

Tutorial

4

EDV INNOVATION & CONSULTING

Manuela Kulpa

Gerhard-Domagk-Str. 6
D-41540 Dormagen
<http://www.kulpa-online.com>
Email: stefan@kulpa-online.com

**STRUKTUR-
ÄNDERUNG
AN DER DATENBANK**

Inhaltsverzeichnis

WESHALB SOLL/MUSS DIE STRUKTUR GEÄNDERT WERDEN?	4
Ein paar Worte zum Anfang!.....	4
Vorbereitung zum Strukturupdate.....	5
Benutzerüberprüfung.....	5
Existiert eine Tabelle	7
TABELLEN ANLEGEN UND MODIFIZIEREN.....	9
Tabelle anlegen	9
DAO-Variante	10
ADOX-Variante	12
SQL-DDL-Variante.....	14
TABELLE LÖSCHEN.....	17
DAO-Variante	17
ADOX-Variante	17
SQL-DDL-Variante.....	18
FELDBEARBEITUNG	20
Existiert ein Feld.....	20
DAO-Variante	20
ADOX-Variante	20
Stimmt die Feldgröße eines Textfeldes.....	20
DAO-Variante	21
ADOX-Variante	21
Welchen Felddatentyp besitzt das Tabellenfeld	21
DAO-Variante	22
ADOX-Variante	22
Feld anlegen.....	22
DAO-Variante	23
ADOX-Variante	23
SQL-DDL-Variante.....	24
Feld löschen	25
DAO-Variante	26
ADOX-Variante	26
SQL-DDL-Variante.....	26
Feldname ändern	27
DAO-Variante	27
ADOX-Variante	27
Feldgröße/-typ ändern.....	27
DAO-Variante	28
ADOX-Variante	29
SQL-DDL-Variante (ab Jet 4.0).....	30
INDIZES ANLEGEN/MODIFIZIEREN.....	32
Index prüfen.....	32
DAO-Variante	32
ADOX-Variante	32
Index setzen (mit Duplikate / ohne Duplikate / Primärschlüssel).....	33

DAO-Variante	33
ADOX-Variante	34
SQL-DLL-Variante.....	35
Index löschen	37
DAO-Variante	37
ADOX-Variante	37
SQL-DLL-Variante.....	38
REFERENZINTEGRITÄTEN UND BEZIEHUNGEN	39
Beziehungen erstellen.....	39
DAO-Variante	39
ADOX-Variante	41
SQL-DDL-Variante.....	42
Beziehungen prüfen.....	43
DAO-Variante	44
ADOX-Variante	44
Beziehungen löschen	44
DAO-Variante	44
ADOX-Variante	45
SQL-DDL-Variante.....	45

WESHALB SOLL/MUSS DIE STRUKTUR GEÄNDERT WERDEN?

Ein paar Worte zum Anfang!

Ziel dieses Workshops ist, Ihren Wissensschatz in punkto Strukturänderungen von Datenbanken zu erweitern. Wir gehen davon aus, dass Sie bereits Erfahrungen mit VB/A gesammelt haben und Ihnen die Konzepte relationaler Datenbanken nichts Neues sind; von SQL und Datenzugriffsobjekten sollten Sie zumindest schon einmal etwas gehört haben.

In den einzelnen Kapiteln finden Sie fast alles, was Sie zur Strukturänderungen von Datenbanken in der Eigenentwicklung benötigen. Gerade weil sich das Thema nur innerhalb gewisser Grenzen schrittweise abhandeln lässt, haben wir viel Arbeit in eine mögliche lineare Folge der einzelnen Kapitel investiert. Anders gesagt: Die einzelnen Kapitel bauen weitgehend aufeinander auf.

Dieser Workshop enthält sehr viele Beispiele. Damit Sie auch die Möglichkeit haben, die Beispiele praxisnah auszuprobieren, fügen wir diesem Workshop die Beispiele als Access 2000 bzw. Access 97 Datenbank zu. In der Access 97 Datenbank haben wir auf die ADO/X-Beispiele verzichtet.

Berücksichtigen Sie bitte, dass Sie auf Grund von Verweis- bzw. Konvertierungs-Problemen ggf. die Verweise neu setzen müssen. Für die Beispiele benötigen Sie nachstehende Verweise in der jeweils höchsten Version (die Ihnen vorliegt):

- Microsoft DAO 3.X Library
- Microsoft ADO Ext. 2.X
- Microsoft ActiveX Data Objects 2.X Library

Kunden arbeiten mit Ihrer Anwendung und Sie haben ein Update, bei welchem die Struktur der Tabellen innerhalb des Backends geändert werden müssen. Was nun? Eine Lösung muss her, aber wie sieht diese aus? Das erfahren Sie in unserem Workshop.

Nachfolgendes Szenario ist sicherlich vielen bereits mehr als bekannt. Eine Anwendung, die Sie in Access oder Visual Basic erstellt haben ist mehrfach bei Kunden bzw. Anwendern im Einsatz. Sie haben die Anwendung erweitert. Neue Funktionen und Programmteile benötigen zusätzliche Tabellen bzw. Tabellenfelder. Unter Umständen muss die Feldgröße geändert

oder sogar der Feldtyp geändert werden. Oft kommt es auch vor, dass nachträglich ein Index gesetzt, geändert oder gelöscht werden muss. Bestimmt haben Sie nicht bei allen Kunden die Möglichkeit, das Backend per Remote (DFÜ) anzupassen. Zudem ist es auch sehr zeitintensiv und je mehr Kunden und Anwender Sie haben, desto größer ist der Aufwand. Sicherlich möchten Sie Ihre Kunden bei einem Update Ihrer Anwendung auch nicht immer persönlich aufsuchen um manuell die Anpassungen vorzunehmen.

Was bleibt ist eine automatische Prüfung und Anpassung per Code mit VB bzw. VBA. Prüfung deshalb, da verschiedene Kunden auch verschiedene Versionsstände einsetzen. Die Erfahrung zeigt, dass nicht jeder Kunde auch jedes Update installiert bzw. verwendet. Sie müssen daher immer prüfen, ob die Tabelle bzw. das Feld bereits vorhanden und sie bzw. es ggf. anlegen. Bei Änderung der Feldgröße muss zunächst geprüft werden, ob das Feld nicht bereits angepasst wurde.

Für jede Tabelle und jedes Feld müssen Sie zunächst eine Prüfung durchlaufen und danach die Datenbank anpassen. Für die verschiedenen Anpassungsmöglichkeiten geben wir Ihnen in diesem Workshop verschiedene Funktionen zur Hand. Da das Ändern **immer über DAO** (Data Access Objects) erfolgt und es verschiedene Versionen gibt, zeigen wir immer beide Methoden. Access 97 läuft mit der Jet-Engine 3.5x und kennt nicht alle Befehle der Jet-Version 4.0 (ab Access 2000). Access 97 bzw. die von dieser Access-Version verwendete Jet-Engine kennt zwar manche Befehle von **Db.Execute**, aber der volle Befehlsumfang steht erst ab Access 2000 zur Verfügung.

Vorbereitung zum Strukturupdate

Bevor Sie die Datenbank anpacken und updaten, sollten Sie sicher sein, dass nur ein Anwender, nämlich der, der das Update gerade aufruft, auf die Datenbank bzw. das Backend zugreift. Arbeiten mehrere Anwender mit der Datenbank kann entweder diese nicht modifiziert werden oder es kann zu Fehlern kommen. Für jede Datenbank legt die Jet-Engine, sobald die Datei (MDB/MDE) in Benutzung ist, eine LDB-Datei an. Die Datei trägt den gleichen Dateinamen wie die Datenbank. Anstatt der Endung MDB bzw. MDE ist hier das Präfix LDB. In dieser Datei speichert die Jet-Engine alle Benutzer und Hostnamen, die auf die Datenbank zugreifen. Mit den folgenden Codezeilen kann ermittelt werden, wie viele Anwender auf eine Datenbank zugreifen. Möchten Sie eine Strukturänderung durchführen, sollten Sie sicher sein, dass nicht mehr als ein Anwender auf diese Datenbank zugreift.

Zunächst wird ein neuer Datentyp angelegt. In diesen werden später die Informationen, die aus der LDB-Datei ausgelesen werden, abgelegt.

Benutzerüberprüfung

```
Private Type mtypLdbUserInfo
    Computername As String * 32
    UserName As String * 32
End Type
```

Die Funktion CheckUserCount liefert die Zahl der Anwender dieser Datenbank zurück, die sich in ihr befinden oder befanden. Es spielt dabei keine Rolle, ob der Anwender die Datenbank direkt geöffnet hat oder über „verknüpfte Tabellen“ darauf zugreift. Liefert die Funktion einen größeren Wert als 1 zurück, sollte kein Update erfolgen.

Hinweis: Wenn ein User aus der DB rausgeht, so wird der entsprechende Eintrag in der LDB nicht gelöscht! Für den sicheren Vollzug der DB-Strukturänderungen reicht es unter Umständen, nur die Existenz der LDB abzufragen. Wenn sie nicht existiert, kann's gemacht werden, wenn doch, dann nicht. (Auch wenn kein User in der DB ist, die LDB aber trotzdem existiert, z.B., weil Access abstürzte, sollten die Strukturänderungen nicht ablaufen, weil das ein Hinweis darauf sein könnte, dass die DB beschädigt ist oder zumindest erst mit CompactRepair behandelt werden sollte.)

```
Public Function CheckUserCount(ByVal psDBName As String) As Long

    Dim F           As Integer
    Dim lNUsers     As Long
    Dim lI          As Long
    Dim UserInfo    As mtypLdbUserInfo

    If LCase(Right(psDBName, 4)) = ".mdb" Then
        psDBName = Left(psDBName, Len(psDBName) - 4) + ".ldb"
    End If

    If CheckFile(psDBName) Then
        F = FreeFile
        Open psDBName For Random Shared As #F Len = Len(UserInfo)
        lNUsers = Int(LOF(F) / Len(UserInfo))
        Close #F
    End If

    CheckUserCount = lNUsers

End Function
```

In der Funktion **CheckUserCount** wird eine weitere Funktion **CheckFile** aufgerufen. Sie können diese Prüfung zwar beiseite lassen, aber es empfiehlt sich immer, alle Fehlerquellen ab-zuprüfen. Es kann sein, dass die Datenbank von einer CD exklusiv und schreibgeschützt ge-öffnet wurde. In diesem Fall wird keine LDB-Datei im Datenbankverzeichnis angelegt. Ohne Prüfung auf die existierende Datei würde ein Programm- bzw. Anwenderfehler auftreten.

Wenn man eine externe Datenbank modifiziert, sollte man auch prüfen, ob diese Datei existiert. Diese Prüfung kann man ebenfalls mit der Funktion **CheckFile** vornehmen. Nachfolgend die Funktion zum Prüfen auf eine existierende Datei. Als Ergebnis wird **True** geliefert, wenn die Datei vorhanden ist. Wenn die Datei nicht gefunden ist das Ergebnis **False**.

1. Möglichkeit über VBA

```
Public Function CheckFile(ByVal psFileName As String) As Boolean

    CheckFile = (Len(VBA.Dir(psFileName)) > 0)

End Function
```

2. Möglichkeit über API

```
' Deklarationen und Typen
Const mclAXPATH          As Long = 260

Type gtypFILETIME
    dwLowDateTime        As Long
    dwHighDateTime       As Long
End Type
Type gtypWIN32_FIND_DATA
    dwFileAttributes     As Long
    ftCreationTime       As gtypFILETIME
    ftLastAccessTime     As gtypFILETIME
    ftLastWriteTime      As gtypFILETIME
    nFileSizeHigh        As Long
    nFileSizeLow         As Long
    dwReserved0          As Long
    dwReserved1          As Long
    cFileName            As String * mclAXPATH
    cAlternate           As String * 14
End Type
Declare Function FindFirstFile Lib "kernel32" Alias _
    "FindFirstFileA" _
    (ByVal lpFileName As String, _
    lpFindFileData As gtypWIN32_FIND_DATA) _
    As Long

Declare Function FindClose Lib "kernel32" _
    (ByVal hFindFile As Long) _
    As Long

Public Function APIFileExists(ByVal psSource As String) As Boolean

    Const cINVALID_HANDLE_VALUE As Long = -1
    Dim WFD As gtypWIN32_FIND_DATA
    Dim lFile As Long

    lFile = FindFirstFile(psSource, WFD)
    '// Prüfung auf gültigen Datei-Handle
    APIFileExists = lFile <> cINVALID_HANDLE_VALUE
    Call FindClose(lFile)

End Function
```

Existiert eine Tabelle

Zum Prüfen, ob eine Tabelle bereits existiert gehen Sie wie folgt vor:

1. Variante DAO

```
Public Function TableExistsDAO(pDb As DAO.Database, _
    ByVal psName As String) _
    As Boolean

    Dim s As String

    On Error Resume Next
    s = pDb.TableDefs(psName).Name
    TableExistsDAO = (Err.Number = 0)

End Function
```

2. Variante ADOX

```
Public Function TableExistsADOX(pcnn As ADODB.Connection, _  
                                ByVal psName As String) _  
                                As Boolean  
  
    Dim s As String  
    Dim cat As New ADOX.Catalog  
  
    On Error Resume Next  
    cat.ActiveConnection = pcnn  
    s = cat.Tables(psName).Name  
    TableExistsADOX = (Err.Number = 0)  
    If Not cat Is Nothing Then Set cat = Nothing  
  
End Function
```

TABELLEN ANLEGEN UND MODIFIZIEREN

Sie haben mehrere Möglichkeiten eine Tabelle anzulegen (über DAO, ADOX und SQL-DDL). Man kann nicht unbedingt eine Standardroutine verwenden um eine Tabelle anzulegen, da beim Anlegen immer auch sämtliche Felder mit angegeben werden sollten. Da sich jede Tabelle von der anderen unterscheidet, sieht der Code zum Erstellen auch sehr individuell aus.

Tabelle anlegen

Das Erstellen einer Tabelle erfolgt mit DAO oder ADOX auf dieselbe Weise. Zuerst wird das Objekt erstellt (**TableDef** oder **Table**), dann werden die Spalten hinzugefügt (**Field**- oder **Column**-Objekte), und abschließend wird die Tabelle der Auflistung hinzugefügt. Obwohl der Ablauf gleich ist, weicht die Syntax leicht ab.

Bei ADOX ist es nicht notwendig, eine Methode zum Erstellen der Spalte zu verwenden, bevor diese zur Auflistung hinzugefügt wird. Die **Append**-Methode kann sowohl zum Erstellen als auch zum Anhängen der Spalte verwendet werden. Leider sind die Datentypennamen für die Spalten in DAO, ADOX und SQL unterschiedlich. In der folgenden Tabelle ist aufgeführt, welche für Microsoft Jet-Datenbanken anwendbare DAO-Datentypen den ADO-Datentypen entsprechen.

DAO-Datentyp	ADO-Datentyp	SQL
dbBinary	adBinary	BINARY
dbBoolean	adBoolean	BIT
dbByte	adUnsignedTinyInt	BYTE
dbCurrency	adCurrency	CURRENCY
dbDate	adDate	DATE
dbDecimal	adNumeric	DECIMAL
dbDouble	adDouble	DOUBLE
dbGUID	adGUID	GUID
dbInteger	adSmallInt	SMALLINT
dbLong	adInteger	INT
dbLongBinary	adLongVarBinary	LONGBINARY
dbMemo	adLongVarChar	MEMO
dbSingle	adSingle	SINGLE
dbText	adVarChar	TEXT

Detaillierte Informationen über alle möglichen Feldtypen/-größen finden Sie in der Online-Hilfe von Visual-Basic bzw. Access.

Obwohl in den unteren Beispielen nicht immer dargestellt ist, gibt es für Tabellen oder Spalten eine Reihe weiterer Attribute, die beim Erstellen einer Tabelle oder Spalte mit der DAO **Attributes**-Eigenschaft eingestellt werden können. In der Tabelle unten ist dargestellt, wie diese Attribute den ADO- und Microsoft Jet Provider-spezifischen Eigenschaften zugeordnet werden.

DAO-TableDef	Wert	ADOX-Table	Wert
Attribute	dbAttachExclusive	Jet OLEDB:Exclusive Link	True
Attribute	dbAttachSavePWD	Jet OLEDB:Cache Link Name/Password	True
Attribute	dbAttachedTable	Type	"LINK"
Attribute	dbAttachedODBC	Type	"PASS-THROUGH"
Attribute	dbAutoIncrField	AutoIncrement	True
Attribute	dbFixedField	ColumnAttributes	adColFixed
Attribute	dbHyperlinkField	Jet OLEDB:Hyperlink	True
Attribute	dbSystemField	Keine Entsprechung	–
Attribute	dbUpdatableField	Attributes (Field-Objekt)	adFldUpdatable
Attribute	dbVariableField	ColumnAttributes	Not adColFixed

DAO-Variante

In nachstehenden Beispiel wird eine neue Tabelle mit allen möglichen Datentypen (falls nicht vorhanden) erstellt. Exemplarisch wird auch aufgeführt, wie man ein **AutoWert-Feld** erstellt und wie man die diversen Eigenschaften wie z.B. **Standardwert**, **"Eingabe erforderlich"**, **"Leere Zeichenfolge"** und **Hyperlink** setzen kann.

Es gibt zwar noch mehr Feldtypen, aber diese sind nur in der Zusammenarbeit mit der MSDE bzw. dem SQL-Server über ADO zulässig und können über diese Funktionen nicht angelegt werden.

```
Public Function DAO_CreatedNewTable(pDbs As DAO.Database) As Boolean

    On Error GoTo HandleErr

    '// Die benötigten Objektvariablen
    '// ein Verweis auf Microsoft DAO 3.X Object Library
    '// muss gesetzt sein
    Dim tdef As DAO.TableDef
    Dim tfld As DAO.Field
    Dim tidx As DAO.Index

    '// Initialisierung
    DAO_CreatedNewTable = False

    '// Ist die Tabelle bereits vorhanden
    If TableExistsDAO(pDbs, "NewTableDao") = False Then
```

```

'// Erstellung der neuen Tabelle
Set tdef = pDbs.CreateTableDef("NewTableDao")

With tdef
'// neues AutoWert-Feld erstellen
Set tfld = .CreateField("fAutoWert", dbLong)
' z.B. AutoWert-Eigenschaft setzen
tfld.Attributes = tfld.Attributes Or dbAutoIncrField
' Fügt das Feld in die Tabelle hinzu.
.Fields.Append tfld
'// *****

'// neues Byte-Feld erstellen
Set tfld = .CreateField("fByte", dbByte)
' z.B. Standardwert hinzufügen
tfld.DefaultValue = 3
.Fields.Append tfld
'// *****

'// neues Integer-Feld erstellen
Set tfld = .CreateField("fInt", dbInteger)
' z.B. Eingabe erforderlich: Ja
tfld.Required = True
.Fields.Append tfld
'// *****

'// neues Long-Feld erstellen
Set tfld = .CreateField("fLong", dbLong)
.Fields.Append tfld
'// *****

'// neues Single-Feld erstellen
Set tfld = .CreateField("fSingle", dbSingle)
.Fields.Append tfld
'// *****

'// neues Double-Feld erstellen
Set tfld = .CreateField("fDouble", dbDouble)
.Fields.Append tfld
'// *****

'// neues Währungsfeld erstellen
Set tfld = .CreateField("fCurrency", dbCurrency)
.Fields.Append tfld
'// *****

'// Neues Datums-Feld erstellen
Set tfld = .CreateField("fDateTime", dbDate)
.Fields.Append tfld
'// *****

'// Neues Replikations-ID-Feld erstellen
Set tfld = .CreateField("fGUID", dbGUID)
.Fields.Append tfld
'// *****

'// Neues Ja/Nein-Feld erstellen
Set tfld = .CreateField("fBoolean", dbBoolean)
.Fields.Append tfld
'// *****

'// Neues Text-Feld erstellen
Set tfld = .CreateField("fText", dbText)
With tfld
' z.B. Feldgröße
.Size = 30

```

```

        ' z.B. Leere Zeichenfolge: Ja
        .AllowZeroLength = True
    End With
    .Fields.Append tfld
    '// *****

    '// Neues Memo-Feld erstellen
    Set tfld = .CreateField("fMemo", dbMemo)
    tfld.Attributes = dbVariableField
    .Fields.Append tfld
    '// *****

    '// Neues Hyperlink-Feld erstellen (erst ab Jet 4.0)
    Set tfld = .CreateField("fLink", dbMemo)
    With tfld
        ' Hyperlink-Eigenschaft festlegen
        .Attributes = dbHyperlinkField
        .AllowZeroLength = True
    End With
    .Fields.Append tfld
    '// *****

    '// Neues Blob-Feld mit fester Feldgröße erstellen
    Set tfld = .CreateField("fBinary", dbBinary)
    .Fields.Append tfld
    '// *****

    ' // Neues Ole-Objekt-Feld erstellen
    Set tfld = .CreateField("fOle", dbLongBinary)
    .Fields.Append tfld
    '// *****

    End With

    '// Neue Tabelle in der Datenbank erstellen
    pDbs.TableDefs.Append tdef
    DAO_CreatedNewTable = True
End If

HandleExit:
    '// Speicher freigeben
    If Not tdef Is Nothing Then Set tdef = Nothing
    If Not tfld Is Nothing Then Set tfld = Nothing
    Exit Function

HandleErr:
    Select Case Err.Number
        Case Else
            MsgBox "Fehler " & Err.Number & ": " & _
                Err.Description, vbCritical, _
                "modKap02.DAO_CreatedNewTable"
    End Select
    DAO_CreatedNewTable = False
    Resume HandleExit
End Function

```

ADOX-Variante

Nun das gleiche Beispiel mit der ADOX-Bibliothek. Beachten Sie dabei die Zeile, in der die Eigenschaft **ParentCatalog** gesetzt wird. Diese Eigenschaft gibt den übergeordneten Katalog einer Tabelle oder Spalte an, um den Zugriff auf providerspezifische Eigenschaften bereitzustellen.

Ein kleiner Hinweis zur Bibliothek! Zwei Bereiche werden von ADO nicht abgedeckt. Erstellung von Datenbanken, Tabellen und Abfragen sowie Sicherheitsfunktionen. Für das Fehlen gibt es zwei Gründe: Erstens sind beide Bereiche durch SQL-Befehle abgedeckt (s.u.), die über Command- oder Connection-Objekte ausgeführt werden können und zweitens sind die Unterschiede zwischen Datenbanksystemen gerade in diesen Bereichen am größten.

Microsoft hat zusätzlich die ADOX-Bibliothek entwickelt, die einen einfachen, vom Datenbankprodukt unabhängigen Zugriff auf Datendefinitionen- und Sicherheitsfunktionen ermöglichen sollen. Voraussetzung für den Einsatz von ADOX ist allerdings, dass der OLE-DB-Provider die entsprechende Funktionalität bereitstellt. Allgemein ist es jedoch so, dass nur der Jet 4.0 OLE-DB-Provider fast alle ADOX Funktionen unterstützt. Letztendlich bedeutet dies, dass ADOX eigentlich nur für Access-Datenbanken eingesetzt werden kann.

```
Function ADO_CreatedNewTable(pcnn As ADODB.Connection) As Boolean

    '// Die benötigten Objektvariablen
    '// ein Verweis auf Microsoft ADO Ext. 2.X
    '// for DDL and Security muss gesetzt sein
    On Error GoTo HandleErr
    Dim cat As New ADOX.Catalog
    Dim tbl As New ADOX.Table
    Dim col As New ADOX.Column

    '// Initialisierung
    ADO_CreatedNewTable = False

    cat.ActiveConnection = pcnn
    '// Ist die Tabelle bereits vorhanden
    If TableExistsADOX(cat, "NewTableAdo") = False Then
        With tbl
            ' Diese Zeile ist notwendig,
            ' damit die Datenbankspezifischen
            ' Eigenschaften wie z.B. AutoWert
            ' etc. pp. gesetzt werden können
            .ParentCatalog = cat

            '// Erstellung der neuen Tabelle
            .Name = "NewTableAdo"

            '// neue Felder hinzufügen s.o.
            .Columns.Append "fAutoWert", adInteger
            ' z.B. AutoWert-Eigenschaft setzen
            Set col = tbl.Columns("fAutoWert")
            col.Properties("Autoincrement") = True

            '// Neues Byte-Feld hinzufügen
            .Columns.Append "fByte", adUnsignedTinyInt
            Set col = .Columns("fByte")
            ' z.B. Standardwert hinzufügen
            col.Properties("Default") = 3

            .Columns.Append "fInt", adSmallInt
            .Columns.Append "fLong", adInteger

            Set col = .Columns("fLong")
            ' Entgegen zu DAO wird bei der Erstellung
            ' von Feldern die Required grundsätzlich
            ' immer auf True gesetzt. Deswegen, sollte
            ' man darauf achten dies ggf. zu ändern
            ' z.B. Eingabe erforderlich = nein
            col.Properties("Nullable").Value = True
        End With
    End If
End Function
```

```

.Columns.Append "fSingle", adSingle
.Columns.Append "fDouble", adDouble
.Columns.Append "fCurrency", adCurrency
.Columns.Append "fDateTime", adDate
.Columns.Append "fGUID", adGUID
.Columns.Append "fBoolean", adBoolean

'// Neues Textfeld erstellen
.Columns.Append "fText", adWChar
Set col = .Columns("fText")
With col
' z.B. Feldgröße setzen
.DefinedSize = 30
' z.B. Leere Zeichenfolge setzen
col.Properties("Jet OLEDB:Allow Zero Length") = True
End With

.Columns.Append "fMemo", adLongVarChar

'// Neues Hyperlink-Feld erstellen (erst ab Jet 4.0)
.Columns.Append "fLink", adLongVarChar
Set col = .Columns("fLink")
With col
' z.B. Leere Zeichenfolge setzen
.Properties("Jet OLEDB:Allow Zero Length") = True
' z.B. Hyperlink-Eigenschaft setzen
.Properties("Jet OLEDB:Hyperlink") = True
End With

.Columns.Append "fBinary", adVarBinary
.Columns.Append "fOle", adLongVarBinary

End With
cat.Tables.Append tbl
ADO_CreatedNewTable = True
End If

HandleExit:
'// Speicher freigeben
If Not col Is Nothing Then Set col = Nothing
If Not tbl Is Nothing Then Set tbl = Nothing
If Not cat Is Nothing Then Set cat = Nothing
Exit Function

HandleErr:
Select Case Err.Number
Case Else
MsgBox "Fehler " & Err.Number & ": " & _
Err.Description, vbCritical, _
"modKap02.ADO_CreatedNewTable"
End Select
ADO_CreatedNewTable = False
Resume HandleExit
End Function

```

SQL-DDL-Variante

Es gibt verschiedene Wege, um über SQL-DDL-Anweisungen mit Microsoft Jet zu kommunizieren. Diese Möglichkeit bedient sich der Execute-Methode bei der SQL-Anweisungen übergeben werden. Dabei wird das Common DDL (Data Definition Language) SQL Statement verwendet. Man sollte dabei erwähnen, dass zum Anlegen von Tabellen einige Möglichkeiten zum Einstellen von Feldeigenschaften fehlen und daher die Varianten DAO/ADOX empfeh-

lenswerter sind. Die Execute-Methode ist bei anderen Aktionen (z.B. ändern der Feldgröße) einfacher anzuwenden, als die herkömmliche Möglichkeit über TableDefs bzw. Tables.

Wenn Sie eine neue Tabelle mit einem SQL-DDL-Kommando anlegen, muss die einfachste Formulierung folgendem Muster genügen: **CREATE TABLE MeineTabelle (MeinTextfeld TEXT)**. Und nun auf unser o.g. Beispiel bezogen:

```

'// DAO-Variante
Public Function DDL_CreateNewTableDAO(pdb As DAO.Database) As Boolean

    On Error GoTo HandleErr
    Dim sSql As String

    '// Initialisierung
    DDL_CreateNewTableDAO = False

    '// Die SQL-DDL Anweisung
    sSql = "CREATE TABLE NewTableSQL ( fAutoWert COUNTER, fByte BYTE " & _
        "NOT NULL, fInt SMALLINT, fLong INT, fSingle SINGLE, " & _
        "fDouble DOUBLE, fCurrency CURRENCY, fDateTime DATE, " & _
        "fGUID GUID, fBoolean BIT, fText CHAR (30), fMemo MEMO, " & _
        "fBinary BINARY, fOLE LONGBINARY );"

    '// Falls die Tabelle noch nicht vorhanden ist
    If TableExistsDAO(pdb, "NewTableSQL") = False Then
        pdb.Execute sSql, dbFailOnError
        DDL_CreateNewTableDAO = True
    End If

HandleExit:
    Exit Function

HandleErr:
    Select Case Err.Number
        Case Else
            MsgBox "Fehler " & Err.Number & ": " & _
                Err.Description, vbCritical, _
                "modKap02.DDL_CreateNewTableDAO"
    End Select
    DDL_CreateNewTableDAO = False
    Resume HandleExit
End Function

'// ADO-Variante
Public Function DDL_CreateNewTableADO(pcnn As ADO.Connection) As Boolean

    On Error GoTo HandleErr
    Dim sSQL As String

    '// Initialisierung
    DDL_CreateNewTableADO = False

    '// Die SQL-DDL Anweisung
    '// unter ADO-SQL-DDL ist es auch möglich, einen
    '// Standardwert anzugeben (s. fByte)
    sSQL = "CREATE TABLE NewTableSQL ( fAutoWert COUNTER, fByte BYTE DEFAULT 3
" & _
        "NOT NULL, fInt SMALLINT, fLong INT, fSingle SINGLE, " & _
        "fDouble DOUBLE, fCurrency CURRENCY, fDateTime DATE, " & _
        "fGUID GUID, fBoolean BIT, fText CHAR (30), fMemo MEMO, " & _
        "fBinary BINARY, fOLE LONGBINARY );"

    '// Falls die Tabelle noch nicht vorhanden ist
    If TableExistsADOX(pcnn, "NewTableSQL") = False Then
        pcnn.Execute sSQL, dbFailOnError
        DDL_CreateNewTableADO = True
    End If

```

```
HandleExit:
    Exit Function

HandleErr:
    Select Case Err.Number
        Case Else
            MsgBox "Fehler " & Err.Number & ": " & _
                Err.Description, vbCritical, _
                "modKap02.DDL_CreateNewTableADO"
    End Select
    Resume HandleExit
End Function
```

TABELLE LÖSCHEN

Zum Löschen einer Tabelle gibt es ebenfalls mehrere Möglichkeiten. Bei allen Varianten werden als Parameter die Verbindung zur Database und der Tabellename übergeben. Bitte berücksichtigen Sie, falls vorhanden, die Gewährleistung der referentiellen Integrität!

DAO-Variante

Wenn Sie eine Tabelle mit DAO löschen wollen, müssen Sie das dazugehörige TableDef-Objekt aus der TableDefs-Auflistung entfernen.

```
Public Function DAO_DeleteTable(pdb As DAO.Database, psTable As String) As Boolean

    On Error GoTo HandleErr

    If TableExistsDAO(pdb, psTable) Then
        pdb.TableDefs.Delete psTable
        DAO_DeleteTable = True
    End If

HandleExit:
    Exit Function

HandleErr:
    Select Case Err.Number
        Case Else
            MsgBox "Fehler " & Err.Number & ": " & _
                Err.Description, vbCritical, _
                "modKap02.DAO_DeleteTable"
    End Select
    DAO_DeleteTable = False
    Resume HandleExit
End Function
```

ADOX-Variante

Hier beziehen Sie sich wieder auf das Catalog-Objekt und der entsprechenden Tables-Auflistung.

```
Public Function ADOX_DeleteTable(pcnn As ADODB.Connection, psTable As String)

    On Error GoTo HandleErr
    Dim cat As New ADOX.Catalog

    If TableExistsADOX(pcnn, psTable) Then
```

```

        cat.ActiveConnection = pcnn
        cat.Tables.Delete psTable
        ADO_DeleteTable = True
    End If

HandleExit:
    If Not cat Is Nothing Then Set cat = Nothing
    Exit Function

HandleErr:
    Select Case Err.Number
        Case Else
            MsgBox "Fehler " & Err.Number & ": " & _
                Err.Description, vbCritical, _
                "modKap02.ADO_DeleteTable"
    End Select
    ADO_DeleteTable = False
    Resume HandleExit
End Function

```

SQL-DDL-Variante

Mit Hilfe der **DROP TABLE**-Anweisung können Sie eine existierende Tabelle aus der Datenbank entfernen. Hierzu müssen Sie lediglich noch den Namen der zu löschenden Tabelle kennen, den Sie der der Anweisung spezifizieren müssen.

```

'// DAO-Variante
Public Function DLL_DeleteTableDao(pdb As DAO.Database, psTable As
String) As Boolean

    On Error GoTo HandleErr
    Dim sSQL As String

    If TableExistsDAO(pdb, psTable) Then
        sSQL = "DROP TABLE " & psTable
        pdb.Execute sSQL, dbFailOnError
        DLL_DeleteTableDao = True
    End If

HandleExit:
    Exit Function

HandleErr:
    Select Case Err.Number
        Case Else
            MsgBox "Fehler " & Err.Number & ": " & _
                Err.Description, vbCritical, _
                "modKap02.DLL_DeleteTable"
    End Select
    DLL_DeleteTableDao = False
    Resume HandleExit
End Function

'// ADO-Variante
Public Function DLL_DeleteTableAdo(pcnn As DAO.Database, psTable As
String) As Boolean

    On Error GoTo HandleErr
    Dim sSQL As String

    If TableExistsDAO(pdb, psTable) Then
        sSQL = "DROP TABLE " & psTable
        pcnn.Execute sSQL, dbFailOnError

```

```
        DLL_DeleteTableAdo = True
    End If

HandleExit:
    Exit Function

HandleErr:
    Select Case Err.Number
        Case Else
            MsgBox "Fehler " & Err.Number & ": " & _
                Err.Description, vbCritical, _
                "modKap02.DLL_DeleteTableAdo"
    End Select
    DLL_DeleteTableAdo = False
    Resume HandleExit
End Function
```

FELDBEARBEITUNG

Existiert ein Feld

Bevor man neues Feld anlegt bzw. ein vorhandenes ändert, sollte man prüfen, ob dieses bereits vorhanden ist. Mit der folgenden Funktion erhalten Sie ein **True** zurück, wenn es das Feld in der angegebenen Datenbank und Tabelle gibt.

DAO-Variante

```
Public Function DAO_FieldExists(pdb As DAO.Database, _
                               ByVal psTable As String, _
                               ByVal psField As String) As Boolean

    Dim S As String
    On Error Resume Next
    S = pdb.TableDefs(psTable).Fields(psField).Name
    DAO_FieldExists = (Err.Number = 0)

End Function
```

ADOX-Variante

```
Public Function ADO_FieldExists(pcnn As ADODB.Connection, _
                                ByVal psTable As String, _
                                ByVal psField As String) As Boolean

    Dim S As String
    Dim cat As New ADOX.Catalog

    On Error Resume Next
    cat.ActiveConnection = pcnn
    S = cat.Tables(psTable).Columns(psField).Name
    ADO_FieldExists = (Err.Number = 0)
    If Not cat Is Nothing Then Set cat = Nothing

End Function
```

Stimmt die Feldgröße eines Textfeldes

Wenn man die Größe eines Feldes vom Typ **Text** geändert hat sollte man vor einem Update bzw. einer Anpassung prüfen, ob die Größe bereits angepasst wurde. Die nachfolgende Funktion prüft, ob die Feldgröße der übergebenen Größe entspricht. Ist dies der Fall, wird **True** zurückgeliefert.

DAO-Variante

```
Public Function DAO_TextFieldSize(pdb As DAO.Database, _
    ByVal psTable As String, _
    ByVal psField As String, _
    ByVal piSize As Integer) As Boolean

    On Error GoTo HandleErr
    DAO_TextFieldSize = pdb.TableDefs(psTable).Fields(psField).Size =
piSize

HandleExit:
    Exit Function

HandleErr:
    Select Case Err.Number
        Case Else
            MsgBox "Fehler " & Err.Number & ": " & _
                Err.Description, vbCritical, _
                "modKap02.DAO_TextFieldSize"
    End Select
    DAO_TextFieldSize = False
    Resume HandleExit
End Function
```

ADOX-Variante

```
Public Function ADO_TextFieldSize(pcnn As ADODB.Connection, _
    ByVal psTable As String, _
    ByVal psField As String, _
    ByVal piSize As Integer) As Boolean

    On Error GoTo HandleErr
    Dim cat As New ADOX.Catalog
    cat.ActiveConnection = pcnn
    ADO_TextFieldSize = cat.Tables(psTable).Columns(psField).DefinedSize =
piSize

HandleExit:
    If Not cat Is Nothing Then Set cat = Nothing
    Exit Function

HandleErr:
    Select Case Err.Number
        Case Else
            MsgBox "Fehler " & Err.Number & ": " & _
                Err.Description, vbCritical, _
                "modKap02.ADO_TextFieldSize"
    End Select
    ADO_TextFieldSize = False
    Resume HandleExit
End Function
```

Welchen Felddatentyp besitzt das Tabellenfeld

Wenn man prüfen will, ob das Feld bereits den erforderlichen Felddatentyp besitzt sollte man folgende Funktion verwenden. Als Rückgabe erhält man eine Zahl, die man über eine Case-Schleife auflösen und den Datentyp zuordnen kann. Mehr Infos erhalten Sie über die Online-Hilfe.

DAO-Variante

```
Public Function DAO_CheckFieldType(pdb As DAO.Database, _
    ByVal psTable As String, _
    ByVal psField As String) As Long

    On Error GoTo HandleErr
    DAO_CheckFieldType = pdb.TableDefs(psTable).Fields(psField).Type

HandleExit:
    Exit Function

HandleErr:
    Select Case Err.Number
        Case Else
            MsgBox "Fehler " & Err.Number & ": " & _
                Err.Description, vbCritical, _
                "modKap02.DAO_CheckFieldType"
    End Select
    DAO_CheckFieldType = False
    Resume HandleExit
End Function
```

ADOX-Variante

```
Public Function ADO_CheckFieldType(pcnn As ADO.Connection, _
    ByVal psTable As String, _
    ByVal psField As String) As Long

    On Error GoTo HandleErr
    Dim cat As New ADOX.Catalog
    cat.ActiveConnection = pcnn
    ADO_CheckFieldType = cat.Tables(psTable).Columns(psField).Type

HandleExit:
    If Not cat Is Nothing Then Set cat = Nothing
    Exit Function

HandleErr:
    Select Case Err.Number
        Case Else
            MsgBox "Fehler " & Err.Number & ": " & _
                Err.Description, vbCritical, _
                "modKap02.ADO_CheckFieldType"
    End Select
    ADO_CheckFieldType = False
    Resume HandleExit
End Function
```

Feld anlegen

Für den Fall, dass eine Tabelle bereits existiert und nur ein zusätzliches Feld angehängt werden soll, gibt es verschiedene Möglichkeiten. Die Einfachste ist sicherlich alle Feldtypen in einer Funktion abzuhandeln.

Bei allen nachfolgenden Funktionen wird **True** als Rückgabe geliefert, wenn das Feld erfolgreich erstellt bzw. angehängt werden konnte. Zudem werden neben dem Datenbankobjekt

und dem Tabellennamen der anzulegende Feldname, Feldtype und ggf. die Feldgröße mit übergeben.

DAO-Variante

```
Public Function DAO_CreateField(pdb As DAO.Database, _
                               psTable As String, _
                               psFieldName As String, _
                               plFieldType As DAO.DataTypeEnum, _
                               Optional plFieldSize As Long = 0) _
    As Boolean

    ' Benötigte Objektvariablen
    On Error GoTo HandleErr
    Dim tdef As DAO.TableDef
    Dim tfld As DAO.Field

    '// Tabelle zuweisen
    Set tdef = pdb.TableDefs(psTable)
    '// Neues Feld anlegen
    If plFieldType = dbText Then
        ' Wenn Textfeld, dann
        Set tfld = tdef.CreateField(psFieldName, plFieldType,
plFieldSize)
    Else
        Set tfld = tdef.CreateField(psFieldName, plFieldType)
    End If

    '// Neues Feld in die Tabelle einfügen
    tdef.Fields.Append tfld
    '// Erfolgreich
    DAO_CreateField = True

HandleExit:
    '// Speicher freigeben
    If Not tdef Is Nothing Then Set tdef = Nothing
    If Not tfld Is Nothing Then Set tfld = Nothing
    Exit Function

HandleErr:
    Select Case Err.Number
        Case Else
            MsgBox "Fehler " & Err.Number & ": " & _
                Err.Description, vbCritical, _
                "modKap02.DAO_CreateField"
    End Select
    DAO_CreateField = False
    Resume HandleExit
End Function
```

ADOX-Variante

```
Public Function ADO_CreateField(pcnn As ADODB.Connection, _
                               psTable As String, _
                               psFieldName As String, _
                               plFieldType As ADODB.DataTypeEnum, _
                               Optional plFieldSize As Long = 0) _
    As Boolean

    ' Benötigte Objektvariablen
    On Error GoTo HandleErr
    Dim cat As New ADOX.Catalog
```

```

Dim tbl As ADOX.Table

cat.ActiveConnection = pcnn

'// Tabelle zuweisen
Set tbl = cat.Tables(psTable)

'// Neues Feld anlegen und einfügen
If plFieldType = adWChar Then
    ' Wenn Textfeld, dann
    tbl.Columns.Append psFieldName, plFieldType, plFieldSize
Else
    tbl.Columns.Append psFieldName, plFieldType
End If

'// Erfolgreich
ADO_CreateField = True

HandleExit:
'// Speicher freigeben
If Not tbl Is Nothing Then Set tbl = Nothing
If Not cat Is Nothing Then Set cat = Nothing
Exit Function

HandleErr:
Select Case Err.Number
    Case Else
        MsgBox "Fehler " & Err.Number & ": " & _
            Err.Description, vbCritical, _
            "modKap02.ADO_CreateField"
End Select
ADO_CreateField = False
Resume HandleExit
End Function

```

SQL-DDL-Variante

```

' DAO-Variante
Public Function DDL_CreateFieldDao(pdb As DAO.Database, _
    psTable As String, _
    psFieldName As String, _
    psFieldType As String, _
    Optional plFieldSize As Long = 0) _
    As Boolean

    On Error GoTo HandleErr
    Dim sSQL As String
    Const csText As String = "TEXT"

    sSQL = "ALTER TABLE " & psTable & " ADD COLUMN " & psFieldName & _
        " " & psFieldType
    If psFieldType = csText Then
        If plFieldSize > 0 Then
            sSQL = sSQL & "(" & plFieldSize & ")"
        End If
    End If

    pdb.Execute sSQL, dbFailOnError
    DDL_CreateFieldDao = True

HandleExit:
Exit Function

HandleErr:

```

```

Select Case Err.Number
    Case Else
        MsgBox "Fehler " & Err.Number & ": " & _
            Err.Description, vbCritical, _
            "modKap02.DDL_CreateFieldDao"
    End Select
DDL_CreateFieldDao = False
Resume HandleExit
End Function

' ADO-Variante
Public Function DDL_CreateFieldAdo(pcnn As ADODB.Connection, _
    psTable As String, _
    psFieldName As String, _
    psFieldType As String, _
    Optional plFieldSize As Long = 0) _
    As Boolean

    On Error GoTo HandleErr
    Dim sSQL As String
    Const csText As String = "TEXT"

    sSQL = "ALTER TABLE " & psTable & " ADD COLUMN " & psFieldName & _
        " " & psFieldType
    If psFieldType = csText Then
        If plFieldSize > 0 Then
            sSQL = sSQL & "(" & plFieldSize & ")"
        End If
    End If

    pcnn.Execute sSQL, dbFailOnError
    DDL_CreateFieldAdo = True

HandleExit:
    Exit Function

HandleErr:
    Select Case Err.Number
        Case Else
            MsgBox "Fehler " & Err.Number & ": " & _
                Err.Description, vbCritical, _
                "modKap02.DDL_CreateFieldAdo"
        End Select
    DDL_CreateFieldAdo = False
    Resume HandleExit
End Function

```

Feld löschen

Sollte es die Situation erfordern, muss auch schon mal ein Feld gelöscht werden. Bitte beachten Sie, dass nur Felder gelöscht werden können, die keinen Index besitzen und achten Sie darauf, dass die Datenintegrität gewährleistet bleibt. Falls nötig, löschen Sie zunächst den Index und dann erst das Feld.

DAO-Variante

```
Public Function DAO_DeleteField(pdb As DAO.Database, _
                               ByVal psTable As String, _
                               ByVal psField As String) As Boolean

    On Error Resume Next
    pdb.TableDefs(psTable).Fields.Delete psField
    DAO_DeleteField = (Err.Number = 0)

End Function
```

ADOX-Variante

```
Public Function ADO_DeleteField(pcnn As ADODB.Connection, _
                               ByVal psTable As String, _
                               ByVal psField As String) As Boolean

    Dim cat As New ADOX.Catalog

    On Error Resume Next
    cat.ActiveConnection = pcnn
    cat.Tables(psTable).Columns.Delete psField
    ADO_DeleteField = (Err.Number = 0)
    If Not cat Is Nothing Then Set cat = Nothing

End Function
```

SQL-DDL-Variante

```
' DAO-Variante
Public Function DDL_DeleteFieldDao(pdb As DAO.Database, _
                                   psTable As String, _
                                   psFieldName As String) As Boolean

    On Error Resume Next
    Dim sSQL As String

    sSQL = "ALTER TABLE " & psTable & " DROP COLUMN " & psFieldName

    pdb.Execute sSQL, dbFailOnError
    DDL_DeleteFieldDao = (Err.Number = 0)

End Function

' ADO-Variante
Public Function DDL_DeleteFieldADO(pcnn As ADODB.Connection, _
                                   psTable As String, _
                                   psFieldName As String) As Boolean

    On Error Resume Next
    Dim sSQL As String

    sSQL = "ALTER TABLE " & psTable & " DROP COLUMN " & psFieldName

    pcnn.Execute sSQL, dbFailOnError
    DDL_DeleteFieldADO = (Err.Number = 0)

End Function
```

Feldname ändern

Die Funktion, einen Feldnamen zu ändern, dürfte man eigentlich fast nie benötigen, da in diesem Fall der Name in der gesamten Anwendung geändert werden muss. Der Vollständigkeit halber möchten wir jedoch auch aufzeigen, wie ein Feldname geändert werden kann.

DAO-Variante

```
Public Function DAO_RenameField(pdb As DAO.Database, _
                               ByVal psTable As String, _
                               ByVal psField As String, _
                               ByVal psNewField As String) As Boolean

    On Error Resume Next
    pdb.TableDefs(psTable).Fields(psField).Name = psNewField
    DAO_RenameField = (Err.Number = 0)

End Function
```

ADOX-Variante

```
Public Function ADO_RenameField(pcnn As ADODB.Connection, _
                               ByVal psTable As String, _
                               ByVal psField As String, _
                               ByVal psNewField As String) As Boolean

    Dim cat As New ADOX.Catalog

    On Error Resume Next
    cat.ActiveConnection = pcnn
    cat.Tables(psTable).Columns(psField).Name = psNewField
    ADO_RenameField = (Err.Number = 0)
    If Not cat Is Nothing Then Set cat = Nothing

End Function
```

Feldgröße/-typ ändern

Es besteht unter DAO/ADOX keine Möglichkeit, den Datentyp und die Größe eines Feldes nachträglich zu verändern. Der einzige Ausweg ist, eine weitere Spalte mit den gewünschten Eigenschaften hinzuzufügen, eine Aktualisierungsabfrage durchzuführen, die die Datenwerte von der alten in die neue Spalte überträgt und dann die alte Spalte zu löschen. Hier ist es zu empfehlen, auf die SQL-DDL Anweisungen zurückzugreifen, da dies die Durchführung wesentlich beschleunigt. Bitte beachten Sie auch hier, dass, falls das zu verändernde Feld einen Index enthält, diesen Index vorab zu löschen.

Ab Jet 4.0 (Access 2000 oder höher), benötigen Sie diesen umständlichen Weg nicht mehr. Hier können Sie es mit einer einfachen SQL-DDL-Anweisung realisieren.

DAO-Variante

```

Public Function DAO_ChangeFieldType(pdb As DAO.Database, _
    psTable As String, _
    psField As String, _
    psFieldType As String, _
    Optional plFieldSize As Long = 0) _
    As Boolean

    On Error GoTo HandleErr

    Dim sSQL As String
    Const csText As String = "TEXT"

    '// Erstellung eines Dummyfeldes mit dem
    '// neuen Feldtypen
    sSQL = "ALTER TABLE [" & psTable & _
        "] ADD COLUMN AlterTempField " & psFieldType

    If psFieldType = csText Then
        If plFieldSize > 0 Then
            sSQL = sSQL & "(" & plFieldSize & ")"
        End If
    End If

    pdb.Execute sSQL, dbFailOnError

    '// Die bestehenden Daten in das Dummyfeld kopieren
    sSQL = "UPDATE DISTINCTROW [" & psTable & _
        "] SET AlterTempField = [" & psField & "]"
    pdb.Execute sSQL, dbFailOnError

    '// das alte Feld löschen und die Auflistung aktualisieren
    sSQL = "ALTER TABLE [" & psTable & _
        "] DROP COLUMN [" & psField & "]"
    pdb.Execute sSQL, dbFailOnError

    pdb.TableDefs "[" & psTable & "']").Fields.Refresh

    ' Das Dummyfeld mit der alten Bezeichnung umbenennen
    pdb.TableDefs "[" & psTable & _
        "']").Fields("AlterTempField").Name = psField

    DAO_ChangeFieldType = True

HandleExit:
    Exit Function

HandleErr:
    Select Case Err.Number
        Case Else
            MsgBox "Fehler " & Err.Number & ": " & _
                Err.Description, vbCritical, _
                "modKap02.DAO_ChangeFieldType"
    End Select
    DAO_ChangeFieldType = True
    Resume HandleExit
End Function

```

ADOX-Variante

```

Public Function ADO_ChangeFieldType(pcnn As ADODB.Connection, _
    psTable As String, _
    psField As String, _
    psFieldType As String, _
    Optional plFieldSize As Long = 0) _
    As Boolean

    On Error GoTo HandleErr

    Dim sSQL As String
    Dim cat As New ADOX.Catalog
    Const csText As String = "TEXT"

    cat.ActiveConnection = pcnn

    '// Erstellung eines Dummyfeldes mit dem
    '// neuen Feldtypen
    sSQL = "ALTER TABLE [" & psTable & _
        "] ADD COLUMN AlterTempField " & psFieldType

    If psFieldType = csText Then
        If plFieldSize > 0 Then
            sSQL = sSQL & "(" & plFieldSize & ")"
        End If
    End If

    pcnn.Execute sSQL, dbFailOnError

    '// Die bestehenden Daten in das Dummyfeld kopieren
    sSQL = "UPDATE DISTINCTROW [" & psTable & _
        "] SET AlterTempField = [" & psField & "]"
    pcnn.Execute sSQL, dbFailOnError

    '// das alte Feld löschen und die Auflistung aktualisieren
    sSQL = "ALTER TABLE [" & psTable & _
        "] DROP COLUMN [" & psField & "]"
    pcnn.Execute sSQL, dbFailOnError

    ' Das Dummyfeld mit der alten Bezeichnung umbenennen
    cat.Tables(psTable).Columns("AlterTempField").Name = psField

    ADO_ChangeFieldType = True

HandleExit:
    If Not cat Is Nothing Then Set cat = Nothing
    Exit Function

HandleErr:
    Select Case Err.Number
        Case Else
            MsgBox "Fehler " & Err.Number & ": " & _
                Err.Description, vbCritical, _
                "modKap02.ADO_ChangeFieldType"
    End Select
    ADO_ChangeFieldType = False
    Resume HandleExit
End Function

```

SQL-DDL-Variante (ab Jet 4.0)

```

' DAO-Variante
Public Function DDL_ChangeFieldDao(pdb As DAO.Database, _
                                psTable As String, _
                                psFieldName As String, _
                                psFieldType As String, _
                                Optional plFieldSize As Long = 0) _
    As Boolean

    On Error GoTo HandleErr
    Dim sSQL As String
    Const csText As String = "TEXT"

    sSQL = "ALTER TABLE " & psTable & " ALTER COLUMN " & psFieldName & _
        " " & psFieldType
    If psFieldType = csText Then
        If plFieldSize > 0 Then
            sSQL = sSQL & "(" & plFieldSize & ")"
        End If
    End If

    pdb.Execute sSQL, dbFailOnError
    DDL_ChangeFieldDao = True

HandleExit:
    Exit Function

HandleErr:
    Select Case Err.Number
        Case Else
            MsgBox "Fehler " & Err.Number & ": " & _
                Err.Description, vbCritical, _
                "modKap02.DDL_ChangeFieldDao"
    End Select
    DDL_ChangeFieldDao = False
    Resume HandleExit
End Function

' ADO-Variante
Public Function DDL_ChangeFieldAdo(pcnn As ADODB.Connection, _
                                psTable As String, _
                                psFieldName As String, _
                                psFieldType As String, _
                                Optional plFieldSize As Long = 0) _
    As Boolean

    On Error GoTo HandleErr
    Dim sSQL As String
    Const csText As String = "TEXT"

    sSQL = "ALTER TABLE " & psTable & " ALTER COLUMN " & psFieldName & _
        " " & psFieldType
    If psFieldType = csText Then
        If plFieldSize > 0 Then
            sSQL = sSQL & "(" & plFieldSize & ")"
        End If
    End If

    pcnn.Execute sSQL, dbFailOnError
    DDL_ChangeFieldAdo = True

HandleExit:
    Exit Function

```

```
HandleErr:
  Select Case Err.Number
    Case Else
      MsgBox "Fehler " & Err.Number & ": " & _
        Err.Description, vbCritical, _
        "modKap02.DDL_ChangeFieldAdo"
  End Select
  DDL_ChangeFieldAdo = False
  Resume HandleExit
End Function
```

INDIZES

ANLEGEN/MODIFIZIEREN

Die bisherigen Themen drehten sich bislang nur um Techniken, wie man Tabellen mit einem oder mehreren Felder anlegt und umgestaltet. Der Zugriff auf Tabellen ist allerdings nur effizient, wenn diese Indizes bereitstellen, die das schnelle Auffinden und Sortieren der einzelnen Datensätze erst ermöglichen. Ist kein Index vorhanden, so muss die Jet-Engine die Datensätze einer Tabelle der Reihe nach untersuchen, um einen bestimmten herauszufinden.

Index prüfen

Bevor Sie Indizes erstellen bzw. bearbeiten, sollten Sie vorab überprüfen, ob diese überhaupt vorhanden sind.

DAO-Variante

```
Public Function DAO_IndexExists(pdb As DAO.Database, _
                               ByVal psTable As String, _
                               ByVal psIDXName As String) As Boolean

    Dim S As String
    On Error Resume Next
    S = pdb.TableDefs(psTable).Indexes(psIDXName).Name
    DAO_IndexExists = (Err.Number = 0)

End Function
```

ADOX-Variante

```
Public Function ADO_IndexExists(pcnn As ADODB.Connection, _
                               ByVal psTable As String, _
                               ByVal psIDXName As String) As Boolean

    Dim S As String
    Dim cat As New ADOX.Catalog

    On Error Resume Next
    cat.ActiveConnection = pcnn
    S = cat.Tables(psTable).Indexes(psIDXName).Name
    If Not cat Is Nothing Then Set cat = Nothing
    ADO_IndexExists = (Err.Number = 0)

End Function
```

Index setzen (mit Duplikate / ohne Duplikate / Primärschlüssel)

Sie können für eine Tabelle einen oder mehrere Indizes erstellen – entweder sofort nach dem Anlegen der Tabelle oder zu einem späteren Zeitpunkt.

Es ist üblich, beim Anlegen einer Tabelle, ein Feld oder eine Zusammensetzung aus mehreren Feldern mit einem eindeutigen Index, dem so genannten Primärschlüssel, zu versehen, der fortan die Standardordnung über die Tabelle verkörpert.

Da ein Primärschlüssel laut Definition ein eindeutiger Index sein muss, kann man sich die Frage stellen, was der Unterschied zwischen den Eigenschaften **Primary** und **Unique** ist.

Der wesentliche Unterschied besteht darin, dass ein Primärschlüssel unter keinen Umständen Nullwerte erlaubt, während ein Index mit auf True gesetzter Eigenschaft Unique in Kombination einer auf True gesetzten Eigenschaft **IgnoreNulls** Nullwerte akzeptiert, indem er sie ignoriert.

Die nachstehenden Beispiele zeigen Ihnen eine Möglichkeit, sowohl einen Primärschlüssel, als auch nur Indizes mit Duplikate bzw. ohne Duplikate zu hinterlegen.

DAO-Variante

```
Public Function DAO_CreateIndex(pdb As DAO.Database, _
    psTable As String, _
    psField As String, _
    pbPrimary As Boolean, _
    pbUnique As Boolean, _
    Optional psIDXName _
    As String = vbNullString) _
    As Boolean

    On Error GoTo HandleErr

    '// Benötigte Objektvariablen
    Dim tdef As DAO.TableDef
    Dim tfld As DAO.Field
    Dim tid As DAO.Index

    '// Zuweisung der Tabelle
    Set tdef = pdb.TableDefs(psTable)
    '// Index erstellen
    Set tid = tdef.CreateIndex

    With tid
        '// falls angegeben Indexname
        If Len(psIDXName) > 0 Then
            .Name = psIDXName
        Else
            .Name = psField
        End If
        '// falls angegeben, Primary-Eigenschaft setzen
        If pbPrimary Then tid.Primary = True
        '// Unique-Eigenschaft (Eindeutigkeit) setzen
        .Unique = pbUnique
        '// Index für das Feld festlegen
        Set tfld = .CreateField(psField)
    End With
End Function
```

```

    '// Index für das Feld übernehmen
    .Fields.Append tfld
End With

    '// Index in der Auflistung übernehmen
tdef.Indexes.Append tidx
    '// Auflistung aktualisieren
tdef.Indexes.Refresh

    '// Erfolgreich
DAO_CreateIndex = True

HandleExit:
    '// Speicher freigeben
If Not tfld Is Nothing Then Set tfld = Nothing
If Not tidx Is Nothing Then Set tidx = Nothing
If Not tdef Is Nothing Then Set tdef = Nothing
Exit Function

HandleErr:
    Select Case Err.Number
        Case Else
            MsgBox "Fehler " & Err.Number & ": " & _
                Err.Description, vbCritical, _
                "modKap03.DAO_CreateIndex"
    End Select
    DAO_CreateIndex = False
    Resume HandleExit
End Function

```

ADOX-Variante

```

Public Function ADO_CreateIndex(pcnn As ADODB.Connection, _
    psTable As String, _
    psField As String, _
    pfBPrimary As Boolean, _
    pfBUnique As Boolean, _
    Optional psIDXName As String =
vbNullString) _
    As Boolean

    On Error GoTo HandleErr

    '// Benötigte Objektvariablen
Dim cat As New ADOX.Catalog
Dim tbl As ADOX.Table
Dim col As ADOX.Column
Dim idx As ADOX.Index

    cat.ActiveConnection = pcnn

    '// Zuweisung der Tabelle
Set tbl = cat.Tables(psTable)

    '// Index erstellen
Set idx = New Index
With idx
    '// falls angegeben Indexname
    If Len(psIDXName) > 0 Then
        .Name = psIDXName
    Else
        .Name = psField
    End If
    '// falls angegeben, Primary-Eigenschaft setzen

```

```

If pfBPrimary Then
    .IndexNulls = adIndexNullsDisallow
Else
    .IndexNulls = adIndexNullsAllow
End If
If pfBPrimary Then
    .PrimaryKey = pfBPrimary
End If
'// Unique-Eigenschaft (Eindeutigkeit) setzen
.Unique = pfBUnique

'// Index für das Feld festlegen
Set col = New Column
col.Name = psField
'// Index für das Feld übernehmen
.Columns.Append col
'// Index in der Auflistung übernehmen
tbl.Indexes.Append idx
End With
ADO_CreateIndex = True

HandleExit:
'// Speicher freigeben
If Not idx Is Nothing Then Set idx = Nothing
If Not col Is Nothing Then Set col = Nothing
If Not tbl Is Nothing Then Set tbl = Nothing
If Not cat Is Nothing Then Set cat = Nothing
Exit Function

HandleErr:
Select Case Err.Number
Case Else
    MsgBox "Fehler " & Err.Number & ": " & _
        Err.Description, vbCritical, _
        "modKap03.ADO_CreateIndex"
End Select
ADO_CreateIndex = False
Resume HandleExit
End Function

```

SQL-DLL-Variante

```

' DAO-Variante
Public Function DDL_CreateIndexDAO(pdb As DAO.Database, _
    psTable As String, _
    psField As String, _
    pfBPrimary As Boolean, _
    pfBUnique As Boolean, _
    Optional psIDXName _
    As String = vbNullString) _
    As Boolean

On Error GoTo HandleErr
Dim sSQL As String

sSQL = "Create "
'// falls angegeben, Unique-Eigenschaft (Eindeutigkeit) setzen
If pfBUnique And Not pfBPrimary Then
    sSQL = sSQL & "UNIQUE "
End If
sSQL = sSQL & "INDEX "

'// falls angegeben Indexname
If Len(psIDXName) > 0 Then
    sSQL = sSQL & psIDXName

```

```

Else
    sSQL = sSQL & psField
End If

sSQL = sSQL & " ON " & psTable & " (" & psField & ")"

'// falls angegeben, Primary-Eigenschaft setzen
If pfBPrimary Then
    sSQL = sSQL & " WITH PRIMARY"
End If

'// Index erstellen
pDBs.Execute sSQL, dbFailOnError
DDL_CreateIndexDAO = True

HandleExit:
Exit Function

HandleErr:
Select Case Err.Number
    Case Else
        MsgBox "Fehler " & Err.Number & ": " & _
            Err.Description, vbCritical, _
            "modKap03.DDL_CreateIndexDAO"
End Select
DDL_CreateIndexDAO = False
Resume HandleExit
End Function

' ADO-Variante
Public Function DDL_CreateIndexADO(pcnn As ADODB.Connection, _
    psTable As String, _
    psField As String, _
    pfBPrimary As Boolean, _
    pfBUnique As Boolean, _
    Optional psIDXName _
    As String = vbNullString) _
    As Boolean

    On Error GoTo HandleErr
    Dim sSQL As String

    sSQL = "Create "
    '// falls angegeben, Unique-Eigenschaft (Eindeutigkeit) setzen
    If pfBUnique And Not pfBPrimary Then
        sSQL = sSQL & "UNIQUE "
    End If
    sSQL = sSQL & "INDEX "

    '// falls angegeben Indexname
    If Len(psIDXName) > 0 Then
        sSQL = sSQL & psIDXName
    Else
        sSQL = sSQL & psField
    End If

    sSQL = sSQL & " ON " & psTable & " (" & psField & ")"

    '// falls angegeben, Primary-Eigenschaft setzen
    If pfBPrimary Then
        sSQL = sSQL & " WITH PRIMARY"
    End If

    '// Index erstellen
    pcnn.Execute sSQL, dbFailOnError
    DDL_CreateIndexADO = True

```

```

HandleExit:
    Exit Function

HandleErr:
    Select Case Err.Number
        Case Else
            MsgBox "Fehler " & Err.Number & ": " & _
                Err.Description, vbCritical, _
                "modKap03.DDL_CreateIndexADO"
    End Select
    DDL_CreateIndexADO = False
    Resume HandleExit
End Function

```

Index löschen

Wenn Sie einen Index nicht mehr benötigen, können Sie diesen natürlich auch löschen. Beachten Sie aber, dass Beziehungsdefinitionen zwischen Tabellen existieren können, die das Löschen eines Index wirksam verhindern. Die Jet Engine kann zwischen zwei Tabellen nur die Referenzintegrität gewährleisten, wenn für die Verknüpfungsfelder Indizes existieren.

DAO-Variante

```

Public Function DAO_DropIndex(pdb As DAO.Database, _
    psTable As String, _
    psField As String, _
    Optional psIDXName _
    As String = vbNullString) _
    As Boolean

    On Error Resume Next
    '// Falls ein Indexname angegeben wurde
    If Len(psIDXName) > 0 Then
        pdb.TableDefs(psTable).Indexes.Delete psIDXName
    Else
        '// Wenn nein, ist es meistens so, das die Feldbe-
        '// zeichnung als Name für den Index verwendet wurde
        pdb.TableDefs(psTable).Indexes.Delete psField
    End If
    DAO_DropIndex = (Err.Number = 0)
End Function

```

ADOX-Variante

```

Public Function ADOX_DropIndex(pcnn As ADOX.Connection, _
    psTable As String, _
    psField As String, _
    Optional psIDXName _
    As String = vbNullString) _
    As Boolean

    On Error Resume Next
    Dim cat As New ADOX.Catalog
    cat.ActiveConnection = pcnn
    '// Falls ein Indexname angegeben wurde
    If Len(psIDXName) > 0 Then
        cat.Tables(psTable).Indexes.Delete psIDXName
    Else

```

```

    '// Wenn nein, ist es meistens so, das die Feldbe-
    '// zeichnung als Name für den Index verwendet wurde
    cat.Tables(psTable).Indexes.Delete psField
End If
If Not cat Is Nothing Then Set cat = Nothing
ADO_DropIndex = (Err.Number = 0)

```

```
End Function
```

SQL-DLL-Variante

```

' DAO-Variante
Public Function DLL_DropIndexDAO(pdb As DAO.Database, _
    psTable As String, _
    psField As String, _
    Optional psIdxName _
    As String = vbNullString) _
    As Boolean

    On Error Resume Next
    Dim sSQL As String

    sSQL = "DROP INDEX "
    If Len(psIdxName) > 0 Then
        sSQL = sSQL & psIdxName & " ON " & psTable
    Else
        sSQL = sSQL & psField & " ON " & psTable
    End If
    pdb.Execute sSQL, dbFailOnError
    DLL_DropIndexDAO = (Err.Number = 0)

End Function

' ADO-Variante
Public Function DLL_DropIndexADO(pcnn As ADODB.Connection, _
    psTable As String, _
    psField As String, _
    Optional psIdxName _
    As String = vbNullString) _
    As Boolean

    On Error Resume Next
    Dim sSQL As String

    sSQL = "DROP INDEX "
    If Len(psIdxName) > 0 Then
        sSQL = sSQL & psIdxName & " ON " & psTable
    Else
        sSQL = sSQL & psField & " ON " & psTable
    End If
    pcnn.Execute sSQL, dbFailOnError
    DLL_DropIndexADO = (Err.Number = 0)

End Function

```

REFERENZINTEGRITÄTEN UND BEZIEHUNGEN

Bisher haben wir die bestehenden Tabellenobjekte nur isoliert betrachtet. Es ging bisher darum, wie man in einer bestehenden Datenbank Tabellen-Objekte und in diesen wiederum, einzelne Feld-Objekte hinzufügt und bearbeitet. Zudem wurde aufgezeichnet, wie man die Suche über die Tabellen durch die Bereitstellung von Indizes für Tabellen beschleunigt.

Die Jet unterstützt aber auch weitergehende Konstrukte, wie die Spezifikation von Beziehungen mit vielfältigen Regeln, die als Relationen das **Miteinander** von Tabellen definiert. Sollten Sie im Nachhinein weitere Tabellen hinzugefügt haben, müssen Sie unter Umständen die Integritäten und Beziehungen bearbeiten.

Beziehungen erstellen

Grundsätzlich besteht programmiertechnisch gesehen keine großen Unterschiede zwischen dem Anlegen von Relationen und dem Anlegen von Indizes oder Tabellen. Sie vereinbaren eine Objektvariable für ein Objekt von Type Relation (DAO) bzw. Key (ADOX) oder Verwenden die CONSTRAINT (DDL) Anweisung, charakterisieren es, fügen ein oder mehrere Feld Objekte in dessen Auflistung ein und fügen das Ganze dann in die Relationen Auflistung des Datenbankobjekts ein. Die Charakterisierung legt die Art der Beziehung fest, zum Beispiel ob sie kaskadierte Aktualisierungen und Löschooperationen impliziert.

DAO-Variante

```
Public Function DAO_CreateRelation(pdb As DAO.Database, _
    psPTable As String, _
    psFTable As String, _
    psPField As String, _
    psFField As String, _
    psRelName As String, _
    Optional plRelType _
    As DAO.RelationAttributeEnum = 0) _
    As Boolean

    '/// =====
    '/// Methode | Erstellt eine Beziehung under DAO
    '/// =====
    '/// Parameter | pdbs      - Datenbankobjekt
    '                psPTable - Primärtabelle (1-Seite)
    '                psFTable - Sekundärtabelle (n-Seite)
    '                psPField - Primärfeld
    '                psFField - Sekundärfeld
    '                psRelName - Eindeutige Bezeichnung
```

```

'           plRelType - Relationen-Eigenschaften z.B.
'// -----
'   Löschweitergabe           - dbRelationDeleteCascade
'   Aktualisierungswweitergabe - dbRelationUpdateCascade
'   1-1-Beziehung             - dbRelationUnique
'   man kann es auch kombiniert z.B.
'   dbRelationUnique + dbRelationDeleteCascade ect.
'// -----
'// Rückgabe | Boolean - Erfolgreich True, sonst False
'// -----
'// Erstellt | Manuela Kulpa - Nov 09, 2003
'//          | EDV Innovation & Consulting - Dormagen
'// =====

On Error GoTo HandleErr

Dim rel As DAO.Relation
Dim fld As DAO.Field

' Erstellen der Beziehung
Set rel = pdbc.CreateRelation()
With rel
    ' Eindeutige Bezeichnung der Relation
    rel.Name = psRelName
    ' Angabe Primärtabelle (1-Seite)
    rel.Table = psPTable
    ' Angabe Sekundärtabelle (n-Seite)
    rel.ForeignTable = psFTable
End With

' Angabe des Feldes der Primärseite (1-Seite)
Set fld = rel.CreateField(psPField)

' Einstellen der ForeignName-Eigenschaft des
' Feldes auf den Namen des entsprechenden Feldes
' in der Primärtabelle, hier die Sekundärseite
fld.ForeignName = psFField
If plRelType > 0 Then
    rel.Attributes = rel.Attributes Or plRelType
End If

rel.Fields.Append fld

' Anhängen der Beziehung zur Auflistung
pdbc.Relations.Append rel
DAO_CreateRelation = True

HandleExit:
If Not fld Is Nothing Then Set fld = Nothing
If Not rel Is Nothing Then Set rel = Nothing
Exit Function

HandleErr:
Select Case Err.Number
    Case Else
        MsgBox "Fehler " & Err.Number & ": " & _
            Err.Description, vbCritical, _
            "modKap04.DAO_CreateRelation"
End Select
DAO_CreateRelation = False
Resume HandleExit
End Function

```

ADOX-Variante

```

Public Function ADO_CreateRelation(pcnn As ADODB.Connection, _
                                psPTable As String, _
                                psFTable As String, _
                                psPField As String, _
                                psFField As String, _
                                psRelName As String, _
                                Optional pfBUpdateRule As Boolean, _
                                Optional pfBUpDeleteRule As Boolean) _
    As Boolean

    '// =====
    '// Methode | Erstellt eine Beziehung und ADOX
    '// -----
    '// Parameter | pcnn          - Connection
    '//              psPTable     - Primärtabelle
    '//              psFTable     - Sekundärtabelle
    '//              psPField     - Primärfeld
    '//              psFField     - Sekundärfeld
    '//              psRelName    - Eindeutige Bezeichnung
    '//              pfBUpdateRule - Löschweitergabe
    '//              pfBUpDeleteRule - Aktualisierungsweiterg.
    '// -----
    '// Rückgabe | Boolean -
    '// -----
    '// Erstellt | Manuela Kulpa - Nov 09, 2003
    '//           | EDV Innovation & Consulting - Dormagen
    '// =====

    On Error GoTo HandleErr
    Dim cat As New ADOX.Catalog
    Dim tbl As ADOX.Table
    Dim rel As New ADOX.key

    ' Öffnen des Katalogs
    cat.ActiveConnection = pcnn

    ' Angabe Sekundärtabelle (n-Seite)
    Set tbl = cat.Tables(psFTable)

    With rel
        ' Eindeutige Bezeichnung der Relation
        .Name = psRelName
        ' Angabe des Beziehungstypes
        ' auch wenn eine 1-1-Beziehung vorliegt,
        ' ist unter ADOX keine Möglichkeit gegeben,
        ' dies explizit anzugeben, Jet erkennt es
        ' aber, es wird im Beziehungsfenster
        ' aber nicht korrekt dargestellt
        .Type = adKeyForeign
        ' Angabe Primärtabelle (1-Seite)
        .RelatedTable = psPTable

        ' Angabe des Sekundärfeldes (n-Seite)
        .Columns.Append psFField

        ' Einstellen der RelatedColumn-Eigenschaft
        ' auf den Namen der entsprechenden Spalte
        ' in der Primärtabelle
        .Columns(psFField).RelatedColumn = psPField
    End With
    If pfBUpdateRule Then
        .UpdateRule = adRICascade
    End If
    If pfBUpDeleteRule Then

```

```

        .DeleteRule = adRiCascade
    End If
End With

' Anhängen der Beziehung zur Auflistung
tbl.Keys.Append rel
ADO_CreateRelation = True

HandleExit:
If Not rel Is Nothing Then Set rel = Nothing
If Not tbl Is Nothing Then Set tbl = Nothing
If Not cat Is Nothing Then Set cat = Nothing
Exit Function

HandleErr:
Select Case Err.Number
    Case Else
        MsgBox "Fehler " & Err.Number & ": " & _
            Err.Description, vbCritical, _
            "modKap04.ADO_CreateRelation"
End Select
ADO_CreateRelation = False
Resume HandleExit
End Function

```

SQL-DDL-Variante

```

' DAO-Variante
Public Function DDL_CreateRelationDao(pdb As DAO.Database, _
    psPTable As String, _
    psFTable As String, _
    psPField As String, _
    psFField As String, _
    psRelName As String) _
    As Boolean

    On Error GoTo HandleErr

    Dim sSQL As String

    sSQL = "ALTER TABLE " & psFTable
    sSQL = sSQL & " ADD CONSTRAINT " & psRelName
    sSQL = sSQL & " FOREIGN KEY ([" & psFField
    sSQL = sSQL & "]) REFERENCES " & psPTable
    sSQL = sSQL & " ([" & psPField & "])"

    pdb.Execute sSQL, dbFailOnError
    DDL_CreateRelationDao = True

HandleExit:
Exit Function

HandleErr:
Select Case Err.Number
    Case Else
        MsgBox "Fehler " & Err.Number & ": " & _
            Err.Description, vbCritical, _
            "modKap04.DDL_CreateRelationDao"
End Select
DDL_CreateRelationDao = False
Resume HandleExit
End Function

```

```

' ADO-Variante
Public Function DDL_CreateRelationADO(pcnn As ADODB.Connection, _
    psPTable As String, _
    psFTable As String, _
    psPField As String, _
    psFField As String, _
    psRelName As String, _
    Optional pfBUpdateRule As Boolean,
    _
    Optional pfBUpDeleteRule As
Boolean) _
    As Boolean

    On Error GoTo HandleErr

    Dim sSQL As String

    sSQL = "ALTER TABLE " & psFTable
    sSQL = sSQL & " ADD CONSTRAINT " & psRelName
    sSQL = sSQL & " FOREIGN KEY ([" & psFField
    sSQL = sSQL & "]) REFERENCES " & psPTable
    sSQL = sSQL & " ([" & psPField & "])"
    ' Nur unter ADO möglich
    If pfBUpdateRule Then
        sSQL = sSQL & " ON UPDATE CASCADE"
    End If
    If pfBUpDeleteRule Then
        sSQL = sSQL & " ON DELETE CASCADE"
    End If

    pcnn.Execute sSQL, dbFailOnError
    DDL_CreateRelationADO = True

HandleExit:
    Exit Function

HandleErr:
    Select Case Err.Number
        Case Else
            MsgBox "Fehler " & Err.Number & ": " & _
                Err.Description, vbCritical, _
                "modKap04.DDL_CreateRelationADO"
    End Select
    DDL_CreateRelationADO = False
    Resume HandleExit
End Function

```

Beziehungen prüfen

Bevor Sie Beziehungen erstellen bzw. bearbeiten, sollten Sie vorab überprüfen, ob diese überhaupt vorhanden sind.

DAO-Variante

```
Public Function DAO_RelationExists(pdb As DAO.Database, _
    ByVal psRelName As String) As Boolean
    Dim S As String
    On Error Resume Next
    S = pdb.Relations(psRelName).Name
    DAO_RelationExists = (Err.Number = 0)
End Function
```

ADOX-Variante

```
Public Function ADO_RelationExists(pcnn As ADODB.Connection, _
    ByVal psRelName As String, _
    ByVal psFTable As String) As Boolean

    Dim S As String
    Dim cat As New ADOX.Catalog

    On Error Resume Next
    cat.ActiveConnection = pcnn
    ' Wichtig, hier muss die Sekundärtabelle
    ' angegeben werden
    S = cat.Tables(psFTable).Keys(psRelName).Name

    If Not cat Is Nothing Then Set cat = Nothing
    ADO_RelationExists = (Err.Number = 0)
End Function
```

Beziehungen löschen

Wenn Sie eine Beziehung nicht mehr benötigen, können Sie diesen natürlich auch löschen. Beachten Sie aber, dass Referenzintegrität gegebenenfalls nicht mehr gewährleisten ist.

DAO-Variante

```
Public Function DAO_DropRelation(pdb As DAO.Database, _
    psRelName As String) _
    As Boolean

    On Error Resume Next
    pdb.Relations.Delete psRelName

    DAO_DropRelation = (Err.Number = 0)
End Function
```

ADOX-Variante

```
Public Function ADO_DropRelation(pcnn As ADODB.Connection, _
                               ByVal psRelName As String, _
                               ByVal psFTable As String) As Boolean

    Dim S As String
    Dim cat As New ADOX.Catalog

    On Error Resume Next
    cat.ActiveConnection = pcnn
    ' Wichtig, hier muss die Sekundärtabelle
    ' angegeben werden
    cat.Tables(psFTable).Keys.Delete psRelName

    If Not cat Is Nothing Then Set cat = Nothing
    ADO_DropRelation = (Err.Number = 0)

End Function
```

SQL-DDL-Variante

```
' DAO-Variante
Public Function DDL_DropRelationDao(pdb As DAO.Database, _
                                   psFTable As String, _
                                   psRelName As String) _
                                   As Boolean

    Dim sSQL As String

    On Error Resume Next
    sSQL = "ALTER TABLE " & psFTable
    sSQL = sSQL & " DROP CONSTRAINT " & psRelName

    pdb.Execute sSQL, dbFailOnError
    DDL_DropRelationDao = (Err.Number = 0)

End Function
' ADO-Variante
Public Function DDL_DropRelationAdo(pcnn As ADODB.Connection, _
                                   psFTable As String, _
                                   psRelName As String) _
                                   As Boolean

    Dim sSQL As String

    On Error Resume Next
    sSQL = "ALTER TABLE " & psFTable
    sSQL = sSQL & " DROP CONSTRAINT " & psRelName

    pcnn.Execute sSQL, dbFailOnError
    DDL_DropRelationAdo = (Err.Number = 0)

End Function
```