Dynamische Abfragen im MS-Access

Karel Štípek

Die einfache Erstellung von Formularen, Abfragen, Berichten im MS-Access ermöglicht eine schnelle Programmentwicklung. Mit den laufenden Erweiterungen einer Access-Lösung steigt die Anzahl der notwendigen Datenbankobjekte und das Programm kann unübersichtlich und dadurch nur schwer anpassungsfähig werden.

Im vorgelegten Artikel werden zuerst einige allgemeine Empfehlungen zur Erstellung komplexer Access-Programme zusammengefasst. Ausführlichere Informationen dieser Art können Sie dem Artikel "Professionell programmieren mit Microsoft Access" in PCNEWS-71 entnehmen. Als Hauptthema wird die Arbeit mit dynamisch definierten Abfragen erklärt und an konkreten praktischen Beispielen präsentiert.

Allgemeine Empfehlungen

Namenskonventionen

Die konsequente Einführung der Namenspräfixe je nach dem Typ der Datenbankobjekte erhöht die Übersichtlichkeit des Programms. Gute Vorschläge kann man praktisch jedem Access-Buch entnehmen. Außer den Standardkonventionen ist es noch empfehlenswert, die Tabellen und Abfragen aus der Sicht der Bedeutung für die Entwicklung und der Veränderbarkeit beim Programmablauf weiter zu unterscheiden.

Folgende Tabelle enthält ein paar Tipps

Präfix	Verwendung
`tbl`	Standard-Tabelle mit Daten, die eingegeben oder importiert werden.
`stbl`	Systemtabelle mit Projektdefinitionen, die nur vom Entwickler einmalig befüllt wird.
`qry`	Standard-Abfrage mit einer konstanten Definition.
`qrd`	Dynamische Abfrage, derer Definition während des Programmablaufs verändert wird. Der gleiche Präfix wird auch für eine berechnete Tabelle (aus Gründen der Performance temporär gespeicherte Abfrage) verwendet.
qrs	Hilfsabfrage für die Entwicklung, die beim normalen Programmablauf nicht verwendet wird.
a_	Alle temporären Objekte (ältere Versionen, Testobjekte), die für den produktiven Einsatz nicht geplant sind und vor der Auslieferung des Programms gelöscht werden können.

Systemtabellen

Verschiedene Programmdefinitionen können in Tabellen statt im Code gespeichert werden. So gewinnt das Programm nicht nur an Übersichtlichkeit, sondern auch an Flexibilität. Einige Änderungen können dann nur durch die Modifikation der Einträge in der Tabelle statt der Eingriffe in den Source-Code realisiert werden.

Dynamische Definition von Objekten

Einige Objekte (vor allem Tabellen und Abfragen) können erst beim Programmablauf dynamisch erstellt oder geändert werden. So müssen z. B. die Ergebnistabellen für verschiedene statistische Auswertungen nicht mitgeliefert werden, sondern können erst beim ersten Programmstart mit der aktuellen Datenversion berechnet werden. Weiter ist es möglich, das Layout oder das Verhalten des Programms den konkreten Aufrufbedingungen (Userberechtigung, Jahr der Verarbeitung) automatisch anzupassen. Die Abfragen, die nicht gleichzeitig zum Einsatz kommen, müssen nicht alle existieren. Es kann ein gemeinsames Objekt während des Programmablaufs überschrieben werden. So reduziert man die Anzahl der Datenbankobjekte, die Datenbank wird kleiner und übersichtlicher.

Dynamische Abfragen

Eine Access-Abfrage ist im Prinzip nichts anderes als ein gespeicherter SQL-Ausdruck. Diesen Ausdruck kann man bequem mit der graphischen Oberfläche des Abfragengenerators generieren oder in der SQL-Ansicht manuell eingeben, bzw. modifizieren. Die Abfrage bleibt statisch, das bedeutet, dass sich ihre Definition nur durch den Eingriff des Entwicklers verändern kann.

Bei einer dynamischen Abfrage wird der SQL-Ausdruck zuerst in einem Code-Modul in einer String-Variablen gebildet und dann der Eigenschaft `SQL` der Abfragedefinition zugewiesen. (Die Variable wird in den Beispielen `s` benannt, damit bei längeren Ausdrücken möglichst viel von dem tatsächlichen Inhalt auf der Bildschirmbreite sichtbar bleibt.)

Für die Erstellung einer dynamischen Abfrage sind mehrere Methoden möglich:

Eine benannte Abfrage wird verändert.

Dieser Fall ist dann sinnvoll, wenn die Abfrage nach der Modifikation mehrmals im Programm verwendet wird, bzw. unbeschränkt zur Verfügung stehen soll.

```
CurrentDb.QueryDefs("qrdAbfrage").sql = s
```

Die Abfrage hat eine zeitlich begrenzte Bedeutung

Sie dient z. B. dient zur Erstellung einer berechneten Tabelle. Nach der Ausführung ist es nicht notwendig sie weiter zu speichern. In dem Fall kann eine temporäre namenslose Abfrage erstellt und sofort ausgeführt werden.

```
CurrentDb.CreateQueryDef("", s).Execute
```

Eine temporäre benannte Abfrage

Die dritte Möglichkeit stellt eine sinnvolle Kombination der ersten und der zweiten Methode dar. Eine benannte dynamische Abfrage wird neu erstellt. Da sie nicht sofort gelöscht wird, kann man den SQL-Ausdruck auch später überprüfen (es ist günstig für die Fehlersuche). Der Vorgang wurde in eine allgemeine Prozedur `QueryCreate` eingekapselt, die in den Beispielen mehrmals eingesetzt wird.

Prozedur QueryCreate

Die Prozedur wird mit zwei Parametern aufgerufen:

- `qrdname` Name der erstellten Abfrage
- `s` Der SQL-Ausdruck.

Die Abfrage wird neu erstellt, daher ist es notwendig, das eventuell existierende gleichnamige Objekt zuerst zu löschen. Es kann sich nicht nur um eine Abfrage, sondern auch um eine berechnete Tabelle handeln (siehe das letzte Beispiel). Deshalb werden zur Sicherheit beide Löschanweisungen ausgegeben. Die Fehlerbehandlung wird vorübergehend ausgeschaltet, damit keine Fehlermeldung angezeigt wird, wenn das gelöschte Objekt nicht existiert.

```
Public Sub QueryCreate (qrdname$, s$)
...
Set db = CurrentDb
On Error Resume Next
db.TableDefs.Delete qrdname
db.QueryDefs.Delete qrdname
On Error GoTo 0
Set qrd1 = db.CreateQueryDef(qrdname, s)
End Sub
```

Vorteile dynamischer Abfragen

Code-Editor ist besser als Abfragen-Editor

Das Editieren des SQL-Ausdrucks direkt in der SQL-Ansicht des Abfragengenerators ist mühsam, weil der Text nicht formatiert werden kann, bzw. bei jedem Öffnen der Abfrage immer neu umformatiert wird. Im Codemodul dagegen kann z.B. jeder Feldausdruck in einer Zeile stehen. Außerdem können beliebige Kommentare eingefügt werden.

SQL-Ausdruck mehrmals verwenden

Wenn mehrere Abfragen gemeinsame Teile des SQL-Ausdrucks enthalten, können diese Teile in extra Variablen gespeichert und bei der Erstellung der gesamten Ausdrücke wiederverwendet werden.

If-, Case-Anweisungen

Die Abfrage kann mit If- oder Case-Anweisungen auch strukturell verändert werden, z.B. andere Tabellen verknüpfen, andere Felder enthalten, usw., je nach dem welche konkreten Werten die verschiedenen Parameter beim Programmablauf haben.

Konkrete Werte statt Verweisen

Die Werte von globalen Variablen oder Inhalten der Steuerelemente können in

die dynamische Abfrage eingefügt werden. Die Abfrage wird dann schneller ausgeführt und bleibt funktionsfähig, auch wenn z.B. ein Selektionsformular nicht mehr offen ist.

Feldausdrücke in einer Schleife generieren
Bei der gezielten Benennung von Tabellenfeldern (der Name enthält eine Reihenfolgenummer), können Teile des SQL-Ausdrucks in einer For-Next-Schleife generiert werden.

Einsatz von Systemtabellen
Die für die Abfragendefinition notwendigen Angaben können aus einer Systemtabelle ausgelesen werden.

Access-Funktion `Eval()`
Auf dem Weg zur Modularisierung und Parametrisierung eines Access-Programms kann man noch weiter gehen, als den SQL-Ausdruck einer dynamisch erstellten Abfrage zuweisen und diese dann verwenden oder ausführen. Der SQL-Ausdruck kann auch als Rückgabewert einer sog. SQL-Funktion vermittelt werden.

Die SQL-Funktion muss nicht im Code explizit aufgerufen werden, sondern läßt sich mit der Funktion `Eval()` auswerten. Dabei kann man den Namen der SQL-Funktion z.B. in einer Systemtabelle speichern. Eine allgemeine Funktion kann beim Programmablauf den (bei der Entwicklung noch nicht festgelegten) Funktionsnamen aus der Tabelle auslesen und die SQL-Funktion mit `Eval()` auswerten.

Beispiel 1: Datenselektion

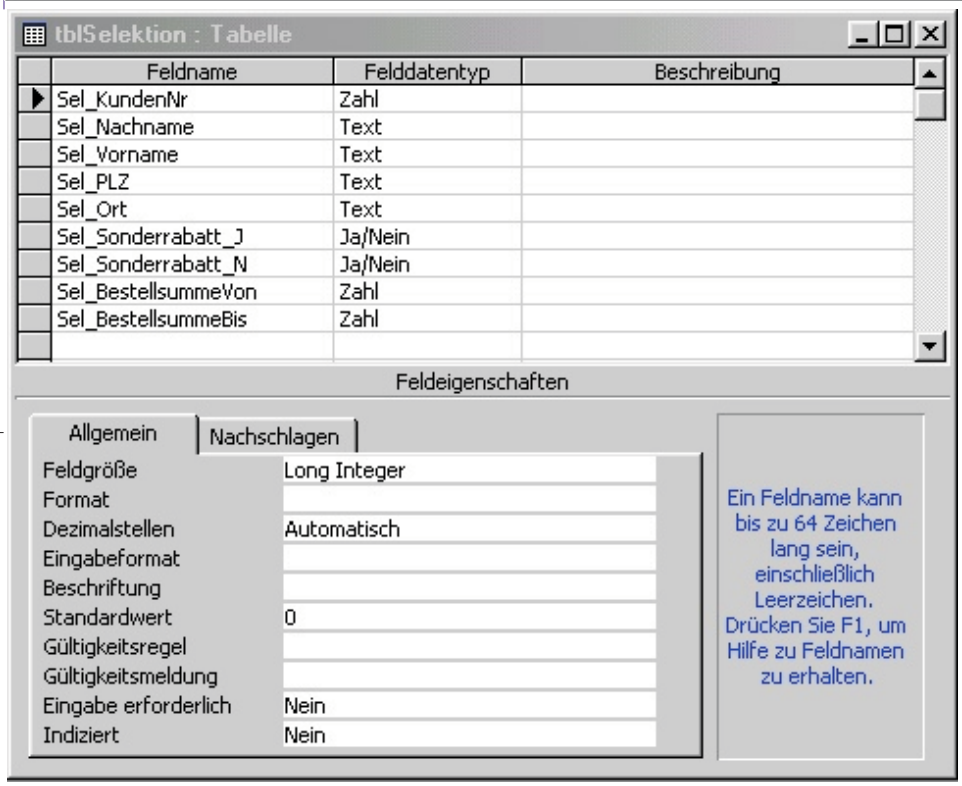
Die im weiteren Text besprochenen Beispiel-Objekte stehen in der Access97-Datenbank **`DynQuery.mdb`** zur Verfügung. Alle Beispiele sind möglichst einfach, damit durch ein komplexes Programm die Anschaulichkeit der programmtechnischen Tipps nicht verloren geht. Aus dem gleichen Grund wurde auch die in der Praxis unentbehrliche Fehlerbehandlung meistens weggelassen.

Das erste Beispiel stellt eine Lösung der Datenselektion dar. Das Bild (***Bild oben***) präsentiert eine bereits erfolgte Selektion der Anfangszeichen des Nachnamen und der Postleitzahl und der Bedingung für Sonderrabatt:

Zum Testen ist das Formular `frmKunden` zu öffnen. Das entsprechende Programm-Code ist im Klassenmodul des Formulars und im Modul `modSelektion` enthalten.

Die Felder für die Eingabe der Selektionskriterien sind an die Tabelle `tblSelektion` gebunden. Ihre Struktur entspricht den vorhandenen Selektionsfeldern (***Bild unten***).

In der Tabelle `stblKriterien` werden die Angaben für die dynamische Erstellung der Selektionsabfrage gespeichert. Die Namen beider Tabellen sind als Konstanten im Modul `g` gespeichert (siehe oben).



Die Felder der Kriterientabelle `stblKriterien` haben folgende Bedeutung:

Feld	Bedeutung
`KritGruppe`	Gruppenbezeichnung der zusammengehörigen Selektionskriterien
`KritIndex`	Ordnungsbegriff
`DatenFeld`	Name des Datenfeldes in der Quellentabelle oder -Abfrage
`Operator`	Vergleichsoperator
`SelektionsFeld`	Name des Feldes in der Selektionstabelle

Die Kriterientabelle für dieses Beispiels sieht man im ***Bild auf der folgenden Seite***.

Die Tabelle `qrdBestellSumme` ist eine gespeicherte Abfrage, die für dieses Beispiel aus den Kunden- und Bestelldaten erstellt wurde. Die Ergebnisse der Selektion sind unten im Unterformular sichtbar.

Datensätze auswählen
Nach der Eingabe der Kriterien und Aktivieren der Schaltfläche ***"Auswählen"*** wird die Funktion `SQL_Selektion()` aufgerufen. Sie hat zwei Parameter:

`QuellName`
Name der Quellentabelle oder -Abfrage

`Gruppe`
Name der Kriteriengruppe

Die Selektionstabelle mit den gesuchten Kriterien wird geöffnet:

```
Set selrec =
db.OpenRecordset(g.TABELLE_SELEKTION,
dbOpenSnapshot)
selrec.MoveFirst
```

Die notwendigen Datensätze aus der Kriterientabelle werden ausgelesen und in einer Schleife bearbeitet. Die Variable `sErg` enthält den dynamisch erstellten SQL-Ausdruck. Mit jedem Selektionskriterium wird er um einen Teil der WHERE-Bedingung erweitert. Der Vorgang unterscheidet sich beim ersten Kriterium, in dem Fall ist die Variable krit1 auf `True` gesetzt..

```
s = "SELECT * FROM " + g.TABELLE_KRITERIEN
s = s + " WHERE KritGruppe ='" & Gruppe &
"'"
s = s + " ORDER BY KritIndex;"
Set kritrec = db.OpenRecordset(s,
dbOpenSnapshot)
...
    kritrec.MoveFirst
    krit1 = True
    sErg = "SELECT * FROM " + QuellName
    Do While Not kritrec.EOF
```

Für jeden Satz der Kriterientabelle wird zuerst überprüft, ob im zugehörigen Selektionsfeld etwas eingegeben wurde (Test auf Null-Wert). Wenn ja, dann wird je nach dem Typ des Feldes in der Selektionstabelle der Teil des WHERE-Ausdrucks formuliert.

```
s = ""
KritWert =
selrec.Fields(kritrec!SelektionsFeld).Value
If Not IsNull(KritWert) Then
  oper = Trim(kritrec!Operator)
  Select Case VarType(KritWert)
    Case vbString
      KritWert = Trim$(KritWert)
      If (Right$(KritWert, 1)  "*")
      And (oper = "like") Then
        KritWert = KritWert & "*"
    ...
End If
s = kritrec!Datenfeld + " " + oper + " '" +
KritWert + "'"
...
```

Am Ende der Schleife wird der Teilausdruck `s` mit Berücksichtigung der Variablen `krit1` an die Ergebnisvariable `sErg` angehängt und der nächste Satz der Kriterientabelle bearbeitet.

```
If krit1 Then
  sErg = sErg + " WHERE (" + s + ")"
  krit1 = False
Else
  sErg = sErg + " AND " + s
End If
      ...
      kritrec.MoveNext
```

Das Ergebnis dieser Funktion ist ein SQL-Ausdruck, der direkt als RecordSource dem Unterformular zugewiesen werden kann. Damit die Funktionalität besser getestet, bzw. der SQL-Ausdruck angeschaut werden kann, wird mit Hilfe der oben beschriebenen Prozedur `QueryCreate` die Abfrage `qrdKundenSelektion` dynamisch erstellt und als `RecordSource` des Unterformulars verwendet.

```
QueryCreate "qrdKundenSelektion", s
Me.fraKunden.Form.RecordSource =
"qrdKundenSelektion"
Kriterien entfernen
```

Die Aufgabe der Schaltfläche ***"Kriterien entfernen"*** ist, den Inhalt aller Steuerelemente zu entfernen und die Selektion erneut ausführen, damit im Unterformular alle Datensätze angezeigt werden. Die Steuerelemente müssen nicht mit ihren Namen angesprochen werden, weil nur ihr Typ relevant ist. Sie werden in einer Schleife mit der allgemeinen Prozedur `ControlsInit` abgearbeitet.

stblKriterien : Tabelle

KritGruppe	KritIndex	DatenFeld	Operator	SelektionsFeld
Sel_Kunden	1	KundenNr	=	Sel_KundenNr
Sel_Kunden	2	Nachname	LIKE	Sel_Nachname
Sel_Kunden	3	Vorname	LIKE	Sel_Vorname
Sel_Kunden	4	PLZ	LIKE	Sel_PLZ
Sel_Kunden	5	Ort	LIKE	Sel_Ort
Sel_Kunden	6	Sonderrabatt	=	Sel_Sonderrabatt_J
Sel_Kunden	7	Sonderrabatt	<>	Sel_Sonderrabatt_N
Sel_Kunden	8	Bestellsumme	>=	Sel_BestellsummeVon
Sel_Kunden	9	Bestellsumme	<=	Sel_BestellsummeBis
	0			

Datensatz: 1 von 9

```
Public Sub ControlsInit(frm As Form)
Dim ctl As Control
  For Each ctl In frm.Controls
    Select Case ctl.ControlType
      Case acTextBox
        ctl.Value = Null
      Case acCheckBox
        ctl.Value = False
    End Select
  Next ctl
End Sub
```

Der Verweis auf das konkrete Selektionsformular, das initialisiert werden soll, bekommt die Funktion mittels der Parameterübergabe der Eigenschaft `Me`. Danach wird die Event-Prozedur der Schalfläche ***"Auswählen"*** aufgerufen, das hat die Aktualisierung der Anzeige im Unterformular zur Folge.

```
Private Sub cmdInit_Click()
  ControlsInit Me
  cmdSelect_Click
End Sub
```

Die vorgestellte Lösung scheint unnötig kompliziert zu sein, man kann doch auf die Selektionsfelder im Formular direkt in einer Abfrage verweisen. Diese Technik bringt aber beim Einsatz in komplexen Programmen folgende Vorteile:

Kriterien bleiben gespeichert
Da die Selektionsfelder an eine Tabelle gebunden sind, bleiben die ausgewählten Kriterien auch nach dem Schließen des Selektionsformulars gespeichert und können eventuell teilweise für die nächste Selektion verwendet werden.

Kriterien gemeinsam nutzen
Die Kriterien können zwischen mehreren Selektionsformularen automatisch übertragen werden. Das kann für die Auswertung der gleichen Datensätze in verschiedenen Teilen des Programms günstig sein.

Allgemeine Funktion
Die Funktion `SQL_Selektion()` kann für jede beliebige Selektion im Programm verwendet werden. Es sind nur die notwendigen Datensätze in die Tabelle `stblKriterien` einzufügen und die Aufrufparameter dementsprechend anzupassen.

Selektierte Daten stehen zur Verfügung
Man kann die dynamisch erstellte Abfrage jederzeit überprüfen. Die ausgewählten Daten können auch z.B. außerhalb des Programms exportiert werden, usw.

Beispiel 2: Import von Excel-Dateien

Eine Excel-Tabelle kann man einfach in Access manuell importieren. Der standardmäßige Vorgang hat aber seine Schwächen:

Feldnamen
Es kann ausgewählt werden, ob die erste Zeile der Excel-Tabelle die Feldnamen enthält oder nicht. Keine der Möglichkeiten passt, wenn die Tabelle mehrere Zeilen mit Aufschriften enthält. Außerdem müssen die Excel-Spaltennamen den Access-Namenskonventionen entsprechen, sonst können sie nicht übernommen werden.

Feldtypen
Datentyp der importierten Felder kann nicht geändert werden. Es können keine Berechnungen neuer Felder oder Konvertierungen der übernommenen Werte vorgenommen werden.

Import nicht einfach wiederholbar
Die manuell erstellte Import-Spezifikation kann für die Wiederholung des strukturell gleichen Imports nicht gespeichert werden.

Importierte Sätze können nicht überprüft werden
Die Zeilen der Excel-Tabelle können nicht sofort beim Import überprüft und die ungültigen Zeilen eventuell ausgefiltert werden.

Die oben genannten Punkte sind wichtig, wenn man z.B. Daten aus mehreren Excel-Lösungen in ein gemeinsames Access-Programm übernehmen soll. Die Excel-Tabellen haben oft auch für logisch gleiche Daten unterschiedliche Strukturen, redundante Spalten und andere Probleme, die den standardmäßigen Import erschweren.

Zweistufiges Import-Verfahren
Im nachfolgenden Beispiel wird ein Verfahren zum Exportieren von Excel-Tabellen vorgestellt, das übersichtlicher und funktionell umfangreicher als der standardmäßige Access-Import ist. Es wird eine zweistufige Technik verwendet:

- Aus der Excel-Tabelle wird ein Bereich explizit ausgewählt in eine temporäre Access-Tabelle `tblImportTemp` importiert.
- Die Zieltabelle wird bei jedem Import mit einer dynamischen Abfrage aus der Tabelle `tblImportTemp` neu erstellt. Die für den SQL-Ausdruck notwendigen Angaben

werden der Systemtabelle `stblImport` entnommen.

Das Makro `TestImport_aus_Excel` in der Beispieldatenbank ruft die Funktion `ExcelImport()` auf, die beide Stufen des Imports durchführt. Der Name der importierten Excel-Datei **`Test.xls`** wird in diesem Beispiel direkt als Parameter eingegeben, auch wenn es im wirklichen Programm z.B. mittels eines Öffnen-Dialogs abgefragt werden könnte.

Tabelle `stblImport`

Die Tabelle `stblImport` speichert in jedem Satz die Angaben für die Erstellung eines Feldes der Zieltabelle. Sie besteht aus folgenden Feldern:

Name	Typ	Beschreibung
`ImportSpezifikation`	**`Text`**	Name der Struktur der Zieltabelle
`FeldIndex`	**`Integer`**	Reihenfolge des Feldes
`FeldName`	**`Text`**	Name des Zielfeldes
`FeldFormel`	**`Text`**	Formel für die Berechnung des Inhalts

Den Inhalt für die Tabelle `stblImport` im vorgelegte Beispiel sieht man auf dem ***Bild auf dieser Seite***.

Der Name der Importspezifikation kann, muss aber nicht gleich dem Namen der Zieltabelle sein. Es wird vorausgesetzt, dass eine bestimmte Import-Struktur für mehrere physikalische Tabellen gültig sein kann.

Funktion `ImportExcel()`

Die Funktion `ImportExcel()` wird mit folgenden Parametern aufgerufen:

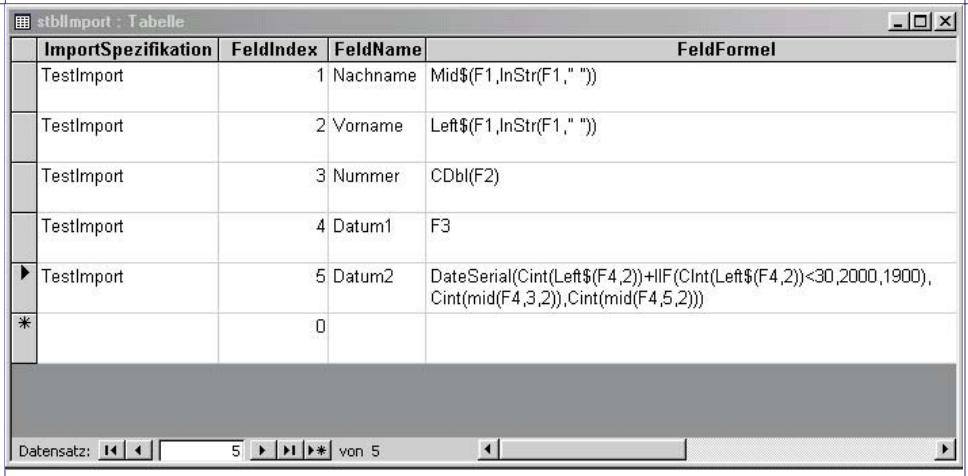
- xlsname Pfad und Name der importierten Excel-Datei
- `importspez` Name der Importspezifikation
- `zieltab` Name der Access-Tabelle
- `excelrange` Der ausgewählte Excel-Bereich
- `swhere` Zusätzlicher Filterausdruck

Zuerst werden die temporäre Tabelle `tblImportTemp` und die Zieltabelle gelöscht. Der ausgewählte Excel-Bereich wird dann in die neu erstellte Tabelle `tblImportTemp` importiert.

```
DoCmd.TransferSpreadsheet acImport,
acSpreadsheetTypeExcel97, "tblImportTemp",
xlsname, False, excelrange
```

In der ersten Stufe wird der Datentyp des Access-Felder automatisch je nach dem Inhalt der Excel-Spalten festgelegt. Wenn in einer Zelle der nummerischen Spalte ein nicht-nummerischer Ausdruck steht, wird ein Access-Feld vom Typ **`Text`** erstellt. Eine mögliche Lösung eines solchen Problems wird später erklärt.

Weiter wird überprüft, ob überhaupt einige Datensätze übernommen wurden und wenn nicht, wird die Funktion beendet. Der Rückgabewert bleibt in dem Fall `False`. Durch die Abfrage des Rückgabewertes können in dem aufrufenden Programmteil weitere notwendige Aktionen ausgelöst werden.



Die zweite Stufe beginnt mit dem Auslesen der Datensätze für die jeweilige Spezifikation aus der Tabelle `stblImport`.

```
Set rec = db.OpenRecordset("SELECT * FROM
stblImport WHERE ImportSpezifikation = '" +
importspez + "' ORDER BY FeldIndex",
DB_OPEN_SNAPSHOT)
```

Alle Sätze des Recordsets werden in einer Schleife bearbeitet und der erste Teil des SQL-Ausdrucks für das Erstellen der Zieltabelle gebildet.

```
s = "SELECT "
Do While Not rec.EOF
  s = s + rec!FeldFormel + " AS [" +
      rec!FeldName + "],"
  rec.MoveNext
Loop
```

Nach dem Abarbeiten aller Sätze wird das letzte Zeichen (Komma) entfernt und die Anweisungen `INTO` und `FROM` hinzugefügt.

```
s = Left$(s, Len(s) - 1)
s = s + " INTO " + zieltab
s = s + " FROM tblImportTemp"
```

Der Parameter `swhere` kann eine zusätzliche Filterbedingung enthalten. Mit dieser Bedingung können ungültige Datensätze aus der Excel-Tabelle sofort beim Import übersprungen werden. In unserem Beispiel werden die Datensätze nicht importiert, bei denen das zweite Feld nicht nummerisch ist.

```
If Len(swhere) > 0 Then
  s = s + " WHERE " + swhere
End If
```

Der komplette SQL-Ausdruck wird einer temporären Abfrage zugewiesen und die wird sofort ausgeführt. So wird die Zieltabelle erstellt. Die Abfrage `qrdTemp` wird statt einer temporären unbenannten Abfrage für die einfachere Fehlersuche (siehe oben) verwendet.

```
Set qdf = db.QueryDefs("qrdTemp")
qdf.SQL = s
qdf.Close
qdf.Execute
Feldformeln
```

Grundsätzlich kann man sagen, dass eine Feldformel alle SQL-syntaktisch richtige Ausdrücke enthalten kann. Bei den nummerischen Feldern ist es empfohlen, den Feldtyp mit Konvertierungsfunktionen wie z.B. `CInt()`, `CDbl()`, usw. explizit festzulegen und die Datensätze mit möglichen nicht-nummerischen Inhalten mit der Zusatz-Filterbedingung auszuschließen. Als Namen der Felder der Tabelle `tblImportTemp` treten die vom Access automatisch generierten Namen `F1`, `F2`, `F3`, usw. auf. Die in diesem Beispiel eingesetzten Formeln werden jetzt aufgelistet.

Das erste Feld (**`Vorname`** und **`Nachname`**) wird in zwei getrennte Feldern (**`Nachname`**, **`Vorname`**) gesplittet.

```
Mid$(F1,InStr(F1," "))
Left$(F1,InStr(F1," "))
```

Das zweite Feld wird mit `CDbl()` in eine Zahl umgewandelt und das dritte Datumsfeld wird einfach ohne Veränderung übernommen.

Das letzte Feld der Excel-Tabelle enthält ein Textfeld mit Datum im Format **`JJMMTT`**. Der Wert wird in ein Datumsfeld umgewandelt, dabei wird auch das Y2K-Problem mit einem Grenzjahr gelöst.

```
DateSerial(Cint(Left$(F4,2))+
IIF(CInt(Left$(F4,2))<30,2000,1900),
Cint(mid(F4,3,2)),Cint(mid(F4,5,2)))
```

Data-Repository

Die Speicherung der Angaben über alle Importstrukturen in der Tabelle `stblImport` hat ein wichtiges Nebeneffekt. Es entsteht automatisch ein Data-Repository. Wenn die Tabelle nach dem Feld **`FeldName`** sortiert wird, kann schnell gefunden werden, in welcher Struktur sich ein bestimmtes importiertes Feld befindet und wie es aus dem Excel berechnet wurde.

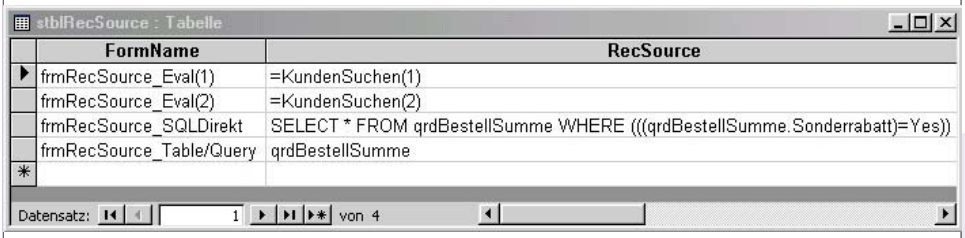
Beispiel 3: Formular mit einer dynamisch definierten Datenquelle

Manchmal verwendet man im Programm Formulare, die das gleiche Layout haben, andere Daten aber präsentieren sollen. Typisch ist z. B. der Fall, wenn die gleiche Auswertung für das aktuelle und das vorige Jahr implementiert wird.

Die Datenquelle für ein Formular kann in der Eigenschaft RecordSource angegeben werden. Eine andere Möglichkeit ist, die Eigenschaft erst beim Öffnen des Formulars (standardmäßig beim Ereignis `OnLoad`) zu befüllen. Die dafür notwendigen Angaben müssen nicht in jedem einzelnen Formular kodiert werden, sondern können in einer Systemtabelle für alle Formulare des Programms gespeichert werden. (Alles was hier über Formulare geschrieben wurde, gilt natürlich auch für die Berichte).

Beispielformulare

Im vorgelegten Beispiel können vier gleiche Formulare `frmRecourdSource_...` geöffnet werden. Die Tabelle `stblRecordSource` enthält zwei Spalten - den Formularnamen und die Datenquelle und ist mit folgenden Angaben befüllt:.



stblRecSource : Tabelle

FormName	RecSource
frmRecSource_Eval(1)	=KundenSuchen(1)
frmRecSource_Eval(2)	=KundenSuchen(2)
frmRecSource_SQLDirekt	SELECT * FROM qrdBestellSumme WHERE (((qrdBestellSumme.Sonderrabatt)=Yes))
frmRecSource_Table/Query	qrdBestellSumme

Datensatz: 1 von 4

Jedes Formular holt sich die notwendigen Informationen über seine Daten aus dieser Tabelle mit der Hilfe einer allgemeinen Funktion `DefineRecSource()`, die im Ereignis `OnLoad` aufgerufen wird.

```
Private Sub Form_Load()
  Me.RecordSource = DefineRecSource(Me.Name)
End Sub
```

Für die Formulare `frmRecSource_Table/Query` und `frmRecSource_SQLDirekt` steht in der Tabelle `stblRecordSource` eigentlich der gleiche Inhalt, den man auch direkt in die Formulareigenschaft schreiben kann. Im ersten Fall ist es der Name einer Tabelle oder Abfrage, im zweiten Fall ein direkter SQL-Ausdruck.

Der Trick, der hier präsentiert wird, ist in den zwei restlichen Zeilen der Systemtabelle zu sehen. Es steht hier nach dem Zeichen "=" (wie bei einem Ereignis im Eigenschaftsfenster) der Name einer Funktion, die auch einen (oder mehrere) Parameter annehmen kann.

Die Funktion `KundenSuchen()` liegt im Modul `modRecSource` und liefert einfach den notwendigen SQL-Ausdruck für die Anzeige der Kunden, derer Bestellsumme je nach dem Wert des Parameters unter oder über einer bestimmten Grenze liegt.

`Funktion DefineRecSource()`

Wie kommt der Rückgabewert der globalen Funktion aus einem Modul in die Datenquelle des Formulars? In der Funktion `DefineRecSource()` wird zuerst der Inhalt des Felds `RecSource` aus der Tabelle `stblRecordSource` ausgelesen:

```
s = DLookup("RecSource", "stblRecSource",
"FormName='" + frmname + "'")
```

Wenn der Ausdruck auf der ersten Position das Zeichen "=" enthält, wird er ab dem zweiten Zeichen mit der Funktion `Eval()` ausgewertet, sonst nur zurückgegeben.

```
  If Left$(s, 1) = "=" Then
    DefineRecSource = Eval(Mid$(s, 2))
  Else
    DefineRecSource = s
  End If
```

In der Praxis kann eine Systemtabelle wie die `stblRecordSource` von einem übergeordneten System verwendet werden, das dann sowohl das Formular als auch seine Datenquelle dynamisch öffnen kann. In einem der nächsten Artikel wird die Parametrisierung der Arbeit mit einem Multipage-Objekt basierend auf dem gerade vorgestellten Konzept präsentiert.

Beispiel 4: Tabellen/Abfragen erstellen

Komplexe Auswertungen, die mehrere Tabellen verknüpfen (JOIN) müssen, um die Ergebnisse zu berechnen, laufen oft zu langsam. Es läßt sich Einiges optimieren, wenn es aber nicht besser geht, bleibt nur eine Möglichkeit: die komplexesten Abfragen als Tabellen zwischenspeichern. Die Lösung ist ganz gut annehmbar, wenn der Anwender die Daten z.b. nur einmal im Monat geliefert bekommt und sie nur auswertet, also keine Änderungen macht. Eine verlorene halbe Stunde einmal im Monat lohnt, dafür läuft das Programm dann schneller.

Abfrage als Tabelle speichern ?

Es ist in der Entwicklungsphase nicht immer möglich das Verhalten des Programms richtig abzuschätzen und entscheiden, ob man zu der o.g. Maßnahme greifen soll oder nicht. Außerdem läuft das Produkt einmal auf einem langsameren, einmal auf einem schnelleren Rechner. Optimal wäre also die Möglichkeit, zwischen beiden Techniken der Datenbereitstellung ohne besonderen Programmieraufwand umschalten zu können.

In der Beispieldatenbank sind stehen zwei Makros zur Verfügung, mit denen man die o.g. Funktionalität testen kann.

`Abfrage_qrdBestellOrt_erstellen`

erstellt aus der Tabelle `qrdBestellSumme` eine Selektionsabfrage `qrdBestellOrt`

`Tabelle_qrdBestellOrt_erstellen`

speichert die gleichen Selektion in einer Tabelle.

`Funktion TabQueryCreate()`

Beide Makros rufen die Funktion `TabQueryCreate()` auf. Der erste Aufrufparameter ist der Name der SQL-Funktion, die weiteren Parameter können in Anzahl und Variablentyp beliebig sein und werden also als ein **`ParamArray`** deklariert.

Das erste Element der Arrays `p(0)` bestimmt, ob eine Tabelle oder eine Abfrage erstellt wird, das zweite `p(1)` übergibt den Namen des erstellten Objekts.

Alle Parameter werden in einen String konvertiert, der dann mit dem Namen der SQL-Funktion an die Funktion `Eval()` übergeben wird.

```
ps = ""
For i = 0 To UBound(p)
  If VarType(p(i)) = vbString Then
    ps = ps + IIf(i > 0, ",", "") + "'" +
         CStr(p(i)) + "'"
  Else
    ps = ps + IIf(i > 0, ",", "") +
         CStr(p(i))
  End If
Next i
s = Eval(sqlfunname + "(" + ps + ")")
```

Der weitere Vorgang unterscheidet sich je nachdem, ob eine Tabelle oder Abfrage erwünscht ist. Im ersten Fall wird das Zielobjekt zuerst gelöscht und dann wird mit Hilfe der temporären Abfrage `qrdTemp` die Tabelle neu erstellt. Im zweiten Fall wird mit der Prozedur `QueryCreate` die Selektionsabfrage angelegt.

```
If p(0) Then
  On Error Resume Next
  db.TableDefs.Delete p(1)
  db.QueryDefs.Delete p(1)
  On Error GoTo 0
  QueryCreate "qrdTemp", s
  db.QueryDefs.Refresh
  db.QueryDefs("qrdTemp").Execute
Else
  objname = p(1)
  QueryCreate objname, s
End If
```

Funktion SQL_OrtSumme()

In diesem Beispiel muss die SQL-Funktion `SQL_OrtSumme()` den richtigen SQL-Ausdruck sowohl für die Selektions- als auch für die Tabellenerstellungsabfrage bereitstellen. Sie wird mit zwei Parametern aufgerufen:

- `tabelle` `True`, wenn eine Tabelle erstellt wird
- `tabname` Name der erstellten Tabelle (relevant nur, wenn `tabelle=True`)

Der SQL-Ausdruck einer Tabellenerstellungsabfrage unterscheidet sich von dem einer Selektionsabfrage nur in der zusätzlichen Anweisung `INTO`, sonst hat die Funktion den schon bekannten Aufbau.

```
If tabelle Then
  s = s + " INTO " + tabname
End If
```

In der Praxis kann als Parameter `tabelle` z.B. eine globale Variable verwendet werden. So kann mit der Einstellung ihres Wertes das Verhalten des gesamten Programms verändert werden.

Schlusswort

Je komplexer die Aufgabe ist, desto wichtiger ist es, nachzudenken und weniger Code zu tippen. Die scheinbar verlorene Zeit im Vergleich zu der schnellen "ein paar Mausklick"-Lösung bekommt man bei den späteren Änderungen oder Erweiterungen des Programms mehrfach zurück.

Ich hoffe, Ihnen mit diesen Tipps beim Programmieren von Datenbanken geholfen zu haben. Die Beispieldatenbank zu diesem Artikel finden Sie bei der Web-Version dieses Artikels unter http://pcnews.at/ins/pcn/0xx/07x/074/-074.htm.