Überlegungen zur Entwicklung von Access-Anwendungen

A. Prinzipielle Strategien

1. Zuerst die Datenstruktur!

Das Design einer Access-Anwendung beginnt bei der Datenstruktur.

Eine Vorgangsweise, die sich bewährt hat:

1. Die Datenstruktur auf Papier entwerfen
    - Welche Daten müssen gespeichert werden?
    - Welche Zusammenhänge bestehen zwischen den Daten
    - Großes Papier!
    - Viel Papier!
2. Die Datenstruktur im Backend implementieren
    - Benennungsschema festlegen und strikt einhalten
    - Tabellen
    - Felder
    - Datentypen
    - Eingabe erforderlich
    - Leere Zeichenfolge (geänderter Standard ab A2002!)
    - Indizes (eindeutige Felder, Mehrfelderindizes)
    - Gültigkeitsregeln (auch auf Tabellenebene)
    - Referenzielle Integrität
    - Von vielen verpönt, IMHO dennoch sehr zu empfehlen:
        - Beschriftungen
        - Nachschlagefelder
3. Mithilfe einiger Testdatensätze typische Szenarien auf Tabellenebene durchspielen
    - Kann ich jede Information eingeben?
    - Sind unsinnige Werte möglichst verhindert?
    - Muss ich ggflls. weiter normalisieren?

> Überlegungen zur GUI sind hier ABSOLUT FEHL am Platz!

2. Fehler sollen gar nicht passieren können

„Freiheiten möglichst früh beschränken"

a. Kleinst möglicher Gültigkeitsbereich

Die Komplexität und Wartbarkeit einer Software hängt u.a. sehr stark davon ab,

- wie klar die Teile von einander zu unterscheiden sind
- dass die Teile ihre Implementierungsdetails voreinander verbergen
- dass die Schnittstellen zwischen den Teilen möglichst eng gestaltet sind.

Daher

- Die Sichtbarkeit von Variablen und Methoden möglichst stark einschränken (interne Details Private!)
- Alle Zugriffsmodifizierer explizit angeben (Default ist meist Public!)
- Bei Systemen aus mehreren Komponenten kann auch der Modifizierer Friend (sichtbar innerhalb der Komponente) sinnvoll sein.

b. Parameter nur bewusst und explizit ByRef definieren

Funktion mit einem Parameter (implizite Übergabe als Referenz):

```
Private Function GetSteuer(Summe As Currency) As Currency
   Summe = ...      ' Hier kann der Wert der übergebenen Variable geändert werden!
End Function
```

Aufruf:

```
   Dim curSumme As Currency

   curSumme = ...
   ... = GetSteuer(curSumme)
   ' Hier könnte curSumme bereits verändert worden sein!
```

Stattdessen:

```
Private Function GetSteuer(ByVal Summe As Currency) As Currency
   Summe = ...   ' Hier kann nur der Wert der lokalen Variable geändert werden,
                 ' der Wert der übergebenen Variable kann nicht verändert werden.
End Function
```

Da ByRef in Visual Basic Classic der Standard ist, aber moistens nur ByVal benötigt wird, ByVal explizit angeben. Nur wenn tatsächlich eine Übergabe per Referenz benötigt wird, das Schlüsselwort ByRef angeben.

3. Für die eigentliche Problemlösung auf logischer Ebene arbeiten

Auf technischer/implementatorischer Ebene:

- „Mache irgendetwas, wenn jemand auf die Schaltfläche btnSave klickt."

Auf logischer Ebene

- „Mache irgendetwas, wenn der Benutzer speichern will."

Umsetzung in Access:

- Durch Tool-Methoden und -Klassen möglichst rasch die technische Ebene verlassen und auf die logische Ebene kommen.
- Ereignisse!

4. Für die eigentliche Problemlösung möglichst wenig programmieren

Stattdessen:

- Designer verwenden (Voreinstellungen für Formulare in der Entwurfsansicht vornehmen, Abfragen nicht per SQL in VBA zusammenbasteln, sondern gespeicherte Abfragen verwenden)
- Mithilfe von Code-Generatoren gefährliche Teile generieren lassen und auf eine vom Compiler überprüfbare Ebene bringen.
- Frameworks anwenden („gefährliche" Codeteile zentralisiert spezifizieren und dann per Eigenschaft oä darauf zugreifen)

5. Features möglichst deklarativ implementieren

Beispiel: Constraints in der Datenstruktur:

Nicht bei jedem Zugriff auf ein Feld kontrollieren, dass es nicht leer gelassen oder nachträglich geleert wird, sondern das Feld auf Tabellenebene erforderlich machen.

Aber auch: SQL

- Nicht:
  Mache eine Schleife über alle Datensätze der Kundentabelle. Wenn du bei einem Datensatz siehst, dass der Nachname mit E beginnt und die Postleitzahl eine Wiener Postleitzahl ist, dann füge Nachname, Vorname und Adresse an die Ergebnistabelle an.
- Sondern:
  Liefere mir Nachname, Vorname und Adresse aller Wiener Kunden deren Nachname mit E beginnt:
  SELECT Nachname, Vorname, Adresse
  WHERE Nachname LIKE "E*"
  AND PLZ BETWEEN 1000 AND 1999

6. Eingebaute Fehler vom Compiler erkennbar machen

a. Vermeidung nicht trivialer Literale

Triviale Literale können (je nach Kontext) sein:

- 0, 1, -1
- 2 (als Faktor oder Divisor)
- "" (leerer String)

Nichttriviale Literale

- Tabellennamen
- Formularnamen
- Dateiendungen,
- ...

wenigstens in Form von Konstanten zentral definieren:

Dort definieren, wo sie hingehören (als Eigenschaft betrachten):

- Eigenschaft der gesamten Anwendung (z.B. Titel der Anwendung für MsgBox): Als globale Konstante
- Eigenschaft eines Moduls (z.B. Name des Moduls für Fehlermeldungen): Als Modul-Konstante
- Nur in seltenen Fällen Konstanten auf Methodenebene definieren

Auch triviale Literale lassen sich manchmal vermeiden

Beispiel (For-Schleifen über alle Elemente eines Arrays):

```
For i = 0 To mc_intAnzahl - 1
   ... = ... * aZahlen(i)
Next i
```

Aussage: Eine Schleife von „einer Grenze bis zu einer anderen Grenze".

Stattdessen:

```
For i = LBound(aZahlen) To UBound(aZahlen)
   ... = ... * aZahlen(i)
Next i
```

Aussage: Eine Schleife von „über alle Elemente" -> wesentlich klarer und weniger fehleranfällig.

Im Speziellen: Dot (.) statt Bang (!)

Zugriff auf Steuerelemente und Felder in Formularen nicht über die Controls-Auflistung, sondern über die jeweilige Eigenschaft:

Beispiel:

```
Me!lblInfo = ...
```

ist nichts anderes als eine Kurzschreibweise für

```
Me.Controls.Item("lblInfo").Caption = …
```

Da ist also ein String-Literal versteckt! Wenn man sich hier vertippt oder das Steuerelement nicht mehr existiert oder umbenannt wurde, fällt der Fehler erst zur Laufzeit auf (Katastrophe). Weiters ist der Typ des Steuerelements erst zur Laufzeit bekannt (was ist, wenn das angesprochene Steuerelement kein Label (mehr) ist, und daher auch keine Eigenschaft Caption hat? -> Laufzeitfehler).

Stattdessen:

```
Me.lblInfo.Caption = ...
```

Warum funktioniert das?

Weil Access für jedes zur Entwurfszeit bereits vorhandene Steuerelement und Feld der Datensatzherkunft der Formularklasse (Form_<Formularname>) eine streng typisierte Eigenschaft verpasst. Wenn sich nun der Name oder der Typ ändern, oder das Steuerelement umbenannt oder gelöscht wird, tritt bereits beim Kompilieren ein Fehler auf.

b. Strenge Typisierung

Möglichst den „engsten" Datentyp verwenden.

Boolean statt Integer

Schlechtes Beispiel aus Access: Der Parameter Cancel in Form_Unload, Control_BeforeUpdate, etc.:

```
Private Sub Form_BeforeUpdate(Cancel As Integer)
End Sub
```

Warum hier Integer?

Besser wäre:

```
Private Sub Form_BeforeUpdate(Cancel As Boolean)
End Sub
```

Currency statt Double

Currency kann vier dezimale Nachkommastellen *exakt* abbilden, Double und Float können das nur in Spezialfällen!

Currency ist kann daher z.B. auch gut für Prozentsätze verwendet werden.

Zugriff auf anderes Formular

Beispiel: Bezug auf ein Steuerelement eines Unterformulars aus dem Hauptformular

Statt:

```
... = Me.frmsBestellungen.Form.Artikel.Value
```

Streng typisiert:

```
Dim objfrmsBestellungen As Form_frmKunden_frmsBestellungen

Set objfrmsBestellungen = Me.frmsBestellungen.Form
... = objfrmsBestellungen.Artikel.Value
Set objfrmsBestellungen = Nothing
```

Noch besser:

Keinen direkten Zugriff auf Steuerelemente eines anderen Formulars, stattdessen eine Eigenschaft vorsehen.

Im Unterformular (Beispiel von oben):

```
Public Property Get Artikelname() As String
   Artikelname = Nz(Me.Artikel.Value, "")
End Property
```

Im Hauptformular:

```
Dim objfrmsBestellungen As Form_frmKunden_frmsBestellungen

Set objfrmsBestellungen = Me.frmsBestellungen.Form
... = objfrmsBestellungen.Artikelname
Set objfrmsBestellungen = Nothing
```

B. Trennung Funktionalität – GUI

1. In Formularen nur Code zur GUI-Steuerung

Code nur dann in CBF, wenn er unmittelbar mit der Steuerung der GUI zu tun hat.
Jeglicher Businesscode unabhängig von Formularen

2. Keine Referenzierungen „aus dem Off"

... wie Forms!frmKunden!Titel

Stattdessen:

- Wenn der Code die GUI steuert, dann den Code CBF schreiben, dort Referenzierungen direkt über Me (wie Me.Titel.Value)
- Wenn Code in Modul ausgelagert ist, und dort nur einzelne Werte aus Steuerelementen des Formulars benötigt werden, dann die Werte als Parameter übergeben.
- Wenn der Code tatsächlich ein Formularobjekt benötigt, dann die Referenz auf das Formular übergeben (Me)

3. Möglichst „enge" Schnittstellen zwischen den Code-Teilen

Die Code-Teile in einer Anwendung sollten durch möglichst genau definierte (Sprachelemente!) und minimale Schnittstellen miteinander kommunizieren:

Gut geeignet:

- Parameterübergabe
- Rückgabewerte
- Ereignisse

Schlecht wartbar:

- Referenzierungen „aus dem Off"
- Globale Variablen

C. Abfragen: gespeichert vs. SQL per Code

Vorteile gespeicherter Abfragen:

- Interaktive Manipulation (Joins!)
- Interaktiver Test
- Leichter verstehbar
- Leichter wartbar
- Abfrageplan ist schon gespeichert

Nachteile gespeicherter Abfragen:

- SQL-Code ist (mit Boardmitteln) nicht durchsuchbar
- Mehr Code zum Ausführen einer Aktionsabfrage

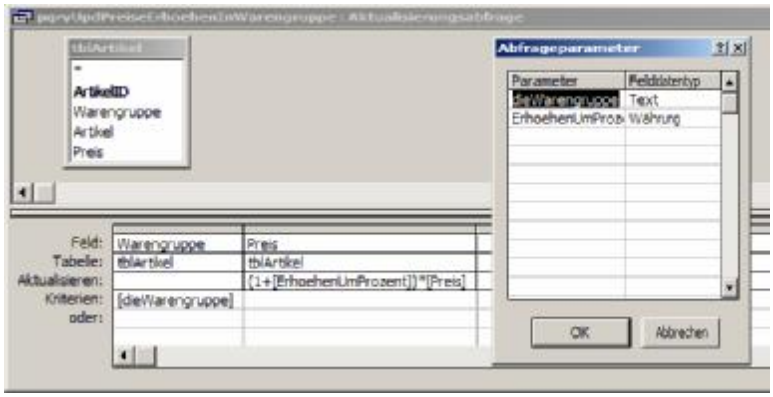
Beispiel (Ausführen einer Aktionsabfrage zur Erhöhung der Preise aller Artikel einer bestimmten Warengruppe):

Herkömmliche Variante:

```
Dim strSQL As String
Dim dbs As DAO.Database

strSQL = "UPDATE tblArtikel " & _
         "SET Preis = " & Str(1 + curErhoehung) & " * Preis " & _
         "WHERE Warengruppe = """ & strWarengruppe & """"
Set dbs = CurrentDb()
dbs.Execute strSQL, dbFailOnError
... = dbs.RecordsAffected
Set dbs = Nothing
```

Stattdessen:




```
Dim dbs As DAO.Database
Dim qdf As DAO.QueryDef

Set dbs = CurrentDb()
Set qdf = dbs.QueryDefs("pqryUpdPreiseErhoehenInWarengruppe")
qdf.Parameters("dieWarengruppe").Value = strWarengruppe
qdf.Parameters("ErhoehenUmProzent").Value = curErhoehung

qdf.Execute dbFailOnError
... = qdf.RecordsAffected

If Not (qdf Is Nothing) Then qdf.Close
Set qdf = Nothing
Set dbs = Nothing
```

D. Benennungen

1. Konsistente Benennung von Datenbankobjekten

Mit „Datenbankobjekte" sind die Tabellen, Abfragen, Formulare, Berichte und Code-Module einer Access-Anwendung (also alles, was im Datenbankfenster zu sehen ist) gemeint, nicht VB-Objekte.

a. Überlegungen für alle Typen

- Präfixe ja/nein?
- Wenn Präfixe: wie detailliert?
  - Lokale vs. temporäre Tabellen vs. Code-Tabellen
  - Auswahl vs. Aktionsabfragen
  - Unterformulare vs. Hauptformulare, etc.)
- Unterstriche?
- Nur Buchstaben und Ziffern

b. Tabellen

- Einzahl oder Mehrzahl (tblKunde oder tblKunden?)

2. Benennung von Methoden und Eigenschaften

Entscheidend ist die unterschiedliche Semantik:

- Eigenschaften geben Auskunft über einen bestimmten Datenwert und/oder legen diesen fest
- Methoden stoßen eine Aktion an

Auswirkung auf die Benennung:

- Eigenschaften nennen den Datenwert (z.B. Nachname)
- Methoden benennen eine Aktion (z.B. RechneRabatt)

Übliche Konventionen beibehalten

- GetXxx()
- SetXxx()