7 Erweiterungen

7.1 Prozess-Kommunikation mit Datenbanken

Im Buch „Einstieg in das Programmieren mit MATLAB“ wird im Abschnitt 4.8 das Thema Prozess-Kommunikation am Beispiel von MS-Excel behandelt. Als zweites Beispiel wollen wir die Prozess-Kommunikation am Beispiel eines Datenbank-Programms demonstrieren - an Access, das ebenfalls im Microsoft Office-XP-Paket enthalten ist. Vorher aber noch eine knappe Einführung in Datenbanken.

7.1.1 Datenbanken

Eine Datenbank ist eine Sammlung von Daten, die miteinander in Verbindung stehen. Datenbanken kommen in recht unterschiedlichen Anwendungen vor:

- Unter einer Datenbank versteht man normalerweise ein Standard-Datenbank-System, z.B. MS-Access, Oracle oder MySQL. Die Datenstruktur einer solchen Datenbank kann man als Anwender ziemlich frei an alle möglichen Bedingungen anpassen.
- Spezielle Anwendungen, zum Beispiel CAD-Systeme, haben als Kern meist eine proprietäre (selbst entwickelte) Datenbank, deren Struktur von vorne herein durch die Art der Anwendung eingeschränkt ist. Bei CAD-Systemen werden primär Geometrie- und Topologie-Daten und die Zusammenbau-Informationen verwaltet.
- Andere Anwendungen, zum Beispiel PDM-Systeme (Produkt-Daten-Management-Systeme) haben als Basis eine Standard-Datenbank, der eine weitere Struktur (Metadaten) übergestülpt ist. Die Datenstruktur der Datenbank ist bei der Auslieferung fest vorgegeben. Nur die Metadaten können vom Anwender angepasst werden.

Die Hauptbestandteile eines Standard-Datenbank-Systems sind:

- Datenstruktur
- Dateninhalt
- Managementfunktionen (DBMS = Database Management System)
- Abfragesprache als Programmier-Schnittstelle, z.B. SQL (Structured Query Language) und/oder ein Zugriff über eine COM-Schnittstelle.

In einer Datenbank werden die Informationen in Tabellen (Datenstruktur) zusammengefasst, die aus einzelnen Feldern bestehen. Die Spalten einer Tabelle definieren die Attribute dieser Tabelle. Die Attribute legen fest, welche Art von Daten für die Tabelle relevant sind, etwa analog der Komponenten eines MATLAB-structs. Die eigentlichen Daten fol-

gen zeilenweise. Jede Zeile der Tabelle enthält die Daten zu genau einem Objekt. Eine Zeile einer Tabelle wird Tupel genannt.

Das folgende Beispiel soll diese Struktur verdeutlichen. Ich habe eine sehr einfache Tabelle mit Verkäufer-Daten gewählt. Unsere Objektes, die Verkäufer, haben als erstes Attribut eine eindeutige Nummer. Eine Datenbank benötigt solch eindeutige Einträge (Primärschlüssel) zur Identifizierung eines Objektes. Als weitere, nicht notwendig eindeutige, Attribute habe ich nur noch den Namen des Verkäufers und seine Provision eingebaut:

Verkäufer - nummer	Verkäufer - Name	Provision in Prozent
007	Bond	20
123	Mayer	15
222	Otto	15

Abbildung 7.1 Verkäufer-Tabelle

Die drei unteren Zeilen (Tupel) enthalten jeweils eine Objekt, also die Daten zu einem bestimmten Verkäufer.

Eine Firma, die in ihrer Datenbank nur Verkäufer hat, wird es nicht weit bringen. Neben vielen weiteren Tabellen wird es sicher eine Tabelle mit den Kunden-Daten geben. Wie aber stellt man Verknüpfungen (Referenzen) zwischen den einzelnen Tabellen her?

Verkäufer - nummer	Verkäufer - Name	Provision in Prozent
007	Bond	20
123	Mayer	15
222	Otto	15

Kunden - nummer	Verkäufer - nummer	Stadt
0815	007	Hamburg
1234	222	Düsseldorf
2100	123	Frankfurt
9999	007	Lüneburg

Abbildung 7.2 Verknüpfung zwischen Tabellen (Join)

In einem Relationalen Datenmodell werden sowohl die Inhalte als auch die Referenzen mit Hilfe von Tabellen (Relationen) realisiert. Für unsere Datenbank habe ich dazu noch eine zweite Tabelle erzeugt, in der jeder Kunden über eindeutig mit einem Verkäufer verknüpft ist. Der Primärschlüssel der zweiten Tabelle ist die Kundennummer, über die eindeutig weitere Kundendaten abgefragt werden können.

Uns interessiert aber diesmal das zweite Attribut, die Verkäufernummer. Diese Verkäufernummer identifiziert eindeutig einen Verkäufer in der Verkäufer-Tabelle, da dort die Verkäufernummer der Primärschlüssel ist. Diese Art von Verknüpfung nennt man Join.

Noch kurz ein Wort zu den Aufgaben des Database Management Systems. Das DBMS synchronisiert parallele Zugriffe auf den Datenbestand, verwaltet die Zugriffsbedingungen und gewährleistet die Datenintegrität (zum Beispiel über Rollback-Mechanismen).

7.1.2 Adress-Datenbank

Für diesen Abschnitt benötigen wir die Datenbank Access aus dem Microsoft Office-Paket, die zusätzlich zu MATLAB gestartet wird. Um irgendwelche Daten zum Austausch zu haben, folgt jetzt ein Access-Schnellkurs zum Anlegen einer einfachen Adress-Datenbank:

- Wählen Sie in Access-Menü "Datei+Neu". Es erscheint die Auswahl "Neue Datei".
- Wählen Sie den Typ "Leere Datenbank" und geben Sie Ihrer Datenbank einen Namen, zum Beispiel "Adressen.mdb".

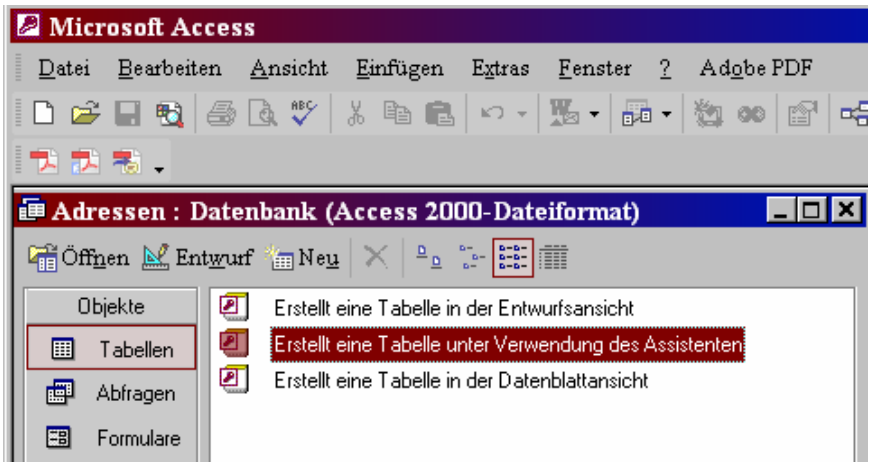

Abbildung 7.3 Leere Datenbank erzeugen

- Wählen Sie die Option "Erstellt eine Tabelle unter Verwendung des Assistenten" und stellen Sie sich die Felder Ihrer Tabelle zusammen, beispielsweise wie in der folgenden

Abbildung. Access bietet einige vordefinierte Felder an, die zusätzlich nach „geschäftlich“ und „privat“ gruppiert sind.

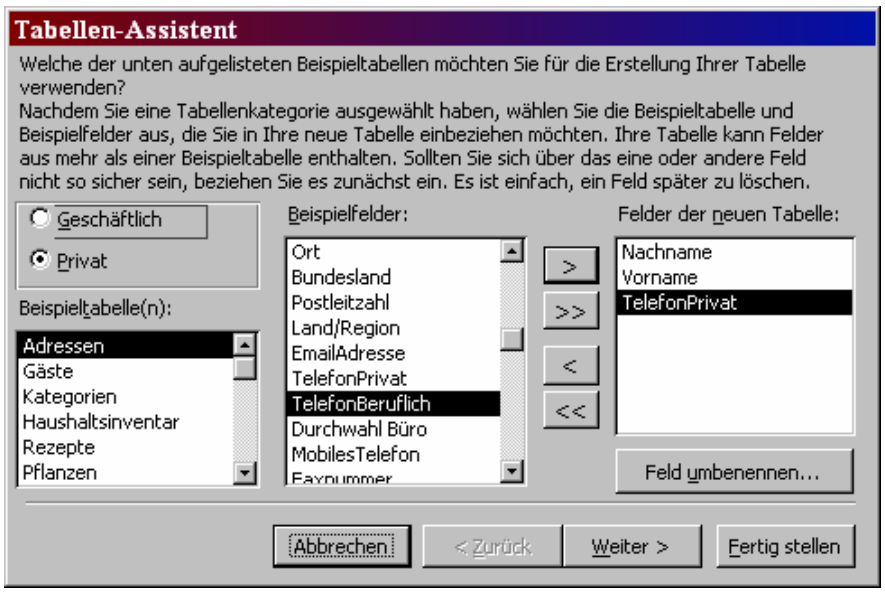

Abbildung 7.4 Tabellenfelder festlegen

- Fahren Sie fort mit "Weiter".

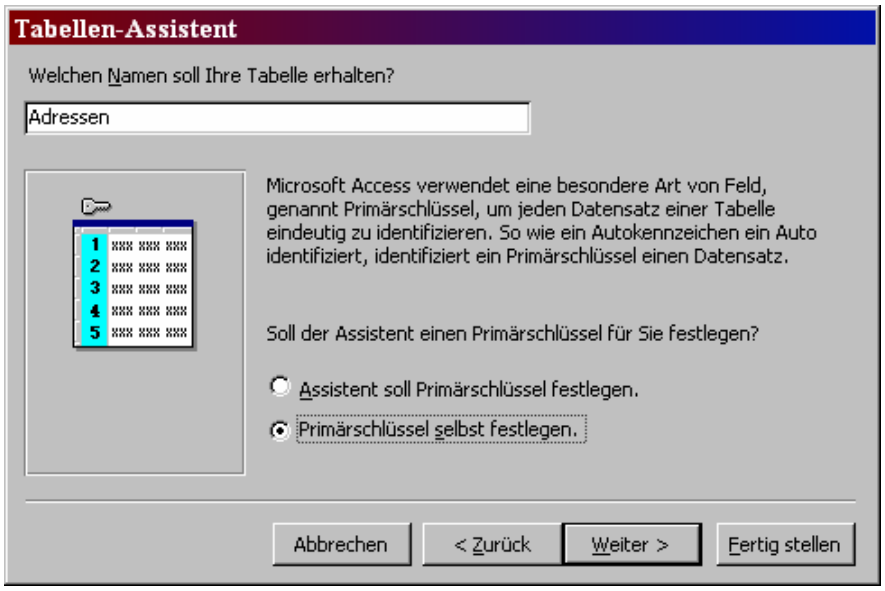

Abbildung 7.5 Tabellennamen und Option Primärschlüssel

- Geben Sie danach Ihrer Datei den Namen "Adressen" und wählen Sie die Option "Primärschlüssel selbst festlegen", bevor Sie "Weiter" drücken:
- Wählen Sie als Primärschlüssel das Feld "Nachname" (das damit immer eindeutige Daten enthalten muss, notfalls also "Meyer Zwo" – in kommerziellen Datenbanken wird als Primärschlüssel deshalb an Stelle des Namens eine eindeutige Personalnummer gewählt, um Überschneidungen zu vermeiden).
- Drücken Sie dann "Weiter":

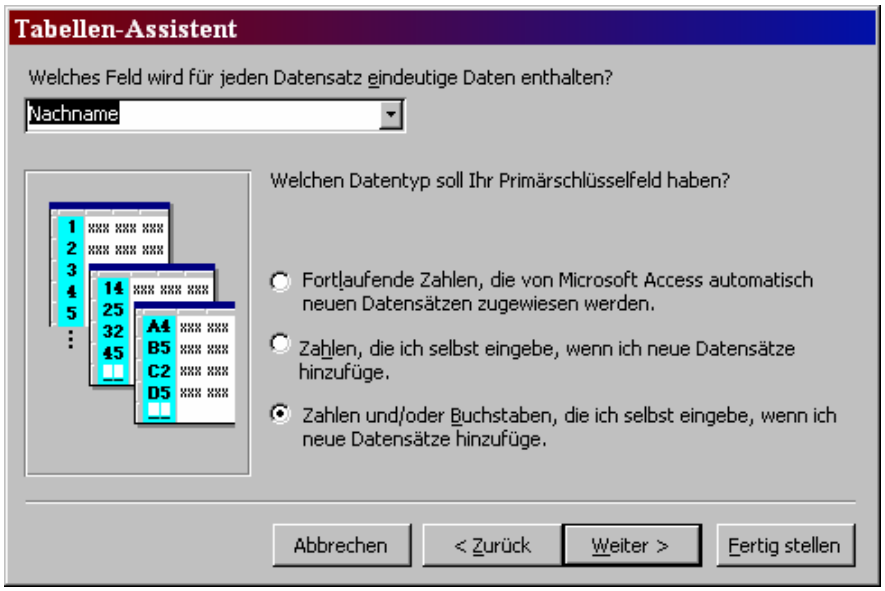

Abbildung 7.6 Primärschlüssel festlegen

- Im nächsten Dialog legen wir fest, dass wir die Daten direkt in die Tabelle eintragen möchten.
- Danach können wir die Tabelle endlich "Fertig stellen".

Abbildung 7.7 Option: Direkt Daten eingeben

Abbildung 7.8 Tabelle für die Adressen-Datenbank

In diese Tabelle tragen wir testweise die Daten einiger Personen ein:

Abbildung 7.9 Tabelle mit Daten-Einträgen

Speichern Sie danach diese Daten ab und beenden Sie Access.

7.1.3 Kontakt zu MS-Access

Zurück zu MATLAB. Wie im Fall von Excel wollen wir versuchen, von MATLAB aus an die Daten eines Microsoft-Access-Prozesses zu kommen.

Der Aufruf von 'actxserver' startet den Access-Prozess:

```
>> h = actxserver ( 'Access.Application' )
   h = COM.Access_Application
```

Über die Eigenschaft 'Visible' machen wir die Oberfläche von Access sichtbar:

```
>> h.Visible = 1;
```

Auf dem Desktop erscheint nun die gestartete Access-Anwendung – aber aktuell noch leer, ohne dass eine Datenbank angezeigt wird.

Zu den Methoden unserer Access-Anwendung gehört unter anderem 'NewCurrentDatabase', mir der sich eine neue Datenbank anlegen lässt:

```
>> h.invoke
    ...
    NewCurrentDatabase = void NewCurrentDatabase(handle, string)
    CurrentDb = handle CurrentDb(handle)
>> h.NewCurrentDatabase( 'TestDb.mdb' )
```

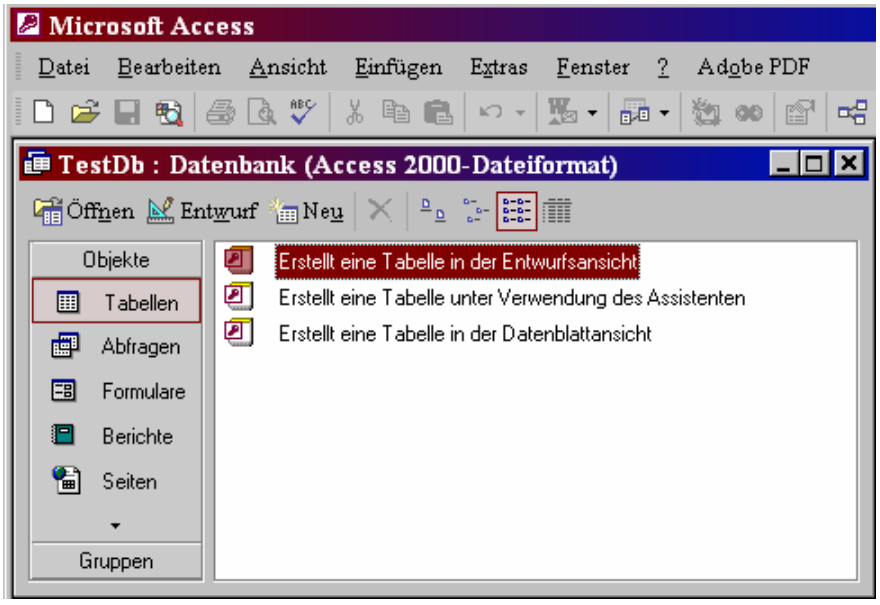

Abbildung 7.10 Neue Datenbank TestDb

Diese Methode gibt keinen Handle auf die neue Datenbank zurück, deshalb müssen wir uns den über die Methode 'CurrentDb' besorgen:

```
>> hDB = h.CurrentDb
   hDB = Interface.Microsoft_DAO_3.51_Object_Library.Database
```

In Access wurde eine neue Datenbank mit dem Namen TestDb angelegt, in der aber noch keine Tabellen angelegt sind.

Die Datenbank besitzt auch die Methode 'CreateTableDef', mit der sich eine Tabellen-Definition erstellen lässt, der man dann Attribute zuordnen kann. Diese Tabellen-Definition kann man danach mit der Datenbank verbinden.

7.1.4 SQL-Aufrufe

Wir wollen jetzt jedoch den direkten Weg zum Anlegen der Tabellen verlassen und eine andere Methode wählen, um mit der Datenbank zu kommunizieren. Es gibt für Datenbank-Operationen nämlich die standardisierte Skript-Sprache SQL, die nicht nur auf die Microsoft-Datenbank Access beschränkt ist. Und die COM-Schnittstelle von Access erlaubt es auch, (einige, aber leider nicht alle) SQL-Befehle abzusetzen.

Der Aufruf erfolgt über das Interface 'DoCmd':

```
>> h.get
   ...
   DoCmd: [1x1 Interface.Microsoft_Access_10.0_Object_Library.DoCmd]
>> DoCmd = h.DoCmd
   DoCmd = Interface.Microsoft_Access_10.0_Object_Library.DoCmd
```

'DoCmd' selbst hat die Methode 'RunSQL' zum Aufruf von SQL-Befehlen:

```
>> DoCmd.invoke
    ...
    RunSQL = void RunSQL(handle, Variant, Variant(Optional))
```

Mit diesem Befehl können wir eine neue Tabelle anlegen. Der SQL-Befehl dazu lautet

'CREATE TABLE <Name der Tabelle> (Liste der Attribute mit Attribut-Typ)'.

Unsere Tabelle soll den Namen 'Privat' tragen und die beiden Attribute 'Name' und 'Vorname' enthalten. Beides seien Text-Attribute mit der Länge von maximal 20 Zeichen:

Der zugehörige SQL-Befehl lautet:

```
CREATE TABLE Privat (Name text (20), Vorname text (20) );
```

In MATLAB definieren wir zuerst den Befehls-String:

```
>> SQLCmd = ...
      'CREATE TABLE Privat (Name text (20), Vorname text (20) );';
```

und übergeben diesen Text dann an den Befehl 'RunSQL':

```
>> DoCmd.RunSQL( SQLCmd );
```

In Access ist diese Tabelle aber noch nicht zu sehen, da dort die Ansicht nicht aktualisiert wurde. Klicken Sie in Access einmal auf einen anderen Button unter "Objekte", zum Beispiel auf "Abfragen", und wählen Sie danach wieder die Ansicht "Tabellen". Jetzt erscheint auch unsere Tabelle 'Privat' am Ende der Liste. Ein Doppel-Klick auf diesen Eintrag öffnet die neue Tabelle:

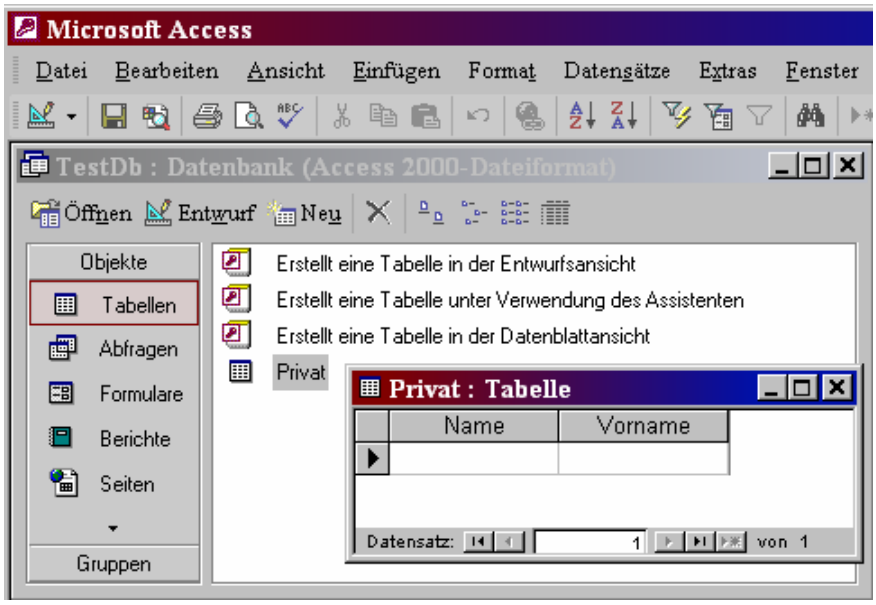

Abbildung 7.11 Tabelle, über SQL-Kommando erzeugt

Jetzt wollen wir noch eine Person in die Tabelle eintragen:

```
>> SQLCmd = ...
     'INSERT INTO Privat(Name,Vorname) VALUES ("Bond","James");';
>> DoCmd.RunSQL( SQLCmd  );
```

Zum Aktualisieren der Daten müssen wir die Tabelle 'Privat' einmal schließen und mit einem Doppelklick wieder öffnen:

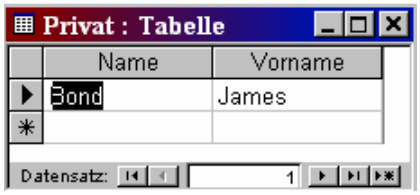

Abbildung 7.12 Tabellen-Einträge, über SQL-Kommando

Jetzt schließen wir die Datenbank, beenden Access und melden den COM-Server ab:

```
>> h.CloseCurrentDatabase
>> h.Quit
>> h.delete
```

7.1.5 VBA-Kontakt zu Access

Als Vergleich soll nun gezeigt werden, wie man die Access-Abfragen des letzten Abschnittes mit Hilfe von VBA (Visual Basic for Applications) realisiert. Wir verwenden wieder den in Word-2002 integrierten Makro-Editor und gehen genauso vor wie im Excel-Beispiel.

Da wir von Word aus gestartet sind, müssen wir VBA noch ein wenig schlau machen in Bezug auf Access. Gehen Sie im VBA-Editor im Menü "Extras" unter "Verweise". Es öffnet sich die Liste mit allen auf dem Rechner verfügbaren COM-Bibliotheken. Markieren Sie zusätzlich "Microsoft Access 10.0 Object Library" (hier Version 10.0 für Office-XP), wie wir es auch im Fall von Excel getan haben. Unter dem VBA-Menü-Eintrag "Ansicht+Objektkatalog" können Sie sich jetzt die verfügbaren COM-Methoden und Interfaces der einzelnen Bibliotheken anzeigen lassen.

In den Callback des Command-Buttons schreiben wir diesmal:

```
Private Sub CommandButton1_Click()
  Set h = CreateObject("Access.Application")
  h.Visible = 1
  h.NewCurrentDatabase ("TestDb.mdb")
  Set hDB = h.CurrentDb
  SQLCmd =
       "CREATE TABLE Privat (Name text (20), Vorname text (20) );"
  DoCmd.RunSQL (SQLCmd)
  SQLCmd =
       "INSERT INTO Privat(Name,Vorname) VALUES ('Bond','James');"
  DoCmd.RunSQL (SQLCmd)
```

```
    h.CloseCurrentDatabase
    h.Quit
    Set h = Nothing
End Sub
```

Wie im Fall von Excel können Sie jetzt mit der Funktionstaste F8 den Programm-Ablauf Zeile für Zeile im Debugger durchlaufen. Sie sehen - die COM-Operationen werden unter VBA genauso wie unter MATLAB ausgeführt und die entsprechenden Code-Zeilen sehen sehr ähnlich aus.