Typisierte Random-Dateien

Autor & Copyright: Dipl.-Ing. Harald Nahrstedt

Version: 2016 / 2019 / 2021 / 365

Erstellungsdatum: 12.02-2012

Überarbeitung: 01.12.2022

Beschreibung:

Dieses Kapitel behandelt die Möglichkeit, Random-Dateien mit fester Datensatzlänge zu verwalten. Verwendet wird dazu ein typisierter Satzaufbau. Damit die Datensätze übersichtlich bleiben, wird ein kleiner Trick benutzt. In einem weiteren Schritt wird die Verwaltung der Datensätze nach Schlüsseln gezeigt. Der erste Schritt zum Aufbau von relationalen Datenbanken.

Anwendungs-Dateien:

AE-019_TypisierteRandomDateien1.xlsm
AE-019_TypisierteRandomDateien2.xlsm
AE-019_TypisierteRandomDateien3.xlsm

1 Typisierte Random Dateien

Random-Dateien lassen, wie der Name schon sagt, einen wahlfreien Zugriff auf Datensätze zu. Anders wie bei sequentiellen Dateien, kann ein beliebiger Datensatz jederzeit gelesen und beschrieben werden. Grundvoraussetzung ist eine feste Datensatzlänge. Da bietet sich die Definition eines Datensatz als benutzerdefinierten Typ an.

Als einfaches Beispiel benutzen wir den Aufbau eines Adress-Datensatzes. Alle Elemente sind vom Typ String und haben eine feste Länge (* n), so dass auch deren Summe und damit der Datensatz eine feste Länge besitzt. Ein kleiner Trick, damit die Datei später auch mit einem einfachen Texteditor lesbar ist, ist das Anfügen der Steuerzeichen Crt für *Carriage Return* (Wagenrücklauf) und Lfd für *Line Feed* (Zeilenvorschub).

Codeliste 1. Aufbau einer Adress-Datei im Modul modAdressen

```
Option Explicit

Type Adressen
   Key      As String * 3
   Name     As String * 20
   Vorname  As String * 20
   Strasse  As String * 30
   PLZ      As String * 10
   Ort      As String * 30
   Crt      As String * 1
   Lfd      As String * 1
End Type

Public lAdrLen As Long     'Länge des Datensatzes
Public lAdrNum As Long     'Aktuelle Dateinummer
Public lAdrMax As Long     'Maximale Anzahl vorhandener Datensätze
Public vAdrRec As Variant  'aktuelle Datensatznummer
```

Die drei Long-Definitionen beinhalten Werte, die den Umgang mit der Datei vereinfachen. Die Datensatznummer muss vom Type Variant sein, siehe nachfolgende *Put*-Definition. Diesen Code setzen wir in ein Modul mit dem Namen *modAdressen*. Außerdem bekommt dieses Modul später noch elementare Funktionen zur Handhabung der Datei.

Die allgemeine Syntax zum Öffnen einer Randomdatei lautet

```
Open Dateiname For Random Shared As Dateinummer Len = Satzlänge
```

Tabelle 1. Attribute der Methode Open

Parameter	Beschreibung
Dateiname	Pfad und Dateiname der zu öffnenden Datei.
Dateinummer	Gültig im Bereich von 1 bis 511. Mit der *FreeFile*-Funktion erhält man die nächste verfügbare Dateinummer. Unter dieser Nummer erfolgen dann alle weiteren Zugriffe des Programms auf die Datei.

Die *Put*-Anweisung schreibt Daten aus einer Variablen in eine Datei. Die Syntax lautet

```
Put [#] Filenumber, [Recordnumber], VarName
```

Tabelle 2. Attribute der Methode Put

Parameter	Beschreibung
Filenumber	Erforderlich. Eine beliebige gültige Dateinummer.
Recnumber	Optional. Wert vom Typ Variant (Long). Datensatznummer (Dateien im Random-Modus) oder Byte-Nummer (Dateien im Binary-Modus), an der der Schreibvorgang beginnt.
VarName	Erforderlich. Name der Variablen, die Daten enthält, die auf den Datenträger geschrieben werden sollen.

Mit der *Put*-Methode geschriebene Daten werden in der Regel aus einer Datei mit der *Get*-Methode gelesen.

Der erste Datensatz oder das erste Byte in einer Datei beginnt an Position 1, der zweite Datensatz oder das zweite Byte an Position 2 usw. Wird die Recnumber weglassen, wird der nächste Datensatz oder das nächste Byte nach der letzten *Get* - oder *Put*-Anweisung oder auf den durch die letzte *Seek*-Funktion verwiesen wird, geschrieben. Es müssen Kommas als Trennzeichen angeben werden.

Damit können wir nun eine Funktion schreiben, die Pfad und Dateinamen als Parameter bekommt. Außerdem ist noch die Angabe der Datensatzlänge erforderlich. In der Funktion wird zunächst das Vorhandensein der Datei abgefragt, denn wir wollen nicht unbedingt jedesmal eine Datei erzeugen mit der ersten Abfrage, falls es sie noch nicht gibt. Für das Erstellen einer neuen Datei schreiben wir eine eigene Prozedur. Dazu öffnen wir lediglich die Datei kurz im Modus Output. In diesem Modus wird einfach nur eine Dateihülle angelegt. Sollte, wider Erwarten, eine Datei mit gleichem Namen existieren, wird sie zerstört. Natürlich muss auch eine Datei gelöscht werden können. Dies übernimmt die Funktion *AdrDatei_Löschen*.

Codeliste 2. Funktionen zur Handhabung der Random-Datei

```
Public Function AdrDateiÖffnen _
   (sDatei As String, lLänge As Long) As Boolean
   Dim Data As Adressen

   If Not FileExist(sDatei) Then
      AdrDateiÖffnen = False
      Exit Function
   End If
   AdrDateiÖffnen = True
   lAdrNum = FreeFile
   Open sDatei For Random Shared As lAdrNum Len = lLänge
   lAdrMax = LOF(lAdrNum) / lLänge
   Do
      If lAdrMax > 0 Then
         Get lAdrNum, lAdrMax, Data
         If Data.Key = Space(Len(Data.Key)) Then
            lAdrMax = lAdrMax - 1
         End If
      End If
```

```
   Loop While Data.Key = Space(Len(Data.Key)) _
      And lAdrMax > 0
End Function

Public Function AdrDateiErstellen _
   (sDatei As String) As Boolean
   If FileExist(sDatei) Then
      AdrDateiErstellen = False
      Exit Function
   End If
   lAdrNum = FreeFile
   Open sDatei For Random As lAdrNum
   AdrDateiErstellen = True
End Function
```

Hilfsprozeduren bekommen ein eigenes Modul.

Codeliste 3. Hilfsprozedur im Modul modHilfe

```
Option Explicit

Public Function FileExist(sDateiname As String) As Boolean
   On Error GoTo ErrorHandler
   FileExist = Dir(sDateiname) <> ""
   Exit Function
ErrorHandler:
   FileExist = False
   Resume Next
End Function
```

Dermaßen ausgerüstet können wir nun beginnen, mit dieser Datei zu arbeiten. Diese Prozeduren setzen wir ebenfalls in ein eigenes Code-Modul *modOrga*. Die nachfolgende Prozedur öffnet die Datei und schreibt einen Datensatz in die Datei, und schließt sie wieder. Die Eingabe werden abgefragt und es werden daraus vereinfachte Datensätze geformt. Die zweite Prozedur stellt einen vorhandenen Datensatz in das Formblatt.

Codeliste 4. Die Prozedur erzeugt Adressen im Modul modOrga

```
Public Const sFileName = "C:\Temp\Random\Daten.dat"

Public Sub AdresseErzeugen()
   Dim Data       As Adressen
   Dim sText      As String
   Dim sKey       As String

   If Not ActiveSheet.Name = "Adressen Formular" Then Exit Sub

   If AdrDateiÖffnen(sFileName, Len(Data)) = False Then
      sText = "Datei " & sFileName & " existiert nicht! Erstellen?"
      If MsgBox(sText, vbYesNo) = vbYes Then
         If Not AdrDateiErstellen(sFileName) Then
            sText = " Datei " & sFileName & " wurde nicht erstellt!"
            MsgBox sText
            Exit Sub
         End If
      Else
         Exit Sub
      End If
   End If
   sKey = InputBox("Schlüssel = ")
```

```
   vAdrRec = InputBox("Satznummer = ")
   LSet Data.Key = sKey
   LSet Data.Name = "Name" & vAdrRec
   LSet Data.Vorname = "Vorname" & vAdrRec
   LSet Data.Strasse = "Strasse" & vAdrRec
   LSet Data.PLZ = "D-PLZ" & vAdrRec
   LSet Data.Ort = "Wohnort" & vAdrRec
   LSet Data.Crt = vbCr
   LSet Data.Lfd = vbLf
   Put lAdrNum, vAdrRec, Data
   Close lAdrNum
End Sub
```

Der Inhalt der Datei, sieht dem Aufbau einer relationalen Datenbank ähnlich (Bild 1).

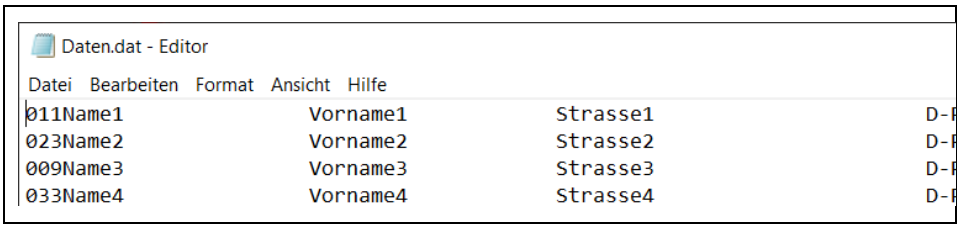

Bild 1. Ansicht des Inhalts der Random-Datei im Texteditor

Die folgende Prozedur überträgt den Inhalt eines Datensatzes aus der Random-Datei in das Arbeitsblatt *Adressen Formular* der Excel-Mappe. Damit führende Nullen im Key nicht in der Excel-Tabelle verloren gehen, bekommt die Zelle ein benutzerdefiniertes Format.

Codeliste 5. Prozedur zur Übertragung eines Datensatzes in die Excel-Tabelle Adressen Formular im Modul modOrga

```
Public Sub AdresseLesen()
   Dim wshAdr     As Worksheet
   Dim Data       As Adressen

   If Not ActiveSheet.Name = "Adressen Formular" Then Exit Sub
   Set wshAdr = Worksheets("Adressen Formular")
   If AdrDateiÖffnen(sFileName, Len(Data)) = True Then
      vAdrRec = InputBox("Satznummer = ")
      Get lAdrNum, vAdrRec, Data
      With wshAdr
         .Cells(vAdrRec, 1) = Data.Key
         .Cells(vAdrRec, 1).NumberFormat = "000"
         .Cells(vAdrRec, 2) = Data.Name
         .Cells(vAdrRec, 3) = Data.Vorname
         .Cells(vAdrRec, 4) = Data.Strasse
         .Cells(vAdrRec, 5) = Data.PLZ
         .Cells(vAdrRec, 6) = Data.Ort
      End With
   End If
   Close lAdrNum
   Set wshAdr = Nothing
End Sub
```

Danach besitzt das Arbeitsblatt ebenfalls die Daten der Datei (Bild 2).

	A	B	C	D	E	F
1	011	Name1	Vorname1	Strasse1	D-PLZ1	Wohnort1
2	023	Name2	Vorname2	Strasse2	D-PLZ2	Wohnort2
3	009	Name3	Vorname3	Strasse3	D-PLZ3	Wohnort3
4	033	Name4	Vorname4	Strasse4	D-PLZ4	Wohnort4

Bild 2. Aufbau der Excel-Tabelle

Es fehlt noch eine Löschprozedur.

Codeliste 6. Prozedur im Modul modOrga löscht die Random-Datei

```
Public Sub AdressenLöschen()
   Dim sText As String

   If AdrDateiLöschen(sFileName) Then
      sText = "Datei " & sFileName & " gelöscht!"
      MsgBox sText
   Else
      sText = "Datei " & sFileName & " nicht gelöscht!"
      MsgBox sText
   End If
End Sub
```

2 Die Problematik des Wahlfreien Zugriffs

In dem vorangegangenen Beispiel haben wir die Datensätze hintereinander in die Datei geschrieben. Doch was ist, wenn wir die Daten des 2. Datensatz als 3. Datensatz speichern. Löschen wir also die Datei und führen wir einen Test durch. Schreiben und Lesen erfolgt ebenfalls problemlos und wir finden auch die Datenelemente im Arbeitsblatt richtig dargestellt (Bild 3).

	A	B	C	D	E	F
1	011	Name1	Vorname1	Strasse1	D-PLZ1	Wohnort1
2						
3	023	Name3	Vorname3	Strasse3	D-PLZ3	Wohnort3
4						

Bild 3. Inhalt nach dem Löschvorgang

Öffnen wir die Datei mit Word, um auch die Steuerzeichen zu sehen, so wird der Platz für den 2. Datensatz durch Steurzeichen besetzt (Bild 4).

```
011Name1···············Vorname1············Strasse1··················
D-PLZ1····Wohnort1······················¶
ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ023Name3··············Vorna
Strasse3·····················D-PLZ3····Wohnort3·····················
```

Bild 4. Inhalts-Anzeige mit Steuerzeichen in Word

Leider gilt dies auch für die Steuerzeichen Crt und Lfd, sodass sich der 3. Datensatz nahtlos anschließt. Erst wenn auch der 2. Datensatz beschrieben wurde, ist die Darstellung im Texteditor wieder homogen.

3 Die Einführung eines Primärschlüssels

Natürlich kann man die Personendaten auch so speichern, dass ihre Personalnummer auch der Datensatznummer entsprechen. Doch dies kann unter Umständen sehr viele Leersätze erzeugen. Sinnvoller ist die Aneinanderreihung von Datensätzen in sortierter Form. Dazu muss ein Datensatzelement als Primärschlüssel definiert werden. Der Einfachheit halber nehmen wir dazu das erste Element. Das muss zwar nicht sein, macht die Programmierung aber etwas einfacher. Für unser Beispiel nehmen wir den bisher wenig genutzten Key. Es könnte aber auch der Name sein. Dann gibt es wieder das Problem der Namensgleichheit, also könnte man den Vornamen mit dazu nehmen. Sind die auch wieder gleich, muss man zusätzlich einen Schlüssel einführen. Zum Beispiel die ersten zwei Buchstaben vom Namen plus der ersten zwei Buchstaben vom Vornamen plus einer fortlaufenden Nummer. Letztlich geht es um die Eindeutigkeit eines Schlüssels. Denn dies ist die Grundvoraussetzung eines Primärschlüssels (Bild 5).

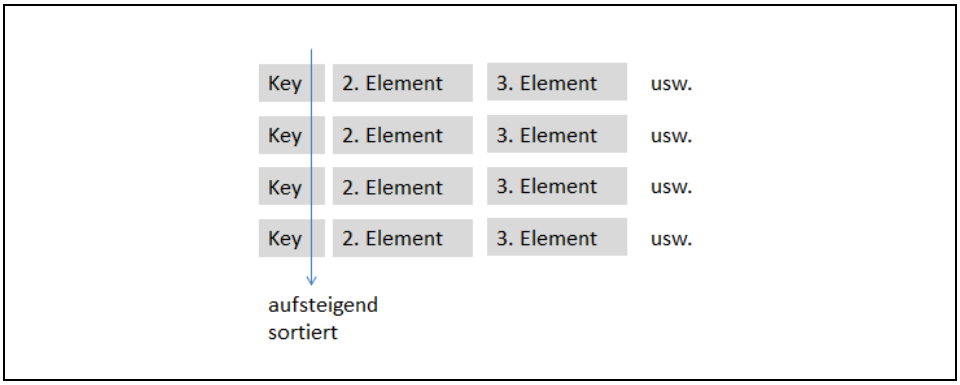

Bild 5. Aufbau der Random-Datei

4 Die Bisektionsmethode

Natürlich könnte man eine sortierte Datei sequentiell lesen, um den Ort zu finden, an der man einen neuen Schlüssel einfügen sollte. Doch diese Methode ist bei großen Datenmengen einfach zu langsam. Daher benutzen wir die Bisektionsmethode.

Ich will sie an einem einfachen Beispiel erklären. Angenommen, Sie suchen einen Namen im Telefonbuch. Dann schlagen Sie es in der Mitte auf. Entweder steht dann der Name in der linken oder rechten Hälfte. Nehmen wir an, er steht in der rechten. Dann teilen Sie diese Hälfte wieder in der Mitte und entscheiden wie zuvor. So kommen Sie relativ schnell auf die Seite mit dem Namen. In meinem Buch *Algorithmen für Ingenieure* finden Sie ein VBA-Beispiel zu diesem Thema, und nachfolgend sehen Sie daraus das Struktogramm.

Tabelle 1. Struktogramm der Bisektionsmethode

<table>
<tr><td colspan="3">Eingabe des Suchbegriffs</td></tr>
<tr><td colspan="3">v=1, b=Anzahl Einträge</td></tr>
<tr><td rowspan="5"></td><td colspan="2">So lange b>=v</td></tr>
<tr><td colspan="2">$m = Int(\frac{v+b}{2})$</td></tr>
<tr><td colspan="2">Ist t > Listeneintrag in Zeile m</td></tr>
<tr><td>Ja</td><td>Nein</td></tr>
<tr><td>v=m+1</td><td>b=m-1</td></tr>
<tr><td colspan="2">i=v</td></tr>
<tr><td colspan="3">Ausgabe Listeneintrag in Zeile i</td></tr>
</table>

5 Sortiert typisierte Random-Dateien

In einem ersten Schritt benötigen wir eine Prozedur, die uns die Datensatznummer liefert, an der der Datensatz nach seinem Primärschlüssel einsortiert werden soll. Undzwar auch dann, wenn die Datei erst angelegt, also noch leer ist.

Die nachfolgende Funktion findet mithilfe der Bisektionsmethode diese Position und liefert sie als Parameter *Rec*. Die Funktion hat außerdem den Wert *True*, falls es den Schlüssel bereits gibt. Sie wird im Modul für die Dateifunktionen mit aufgenommen.

Codeliste 7. Die Prozedur im Modul modAdressen bestimmt die Datensatznummer

```
Public Function AdressenIndex _
   (sKey As String, lMax As Long, vRec As Variant, _
   Data As Adressen) As Boolean
   Dim lVon As Long
   Dim lBis As Long
   Dim lNeu As Long

   lVon = 1
   lBis = lMax
   Do While lBis >= lVon
      lNeu = Int((lVon + lBis) / 2)
      Get lAdrNum, lNeu, Data
      If sKey > Data.Key Then
         lVon = lNeu + 1
      Else
         If sKey = Data.Key Then
            lVon = lNeu
            lBis = lVon - 1
         Else
            lBis = lNeu - 1
```

```
         End If
      End If
   Loop
   vRec = lVon
   If sKey = Data.Key Then
      AdressenIndex = True
   Else
      AdressenIndex = False
   End If
End Function
```

Die ursprüngliche Prozedur zum Schreiben in eine Random Datei wird um diese Funktion ergänzt und hat nun den folgenden Aufbau.

Codeliste 8. Die Prozedur im Modul modOrga schreibt Daten nach der Indexnummer

```
Public Sub AdresseErzeugen()
   Dim Data       As Adressen
   Dim sText      As String
   Dim sKey       As String
   Dim lNum       As Long
   Dim bExist     As Boolean

   If Not ActiveSheet.Name = "Adressen Formular" Then Exit Sub

   If AdrDateiÖffnen(sFileName, Len(Data)) = False Then
      sText = "Datei " & sFileName & " existiert nicht! Erstellen?"
      If MsgBox(sText, vbYesNo) = vbYes Then
         If Not AdrDateiErstellen(sFileName) Then
            sText = " Datei " & sFileName & " wurde nicht erstellt!"
            MsgBox sText
            Exit Sub
         End If
      Else
         Exit Sub
      End If
   End If
   sKey = InputBox("Schlüssel = ")
   bExist = AdressenIndex(sKey, lAdrMax, vAdrRec, Data)
   If bExist Then
      sText = "Daten überschreiben?"
      If MsgBox(sText, vbYesNo) = vbNo Then
         Exit Sub
      End If
   End If
   LSet Data.Key = sKey
   LSet Data.Name = "Name" & vAdrRec
   LSet Data.Vorname = "Vorname" & vAdrRec
   LSet Data.Strasse = "Strasse" & vAdrRec
   LSet Data.PLZ = "D-PLZ" & vAdrRec
   LSet Data.Ort = "Wohnort" & vAdrRec
   LSet Data.Crt = vbCr
   LSet Data.Lfd = vbLf
   Put lAdrNum, vAdrRec, Data
   Close lAdrNum
End Sub
```

Die Prozedur hat eine vereinfachte Eingabe, sodass alle Tests ohne großen Aufwand durchgeführt werden können. Die ersten beiden Datensätze mit den Schlüsseln 011 und 023 werden problemlos eingefügt (Bild 6).

```
011Name1··············Vorname1···········Strasse1·····················
D-PLZ1····Wohnort1·····················¶
023Name2··············Vorname2···········Strasse2·····················
D-PLZ2····Wohnort2·····················¶
```

Bild 6. Inhalts-Anzeige mit Steuerzeichen in Word

So weit, so gut. Die Prozedur, und damit auch die Funktion, arbeiten ohne Probleme. Nur ist es diesmal mit einem einfachen *Put*-Befehl nicht mehr getan. Denn für einen neuen Datensatz mit dem Schlüssel *009* erhalten wir die Datensatznummer 1. Das ist korrekt, denn an der Stelle, an der der erste Datensatz steht, sollte jetzt der neue stehen, und der erste auf der Datensatznummer 2 usw. Doch hier wird er nur überschrieben. Wir benötigen also noch neben einer Funktion zum Schreiben noch eine Funktion zum Einfügen, mit der wir zukünftig alle Datensätze eintragen.

Zunächst die Funktion für das Schreiben eines Datensatzes an eine bestimmte Datensatznummer.

Codeliste 9. Die Funktion in Modul modAdressen schreibt einen Datensatz an eine bestimmte Datensatz-Nummer

```
Public Function AdressenSchreiben _
   (vRec As Variant, Data As Adressen) As Boolean

   On Error GoTo Error_Schreiben
   Put lAdrNum, lRec, Data
   AdressenSchreiben = True
   Exit Function
Error_Schreiben:
   AdressenSchreiben = False
End Function
```

Und nun die Funktion zum Einfügen eines Datensatzes an eine bestimmte Datensatznummer.

Codeliste 10. Die Funktion in Modul modAdressen fügt einen Datensatz an einer bestimmten Datensatz-Nummer ein

```
Public Function AdressenEinfügen _
   (vRec As Variant, lMax As Long, Data As Adressen) As Boolean
   Dim Data2 As Adressen
   Dim lNeu  As Long

   On Error GoTo Error_Einfügen
   If vRec <= lMax Then
      For lNeu = lMax To vRec Step -1
         Get lAdrNum, lNeu, Data2
         Put lAdrNum, lNeu + 1, Data2
      Next lNeu
   End If
   Put lAdrNum, vRec, Data
   lMax = lMax + 1
   AdressenEinfügen = True
   Exit Function
Error_Einfügen:
   AdressenEinfügen = False
End Function
```

Um diese beiden Funktionen nutzen zu können, muss die Schreib-Prozedur noch einmal abgewandelt werden.

Codeliste 11. Die noch einmal abgewandelte Schreib-Prozedur in Modul modOrga

```
Public Sub AdresseErzeugen()
   Dim Data       As Adressen
   Dim sText      As String
   Dim sKey       As String
   Dim lNum       As Long
   Dim bExist     As Boolean

   If Not ActiveSheet.Name = "Adressen Formular" Then Exit Sub

   If AdrDateiÖffnen(sFileName, Len(Data)) = False Then
      sText = "Datei " & sFileName & " existiert nicht! Erstellen?"
      If MsgBox(sText, vbYesNo) = vbYes Then
         If Not AdrDateiErstellen(sFileName) Then
            sText = " Datei " & sFileName & " wurde nicht erstellt!"
            MsgBox sText
            Exit Sub
         End If
      Else
         Exit Sub
      End If
   End If
   sKey = InputBox("Schlüssel = ")
   bExist = AdressenIndex(sKey, lAdrMax, vAdrRec, Data)
   If bExist Then
      sText = "Daten überschreiben?"
      If MsgBox(sText, vbYesNo) = vbNo Then
         Exit Sub
      End If
   End If
   LSet Data.Key = sKey
   LSet Data.Name = "Name" & vAdrRec
   LSet Data.Vorname = "Vorname" & vAdrRec
   LSet Data.Strasse = "Strasse" & vAdrRec
   LSet Data.PLZ = "D-PLZ" & vAdrRec
   LSet Data.Ort = "Wohnort" & vAdrRec
   LSet Data.Crt = vbCr
   LSet Data.Lfd = vbLf

   If bExist = True Then
      If AdressenSchreiben(vAdrRec, Data) = False Then
         sText = "Datensatz konnte nicht geschrieben werden!"
         MsgBox sText
      End If
   Else
      If AdressenEinfügen(vAdrRec, lAdrMax, Data) = False Then
         sText = "Datensatz konnte nicht eingefügt werden!"
         MsgBox sText
      End If
   End If
   Close lAdrNum
End Sub
```

Ein Blick mit einem Texteditor in die Datei zeigt, dass die Einfüge-Prozedur funktioniert (Bild 7).

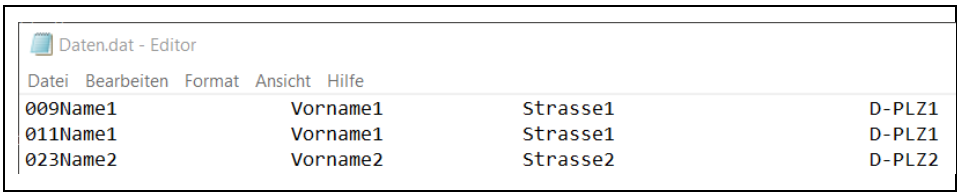

Bild 7. Dateiinhalt nach dem Einfügen

Erneute Aufrufe der Schreib-Prozedur mit anderen Schlüsseln zeigen, dass auch weitere Datensätze richtig einsortiert werden (Bild 8).

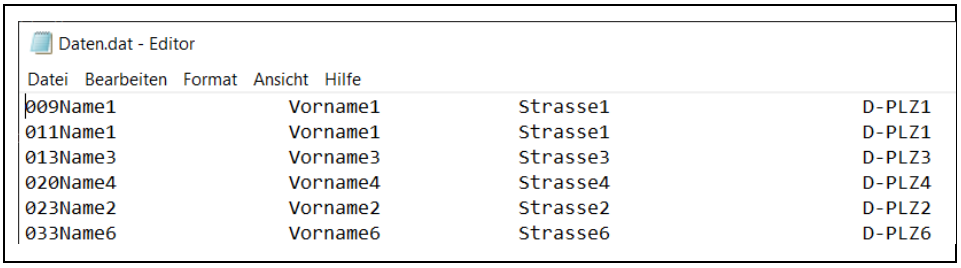

Bild 7. Dateiinhalt nach weiteren Operationen

Letztlich fehlt noch eine Funktion, die einen Datensatz auch wieder entfernen kann. Durch das Entfernen müssen natürlich allen nachfolgenden Datensätze vorgezogen werden.

Codeliste 12. Die Funktion mi Modul modAdressen entfernt einen Datensatz

```
Public Function AdresseEntfernen _
   (vRec As Variant, lMax As Long) As Boolean
   Dim Data2 As Adressen
   Dim lNeu  As Long

   On Error GoTo Error_Entfernen
   If vRec <= lMax Then
      For lNeu = vRec + 1 To lMax
         Get lAdrNum, lNeu, Data2
         Put lAdrNum, lNeu - 1, Data2
      Next lNeu
   End If
   LSet Data2.Key = ""
   LSet Data2.Name = ""
   LSet Data2.Vorname = ""
   LSet Data2.Strasse = ""
   LSet Data2.PLZ = ""
   LSet Data2.Ort = ""
   LSet Data2.Crt = vbCr
   LSet Data2.Lfd = vbLf
   Put lAdrNum, lMax, Data2
   lMax = lMax - 1
   AdresseEntfernen = True
   Exit Function
Error_Entfernen:
   AdresseEntfernen = False
End Function
```

Getestet wird die Funktion mit der nachfolgenden Prozedur.

Codeliste 13. Testprozedur im Modul modOrga

```
Public Sub IndexEntfernen()
   Dim Data       As Adressen
   Dim sText      As String
   Dim sKey       As String

   If AdrDateiÖffnen(sFileName, Len(Data)) = False Then
      sText = "Datei " & sFileName & " existiert nicht!"
      Exit Sub
   End If
   sKey = InputBox("Schlüssel = ")
   If AdressenIndex(sKey, lAdrMax, vAdrRec, Data) = True Then
      If AdresseEntfernen(vAdrRec, lAdrMax) = False Then
         sText = "Datensatz wurde nicht gelöscht!"
         MsgBox sText
      Else
         sText = "Datensatz gelöscht!"
         MsgBox sText
      End If
   Else
      sText = "Schlüssel nicht vorhanden!"
      MsgBox sText
   End If
End Sub
```

Das Ergebnis zeigt eine richtige Funktion und ein kleines Problem (Bild 9).

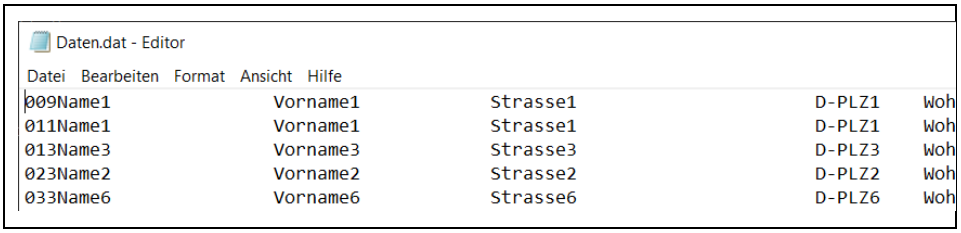

Bild 9. Dateiinhalt nach dem Entfernen von Key 020

Durch das Entfernen von Datensätzen sammeln sich am Ende der Datei leere Datensätze, die aber nicht belegt sind. Sie dürfen bei der Bestimmung der Anzahl vorhandener Datensätze nicht berücksichtigt werden. Die in der Funktion *AdrDateiÖffnen* vorhandene Anweisung

```
   lAdrMax = LOF(lAdrNum) / lLänge
```

muss im Anschluss noch wie folgt korrigiert werden und ist bereits vorhanden.

Codeliste 14. Korrekturanweisungen in der Funktion AdrDateiÖffnen im Modul modAdressen

```
   Do
      If lAdrMax > 0 Then
         Get lAdrNum, lAdrMax, Data
         If Data.Key = Space(Len(Data.Key)) Then
            lAdrMax = lAdrMax - 1
         End If
      End If
   Loop While Data.Key = Space(Len(Data.Key)) _
```

```
       And lAdrMax > 0
```

Nun wird beim nächsten Öffnen der Datei die richtige Satzanzahl erzeugt und weitere Datensätze können hinzugefügt und gelöscht werden.

6 Formulare und Listen

Natürlich macht es keinen Sinn, jedesmal eine Prozedur zu schreiben, wenn man einen Datensatz hinzufügen möchte. Dazu verwenden wir ein Arbeitsblatt als Eingabe-Formular (Bild 10).

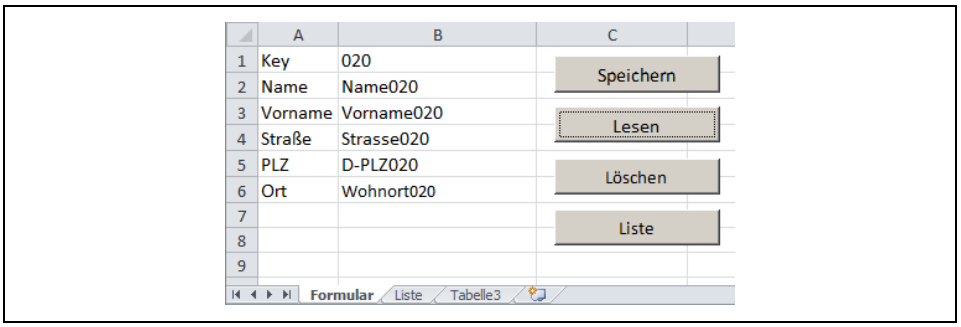

Bild 10. Formular auf dem Arbeitsblatt

Es kann dazu auch eine UserForm verwendet werden. Das Formular verfügt über 4 ActivX-Steuerelemente der Form *CommandButton*, die die nachfolgenden Event-Prozeduren im Codefenster des Arbeitsblattes Formular besitzen. Darin finden sich die Inhalte unserer Testprozeduren in etwas angewandelter Form wieder.

Die Schreibprozedur erwartet als Angaben Werte im Bereich B1:B6. Sie berücksichtigt auch bereits vorhandene Schlüssel.

Codeliste 15. Ereignisprozedur Speichern

```
Private Sub cbnSpeichern_Click()
   Dim FileName   As String
   Dim Data       As Adressen
   Dim Text       As String
   Dim Dat_Rec    As Long
   Dim Dat_Txt    As String
   Dim Key        As String
   Dim Nummer     As Long
   Dim Existent   As Boolean

   FileName = "C:\Temp\Random\Daten.dat"
   If Adressen_Öffnen(FileName, Len(Data)) = False Then
      Text = "Datei " & FileName & " existiert nicht! Erstellen?"
      If MsgBox(Text, vbYesNo) = vbYes Then
         If Not Adressen_Erstellen(FileName) Then
            Text = " Datei " & FileName & " wurde nicht erstellt!"
            MsgBox Text
            Exit Sub
         Else
            If Adressen_Öffnen(FileName, Len(Data)) = False Then
               Text = "Datei " & FileName & _
                  " wurde nicht erstellen?"
```

```
               MsgBox Text
               Exit Sub
            End If
         End If
      Else
         Exit Sub
      End If
   End If

   Key = Right("000" & Trim(Str(Val(Cells(1, 2)))), 3)
   Existent = Adressen_Index(Key, Dat_Max, Dat_Rec, Data)
   If Existent = True Then
      Text = "Daten überschreiben?"
      If MsgBox(Text, vbYesNo) = vbNo Then
         Exit Sub
      End If
   End If
   LSet Data.Nummer = Key
   LSet Data.Name = Cells(2, 2)
   LSet Data.Vorname = Cells(3, 2)
   LSet Data.Strasse = Cells(4, 2)
   LSet Data.PLZ = Cells(5, 2)
   LSet Data.Ort = Cells(6, 2)
   LSet Data.Crt = vbCr
   LSet Data.Lfd = vbLf
   If Existent = True Then
      If Adressen_Schreiben(Dat_Rec, Data) = False Then
         Text = "Datensatz konnte nicht geschrieben werden!"
         MsgBox Text
      End If
   Else
      If Adressen_Einfügen(Dat_Rec, Dat_Max, Data) = False Then
         Text = "Datensatz konnte nicht eingefügt werden!"
         MsgBox Text
      End If
   End If
   Close Dat_Num
End Sub
```

Die Leseprozedur erwartet den zu lesenden Schlüssel in B1. Wird der Schlüssel gefunden, werden alle Datensatzelemente in die Zellen B2:B6 geschrieben.

Codeliste 16. Ereignisprozedur Lesen

```
Private Sub cbnLesen_Click()
   Dim FileName   As String
   Dim Data       As Adressen
   Dim Text       As String
   Dim Dat_Rec    As Long
   Dim Dat_Txt    As String
   Dim Key As String

   FileName = "C:\Temp\Random\Daten.dat"
   If Adressen_Öffnen(FileName, Len(Data)) = False Then
      Text = "Datei " & FileName & " existiert nicht! Erstellen?"
      If MsgBox(Text, vbYesNo) = vbYes Then
         If Not Adressen_Erstellen(FileName) Then
            Text = " Datei " & FileName & " wurde nicht erstellt!"
            MsgBox Text
```

```
            Exit Sub
         Else
            If Adressen_Öffnen(FileName, Len(Data)) = False Then
               Text = "Datei " & FileName & _
                  " wurde nicht erstellen?"
               MsgBox Text
               Exit Sub
            End If
         End If
      Else
         Exit Sub
      End If
   End If

   Key = Right("000" & Trim(Str(Val(Cells(1, 2)))), 3)
   If Adressen_Index(Key, Dat_Max, Dat_Rec, Data) = True Then
      Cells(2, 2) = Data.Name
      Cells(3, 2) = Data.Vorname
      Cells(4, 2) = Data.Strasse
      Cells(5, 2) = Data.PLZ
      Cells(6, 2) = Data.Ort
   Else
      Cells(2, 2) = ""
      Cells(3, 2) = ""
      Cells(4, 2) = ""
      Cells(5, 2) = ""
      Cells(6, 2) = ""
      Text = "Datensatz nicht gefunden!"
      MsgBox Text
   End If
   Close Dat_Num
End Sub
```

Die Löschprozedur erwartet den zu löschenden Schlüssel in B1. Wird der Schlüssel gefunden, wird der Datensatz entfernt.

Codeliste 17. Ereignisprozedur Löschen

```
Private Sub cbnLöschen_Click()
   Dim FileName   As String
   Dim Data       As Adressen
   Dim Text       As String
   Dim Dat_Rec    As Long
   Dim Dat_Txt    As String
   Dim Key        As String
   Dim Nummer     As String

   FileName = "C:\Temp\Random\Daten.dat"
   If Adressen_Öffnen(FileName, Len(Data)) = False Then
      Text = "Datei " & FileName & " existiert nicht!"
      Exit Sub
   End If

   Key = Right("000" & Trim(Str(Val(Cells(1, 2)))), 3)
   If Adressen_Index(Key, Dat_Max, Dat_Rec, Data) = True Then
      If Adressen_Entfernen(Dat_Rec, Dat_Max) = False Then
         Text = "Datensatz wurde nicht gelöscht!"
         MsgBox Text
      Else
         Text = "Datensatz gelöscht!"
```

```
         MsgBox Text
      End If
   Else
      Text = "Schlüssel nicht vorhanden!"
      MsgBox Text
   End If
End Sub
```

Die Listenprozedur erstellt im Arbeitsblatt Liste eine Übersicht aller vorhandenen Datensätze.

Codeliste 18. Ereignisprozedur Liste erstellen

```
Private Sub cbnListe_Click()
   Dim wshListe   As Worksheet
   Dim FileName   As String
   Dim Data       As Adressen
   Dim lCount     As Long

   Set wshListe = Worksheets("Liste")
   wshListe.Activate
   wshListe.Cells.Clear
   FileName = "C:\Temp\Random\Daten.dat"
   If Adressen_Öffnen(FileName, Len(Data)) = True Then
      For lCount = 1 To Dat_Max
         Get Dat_Num, lCount, Data
         With wshListe
            .Cells(lCount, 1) = Data.Nummer
            .Cells(lCount, 2) = Data.Name
            .Cells(lCount, 3) = Data.Vorname
            .Cells(lCount, 4) = Data.Strasse
            .Cells(lCount, 5) = Data.PLZ
            .Cells(lCount, 6) = Data.Ort
         End With
      Next lCount
      wshListe.Range("A:F").Columns.AutoFit
   Else
      MsgBox "Lesevorgang abgebrochen!"
   End If
   Close Dat_Num
End Sub
```

Tests zeigen das folgende Ergebnis (Bild 11).

	A	B	C	D	E	F	G	H
1	9	Name009	Vorname009	Strasse009	D-PLZ009	Wohnort009	Formular	
2	11	Name011	Vorname011	Strasse011	D-PLZ011	Wohnort011		
3	20	Name020	Vorname020	Strasse020	D-PLZ020	Wohnort020		
4								

Bild 11. Testergebnisse

Das Arbeitsblatt der Liste enthält eine Schaltfläche Formular, mit der über die nachfolgende Event-Prozedur zurück zur Formularanzeige geschaltet werden kann. Die Prozedur beinhaltet auch, dass der markierte Datensatz (oder ein Element davon) beim Umschalten auf das Formular darin abgebildet wird. So lassen sich Änderungen am Datensatz, die Kopie eines Datensatzes oder dessen Löschung relativ schnell durchführen.

7 Formulare als UserForm

Natürlich lässt sich das Handling der Daten noch verbessern. Zum Beispiel, indem das Formular als Userform erstellt wird, die bei der Aktivierung der Liste automatisch eingeblendet wird.

Normalisierung und ER-Modell

Das relationale Datenmodell wurde 1970 von dem Mathematiker Codd entwickelt und mithilfe der Mengentheorie beschrieben. Dieses Modell bildet die Basis für Relationale Datenbanken.

Eine Relation ist in der DB-Sprache eine Beziehung zwischen Tabellen mit Daten in Spalten und Zeilen, die Informationen zu bestimmten Objekten beinhaltet. Genauso ist es auch bei den Excel-Tabellen.

Eine Tabelle (Bild 12) ist gekennzeichnet durch:

- Einen eindeutigen Namen. Auch eine Excel-Tabelle besitzt in einer Arbeitsmappe einen eindeutigen Namen.
- Die Tabelle hat mindestens eine, meistens mehrere Attribute (Spalten).
- Eine Tabelle hat keine bis beliebig viele Datensätze (Zeilen).
- Den Schnittpunkt von Zeilen und Spalten bilden die Attribute (Zellen).
- Einen Primärschlüssel, der jeden Datensatz eindeutig identifiziert, und dessen Wert sich während der Existenz des Datensatzes nicht ändert. Bei einer Excel-Tabelle ist dies im einfachsten Fall die Zeilennummer.
- Keine oder mehrere Sekundärschlüssel, die auf Primärschlüssel verweisen.

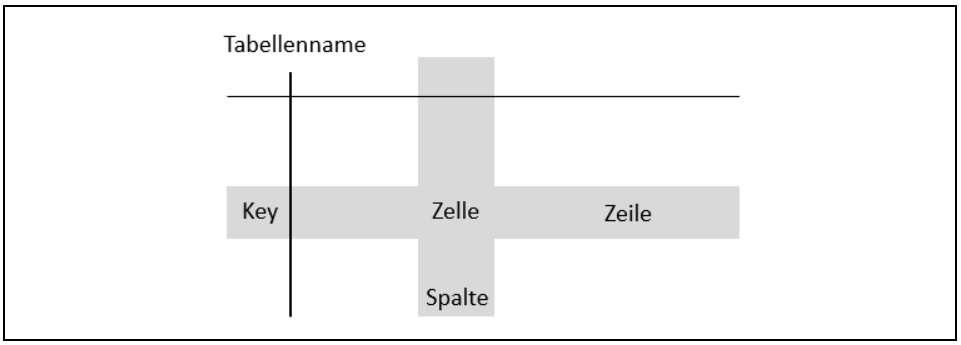

Bild 12. Tabellenstruktur