.
- ▶ Bei der Konstanten `acSelection` wird nur der Bereich gedruckt, der gerade markiert ist.

Für das Argument `Druckqualität` stehen wiederum einige Konstanten zur Verfügung:

- ▶ Die Konstante `acDraft` bezeichnet die Entwurfsqualität.
- ▶ Hinter der Konstanten `acHigh` verbirgt sich die bestmögliche Druckqualität.
- ▶ Die Konstante `acMedium` bezeichnet eine mittlere Druckqualität.
- ▶ Während Sie durch die Konstante `acLow` eine niedrige Druckqualität (Konzeptdruck) erreichen.

Das nächste Argument, `Exemplare`, sagt aus, wie viele Kopien gedruckt werden sollen. Beim letzten Argument, `ExemplareSortieren`, verwenden Sie die Einstellung `True`, um die Exemplare während des Druckvorgangs zu sortieren, und `False`, um sie nicht zu sortieren. Wenn Sie dieses Argument nicht angeben, wird der Standardwert `True` verwendet.

Beim Beispiel in Listing 4.24 wird das Objekt `DoCmd` dazu eingesetzt, einen Bericht aufzurufen und für diesen einen bestimmten Zoomfaktor einzustellen.

Sub BerichtAnzeigen()

```
DoCmd.OpenReport "Kundenetiketten", acPreview
```

```
DoCmd.Maximize
```

```
DoCmd.RunCommand acCmdZoom75
```

End Sub

Listing 4.24 Bericht öffnen und in einer größeren Ansicht anzeigen

Sie können die Methode `Maximize` verwenden, um das aktive Fenster zu vergrößern, sodass es das Microsoft-Access-Fenster ausfüllt. Mit dieser Aktion können Sie so viel wie möglich vom Objekt im aktiven Fenster anzeigen.

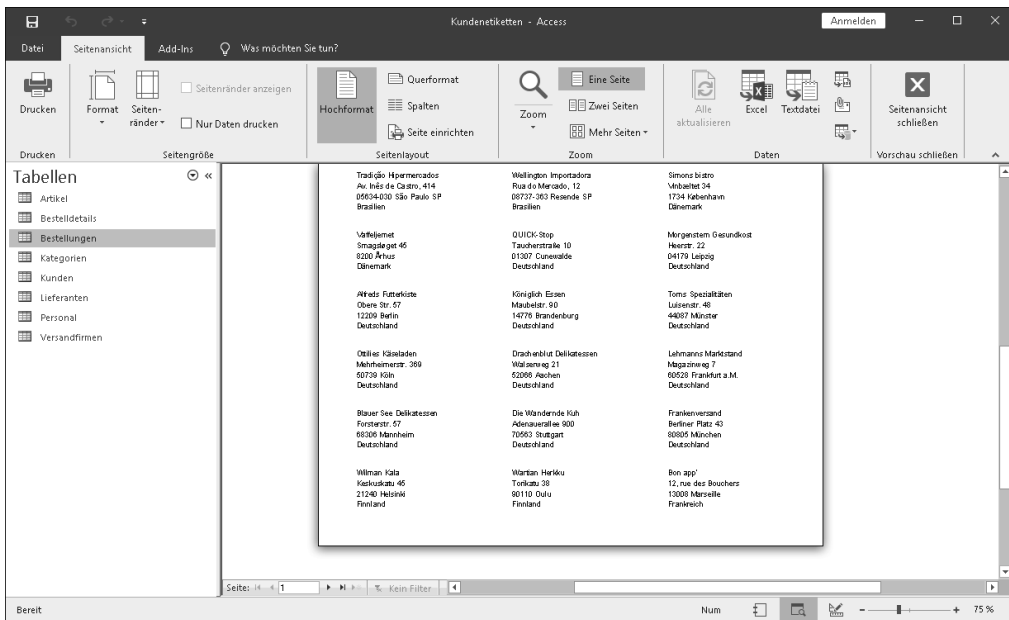


Abbildung 4.7 Einen Bericht in bestimmter Größe anzeigen

Setzen Sie die Methode `RunCommand` ein, um den Zoom einzustellen. Allgemein können Sie über diese Methode nahezu alle eingebauten Symbole und Menübefehle ausführen. Sie führen diesen Klick auf bestimmte Symbole und Menüs demnach »wie von Zauberhand« aus. Alles, was Sie dieser Methode noch bekannt geben müssen, ist das Symbol bzw. der Menübefehl, der angeklickt werden soll. In Access wird das über eine Konstante erreicht. Die Konstante für eine Ansicht von 75 % des Berichts lautet `acCmdZoom75`.

Weitere Konstanten finden Sie in der Onlinehilfe Ihrer Entwicklungsumgebung. Setzen Sie im Quellcode die Einfügemarka auf die Methode `RunCommand`, und drücken Sie die Taste `[F1]`. In der Onlinehilfe klicken Sie dann auf den Hyperlink `ACCOMMAND`.

4.4.2 Tabellen nach Excel exportieren

Eine weitere Einsatzmöglichkeit für das Objekt `DoCmd` ist der Transfer einer Access-Tabelle in eine Excel-Arbeitsmappe. Die Prozedur für diese Aufgabe lautet wie in Listing 4.25.

```
Sub TabelleTransferieren()
```

```
    DoCmd.TransferSpreadsheet acExport, _
        acSpreadsheetTypeExcel9, "Artikel", _
        Application.CurrentProject.Path & "\Artikel.xls", True
```

```
End Sub
```

Listing 4.25 Access-Tabelle nach Excel exportieren

Um eine Access-Datentabelle in eine Excel-Arbeitsmappe oder auch in eine Lotus-Tabelle zu übertragen, setzen Sie die Methode `TransferSpreadsheet` ein.

Die Methode `TransferSpreadsheet` hat folgende Syntax:

```
TransferSpreadsheet(Transfertype, Dateiformat, Tabellenname, Dateiname,
BesitztFeldnamen, Bereich)
```

Im Argument `Transfertype` geben Sie an, welchen Transfer Sie genau durchführen möchten. Sie haben dabei die Auswahl zwischen dem Export (`acExport`), dem Import (`acImport`) oder einer Verknüpfung (`acLink`).

Im Argument `Dateiformat` bestimmen Sie, in welche Excel-Version (bzw. Lotus-Version) Sie die Access-Tabelle exportieren möchten.

Im darauffolgenden Argument `TabelleName` geben Sie den Namen der zu exportierenden Access-Tabelle an. Über das Argument `Dateiname` geben Sie das Ziel für den Datenexport bekannt. Dabei muss die angegebene Datenquelle nicht einmal existieren. Access legt sie neu für Sie an.

Beim Argument `BesitztFeldnamen` verwenden Sie den Wert `True`, um die erste Zeile der Kalkulationstabelle beim Importieren, Exportieren oder Verknüpfen zur Angabe der Feldnamen zu verwenden. Verwenden Sie hingegen den Wert `False`, wenn die erste Zeile als normale Datenzeile gelten soll. Wenn Sie dieses Argument nicht angeben, wird der Standardwert `False` verwendet.

| Artikel-Nr | Artikelname | Lieferant | Kategorie | Lieferereinheit | Einzelpreis | Lagerbestand | Bestellmenge | Mindestbestellmenge | Auslaufartikel |
|------------|----------------|-----------|-----------|------------------------|-------------|--------------|--------------|---------------------|----------------|
| 1 | 1 Chai | 11 | 3 | 10 Kartons x 20 Beutel | | | | 10 | WAHR |
| 2 | 2 Chang | 1 | 1 | 24 x 12-oz- | 19,00 | 17 | 40 | 25 | FALSCH |
| 3 | 3 Aniseed S | 1 | 2 | 12 x 550-m | 10,00 | 13 | 70 | 25 | FALSCH |
| 4 | 4 Chef Antor | 2 | 2 | 48 x 6-oz-C | 22,00 | 53 | 0 | 0 | FALSCH |
| 5 | 5 Chef Antor | 2 | 2 | 36 Kartons | 21,35 | 0 | 0 | 0 | WAHR |
| 6 | 6 Grandma | 3 | 2 | 12 x 8-oz-C | 25,00 | 120 | 0 | 25 | FALSCH |
| 7 | 7 Uncle Bob | 3 | 7 | 12 x 1-lb-P | 30,00 | 15 | 0 | 10 | FALSCH |
| 8 | 8 Northwood | 3 | 2 | 12 x 12-oz- | 40,00 | 6 | 0 | 0 | FALSCH |
| 9 | 9 Mishi Kobi | 4 | 6 | 18 x 500-g | 97,00 | 29 | 0 | 0 | WAHR |
| 10 | 10 Ikura | 4 | 8 | 12 x 200-m | 31,00 | 31 | 0 | 0 | FALSCH |
| 11 | 11 Queso Ca | 5 | 4 | 1-kg-Pake | 21,00 | 22 | 30 | 30 | FALSCH |
| 12 | 12 Queso Ma | 5 | 4 | 10 x 500-g | 38,00 | 86 | 0 | 0 | FALSCH |
| 13 | 13 Konbu | 6 | 8 | 2-kg-Karto | 6,00 | 24 | 0 | 5 | FALSCH |
| 14 | 14 Tofu | 6 | 7 | 40 x 100-g | 23,25 | 35 | 0 | 0 | FALSCH |
| 15 | 15 Genen Sh | 6 | 2 | 24 x 250-m | 15,50 | 39 | 0 | 5 | FALSCH |
| 16 | 16 Pavlova | 7 | 3 | 32 x 500-g | 17,45 | 29 | 0 | 10 | FALSCH |
| 17 | 17 Alice Mutte | 7 | 6 | 20 x 1-kg-C | 39,00 | 0 | 0 | 0 | WAHR |
| 18 | 18 Carnarvon | 7 | 8 | 16-kg-Pak | 62,50 | 42 | 0 | 0 | FALSCH |
| 19 | 19 Teatime C | 8 | 3 | 10 Kartons | 9,20 | 25 | 0 | 5 | FALSCH |
| 20 | 20 Sir Rodney | 8 | 3 | 30 Gesche | 81,00 | 40 | 0 | 0 | FALSCH |
| 21 | 21 Sir Rodney | 8 | 3 | 24 Packun | 10,00 | 3 | 40 | 5 | FALSCH |
| 22 | 22 Gustaf's Ki | 9 | 5 | 24 x 500-g | 21,00 | 104 | 0 | 25 | FALSCH |
| 23 | 23 Tunnbröd | 9 | 5 | 12 x 250-g | 9,00 | 61 | 0 | 25 | FALSCH |
| 24 | 24 Guarani F | 10 | 1 | 12 x 355-m | 4,50 | 20 | 0 | 0 | WAHR |

Abbildung 4.8 Eine Access-Tabelle wurde in eine Excel-Tabelle transferiert.



Argument »Bereich« gilt nur für den Import

Das Argument `Bereich` gilt nur für Importoperationen und darf beim Export nicht angegeben werden. Beim Import werden standardmäßig alle Datensätze importiert, sofern dieses Argument nicht gesetzt wird.

4.4.3 Formular aufrufen und Vorauswahl treffen

Im nächsten Beispiel soll das Formular *Artikel* geöffnet werden. Dabei sollen nur Artikel des Lieferanten Nr. 4 voreingestellt sein, in unserer Beispieldatenbank ist das die Lieferantenummer des Lieferanten »Tokyo Traders«. Die Lösung für diese Aufgabe können Sie Listing 4.26 entnehmen.

Sub `FormularOeffnen()`

```
DoCmd.OpenForm "Artikel", , , "[Lieferanten-Nr] = 4"
```

End Sub

Listing 4.26 Formular öffnen und filtern

Die Methode `OpenForm` setzen Sie ein, um ein Formular zu öffnen. Diese Methode hat folgende Syntax:

```
OpenForm(Formularname, Ansicht, Filtername, Bedingung, Datenmodus, Fenstermodus)
```

Im Argument `Formularname` geben Sie an, welches Formular Sie öffnen möchten. Das Argument `Ansicht` können Sie leer lassen, wenn Sie Ihr Formular in der Normalansicht, also per Doppelklick, öffnen möchten. Ansonsten haben Sie die folgenden Ansichtskonstanten zur Verfügung:

- ▶ `acDesign` öffnet das Formular in der Entwurfsansicht.
- ▶ `acFormDS` öffnet das Formular in einer Tabellenansicht.
- ▶ `acFormPivotChart` stellt das Formular für ein Pivot-Diagramm zur Verfügung.
- ▶ Mit `acFormPivotTable` können Sie die Felder des Formulars für eine Pivot-Tabelle verwenden.
- ▶ `acNormal` öffnet das Formular in gewohnter Weise (Standardeinstellung).
- ▶ `acPreview` zeigt das Formular in der Seitenansicht an.

Über das Argument `Filtername` stellen Sie ein, welche Sätze im Formular nach dem Starten angezeigt werden sollen. Dabei geben Sie den Namen des Feldes an, das Sie filtern möchten. Dazu gehört ebenso das Argument `Bedingung`. Dort formulieren Sie die Bedingung für den Filter.

Im Argument `Datenmodus` können Sie bestimmen, welche Aktionen Anwender im Formular durchführen dürfen. Zu diesem Zweck stehen Ihnen folgende Konstanten zur Verfügung:

- ▶ Bei Angabe der Konstante `acFormAdd` dürfen Anwender neue Daten ins Formular eingeben.
- ▶ `acFormEdit` erlaubt ihnen Änderungen an Feldinhalten.
- ▶ Bei der Standardeinstellung `acFormPropertySettings` dürfen Anwender über das Formular Daten ändern, einfügen und löschen.
- ▶ Während sie bei `acFormReadOnly` keinerlei Änderungen an den Feldinhalten des Formulars vornehmen dürfen.

Im Argument `Fenstermodus` geben Sie an, wie das Formular angezeigt werden soll. Dabei können Sie auswählen, ob Sie das Formular als Dialog (`acDialog`), versteckt (`acHidden`), als Symbol in der Taskleiste (`acIcon`) oder standardmäßig (`acWindowNormal`) anzeigen möchten.

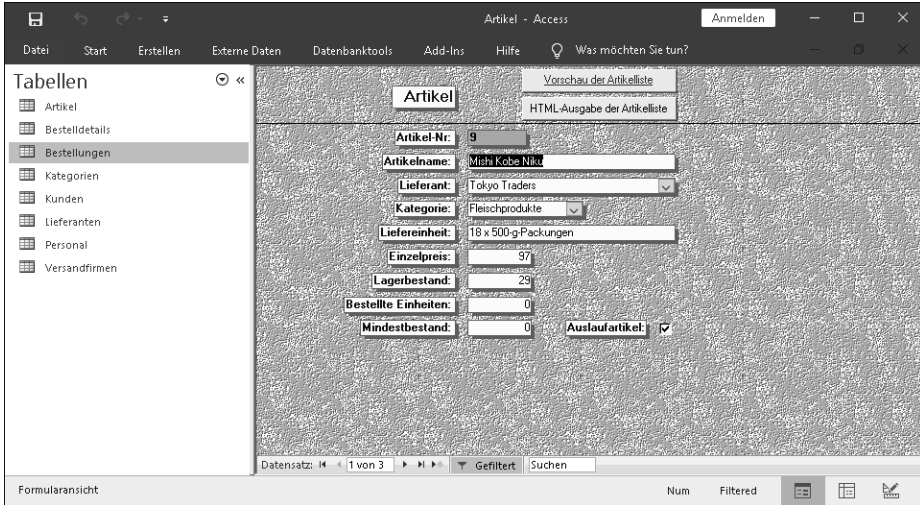


Abbildung 4.9 Formular mit Filterung aufrufen

Das Objekt DoCmd wurde in der neuen Version um mehrere Methoden angereichert:

► **Schließen einer Datenbank**

Mit der Methode CloseCurrentDatabase schließen Sie die aktuelle Datenbank.

► **Schützen der Navigationsleiste**

Über die Methode LockNavigationPane können Sie verhindern, dass Tabellen, Abfragen oder Berichte gelöscht werden. Die Syntax hierfür lautet:

```
DoCmd.LockNavigationPane (true)
```

► **Erweiterte Suchmöglichkeit**

Die Methode SearchForRecord dient dazu, Datensätze anhand von etwas komplexeren Bedingungen aufzuspüren. Im folgenden Beispiel in Listing 4.27 wird das Formular *Artikel* geöffnet und der erste Satz mit einem Lagerbestand von mehr als 10 Stück des Lieferanten mit der Nummer 11 im Formular angezeigt.

```
Sub SucheKomplex()  
  
DoCmd.OpenForm "Artikel"  
DoCmd.SearchForRecord acDataForm, "Artikel", acNext, _  
"Lagerbestand>10 and [Lieferanten-Nr]=11"  
  
End Sub
```

Listing 4.27 Formular öffnen, Datensatz suchen und anzeigen

Die Syntax der Methode SearchForRecord lautet:

```
SearchForRecord(ObjectType, ObjectName, Record, WhereCondition)
```

Im Argument `ObjectType` geben Sie den Objekttyp an:

- ▶ `acActiveDataObject`: Das aktive Objekt enthält den Datensatz.
- ▶ `acDataForm`: Ein Formular enthält den Datensatz.
- ▶ `acDataFunction`: Eine benutzerdefinierte Funktion enthält den Datensatz
- ▶ `acDataQuery`: Eine Abfrage enthält den Datensatz.
- ▶ `acDataReport`: Ein Bericht enthält den Datensatz.
- ▶ `acDataServerView`: Eine Serversicht enthält den Datensatz (nur Microsoft-Access-Projekt).
- ▶ `acDataStoredProcedure`: Eine gespeicherte Prozedur enthält den Datensatz (nur Microsoft-Access-Projekt).
- ▶ `acDataTable`: Eine Tabelle enthält den Datensatz.

Im Argument `ObjectName` geben Sie den Namen des Objekts an.

Das Argument `Record` bestimmt den Anfangspunkt und die Richtung der Suche. Der Standardwert lautet `acFirst`.

Im letzten Argument, `WhereCondition`, geben Sie die Bedingung an, anhand der gesucht werden soll.

4.5 Integrierte Dialoge einsetzen

Bei der Programmierung mit Dialogen müssen Sie nicht unbedingt eigene Dialoge erstellen. Oft reicht es auch aus, wenn Sie bereits vorhandene Dialoge in Access für Ihre eigenen Projekte nutzen. Ein weiterer Vorteil ist, dass Anwender diese Dialoge bereits kennen und sich nicht in fremde Dialoge einarbeiten müssen.

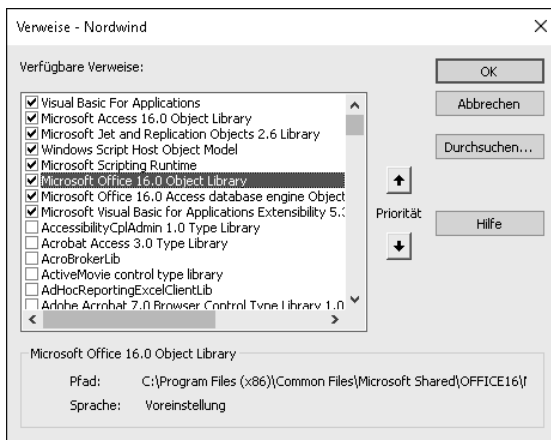


Abbildung 4.10 So setzen Sie einen Verweis auf die »Microsoft Office Object Library« im Menü »Extras • Verweise«.

Bitte beachten Sie, dass Sie für einige der folgenden Beispiele einen Verweis auf die *Microsoft Office Object Library* hinzufügen müssen.

4.5.1 Das Dialogfeld »Öffnen« anzeigen

Wenn Sie aus dem Menü DATEI den Befehl ÖFFNEN aufrufen, zeigen Sie damit das Dialogfeld ÖFFNEN an.

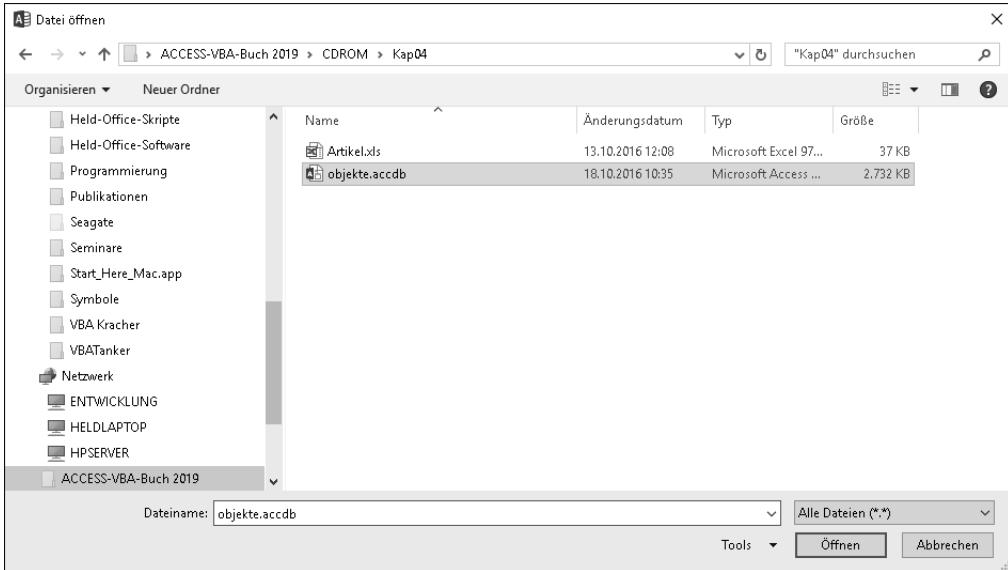


Abbildung 4.11 Das Dialogfeld »Öffnen«

Ein solches Dialogfeld können Sie mitsamt der kompletten Steuerung auch über VBA-Code anzeigen lassen. Sehen Sie sich dazu die Prozedur in Listing 4.28 an.

```
Sub DialogOeffnen()  
    Dim fd As FileDialog  
    Dim varAusgewaehlt As Variant  
  
    Set fd = Application.FileDialog(msoFileDialogOpen)  
  
    With fd  
        If .Show = -1 Then  
            For Each varAusgewaehlt In .SelectedItems  
                MsgBox "Sie haben ausgewählt: " & _  
                    & varAusgewaehlt  
            Next varAusgewaehlt  
        End If  
    End With  
End Sub
```

```

End With

Set fd = Nothing
End Sub

```

Listing 4.28 Das Dialogfeld »Öffnen« anzeigen und auswerten

Im ersten Schritt deklarieren Sie eine Variable als `FileDialog`-Objekt. Danach benötigen Sie ein Array. In diesem Array (`varAusgewaehlt`) werden Sie später die ausgewählten Dateien im Dialogfeld ÖFFNEN verwalten. Im Anschluss an die Variablendeklaration erzeugen Sie das Objekt `FileDialog` und geben ihm die Konstante `msoFileDialogOpen` mit. Dadurch weiß Access, dass das Dialogfeld ÖFFNEN angezeigt werden soll.

Weitere mögliche Konstanten für integrierte Dialoge sind übrigens `msoFileDialogFilePicker` (Dialogfeld DURCHSUCHEN mit Dateiauswahlmöglichkeit), `msoFileDialogFolderPicker` (Dialogfeld DURCHSUCHEN mit Verzeichnisauswahlmöglichkeit) und `msoFileDialogSaveAs` (Dialogfeld SPEICHERN UNTER).

Über die Methode `Show` zeigen Sie den Dialog an. Dabei können Sie sich auch anzeigen lassen, welche Schaltfläche auf dem Dialogfeld angeklickt wurde. Falls ein Anwender auf die Schaltfläche ÖFFNEN klickt, wird der Wert `-1` zurückgegeben. Andernfalls, also wenn der Anwender die Schaltfläche ABBRECHEN klickt oder die Taste `[Esc]` drückt, meldet Access den Wert `0`.

Mit einer `For Each`-Schleife durchlaufen Sie alle im Dialogfeld markierten Dateien, die Sie automatisch über die Eigenschaft `SelectedItems` abfragen können. Die ausgewählten Dateien geben Sie über ein Meldungsfenster am Bildschirm aus.

Vergessen Sie nicht, am Ende das Schlüsselwort `Nothing` einzusetzen, um die Verbindung der Objektvariablen `fd` zum zugehörigen Objekt `FileDialog` aufzuheben. Dadurch werden die Speicher- und Systemressourcen wieder freigegeben, die für dasjenige Objekt reserviert wurden, auf das die Variablen verweisen.

Möchten Sie das Dialogfeld DURCHSUCHEN anzeigen und auswerten lassen, dann setzen Sie die Prozedur aus Listing 4.29 ein.

```

Sub DialogSuchen()
    Dim fd As FileDialog
    Dim varAusgewaehlt As Variant

    Set fd = _
        Application.FileDialog(msoFileDialogFilePicker)
    With fd
        If .Show = -1 Then
            For Each varAusgewaehlt In .SelectedItems

```



```
        MsgBox "Sie haben ausgewählt: " & varAusgewaehlt
    Next varAusgewaehlt
End If
End With
Set fd = Nothing
End Sub
```

Listing 4.29 Das Dialogfeld »Durchsuchen« anzeigen und auswerten

4.5.2 Verzeichnis einstellen

Möchten Sie es den Anwendern überlassen, welches Verzeichnis sie standardmäßig zur Verfügung gestellt bekommen, wenn sie aus dem Menü DATEI den Befehl ÖFFNEN wählen, dann hilft Ihnen die Prozedur aus Listing 4.30 weiter. Dabei haben die Anwender die Möglichkeit, ihr Standardverzeichnis aus dem Dialog DURCHSUCHEN zu wählen.

```
Sub OrdnerEinstellen()
    Dim fd As FileDialog

    Set fd = Application.FileDialog(msoFileDialogFolderPicker)

    With fd
        If .Show = -1 Then
            MsgBox "Sie haben eingestellt: " _
                & fd.SelectedItems(1)
            'Ins ausgewählte Verzeichnis wechseln
            ChDir fd.SelectedItems(1)
        End If
    End With
    Set fd = Nothing
End Sub
```

Listing 4.30 Ordner über das Dialogfeld »Durchsuchen« einstellen

Im ersten Schritt deklarieren Sie eine Variable als `FileDialog`-Objekt. Im Anschluss an die Variablendeklaration erzeugen Sie das `FileDialog`-Objekt und geben ihm die Konstante `msoFileDialogFolderPicker` mit. Dadurch »weiß« Access, dass das Dialogfeld DURCHSUCHEN angezeigt werden soll.

Über die Methode `Show` zeigen Sie das Dialogfeld an. Dabei können Sie zusätzlich sehen, welche Schaltfläche auf dem Dialogfeld geklickt wurde. Falls eine Anwenderin auf die Schaltfläche OK klickt, wird der Wert `-1` zurückgegeben. Andernfalls – also

wenn die Anwenderin auf die Schaltfläche **ABBRECHEN** klickt oder die Taste **Esc** drückt – meldet Access den Wert 0.

Über die Eigenschaft `SelectedItem` können Sie abfragen, welcher Ordner im Dialogfeld **DURCHSUCHEN** gewählt wurde. Den ausgewählten Ordner geben Sie über ein Meldungsfenster am Bildschirm aus. Direkt im Anschluss daran wechseln Sie mit der Anweisung `ChDir` in den ausgewählten Ordner.

Setzen Sie am Ende das Schlüsselwort `Nothing` ein, um die Referenz der Objektvariablen `fd` zum zugehörigen Objekt `FileDialog` aufzuheben und reservierte Ressourcen wieder freizugeben.

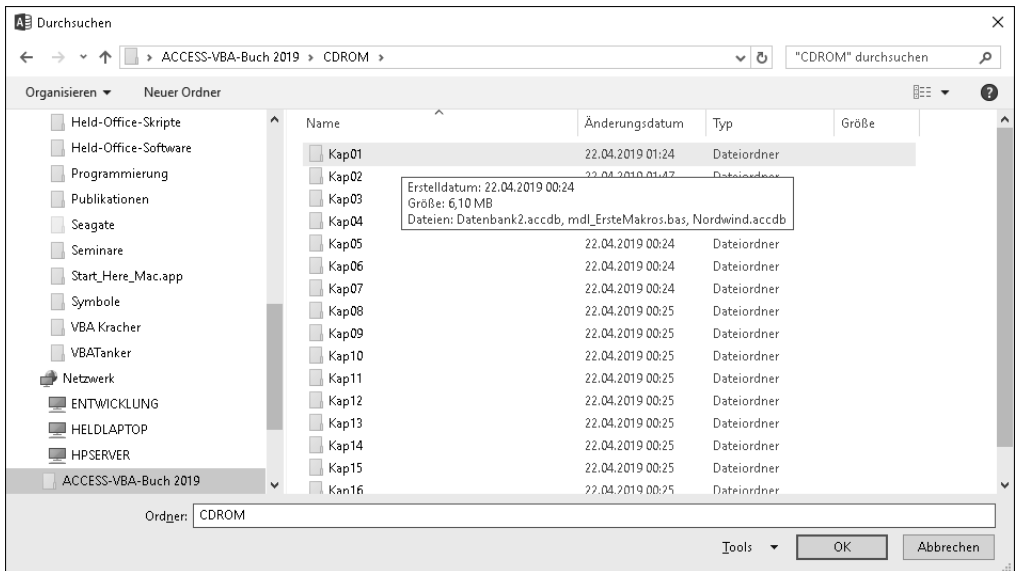


Abbildung 4.12 Der Dialog »Durchsuchen«

4.5.3 Dateien per Filtereinstellung suchen

Das Dialogfeld **DURCHSUCHEN** können Sie einsetzen, um Dateien eines bestimmten Dateityps anzuzeigen.

In Listing 4.31 wird das Dialogfeld **DURCHSUCHEN** angezeigt und listet per eigener Definition nur HTML-Dateien auf.

```
Sub DateienSuchenMitFiltereinstellung()
    Dim fd As FileDialog
    Dim varAusgewaehlt As Variant

    Set fd = Application.FileDialog(msoFileDialogFilePicker)
    With fd
```

```
.Filters.Clear
.Filters.Add "Internet", "*.htm; *.html", 1
If .Show = -1 Then
    For Each varAusgewaehlt In .SelectedItems
        MsgBox "Pfadname: " & varAusgewaehlt
    Next varAusgewaehlt
End If
End With
Set fd = Nothing
```

End Sub

Listing 4.31 Das Dialogfeld »Durchsuchen« mit Voreinstellung anzeigen

Im ersten Schritt deklarieren Sie eine Variable als `FileDialog`-Objekt. Im Anschluss an die Variablendeklaration erzeugen Sie das Objekt `FileObject` und geben ihm die Konstante `msoFileDialogFilePicker` mit. Dadurch »weiß« Access, dass das Dialogfeld DURCHSUCHEN angezeigt werden soll.

Um auf das Dropdown-Listenfeld `DATEITYP` zugreifen zu können, müssen Sie die Eigenschaft `Filters` verwenden. Um den Filter zu initialisieren, wenden Sie die Methode `Clear` an. Danach definieren Sie sich Ihren eigenen Filter über die Methode `Add`. Dadurch fügen Sie einen neuen Dateifilter der Liste der Filter im Dropdown-Listenfeld `DATEITYP` des Dialogfeldes DURCHSUCHEN hinzu.

Möchten Sie im Dropdown-Feld alle Dateien zur Auswahl haben, dann geben Sie das Filterkriterium wie folgt an:

```
.Filters.Add "Alle Dateien", "*.*"
```

Über die Methode `Show` zeigen Sie das Dialogfeld an. Dabei können Sie sich zusätzlich anzeigen lassen, welche Schaltfläche auf dem Dialogfeld angeklickt wurde. In dem Fall, dass ein Anwender auf die Schaltfläche OK klickt, wird der Wert -1 zurückgegeben. Im anderen Fall – also wenn der Anwender auf die Schaltfläche ABBRECHEN klickt oder die Taste `[Esc]` drückt – wird der Wert 0 von Access gemeldet.

Mit einer `For Each`-Schleife durchlaufen Sie alle markierten Dateien im Dialogfeld, die Sie automatisch über die Eigenschaft `SelectedItems` abfragen können. Die ausgewählten Dateien geben Sie über ein Meldungsfenster am Bildschirm aus.

Setzen Sie am Ende das Schlüsselwort `Nothing` ein, um die Referenz der Objektvariablen `fd` zum zugehörigen Objekt `FileDialog` aufzuheben und reservierte Ressourcen wieder freizugeben.

4.5.4 Weitere Dialogfelder verwenden

Selbstverständlich haben Sie auch die Möglichkeit, nahezu alle anderen integrierten Dialogfelder von Access für Ihre Arbeit zu nutzen. Dazu setzen Sie die Methode Run-Command ein. Diese Methode führt einen eingebauten Menü- oder Symbolleistenbefehl aus, was bedeutet, dass Sie in der Lage sind, fast jedes Dialogfeld von Access aufzurufen.

In Listing 4.32 rufen Sie das Dialogfeld OPTIONEN auf.

```
Sub DialogOptionenEinstellen()  
  
    DoCmd.RunCommand acCmdOptions
```

End Sub

Listing 4.32 Das Dialogfeld »Optionen« aufrufen

Diese Prozedur ist gleichbedeutend mit der manuellen Vorgehensweise, bei der Sie aus dem Menü den Dialog ACCESS-OPTIONEN wählen.

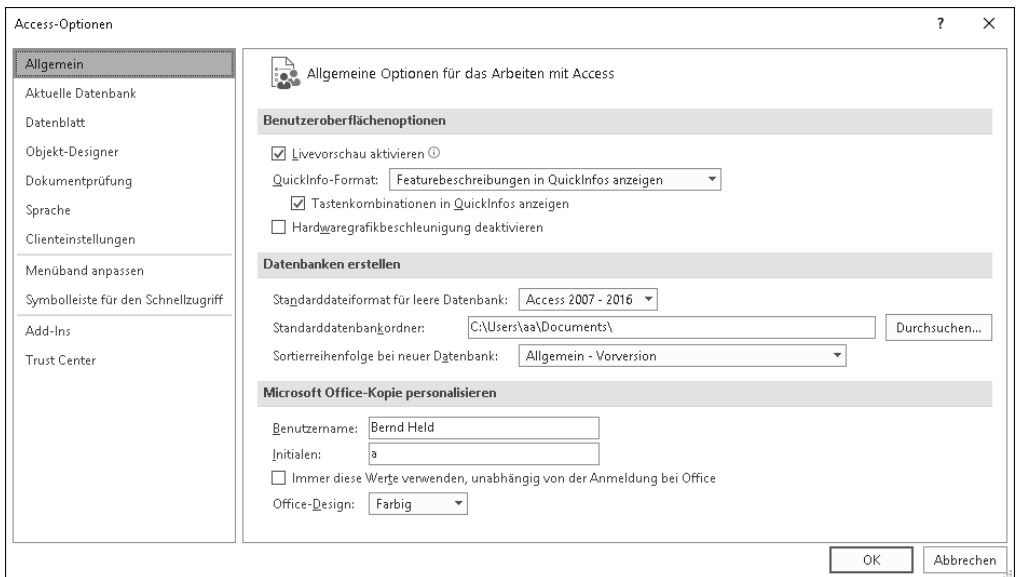


Abbildung 4.13 Das Dialogfeld »Access-Optionen« über VBA aufrufen

Entnehmen Sie Tabelle 4.2, wie Sie die anderen Registerkarten per VBA öffnen können.

| Registerkarte | Befehl |
|---------------|---|
| ABFRAGEN | DoCmd.RunCommand acCmdViewQueries |
| FORMULARE | DoCmd.RunCommand acCmdViewForms |
| BERICHTE | DoCmd.RunCommand acCmdViewReports |
| SEITEN | DoCmd.RunCommand acCmdViewDataAccessPages |
| MAKROS | DoCmd.RunCommand acCmdViewMacros |
| MODULE | DoCmd.RunCommand acCmdViewModules |

Tabelle 4.2 Mögliche Ansichtskonstanten der Methode »RunCommand«

4.6 Das Objekt »FileSystemObject«

Über das `FileSystemObject` gewinnen Sie Zugriff auf Ihre Laufwerke und Ihr Netzwerk. Dieses Objekt ist Bestandteil der Bibliothek *Microsoft Scripting Runtime*. Diese Bibliothek enthält Befehle für den Windows Scripting Host (WSH). Dabei handelt es sich um eine eigene Skriptsprache unter Windows, die Sie auch von Access aus aufrufen können. Skripte können Sie einsetzen, um tägliche Routineaufgaben zu automatisieren – wie das Sichern von Dateien, das Starten von Programmen, das Öffnen und die Bearbeitung von Dateien, die Abfrage von Systeminformationen, das Erstellen und Löschen von Verzeichnissen, die Verbindung und das Trennen von Netzlaufwerken und vieles mehr.

Bevor Sie aber mit diesem Objekt arbeiten, sollten Sie erst einmal die entsprechende Objektbibliothek einbinden und im Objektkatalog nachsehen, welche Methoden und Eigenschaften zur Verfügung stehen.

Um die Bibliothek *Microsoft Scripting Runtime* einzubinden, befolgen Sie die folgenden Arbeitsschritte:

1. Wählen Sie in der Entwicklungsumgebung den Befehl VERWEISE aus dem Menü EXTRAS.
2. Wählen Sie aus dem Listenfeld VERFÜGBARE VERWEISE die Bibliothek MICROSOFT SCRIPTING RUNTIME.
3. Bestätigen Sie Ihre Einstellung mit OK.
4. Drücken Sie gleich im Anschluss die Taste `[F2]`, um den Objektkatalog aufzurufen.
5. Stellen Sie im ersten Dropdown-Feld den Eintrag SCRIPTING ein.

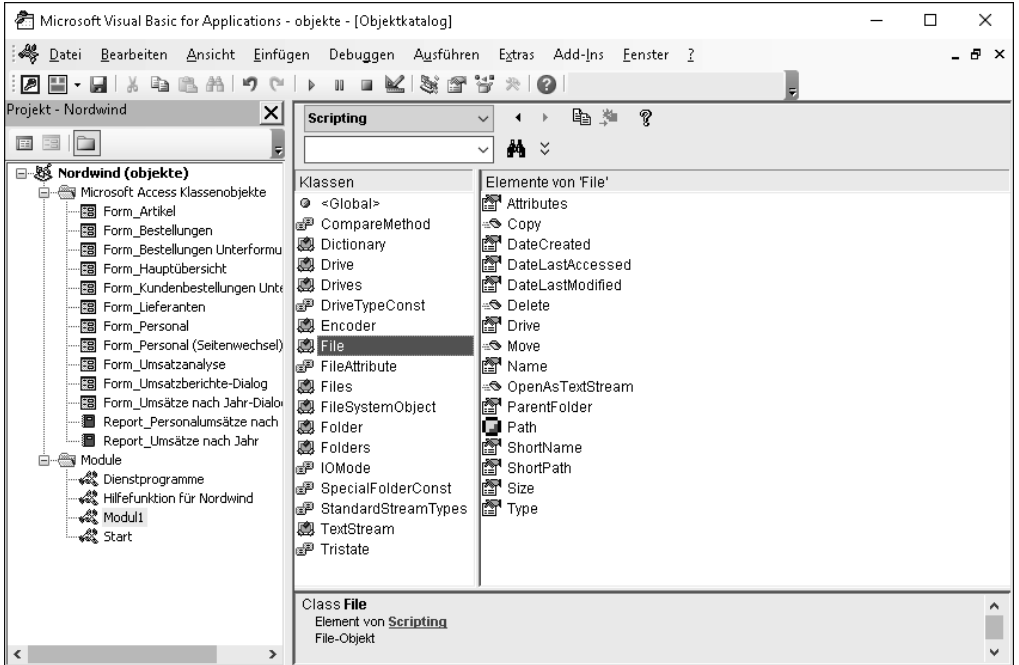


Abbildung 4.14 Alle Methoden und Eigenschaften für die Dateibearbeitung

4.6.1 Computerinformationen anzeigen

Möchten Sie den Namen Ihres Computers sowie den Namen des angemeldeten Benutzers erfahren, dann können Sie die Prozedur aus Listing 4.33 ausführen.

```
Sub ComputerInfosAnzeigen()
    Dim fsNetzwerk As Variant

    Set fsNetzwerk = CreateObject("wscript.network")
    Debug.Print "Computer: " & fsNetzwerk.Computername
    Debug.Print "User   : " & fsNetzwerk.UserName

End Sub
```

Listing 4.33 Computernamen und Anwendernamen ermitteln

Um Informationen bezüglich Ihres Computers abzufragen, müssen Sie auf das Objekt `network` zurückgreifen. Dieses Objekt erstellen Sie mit der Anweisung `CreateObject`. Um Schreibarbeit zu sparen, setzen Sie die Anweisung `Set` ein. Hiermit weisen Sie das Objekt einer Variablen mit dem Namen `Netzwerk` zu. Dies ermöglicht Ihnen später, unter dem Alias `Netzwerk` auf dieses Objekt zuzugreifen.

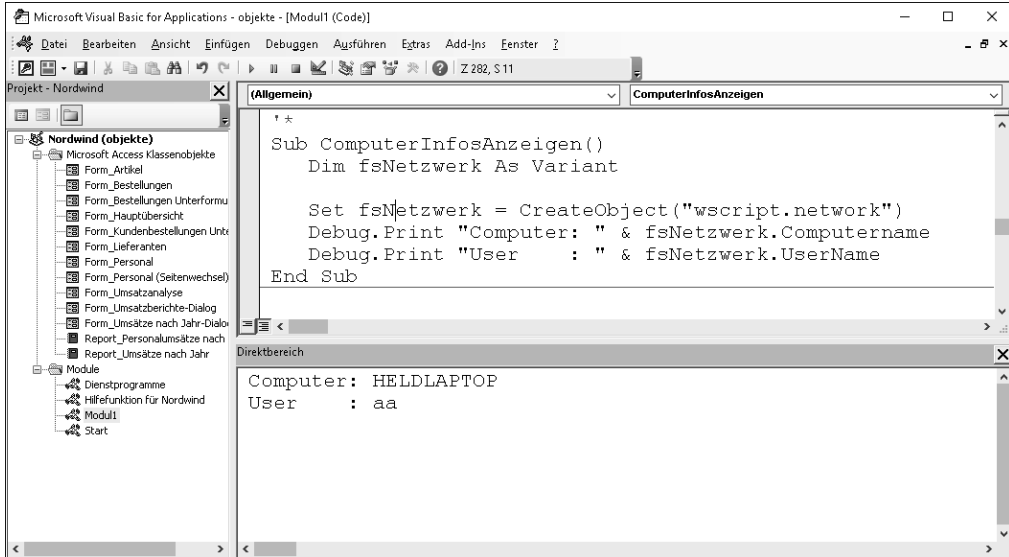


Abbildung 4.15 Den Computernamen und den Anwendernamen im Direktbereich ausgeben

4.6.2 Verzeichnisse ermitteln

Wenn Sie mehrere Benutzer auf Ihrem PC zugelassen haben, ist es gar nicht so einfach, die persönlichen Verzeichnisse auf dem PC zu finden. Um diese Verzeichnisse zu ermitteln, können Sie die Prozedur aus Listing 4.34 einsetzen.

```
Sub VerzeichnisseErmitteln()  
    Dim wshshShell As WshShell  
  
    Set wshshShell = CreateObject("WScript.Shell")  
  
    Debug.Print "Ihr Desktop liegt unter:" & Chr(13) _  
        & wshshShell.SpecialFolders("Desktop") & Chr(13)  
  
    Debug.Print "Ihre Verlaufsliste liegt unter: " & Chr(13) _  
        & wshshShell.SpecialFolders("Recent") & Chr(13)  
  
    Debug.Print "Ihre Favoriten liegen unter: " & Chr(13) _  
        & wshshShell.SpecialFolders("Favorites") & Chr(13)  
  
    Debug.Print "Das Verzeichnis <<Eigene Dateien>> ist: " & Chr(13) _  
        & wshshShell.SpecialFolders("Mydocuments") & Chr(13)
```

```

Debug.Print "Ihr Startverzeichnis befindet sich im " _
    & " Ordner: " & Chr(13) _
    & wshshShell.SpecialFolders("StartMenu") & Chr(13)

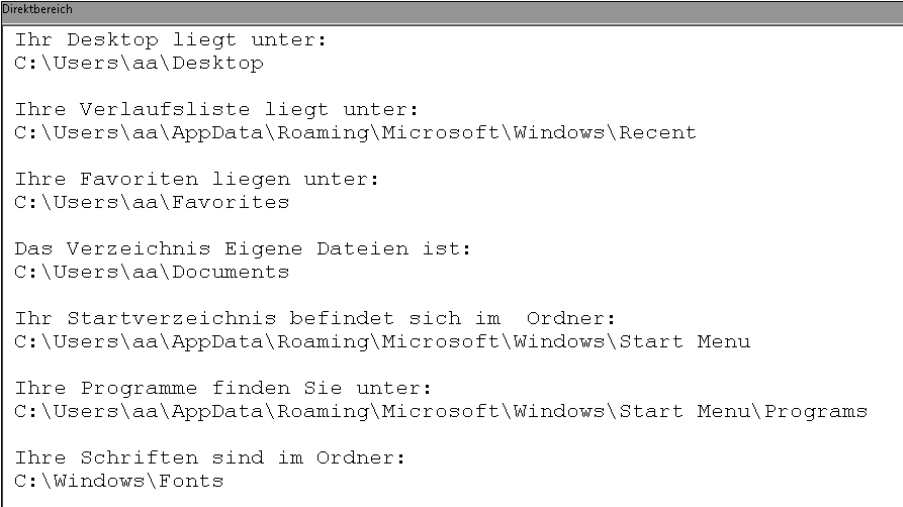
Debug.Print "Ihre Programme finden Sie unter: " & Chr(13) _
    & wshshShell.SpecialFolders("Programs") & Chr(13)

Debug.Print "Ihre Schriften sind im Ordner: "& Chr(13) _
    & wshshShell.SpecialFolders("Fonts") & Chr(13)
End Sub

```

Listing 4.34 Die Standardverzeichnisse auswerten

Erstellen Sie das Objekt `WScript.Shell` mit der Methode `CreateObject`. Das Objekt `WScript.Shell` hat mehrere Eigenschaften. Beispielsweise können Sie die Eigenschaft `SpecialFolders` einsetzen, um Ihre Standardverzeichnisse bekannt zu geben. Die Eigenschaft `SpecialFolders` kennt bestimmte Konstanten, die Sie abfragen können. Unter anderem ist das die Konstante `Fonts`, die auf den Ordner mit den installierten Schriftarten zeigt.



```

Direktbereich
Ihr Desktop liegt unter:
C:\Users\aa\Desktop

Ihre Verlaufsliste liegt unter:
C:\Users\aa\AppData\Roaming\Microsoft\Windows\Recent

Ihre Favoriten liegen unter:
C:\Users\aa\Favorites

Das Verzeichnis Eigene Dateien ist:
C:\Users\aa\Documents

Ihr Startverzeichnis befindet sich im Ordner:
C:\Users\aa\AppData\Roaming\Microsoft\Windows\Start Menu

Ihre Programme finden Sie unter:
C:\Users\aa\AppData\Roaming\Microsoft\Windows\Start Menu\Programs

Ihre Schriften sind im Ordner:
C:\Windows\Fonts

```

Abbildung 4.16 Die Windows-Verzeichnisse abfragen

4.6.3 Tastenkombinationen programmieren

Möchten Sie ein Programm, beispielsweise den Windows-Explorer, über eine Tastenkombination aufrufen, dann führt Sie die Prozedur aus Listing 4.35 zum Ziel.


```
Sub ShortcutErzeugen()  
    Dim wshshShell As WshShell  
    Dim strDesktop As String  
    Dim wshscShortCut As WshShortcut  
  
    Set wshshShell = CreateObject("WScript.Shell")  
    strDesktop = wshshShell.SpecialFolders("Desktop")  
    Set wshscShortCut = _  
    wshshShell.CreateShortcut(strDesktop & "\Explorer.lnk")  
    wshscShortCut.TargetPath = "%windir%\explorer.exe"  
    wshscShortCut.Hotkey = "ALT+CTRL+L"  
    wshscShortCut.Save  
    Debug.Print "Der Shortcut ist: " & wshscShortCut.Hotkey  
  
End Sub
```

Listing 4.35 Einen Shortcut erzeugen

Nach dem Starten dieser Prozedur wird eine Verknüpfung auf Ihrem Desktop erzeugt, über die Sie ganz schnell Ihren Windows-Explorer starten können. Zusätzlich können Sie den Explorer jetzt über die Tastenkombination **[Strg]+[Alt]+[L]** aufrufen.

Erstellen Sie zuerst das Objekt `Wscript.Shell` mit der Methode `CreateObject`. Danach definieren Sie als Speicherort für Ihre Verknüpfung zum Explorer Ihren Windows-Desktop. Dazu setzen Sie die Eigenschaft `SpecialFolders` ein. Jetzt erzeugen Sie mit der Methode `CreateShortcut` eine Verknüpfung auf Ihrem Desktop.

Achten Sie darauf, dass Sie beim Namen der Verknüpfung die Endung `.lnk` oder `.url` anfügen, sonst kommt es zu einer Fehlermeldung. Nun müssen Sie ermitteln, wo das Programm *Explorer* gespeichert ist. Dazu setzen Sie die Eigenschaft `TargetPath` ein und geben mit der Umgebungsvariablen `%windir%` an, dass sich das Programm im Windows-Verzeichnis befindet. Übergeben Sie zusätzlich zur Speicheradresse den vollständigen Namen mit Endung. Mit der Eigenschaft `Hotkey` bestimmen Sie, welche Tastenkombination Sie verwenden möchten. Mit der Methode `Save` speichern Sie letztendlich den Shortcut und geben die Tastenkombination zur Information im Direktbereich aus.

4.6.4 Website-Zugang ganz fix

Wenn Sie im Internet schnell auf eine Seite zugreifen möchten, können Sie in Edge diese Webseite im Favoriten-Ordner speichern und schnell darauf zugreifen. Allerdings können Sie die URL der Internetseite auch als Symbol auf Ihrem Desktop ablegen. Setzen Sie dazu Listing 4.36 ein.

```

Sub InternetSeiteDirekt()
    Dim wshshShell As WshShell
    Dim strDesktop As String
    Dim wshshLink As WshURLShortcut

    Set wshshShell = CreateObject("WScript.Shell")
    strDesktop = wshshShell.SpecialFolders("Desktop")
    Set wshshLink = wshshShell.CreateShortcut(strDesktop _
        & "\Held-Office.url")
    wshshLink.TargetPath = "http://www.held-office.de"
    wshshLink.Save
    Debug.Print "Der schnelle Website-Zugang ist" & _
        "eingrichtet!"

End Sub

```

Listing 4.36 Ein Desktop-Symbol anlegen

Beachten Sie bei der Benennung des Symbols, dass der Name mit `.url` enden muss. Mit einem Doppelklick auf das neue Symbol auf Ihrem Desktop gelangen Sie direkt auf meine Website. Allerdings muss dazu die Internetverbindung aktiviert sein bzw. aktiviert werden.

4.6.5 Laufwerke mappen

Wenn Sie ein Netzwerk betreiben, können Sie einzelne Verzeichnisse durch Laufwerksbuchstaben im Explorer darstellen. Dieser Vorgang wird als *Mapping* bezeichnet. Damit können Sie die gewünschten Verzeichnisse im Explorer schneller erreichen. Gerade bei weit verzweigten Verzeichnisstrukturen sparen Sie so viel Zeit. Wenn Sie diese Arbeit manuell durchführen möchten, wählen Sie im Windows-Explorer aus dem Menü EXTRAS den Befehl NETZLAUFWERK VERBINDEN. Danach weisen Sie das Netzwerkverzeichnis einem noch freien Laufwerksbuchstaben zu und klicken abschließend auf die Schaltfläche FERTIG STELLEN. Dieser Vorgang benötigt Zeit, besonders wenn Sie gleich mehrere Verzeichnisse verbinden möchten. Diese zeitraubende Aufgabe können Sie leicht mit der folgenden Prozedur in Listing 4.37 automatisch erledigen lassen. Dies lohnt sich dann, wenn Sie dasselbe Mapping immer wieder auf neuen Rechnern vornehmen müssen.

```

Sub MappingsAnlegen()
    Dim wshnwNetzwerk As WshNetwork

    Set wshnwNetzwerk = CreateObject("WScript.Network")

```

```
'Mehrere Mappings automatisch erstellen
wshnwNetzwerk.MapNetworkDrive "W:", \\Hpserver\webdaten
wshnwNetzwerk.MapNetworkDrive "S:", \\Hpserver\webdaten\Seminare
wshnwNetzwerk.MapNetworkDrive "P:", \\Hpserver\webdaten\Programmierung

MsgBox "Die Mappings wurden durchgeführt!", vbInformation
End Sub
```

Listing 4.37 Laufwerke automatisch mappen

Um Netzlaufwerke zu verbinden, verwenden Sie das Objekt `WScript.Network` und seine Methode `MapNetworkDrive`. Die Methode verlangt als erstes Argument das Laufwerk, unter dem Sie in Zukunft Ihr Netzlaufwerksverzeichnis ansprechen möchten. Im zweiten Argument geben Sie das Netzverzeichnis an, das Sie zuweisen möchten.

Die Mappings können Sie aber auch wieder auflösen. Auch diese Aufgabe können Sie wieder manuell im Explorer durchführen oder automatisch durch die Prozedur aus Listing 4.38 erledigen lassen.

```
Sub MappingsAufheben()
    Dim wshnwNetzwerk As WshNetwork
    Set wshnwNetzwerk = CreateObject("WScript.Network")

    Debug.Print "Die gemappten Laufwerke werden" _
        & "wieder freigegeben!"

    'Mehrere Mappings automatisch aufheben
    wshnwNetzwerk.RemoveNetworkDrive "W:"
    wshnwNetzwerk.RemoveNetworkDrive "S:"
    wshnwNetzwerk.RemoveNetworkDrive "P:"

End Sub
```

Listing 4.38 Mappings wieder aufheben

Über die Methode `RemoveNetworkDrive` heben Sie die einzelnen Mappings wieder auf.

4.6.6 Gemappte Laufwerke anzeigen

In der nächsten Aufgabe in Listing 4.39 geben Sie zur Information alle gemappten Laufwerke im Direktfenster der Entwicklungsumgebung aus.

```
Sub MappingsAnzeigen()
    Dim wshshShell As WshShell
    Dim wshnwNetzwerk As WshNetwork
```

```

Dim wshcoLaufwerke As WshCollection
Dim intZ As Integer

Set wshnwNetzwerk = CreateObject("WScript.Network")
Set wshcoLaufwerke = wshnwNetzwerk.EnumNetworkDrives

For intZ = 0 To wshcoLaufwerke.Count - 1
    Debug.Print wshcoLaufwerke.Item(intZ) & vbCrLf
Next intZ

End Sub

```

Listing 4.39 Mappings ermitteln und ausgeben

Ermitteln Sie mit der Eigenschaft `Count`, die Sie auf das Objekt `EnumNetworkDrives` anwenden, die Anzahl der gemappten Laufwerke. Diese geben Sie im Direktbereich von Access aus.

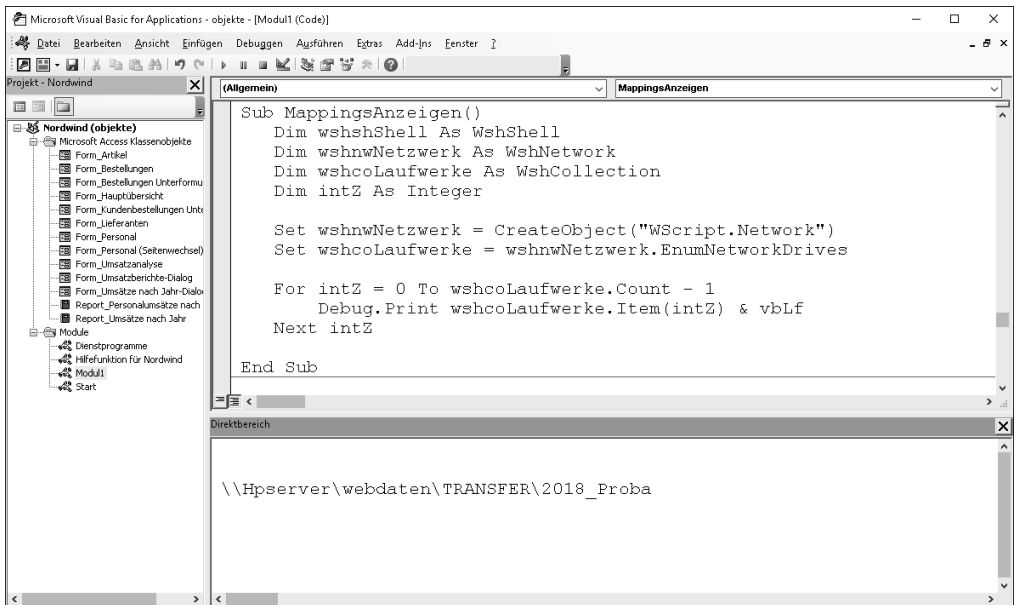


Abbildung 4.17 Alle Mappings anzeigen

4.6.7 Laufwerk auswerten

Den noch freien Speicherplatz Ihrer lokalen Festplatte ermitteln Sie über die Eigenschaft `FreeSpace`, indem Sie sie auf das `FileSystemObject` anwenden. Wie das genau aussieht, sehen Sie in Listing 4.40.

```
Sub FreienPlattenplatzAnzeigen()  
    Dim fs As FileSystemObject  
    Dim drv As Drive  
    Dim str As String  
  
    Set fs = CreateObject("Scripting.FileSystemObject")  
    Set drv = fs.GetDrive(fs.GetDriveName("C:\"))  
    str = "Laufwerk " & UCase("C:\") & " - "  
    str = str & drv.VolumeName & vbCrLf  
    str = str & "Freier Platz: " & _  
        FormatNumber(drv.FreeSpace / 1024, 0)  
    str = str & " Kbytes"  
    MsgBox str, vbInformation  
  
End Sub
```

Listing 4.40 Freien Plattenplatz ermitteln

Mit der Methode `GetDrive` bekommen Sie Zugriff auf das Laufwerk, das Sie mit der Methode `GetDriveName` übergeben. Über die Eigenschaft `VolumeName` lesen Sie den Datenträgernamen des Laufwerkes aus. Die Funktion `FormatNumber` sorgt dafür, dass Sie die so ermittelte Größe in einen formatierten Ausdruck übersetzen. Die Null bedeutet dabei, dass keine Nachkommastellen angezeigt werden sollen. Die Eigenschaft `FreeSpace` ermittelt den momentan noch freien Platz auf Ihrer Festplatte.

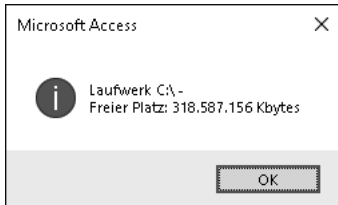


Abbildung 4.18 Freien Plattenplatz ermitteln und anzeigen

Möchten Sie wissen, welche Laufwerke auf Ihrem PC oder Netzwerk zur Verfügung stehen, dann führen Sie die Prozedur aus Listing 4.41 aus.

```
Sub LaufwerksbuchstabenErmitteln()  
    Dim fs As FileSystemObject  
    Dim drv As Drive  
    Dim drvs As Drives  
    Dim strMedium As String
```

```

Set fs = CreateObject("Scripting.FileSystemObject")
Set drvs = fs.Drives
For Each drv In drvs
    Debug.Print drv.DriveLetter
    Select Case drv.DriveType
        Case 0: strMedium = "Unbekannt"
        Case 1: strMedium = "Diskettenlaufwerk"
        Case 2: strMedium = "Festplatte"
        Case 3: strMedium = "Netzwerk"
        Case 4: strMedium = "CDROM"
        Case 5: strMedium = "Wechselplatte"
    End Select

    Debug.Print strMedium
    Debug.Print drv.ShareName & Chr(13)
Next drv

End Sub

```

Listing 4.41 Laufwerksinformationen abfragen

Erzeugen Sie im ersten Schritt ein `FileSystemObject` über die Methode `CreateObject`. Danach fragen Sie über das Auflistungsobjekt `Drives` alle verfügbaren Laufwerke in einer Schleife ab. In der Schleife ermitteln Sie über die Eigenschaft `DriveLetter` den Laufwerksbuchstaben. Über die Eigenschaft `DriveType` finden Sie den Laufwerkstyp heraus. Dabei meldet diese Eigenschaft bestimmte numerische Werte zurück, die Rückschluss auf den Laufwerkstyp geben. Mit der Eigenschaft `ShareName` geben Sie den Namen der Netzwerkressource eines bestimmten Laufwerkes aus.

```

Direktbereich
C
Festplatte

D
CDROM

E
Festplatte

Z
Netzwerk

```

Abbildung 4.19 Laufwerke ermitteln und auswerten

4.6.8 Aktuelle Datenbank sichern

Da Sie standardmäßig immer nur eine Datenbank pro Sitzung geöffnet haben können, ist es recht aufwendig, eine Sicherungskopie Ihrer Datenbank in einem anderen Verzeichnis zu speichern. Der Weg über das Menü DATEI und den Befehl SPEICHERN UNTER ist ebenso unpraktisch wie der Umweg über den Windows-Explorer. Besser ist es, Ihre Datenbank im laufenden Betrieb zu sichern. Dazu können Sie die Methode CopyFile aus Listing 4.42 einsetzen.

```
Sub DatenbankSichern()  
    Dim fs As Scripting.FileSystemObject  
  
    Set fs = New Scripting.FileSystemObject  
    fs.CopyFile CurrentDb.Name, "C:\Users\" & Environ("username") & _  
        "\Documents\Sicherung.accdb", True  
    Set fs = Nothing  
  
End Sub
```

Listing 4.42 Datenbank sichern

Mit der Methode CopyFile haben Sie die Möglichkeit, den aktuellen Stand Ihrer Datenbank zu kopieren und unter einem anderen Namen in einem anderen Verzeichnis zu sichern.

4.6.9 Datenbank-Datumsangaben auswerten

Bei der nächsten Aufgabe soll mithilfe des FileSystemObject eine Datenbank ausgewertet werden. Uns interessieren folgende Informationen:

- ▶ das Erstellungsdatum der Datenbank
- ▶ das Datum der letzten Änderung
- ▶ das Datum des letzten Zugriffs

Diese Aufgabe können Sie beispielsweise lösen wie in Listing 4.43, indem Sie eine Funktion schreiben, der Sie den Namen der Datenbank übergeben, die Sie auswerten möchten. Über einen Index zwischen 1 und 3 geben Sie dann genau an, welches Datum Sie ermitteln möchten.

```
Function DB_Daten(strFileName As String, _  
    intDatNr As Integer) As String  
    Dim fs As Scripting.FileSystemObject  
  
    Set fs = New Scripting.FileSystemObject  
    With fs.GetFile(strFileName)
```

```

If intDatNr = 1 Then DB_Daten = .DateCreated
If intDatNr = 2 Then DB_Daten = .DateLastModified
If intDatNr = 3 Then DB_Daten = .DateLastAccessed
End With

```

End Function

Listing 4.43 Datumsauswertungen der Datenbank vornehmen

- ▶ Über die Eigenschaft `DateCreated` ermitteln Sie das Anlegedatum einer Datenbank oder auch eines Ordners.
- ▶ Die Eigenschaft `DateLastModified` liefert Ihnen den genauen Zeitpunkt der letzten Änderung einer Datenbank.
- ▶ Über die Eigenschaft `DateLastAccessed` ermitteln Sie das Datum des letzten Zugriffs auf eine Datenbank oder einen Ordner.

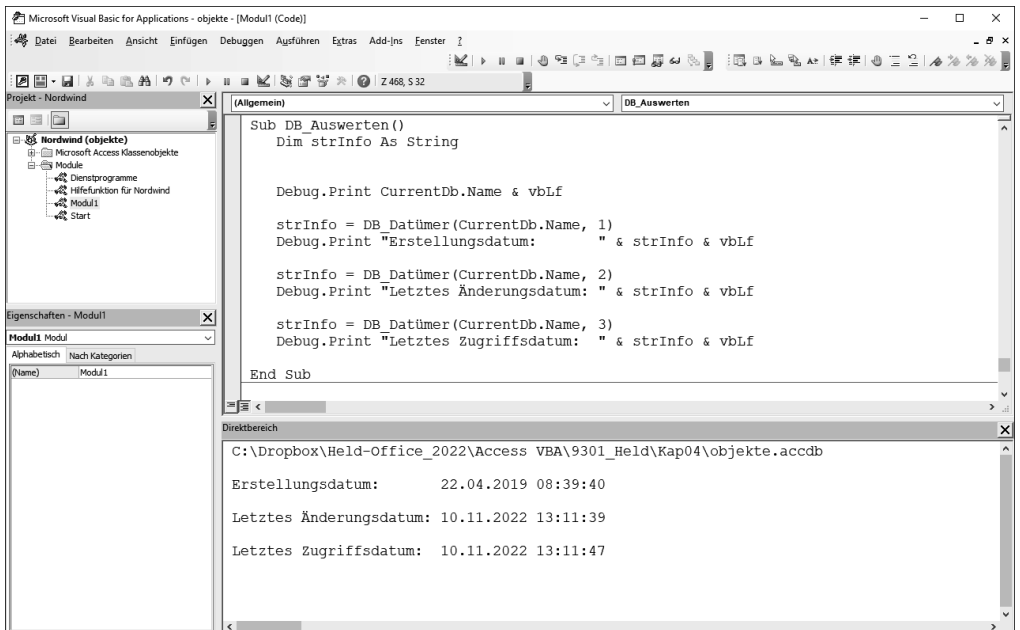


Abbildung 4.20 Alle Informationen zur aktuellen Datenbank

Schreiben Sie nun noch die aufrufende Prozedur wie in Listing 4.44, und übergeben Sie der Funktion nacheinander alle gewünschten Angaben.

```

Sub DB_Auswerten()
    Dim strInfo As String

```



```
Debug.Print CurrentDb.Name & vbCrLf

strInfo = DB_Datümer(CurrentDb.Name, 1)
Debug.Print "Erstellungsdatum:      " & strInfo & vbCrLf

strInfo = DB_Datümer(CurrentDb.Name, 2)
Debug.Print "Letztes Änderungsdatum: " & strInfo & vbCrLf

strInfo = DB_Datümer(CurrentDb.Name, 3)
Debug.Print "Letztes Zugriffsdatum:  " & strInfo & vbCrLf

End Sub
```

Listing 4.44 Datenbanknamen und gewünschten Datumsindex übergeben

4.6.10 Verzeichnisstruktur auslesen

Im Beispiel aus Listing 4.45 wird eine Verzeichnisstruktur ausgelesen.

```
Function OrdnerDurchsuchen(strV As String) As Boolean
    Dim fs As Object
    Dim objOrdner As Object
    Dim objSub As Object
    Dim intZ As Integer

    intZ = 0
    Set fs = CreateObject("Scripting.FileSystemObject")
    Set objOrdner = fs.GetFolder(strV)

    For Each objSub In objOrdner.SubFolders
        Debug.Print "Ordner " & intZ & ": " & objSub.Name
        intZ = intZ + 1
    Next objSub

    Set objOrdner = Nothing
    Set fs = Nothing
End Function

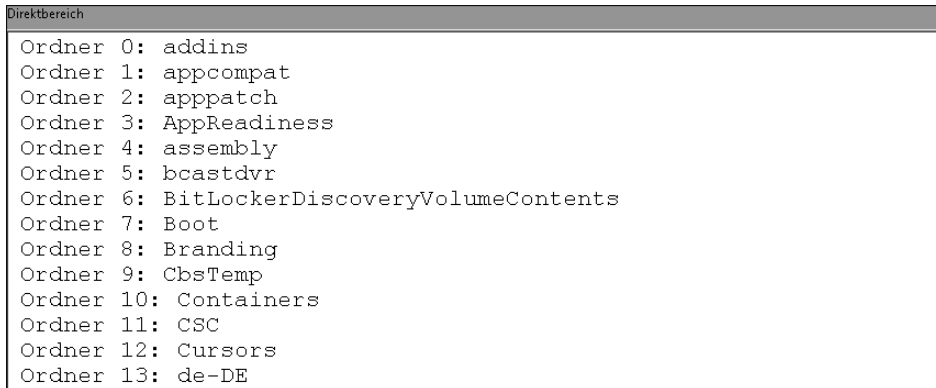
Sub VerzeichnisStrukturAuslesen()
    Dim bool As Boolean
```

```
bool = OrdnerDurchsuchen("C:\Windows")
```

End Sub

Listing 4.45 Alle Unterordner des Windows-Ordners werden ausgelesen.

Erstellen Sie zunächst ein `FileSystemObject`, indem Sie die Methode `CreateObject` anwenden. Danach greifen Sie über die Methode `GetFolder` auf das gewünschte Verzeichnis zu. In einer `For Each . . . Next`-Schleife arbeiten Sie alle Unterverzeichnisse ab. Diese Unterverzeichnisse liegen in der Auflistung `SubFolders` automatisch vor. Den Namen des jeweiligen Ordners fragen Sie über die Eigenschaft `Name` ab.



```
Direktbereich
Ordner 0: addins
Ordner 1: appcompat
Ordner 2: apppatch
Ordner 3: AppReadiness
Ordner 4: assembly
Ordner 5: bcastdvr
Ordner 6: BitLockerDiscoveryVolumeContents
Ordner 7: Boot
Ordner 8: Branding
Ordner 9: CbsTemp
Ordner 10: Containers
Ordner 11: CSC
Ordner 12: Cursors
Ordner 13: de-DE
```

Abbildung 4.21 Alle Unterordner des Windows-Ordners werden aufgelistet.

Kapitel 10

Access im Zusammenspiel mit Office

Das Datenbankprogramm von Microsoft ist keine Anwendung, die auf sich allein gestellt ist. Sie können von Access aus auf andere Anwendungen zugreifen und Daten austauschen.

Oft werden in der Praxis Access-Daten in Textdateien geschrieben oder transferierte Textdateien von Host-Systemen in Access eingelesen. Auch der Datenzugriff innerhalb des Office-Pakets ist geregelt. So können Sie beispielsweise Daten zwischen den Office-Komponenten Access, Excel, PowerPoint und Word austauschen.

In diesem Kapitel werde ich folgende Fragen beantworten:

- ▶ Welche Möglichkeiten habe ich, Daten aus Access in einer Textdatei zu speichern?
- ▶ Wie lese ich Textdateien in eine Access-Tabelle ein?
- ▶ Wie übertrage ich Access-Tabellen nach Word?
- ▶ Wie füge ich Access-Daten punktgenau in Word-Dokumente ein?
- ▶ Wie transferiere ich meine Adressdatenbank in das Adressbuch von Outlook?
- ▶ Wie übertrage ich das Adressbuch von Outlook in eine Access-Tabelle?
- ▶ Wie transferiere ich Termine aus einer Access-Tabelle in den Terminkalender von Outlook?
- ▶ Wie übernehme ich eine Aufgabentabelle in die Aufgabenliste von Outlook?
- ▶ Wie sichere ich E-Mails in einer Access-Datenbank?
- ▶ Wie sende ich eine Sammel-E-Mail an alle Kontakte aus einer Access-Tabelle?
- ▶ Wie exportiere ich Access-Tabellen nach Excel?
- ▶ Wie importiere ich Excel-Tabellen in Access?
- ▶ Wie kann ich eine Access-Datenbankabfrage von Excel aus ausführen?
- ▶ Wie steuere ich Access von Excel aus?
- ▶ Wie implementiere ich eine Benutzerverwaltung für Access?

Die in diesem Kapitel vorgestellten Lösungen finden Sie in den Materialien zum Buch im Ordner *Kap10* unter dem Namen *Nordwind.accdb* und in den Dateien *Wordformular.docx*, *ExcelFragtAccess.xlsm*, *DB_mit_Benutzerverwaltung.accdb* und *Arbeitszeitbericht.xlsm*.

10.1 Textdateien im Zugriff von Access

In Access haben Sie standardmäßig die Möglichkeit, Tabellen als Textdateien zu speichern. Bei Access 2007 und 2013 wählen Sie auf dem Ribbon EXTERNE DATEN in der Gruppe EXPORTIEREN das Element IN EINE TEXTDATEI EXPORTIEREN. Ebenso können Sie Textdateien in Access einlesen. Bei Access 2007, 2013, 2016 und 2019/2021/365 wählen Sie auf dem Ribbon EXTERNE DATEN in der Gruppe IMPORTIEREN das Element TEXTDATEI IMPORTIEREN.

10.1.1 Textdateien speichern

Das Speichern einer Textdatei können Sie selbstverständlich auch über VBA-Code automatisieren. So speichern Sie im folgenden Beispiel die Tabelle *Artikel* aus der Datenbank *Nordwind.accdb* in eine Textdatei. Als Trennzeichen verwenden Sie dabei das Semikolon. Aus diesem Grund geben Sie der Textdatei auch die Endung *.csv*. CSV steht für *Comma-Separated Values*. *.csv* ist eine typische Dateinamenerweiterung für Textdateien, die durch Sonderzeichen getrennte Werte enthalten. Sie könnten der Datei jedoch genauso gut die Endung *.txt* geben.

| Lieferant | ID | Produktcode | Artikelname | Beschreibung | Standardkosten | Listenpreise | Mindestbestände | Ziele für Bestände |
|--------------------------|----|-------------|--|--------------|----------------|--------------|-----------------|--------------------|
| Lieferant J | 1 | NWTB-1 | Northwind Traders Chai | | 13,50 € | 18,00 € | 10 | |
| Lieferant J | 3 | NWTCO-3 | Northwind Traders Syrup | | 7,50 € | 10,00 € | 25 | |
| Lieferant J | 4 | NWTCO-4 | Northwind Traders Cajun Seasoning | | 16,50 € | 22,00 € | 10 | |
| Lieferant J | 5 | NWTO-5 | Northwind Traders Olive Oil | | 16,01 € | 21,35 € | 10 | |
| Lieferant B; Lieferant F | 6 | NWTPJ-6 | Northwind Traders Boysenberry Spread | | 18,75 € | 25,00 € | 25 | |
| Lieferant B | 7 | NWTFN-7 | Northwind Traders Dried Pears | | 22,50 € | 30,00 € | 10 | |
| Lieferant H | 8 | NWTS-8 | Northwind Traders Curry Sauce | | 39,00 € | 40,00 € | 10 | |
| Lieferant B; Lieferant F | 14 | NWTFN-14 | Northwind Traders Walnuts | | 17,44 € | 23,25 € | 10 | |
| Lieferant F | 17 | NWTFV-17 | Northwind Traders Fruit Cocktail | | 29,25 € | 39,00 € | 10 | |
| Lieferant A | 19 | NWBTGM-19 | Northwind Traders Chocolate Biscuits Mix | | 6,90 € | 9,20 € | 5 | |
| Lieferant B; Lieferant F | 20 | NWTPJ-6 | Northwind Traders Marmalade | | 60,75 € | 81,00 € | 10 | |
| Lieferant A | 21 | NWBTGM-21 | Northwind Traders Scones | | 7,50 € | 10,00 € | 5 | |
| Lieferant D | 34 | NWBTB-34 | Northwind Traders Beer | | 10,50 € | 14,00 € | 15 | |
| Lieferant G | 40 | NWTCM-40 | Northwind Traders Crab Meat | | 13,80 € | 18,40 € | 30 | |
| Lieferant F | 41 | NWTCO-41 | Northwind Traders Clam Chowder | | 7,24 € | 9,65 € | 10 | |
| Lieferant C; Lieferant D | 43 | NWTB-43 | Northwind Traders Coffee | | 34,50 € | 46,00 € | 25 | |
| Lieferant J | 48 | NWTCM-48 | Northwind Traders Chocolate | | 9,56 € | 12,75 € | 25 | |
| Lieferant B | 51 | NWTFN-51 | Northwind Traders Dried Apples | | 39,75 € | 53,00 € | 10 | |
| Lieferant A | 52 | NWTCG-52 | Northwind Traders Long Grain Rice | | 5,25 € | 7,00 € | 25 | |
| Lieferant A | 56 | NWTP-56 | Northwind Traders Gnocchi | | 28,50 € | 38,00 € | 30 | |
| Lieferant A | 57 | NWTP-57 | Northwind Traders Ravioli | | 14,63 € | 19,50 € | 20 | |
| Lieferant H | 65 | NWTS-65 | Northwind Traders Hot Pepper Sauce | | 15,79 € | 21,05 € | 10 | |
| Lieferant H | 66 | NWTS-66 | Northwind Traders Tomato Sauce | | 12,75 € | 17,00 € | 20 | |
| Lieferant E | 72 | NWTD-72 | Northwind Traders Mozzarella | | 26,10 € | 34,80 € | 10 | |
| Lieferant J; Lieferant F | 74 | NWTFN-74 | Northwind Traders Almonds | | 7,50 € | 10,00 € | 5 | |
| Lieferant B | 77 | NWTCO-77 | Northwind Traders Mustard | | 9,75 € | 13,00 € | 15 | |
| Lieferant B | 80 | NWTFN-80 | Northwind Traders Dried Plums | | 3,00 € | 3,50 € | 50 | |

Abbildung 10.1 Die Ausgangstabelle vor dem Datentransfer

Beim Datentransfer sollen nur die Datenfelder »Artikelname«, »Liefereinheit«, »Mindestbestand«, »Lagerbestand« und »BestellteEinheiten« in die Textdatei übertragen werden. Sie brauchen bei der Prozedur aus Listing 10.1 die Tabelle *Artikel* nicht zu öffnen. Das Öffnen, Verarbeiten und Transferieren der Daten in die Textdatei erfolgt automatisch.

Kontrollieren Sie vor dem Starten der Prozedur aus Listing 10.1, ob in der Entwicklungsumgebung unter EXTRAS • VERWEISE der Verweis auf die ADO-Bibliothek gesetzt ist.

```

Sub TabelleAlsTextdateiSpeichern()
    Dim rst As New ADODB.Recordset
    Dim str As String
    Dim lngZ As Long

    On Error GoTo Fehler
    rst.Open "Artikel", CurrentProject.Connection
    Open Application.CurrentProject.Path & "\Artikel.csv" For Output As #1
    lngZ = 0
    Do Until rst.EOF
        str = str & rst!Artikelname & ";" & _
            rst!Liefereinheit & ";" & rst!Listenpreis & ";" & _
            rst!Mindestbestand & ";" & rst![Ziel für Bestand]
        rst.MoveNext
        Print #1, str
        lngZ = lngZ + 1
        str = ""
    Loop
    Close #1
    MsgBox "Transfer beendet! Es wurden " & lngZ & " Sätze übertragen!"
    rst.Close
    Set rst = Nothing
    Exit Sub

Fehler:
    MsgBox Err.description

End Sub

```

Listing 10.1 Teile einer Tabelle in eine Textdatei schreiben

Definieren Sie im ersten Schritt ein ADO-Objekt vom Typ `Recordset`. Damit haben Sie Zugriff auf alle Tabellen in Access. Definieren Sie danach eine `String`-Variable, um später die einzelnen Datenfelder in dieser Variablen zwischenspeichern. In der folgenden Variablen `lngZ` sollen die übertragenen Datensätze gezählt und später ausgegeben werden. Öffnen Sie daraufhin mit der Methode `Open` die Tabelle *Artikel* in der aktuell geöffneten Datenbank. Mit der Anweisung `CurrentProject.Connection` fangen Sie den Namen der aktuellen Datenbank ein. Öffnen Sie dann über die Methode `Open` die Textdatei, in die die Daten transferiert werden sollen. Dabei muss diese Textdatei

keineswegs schon angelegt sein. Der Befehl `Open` erstellt sie neu, sofern sie noch nicht existiert. Sehen wir uns diese Methode einmal genauer an:

```
Open Pfadname For Modus [Access-Zugriff] [Sperre] As [#]Dateinummer  
[Len=Satzlänge]
```

Im Argument `Pfadname` geben Sie das Laufwerk sowie den Dateinamen der Textdatei an, die Sie öffnen möchten.

Das Argument `Modus` legt den Zugriffsmodus für die Datei fest. Möglich sind die Modi `Append`, `Binary`, `Input`, `Output` oder `Random`. Wenn kein Modus festgelegt ist, wird die Datei im Zugriffsmodus `Random` geöffnet. In unserem Beispiel in Listing 10.1 wurde die Textdatei im Zugriffsmodus `Output` geöffnet, was darauf schließen lässt, dass Sie Sätze in diese Textdatei transferieren möchten. Würden Sie die Textdatei mit dem Zugriffsmodus `Append` öffnen, würde die Prozedur bei mehrmaligem Start hintereinander die Daten immer unten an die Textdatei hängen, was zur Folge hätte, dass Sie die Datensätze mehrfach vorliegen hätten.

Das nächste optionale Argument, `Access-Zugriff`, legt die Operation fest, die mit der geöffneten Datei ausgeführt werden soll. Dabei setzen Sie entweder `Read`, wenn Sie die Textdatei nur lesen wollen, oder `Write`, wenn Sie etwas in die Textdatei schreiben möchten. Wollen Sie beide Aktionen durchführen, dann setzen Sie die Konstante `Read Write` ein.

Beim optionalen Argument `Sperre` bestimmen Sie, welche Operationen andere Anwendungen mit der geöffneten Datei durchführen dürfen oder nicht. Dabei können Sie bestimmte Zugriffe wie das Lesen und Schreiben zulassen oder verwehren. Zur Auswahl stehen `Shared`, `Lock Read`, `Lock Write` und `Lock Read Write`.

Mithilfe des Arguments `Dateinummer` vergeben Sie eine gültige Dateinummer im Bereich von 1 bis 511. Dabei haben Sie die Möglichkeit, bei den Modi `Binary`, `Input` und `Random` eine Datei mit einer anderen Dateinummer zu öffnen, ohne sie zuvor schließen zu müssen. In den Modi `Append` und `Output` müssen Sie eine Datei erst schließen, bevor diese mit einer anderen Dateinummer geöffnet werden kann. Wenn Sie dennoch versuchen, in eine bereits so geöffnete Textdatei zu schreiben, werden Sie daran gehindert und erhalten eine Fehlermeldung.



Abbildung 10.2 Die Datei ist bereits geöffnet.

Im letzten optionalen Argument, `Satzlänge`, können Sie für die Textdatei noch eine Satzlänge vorgeben. Dabei handelt es sich um eine Zahl, die kleiner oder gleich 32.767

(Bytes) ist. Bei Dateien mit wahlfreiem Zugriff ist dies die Datensatzlänge, bei sequenziellen Dateien die Anzahl der gepufferten Zeichen.

Haben Sie nun sowohl die Tabelle als auch die Textdatei geöffnet, setzen Sie die Zählvariable `lng` auf den Wert 0. Danach setzen Sie eine Schleife auf, die so lange durchlaufen wird, bis der letzte Datensatz in der Tabelle erreicht ist. Diese automatische Überprüfung nimmt die Eigenschaft `EOF` (»End of File«) vor. Sie dürfen allerdings nicht vergessen, den Datenzeiger mit der Methode `MoveNext` jeweils einen Satz weiter zu setzen, sobald Sie einen Satz übertragen haben. Sie erzeugen ansonsten eine Endlosschleife. Ebenfalls innerhalb der Schleife erstellen Sie eine Variable `str`, die die einzelnen Datenfelder aus der Tabelle *Artikel* aufnimmt. Vergessen Sie dabei nicht, nach jeder Feldinformation das Trennzeichen (;) einzusetzen.

Mit der Anweisung `Print` drucken Sie den auf diese Weise geschriebenen aktuellen Inhalt der Variablen `str` direkt in die geöffnete Textdatei. Dabei müssen Sie dieselbe Dateinummer verwenden, die Sie schon beim Öffnen der Textdatei eingesetzt haben. Nur so wird gewährleistet, dass der »Ausdruck« auch in die gewünschte Textdatei geschrieben wird.

Sorgen Sie dafür, dass die Zählvariable `lngZ` bei jedem Schleifendurchlauf um den Wert 1 erhöht wird. Vergessen Sie auch nicht, am Ende eines jeden Schleifendurchlaufs die Variable `str` wieder zu initialisieren.

Wird das Ende der Tabelle *Artikel* erreicht, dann ist das Schleifenabbruchskriterium erfüllt, und die Schleife wird verlassen. Schließen Sie die noch geöffnete Textdatei mit der Methode `Close`. Geben Sie danach eine Bildschirmmeldung aus, die die Anzahl der übertragenen Sätze enthält. Schließen Sie dann ebenfalls über die Methode `Close` die Tabelle *Artikel*, und heben Sie den Objektverweis mit der Anweisung `Set rst = Nothing` wieder auf, um den reservierten Speicher wieder freizugeben.

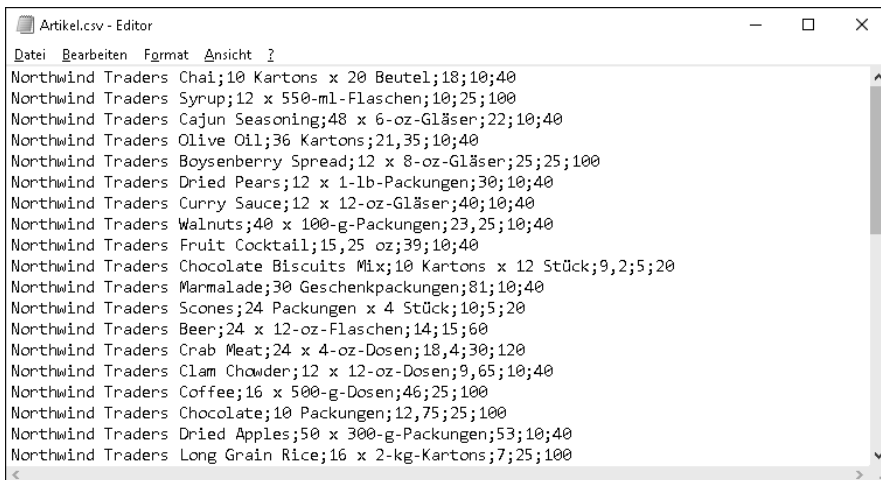


Abbildung 10.3 Die Textdatei mit dem Semikolon als Trennzeichen

10.1.2 Textdateien exportieren

Eine weitere Möglichkeit, Access-Tabellen in Textdateien zu speichern, bietet das DoCmd-Objekt. So speichert die Prozedur aus Listing 10.2 die Tabelle *Artikel* im Verzeichnis *C:\Eigene Dateien* unter dem Namen *Artikel.txt*.

```
Sub TabelleTransferieren()
```

```
DoCmd.OutputTo acOutputTable, "Artikel", _  
    acFormatTXT, Application.CurrentProject.Path & "\Artikel.txt", True
```

```
End Sub
```

Listing 10.2 Komplette Tabelle in eine Textdatei schreiben

Mit der Methode `OutputTo` geben Sie die Daten in einem bestimmten Microsoft-Access-Datenbankobjekt (einem Datenblatt, einem Formular, einem Bericht, einem Modul oder einer Datenzugriffsseite) in verschiedenen Formaten aus. Dabei lautet die Syntax dieser Methode wie folgt:

```
OutputTo(ObjectType, ObjectName, OutputFormat, OutputFile, AutoStart,  
TemplateFile)
```

Über das Argument `ObjectType` legen Sie die Art des Access-Objekts fest, dessen Daten Sie exportieren möchten.

Dabei haben Sie folgende Möglichkeiten:

- ▶ `acOutputForm`: Export der Daten eines Formulars
- ▶ `acOutputFunction`: Export einer Funktion zur Sicherung
- ▶ `acOutputModule`: Export eines kompletten Moduls inklusive aller Funktionen und Prozeduren
- ▶ `acOutputQuery`: Export der Ergebnisse einer Abfrage
- ▶ `acOutputReport`: Export eines Berichts
- ▶ `acOutputServerView`: Export einer Serveransicht
- ▶ `acOutputStoredProcedure`: Export einer gespeicherten Prozedur
- ▶ `acOutputTable`: Export einer Tabelle

Beim Argument `ObjectName` geben Sie den Namen des Objekts an, das Sie exportieren möchten. In der Prozedur aus Listing 10.2 ist dies der Name des Objekts `acOutputTable`, also *Artikel*.

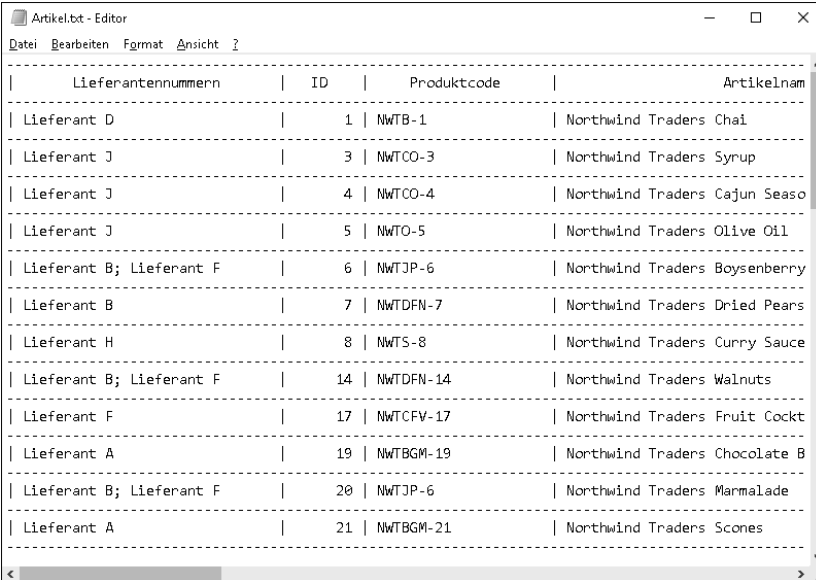
Mit dem Argument `OutputFormat` legen Sie fest, in welchem Datenformat Sie die Daten transferieren möchten. Die bekanntesten Formate heißen dabei wie folgt:

- ▶ Das Argument `acFormatHTML` konvertiert die Daten in das HTML-Format.
- ▶ `acFormatRTF` konvertiert die Daten in das Rich-Text-Format. Dieses Format kann beispielsweise problemlos in Microsoft Word eingelesen werden.
- ▶ Mit dem Format `acFormatTXT` ist das Textformat gemeint.
- ▶ Mit `acFormatXLS` konvertieren Sie die Daten in das Microsoft-Excel-Format.

Beim Argument `OutputFile` geben Sie den Pfad sowie den Dateinamen der Datei an, in die Sie die Daten transferieren möchten. Dabei muss die Datei noch nicht vorhanden sein; Access legt sie bei Bedarf selbst an.

Mit dem Argument `AutoStart` haben Sie die Möglichkeit, die so erstellte Exportdatei gleich zu öffnen. Verwenden Sie den Wert `True`, um die entsprechende auf Windows basierende Anwendung sofort zu starten. Setzen Sie das Argument auf den Wert `False` oder lassen Sie es weg, wenn die Exportdatei nicht geöffnet werden soll.

Das Argument `TemplateFile` ist dann von Interesse, wenn Sie eine Vorlage verwenden möchten, beispielsweise für die HTML-Datei. In diesem Fall ist der komplette Pfad dieser Vorlagendatei anzugeben.



| Lieferantennummern | ID | Produktcode | Artikelname |
|--------------------------|----|-------------|--------------------------------------|
| Lieferant D | 1 | NWTB-1 | Northwind Traders Chai |
| Lieferant J | 3 | NWTCO-3 | Northwind Traders Syrup |
| Lieferant J | 4 | NWTCO-4 | Northwind Traders Cajun Seasoning |
| Lieferant J | 5 | NWTO-5 | Northwind Traders Olive Oil |
| Lieferant B; Lieferant F | 6 | NWTJP-6 | Northwind Traders Boysenberry |
| Lieferant B | 7 | NWTDFN-7 | Northwind Traders Dried Pears |
| Lieferant H | 8 | NWTS-8 | Northwind Traders Curry Sauce |
| Lieferant B; Lieferant F | 14 | NWTDFN-14 | Northwind Traders Walnuts |
| Lieferant F | 17 | NWTCFV-17 | Northwind Traders Fruit Cocktail |
| Lieferant A | 19 | NWTBGM-19 | Northwind Traders Chocolate Biscuits |
| Lieferant B; Lieferant F | 20 | NWTJP-6 | Northwind Traders Marmalade |
| Lieferant A | 21 | NWTBGM-21 | Northwind Traders Scones |

Abbildung 10.4 Die exportierte Textdatei wurde bereits optisch aufbereitet.

10.1.3 Code sichern

Die Methode `OutputTo` bietet Ihnen die Möglichkeit, Ihre Programmierung zu sichern. Die Prozedur aus Listing 10.3 sichert das Modul `MODUL1` in der Textdatei `Code.txt` im Verzeichnis `C:\Eigene Dateien`.

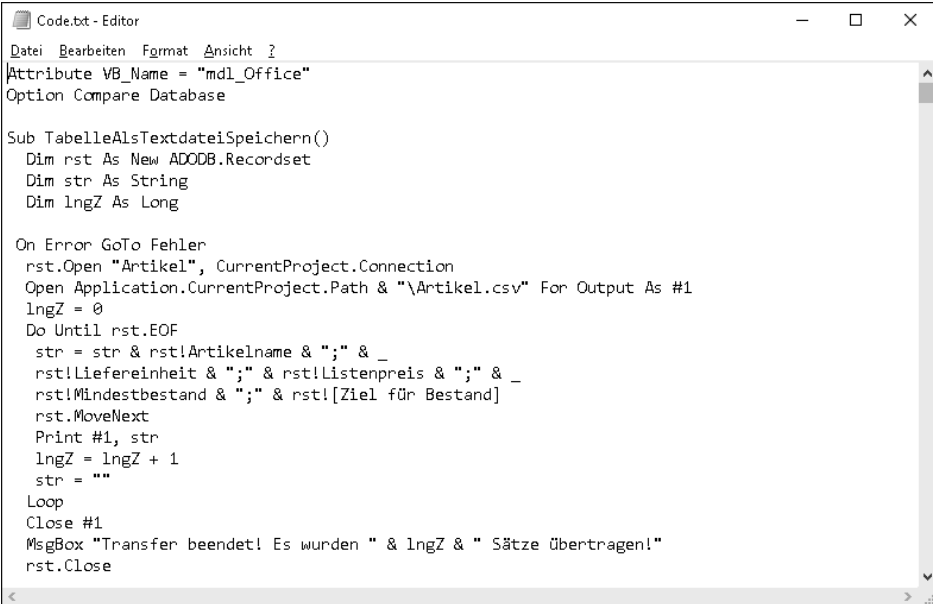
```
Sub ModulTransferieren()
```

```
DoCmd.OutputTo acOutputModule, "mdl_Office", _
    acFormatTXT, Application.CurrentProject.Path & "\Code.txt", True
```

```
End Sub
```

Listing 10.3 Ein Modul sichern

Indem Sie die Konstante `acOutputModule` verwenden, erkennt Access, dass es aus der Entwicklungsumgebung das `MODUL1` exportieren soll. Mit der Konstanten `acFormatTXT` legen Sie fest, dass dieser Transfer im Textformat stattfinden soll.



```
Code.txt - Editor
Datei Bearbeiten Format Ansicht ?
Attribute VB_Name = "mdl_Office"
Option Compare Database

Sub TabelleAlsTextdateiSpeichern()
    Dim rst As New ADODB.Recordset
    Dim str As String
    Dim lngZ As Long

    On Error GoTo Fehler
    rst.Open "Artikel", CurrentProject.Connection
    Open Application.CurrentProject.Path & "\Artikel.csv" For Output As #1
    lngZ = 0
    Do Until rst.EOF
        str = str & rst!Artikelname & ";" & _
            rst!Liefereinheit & ";" & rst!Listenpreis & ";" & _
            rst!Mindestbestand & ";" & rst![Ziel für Bestand]
        rst.MoveNext
        Print #1, str
        lngZ = lngZ + 1
        str = ""
    Loop
    Close #1
    MsgBox "Transfer beendet! Es wurden " & lngZ & " Sätze übertragen!"
    rst.Close
Fehler:

```

Abbildung 10.5 Module sichern

Haben Sie mehrere Module in Ihrer Datenbank untergebracht und möchten Sie sie alle sichern, dann setzen Sie die Prozedur aus Listing 10.4 ein.

```
Sub AlleModuleSpeichern()
```

```
    Dim obj As AccessObject
    Dim dbs As Object
    Dim intZ As Integer
```

```
    Set dbs = Application.CurrentProject
    For Each obj In dbs.AllModules
```

```

intZ = intZ + 1
DoCmd.OutputTo acOutputModule, obj.Name, _
    acFormatTXT, Application.CurrentProject.Path & _
    "\Codes\Code" & intZ & ".txt"
Next obj

```

End Sub

Listing 10.4 Alle Module einer Datenbank sichern

Im Auflistungsobjekt `AllModules` sind alle Module der Datenbank verzeichnet. Sie können damit ganz elegant ein Modul nach dem anderen mithilfe einer Schleife ansprechen. Innerhalb der Schleife wenden Sie die Methode `OutputTo` an, um die einzelnen Module zu sichern. Dabei geben Sie den einzelnen Textdateien fortlaufende Namen, die sich aus dem Text `Code` und der Zählvariablen `intZ` zusammensetzen.

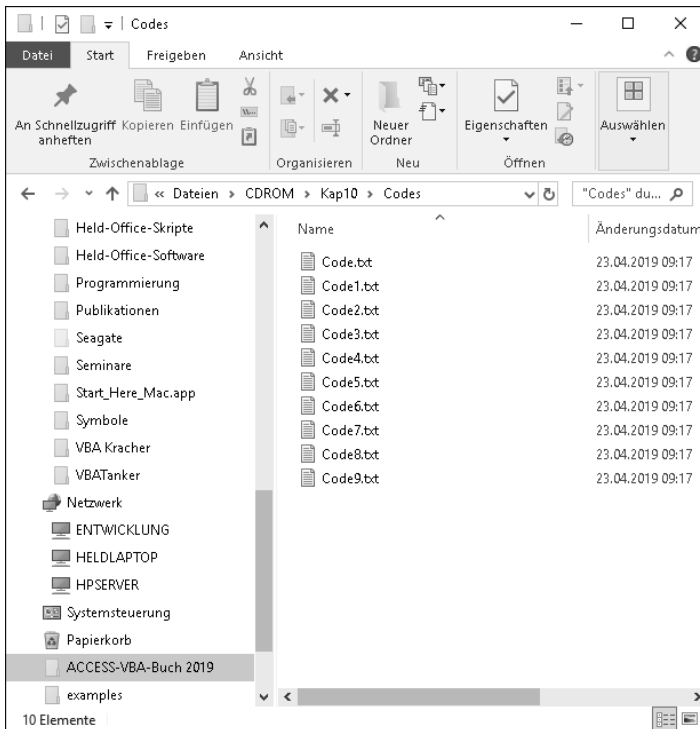


Abbildung 10.6 Alle Module wurden gesichert.

10.1.4 Textdateien einlesen

Beim Einlesen von Textdateien in Access können Sie die Methode `TransferText` einsetzen. Dazu benötigen Sie zuerst eine Importspezifikation. Diese enthält Informa-

tionen (beispielsweise über das Dateiformat, die Reihenfolge der Datumswerte oder die Zahlenformate), die Microsoft Access zum Importieren einer Textdatei mit festgelegtem Format oder mit Trennzeichen verwendet. Die Importspezifikation legen Sie einmalig an und benutzen sie immer wieder.

Sie können eine Importspezifikation mit dem Textimport-Assistenten erstellen, indem Sie die nächsten Arbeitsschritte befolgen:

1. Wählen Sie bei geöffneter Datenbank auf dem Ribbon EXTERNE DATEN aus der Gruppe IMPORTIEREN das Element TEXTDATEI IMPORTIEREN, sofern Sie mit Access 2007 oder höher arbeiten. Die folgenden Schritte beschreiben die Vorgehensweise bei Access 2019/2021/365. Bei früheren Versionen von Access ist die Vorgehensweise jedoch recht ähnlich.
2. Im Dialogfeld EXTERNE DATEN • NEUE DATENQUELLE • AUS DATEI • TEXTDATEI wählen Sie die Textdatei aus, die Sie in eine Access-Tabelle importieren möchten, und entscheiden, ob Sie die Datei importieren, die Datensätze anfügen oder eine Verknüpfung zu der Tabelle erstellen möchten (siehe Abbildung 10.7). Wählen Sie die erste Option, um die Daten zu importieren.



Abbildung 10.7 Textdatei auswählen

3. Klicken Sie auf die Schaltfläche OK.

4. Der TEXTIMPORT-ASSISTENT wird gestartet. Im angezeigten Dialog können Sie auswählen, ob die Daten der zu importierenden Datei durch Trennzeichen oder in Spalten mit fester Breite, also nur durch Leerzeichen getrennt sind (siehe Abbildung 10.8). Da bei der Textdatei *Artikel.csv* keine feste Feldbreite vorliegt und Sie das Semikolon als Trennzeichen vorfinden, aktivieren Sie in diesem Beispiel die erste Option.

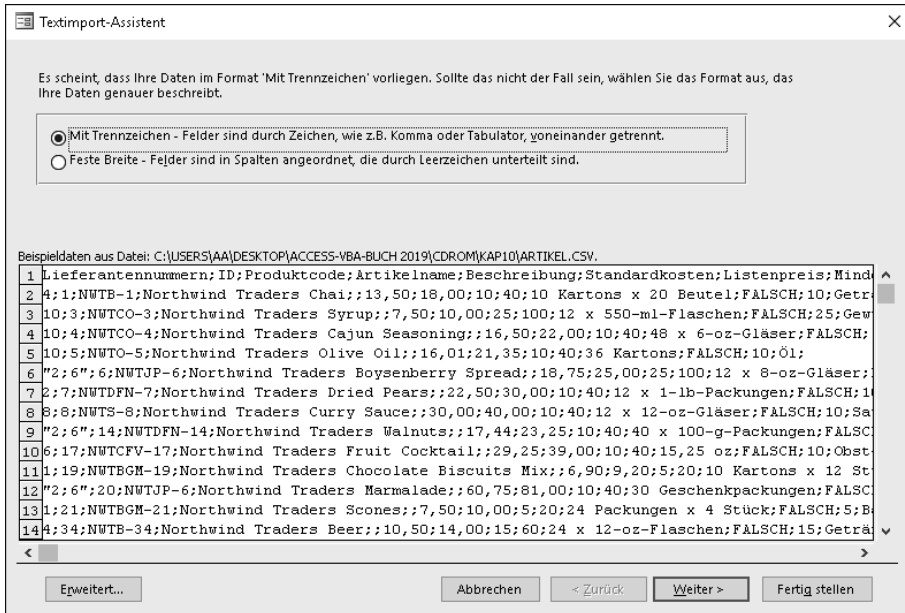


Abbildung 10.8 Der Textimport-Assistent meldet sich.

5. Klicken Sie danach auf die Schaltfläche ERWEITERT... (bitte nicht mit WEITER verwechseln!).
6. Der Dialog ARTIKEL IMPORTSPEZIFIKATION wird angezeigt. Im Gruppenfeld FELD-INFORMATION hat Access anhand der Textdatei *Artikel.csv* bereits die Felder vom Datentyp für Sie automatisch vordefiniert (siehe Abbildung 10.9). Diese Information können Sie natürlich noch anpassen, wenn es nötig ist. In der Spalte ÜBERSPRINGEN können Sie einzelne Datenfelder der Textdatei überspringen. Das heißt, wenn Sie einzelne Felder dort aktivieren, werden diese nicht mit in die Tabelle übernommen.
7. Klicken Sie auf die Schaltfläche SPEICHERN UNTER, um die soeben definierte Importspezifikation zu sichern.

Artikel Importspezifikation

Dateiformat: Mit Trennzeichen Feste Breite
 Feldtrennzeichen: ;
 Textqualifizierer: {kein}

Sprache: Deutsch
 Codepage: Westeuropäisch (Windows)

Daten, Zeiten und Zahlen

Datumsreihenfolge: TMJ Vierstellige Jahreszahlen
 Datumstrennzeichen: . Führende Nullen in Datumswerten
 Zeittrennzeichen: : Dezimalsymbol: .

Feldinformationen:

| Feldname | Datentyp | Indiziert | Überspr | | | | |
|----------|--------------|-----------|--------------------------|--|--|--|--|
| Feld1 | Kurzer Text | Nein | <input type="checkbox"/> | | | | |
| Feld2 | Kurzer Text | Nein | <input type="checkbox"/> | | | | |
| Feld3 | Kurzer Text | Nein | <input type="checkbox"/> | | | | |
| Feld4 | Kurzer Text | Nein | <input type="checkbox"/> | | | | |
| Feld5 | Kurzer Text | Nein | <input type="checkbox"/> | | | | |
| Feld6 | Kurzer Text | Nein | <input type="checkbox"/> | | | | |
| Feld7 | Double | Nein | <input type="checkbox"/> | | | | |
| Feld8 | Double | Nein | <input type="checkbox"/> | | | | |
| Feld9 | Long Integer | Nein | <input type="checkbox"/> | | | | |

Abbildung 10.9 Die Importspezifikation definieren

- Geben Sie der Importspezifikation einen Namen (siehe Abbildung 10.10), und bestätigen Sie mit OK. Merken Sie sich diesen Namen, Sie werden ihn später in Ihrer Prozedur benötigen.

Import/Export-Spezifikation speichern

Spezifikationsname: Artikel Importspezifikation

OK Abbrechen

Abbildung 10.10 Importspezifikation speichern

- Klicken Sie danach auf die Schaltfläche OK, um das Dialogfeld ARTIKEL IMPORT-SPEZIFIKATIONEN zu schließen.
- Sie kehren dadurch in den Textimport-Assistenten zurück. Klicken Sie auf WEITER, um zum nächsten Importschritt zu gelangen.
- Übergehen Sie auch den nächsten Schritt (siehe Abbildung 10.11) mit einem Klick auf die Schaltfläche WEITER.
- Im nächsten Dialog (siehe Abbildung 10.12) können Sie noch einmal festlegen, wie die einzelnen Felder definiert werden sollen. Da Sie diese Aufgabe aber bereits vorher über die Importspezifikation erledigt haben, klicken Sie hier auf die Schaltfläche WEITER.

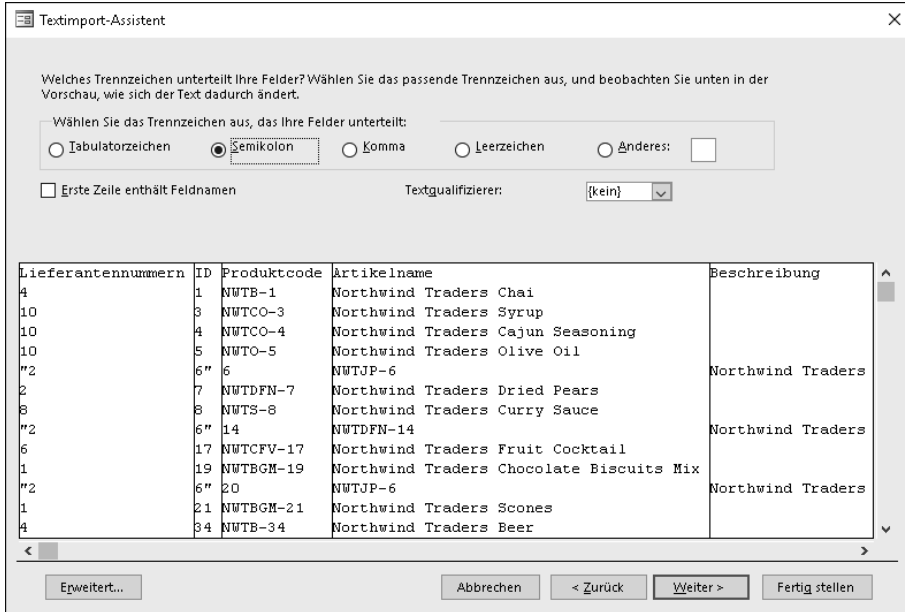


Abbildung 10.11 Trennzeichen festlegen

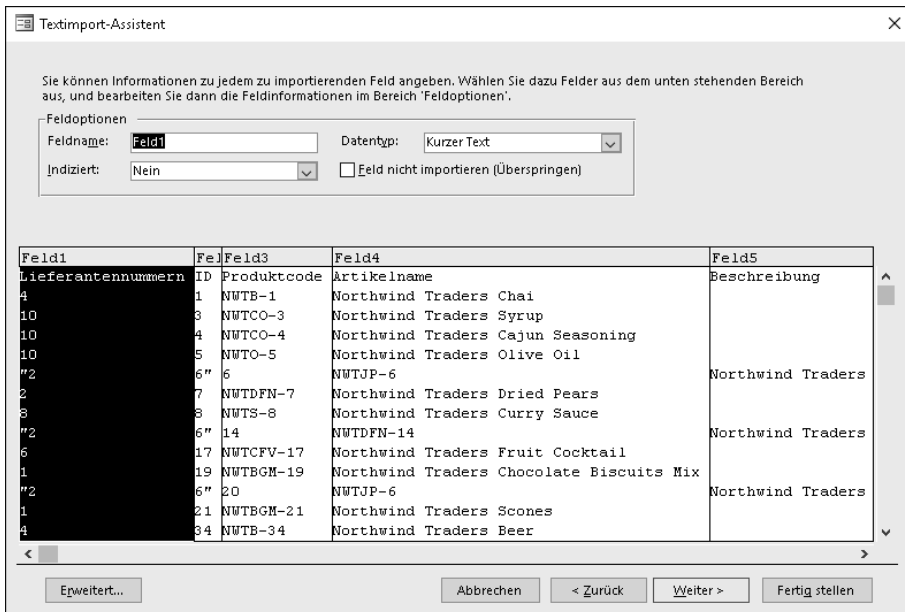


Abbildung 10.12 Felder festlegen

13. Im nächsten Schritt des Assistenten legen Sie fest, ob Sie einen Primärschlüssel anlegen möchten (siehe Abbildung 10.13). Dabei können Sie diese Aufgabe von Access selbst ausführen lassen. Klicken Sie danach auf WEITER.

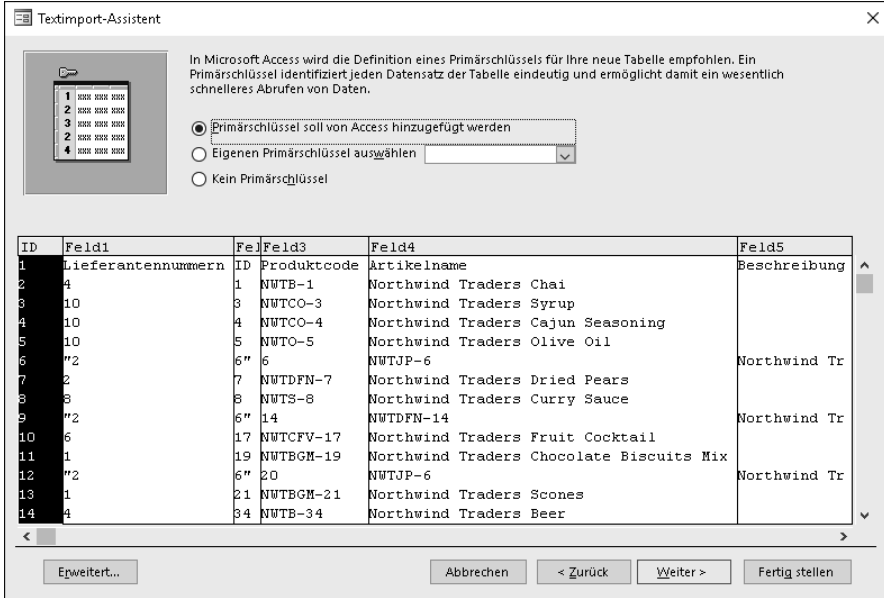


Abbildung 10.13 Ein Primärschlüssel kann angelegt werden.

14. Geben Sie nun an, in welche Tabelle Sie die Textdatei einfügen möchten (siehe Abbildung 10.14). Sollte diese Tabelle noch nicht existieren, legt Access sie automatisch an.



Abbildung 10.14 Zieltabelle angeben

15. Klicken Sie abschließend auf die Schaltfläche FERTIG STELLEN.

Sie haben nun einmalig den Texttransfer durchgeführt und eine Importspezifikation erstellt. Die soeben ausgeführten Arbeitsschritte müssen Sie in Zukunft nicht mehr wiederholen. Sie können stattdessen die Prozedur aus Listing 10.5 verwenden, die sich die Importspezifikation holt und den Datentransfer automatisch ausführt.

```
Sub TextdateiInTabelleEinlesen()
```

```
DoCmd.TransferText acImportDelim, "ArtikelImportspezifikation", _
    "Artikel2", Application.CurrentProject.Path & "\Artikel.csv", False
```

```
End Sub
```

Listing 10.5 Textdatei in eine Tabelle einlesen

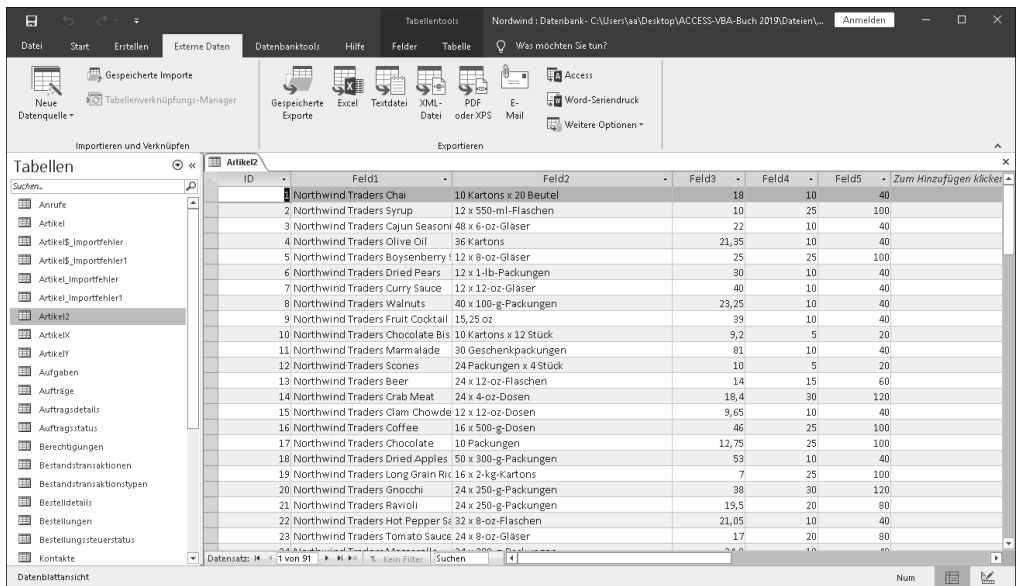


Abbildung 10.15 Die Textdatei wurde in eine neue Tabelle importiert.

Die Methode `TransferText` hat folgende Syntax:

```
TransferText(Transfertype, Spezifikationsname, Tabellename, Dateiname,
    Besitztfeldnamen, HTML-Tabellename)
```

- Im Argument `Transfertype` geben Sie über eine Konstante an, was Sie konkret machen möchten. Sie haben dabei die Auswahl zwischen Import- und Exportkonstanten. Die genauen Bezeichnungen dieser Konstanten können Sie der Onlinehilfe entnehmen.

- ▶ Im Argument `Spezifikationsname` geben Sie den Namen der Spezifikation an, den Sie vorher bestimmt haben.
- ▶ Das Argument `Tabellename` gibt den Namen der Zieltabelle an, in die der Transfer führen soll.
- ▶ Die Datenquelle übergeben Sie im Argument `Dateiname`. Geben Sie hierfür den kompletten Dateipfad sowie den Dateinamen an.
- ▶ Setzen Sie das Argument `BesitztFeldnamen` auf den Wert `True`, wenn die zu importierende Textdatei als erste Zeile eine Überschriftenzeile enthält. Liegen nur die Daten ohne Überschriftenzeile vor, dann setzen Sie dieses Argument auf den Wert `False` oder lassen es weg.
- ▶ Das optionale Argument `HTML-Tabellename` ist nur dann von Interesse, wenn Sie HTML-Dateien in eine Datenbank importieren möchten. Damit geben Sie den Namen der Tabelle oder Liste in der HTML-Datei an, die Sie importieren möchten.

10.2 Access im Zusammenspiel mit Word

Möchten Sie Daten von Access nach Word übertragen, setzen Sie im ersten Schritt die Methoden `CreateObject` und `GetObject` ein. So rufen Sie Ihre Textverarbeitung Word auf. Dabei prüfen Sie mit der Funktion `GetObject`, ob Word bereits gestartet ist. Wenn nicht, bekommen Sie den Fehler Nummer 429 zurück, der besagt, dass die Objektkomponente nicht verfügbar ist. In diesem Fall erstellen Sie über die Funktion `CreateObject` einen Verweis auf Word.

Im nächsten Beispiel soll eine Access-Tabelle in ein neues Word-Dokument eingefügt werden. Dabei soll in Word eine neue, leere Tabelle entstehen, die danach mit den einzelnen Datenfeldern aus der Datenbanktabelle gefüllt wird. Die etwas längere Prozedur für diese Aufgabe sehen Sie in Listing 10.6.

```
Sub AccessTabelleNachWord()  
    Dim objWordApp As Object  
    Dim objWordDoc As Object  
    Dim objAktPos As Object  
    Dim rst As New ADODB.Recordset  
    Dim tbl As Variant  
    Dim x As Integer  
  
    On Error GoTo Fehler  
    rst.Open "Artikel", CurrentProject.Connection  
    On Error Resume Next  
    Set objWordApp = GetObject(, "Word.Application")  
    If Err.Number = 429 Then
```

```
Set objWordApp = CreateObject("Word.Application")
Err.Number = 0
End If

objWordApp.Visible = True
Set objWordDoc = objWordApp.Documents.Add

With objWordApp.Selection
    .TypeText Text:="Artikelliste aus: " & CurrentProject.Name
    .TypeParagraph
    .TypeText Text:="vom " & Format(Now(), "dd-mmm-yyyy")
    .TypeParagraph
End With

Set objAktPos = objWordDoc.Range(Start:=0, end:=0)
objWordDoc.Tables.Add Range:=objAktPos, NumRows:=80, NumColumns:=6
Set tbl = objWordDoc.Tables(1)
x = 1

Do Until rst.EOF

    With tbl
        If IsNull(rst.Fields("Artikelname").Value) Then
            .Cell(x, 1).Range.Text = "Kein Name"
        Else
            .Cell(x, 1).Range.Text = rst!Artikelname
        End If

        .Cell(x, 2).Range.Text = rst!Kategorie
        .Cell(x, 3).Range.Text = rst!Listenpreis
        .Cell(x, 4).Range.Text = rst!Standardkosten
        x = x + 1

        rst.MoveNext
    End With

Loop

Set objWordApp = Nothing
Set objWordDoc = Nothing
Exit Sub
```

Fehler:

```
MsgBox Err.Description  
End Sub
```

Listing 10.6 Teile einer Tabelle nach Word übertragen

Legen Sie im ersten Schritt die benötigten Objektvariablen an. Unter anderem brauchen Sie ein Objekt, um die Textverarbeitung Word zu verwalten, und eines, um das neue Dokument darin anzusprechen. Für Ihre Access-Datentabelle benötigen Sie ein Recordset-Objekt, über das Sie alle Sätze in der Tabelle *Artikel* programmiertechnisch bearbeiten können. Nachdem Sie Ihre Tabelle sowie die Textverarbeitung Word mit der Methode `Open` bzw. `CreateObject` gestartet haben, machen Sie die Word-Anwendung sichtbar. Dazu setzen Sie die Eigenschaft `Visible` auf den Wert `True`.

Mit der Methode `Add` fügen Sie jetzt ein neues, noch leeres Dokument ein. Damit Sie später dieses neue Dokument weiterverarbeiten können, speichern Sie es in der Objektvariablen `WordDoc`.

Sie haben jetzt eine Mischung aus Access- und Word-VBA-Befehlen. Unterscheiden können Sie diese, indem Sie immer das anführende Objekt ansehen. `WordObj` und `WordDoc` enthalten die Word-VBA-Befehle, und das Objekt `rst` enthält alle Access-VBA-Befehle.

```
With objWordApp.Selection  
  .TypeText Text:="Artikelliste aus: " & _  
  CurrentProject.Name  
  .TypeParagraph  
  .TypeText Text:="vom " & Format(Now(), _  
    "dd-mmm-yyyy")  
  .TypeParagraph  
End With
```

Mit der Word-Eigenschaft `Selection` geben Sie einen markierten Bereich oder eine Einfügestelle im Dokument an. Über die Methode `TypeText` fügen Sie an der Einfügestelle einen beliebigen Text ein. Das Access-Objekt `CurrentProject` in Verbindung mit der Eigenschaft `Name` gibt Auskunft darüber, woher die Daten stammen, die übertragen werden sollen. Nach dem Einfügen des Textes fügen Sie einen leeren Absatz ein und geben danach das formatierte Tagesdatum aus.

Fügen Sie jetzt eine neue Tabelle in Ihr Dokument ein. Dazu müssen Sie vorher die Position im Dokument bestimmen, an der die neue Tabelle eingefügt werden soll. Diese Information speichern Sie in der Objektvariablen `AktPos`. Mit der Methode `Add`, die Sie auf das Objekt `Tables` anwenden, fügen Sie Ihre neue Tabelle in das Dokument ein.

Die Methode `Add` zum Einfügen einer Tabelle hat folgende Syntax:

```
Add(Range, NumRows, NumColumns DefaultTableBehavior, AutoFitBehavior)
```

- ▶ Über das Argument `Range` geben Sie die genaue Einfügeposition der Tabelle bekannt.
- ▶ Im Argument `NumRows` legen Sie die Anzahl der Zeilen fest, die in der Tabelle enthalten sein sollen.
- ▶ Das Argument `NumColumns` legt die Anzahl der Spalten fest, die in der Tabelle enthalten sein sollen.
- ▶ Mithilfe des Arguments `DefaultTableBehavior` können Sie entscheiden, ob sich die Zellengröße in der Tabelle automatisch ändert, wenn zu viele Zeichen in eine Zelle übertragen werden. Standardmäßig ist diese Einstellung über die Konstante `wdWord9TableBehavior` aktiviert und kann daher auch weggelassen werden. Möchten Sie, dass sich die Zellgröße nicht ändert, dann verwenden Sie die Konstante `wdWord8TableBehavior`.
- ▶ Im letzten Argument, `AutoFitBehavior`, legen Sie für das AutoAnpassen der Zellen die Regeln fest, nach denen die Tabellengröße in Word geändert wird. Dies kann eine der folgenden `WdAutoFitBehavior`-Konstanten sein: `wdAutoFitContent`, `wdAutoFitFixed` oder `wdAutoFitWindow`. Wenn `DefaultTableBehavior` auf `wdWord8TableBehavior` gesetzt ist, wird dieses Argument ignoriert.

Um die Tabelle elegant ansprechen zu können, bilden Sie die Objektvariable `TabWort` und speichern in ihr die erste Tabelle in Ihrem Dokument.

In einer anschließenden Schleife wird die Access-Tabelle *Artikel* durchlaufen, und die einzelnen Feldinhalte werden in die dafür vorgesehenen Zellen der Word-Tabelle übertragen. Setzen Sie für diese Aufgabe das Word-Objekt `Cell` ein. Dieses Objekt erwartet einen Wert, der die jeweilige Zeile sowie die Spalte identifiziert. Mit dem Objekt `Range` ist die Zellenfläche der einzelnen Tabellenzelle gemeint, die Sie über die Eigenschaft `Text` füllen können. Als Text setzen Sie dabei natürlich die einzelnen Datenfelder Ihrer Access-Tabelle ein. Addieren Sie bei jedem eingelesenen Satz die Zählvariable `x`, die für die Zeile steht, um den Wert 1 zu erhöhen.

Vergessen Sie nicht, am Ende der Schleife den Zeiger über die Methode `MoveNext` auf den nächsten Datensatz in der Artikeltabelle zu setzen, da Sie sonst eine Endlosschleife produzieren.

Geben Sie am Ende der Prozedur aus Listing 10.6 den reservierten Speicher wieder frei, den Sie für die Objektvariablen `WordObj` und `WordDoc` benötigt haben.

Eine weitere sehr komfortable Art und Weise, eine Tabelle in ein Word-Dokument zu transferieren, ist Methode `OutputTo`. Die Methode aus Listing 10.7 nimmt Ihnen den Datentransfer weitestgehend ab. Es wird die komplette Tabelle übertragen.

Sub TabelleNachWordTransferieren()

```
DoCmd.OutputTo acOutputTable, "Artikel", acFormatRTF, _
Application.CurrentProject.Path & "\Artikel.doc", True
```

End Sub

Listing 10.7 Komplette Tabelle nach Word übertragen

Da ich diese Methode bereits weiter oben im Kapitel besprochen habe, gehe ich an dieser Stelle nicht weiter darauf ein.

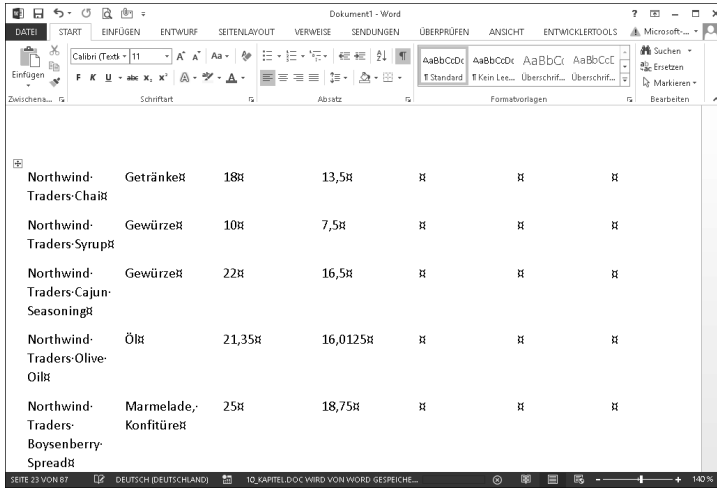


Abbildung 10.16 Die Ausgabe der Artikeltabelle in einem Word-Dokument

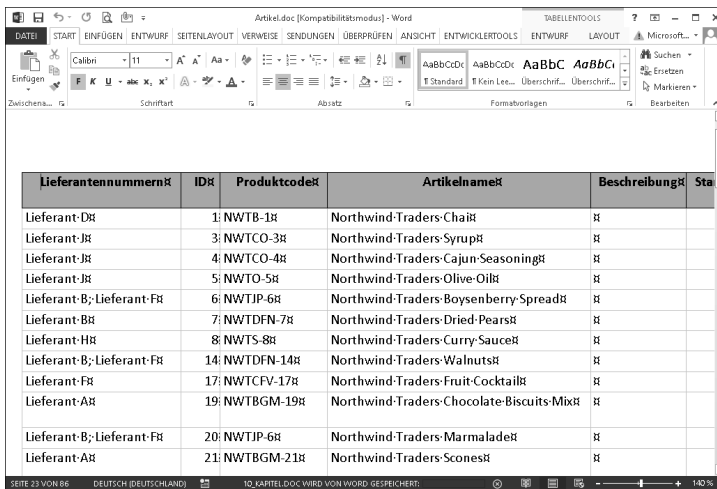


Abbildung 10.17 Die in ein Word-Dokument übertragene Access-Tabelle

10.2.1 Die Adressdatenbank anlegen

Die Adressdatenbank, die später als Datenquelle für Ihre Textverarbeitung dienen soll, enthält die Datenfelder KUNDENNR, KUNDENNAME, STRASSE, PLZ und ORT. Erstellen Sie jetzt eine neue Datenbank, und speichern Sie sie unter dem Namen *KundenDB.accdb* im Verzeichnis *C:\Eigene Dateien*.

Danach erstellen Sie die Tabelle *Adressen*, die in der Entwurfsansicht den Aufbau aus Abbildung 10.18 hat.

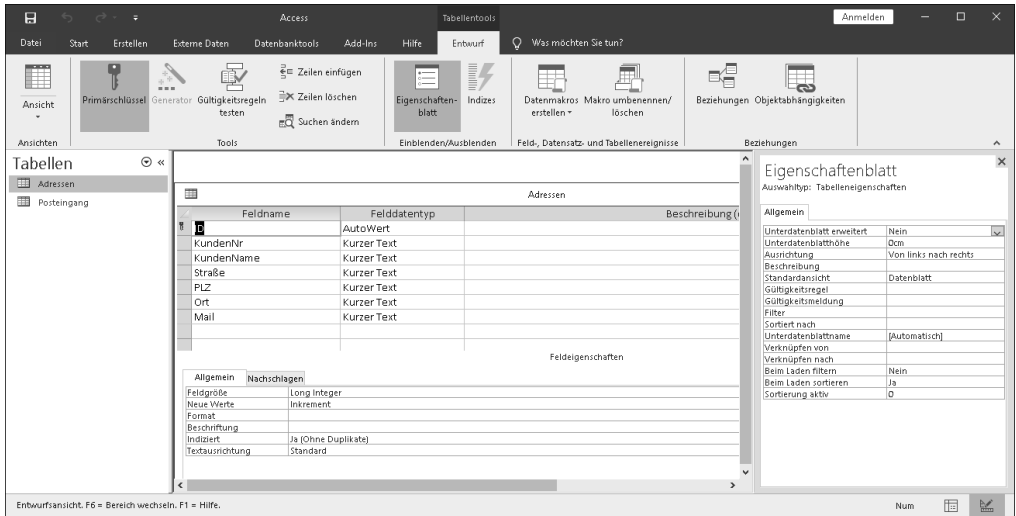


Abbildung 10.18 Die Datenfelder der Tabelle »Adressen«

Schließen Sie danach die Entwurfsansicht, und öffnen Sie die Tabelle *Adressen*, um ein paar Kundenadressen zu erfassen. Schließen Sie dann die Datenbank, und starten Sie Ihre Textverarbeitung.

10.2.2 Das Word-Dokument anlegen

Jetzt benötigen Sie ein Word-Dokument, von dem aus auf die Access-Datenbank zugegriffen werden soll. Da es hier lediglich um das Prinzip gehen soll, reicht uns dabei ein recht einfaches Dokument. Fügen Sie also ein neues Dokument ein, und integrieren Sie eine Tabelle mit zwei Spalten und drei Zeilen. In der ersten Spalte erfassen Sie die Feldbezeichnungen, die Ihren Anwendern als Orientierungshilfe dienen sollen (siehe Abbildung 10.19).

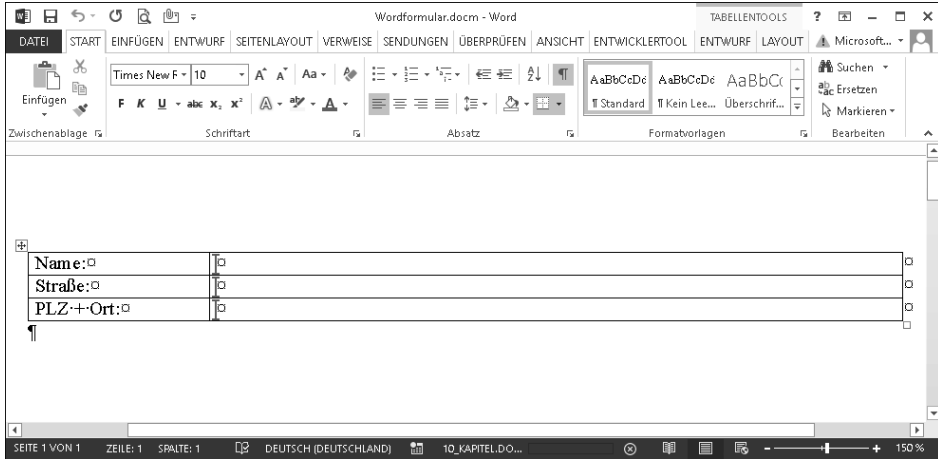


Abbildung 10.19 Das Word-Dokument im Rohbau

Fügen Sie nun folgende Textmarken in der zweiten Spalte ein: KU_NA, KU_STR, KU_PLZ und KU_ORT (siehe Abbildung 10.20). Die Textmarken fügen Sie ein, indem Sie den Mauszeiger auf die gewünschte Zelle in der Tabelle setzen und aus dem Ribbon EINFÜGEN den Befehl TEXTMARKE wählen.

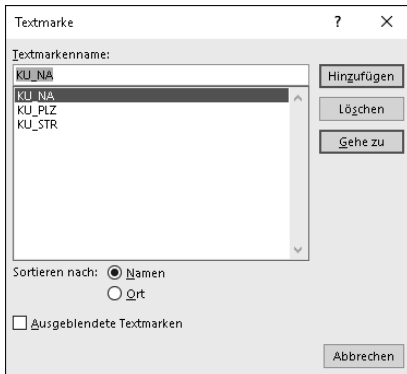


Abbildung 10.20 Textmarke einfügen

Haben Sie alle Textmarken eingefügt, setzen Sie den Code ein, der die Daten aus der Access-Tabelle *Adressen* holen und im Dokument an der gewünschten Stelle positionieren soll. Da Sie dabei nicht die komplette Datentabelle übertragen wollen, definieren Sie als Suchkriterium die »KundenNr«. In diesem Beispiel ist vorgesehen, dass ein Kunde bei Ihrer Firma anruft und seine Kundennummer durchgibt. Der jeweilige Mitarbeiter soll dann diese Kundennummer in ein Dialogfeld eingeben und auf die Schaltfläche OK klicken. Im Hintergrund wird anschließend anhand dieser Kundennummer der gewünschte Kunde ermittelt. Die so ermittelten Adressdaten werden dann automatisch in das Word-Formular eingefügt.

10.2.3 Den VBA-Code erfassen

Sie finden den kompletten Code aus Listing 10.8 in den Materialien zum Buch im Ordner *Kap10* unter dem Namen *Wordformular.docm*.

Geben Sie nun den Code ein, indem Sie die nächsten Arbeitsschritte ausführen:

1. Drücken Sie die Tastenkombination **Alt**+**F11**.
2. Klicken Sie mit der rechten Maustaste in den Projekt-Explorer, und wählen Sie den Befehl **EINFÜGEN • MODUL**.
3. Erfassen Sie auf der rechten Seite die Prozedur aus Listing 10.8.

```

Sub DatenVonACCESSNachWORD()
    Dim objWord As Object
    Dim con As ADODB.Connection
    Dim rst As ADODB.Recordset
    Dim str As String
    Dim strKundenNr As String
    Dim strKundenname As String
    Dim strStraße As String
    Dim strPLZ As String
    Dim strOrt As String

    str = InputBox("Geben Sie die Kundennummer ein!")

    If str = "" Then Exit Sub
    str = "KundenNr=" & str & ""

    Set con = New ADODB.Connection
    With con
        .Provider = "Microsoft.ACE.OLEDB.12.0"
        .Open ThisDocument.Path & "\KundenDB.accdb"
    End With

    Set rst = New ADODB.Recordset
    With rst
        .Open Source:="Adressen", ActiveConnection:=con, _
            CursorType:=adOpenKeyset, LockType:=adLockOptimistic

        .Find Criteria:=str, SearchDirection:=adSearchForward

        If Not .EOF Then
            strKundenNr = .Fields("KundenNr").Value
            strKundenname = .Fields("KundenName").Value
            strStraße = .Fields("Straße").Value
        End If
    End With
End Sub

```

```

        strPLZ = .Fields("PLZ").Value
        strOrt = .Fields("Ort").Value
    Else
        MsgBox "Datensatz nicht gefunden"
    End If
    .Close
End With
con.Close

Set rst = Nothing

On Error Resume Next
Set objWord = GetObject(, "Word.Application")

With objWord
    .ActiveDocument.Bookmarks("KU_NA").Range.Text = strKundenname
    .ActiveDocument.Bookmarks("KU_STR").Range.Text = strStraße
    .ActiveDocument.Bookmarks("KU_PLZ").Range.Text = strPLZ
    .ActiveDocument.Bookmarks("KU_ORT").Range.Text = strOrt
    .ActiveDocument.Save
End With

Set objWord = Nothing
End Sub

```

Listing 10.8 Gezielt auf eine Access-Tabelle zugreifen und Daten nach Word übertragen

4. Definieren Sie im ersten Schritt die Variablen, die Sie für diese Aktion brauchen. Dazu gehören unter anderem die `String`-Variablen, in denen Sie die Ergebnisse aus der Access-Tabelle zwischenspeichern. Des Weiteren benötigen Sie Objekte, mit denen Sie die Anwendungen Access und Word steuern, und eine Objektvariable vom Typ `Recordset`, mit der Sie die Access-Tabelle verarbeiten.
5. Gleich im Anschluss daran rufen Sie die Funktion `InputBox` auf und bitten den Anwender oder die Anwenderin, eine Kundennummer einzugeben. Prüfen Sie die Eingabe, und bilden Sie danach den Suchstring. Dieser Suchstring muss denselben Namen enthalten wie derjenige, den Sie in Ihrer Access-Tabelle definiert haben.
6. Öffnen Sie danach die beteiligte Datenbank sowie die Tabelle *Adressen*. Setzen Sie die Methode `Find` ein, und übergeben Sie ihr den Suchstring, den Sie sich in der Variablen `str` erstellt haben. War die Suche über die Kundennummer erfolgreich, dann meldet die Eigenschaft `EOF` den Wert `False`. Diese Eigenschaft hätte übrigens den Wert `True` gemeldet, wenn die Suche erfolglos gewesen wäre. Dann wäre näm-

lich der letzte Satz in der Tabelle erreicht. Weisen Sie nun den Variablen die gefundenen Werte zu, und schließen Sie die Access-Tabelle gleich danach über die Methode Close. Gleichzeitig schließen Sie die Datenbank.

- Über die Funktion GetObject stellen Sie einen Verweis zum geöffneten Dokument her und speichern diesen Verweis unter der Objektvariablen WordObj. Sie haben jetzt Zugriff auf alle Word-VBA-Befehle.

Auf die Textmarken greifen Sie über die Auflistung Bookmarks zu. Übertragen Sie danach die Inhalte der Variablen in die dazugehörigen Textmarken.

- Starten Sie nun die Prozedur, indem Sie in Word im Ribbon ENTWICKLERTOOLS auf die Schaltfläche MAKROS klicken und das entsprechende Makro aufrufen.

Es wird daraufhin ein Dialogfeld angezeigt, in das Sie z. B. die Kundennummer »K101« eingeben und dies mit OK bestätigen (siehe Abbildung 10.21).

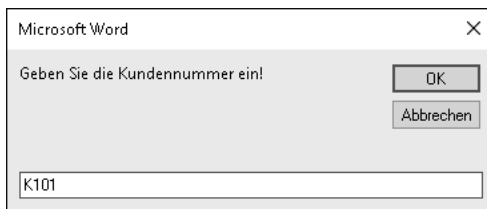


Abbildung 10.21 Die Eingabe der Kundennummer erfolgt über eine »InputBox«.

Selbstverständlich können Sie diese Lösung weiter ausbauen, da Sie mithilfe von Textmarken jede Stelle in Dokumenten ansteuern und füllen können.

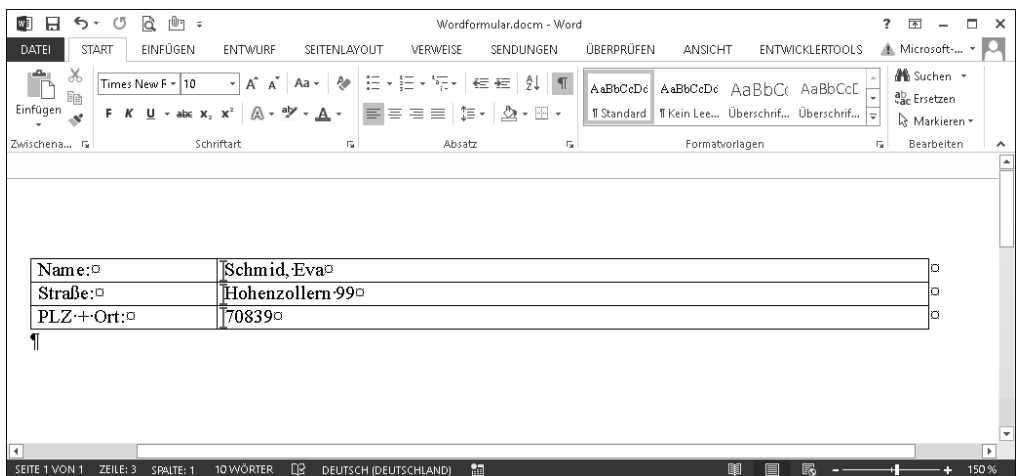


Abbildung 10.22 Die Daten wurden erfolgreich übertragen.



ADO-Bibliothek einbinden

Bevor Sie die gerade erstellte Prozedur starten, binden Sie die ADO-Bibliothek noch in die Entwicklungsumgebung von Word ein. Dazu rufen Sie den Menübefehl **EXTRAS • VERWEISE** in der Entwicklungsumgebung auf. Aktivieren Sie die Bibliothek *Microsoft ActiveX Data Objects*. Welche Version Sie von dieser Komponente zur Auswahl haben, hängt von Ihrer Access-Version ab. Bei Access bzw. Office 2016/2019/2021 oder 365 finden Sie beispielsweise die Version 6.1 vor. Bestätigen Sie diese Einstellung mit OK.

10.3 Outlook und Access

Der Datenaustausch zwischen Outlook und Access bietet einige interessante Anwendungsmöglichkeiten:

- ▶ Verwenden Sie Microsoft Outlook als Mailing-Programm.
- ▶ Setzen Sie die Kontakte in Outlook ein, um Kontakte einzugeben und abzufragen.
- ▶ Sichern Sie Ihre Kontakte regelmäßig.
- ▶ Setzen Sie Microsoft Access als Kommunikationspartner im Office-Paket ein.

10.3.1 Adresstabelle in den Outlook-Kontaktordner übertragen

Stellen Sie sich vor, Sie haben erst seit Kurzem Outlook installiert. Ihre Adressen haben Sie bisher in einer Access-Tabelle verwaltet. Wie bekommen Sie jetzt Ihre Access-Adressdaten zu Outlook? Bevor Sie diese Aufgabe über eine Prozedur lösen, sehen Sie sich in Abbildung 10.23 den Aufbau der Kontaktstabelle in Access an.

Wenn Sie dieses Beispiel Schritt für Schritt nachverfolgen, dann achten Sie auf die Benennung der Datenfelder. Diese entsprechen später den Bezeichnungen in der Prozedur. Speichern Sie die Adresstabelle unter dem Namen *Kontakte*.

Im nächsten Schritt öffnen Sie die Tabelle *Kontakte* und geben ein paar Adressdaten ein (siehe Abbildung 10.24).

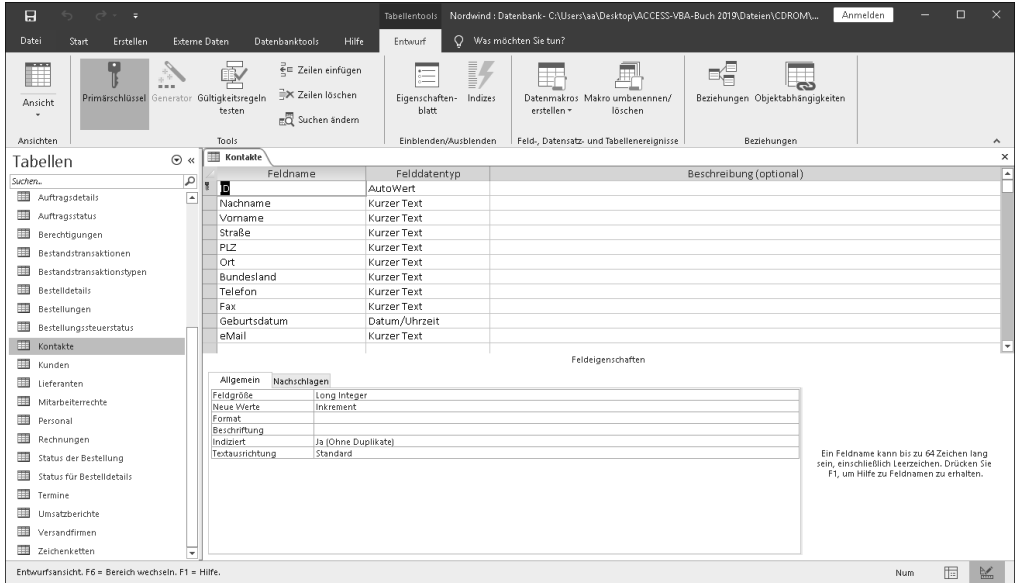


Abbildung 10.23 Der Aufbau der Adressentabelle in der Entwurfsansicht

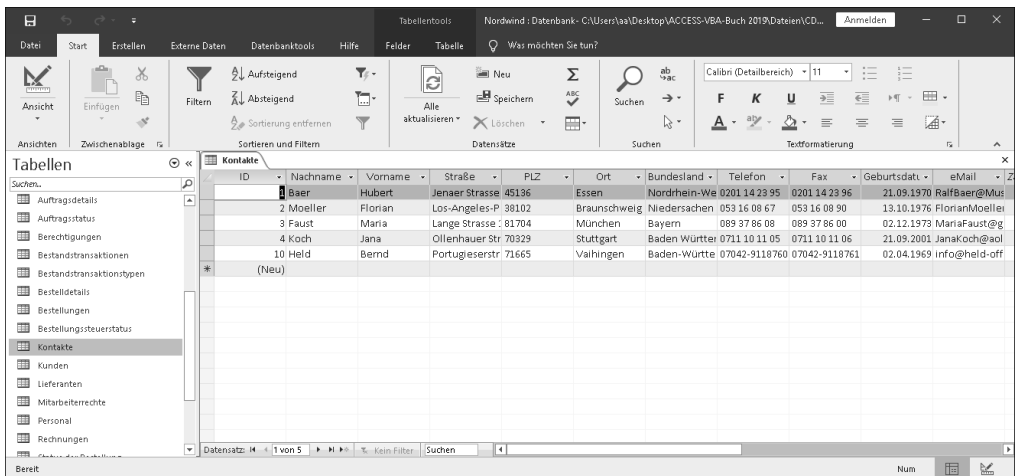


Abbildung 10.24 Ein paar Adressen, bereit für die Übernahme nach Outlook

Bevor Sie die Prozedur erstellen, binden Sie die Bibliothek *Microsoft Outlook* ein. Dazu wechseln Sie über die Tastenkombination **[Alt] + [F11]** in die Entwicklungsumgebung von Access und rufen im Menü **EXTRAS** den Befehl **VERWEISE** auf (siehe Abbildung 10.25).

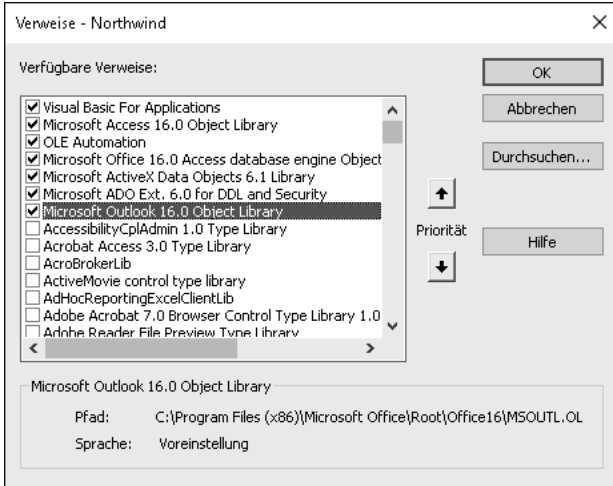


Abbildung 10.25 Die Outlook-Bibliothek einbinden

Suchen Sie in der Liste VERFÜGBARE VERWEISE nach dem Eintrag MICROSOFT OUTLOOK 15.0 OBJECT LIBRARY für Office 2013 oder MICROSOFT OUTLOOK 16.0 OBJECT LIBRARY für Office 2016/2019/2021/365. Aktivieren Sie diesen Eintrag, und bestätigen Sie mit der Schaltfläche OK.

Fügen Sie jetzt die Prozedur aus Listing 10.9 ein.

```

Sub KontakteVonAccessInOutlookUebertragen()
    Dim objOutlApp As Outlook.Application
    Dim objContactItem As Outlook.ContactItem
    Dim rst As New ADODB.Recordset
    Dim intZ As Integer

    On Error GoTo Fehler
    Set objOutlApp = CreateObject("Outlook.Application")

    rst.Open "Kontakte", CurrentProject.Connection

    Do While Not rst.EOF
        If KontaktSchonDa(rst!ID) = False Then
            Set objContactItem = objOutlApp.CreateItem(olContactItem)

            With objContactItem

                .LastName = rst!Nachname
                .FirstName = rst!Vorname
                .BusinessAddressStreet = rst!Straße
            End With
        End If
    End Do
End Sub

```

```

        .BusinessAddressPostalCode = rst!PLZ
        .BusinessAddressCity = rst!Ort
        .BusinessAddressState = rst!Bundesland
        .BusinessTelephoneNumber = rst!Telefon
        .BusinessFaxNumber = rst!Fax
        .Birthday = rst!Geburtsdatum
        .EmailAddress = rst!Email
        .Account = rst!ID
        .Save
        intZ = intZ + 1

    End With
End If
    rst.MoveNext
Loop

MsgBox "Es wurden " & intZ & " Kontakte übertragen!", vbInformation

rst.Close
Set objContactItem = Nothing
Set objOutlApp = Nothing
Exit Sub

Fehler:
    MsgBox "Fehler aufgetreten!"
End Sub

```

Listing 10.9 Adressdaten aus Access in den Kontaktordner von Outlook übertragen

Definieren Sie im ersten Schritt zwei Objektvariablen für Outlook. Die eine Variable, `objOutlApp`, gibt Ihnen die Möglichkeit, das Mailing-Programm direkt über VBA-Befehle anzusprechen. Die zweite Objektvariable, `objContactItem`, wählen Sie, um später auf den Kontaktordner von Outlook zugreifen zu können.

Setzen Sie danach die Funktion `CreateObject` ein, um einen Verweis auf die *Outlook*-Bibliothek zu setzen.

Öffnen Sie im nächsten Schritt Ihre Access-Tabelle *Kontakte* über die Methode `Open`. Setzen Sie danach eine Schleife auf, die so lange durchlaufen wird, bis der letzte Datensatz in der Tabelle abgearbeitet ist. Damit Satz für Satz verarbeitet werden kann, müssen Sie daran denken, am Ende der Schleife die Methode `MoveNext` aufzurufen.

Mit der Methode `CreateItem` erstellen Sie ein neues Outlook-Objekt. Welches Objekt Sie genau brauchen, können Sie über eine Konstante festlegen. Dabei stehen Ihnen folgende Konstanten zur Verfügung:

- ▶ `olAppointmentItem` fügt einen neuen Termin in Ihren Terminkalender ein.
- ▶ `olContactItem` erstellt einen neuen Kontakt.
- ▶ `olDistributionListItem` erstellt einen Eintrag in der Verteilerliste von Outlook.
- ▶ `olJournalItem` legt einen neuen Journaleintrag an.
- ▶ `olMailItem` erzeugt einen neuen E-Mail-Eintrag.
- ▶ `olNoteItem` legt eine neue Notiz an.
- ▶ `olPostItem` verschickt eine E-Mail.
- ▶ `olTaskItem` fügt einen neuen Eintrag in Ihre Aufgabenliste ein.

Für unser Beispiel erstellen Sie also einen Kontakteintrag und verwenden daher die Konstante `olContactItem`. Damit wird der Kontakt angelegt und wartet auf seine Befüllung. Übertragen Sie die einzelnen Datenfelder aus Ihrer Access-Tabelle, indem Sie die Eigenschaften aus Tabelle 10.1 verwenden.

| Eigenschaft | Bedeutung |
|--|--|
| <code>LastName</code> | Nachname der Kontaktperson |
| <code>FirstName</code> | Vorname der Kontaktperson |
| <code>BusinessAddressStreet</code> | Straßenadresse des Arbeitsplatzes der Kontaktperson |
| <code>BusinessAddressPostalCode</code> | Postleitzahl des Arbeitgebers |
| <code>BusinessAddressCity</code> | Geschäftsstandort |
| <code>BusinessAddressState</code> | Bundesland des Arbeitgebers |
| <code>BusinessTelephoneNumber</code> | geschäftliche Telefonnummer |
| <code>BusinessFaxNumber</code> | Faxnummer am Arbeitsplatz |
| <code>Birthday</code> | Geburtstag der Kontaktperson |
| <code>EmailAddress</code> | E-Mail-Adresse der Kontaktperson |
| <code>Account</code> | Dieses Feld wird in dieser Lösung verwendet, um die eindeutige ID des Kontakts zu verwalten. |

Tabelle 10.1 Einige Angaben zur Kontaktperson

Wenn Sie alle Informationen in den Outlook-Kontakt übertragen haben, verwenden Sie die Methode `Save`, um den Kontakt im Kontaktordner zu speichern. Um am Ende der Prozedur ausgeben zu können, wie viele Kontakte übertragen wurden, erhöhen Sie die Zählvariable `i` nach jedem Schleifendurchlauf um den Wert 1. Wurden alle Da-

tenbankfelder der Tabelle *Kontakte* abgearbeitet, wird die Schleife verlassen. Geben Sie am Ende der Prozedur in einer Bildschirmmeldung die Anzahl der übertragenen Adressen aus. Vergessen Sie nicht, die Verweise auf die Outlook-Objekte zu entfernen, um den reservierten Arbeitsspeicher wieder freizugeben.

Bevor Sie den Kontakt aus der Access-Tabelle in Outlook anlegen, prüfen Sie über die Funktion `KontaktSchonDa` wie in Listing 10.10, ob er nicht bereits existiert; in diesem Fall muss er nicht noch einmal angelegt werden.

```
Function KontaktSchonDa(ID As Long) As Boolean
    Dim OutL As Object
    Dim oFolder As Outlook.MAPIFolder
    Dim objNameSpace As Outlook.NameSpace
    Dim intZ As Integer

    Set OutL = CreateObject("Outlook.Application")
    Set objNameSpace = OutL.GetNamespace("MAPI")
    Set oFolder = objNameSpace.GetDefaultFolder(olFolderContacts)

    For intZ = 1 To oFolder.Items.Count
        If oFolder.Items(intZ).Account = ID Then
            KontaktSchonDa = True
            Exit Function
        End If
    Next intZ

End Function
```

Listing 10.10 Diese Funktion prüft, ob ein Kontakt mit der ID bereits angelegt wurde.

Selbstverständlich gibt es noch einige weitere Felder, die Sie in Outlook im Kontaktordner speichern können. Wenn Sie alle verfügbaren Outlook-Kontaktfelder im Direktfenster der Entwicklungsumgebung auslesen möchten, dann starten Sie die Prozedur aus Listing 10.11.

```
Sub KontakteFelderauslesen()
    Dim OutKontakt As ContactItem
    Dim OutEigenschaft As ItemProperties
    Dim intZ As Integer

    Set OutObjekt = New Outlook.Application
    Set OutKontakt = OutObjekt.CreateItem(olContactItem)
    Set OutEigenschaft = OutKontakt.ItemProperties
```

```

For intZ = 1 To OutEigenschaft.Count
    On Error Resume Next
    Debug.Print OutEigenschaft.Item(intZ).Name
Next intZ

End Sub

```

Listing 10.11 Alle Outlook-Kontaktfelder auslesen

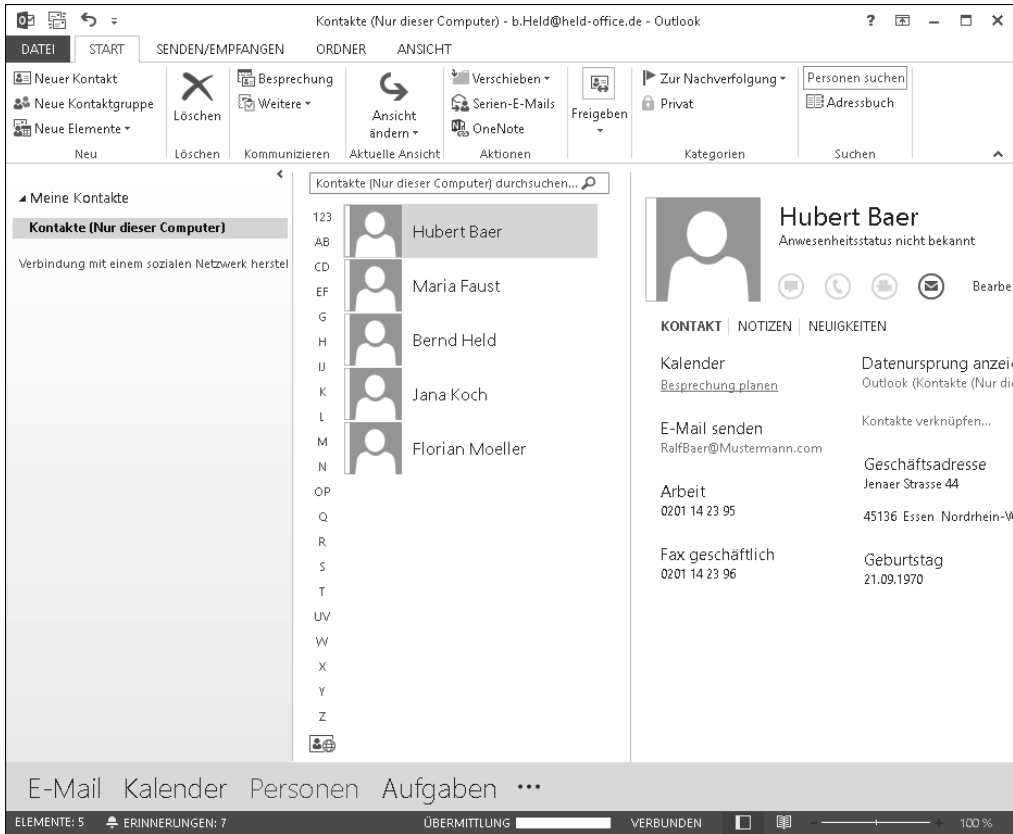


Abbildung 10.26 Die Adressdaten wurden in den Kontaktordner von Outlook transferiert.

Erstellen Sie im ersten Schritt der Prozedur aus Listing 10.11 einen Verweis auf die Outlook-Bibliothek. Danach legen Sie über die Methode `CreateItem` einen leeren Kontakt an. An alle verfügbaren Felder eines Kontakts kommen Sie über die Auflistung `ItemProperties` heran. In einer `For ... Next`-Schleife arbeiten Sie ein Feld nach dem anderen ab und geben den jeweiligen Feldnamen über die Eigenschaft `Name` mit der Anweisung `Debug.Print` im Direktfenster der Entwicklungsumgebung aus.

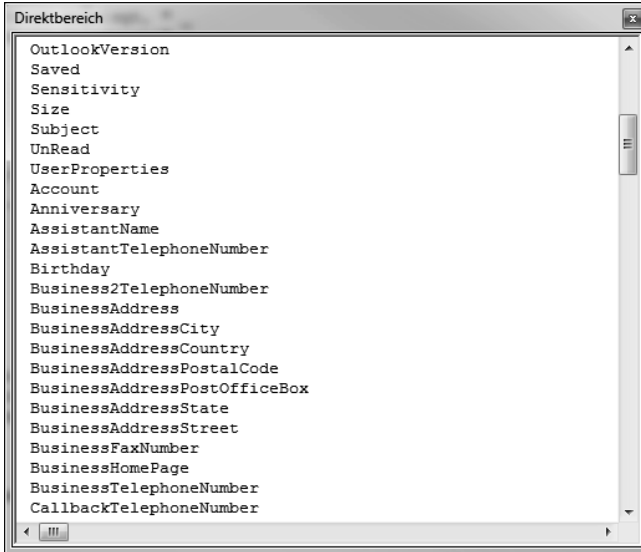


Abbildung 10.27 Alle verfügbaren Kontaktfelder von Outlook im Direktfenster

10.3.2 Den Kontaktorder in einer Access-Tabelle sichern

In der vorherigen Aufgabe haben Sie Kontakte aus einer Access-Tabelle als Kontakte in Ihrem Outlook-Kontaktordner angelegt. Dabei haben Sie die ID der Tabelle verwendet, um zu prüfen, ob der Kontakt bereits in Outlook angelegt wurde. Für diesen Zweck haben Sie in das Outlook-Feld »Account« die ID der Access-Tabelle geschrieben.

Jetzt werden jedoch alle Outlook-Kontakte in die gleiche Access-Tabelle zurückgeschrieben. Auch hier muss geprüft werden, ob ein neuer Kontakt im Outlook-Ordner angelegt wurde. Wenn ja, dann ist das Outlook-Feld »Account« leer. Starten Sie die Prozedur aus Listing 10.12, um neue Kontakte in Access anzulegen bzw. bestehende Kontakte anzupassen.

```
Sub KontakteAusOutlookHolen()
    Dim objArbeitsverz As Object
    Dim objKon As Object
    Dim rst As Recordset
    Dim db As Database
    Dim intZ As Integer
    Dim intNeu As Integer
    Dim intUpd As Integer
    Dim str As String
    Dim objOutlApp As New Outlook.Application
```

```
On Error GoTo Fehler
Set db = CurrentDb

Set rst = db.OpenRecordset("Kontakte", dbOpenDynaset)

Set objArbeitsverz =
objOutlApp.GetNamespace("MAPI").GetDefaultFolder(olFolderContacts)

For intZ = 1 To objArbeitsverz.Items.Count
    Set objKon = objArbeitsverz.Items(intZ)
    str = objKon.Account

    If str = "" Then
        str = "ID = 0"
    Else
        str = "ID = " & str
    End If

    With objKon

        rst.FindFirst str
        If rst.NoMatch Then
            rst.AddNew
            intNeu = intNeu + 1
        Else
            rst.Edit
            intUpd = intUpd + 1
        End If
        rst!Nachname = .LastName
        rst!Vorname = .FirstName
        rst!Straße = .BusinessAddressStreet
        rst!PLZ = .BusinessAddressPostalCode
        rst!Ort = .BusinessAddressCity
        rst!Bundesland = .BusinessAddressState
        rst!Telefon = .BusinessTelephoneNumber
        rst!Fax = .BusinessFaxNumber
        rst!Geburtsdatum = .Birthday
        rst!Email = .Email1Address
        rst.Update
    End With
```

```

Next intZ
MsgBox "Datentransfer erfolgreich beendet! " _
    & vbCrLf & "Es wurden " & intNeu & " Sätze angelegt, " _
    & vbCrLf & "Es wurden " & intUpd & " Sätze upgedatet!"
rst.Close
Set objKon = Nothing
Set objOutlApp = Nothing
Exit Sub

```

Fehler:

```

MsgBox "Es ist ein Fehler aufgetreten!"
End Sub

```

Listing 10.12 Den Kontaktordner von Outlook in eine Access-Tabelle übertragen

Die Prozedur ermittelt zuerst, wie viele Kontakte im Outlook-Kontaktordner angelegt sind. Abhängig davon wird eine Schleife durchlaufen, die alle Kontakte aus der lokalen Outlook-Datei (*Outlook.pst*) in die zentrale Access-Kontakttable überträgt. Dabei wird geprüft, ob sich der zu übertragende Satz bereits in der Access-Kontakttable befindet. Als Prüfkriterium wird das Feld »Account« herangezogen, das in der Variablen *str* zwischengespeichert wird. Mit der *FindFirst*-Methode wird der erste Datensatz in der Tabelle gesucht, der dem Kriterium entspricht. Wird kein zutreffender Satz gefunden, gibt die Eigenschaft *NoMatch* den Rückgabewert *True* zurück. In diesem Fall muss der komplette Kontakt aus Outlook in die Access-Kontakttable übernommen werden. Liefert die Eigenschaft *NoMatch* hingegen den Wert *False* zurück, entspricht das Suchkriterium *Account* einer bereits angelegten ID in der Access-Tabelle. Das Update wird über die Methode *Edit* eingeleitet, die den aktuellen Datensatz aus dem zu aktualisierenden *Recordset*-Objekt in den Kopierpuffer kopiert, damit er anschließend bearbeitet werden kann. Jetzt werden die einzelnen Informationen übertragen.

Das Neuanlegen eines Datensatzes geschieht über die *AddNew*-Methode. Gleich danach werden alle Kontaktfelder aus Outlook in den Kopierpuffer kopiert.

Egal, ob es sich um ein Update oder ein Neuanlegen handelt – erst die Methode *Update* sorgt dafür, dass der Inhalt des Kopierpuffers letztendlich in den Datensatz geschrieben wird. Für den Update-Fall sowie für den Fall des Neuanlegens werden zwei verschiedene Zähler verwendet, die am Ende des Datenaustauschs am Bildschirm ausgegeben werden. Um den Speicher am Ende wieder freizugeben, verwenden Sie das Schlüsselwort *Nothing*. Damit heben Sie die Verbindung der Objektvariablen zu den zugehörigen Objekten wieder auf.

10.3.3 Termine in den Terminkalender übertragen

Im nächsten Beispiel werden Sie Termine, die Sie in einer Access-Tabelle eintragen und verwalten, nach Outlook in Ihren Terminkalender übertragen. Als Ausgangstabelle liegt Ihnen eine Access-Tabelle mit dem Aufbau aus Abbildung 10.28 vor.

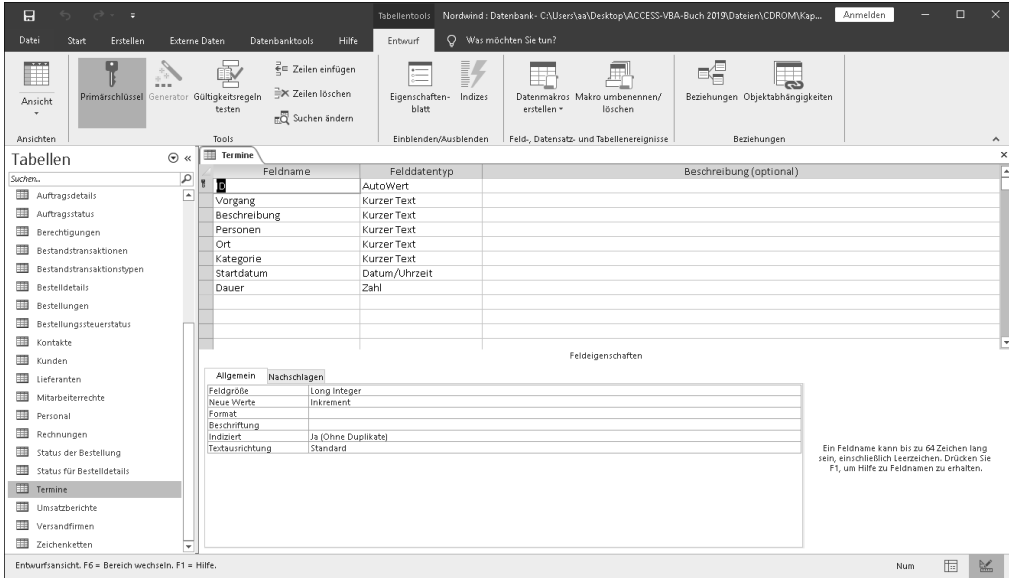


Abbildung 10.28 Die Termitabelle in Access

Speichern Sie diese Tabelle unter dem Namen *Termine*. Geben Sie danach ein paar Termine ein (siehe Abbildung 10.29).

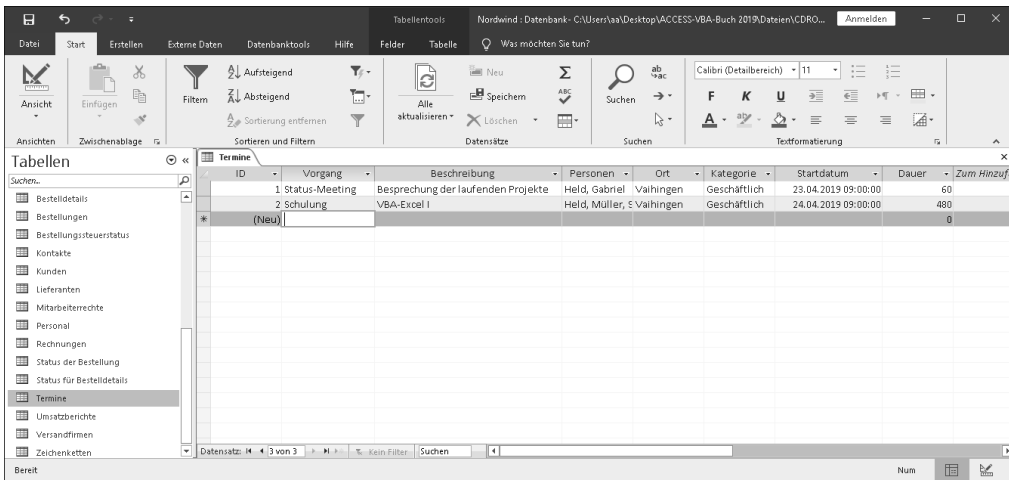


Abbildung 10.29 Zwei Termine zum Testen eintragen

Beim STARTDATUM erfassen Sie sowohl das Datum als auch die Uhrzeit des Termins. Im Datenfeld DAUER geben Sie die Dauer in Minuten ein.

Übertragen Sie jetzt Ihre Termine in den Terminkalender von Outlook. Starten Sie hierfür die Prozedur aus Listing 10.13.

```

Sub TermineVonAccessInOutlookUebertragen()
    Dim objOutlApp As Outlook.Application
    Dim objOutlTermin As Outlook.AppointmentItem
    Dim rst As New ADODB.Recordset
    Dim intZ As Integer

    Call OutlookKalenderBereinigen
    Set objOutlApp = CreateObject("Outlook.Application")
    rst.Open "Termine", CurrentProject.Connection

    On Error GoTo Fehler
    Do While Not rst.EOF
        Set objOutlTermin = objOutlApp.CreateItem(olAppointmentItem)
        With objOutlTermin
            .Subject = "[" & rst!Vorgang & "]"
            .Body = rst!Beschreibung
            .RequiredAttendees = rst!Personen
            .Location = rst!Ort
            .Categories = rst!Kategorie
            .Start = rst!StartDatum
            .Duration = rst!Dauer
            .ReminderMinutesBeforeStart = 10
            .ReminderPlaySound = True
            .ReminderSet = True
            .Save
            intZ = intZ + 1
            rst.MoveNext
        End With
    Loop
    rst.Close

    MsgBox "Es wurden " & i & " Termine übertragen!"

    Set objOutlTermin = Nothing
    Set objOutlApp = Nothing
Exit Sub

```


Fehler:

```
MsgBox "Fehler: " & Err.Description
```

```
End Sub
```

Listing 10.13 Termine in den Terminkalender von Outlook transportieren

Die Prozedur aus Listing 10.13 funktioniert im Prinzip ähnlich wie die Prozedur aus Listing 10.12. Der einzige Unterschied ist, dass Sie bei der Methode `CreateItem` eine andere Konstante angeben. Mit der Konstanten `olAppointmentItem` greifen Sie auf den Terminkalender von Outlook zu.



Hinweis: Termine nicht mehrfach anlegen

Beim Übertragen von Terminen haben wir im Prinzip das gleiche Problem wie schon beim Übertragen von Kontakten: Wir brauchen auch hier einen Mechanismus, der dafür sorgt, dass wir Termine nicht mehrfach anlegen, wenn die Übertragungsprozedur mehrfach ausgeführt wird.

Bei der Übertragung der Kontakte haben wir ein Outlook-Feld benutzt, um die ID des Access-Datensatzes mit zu übertragen. So konnten wir beim erneuten Übertragen der Kontakte feststellen, ob sich ein entsprechender Kontakt bereits im Adressbuch befand.

Bei diesem Beispiel wählen wir eine andere Vorgehensweise. Wir schreiben beim Feld `Subject` eine führende sowie eine schließende eckige Klammer. Bevor wir danach die Übertragungsprozedur erneut starten, läuft im Vorfeld eine Prozedur, die alle so gekennzeichneten Vorgänge aus dem Terminkalender löscht. Das bedeutet, dass die Termine immer neu komplett aus der Access-Tabelle in den Terminkalender von Outlook eingetragen werden. Werden demnach Termine aus der Access-Tabelle gelöscht, werden sie automatisch auch nicht mehr im Kalender geführt.

Die typischen Eigenschaften, die Sie beim Terminkalender einsetzen können, sehen Sie in Tabelle 10.2.

| Eigenschaft | Beschreibung |
|-------------------|--|
| Subject | Mit dieser Eigenschaft legen Sie den Betreff des Termins fest. |
| Body | Hier können Sie eine nähere Beschreibung hinterlegen, die im Textkörper ausgegeben wird. |
| RequiredAttendees | Listet die am Termin beteiligten Personen auf. |
| Location | Legt den Ort des Termins fest. |

Tabelle 10.2 Einige Angaben zum Terminkalender

| Eigenschaft | Beschreibung |
|----------------------------|--|
| Categories | Über diese Eigenschaft können Sie den Termin in einer Obergruppe zusammenfassen. |
| Start | Bestimmt das Startdatum sowie die Startzeit. |
| End | Über diese Eigenschaft bestimmen Sie das Ende eines Termins. |
| Duration | Gibt die Dauer der Besprechung bzw. des Termins in Minuten an. |
| ReminderMinutesBeforeStart | Diese Eigenschaft gibt die Zahl der Minuten an, die eine Erinnerung vor dem Beginn eines Termins auf dem Bildschirm angezeigt werden soll. |
| ReminderPlaySound | Über diese Eigenschaft können Sie die Erinnerungsmeldung auf dem Bildschirm zusätzlich mit einem Sound untermalen lassen. |
| ReminderSet | Schaltet die Erinnerungsfunktion ein. |

Tabelle 10.2 Einige Angaben zum Terminkalender (Forts.)

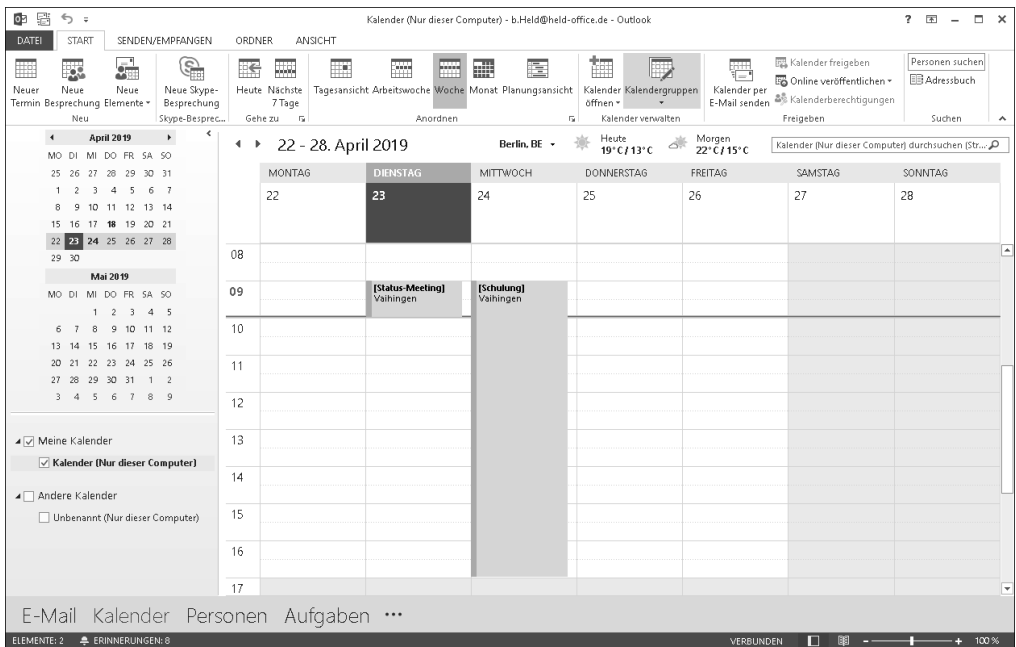


Abbildung 10.30 Die Termine wurden nach Outlook übertragen.

Bei einem erneuten Starten der Prozedur aus Listing 10.13 würden die Termine aus der Access-Tabelle doppelt angelegt. Daher muss innerhalb dieser Prozedur zu Beginn die Prozedur aus Listing 10.14 aufgerufen werden.

```
Sub OutlookKalenderBereinigen()  
    Dim objOutlApp As Outlook.Application  
    Dim objOutlTermin As Outlook.AppointmentItem  
    Dim objOutOrdner As Outlook.MAPIFolder  
    Dim objOutItems As Outlook.Items  
    Dim objOutMAPI As Outlook.NameSpace  
    Dim lngMax As Long  
    Dim lngZ As Long  
  
    On Error GoTo Fehler  
    Set objOutlApp = New Outlook.Application  
    Set objOutMAPI = objOutlApp.GetNamespace("MAPI")  
    Set objOutOrdner = objOutMAPI.GetDefaultFolder(olFolderCalendar)  
    Set objOutItems = objOutOrdner.Items  
  
    lngMax = objOutItems.Count  
    For lngZ = lngMax To 1 Step -1  
        If Left(objOutItems(lngZ).Subject, 1) = "[" Then  
            objOutItems(lngZ).Delete  
        End If  
    Next lngZ  
    Exit Sub  
  
Fehler:  
    MsgBox "Fehler: " & Err.Description  
  
End Sub
```

Listing 10.14 Bestimmte Termine aus dem Outlook-Kalender entfernen

Erstellen Sie im ersten Schritt über die Anweisung `New` einen Verweis auf die Outlook-Bibliothek. Danach greifen Sie auf das MAPI-Protokoll zu und holen sich mit der Methode `GetDefaultFolder` den Kalender von Outlook, den Sie über die Konstante `olFolderCalendar` ansprechen. In diesem Kalender befinden sich Einträge (`Items`). Diese zählen Sie noch vor Beginn der `For ... Next`-Schleife, die im Übrigen immer dann rückwärts durchlaufen werden muss, wenn es darum geht, Einträge zu löschen. Innerhalb der Schleife prüfen Sie das erste Zeichen des Feldes `Subject` (Titel des Termins) und löschen den Termin über die Methode `Delete`, wenn er mit einer eckigen Klammer als erstem Zeichen beginnt.

10.3.4 Aufgaben in die Aufgabenliste von Outlook übertragen

Nehmen wir an, Sie haben bisher Ihre Aufgaben in eine Access-Tabelle eingetragen und dort verwaltet. Die Verwaltung Ihrer Aufgaben lässt sich jedoch wesentlich besser über Outlook erledigen. Dabei können Sie auf integrierte Warnmeldungen zurückgreifen, die Outlook automatisch anzeigt, wenn eine Aufgabe noch zu erledigen ist.

Als kleine Vorarbeit für diese Aufgabe legen Sie eine Access-Tabelle an, die den in Abbildung 10.31 gezeigten Aufbau hat.

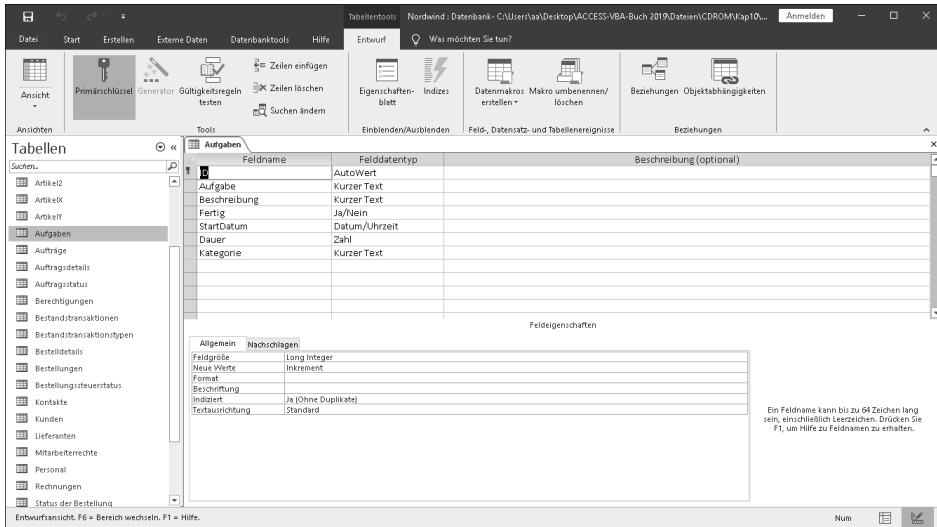


Abbildung 10.31 Die Ausgangstabelle für den Datentransfer

Definieren Sie das Datenfeld **FERTIG** mit dem Felddatentyp **JA/NEIN**. Speichern Sie die Tabelle unter dem Namen **Aufgaben**, und geben Sie ein paar Testdaten ein (siehe Abbildung 10.32).

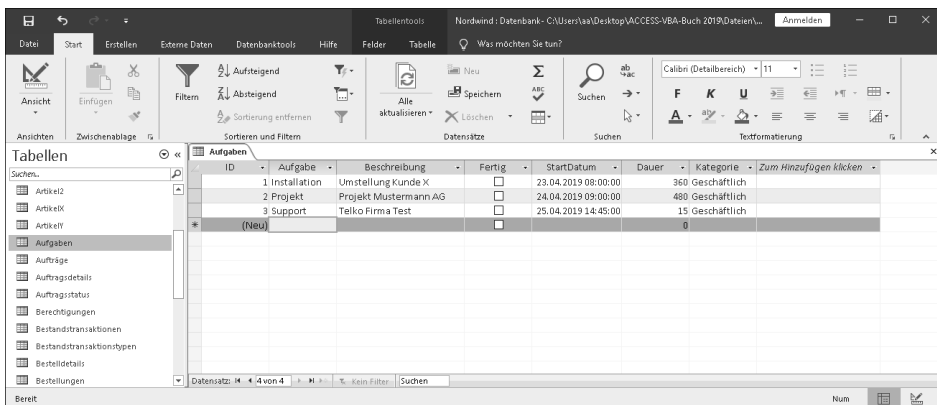


Abbildung 10.32 Die drei Aufgaben zum Testen

Erfassen Sie jetzt die Prozedur aus Listing 10.15, und starten Sie sie.

```
Sub AufgabenVonAccessNachOutlookUebertragen()  
    Dim objOutlApp As Outlook.Application  
    Dim objOutlAufg As Outlook.TaskItem  
    Dim rst As New ADODB.Recordset  
    Dim intZ As Integer  
  
    Set objOutlApp = CreateObject("Outlook.Application")  
    rst.Open "Aufgaben", CurrentProject.Connection  
  
    On Error GoTo Fehler  
    Do While Not rst.EOF  
        Set objOutlAufg = objOutlApp.CreateItem(olTaskItem)  
  
        With objOutlAufg  
            .Subject = rst!Aufgabe  
            .Body = rst!Beschreibung  
            .Complete = rst!Fertig  
            .StartDate = rst!StartDatum  
            .ActualWork = rst!Dauer  
            .Categories = rst!Kategorie  
  
            If .Complete = False Then  
                .ReminderSet = True  
                .ReminderTime = DateAdd("n", -5, rst!StartDatum)  
            End If  
  
            .Save  
            intZ = intZ + 1  
            rst.MoveNext  
        End With  
    Loop  
  
    rst.Close  
  
    MsgBox "Es wurden " & i & " Aufgaben übertragen!"  
    Set objOutlAufg = Nothing  
    Set objOutlApp = Nothing  
    Exit Sub
```

Fehler:

```
MsgBox "Fehler: " & Err.Description
```

```
End Sub
```

Listing 10.15 Aufgaben aus einer Tabelle in die Aufgabenliste von Outlook transferieren

Möchten Sie die Aufgabenliste von Outlook ansprechen, dann müssen Sie der Methode `CreateItem` die Konstante `olTaskItem` mitgeben. Damit haben Sie unter anderem Zugriff auf die Eigenschaften, die in Tabelle 10.3 aufgelistet sind.

| Eigenschaft | Beschreibung |
|----------------------------|--|
| Subject | Durch diese Eigenschaft legen Sie den Betreff der Aufgabe fest. |
| Body | Hier können Sie eine nähere Beschreibung hinterlegen, die im Textkörper ausgegeben wird. |
| Complete | Diese Eigenschaft setzen Sie auf den Wert <code>True</code> , wenn Sie die Aufgabe bereits abgeschlossen haben. |
| StartDate | Legt den Starttermin der Aufgabe fest. |
| DueDate | Bestimmt den Fälligkeitstermin Ihrer Aufgabe. |
| ActualWork | Über diese Eigenschaft können Sie einen Wert festlegen, der den Ist-Aufwand Ihrer Aufgabe in Minuten angibt. |
| Categories | Fasst den Termin in einer Obergruppe zusammen. |
| Complete | Dieser Eigenschaft geben Sie den Wert <code>True</code> , wenn die Aufgabe abgeschlossen ist. |
| ReminderMinutesBeforeStart | Gibt die Zahl der Minuten an, die eine Erinnerung vor dem Beginn einer Aufgabe auf dem Bildschirm angezeigt werden soll. |
| ReminderPlaySound | Mit dieser Eigenschaft können Sie die Erinnerungsmeldung auf dem Bildschirm zusätzlich von einem Sound untermalen lassen. |
| ReminderSet | Schaltet die Erinnerungsfunktion ein. |
| ReminderSoundFile | Über diese Eigenschaft können Sie eine andere Sounddatei angeben, die erklingen soll, wenn das Erinnerungsfenster Sie an eine Aufgabe erinnern soll. |

Tabelle 10.3 Einige Eigenschaften zur Aufgabenliste von Outlook

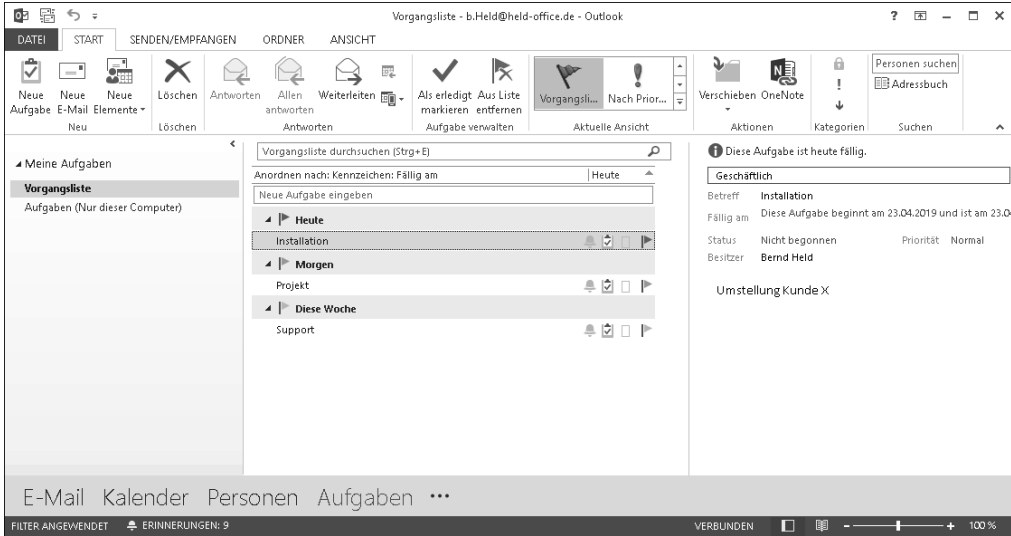


Abbildung 10.33 Die Aufgaben wurden automatisch angelegt.

10.3.5 E-Mails in einer Access-Datenbank speichern

Bei der folgenden Lösung werden alle E-Mails aus dem Posteingang in einer Access-Datenbank abgelegt. Dazu wird eine Access-Datentabelle hinterlegt, die so aufgebaut ist wie in Abbildung 10.34.

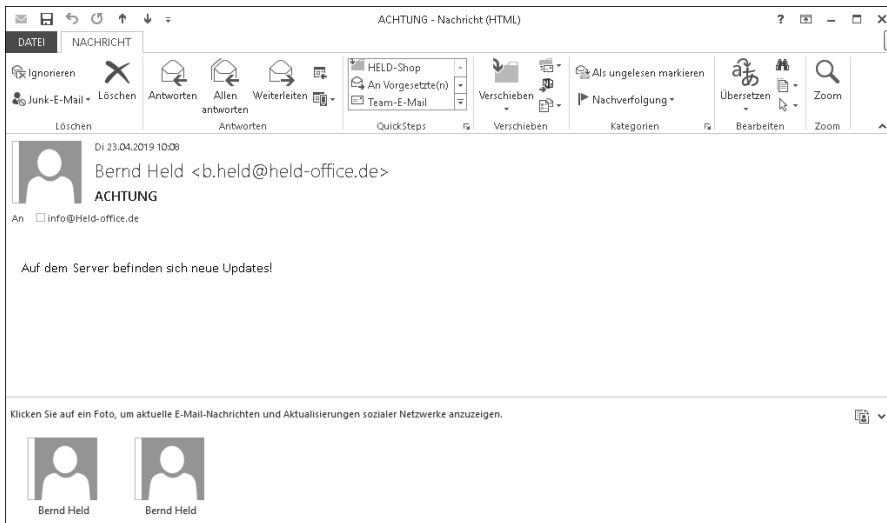


Abbildung 10.34 Der Aufbau der Tabelle für die E-Mail-Speicherung

Starten Sie die Prozedur aus Listing 10.16, um sämtliche E-Mails in diese Tabelle zu schreiben.

```

Sub EMailSichern()
    Dim objOutFold As Outlook.MAPIFolder
    Dim intGes As Integer
    Dim intZ As Integer
    Dim conn As New ADODB.Connection
    Dim rst As ADODB.Recordset

    On Error GoTo Fehler
    Set conn = CurrentProject.Connection
    Set rst = New ADODB.Recordset
    rst.Open "Posteingang", conn, adOpenKeyset, adLockOptimistic

    Set objOutFold = GetObject("", "Outlook.Application").GetNamespace _
        ("MAPI").GetDefaultFolder(olFolderInbox)

    intGes = objOutFold.Items.Count

    For intZ = 1 To intGes
        'Nur wenn es sich um ein Item vom Typ "MailItem" handelt.
        'Lesebestätigungen haben beispielsweise den Typ "Reportitem"
        'und werden hier nicht gespeichert
        If TypeName(objOutFold.Items(intZ)) = "MailItem" Then

            With objOutFold.Items(intZ)
                rst.AddNew
                rst!Titel = .Subject
                rst!Absender = .SenderName
                rst!Inhalt = .Body
                rst!Erhalten = Format(.ReceivedTime, "dd.mm.yyyy hh:mm")
                rst!Anhang = .Attachments.Count
                rst!Größe = .Size
                If Not .UnRead = -1 Then
                    rst!Gelesen = 1
                Else
                    rst!Gelesen = 0
                End If
                rst.Update
            End With
        End If
    Next intZ

    rst.Close

```



```
Set objOutFold = Nothing
Set rst = Nothing
Set conn = Nothing
Exit Sub
```

Fehler:

```
MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 10.16 Alle E-Mails werden in einer Access-Tabelle gespeichert.

Deklarieren Sie zu Beginn der Prozedur aus Listing 10.16 einige Objektvariablen, um den Posteingangskorb, die Verbindung und ein `Recordset` ansprechen zu können. Danach holen Sie sich über die Methode `GetDefaultFolder` den Posteingangskorb, der über die Konstante `olFolderInbox` angesprochen werden kann. Danach zählen Sie mit der Funktion `Count` die darin enthaltenen E-Mails. Anschließend arbeiten Sie alle E-Mails über eine `For ... Next`-Schleife nacheinander ab. Innerhalb der Schleife prüfen Sie, ob es sich um eine E-Mail handelt; wenn ja, dann liefert Ihnen die Funktion `TypeName` die Konstante `MailItem`. Lesebenachrichtigungen vom Typ `ReportItem` werden nicht dokumentiert. Handelt es sich also um eine E-Mail, dann wenden Sie die Methode `AddNew` an, um ein leeres `Recordset` anzulegen. Befüllen Sie danach die einzelnen Felder der Access-Tabelle, und weisen Sie ihnen die dazugehörigen E-Mail-Felder zu. Über die Methode `Update` wird der jeweilige Datensatz dauerhaft in der Datenbank angelegt.

10.3.6 Sammel-E-Mails versenden

Im folgenden Beispiel wird an alle Kontakte in der Access-Tabelle *Adressen* eine E-Mail versendet. Dazu wird das Feld »Mail« der Tabelle *Adressen* in der Datenbank *Kundendb.accdb* angesteuert. Wie das genau aussieht, entnehmen Sie der Prozedur aus Listing 10.17.

```
Sub MehrereEMailsVersenden()
    Dim objOutVerz As Object
    Dim objOutMail As Object
    Dim objOutApp As New Outlook.Application
    Dim conn As New ADODB.Connection
    Dim rst As ADODB.Recordset
    Dim strText As String

    Const Titel = "ACHTUNG"
    Const MailText = "Auf dem Server befinden sich neue Updates!"
```

```

On Error GoTo Fehler
Set objOutVerz = _
objOutApp.GetNamespace("MAPI").GetDefaultFolder(olFolderContacts)

Set conn = CurrentProject.Connection
strText = "SELECT Mail from Adressen"
Set rst = New ADODB.Recordset
rst.Open strText, conn, adOpenKeyset, adLockOptimistic

Do Until rst.EOF
    If Not IsNull(rst!Mail) Then
        Set objOutMail = CreateItem(olMailItem)
        With objOutMail
            .Subject = Titel
            .Body = MailText
            .To = rst!Mail
            .Send
        End With
    End If
    rst.MoveNext
Loop

Set objOutVerz = Nothing
Set objOutMail = Nothing
Set rst = Nothing
Set conn = Nothing
Exit Sub

```

Fehler:

```

MsgBox Err.Number & " " & Err.Description
End Sub

```

Listing 10.17 Eine Sammel-E-Mail an alle Kontakte versenden

Deklariieren Sie zu Beginn der Prozedur aus Listing 10.17 einige Objektvariablen, um die Verbindung und ein Recordset ansprechen zu können. Die SQL-Anweisung können Sie der besseren Übersicht halber in einem String bekannt geben und später der Methode `Open` übergeben.

Nach dem Öffnen des Recordset stehen darin alle E-Mail-Adressen. Über eine `Do Until`-Schleife werden diese dann nacheinander abgearbeitet. In der Schleife prüfen Sie, ob das Feld »Mail« überhaupt gefüllt ist. Wenn ja, dann wenden Sie die Methode `CreateObject` mit der Konstanten `olMailItem` an, um eine noch leere E-Mail zu erstellen.

Übertragen Sie danach die einzelnen Felder wie den Titel und den Mail-Text über die Eigenschaften `Subject` und `Body` in die E-Mail. Den Adressaten der E-Mail können Sie über die Eigenschaft `To` ansprechen. Das eigentliche Versenden der E-Mail erledigt die Methode `Send`.

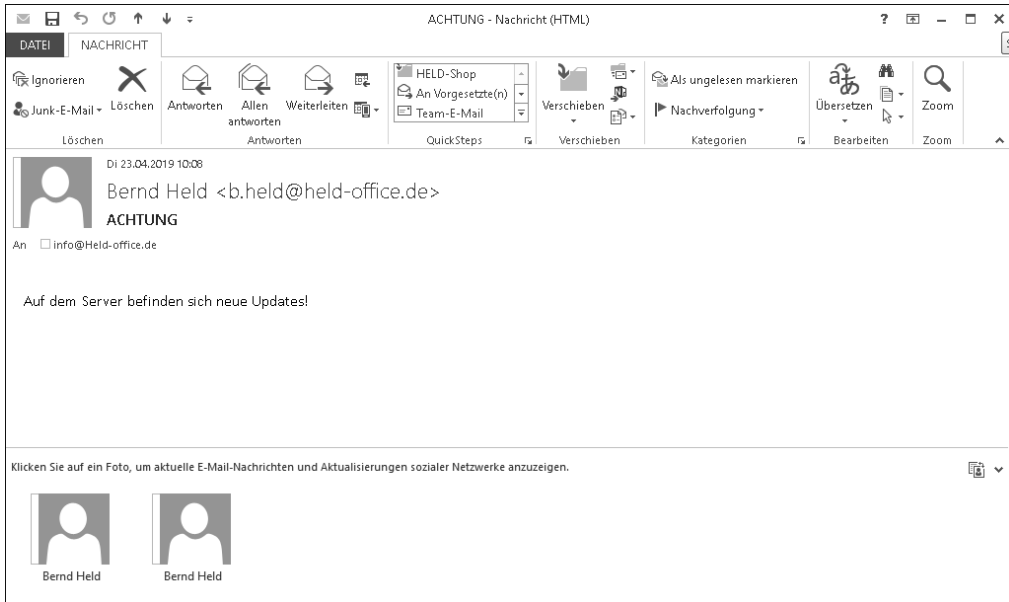


Abbildung 10.35 Diese E-Mail wurde an alle Kontakte versendet.

10.4 Access im Duett mit Excel

Beim Zusammenspiel von Access und Excel haben Sie sowohl die Möglichkeit, Daten nach Excel zu exportieren, als auch die Option, Daten von Excel zu empfangen. Gerade wenn Sie eine Access-Tabelle haben und Ihr Kunde möglicherweise kein Access zur Verfügung hat, können Sie relativ leicht Ihre Access-Tabelle in eine Excel-Datei umwandeln.

10.4.1 Access-Tabelle in eine Excel-Tabelle umwandeln

Im Beispiel aus Listing 10.18 wird die Access-Tabelle `Artikel` in eine Excel-Arbeitsmappe `Artikel.xls` umgewandelt. Dafür müssen Sie lediglich die Methode `OutputTo` einsetzen und das gewünschte Format festlegen.

```
Sub TabelleNachExcelTransferieren()
```

```
DoCmd.OutputTo acOutputTable, "Artikel", _
```

```
acFormatXLS, Application.CurrentProject.Path & "\Artikel.xls", True
```

End Sub

Listing 10.18 Access-Tabelle in eine Excel-Tabelle transferieren (Methode 1)

Das Ergebnis aus diesen wenigen Zeilen Code kann sich sehen lassen (siehe Abbildung 10.36).

| | A | B | C | D | E | F | G | H |
|----|--------------------------|----|-------------|--|--------------|----------------|-------------|----------------|
| 1 | Lieferantennummern | ID | Produktcode | Artikelname | Beschreibung | Standardkosten | Listenpreis | Mindestbestand |
| 2 | Lieferant D | 1 | NWTB-1 | Northwind Traders Chai | | 13,50 € | 18,00 € | 10 |
| 3 | Lieferant J | 3 | NWTCO-3 | Northwind Traders Syrup | | 7,50 € | 10,00 € | 25 |
| 4 | Lieferant J | 4 | NWTCO-4 | Northwind Traders Cajun Seasoning | | 16,50 € | 22,00 € | 10 |
| 5 | Lieferant J | 5 | NWTO-5 | Northwind Traders Olive Oil | | 16,01 € | 21,35 € | 10 |
| 6 | Lieferant B; Lieferant F | 6 | NWTJP-6 | Northwind Traders Boysenberry Spread | | 18,75 € | 25,00 € | 25 |
| 7 | Lieferant B | 7 | NWTDFN-7 | Northwind Traders Dried Pears | | 22,50 € | 30,00 € | 10 |
| 8 | Lieferant H | 8 | NWTS-8 | Northwind Traders Curry Sauce | | 30,00 € | 40,00 € | 10 |
| 9 | Lieferant B; Lieferant F | 14 | NWTDFN-14 | Northwind Traders Walnuts | | 17,44 € | 23,25 € | 10 |
| 10 | Lieferant F | 17 | NWTFCV-17 | Northwind Traders Fruit Cocktail | | 29,25 € | 39,00 € | 10 |
| 11 | Lieferant A | 19 | NWTBGM-19 | Northwind Traders Chocolate Biscuits Mix | | 6,90 € | 9,20 € | 5 |
| 12 | Lieferant B; Lieferant F | 20 | NWTJP-6 | Northwind Traders Marmalade | | 60,75 € | 81,00 € | 10 |
| 13 | Lieferant A | 21 | NWTBGM-21 | Northwind Traders Scones | | 7,50 € | 10,00 € | 5 |
| 14 | Lieferant D | 34 | NWTB-34 | Northwind Traders Beer | | 10,50 € | 14,00 € | 15 |
| 15 | Lieferant G | 40 | NWTQM-40 | Northwind Traders Crab Meat | | 13,80 € | 18,40 € | 30 |
| 16 | Lieferant F | 41 | NWTSO-41 | Northwind Traders Clam Chowder | | 7,24 € | 9,65 € | 10 |
| 17 | Lieferant C; Lieferant D | 43 | NWTB-43 | Northwind Traders Coffee | | 34,50 € | 46,00 € | 25 |
| 18 | Lieferant J | 48 | NWTCA-48 | Northwind Traders Chocolate | | 9,56 € | 12,75 € | 25 |
| 19 | Lieferant B | 51 | NWTDFN-51 | Northwind Traders Dried Apples | | 39,75 € | 53,00 € | 10 |
| 20 | Lieferant A | 52 | NWTG-52 | Northwind Traders Long Grain Rice | | 5,25 € | 7,00 € | 25 |
| 21 | Lieferant A | 56 | NWTP-56 | Northwind Traders Gnocchi | | 28,50 € | 38,00 € | 30 |

Abbildung 10.36 Die Excel-Datei ist schon vorformatiert.

Sollte die Exportdatei *Artikel.xls* noch nicht vorliegen, dann erstellt Access diese Datei selbstverständlich automatisch für Sie.

Eine weitere Möglichkeit, Daten von Access nach Excel zu transportieren, bietet die Methode *TransferSpreadsheet* aus Listing 10.19.

```
Sub TabelleNachExcelTransferieren02()
```

```
DoCmd.TransferSpreadsheet acExport, acSpreadsheetTypeExcel12, _
    "Artikel", Application.CurrentProject.Path & "\Artikel2.xls", True
```

End Sub

Listing 10.19 Access-Tabelle in eine Excel-Tabelle transferieren (Methode 2)

Um eine Access-Datentabelle in eine Excel-Arbeitsmappe oder in eine Lotus-Tabelle zu übertragen, setzen Sie die Methode `TransferSpreadsheet` ein.

Die Methode `TransferSpreadsheet` hat folgende Syntax:

`TransferSpreadsheet`(Transfertyp, Dateiformat, Tabellename, Dateiname, BesitztFeldnamen, Bereich)

Im Argument `Transfertyp` geben Sie an, welchen Transfer Sie genau durchführen möchten. Sie haben dabei die Auswahl zwischen dem Export (`acExport`), dem Import (`acImport`) oder einer Verknüpfung (`acLink`).

Im Argument `Dateiformat` geben Sie an, in welche Excel-Version (bzw. Lotus-Version) Sie die Access-Tabelle exportieren möchten.

Im darauffolgenden Argument `Tabellename` geben Sie den Namen der zu exportierenden Access-Tabelle an. Über das Argument `Dateiname` geben Sie das Ziel für den Datenexport bekannt. Dabei muss die angegebene Datenquelle nicht einmal existieren. Access legt sie neu für Sie an.

Beim Argument `BesitztFeldnamen` verwenden Sie den Wert `True`, um die erste Zeile der Kalkulationstabelle beim Importieren, Exportieren oder Verknüpfen zur Angabe der Feldnamen zu verwenden. Geben Sie hingegen den Wert `False` an, wenn die erste Zeile als normale Datenzeile gelten soll. Wenn Sie dieses Argument nicht angeben, wird der Standardwert `False` verwendet.

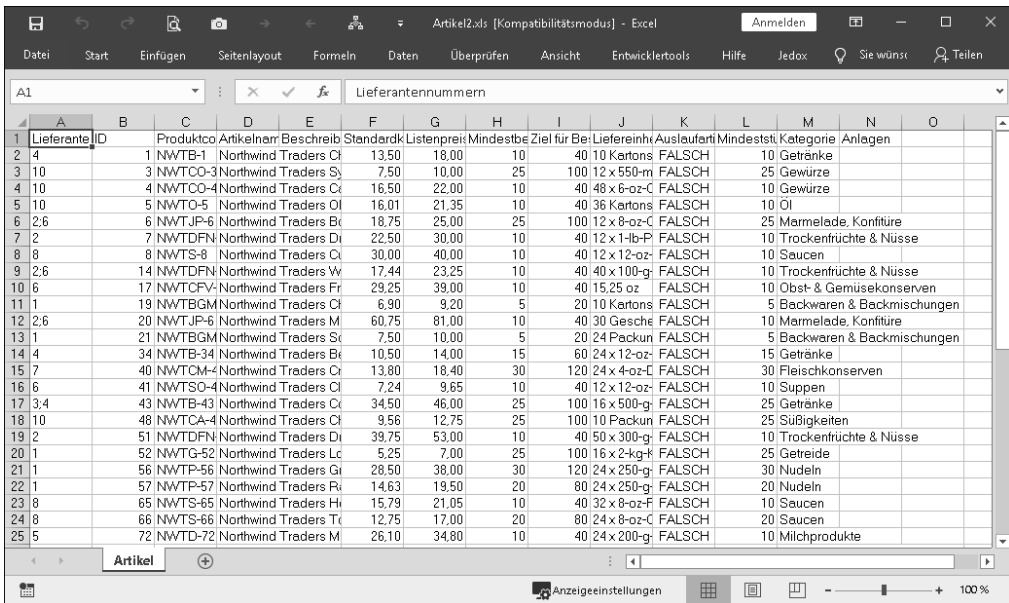


Abbildung 10.37 Das Ergebnis der Methode »TransferSpreadsheet«

Das Argument `Bereich` gilt nur für Importoperationen und darf beim Export nicht angegeben werden. Beim Import werden standardmäßig immer alle Datensätze importiert, sofern dieses Argument nicht gesetzt wird.

Bei dieser Methode werden die Verknüpfungen zwischen den einzelnen Datentabellen leider nicht umgesetzt, sodass beispielsweise im Feld `LIEFERANTEN-ID` eine ID statt des Lieferantennamens angezeigt wird.

10.4.2 Excel-Daten in eine Access-Tabelle transferieren

Mit der gerade vorgestellten Lösung auf Basis der Methode `TransferSpreadsheet` können Sie auch Excel-Tabellen in eine Access-Datenbank importieren.

Die Prozedur für diese Aufgabe sehen Sie in Listing 10.20.

```
Sub TabelleVonExcelImportieren()
```

```
    DoCmd.TransferSpreadsheet acImport, _
        acSpreadsheetTypeExcel12, _
        "ArtikelY", Application.CurrentProject.Path & "\Artikel.xls", True
```

```
End Sub
```

Listing 10.20 Excel-Tabelle in eine Access-Tabelle überführen

Übergeben Sie der Methode `TransferSpreadsheet` die Konstante `acImport`, um mitzuteilen, dass Access einen Import vornehmen soll. Als zweite Konstante geben Sie die Excel-Version an, in der die Tabelle vorliegt.

Dabei haben Sie folgende Möglichkeiten:

- ▶ `acSpreadsheetTypeExcel19` (Excel 2000)
- ▶ `acSpreadsheetTypeExcel10` (Excel 2002)
- ▶ `acSpreadsheetTypeExcel11` (Excel 2003)
- ▶ `acSpreadsheetTypeExcel12` (Excel 2007 und Excel 2010)
- ▶ `acSpreadsheetTypeExcel12Xml` (für alle nachfolgenden Versionen: Microsoft Excel 2010/2013/2016/2019/2021; XML-Dateiformat `.xlsx`, `.xlsm`, `.xlsb`)

Wollen Sie nur einen Teil der Tabelle aus Excel in Ihre Access-Tabelle übernehmen, so geben Sie im letzten Argument der Methode `TransferSpreadsheet` den Bereich an, der übertragen werden soll.

In der Prozedur aus Listing 10.21 werden die ersten zehn Zeilen und die ersten vier Spalten der Tabelle übertragen.

```
Sub TeilVonTabelleVonExcelImportieren()
    DoCmd.TransferSpreadsheet acImport, _
        acSpreadsheetTypeExcel12, "ArtikelY", Application.CurrentProject.Path & _
            "\Artikel.xls", True, "A1:D10"
End Sub
```

Listing 10.21 Einen Teil einer Excel-Tabelle in eine Access-Tabelle überführen

Bei den letzten beiden Prozeduren ist das Problem, dass verknüpfte Felder nicht automatisch angepasst werden und daher nicht ordentlich übertragen werden. Daher eignet sich diese Methode nur dann, wenn es sich um nicht verknüpfte Datentabellen handelt.

10.4.3 Automatisches Anlegen einer Access-Tabelle mit anschließendem Import

In der nächsten Aufgabe erzeugen Sie eine neue Access-Tabelle, zapfen dann eine Excel-Tabelle an und übertragen die Datensätze. Bei dieser Aufgabe wurde in Excel eine Anrufliste angelegt. Die Felder der Liste sehen Sie in Abbildung 10.38.

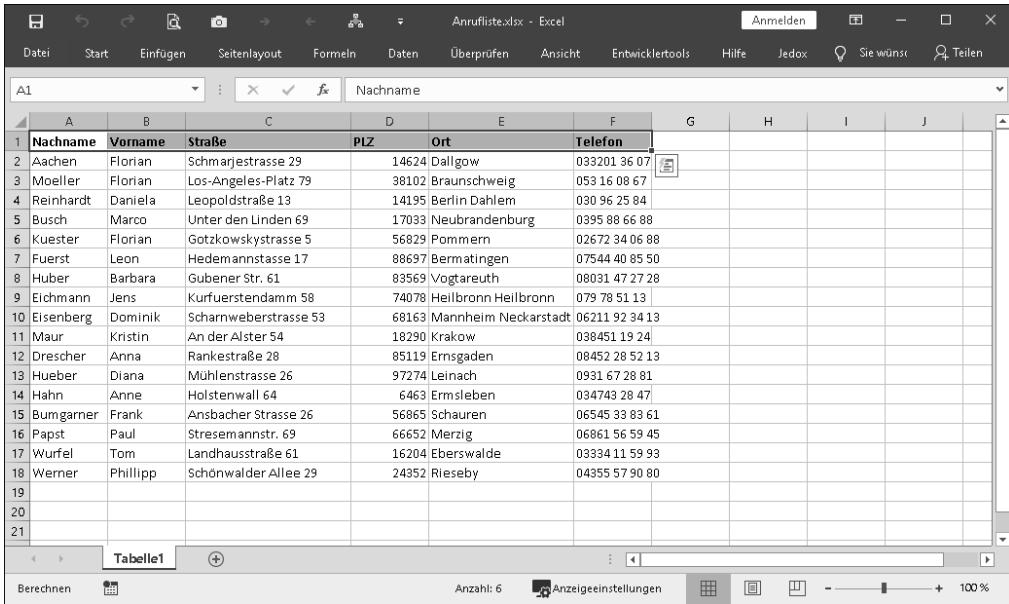


Abbildung 10.38 Die Ausgangstabelle in Excel

Jetzt sollen die Daten der Kunden in die neu zu erstellende Access-Tabelle *Anrufe* übertragen werden, was Sie in Listing 10.22 sehen.

Um diese Aufgabe zu lösen, brauchen Sie eine zusätzliche Bibliothek mit dem Namen *Microsoft ADO Ext. 6.0 für DD Land Security*. Diese Bibliothek benötigen Sie, um eine neue Tabelle mit vordefinierten Feldern anzulegen. Dazu gehen Sie in die Entwicklungsumgebung und binden unter EXTRAS • VERWEISE diese Bibliothek ein.

```

Sub TabelleErstellenUndExcelEinlesen()
    Dim con As ADODB.Connection
    Dim cat As ADOX.Catalog
    Dim tbiTabInfo As ADOX.Table
    Dim objXlApp As Object
    Dim objXlTab As Object
    Dim rst As ADODB.Recordset
    Dim lngZeile As Long
    Dim lngZeileMax As Long

    Set cat = New ADOX.Catalog
    cat.ActiveConnection = CurrentProject.Connection

    On Error GoTo Fehler
    Set tbiTabInfo = New ADOX.Table

    With tbiTabInfo
        .Name = "Anrufe"
        Set .ParentCatalog = cat
        With .Columns
            .Append "Nachname", adVarChar
            .Append "Vorname", adVarChar
            .Append "Straße", adVarChar
            .Append "PLZ", adVarChar
            .Append "Ort", adVarChar
            .Append "Telefon", adVarChar, 20
        End With
    End With

    cat.Tables.Append tbiTabInfo
    Set cat = Nothing
    Set con = CurrentProject.Connection
    Set rst = New ADODB.Recordset

    rst.Open "Anrufe", con, adOpenKeyset, adLockOptimistic

    On Error Resume Next
    Set objXlApp = GetObject(, "Excel.Application")

```



```
If Err.Number = 429 Then
    Set objXlApp = CreateObject("Excel.Application")
    Err.Number = 0
End If

objXlApp.Visible = True
Set objXlTab = objXlApp.workbooks.Open(Application.CurrentProject.Path & _
"\Anrufliste.xlsx")

With objXlTab.Worksheets(1)
    lngZeileMax = .usedrange.rows.Count

    For lngZeile = 2 To lngZeileMax

        rst.AddNew
        rst!Nachname = .cells(lngZeile, 1).Value
        rst!Vorname = .cells(lngZeile, 2).Value
        rst!Straße = .cells(lngZeile, 3).Value
        rst!PLZ = .cells(lngZeile, 4).Value
        rst!Ort = .cells(lngZeile, 5).Value
        rst!Telefon = .cells(lngZeile, 6).Value
        rst.Update

    Next lngZeile

End With

rst.Close
objXlTab.Close

Set rst = Nothing
Set con = Nothing
Exit Sub

Fehler:
    MsgBox "Fehler: " & Err.Description
End Sub
```

Listing 10.22 Automatische Tabellenerstellung und -befüllung aus einer Excel-Tabelle

Legen Sie zuerst eine Objektvariable vom Typ `Catalog` an. Über dieses Auflistungsobjekt können Sie auf Objekte wie Tabellen und Abfragen zugreifen. Stellen Sie über die Eigenschaft `ActiveConnection` die Verbindung zu Ihrem Provider und zu Ihrer Daten-

quelle her. Erzeugen Sie danach ein neues Table-Objekt, und definieren Sie die gewünschten Datenfelder. Wie das genau funktioniert, können Sie in Kapitel 5, »Tabellen programmieren«, nachlesen.

Öffnen Sie danach die Tabelle *Anrufe* mit der Methode *Open*. Nun ist die Access-Tabelle bereit, die Excel-Daten zu empfangen.

Im ersten Schritt erzeugen Sie ein Excel-Objekt, mit dem Sie Zugriff auf die Excel-VBA-Methoden und Eigenschaften bekommen. Dazu setzen Sie die Methode *CreateObject* ein und übergeben ihr das Applikationsobjekt von Excel.

Natürlich ist es möglich, dass Sie die Anwendung Excel bereits geöffnet haben. In diesem Fall brauchen Sie kein neues Excel-Objekt anzulegen. Hier reicht es, wenn Sie über die Methode *GetObject* dieses geöffnete Objekt in den Vordergrund bringen. Wenn Sie versuchen, die Methode *GetObject* auf ein Objekt anzuwenden, das augenblicklich nicht zur Verfügung steht, weil es beispielsweise geschlossen ist, wird Ihnen die Fehlernummer 429 zurückgemeldet.

Nach Erzeugung des neuen Excel-Objekts bzw. vor dessen Aktivierung setzen Sie die Eigenschaft *Visible* ein, um das Excel-Programmfenster anzuzeigen.

Definieren Sie nun ein weiteres Excel-Objekt, das die zu öffnende Arbeitsmappe darstellt. Die Arbeitsmappe wird über die Methode *Open* geöffnet, bei der Sie den Namen und den Pfad der Excel-Mappe angeben müssen.

Die Methode *Open* hat dabei folgende Syntax:

```
Workbooks.Open(FileName, UpdateLinks, ReadOnly, Format, Password, WriteResPassword, IgnoreReadOnlyRecommended, Origin, Delimiter, _
    Editable, Notify, Converter, AddToMRU)
```

Besonders wichtig ist das Argument *UpdateLinks*. Sicher haben Sie schon einmal beim Öffnen einer Arbeitsmappe die Meldung mit der Frage erhalten, ob Sie die Verknüpfungen in der Arbeitsmappe aktualisieren möchten oder nicht. Diese Abfrage können Sie unterdrücken, indem Sie ein entsprechendes Argument 0 bis 3 einsetzen. Die nachfolgende Tabelle 10.4 zeigt die Bedeutung der verschiedenen Werte.

| Konstante | Bedeutung |
|-----------|---|
| 0 | keine Aktualisierung von Bezügen |
| 1 | Aktualisierung von externen Bezügen, jedoch nicht von Fernbezügen |
| 2 | Aktualisierung von Fernbezügen, jedoch nicht von externen Bezügen |
| 3 | Aktualisierung von externen Bezügen und Fernbezügen |

Tabelle 10.4 Die Aktualisierungskonstante der Methode »Open«

Alle weiteren Argumente können Sie jederzeit in der Excel-VBA-Onlinehilfe nachlesen.

Nachdem die Excel-Mappe *Anrufe.xlsx* geöffnet ist, prüfen Sie, ob die richtige Zelle markiert ist.

Bevor Sie die Startzelle über die Eigenschaft *Range* festlegen, sollten Sie sicher sein, dass Sie auch auf dem gewünschten Tabellenblatt sind. Setzen Sie dazu das Auflistungsobjekt *Worksheets* ein. Geben Sie bei diesem Objekt den Index der Tabelle an, auf die Sie zugreifen möchten. Da die Mappe nur aus einer Tabelle besteht, haben Sie es hier leicht.

Starten Sie danach eine *For . . . Next*-Schleife, die so lange durchlaufen wird, bis alle Zeilen der Tabelle abgearbeitet wurden. Um die einzelnen Zellen einer Excel-Tabelle anzusprechen, setzen Sie die Eigenschaft *Cells* ein. Diese Eigenschaft benötigt zwei Argumente: Das erste Argument steht für die aktuelle Zeile, das zweite symbolisiert die Spalte.

Über die Methode *AddNew* fügen Sie einen neuen Satz in der Access-Tabelle *Anrufe* ein. Danach füllen Sie die einzelnen Datenfelder mit dem Inhalt der Zellen. Erst durch den Einsatz der Methode *Update* speichern Sie diese neuen Informationen dauerhaft in der Access-Tabelle.

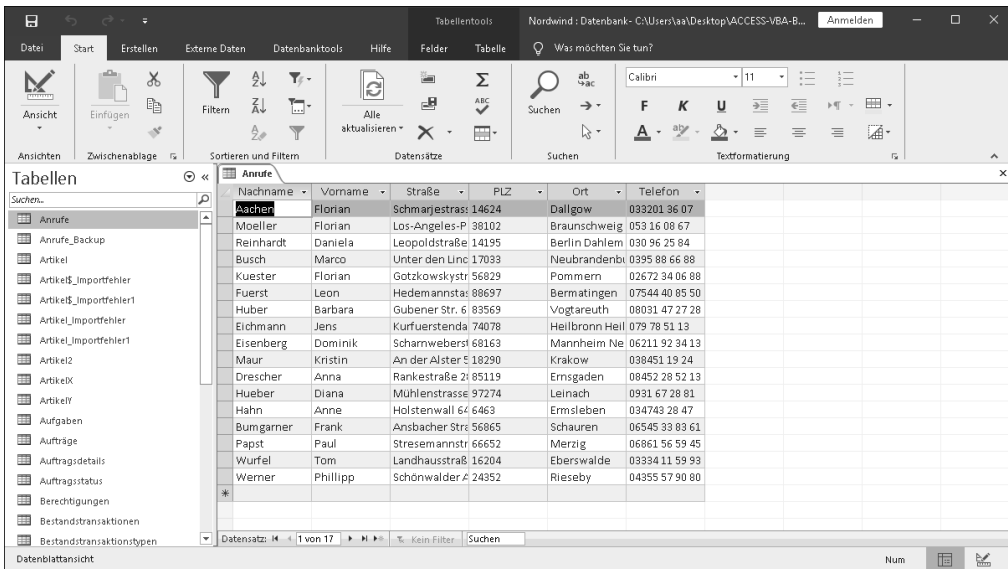


Abbildung 10.39 Die Daten aus Excel wurden direkt in eine neue Access-Tabelle transferiert.

Nach der Verarbeitung schließen Sie sowohl die Excel-Mappe als auch die Access-Tabelle über die Methode *Close*. Heben Sie am Ende der Prozedur die Objektverweise

wieder auf, indem Sie die Anweisung Set Objektvariable = Nothing einsetzen. Damit geben Sie den reservierten Platz im Arbeitsspeicher wieder frei.

10.4.4 Aus Excel auf Access zugreifen

Das folgende Beispiel demonstriert den Zugriff auf eine Access-Tabelle aus einer Excel-Tabelle heraus. Dabei werden bestimmte Daten aus einer Access-Tabelle (*Personal*) von Excel aus abgefragt und anschließend in eine Excel-Tabelle eingefügt.

Für diese Aufgabe können Sie die Zugriffsmethode ADO (ActiveX Data Objects) verwenden. Dazu wechseln Sie über die Tastenkombination **[Alt] + [F11]** in die Entwicklungsumgebung von Microsoft Excel. Wählen Sie aus dem Menü EXTRAS den Befehl VERWEISE.

Im Listenfeld VERFÜGBARE VERWEISE (siehe Abbildung 10.40) aktivieren Sie die Bibliothek MICROSOFT ACTIVEX DATA OBJECTS 6.1 LIBRARY. Bestätigen Sie mit OK.

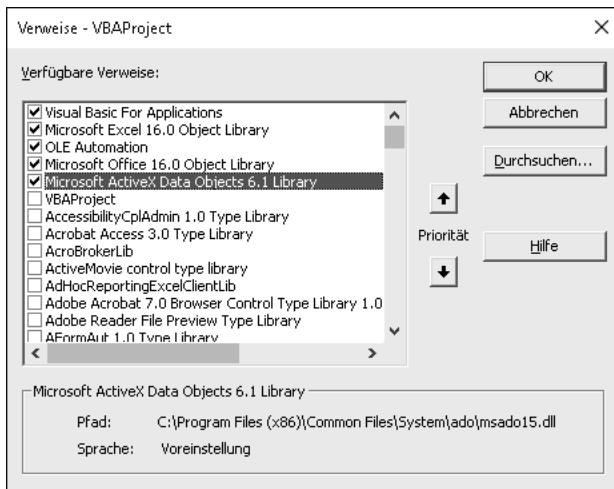


Abbildung 10.40 Die ADO-Bibliothek einbinden

Starten Sie die Prozedur aus Listing 10.23, um die Daten aus Access nach Excel zu holen.

```
Sub AccessTabelleAbfragen()
    Dim strDB As String
    Dim objCon As ADODB.Connection
    Dim rst As ADODB.Recordset
    Dim i As Integer

    strDB = ThisWorkbook.Path & "\Nordwind.accdb"
```

```
Set objCon = New ADODB.Connection
objCon.Open "Provider=Microsoft.ACE.OLEDB.12.0; Data Source=" & strDB

Set rst = New ADODB.Recordset
rst.Open "SELECT * FROM Personal WHERE Ort='Seattle'", objCon, _
        adOpenKeyset, adLockOptimistic

With Tabelle1
    .UsedRange.Clear

    'Überschriften holen
    For i = 0 To rst.Fields.Count - 1
        .Range("A1").Offset(0, i).Value = rst.Fields(i).Name
    Next i

    .Rows(1).Font.Bold = True
    .Range("A2").CopyFromRecordset rst

End With

Set rst = Nothing

objCon.Close
Set objCon = Nothing

End Sub
```

Listing 10.23 Überschriften und Daten aus einer Access-Tabelle holen

Geben Sie zu Beginn der Prozedur aus Listing 10.23 an, wo die Access-Datenbank liegt. Danach geben Sie den Provider sowie den Namen und den Pfad der Datenbank über die Methode `Open` bekannt. Legen Sie dann über die Anweisung `Set` ein neues `Recordset` an. Mit der Methode `Open` übergeben Sie eine SQL-Anweisung mit einer zusätzlichen `WHERE`-Bedingung. Jetzt befinden sich alle abgefragten Daten im `Recordset` `rst`.

Nachdem Sie die Zieltabelle über die Methode `Clear` gelöscht haben, fragen Sie über eine `For ... Next`-Schleife zunächst die Feldnamen über die Eigenschaft `Name` ab. Danach übertragen Sie das komplette `Recordset` mit der Methode `CopyFromRecordset` in die Tabelle.

| ID | Firma | Nachname | Vorname | E-Mail-Adresse | Position | Telefon (gesamt) | Telefon privat | Mobiltelefon | Faxnummer | Straße | Ort |
|----|----------------|--------------|---------|----------------|--------------|------------------|-----------------|--------------|-----------|-----------------------------|---------|
| 1 | Northwind Tr | Freehafer | Nancy | nancy@north | Vertriebsmit | (1 23) 55 50 | :(1 23) 5 55 01 | 02 | | (1 23) 5 55 0; 123 1st Aven | Seattle |
| 3 | 5 Northwind Tr | Thorpe | Steven | steven@north | Vertriebsmar | (1 23) 55 50 | :(1 23) 5 55 01 | 02 | | (1 23) 5 55 0; 123 5th Aven | Seattle |
| 4 | 7 Northwind Tr | Zare | Robert | robert@north | Vertriebsmit | (1 23) 55 50 | :(1 23) 5 55 01 | 02 | | (1 23) 5 55 0; 123 7th Aven | Seattle |
| 5 | 9 Northwind Tr | Hellung-Lars | Anne | anne@north | Vertriebsmit | (1 23) 55 50 | :(1 23) 5 55 01 | 02 | | (1 23) 5 55 0; 123 9th Aven | Seattle |

Abbildung 10.41 Eine Tabelle mit Namen und Adressen aller Angestellten aus Seattle wurde nach Excel transferiert.

10.4.5 Suchen, Anlegen, Ändern und Löschen

Bei den folgenden Prozeduren werden Kernfunktionen an der Datenbank *Kundendb.accdb* von Excel aus im Hintergrund durchgeführt. Diese Lösungen sind dann sehr gut, wenn man selbst keine Access-Installation zur Verfügung hat und trotzdem über den ADO-Treiber Access als Runtime-Version nutzen möchte.

Bei den folgenden Kernfunktionen gehen Sie von der Excel-Tabelle in Abbildung 10.42 aus.

| ID | Kunden-Nr | Name, Vorname | Straße | PLZ | Ort |
|----|-----------|---------------|--------|-----|-----|
| | | | | | |

Suche nach Kunden-Nr.

Neuanlage / Update

Löschen anhand der ID

Abbildung 10.42 Von dieser Tabelle aus wird Access gesteuert.

Die dazugehörige Access-Datenbank sehen Sie in Abbildung 10.43.

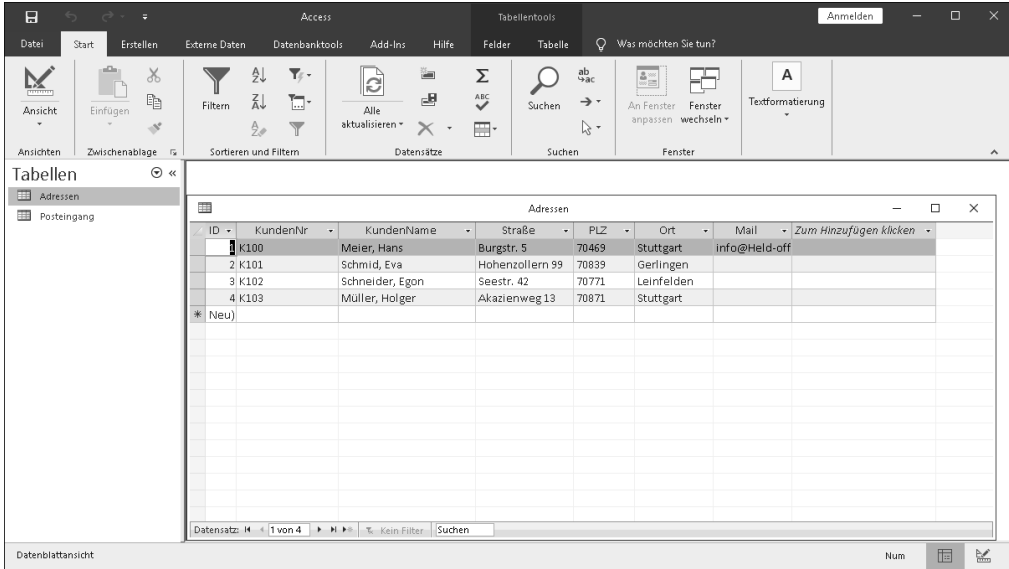


Abbildung 10.43 In dieser Access-Tabelle werden die Adressen verwaltet.

Daten suchen

Bei der ersten Kernfunktion sollen über eine Eingabe der »KundenNr« in Zelle B2 die dazugehörigen Daten aus der Access-Datenbank geholt werden. Erfassen Sie in dieser Zelle beispielsweise die Kundennummer K100, und starten Sie danach die Prozedur aus Listing 10.24.

Sub Suche()

```

Dim strDB As String
Dim strTab As String
Dim objCon As ADODB.Connection
Dim rst As ADODB.Recordset

strDB = ThisWorkbook.Path & "\Kundendb.accdb"
strTab = "Adressen"

Set objCon = New ADODB.Connection
objCon.Open "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" & strDB

With tbl_Daten
    Set rst = New ADODB.Recordset
    rst.Open "SELECT * FROM " & strTab & " WHERE KundenNr='" & _
        .Range("B2").Value & "'"; objCon, adOpenKeyset, adLockOptimistic

```

```

If rst.RecordCount <> 0 Then
    .Range("B1").Value = rst.Fields("ID").Value
    .Range("B3").Value = rst.Fields("KundenName").Value
    .Range("B4").Value = rst.Fields("Straße").Value
    .Range("B5").Value = rst.Fields("PLZ").Value
    .Range("B6").Value = rst.Fields("Ort").Value
Else
    MsgBox "Es konnte kein Kunde mit der Nummer " & _
    .Range("B2").Value & " gefunden werden!", vbInformation
End If

End With

```

End Sub

Listing 10.24 Die Suche in der Access-Datenbank erfolgt über die Eingabe der Kundennummer.

Geben Sie zu Beginn der Prozedur aus Listing 10.24 bekannt, über welchen Pfad die Datenbank zu finden ist, wie diese heißt und in welcher Tabelle die eigentlichen Daten verzeichnet sind. Danach erstellen Sie ein Connection-Objekt und geben den Namen der Access-Datenbank sowie den Provider bekannt. Legen Sie anschließend eine Recordset-Objektvariable mithilfe der Anweisung Set an. Im Anschluss daran wenden Sie die Methode Open an und übergeben eine SQL-Anweisung, die anhand der Kundennummer die dazugehörigen Kundendaten in der vorher bekannt gegebenen Access-Tabelle abfragt. Danach ist das Recordset gefüllt, sofern die angegebene Kundennummer in der Access-Tabelle gefunden wurde. In diesem Fall liefert die Funktion RecordCount einen Wert > 0 zurück. Dann greifen Sie auf die einzelnen Felder im Recordset-Objekt über die Auflistung Fields zu und übertragen die Inhalte aus dem Recordset in die dafür vorgesehenen Excel-Zellen (siehe Abbildung 10.44).

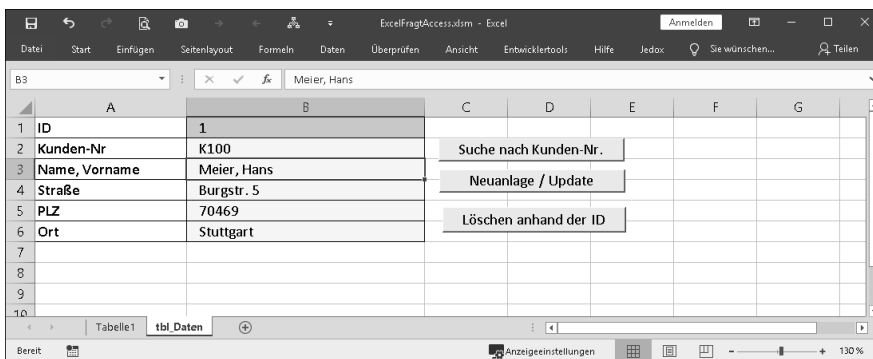


Abbildung 10.44 Die zur »Kunden-Nr.« gehörigen Daten wurden aus der Access-Datenbank geholt.

Daten anlegen und ändern

Bei der nächsten Kernfunktion wird die Access-Datenbank über die Excel-Tabelle um einen neuen Datensatz erweitert. Existiert bereits ein Datensatz in der Datenbank, erfolgt ein Update. Sehen Sie sich vorab Abbildung 10.45 an.

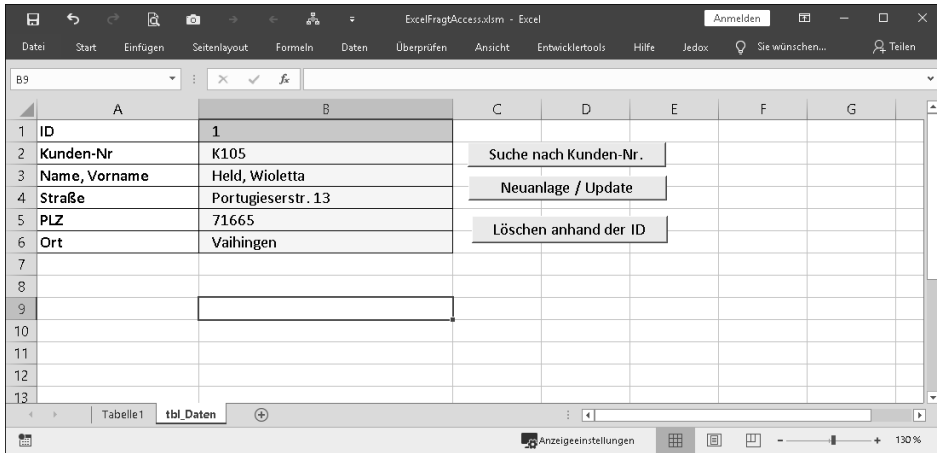


Abbildung 10.45 Ein neuer Datensatz wurde in Excel eingetragen. (Das Feld »ID« bleibt leer.)

Starten Sie die Prozedur aus Listing 10.25, um den neuen Datensatz in der Access-Tabelle anzulegen.

Sub Neuanlage()

```
Dim strDB As String
```

```
Dim strTab As String
```

```
Dim objCon As ADODB.Connection
```

```
Dim rst As ADODB.Recordset
```

```
strDB = ThisWorkbook.Path & "\Kundendb.accdb"
```

```
strTab = "Adressen"
```

```
Set objCon = New ADODB.Connection
```

```
objCon.Open "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" & strDB
```

```
With tbl_Daten
```

```
If Application.WorksheetFunction.CountA(.Range("B2:B6")) = 5 Then
```

```
Set rst = New ADODB.Recordset
```

```
If .Range("B1").Value <> "" Then
```

```
rst.Open "SELECT * FROM " & strTab & " WHERE ID=" & _
```

```
.Range("B1").Value & ";", objCon, adOpenKeyset, adLockOptimistic
```

```
Else
```

```

rst.Open "SELECT * FROM " & strTab & " WHERE ID=" & _
0 & ";", objCon, adOpenKeyset, adLockOptimistic
End If

If rst.RecordCount = 0 Then
  'Neuanlage
  rst.AddNew
  rst.Fields("KundenNr") = .Range("B2").Value
  rst.Fields("KundenName") = .Range("B3").Value
  rst.Fields("Straße") = .Range("B4").Value
  rst.Fields("PLZ") = .Range("B5").Value
  rst.Fields("Ort") = .Range("B6").Value
  .Range("B1").Value = rst.Fields("ID")
  rst.Update
  MsgBox "Satz neu eingefügt!", vbInformation
  objCon.Close
  Set objCon = Nothing
Else
  'Update
  rst.Fields("KundenName") = .Range("B3").Value
  rst.Fields("Straße") = .Range("B4").Value
  rst.Fields("PLZ") = .Range("B5").Value
  rst.Fields("Ort") = .Range("B6").Value
  rst.Update
  MsgBox "Satz neu geändert", vbInformation
  objCon.Close
  Set objCon = Nothing
End If

Else
  MsgBox "Ein gefordertes Feld ist nicht gefüllt!", vbCritical
End If
End With

End Sub

```

Listing 10.25 Das Neuanlegen oder Aktualisieren eines Datensatzes in Access über die Excel-Tabelle

Ob es sich um ein Neuanlegen oder um ein Update eines bereits bestehenden Satzes in der Datenbank handelt, entscheidet die ID in Zelle B1 der Excel-Tabelle. Ist diese leer, dann übergeben Sie der SQL-Anweisung den Wert 0 bei der Suche. Es kann kein Wert 0 in der Access-Tabelle im Feld »ID« gefunden werden. Daher erfolgt ein Neuanlegen über die Methode AddNew. Danach werden die Feldinhalte des Recordset rst über

die Zellen B2:B6 gespeist. Die Methode Update sorgt dafür, dass das Neuanlegen abgeschlossen wird.

Im anderen Fall ergibt die Funktion RecordCount einen Wert > 0. Wenn dies geschieht, dann handelt es sich um ein Update. In diesem Fall werden die Feldinhalte des Recordset rst über die Zellen aus der Excel-Tabelle gefüllt und mit der Methode Update gespeichert.

Daten löschen

Bei der letzten Kernfunktion soll ein bereits angelegter Datensatz aus der Access-Datenbank wieder entfernt werden. Die Löschung soll dabei über die ID erfolgen, wie in Listing 10.26 gezeigt.

```
Sub Löschen()  
    Dim strDB As String  
    Dim strTab As String  
    Dim objCon As ADODB.Connection  
    Dim rst As ADODB.Recordset  
  
    strDB = ThisWorkbook.Path & "\Kundendb.accdb"  
    strTab = "Adressen"  
  
    Set objCon = New ADODB.Connection  
    objCon.Open "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" & strDB  
  
    With tbl_Daten  
        If .Range("B1").Value <> "" Then  
            Set rst = New ADODB.Recordset  
            rst.Open "SELECT * FROM " & strTab & " WHERE ID=" & _  
                .Range("B1").Value & ";", objCon, adOpenKeyset, adLockOptimistic  
  
            If rst.RecordCount <> 0 Then  
                rst.Delete  
                .Range("B1:B6").ClearContents  
                objCon.Close  
                Set objCon = Nothing  
            Else  
                MsgBox "Es konnte kein Kunde mit der ID " & _  
                    .Range("B1").Value & " gefunden werden!", vbInformation  
            End If  
        End If  
    End With
```

```
End With
```

```
End Sub
```

Listing 10.26 Das Löschen aus der Access-Tabelle erfolgt über die eindeutige ID.

Das Muster beim Löschen eines Datensatzes aus der Access-Datenbank gleicht den vorher bereits vorgestellten Kernfunktionen. Übergeben Sie der SQL-Anweisung die ID, die Sie Zelle B1 der Excel-Tabelle entnehmen. Wird der Datensatz in der Access-Tabelle gefunden, dann liefert Ihnen die Funktion `RecordCount` einen Wert >0 . In diesem Fall wenden Sie die Methode `Delete` an, um den Datensatz aus der Access-Tabelle zu entfernen.

10.4.6 Benutzerverwaltung für Access-Anwendungen

Bis Access 2003 gab es die Möglichkeit einer Benutzerverwaltung für Access-Datenbanken. In den neueren Versionen ist sie weggefallen. Das ist nicht sonderlich tragisch, denn erstens war sie nicht besonders sicher und zweitens können Sie eine solche Benutzerverwaltung selbst viel individueller und umfangreicher erstellen.

Was Sie dazu benötigen? Das hängt vom Umfang der einzelnen Berechtigungen ab. In kleinen Projekten reicht eine Tabelle, in der Sie die Benutzer verwalten, vielleicht den Namen, den Usernamen und das Passwort. Bei größeren Projekten können Sie auch Rollen – gegebenenfalls sogar in mehreren Ebenen – vergeben.

Für unser Beispiel gehen wir von einem mittelgroßen Projekt aus: der Datenbank eines mittelständischen Unternehmens mit Produktionsbetrieb. Aus der Beispieldatenbank haben wir uns nur einen kleinen Bereich herausgegriffen: die Arbeitszeiterfassung der Mitarbeiter. Auch die erfolgt nur in einer abgespeckten Version, um das Beispiel nicht unnötig aufzublähen.

Es gibt zunächst eine Tabelle *tabMitarbeiter*, in der alle Mitarbeiter mit Namen, Personalnummer und Abteilung gepflegt werden (siehe Abbildung 10.46). Die Abteilungen sind dabei in einer eigenen Tabelle hinterlegt und hier nur verknüpft.

Die Arbeitszeiten werden in der Tabelle *tabZeiterfassung* gespeichert. Neben Datum, Arbeitsbeginn und -ende wird hier die Personalnummer und gegebenenfalls ein Abwesenheitskennzeichen gespeichert. Hier steht »F« für Feiertag und »U« für Urlaub (siehe Abbildung 10.47). Verwaltet werden die Abwesenheitskennzeichen in der Tabelle *tabAbwesenheiten*. Sie liegen in der Zeiterfassung wieder nur als Verknüpfung vor.

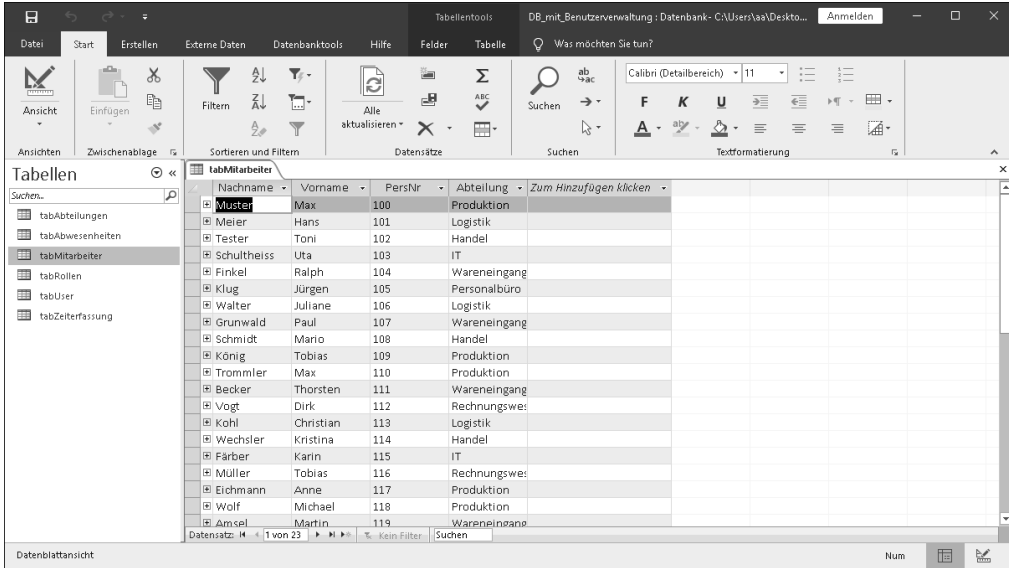


Abbildung 10.46 Die Ausgangssituation

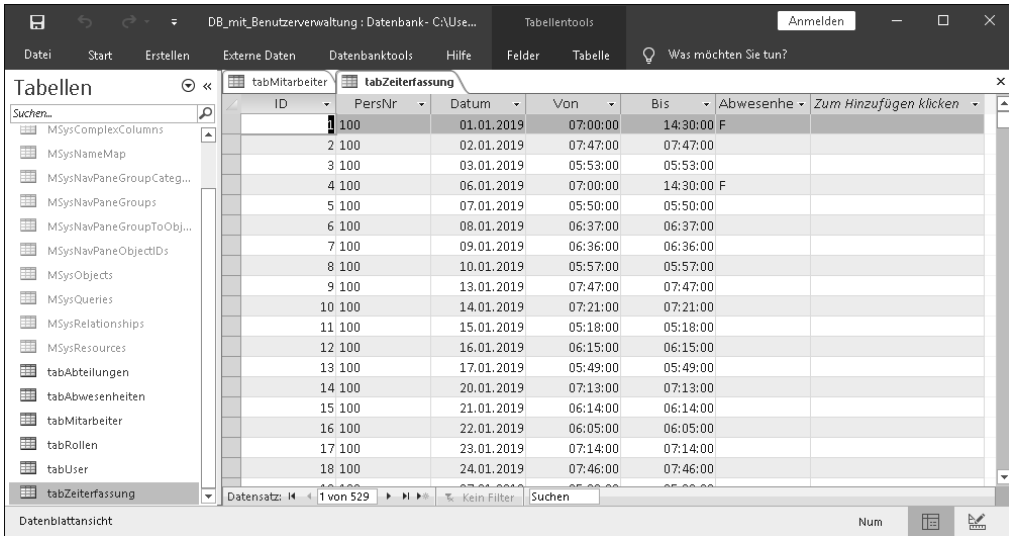


Abbildung 10.47 Die Arbeitszeiten

Unser Ziel soll nun sein, einen Arbeitszeit-Report in Excel verfügbar zu machen. Er soll für einen einzelnen Mitarbeiter für einen gewählten Monat alle Arbeitszeiten auflisten. Dabei soll jedoch nicht jeder die Arbeitszeiten aller Mitarbeiter sehen. Die dürfen nur für die Geschäftsleitung und das Personalbüro komplett verfügbar sein.

Ein Abteilungsleiter wiederum darf nur die Arbeitszeiten der Mitarbeiter seiner Abteilung sehen.

Entsprechend werden nun Benutzer für die Datenbank und die möglichen Rollen angelegt. Schauen wir uns zunächst die Tabelle *tabRollen* an (siehe Abbildung 10.48). Sie enthält lediglich die Bezeichnung der jeweiligen Rolle. Das sind in unserem Beispiel eine Praktikantenrolle, die nur sehr eingeschränkte Berechtigungen für die Daten bekommen wird, der Abteilungsleiter, der im Report, wie bereits festgehalten, nur seine eigenen Mitarbeiter sehen soll, und die beiden letzten Rollen, Gehaltsabrechner und Geschäftsleitung, die – zumindest für die Zeiterfassungswerte – volle Berechtigungen bekommen werden.

| ID | Bezeichnung |
|----|------------------|
| 1 | Praktikant |
| 2 | Abteilungsleiter |
| 3 | Gehaltsabrechner |
| 4 | Geschäftsleitung |
| | (Neu) |

Abbildung 10.48 Die Rollen

In der Tabelle *tabUser* werden diese Rollen dann den einzelnen Benutzern zugeordnet (siehe Abbildung 10.49). Hier werden außerdem der USERNAME, das PASSWORT und die ABTEILUNG, zu der er oder sie gehört, gespeichert. Letztere liegt wieder als Verknüpfung zu *tabAbteilungen* vor.

Das Passwort ist hier als Klartext gespeichert. Es empfiehlt sich, Passwörter immer in verschlüsselter Form abzuspeichern. Beim Anmeldevorgang müsste dann das eingegebene Passwort nach derselben Methode verschlüsselt und mit dem Wert in der Datenbank verglichen werden. Welchen Algorithmus Sie dafür wählen, hängt maßgeblich davon ab, wie sicher das Ganze werden soll. Für die meisten kleineren Anwendungen reicht schon der Caesar-Algorithmus, bei dem eine einfache Verschiebung der einzelnen Zeichen stattfindet. Für unser Beispiel verzichten wir aber auf eine derartige Chiffrierung.

The screenshot shows the Microsoft Access interface with the 'tabUser' table open. The table has the following data:

| ID | Username | Passwort | Rolle | Abteilung |
|----|----------|----------|------------------|--------------|
| 1 | Meier | Test | Praktikant | Logistik |
| 2 | Schulz | Test | Gehaltsabrechner | Personalbüro |
| 3 | Anders | Test | Abteilungsleiter | Logistik |
| 4 | Wichtig | Test | Geschäftsleitung | |

Abbildung 10.49 Die Tabelle für die Passwortaufnahme

Wenn Sie das Excel-Tool dazu öffnen, verwenden Sie bitte eine der hier aufgeführten Anmeldungen. Beachten Sie, dass die verfügbaren Daten in dieser fertigen Lösung bereits eingeschränkt und vom jeweiligen Benutzer abhängig sind. Wenn Sie als Anmeldung den Praktikanten wählen, werden Sie erst mal nicht viel sehen; greifen Sie hier also eher zu Herrn oder Frau Wichtig.

Schauen wir uns jetzt den Report an. Die Mappe ARBEITSZEITBERICHT besteht aus zwei Blättern. Im ersten sieht man nur eine Willkommensnachricht und einen Verweis auf das zweite Blatt, sofern man die notwendigen Berechtigungen hat.

Im zweiten Blatt befindet sich das Grundgerüst für den Arbeitszeit-Report (siehe Abbildung 10.50). In der ersten Zeile sollen MITARBEITER, JAHR und MONAT ausgewählt werden. Die drei Zellen, in denen die Werte stehen, sind mit Namen versehen, um sie im Code besser ansprechen zu können.

Bei JAHR und MONAT wurde bereits ein Dropdown-Feld über den Befehl DATENÜBERPRÜFUNG hinterlegt. Für die Mitarbeiter soll das genauso realisiert werden, allerdings kommt der Inhalt hier aus der Datenbank, und zwar aus der Tabelle *tabMitarbeiter*.

Wird eines der drei Felder – MITARBEITER, MONAT, JAHR – verändert, sollen die Daten im Bereich ab Zeile 4 aktualisiert werden.

Da aber nun nicht jeder Anwender alle Arbeitszeiten sehen darf, muss sich ein Anwender zunächst im Tool anmelden. Dafür entwerfen wir einen kleinen Anmelde-dialog im VBA-Projekt der Excel-Mappe (siehe Abbildung 10.51).

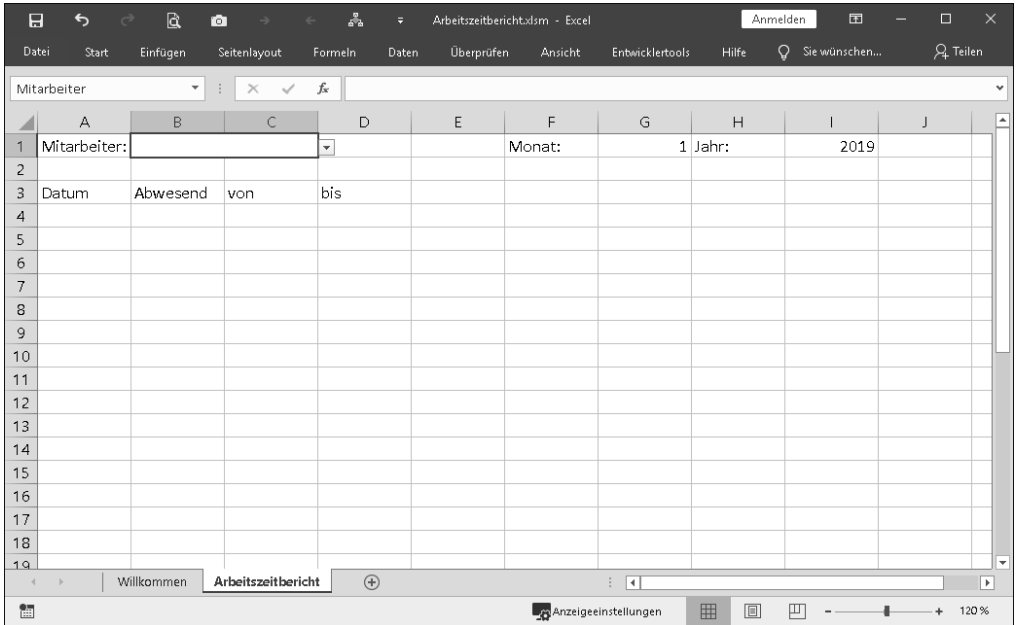


Abbildung 10.50 Das Grundgerüst für den Arbeitszeit-Report

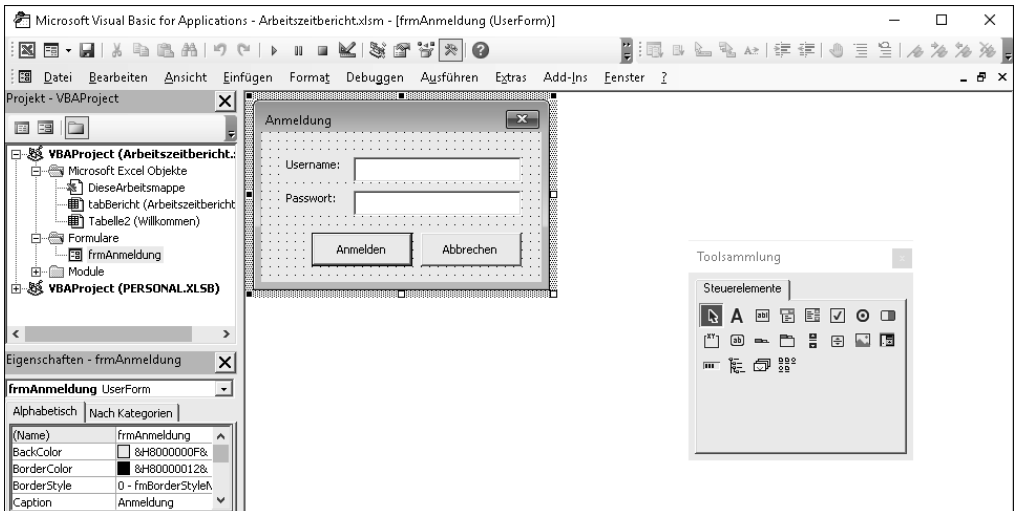


Abbildung 10.51 Der Anmeldedialog

Es handelt sich lediglich um zwei Textfelder für Username und Passwort sowie um zwei Buttons, die das Tool direkt wieder schließen oder die Anmeldung bestätigen sollen.

Nach dem Anmelden wollen wir die Rolle und Abteilung des Users in globalen Variablen speichern. Dafür sind diese vorher im Modul MDLGLOBALS deklariert worden:

```
Global Rolle As String
Global Abteilung As String
```

Der Code hinter dem ABBRECHEN-Button ist denkbar kurz (siehe Listing 10.27). Er schließt die komplette Mappe, ohne irgendwelche Änderungen zu speichern.

```
Private Sub btnClose_Click()
    ThisWorkbook.Close False
End Sub
```

Listing 10.27 Die Prozedur für den Abbruch

Beim ANMELDEN-Button ist der Code dafür umso umfangreicher. Schauen wir uns einmal die komplette Prozedur in Listing 10.28 an.

```
Private Sub btnAnmelden_Click()

    Dim rs As New ADODB.Recordset
    Dim conn As New ADODB.Connection

    If txtPasswort.Tag = "" Then txtPasswort.Tag = 3

    conn.Open "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" & _
    ThisWorkbook.Path & "\DB_mit_Benutzerverwaltung.accdb"

    With rs
        .ActiveConnection = conn
        .CursorType = adOpenKeyset
        .LockType = adLockOptimistic

        .Open "SELECT Passwort, tabRollen.Bezeichnung AS Rolle, Abteilung _
        FROM tabUser INNER JOIN tabRollen ON tabUser.Rolle = _
        tabRollen.ID WHERE Username = '" & txtUser.Text & "'"

    End With

    If rs.RecordCount = 0 Then
        MsgBox "User nicht gefunden"
        rs.Close
    Else
        If rs.Fields("Passwort") <> txtPasswort.Text Then
            txtPasswort.Tag = txtPasswort.Tag - 1
        End If
    End If
End Sub
```

```

If txtPasswort.Tag = 0 Then
    MsgBox "Passwort wurde 3 mal falsch eingegeben.
        Tool wird geschlossen."
    rs.Close
    conn.Close
    Set rs = Nothing
    Set conn = Nothing
    ThisWorkbook.Close False
Else
    MsgBox "Passwort falsch. Noch " & txtPasswort.Tag & _
        " Versuche."
End If
Else
    Rolle = rs.Fields("Rolle")
    If Not IsNull(rs.Fields("Abteilung")) Then
        Abteilung = rs.Fields("Abteilung")
    Else
        Abteilung = ""
    End If
    rs.Close
    Select Case Rolle
        Case "Geschäftsleitung", "Gehaltsabrechner"
            tabBericht.Visible = xlSheetVisible
            rs.Open "SELECT PersNr FROM tabMitarbeiter"
            With tabBericht.Range("Mitarbeiter").Validation
                .Delete
                .Add xlValidateList, xlValidAlertStop, xlBetween, _
                    rs.GetString(, , , ",")
                .IgnoreBlank = False
                .InCellDropdown = True
                .ShowInput = True
                .ShowError = True
            End With
            rs.Close
        Case "Abteilungsleiter"
            tabBericht.Visible = xlSheetVisible
            rs.Open "SELECT PersNr FROM tabMitarbeiter WHERE
                Abteilung = " & Abteilung
            With tabBericht.Range("Mitarbeiter").Validation
                .Delete
                .Add xlValidateList, xlValidAlertStop, xlBetween, _
                    rs.GetString(, , , ",")
                .IgnoreBlank = False
            End With
        Case Else
            rs.Close
    End Select
End If

```

```
        .InCellDropdown = True
        .ShowInput = True
        .ShowError = True
    End With
    rs.Close
Case Else
    tabBericht.Visible = xlSheetVeryHidden
End Select
Unload Me
End If

End If

conn.Close

Set rs = Nothing
Set conn = Nothing

End Sub
```

Listing 10.28 Die Prozedur für das Anmelden

Um auf die Datenbank zugreifen zu können, wird ADO verwendet. Die Bibliothek muss entsprechend als Verweis gesetzt sein. Für den Zugriff werden ein `Recordset` und eine `Connection` verwendet. Da beide auch auf jeden Fall direkt zum Einsatz kommen, kann mit dem Schlüsselwort `New` auch direkt beim Deklarieren eine Instanz gebildet werden.

Da wir nicht unendlich viele Anmeldeversuche akzeptieren wollen, sondern nach dreimaliger Falscheingabe des Passwortes das Tool geschlossen werden soll, brauchen wir einen Ort, an dem wir die misslungenen Anmeldeversuche zählen. Dafür eignet sich die `Tag`-Eigenschaft, die alle Steuerelemente aufweist, hervorragend. Sie bekommt beim ersten Anmeldeversuch den Wert 3 zugewiesen. Jedes Mal, wenn das Passwort falsch eingegeben wird, wird hier um 1 heruntergezählt, und dann wird das Tool beendet. Dazu erfahren Sie weiter unten mehr.

Nun wird die Verbindung zur Datenbank hergestellt und das `Recordset` mit einigen Standardeigenschaften eingestellt. Danach wird darüber eine Abfrage an die Datenbank abgesetzt:

```
.Open "SELECT Passwort, tabRollen.Bezeichnung AS Rolle, Abteilung FROM tabUser  
INNER JOIN tabRollen ON tabUser.Rolle = tabRollen.ID WHERE Username = '" &  
txtUser.Text & "'"
```

Die SQL-Anweisung wählt die benötigten Daten, nämlich Passwort, Rolle und Abteilung, aus der Tabelle *tabUser* aus. Wir wollen die Rolle des angemeldeten Users als Text speichern, um den Code sprechender zu halten. In der WHERE-Klausel verwenden wir den eingegebenen Usernamen als Bedingung für das entsprechende Feld in der Access-Tabelle.

Da der User sich beim Anmelden auch bei seinem Anmeldenamen vertippt haben könnte, wird zunächst geprüft, ob überhaupt ein Datensatz angekommen ist. Das kann man – je nach *CursorType* – mit der Eigenschaft *RecordCount* machen. Die liefert 0, wenn keine passenden Datensätze gefunden wurden. Aber Achtung: Die Cursor-typen *adOpenDynamic* und *adOpenForwardOnly* geben hier generell immer -1 zurück!

Wurde also kein passender Datensatz gefunden, bekommt der User eine entsprechende Meldung per *MsgBox* ausgegeben, und damit ist die Prozedur für diesen Fall schon erledigt.

Wurde ein Datensatz gefunden, muss zuallererst das Passwort geprüft werden. Dafür erfolgt ein Vergleich zwischen dem Feld im *Recordset* und dem Wert in der *Textbox*:

```
If rs.Fields("Passwort") <> txtPasswort.Text Then
    txtPasswort.Tag = txtPasswort.Tag - 1
    If txtPasswort.Tag = 0 Then
        MsgBox _
            "Passwort wurde 3-mal falsch eingegeben. Tool wird geschlossen."
        rs.Close
        conn.Close
        Set rs = Nothing
        Set conn = Nothing
        ThisWorkbook.Close False
    Else
        MsgBox "Passwort falsch. Noch " & txtPasswort.Tag & " Versuche."
    End If
Else
```

Stimmen die Passwörter nicht überein, wird der Zähler in der *Tag*-Eigenschaft den Textfeldes *txtPasswort* um 1 verringert. Danach erfolgt die Prüfung, ob er nun den Wert 0 hat – ob also alle drei Anmeldeversuche verbraucht sind. Ist dies der Fall, bekommt der User eine entsprechende Meldung, und das Tool wird geschlossen. Vorher werden noch das *Recordset* und die *Connection* geschlossen und die Instanzen entfernt.

Bei erfolgreicher Anmeldung werden zuerst die beiden globalen Variablen für Rolle und Abteilung gefüllt. Da das Feld *ABTEILUNG* in der *User*-Tabelle kein Pflichtfeld ist, muss hier noch auf *Null* geprüft werden. *Null* wird nicht implizit in einen Leerstring

umgewandelt; wenn Sie es einer Stringvariablen zuweisen wollen, wird ein Laufzeitfehler ausgelöst.

```
Rolle = rs.Fields("Rolle")
If Not IsNull(rs.Fields("Abteilung")) Then
    Abteilung = rs.Fields("Abteilung")
Else
    Abteilung = ""
End If
rs.Close
```

Anschließend kann das Recordset schon geschlossen werden. Wir haben nun in unserem Tool die Information zur Rolle und zur Abteilung des Anwenders oder der Anwenderin und können abhängig davon weiterarbeiten.

Im Fall des Arbeitszeit-Reports soll anhand dieser Informationen eingestellt werden, welche Mitarbeiter im Auswahlfeld in der ersten Zeile erscheinen sollen und ob dieses überhaupt sichtbar sein soll. Das wird über eine SelectCase-Direktive erledigt, könnte aber auch mit einer IIf-Funktion erfolgen. Die SelectCase-Konstruktion ist aber weitaus übersichtlicher und kann, wenn neue Rollen dazukommen, besser gepflegt werden.

Im ersten Case-Zweig werden die Rollen GESCHÄFTSLEITUNG und GEHALTSABRECHNER behandelt:

```
Case "Geschäftsleitung", "Gehaltsabrechner"
    tabBericht.Visible = xlSheetVisible
    rs.Open "SELECT PersNr FROM tabMitarbeiter"
    With tabBericht.Range("Mitarbeiter").Validation
        .Delete
        .Add xlValidateList, xlValidAlertStop, xlBetween, rs.GetString(, , , ",")
        .IgnoreBlank = False
        .InCellDropdown = True
        .ShowInput = True
        .ShowError = True
    End With
rs.Close
```

Zuerst wird das Berichtsblatt eingeblendet. Danach müssen die Daten für das Dropdown-Menü ermittelt werden.

Anschließend wird das Dropdown-Menü über die Datenüberprüfung der Mitarbeiterzelle eingestellt. Die alte Datenüberprüfung wird gelöscht und eine neue wird als Liste eingefügt. Der Wert für den letzten Parameter muss eine kommasetrennte Liste sein.

Diese Liste erhalten wir ganz einfach über die Methode `GetString` des `Recordset`-Objekts.

Im Fall der Rolle `ABTEILUNGSLEITER` sieht das fast genauso aus:

```
Case "Abteilungsleiter"
    tabBericht.Visible = xlSheetVisible
    rs.Open "SELECT PersNr FROM tabMitarbeiter WHERE Abteilung = " & Abteilung
    With tabBericht.Range("Mitarbeiter").Validation
        .Delete
        .Add xlValidateList, xlValidAlertStop, xlBetween, _
            rs.GetString(, , , ",")
        .IgnoreBlank = False
        .InCellDropdown = True
        .ShowInput = True
        .ShowError = True
    End With
    rs.Close
```

Das Einzige, was sich hier ändert, ist die `SQL`-Anweisung. Genau an dieser Stelle wird eine Art Sicht auf die Daten definiert, und der Benutzer bekommt nur noch einen bestimmten Teil der Datensätze angezeigt.

Realisiert wird das mithilfe der `WHERE`-Klausel: Sie bestimmt, dass nur Datensätze berücksichtigt werden, die im Feld `ABTEILUNG` denselben Wert aufweisen, der beim Anwender hinterlegt ist. Und der wiederum befindet sich ja in unserer globalen Variablen.

Der Rest ist identisch mit dem vorhergehenden Zweig.

Fehlt zu guter Letzt noch der `Case Else`-Zweig – der z. B. auf unseren Praktikanten zutrifft. Hier wird einfach nur das Berichtsblatt explizit ausgeblendet:

```
Case Else
    tabBericht.Visible = xlSheetVeryHidden
```

Das Anmeldeformular ist damit fertig und muss nun nur noch im `Workbook_Open`-Ereignis aufgerufen werden, wie in Listing 10.29 zu sehen ist.

```
Private Sub Workbook_Open()
    tabBericht.Visible = xlSheetVeryHidden
    frmAnmeldung.Show
End Sub
```

Listing 10.29 Den Anmeldeialog starten

Damit sind bereits bei der Anmeldung alle notwendigen Einstellungen zur Sicht auf die Daten vorgenommen worden. Was nun noch fehlt, ist die Funktionalität, die die Daten aus der Datenbank im Tabellenblatt anzeigt. Die befindet sich im Worksheet_Change-Ereignis des Berichtsblattes (siehe Listing 10.30).

```
Private Sub Worksheet_Change(ByVal Target As Range)

    Dim rs As ADODB.Recordset
    Dim conn As ADODB.Connection

    If Target.Address = tabBericht.Range("Mitarbeiter").Address Or _
        Target.Address = tabBericht.Range("Monat").Address Or _
        Target.Address = tabBericht.Range("Jahr").Address Then

        tabBericht.Range("Ausgabebereich").ClearContents

    Set rs = New ADODB.Recordset
    Set conn = New ADODB.Connection

    conn.Open "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" & _
        ThisWorkbook.Path & "\DB_mit_Benutzerverwaltung.accdb"

    With rs
        .ActiveConnection = conn
        .CursorType = adOpenKeyset
        .LockType = adLockOptimistic

        .Open "SELECT Datum, tabAbwesenheiten.AbwesenheitLang AS Abwesenheit,
            von, bis " & _
            "FROM tabZeiterfassung LEFT JOIN tabAbwesenheiten ON
            tabZeiterfassung.Abwesenheit = tabAbwesenheiten.ID " & _
            "WHERE PersNr = '" & tabBericht.Range("Mitarbeiter").Value & _
            "' AND YEAR(Datum) = '" & tabBericht.Range("Jahr").Value & _
            "' AND MONTH(Datum) = '" & tabBericht.Range("Monat").Value & _
            "' ORDER BY Datum"

        tabBericht.Range("Ausgabebereich").Cells(1).CopyFromRecordset rs
        tabBericht.Range("Ausgabebereich").Columns(3).NumberFormat = "HH:MM"
        tabBericht.Range("Ausgabebereich").Columns(4).NumberFormat = "HH:MM"
    .Close
    End With
```

```

'Name und Vorname holen
rs.Open "SELECT Vorname, Nachname FROM TabMitarbeiter WHERE PersNr=" & _
Chr(39) & tabBericht.Range("Mitarbeiter").Value & Chr(39) & ""
tabBericht.Range("Mitarbeiter").Offset(0, 1).Value = rs.Fields("Vorname")
tabBericht.Range("Mitarbeiter").Offset(0, 2).Value = rs.Fields("Nachname")
conn.Close
Set rs = Nothing
Set conn = Nothing

End If

End Sub

```

Listing 10.30 Daten aus der Datenbank in der Tabelle anzeigen

Benötigt werden wieder ein Recordset und eine Connection – beide diesmal nicht direkt als neue Instanz, da die beide Objekte erst benötigt werden, wenn eine der drei Zellen mit MITARBEITER, MONAT oder JAHR geändert wird.

Genau das wird zuerst abgeprüft, indem die Address-Eigenschaften der Target-Zelle und der drei betreffenden Zellen im Blatt verglichen werden.

Fällt diese Prüfung positiv aus, wird zunächst der Datenbereich geleert (auch hier wurde ein Name vergeben). Danach werden neue Instanzen von Recordset und Connection erstellt und die Verbindung zur Datenbank hergestellt. Es folgen die üblichen Einstellungen am Recordset und anschließend das Absetzen der Datenbankabfrage.

Bei der SQL-Anweisung ist es wichtig, die Verknüpfung zwischen den Tabellen *tabZeiterfassung* und *tabAbwesenheiten* als LEFT JOIN zu definieren, da das Feld ABWESENHEIT in der Zeiterfassungstabelle auch Null sein kann und deshalb der INNER JOIN nur Datensätze erfasst, bei denen auch diese Verbindung hergestellt werden kann, die Abwesenheit also nicht Null ist.

In der WHERE-Klausel wird die PersNr angegeben. Ist nichts ausgewählt, kommen später bei einer Abfrage keine Daten an, da es keinen Datensatz mit leerer PersNr gibt.

Anschließend werden noch der Monat und das Jahr aus dem Datumfeld gefiltert. Über die Funktionen MONTH und YEAR lässt sich das direkt in SQL lösen.

Die Daten werden dann in einem Rutsch mithilfe der Methode CopyFromRecordset des Range-Objekts ausgegeben.

Lediglich das Zahlenformat in den beiden Zeitspalten muss noch angepasst werden, da es beim CopyFromRecordset verloren geht.

| | A | B | C | D | E | F | G | H | I | J |
|----|--------------|----------|-------|-------|--------|--------|---|-------|------|---|
| 1 | Mitarbeiter: | 102 | | Toni | Tester | Monat: | 1 | Jahr: | 2019 | |
| 2 | | | | | | | | | | |
| 3 | Datum | Abwesend | von | bis | | | | | | |
| 4 | 01.01.2019 | Feiertag | 07:00 | 14:30 | | | | | | |
| 5 | 02.01.2019 | Urlaub | 07:00 | 14:30 | | | | | | |
| 6 | 03.01.2019 | Urlaub | 07:00 | 14:30 | | | | | | |
| 7 | 06.01.2019 | Feiertag | 07:00 | 14:30 | | | | | | |
| 8 | 07.01.2019 | Urlaub | 07:00 | 14:30 | | | | | | |
| 9 | 08.01.2019 | Urlaub | 07:00 | 14:30 | | | | | | |
| 10 | 09.01.2019 | Urlaub | 07:00 | 14:30 | | | | | | |
| 11 | 10.01.2019 | Urlaub | 07:00 | 14:30 | | | | | | |
| 12 | 13.01.2019 | | 09:01 | 09:01 | | | | | | |
| 13 | 14.01.2019 | | 08:53 | 08:53 | | | | | | |
| 14 | 15.01.2019 | | 07:23 | 07:23 | | | | | | |
| 15 | 16.01.2019 | | 08:37 | 08:37 | | | | | | |
| 16 | 17.01.2019 | | 08:41 | 08:41 | | | | | | |
| 17 | 20.01.2019 | | 09:26 | 09:26 | | | | | | |
| 18 | 21.01.2019 | | 07:15 | 07:15 | | | | | | |
| 19 | 22.01.2019 | | 07:46 | 07:46 | | | | | | |

Abbildung 10.52 Die Daten des ausgewählten Mitarbeiters