37. Wiener Access-Stammtisch am 12. Juni 2006

Überlegungen zur Entwicklung von Access-Anwendungen

Prinzipielle Strategien

Trennung Funktionalität – GUI

Abfragen: gespeichert vs. SQL per Code

Benennungen

softconcept, Wien
www.softconcept.at

Paul Rohorzka
paul.rohorzka@softconcept.at

Organisatorisches: **Ablauf**

- Vortrag
  - Prinzipielle Strategien
  - Trennung Funktionalität – GUI
  - Abfragen: gespeichert vs. SQL per Code
  - Benennungen
- mit
  - Demos
  - Code-Beispielen
  - Diskussion?
- Pause
- Demo

Prinzipielle Strategien

Prinzipielle Strategien:
Zuerst die Datenstruktur!

- Die Datenstruktur auf Papier entwerfen
  - Welche Daten müssen gespeichert werden?
  - Welche Zusammenhänge bestehen zwischen den Daten
  - Großes Papier!
  - Viel Papier!

Prinzipielle Strategien:
Zuerst die Datenstruktur!

- 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

Prinzipielle Strategien:
Zuerst die Datenstruktur!

- 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!

Prinzipielle Strategien:
Fehler sollen gar nicht passieren können

- Kleinst möglicher Gültigkeitsbereich
- Sichtbarkeit möglichst stark einschränken
  - interne Details: Private!
- Alle Zugriffsmodifizierer explizit angeben
  - Default ist meist Public!
- Systeme aus mehreren Komponenten:
  - Modifizierer Friend (sichtbar innerhalb der Komponente)

Prinzipielle Strategien:
Fehler sollen gar nicht passieren können

- 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
```

Prinzipielle Strategien:
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:
  - Tool-Methoden und –Klassen
    - rasch die technische Ebene verlassen und
    - auf die logische Ebene kommen.
  - Ereignisse!

Prinzipielle Strategien:
Wenig programmieren!

- Stattdessen
  - Designer verwenden
    - Voreinstellungen für Formulare in der Entwurfsansicht vornehmen
    - Abfragen nicht per SQL in VBA zusammenbasteln, sondern gespeicherte Abfragen verwenden
  - Mit 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

Prinzipielle Strategien:
Fehler vom Compiler erkennbar machen

- Vermeidung nicht trivialer Literale
- Triviale Literale (je nach Kontext):
  - 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

Prinzipielle Strategien:
Fehler vom Compiler erkennbar machen

- 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
```

  - Eine Schleife von „einer Grenze bis zu einer anderen Grenze"
- Stattdessen

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

  - Eine Schleife von „über alle Elemente"
  - Wesentlich klarer und weniger fehleranfällig

Prinzipielle Strategien:
Fehler vom Compiler erkennbar machen

- 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 = …
```

    - String-Literal versteckt!
    - Fehler erst zur Laufzeit (Katastrophe)
    - Typ des Steuerelements erst zur Laufzeit
  - Stattdessen

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

Prinzipielle Strategien: **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
  - Vier dezimale Nachkommastellen *exakt*
  - Double und Float können das nur in Spezialfällen!
  - Z.B. auch gut für Prozentsätze verwendbar

Prinzipielle Strategien: **Strenge Typisierung**

- 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

```
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
```

Trennung Funktionalität – GUI

Trennung Funktionalität – GUI:
In Formularen nur Code zur GUI-Steuerung

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

Trennung Funktionalität – GUI:
Keine Referenzierungen „aus dem Off“

- ... wie

```
Forms!frmKunden!Titel
```

- Stattdessen
  - Code steuert GUI
    - den Code CBF schreibendort
    - Referenzierungen direkt über Me (wie Me.Titel.Value)
  - Code in Modul ausgelagert, nur einzelne Werte aus Steuerelementen
    - Werte als Parameter übergeben
  - Code benötigt tatsächlich ein Formularobjekt
    - Referenz auf das Formular übergeben (Me)

Trennung Funktionalität – GUI: **Möglichst enge Schnittstellen**

- Code-Teile durch möglichst genau definierte (Sprachelemente!) und minimale Schnittstellen miteinander kommunizieren
- Gut geeignet
  - Parameterübergabe
  - Rückgabewerte
  - Ereignisse
- Schlecht wartbar
  - Referenzierungen „aus dem Off"
  - Globale Variablen

Abfragen: gespeichert vs. SQL per Code

Abfragen: gespeichert vs. SQL per Code:
Typisches Szenario einer Abfrage per Code

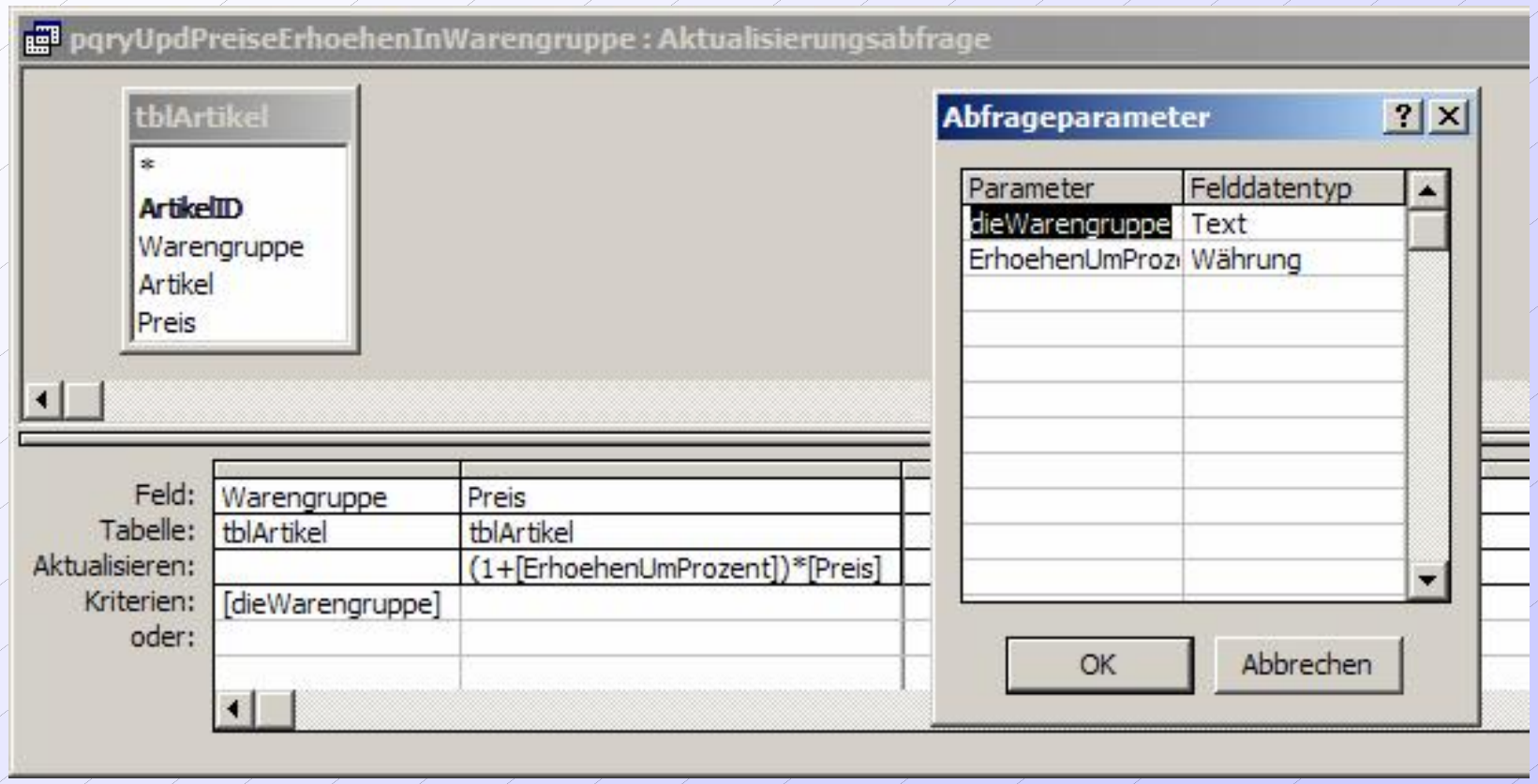
```
Dim strSQL As String
  Dim dbs As DAO.Database
  strSQL = "UPDATE tblArtikel " & _
           "SET Preis = " & Str(1 + curErhoehung) & _
           " * Preis " & _
           "WHERE Warengruppe = " & _
           """" & strWarengruppe & """"
  Set dbs = CurrentDb()
  dbs.Execute strSQL, dbFailOnError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

Abfragen: gespeichert vs. SQL per Code: **Nachteile von SQL per Code**

- Keine interaktive Manipulation (Joins!)
  - Kein interaktiver Test
  - Schwerer verstehbar
  - Schwerer wartbar
  - Abfrageplan ist nicht gespeichert
- Nachteile gespeicherter Abfragen
  - SQL-Code ist (mit Boardmitteln) nicht durchsuchbar
  - Mehr Code zum Ausführen einer Aktionsabfrage

Abfragen: gespeichert vs. SQL per Code:
Verwenden einer gespeicherten Abfrage

- Paremeterabfrage



Abfragen: gespeichert vs. SQL per Code:
Verwenden einer gespeicherten Abfrage

```
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
```

Benennungen

Benennungen:
Konsistente Benennung von Datenbankobjekten

- Alle Typen von Objekten
  - 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
- Tabellen
  - Einzahl oder Mehrzahl (tblKunde oder tblKunden?)

Benennungen:
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()

Uff – Fragen, Bemerkungen?

Nach der Pause

Demo

Zwei Möglichkeiten:

- Such-Formular
- Tabellen-Wiedereinbindung