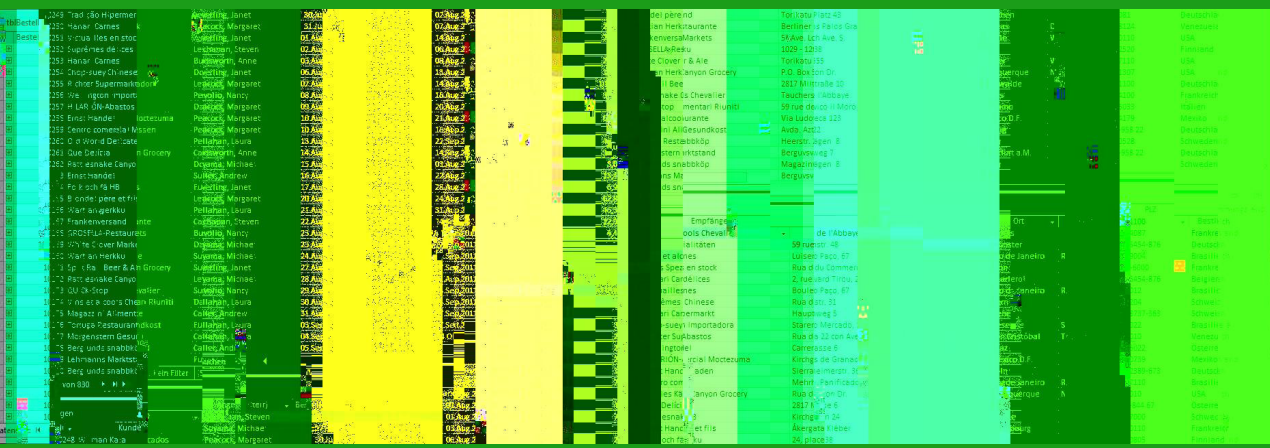
ACCESS FORMULARE



FORMULARE UND STEUERELEMENTE ERSTELLEN UND PROGRAMMIEREN

Leseprobe!

Bestellung unter www.amvshop.de

ANDRÉ MINHORST

André Minhorst Verlag

André Minhorst

Access-Formulare

Formulare und Steuerelemente erstellen und programmieren

André Minhorst – Access-Formulare

ISBN 978-3-944216-06-5

1. Auflage 2019

Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliographie. Detaillierte bibliografische Daten finden Sie im Internet unter http://dnb.d-nb.de.

Context.

Inhalt

1 Schnelleinstieg

Damit wir ein Beispiel haben, um die folgenden Begriffe zu erläutern, erstellen wir schnell ein erstes Formular. Wir gehen dabei davon aus, dass Sie eine Datenbank-Anwendung erstellt und eine Tabelle hinzugefügt haben – beispielsweise eine Tabelle namens *tblKunden* mit den Daten einiger Kunden. Der Haupteinsatzzweck von Access-Formularen ist nämlich, die Daten aus Tabellen oder Abfragen anzuzeigen und zur Bearbeitung bereitzustellen.

1.1 Formular erstellen

Um die Daten einer solchen Tabelle in einem Formular anzuzeigen, sind nur wenige Schritte nötig: Sie erstellen ein neues Formular in der Entwurfsansicht, indem Sie den Ribbon-Eintrag *Erstellen|Formular|Formularentwurf* betätigen (siehe Abbildung 1.1).

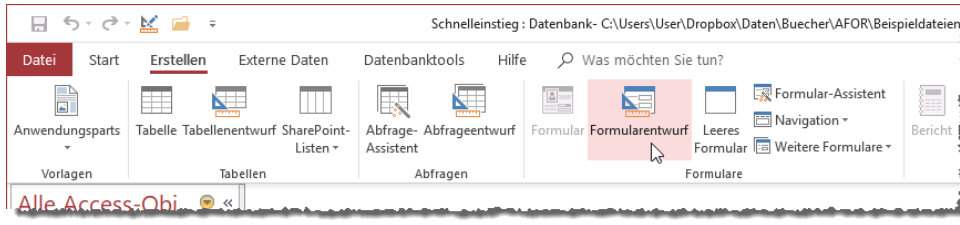


Abbildung 1.1: Anlegen eines neuen Formulars in der Entwurfsansicht

Sie finden dann ein neues, leeres und ungespeichertes Formular in der Registeransicht von Access vor (siehe Abbildung 1.2). Hier sehen Sie auch gleich die wichtigsten Elemente zum Bearbeiten von Formularen: Oben befindet sich das Ribbon mit den Befehlen etwa zum Anlegen neuer Formulare, zum Hinzufügen von Steuerelementen oder zum Anordnen oder Formatieren der im Formular enthaltenen Steuerelemente.

Auf der linken Seite finden Sie den Navigationsbereich, der alle Elemente der Datenbank anzeigt – Tabellen, Abfragen, Formulare, Berichte, Makros und VBA-Module. Der Arbeitsbereich zeigt bei den standardmäßigen Optionen das neue Formular in der Registerkarten-Ansicht an. Hier finden Sie normalerweise den Detailbereich des Formulars vor. Es gibt noch andere Bereiche, die wir weiter unten vorstellen. Auf der rechten Seite finden Sie das Eigenschaftenblatt, das Sie allerdings auch vom rechten Rand ablösen und frei platzieren können. Das Eigenschaftenblatt zeigt die Eigenschaften des aktuell markierten Elements auf fünf Registerkarten verteilt an, von denen die ersten vier verschiedenen Kategorien wie *Format*, *Daten*, *Ereignis* und *Andere* entsprechen und die fünfte die Eigenschaften aller vier vorherigen Registerseiten zusammenfasst.

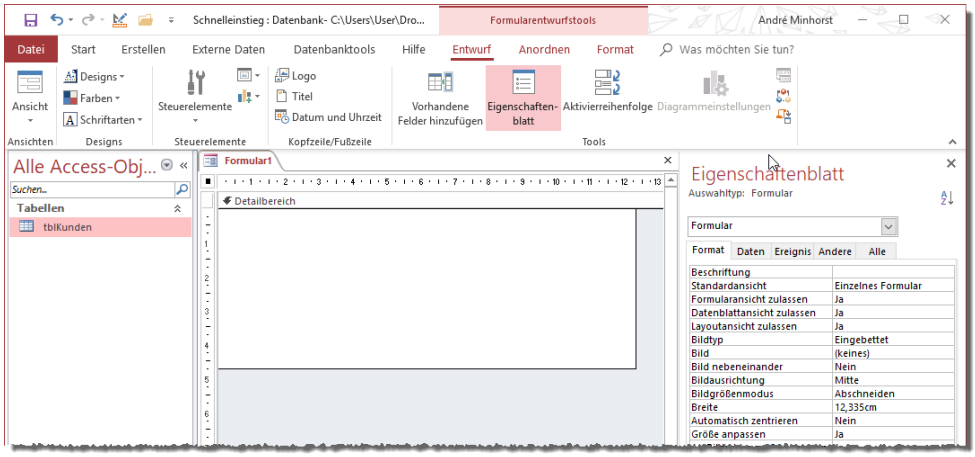


Abbildung 1.2: Ein neues, leeres Formular

1.1.1 Formular an Datensatzquelle binden

Das Formular können wir nun an eine Tabelle der Datenbank binden. Dazu stellen wir die Eigenschaft *Datensatzquelle* auf den Namen der Tabelle ein, an die wir das Formular binden wollen – in diesem Fall an die einzige Tabelle namens *tblKunden*. Das Eigenschaftsfenster sieht dann auf der Seite *Daten* wie in Abbildung 1.3 aus.

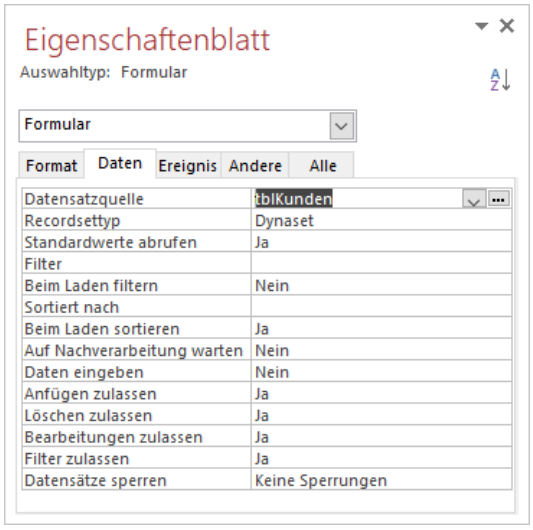


Abbildung 1.3: Einstellen der Datensatzquelle für das Formular auf die Tabelle *tblKunden*

1.1.2 Wechsel in die Formularansicht

Was ist nun geschehen? Äußerlich nicht viel. Wenn Sie jedoch nun in die Formularansicht wechseln, was Sie mit dem Ribbon-Befehl *Start|Ansichten|Ansicht* erledigen, erhalten Sie die Ansicht aus Abbildung 1.4.

Das Formular scheint leer zu sein, aber wenn Sie einen Blick nach unten auf die Navigationsleiste werfen, erkennen Sie, dass das Formular offensichtlich 101 Datensätze enthält. Sie können hier auch durch die Datensätze navigieren, aber am dargestellten Inhalt ändert sich nichts – das Formular bleibt leer.

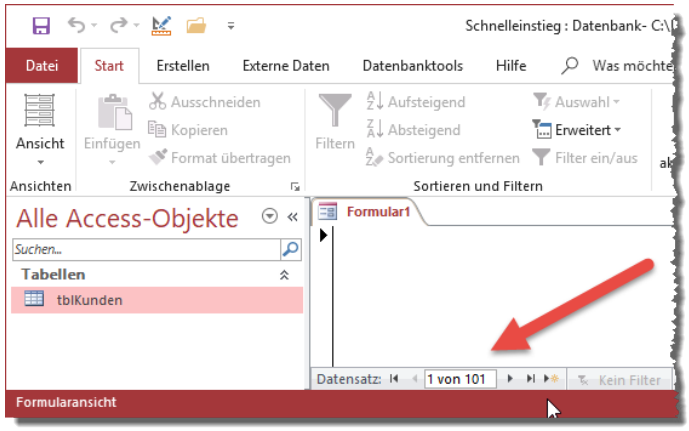


Abbildung 1.4: Das Formular zeigt 101 Datensätze an.

1.1.3 Felder hinzufügen

Wechseln Sie nun zurück in die Entwurfsansicht (Ribbon-Eintrag *Start|Ansichten|Ansicht|Entwurfsansicht*). Klicken Sie dann im Ribbon auf *Entwurf|Tools|Vorhandene Felder hinzufügen*, erscheint statt des Eigenschaftsblattes die Feldliste. Diese zeigt alle Felder der als Datensatzquelle angegebenen Tabelle oder Abfrage.

Und nun kommt ein wenig der Magie, die Access zu einem solch schnellen Tool für die Erstellung von Datenbankanwendungen werden lässt: Markieren Sie mit der Maus alle Einträge der Feldliste, indem Sie den obersten Eintrag markieren und dann bei gedrückter Umschalt-Taste den untersten. Ziehen Sie dann die markierten Einträge in den Detailbereich der Entwurfsansicht des Formulars (siehe Abbildung 1.5).

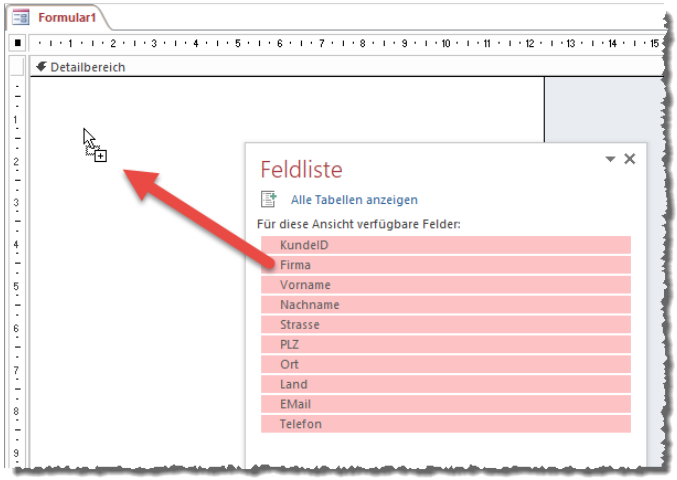


Abbildung 1.5: Hinzufügen von Einträgen aus der Feldliste per Drag and Drop

Lassen Sie dann die Einträge fallen, aber nicht zu weit links – der linke obere Eintrag wird an der Spitze des Mauszeigers platziert, aber links daneben muss noch Platz für die Beschriftungsfelder sein. Das Ergebnis sieht dann wie in Abbildung 1.6 aus. Gar nicht schlecht: Sie haben soeben Ihr erstes Formular erschaffen.

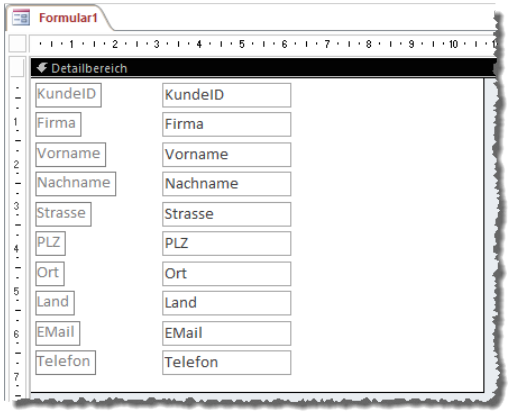


Abbildung 1.6: Die per Drag and Drop hinzugefügten Felder

1.1.4 Formular speichern

Dieses sollten Sie nun noch speichern, damit es nicht verloren geht. Das erledigen Sie am einfachsten, indem Sie die Tastenkombination *Strg + S* betätigen. Access fragt Sie dann in einem

Speichern unter-Dialog nach dem Namen, unter dem das Formular gespeichert werden soll. Geben Sie hier zunächst *frmKunden* ein und betätigen Sie die Eingabetaste.

Das Formular wird nun gespeichert und es erscheint ein Eintrag dafür im Navigationsbereich links im Access-Fenster (siehe Abbildung 1.7).

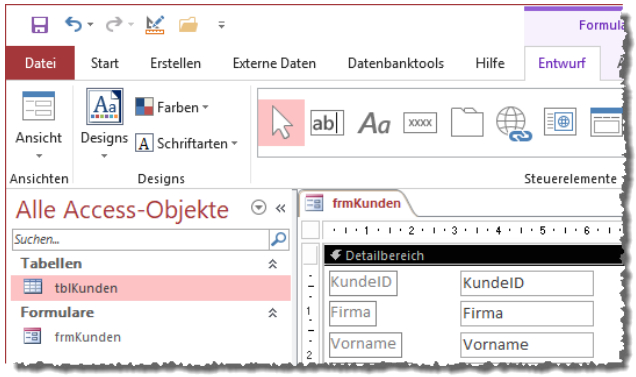


Abbildung 1.7: Das neue Formular im Navigationsbereich

1.1.5 Daten anzeigen

Nun ist es an der Zeit, das Formular in Aktion zu betrachten. Wechseln Sie erneut wie oben beschrieben in die Formularansicht. Nun sieht das Formular schon viel besser aus (siehe Abbildung 1.8). Zwar passen nicht alle Inhalte in die dafür vorgesehenen Felder, aber das ist kein Problem – die Breite der Textfelder können wir ja später noch anpassen.

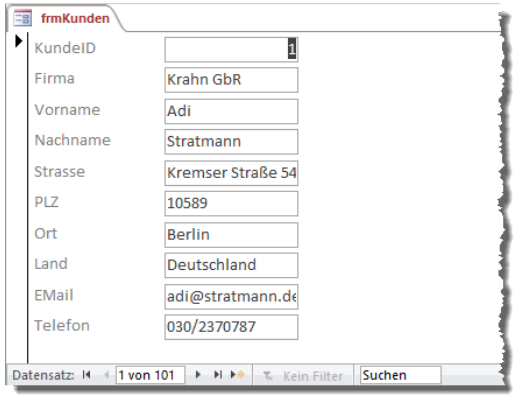


Abbildung 1.8: Das Formular in der Formularansicht

Wichtig ist, dass die Daten des ersten Datensatzes der Tabelle angezeigt werden und bearbeitet werden können. Außerdem können Sie nun, in dem Sie die Navigationsschaltflächen betätigen, zwischen den Datensätzen der Datensatzquelle des Formulars wechseln. Sie können die Inhalte ändern – mit Ausnahme des Feldes *KundeID*, da dieses durch die Autowert-Funktion gefüllt wird. Und Sie können die Änderungen des Datensatzes speichern, indem Sie einfach zum nächsten Datensatz wechseln.

1.1.6 Die Datenblattansicht

Wir zeigen nun noch eine weitere Formularansicht neben den beiden, die Sie bisher kennengelernt haben, also der *Entwurfsansicht* und der Ansicht namens *Formularansicht*. Dabei handelt es sich um die Datenblattansicht. Diese zeigen Sie an, indem Sie einfach den Ribbon-Befehl *Start|Ansichten|Ansicht|Datenblattansicht* auswählen. Das Ergebnis sehen Sie in Abbildung 1.9. Das Formular sieht nun genauso aus wie die Datenblattansicht der als Datensatzquelle verwendeten Tabelle. Das mag auf den ersten Blick unnötig sein, da Sie diese Ansicht ja auch mit der Tabelle erhalten hätten. Allerdings werden Formulare in der Datenblattansicht auch in den meisten Fällen nicht allein verwendet, sondern dienen dann als Formular in einem Unterformular-Steuerelement in einem anderen Formular. Das übergeordnete Formular zeigt dann beispielsweise die die Daten einer Bestellung an und das Unterformular in der Datenblattansicht die der Bestellung zugeordneten Bestellpositionen.

Was wir nur zeigen wollten, ist die Möglichkeit, mit einem Formular gleich zwei verschiedene Ansichten zu liefern. Und um es vorwegzunehmen: Es gibt mit der Endlosansicht noch eine dritte Formularansicht. Um die kümmern wir uns jedoch erst später.


frmKunden

KundeID	Firma	Vorname	Nachname	Strasse	PLZ	Ort	Land	EMail	Telefon
1	Krahn GbR	Adi	Stratmann	Kremser Straß	10589	Berlin	Deutschland	adi@stratmani	030/2370787
2	Göllner GmbH	Heidi	Eich	Moosstraße 42	42289	Wuppertal	Deutschland	heidi@eich.de	0202/7398093
3	Peukert AG	Wernfried	Birk	Wiener Straße	22297	Hamburg	Deutschland	wernfried@bi	040/542662
4	Bruder GmbH	Vitus	Krauße	Burgenlandstr	20355	Hamburg	Deutschland	vitus@krauße.	040/5992692
5	Mader AG	Jadwiga	Oehme	Peter-Rosegge	65187	Wiesbaden	Deutschland	jadwiga@oehr	0611/6051172
6	Fleischhauer G	Niko	Michel	Kindergartens	12051	Berlin	Deutschland	niko@michel.(	030/399421
7	Bolte GmbH	Siegert	Loos	Lenaustraße 6	66115	Saarbrücken	Deutschland	siegert@loos.(	0681/473621
8	Nitzsche AG	Michl	Schroth	Dr. Karl Renne	01129	Dresden	Deutschland	michl@schrotl	0351/927329
9	Ziemann GbR	Florentius	Wittek	Industriestraß	80638	München	Deutschland	florentius@wi	089/056980
10	Krahl AG	Gernulf	Riegel	Erzherzog-Joh	81545	München	Deutschland	gernulf@riege	089/655434
11	Becker GmbH	Tristan	Hübsch	Jahnstraße 7	65193	Wiesbaden	Deutschland	tristan@hübsc	0611/903341
12	Runge AG	Heinfried	Steinert	Kaplanstraße 2	30159	Hannover	Deutschland	heinfried@ste	0511/5196531
13	Albert GbR	Herma	Aigner	Burgstraße 47	22089	Hamburg	Deutschland	herma@aigne	040/5802013

Datensatz: 1 von 101 | Kein Filter | Suchen

Abbildung 1.9: Ein Formular in der Datenblattansicht

Mit diesem Beispiel haben Sie einen kleinen Eindruck gewonnen, was Sie mit Formularen anstellen können und wie die wichtigsten Begriffe heißen.

1.2 Warum Formulare?

Formulare sind das Salz in der Suppe einer Access-Anwendung. Während Sie die Daten in Tabellen speichern, die typischerweise über Beziehungen miteinander verknüpft sind, und diese über Abfragen filtern, sortieren und aus mehreren Tabellen zusammenstellen können, bieten Formulare die Möglichkeit, diese Daten übersichtlich und strukturiert darzustellen und dem Benutzer Steuerelemente zur Bearbeitung der Daten und zur Steuerung der Abläufe bereitstellen.

Ein Formular kann die Daten Ihrer Tabellen in der Formularansicht, der Datenblattansicht oder der Endlosansicht anzeigen. Diese drei Ansichten werden im Anschluss erläutert.

Sie können aber auch die Daten mehrerer Tabellen in einem Formular zusammenführen, und zwar durch den Einsatz eines Unterformulars. In diesem Fall zeigen Sie beispielsweise in einem Hauptformular die Datensätze einer Bestellung an, also den ausgewählten Kunden, das Bestelldatum und so weiter und in einem untergeordneten Formular in der Datenblattansicht alle mit dieser Bestellung verknüpften Bestellpositionen.

Neben der Anzeige der Daten und der Bereitstellung von verschiedenen Steuerelementen ist eine der großen Stärken der Formulare gegenüber den Tabellen oder Abfragen die Möglichkeit, für verschiedene Ereignisse VBA-Code zu hinterlegen, mit dem Sie verschiedene weitere Aktionen anstoßen können. Damit können Sie etwa auf das Anklicken einer Schaltfläche reagieren, die einen neuen Datensatz anlegen, einen Datensatz löschen oder auch das Formular schließen soll. Oder Sie reagieren auf andere Ereignisse wie etwa das Anzeigen eines weiteren Datensatzes beim Blättern durch die Datensätze oder auf das Ändern oder Speichern von Inhalten des aktuellen Datensatzes. Dafür bietet Access eine reichhaltige Auswahl an Ereigniseigenschaften an, für die Sie sogenannte Ereignisprozeduren hinterlegen können.

Formulare bieten aber auch zahlreiche weitere Vorteile gegenüber der einfachen Darstellung in einem Datenblatt:

» Sie können damit Bilder und andere grafische Elemente in der Benutzeroberfläche anzeigen.

» Sie können die Größe von Steuerelementen, Beschriftungen und deren Schriftarten nach ihren Wünschen gestalten.

» Mit Formularen lassen sich die Daten der Tabellen in der Art von Papierformularen abbilden, sodass die Bearbeitung intuitiver wird.

» Sie können den Benutzer durch verschiedene Eingabehilfen wie Hinweistexte unterstützen. Oder Sie setzen neben den Textfeldern spezielle Steuerelemente wie Kontrollkästchen, Kombinationsfelder, Listenfelder, Optionsgruppen und viele mehr ein, um die Eingabe der Daten so einfach wie möglich zu gestalten.

1.3 Die Formularansichten

Für ein Formular können Sie über die Eigenschaft *Standardansicht* eine der drei Ansichten einstellen:

» Formularansicht

» Datenblattansicht

» Endlosansicht

Die folgenden Abschnitte erläutern die Unterschiede zwischen den Ansichten.

1.3.1 Die Formularansicht

Die Formularansicht kommt in folgenden Fällen zum Einsatz:

» Wenn Sie ein ungebundenes Formular wie etwas das Start-Formular einer Anwendung oder eine Übersicht verschiedener Befehle ohne Anzeige der eigentlichen Daten anzeigen wollen.

» Wenn Sie die Daten eines einzigen Datensatzes präsentieren wollen, also etwa eine Bestellung, einen Kunden oder einen Artikel.

» Ein ähnlich aufgebautes Formular können Sie auch nutzen, um einen neuen Datensatz anzulegen und zu speichern. Oft verwendet man das gleiche Formular zum Anlegen neuer Datensätze oder zum Anzeigen/Bearbeiten vorhandener Datensätze.

Weiter oben haben Sie bereits ein Formular in der Formularansicht zur Anzeige der Daten eines einzelnen Kunden kennengelernt.

1.3.2 Die Datenblattansicht

Die Datenblattansicht ist eine von zwei Ansichten, die mehrere Datensätze gleichzeitig liefert – die andere ist die nachfolgend beschriebene Endlosansicht. Die Datenblattansicht eines Formulars ist prinzipiell mit der Datenblattansicht einer Tabelle oder Abfrage identisch. Sie bietet die gleichen Features, aber dadurch, dass es sich bei dieser Ansicht um eine Formularansicht handelt, können Sie diese mit VBA-Code um Funktionen erweitern und diese Ansicht noch viel mächtiger machen.

Zu den bereits vorhandenen Funktionen wie denen zum Sortieren oder Filtern der Daten oder zum Einstellen der Spaltenbreiten oder der Spaltenreihenfolge kommen dann noch weitere, wobei Ihrer Fantasie kaum Grenzen gesetzt sind. Sie können so beispielsweise automatisch dafür sorgen, dass die Spalten immer in der optimalen Breite angezeigt werden.

In der Datenblattansicht können Sie die folgenden Steuerelementtypen verwenden:

» Textfeld

» Kombinationsfeld

» Kontrollkästchen

Auch die Datenblattansicht haben wir uns weiter oben bereits angesehen, um die Liste der verfügbaren Datensätze der Tabelle *tblArtikel* auszugeben.

1.3.3 Die Endlosansicht

Die Endlosansicht bietet nicht so viele eingebaute Features wie die Datenblattansicht – Sie müssen hier etwa auf die Spaltenköpfe der Datenblattansicht verzichten, mit der Sie beispielsweise schnell Datensätze sortieren und filtern oder die Spalten vertauschen, ein- oder ausblenden könnten.

In der Endlosansicht haben Sie dafür eine wesentlich höhere Flexibilität bei der Gestaltung des Detailbereichs. Wenn Sie ein Formular in der Endlosansicht nutzen wollen, bietet es sich an, den Formularkopf-Bereich, den Sie weiter unten kennenlernen, einzublenden und dort die Spaltenüberschriften unterzubringen.

Im Detailbereich legen Sie dann nebeneinander die Steuerelemente zur Anzeige der eigentlichen Daten an. Im Entwurf sieht das für ein schnell zusammengeklicktes Beispielformular etwa wie in Abbildung 1.10 aus (wie das gelingt, zeigen wir später).

Hier wird deutlich, dass wir sowohl die Spaltenüberschriften als auch die Steuerelemente für eigentlichen Daten nur einmal anlegen. Auch wenn später mehrere Datensätze untereinander angezeigt werden sollen, brauchen Sie das Aussehen dieses Bereichs, des Detailbereichs, nur einmalig zu definieren.

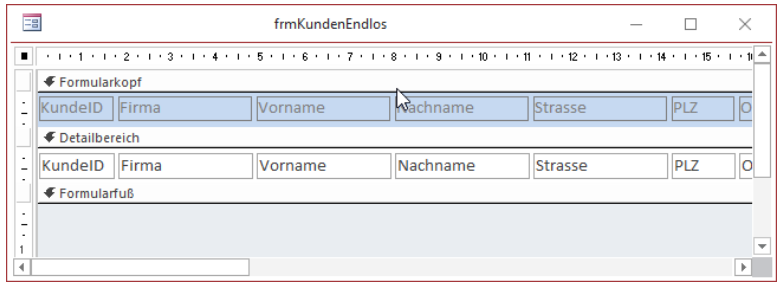


Abbildung 1.10: Entwurf eines Formulars in der Endlosansicht

Wenn Sie nun die Eigenschaft *Standardansicht* auf *Endlosformular* eingestellt haben und in die Formularansicht des Formulars wechseln, sieht das Ergebnis etwa wie in Abbildung 1.11 aus.

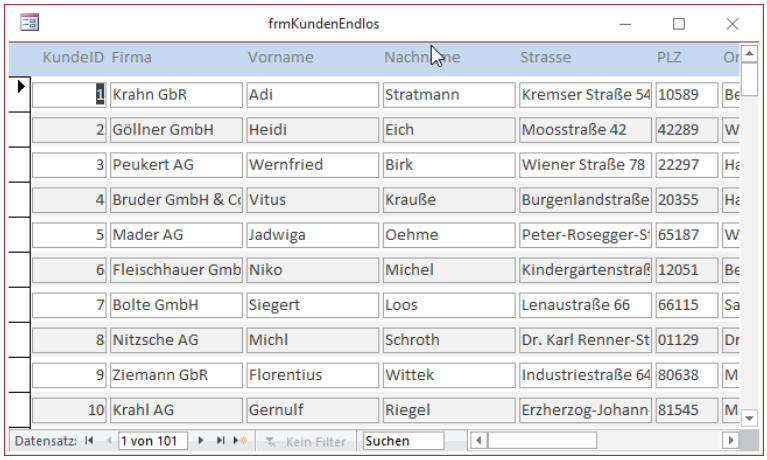


Abbildung 1.11: Einfaches Beispiel für die Endlosansicht

Sie sehen hier, dass die Darstellung ein Hybrid aus der Datenblattansicht und der Formularansicht ist. Es erscheinen alle Datensätze untereinander wie in der Datenblattansicht, aber Sie können das Layout individueller definieren in der Datenblattansicht – mit allen Möglichkeiten, welche die Formularansicht bietet.

1.4 Die Formularbereiche

In Formularen gibt es die folgenden Bereiche:

» Formularkopf

» Detailbereich

» Formularfuß

Welche dieser Bereiche eingeblendet sind oder nicht, stellen Sie beispielsweise über das Kontextmenü eines der Köpfe der bereits eingeblendeten Formularebereiche ein. Hier finden Sie die beiden Einträge *Seitenkopf-/fuß* und *Formularkopf/-fuß* (siehe Abbildung 1.12).

1.4.1 Der Formularkopf

Dieser Bereich wird maximal einmal pro Formular angezeigt, und zwar im oberen Bereich des Formulars. Hier können Sie beispielsweise einen Kopfbereich definieren, der eine Überschrift für das Formular liefert und wichtige Informationen zum Formular anzeigt. Ein Beispiel dazu haben Sie bereits im Endlosformular gesehen, das wir weiter oben vorgestellt haben.

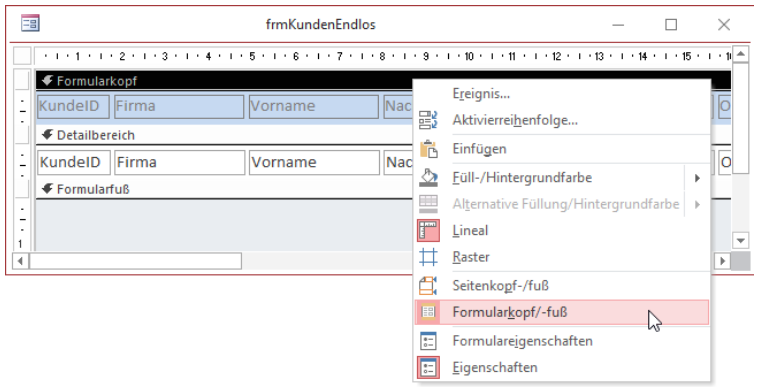


Abbildung 1.12: Ein- oder Ausblenden von Formularkopf und -fuß

1.4.2 Der Detailbereich

Der Detailbereich eines Formulars wird ein- oder mehrmals angezeigt. Die Anzahl wird durch die Anzahl der Datensätze in der Datenherkunft des Formulars beeinflusst. Wenn Sie das Formular in der Formularansicht öffnen, wird der Detailbereich wie der Kopf- und der Fußbereich nur einfach angezeigt und nicht für jeden Datensatz einmal. Dafür können Sie mit den Navigationssteuerelementen am Fuß des Formulars durch die enthaltenen Datensätze navigieren.

1.4.3 Der Formularfuß

Der Formularfuß wird wie der Formularkopf nur maximal einmal pro Formular angezeigt. Hier können Sie beispielsweise standardmäßig in Formularen enthaltene Steuerelemente wie etwa die Schaltflächen *OK* und *Abbrechen* unterbringen.

1.4.4 Seitenkopf und -fuß

Es gibt sogar noch einen Seitenkopf und einen Seitenfuß. Diese beiden Bereiche benötigen Sie aber nur theoretisch, und zwar dann, wenn Sie ein Formular ausdrucken wollen. Das ist zwar möglich, aber in der Regel werden Sie zum Ausdrucken von Daten eher auf Berichte zugreifen. Diese sind jedoch nicht Thema dieses Buchs.

2 Formular-Basics

Es gibt viele Informationen über Formulare, die ihnen die spätere Erstellung der Formulare sowie deren Programmierung wesentlich erleichtern können. Diese haben wir im vorliegenden Kapitel zusammengestellt, bevor es im nächsten Kapitel endlich daran geht, Daten im Formular anzuzeigen. Wir empfehlen bei aller Ungeduld jedoch, sich das vorliegende Kapitel in aller Ruhe zu Gemüte zu führen.

2.1 Konventionen

Für die Benennung von Access-Objekten wir Tabellen, Abfragen, Formularen und so weiter und für Steuerelemente gibt es bestimmte Benennungskonventionen. Auch für die Programmierung von Formularen per VBA gibt es diese.

Hier sind die wichtigsten Benennungsregeln, die Sie bei Ihrer Arbeit mit Access-Formularen berücksichtigen sollten und die wir auch in diesem Buch einhalten.

2.1.1 Konventionen für Access-Objekte

Access-Objekte versehen wir mit den folgenden Präfixen:

» Tabelle: *tbl* (zum Beispiel *tblKunden*)

» Abfragen: *qry* (zum Beispiel *qryKundenMitBestellungen*)

» Formulare: *frm* (zum Beispiel *frmKundenUebersicht*)

» Berichte: *rpt* (zum Beispiel *rptBestellungen*)

» Standardmodule: *mdl* (zum Beispiel *mdlTools*)

» Klassenmodule: *cls* (zum Beispiel *clsKunde*)

2.1.2 Konventionen für Steuerelemente

Auch Steuerelemente wollen wir mit bestimmten Präfixen versehen:

» Textfeld: *txt* (von *Textbox*)

» Bezeichnungsfeld: *lbl* (von *Label*)

» Schaltfläche: *cmd* (von *CommandButton*)

» Umschaltfläche: *tgl* (von *ToggleButton*)

- Kombinationsfeld: *cbo* (von *ComboBox*)
- Listenfeld: *lst* (von *ListBox*)
- Kontrollkästchen: *chk* (von *CheckBox*)
- Optionsgruppe: *ogr* (von *OptionGroup*)
- Optionsfeld: *opt* (von *OptionButton*)
- Registersteuerelement *tab* (von *Tab*)
- Unterformular: *sfm* (von *SubForm*)
- Webbrowsersteuerelement: *web* (von *Webbrowser*)
- Link-Steuerelement: *lbl* (da basierend auf *Label*-Steuerelement)
- Navigationssteuerelement: *nav*
- Linie-Steuerelement: *lin* (von *Line*)
- Rechteck-Steuerelement: *rct* (von *Rectangle*)
- Bild-Steuerelement: *pic* (von Picture)
- Anlage-Steuerelement: *att* (von *Attachment*)
- Gebundenes Objektfeld: *frb* (von *Bound Object Frame*)
- Ungebundenes Objektfeld: *fru* (von *Unbound Object Frame*)

Steuerelemente aus der Feldliste umbenennen

Weiter oben haben Sie bereits erfahren, dass Sie die Felder der Tabelle, an die das Formular gebunden ist, aus der Feldliste in den Formularentwurf ziehen können und damit eine Menge Zeit gegenüber dem manuellen Hinzufügen von Textfeldern, Kombinationsfeldern und Co. sparen, wo Sie dann immer noch die Eigenschaft *Steuerelementinhalt* auf den jeweiligen Feldnamen einstellen müssten.

Wenn Sie Steuerelemente anlegen, sollte Sie diese sofort mit dem entsprechenden Präfix ausstatten. Bei den gebundenen Steuerelementen aus der Feldliste vergisst man das gern mal.

Dann heißen die Steuerelemente und die an das Formular gebundenen Felder genau gleich, zum Beispiel *KategorieID* oder *Kategorie*. Da es manche Eigenschaften für Steuerelemente gibt, die ein Feld nicht anbietet, ist es aber dennoch sinnvoll, die gebundenen Steuerelemente mit den passenden Präfixen zu versehen – also etwa *txt* für Textfelder, *cbo* für Kombinationsfelder oder *chk* für Kontrollkästchen. Das ist nur leider eine langweilige Aufgabe, die wir lieber einer kleinen VBA-Prozedur überlassen wollen. Diese finden Sie weiter hinten unter »Gebundene Steuerelemente aus der Feldliste umbenennen« ab Seite 515.

2.2 Formular auf Registerseiten oder als eigene Fenster?

Sie können anwendungsweit festlegen, ob Sie Formular jeweils über das komplette Access-Fenster erstrecken wollen oder ob diese als einzelne Fenster innerhalb des Access-Fensters erscheinen sollen.

Diese Einstellung legen Sie in den Access-Optionen fest, die Sie über das Ribbon öffnen können. Dazu klicken Sie auf das Ribbon-Tab *Datei* und wählen im nun erscheinenden Bereich unten links die Schaltfläche *Optionen* aus (siehe Abbildung 2.1).

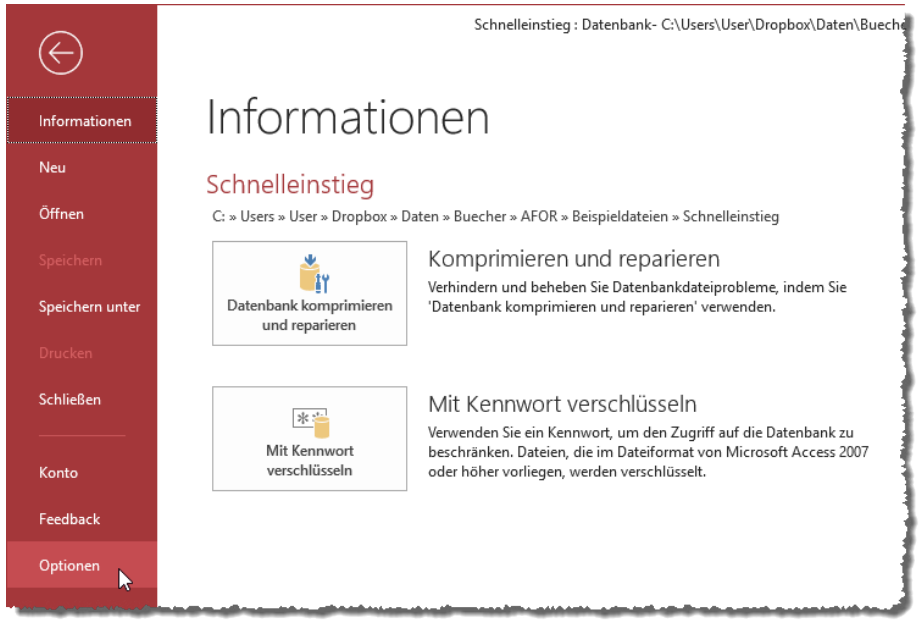


Abbildung 2.1: Öffnen des Optionen-Dialogs von Access

Hier finden Sie nun im Bereich *Aktuelle Datenbank* rechts unter *Anwendungsoptionen* die *Dokumentfensteroptionen*. Die Einstellung *Überlappende Fenster* öffnet jedes Formular als eigenes Fenster.

Die Einstellung *Dokumente im Registerkartenformat* zeigt jedes Formular als Seite eines Registers an. Mit der Option *Dokumentregisterkarten anzeigen* können Sie einstellen, ob die Registerlaschen angezeigt werden sollen oder nicht (siehe Abbildung 2.2).

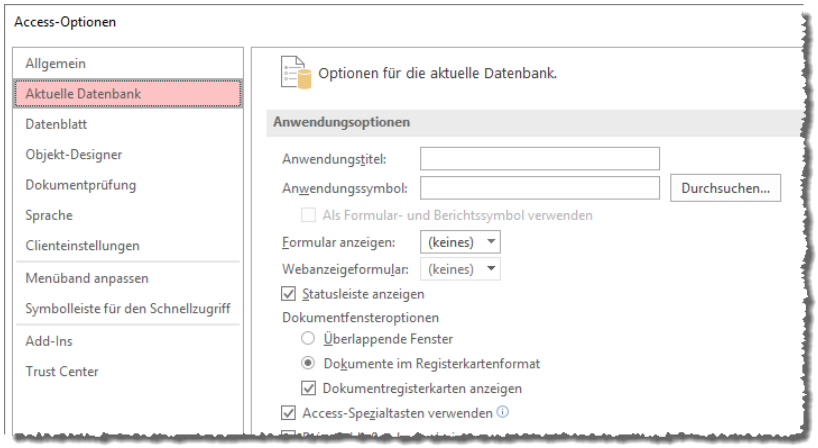


Abbildung 2.2: Einstellen der Dokumentfensteroptionen

Standardmäßig ist die Registerkarten-Anzeige aktiviert. Das heißt, dass ein neues Formular sich über den gesamten Arbeitsbereich von Access erstreckt. Außerdem zeigt Access dann am oberen Rand des Arbeitsbereichs eine Leiste mit Registerkarten an, über die Sie dann auf die geöffneten Elemente wie Formulare oder Berichte auf den Registerseiten zugreifen können.

Es gibt sicher Anwendungszwecke für diese Ansicht, aber die meisten Anwendungen, die ich kenne, verwenden die Anzeige der Elemente als überlappende Fenster im Arbeitsbereich. Also wähle ich standardmäßig die Einstellung *Überlappende Fenster* – und das ist auch die Einstellung, die bei der Erstellung der Beispiele für dieses Buch zum Einsatz kam.

Dies ist auch nicht die wichtigste Einstellung, die es in Access bezüglich der Formulare gibt, aber da fast alle folgenden Beispiele auf der Anzeige von Formularen als überlappende Fenster basieren, sollten Sie Ihre Anwendung, mit der Sie die Beispiele ausprobieren möchten, wie hier beschrieben einstellen.

2.3 Formulare anlegen

Es gibt verschiedene Möglichkeiten, ein Formular anzulegen. Eine haben Sie weiter oben bereits kennengelernt – dort haben wir den Ribbon-Befehl *Erstellen|Formulare|Formularentwurf* genutzt, um ein neues, leeres Formular in der Entwurfsansicht zu öffnen. Es gibt jedoch noch weitere Möglichkeiten, ein neues Formular anzulegen. Die meisten davon finden Sie ebenfalls in dem genannten Bereich des Ribbons (siehe Abbildung 2.3). In dieser Abbildung haben wir das Untermenü *Weitere Formulare* aufgeklappt, das weitere Möglichkeiten bietet. Nachfolgend schauen wir uns die verschiedenen Möglichkeiten an.

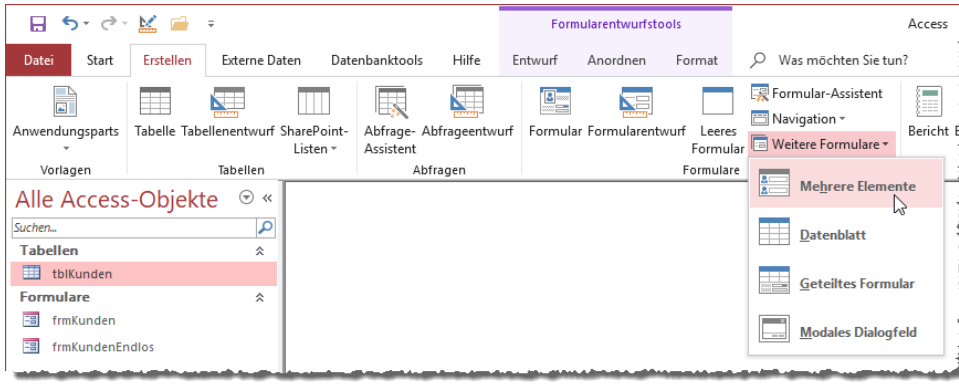


Abbildung 2.3: Ribbon-Befehle zum Anlegen neuer Formulare

» Eintrag *Formular*: Diese Option wird erst aktiviert, wenn eine Tabelle, eine Abfrage oder ein anderes Objekt im Navigationsbereich von Access markiert ist. Dann bietet dieser Befehl nämlich die Möglichkeit an, ein neues Formular auf Basis dieses Objekts zu erstellen. Mehr dazu weiter unten.

» Eintrag *Formularentwurf*: Diesen Eintrag kennen Sie bereits. Er öffnet ein neues, leeres Formular in der Entwurfsansicht.

» Eintrag *Leeres Formular*: Dieser Befehl öffnet ein neues, leeres Formular in der Formularansicht. Er ist uninteressant, weil Sie ein neues, leeres Formular immer zuerst bearbeiten wollen, bevor Sie es in der Formularansicht anschauen.

» Eintrag *Formularassistent*: Dieser Eintrag öffnet den Formular-Assistent, mit dem Sie festlegen können, welche Datensatzherkunft das Formular verwendet, welche Felder davon angezeigt werden sollen, wie das Layout aussehen soll, wie das Formular heißt und wie Sie es nach dem Erstellen öffnen wollen. Mehr dazu weiter unten.

» Eintrag *Navigation*: Dieser Eintrag öffnet ein weiteres Untermenü, mit dem Sie Einträge wie *Horizontale Registerkarten*, *Vertikale Registerkarten, links* et cetera auswählen können. Hiermit legen Sie ein neues Formular an, das direkt mit einem Navigationssteuerelement in einer bestimmten Anordnung ausgestattet ist. Mehr dazu weiter unten.

» Eintrag *Weitere Formulare|Mehrere Elemente*: Legt ein Endlosformular auf Basis des aktuell markierten Elements des Navigationsbereichs an. Mehr dazu weiter unten.

» Eintrag *Weitere Formulare|Datenblatt*: Legt ein Formular auf Basis des aktuell markierten Elements im Navigationsbereich in der Datenblattansicht an. Mehr dazu weiter unten.

» Eintrag *Weitere Formulare|Geteiltes Formular*: Legt ein Formular in einer speziellen Ansicht an, bei der die Daten gleichzeitig in der Formularansicht und in der Datenblattansicht angezeigt werden. Mehr dazu weiter unten.

» Eintrag *Weitere Formulare|Modales Dialogfeld*: Legt ein Formular in einer speziellen Ansicht an, bei dem die üblichen Elemente wie Navigationsschaltflächen oder der Datensatzmarkierer fehlen. Dafür werden aber automatisch zwei Schaltflächen namens *OK* und *Abbrechen* hinzugefügt. Mehr dazu weiter unten.

In den folgenden Abschnitten prüfen wir, wie wir einige der verschiedenen Möglichkeiten zum Anlegen von Formularen nutzen können.

2.3.1 Formular auf Basis einer Tabelle oder Abfrage anlegen

Diese Option im Ribbon wird immer aktiviert, wenn ein passendes Element im Navigationsbereich markiert ist. Das können Sie natürlich gezielt einsetzen: Wenn Sie schnell einmal ein Formular erstellen wollen, das die Daten einer Tabelle oder Abfrage in der Formularansicht anzeigt, markieren Sie einfach die Tabelle oder Abfrage und klicken dann auf den Ribbon-Befehl *Erstellen|Formulare|Formular*.

Wenn Sie das etwa für die von uns weiter oben vorgestellten Tabelle *tblKunden* erledigen, erhalten Sie binnen kürzester Zeit das Formular aus Abbildung 2.4.

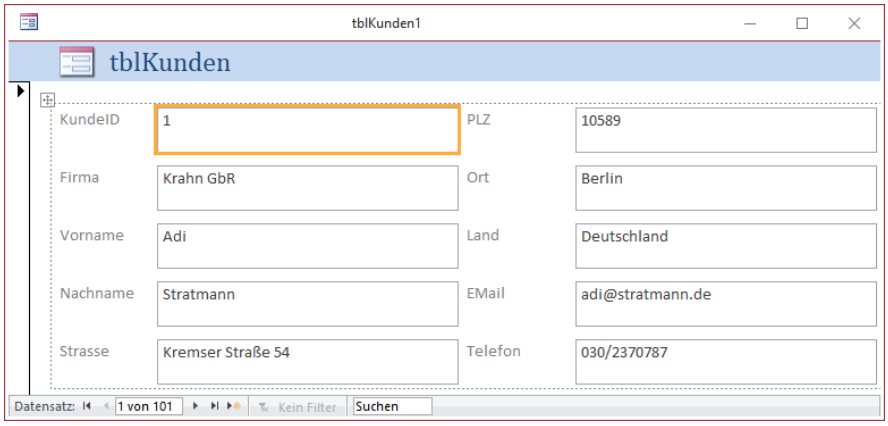


Abbildung 2.4: Ein per Ribbon erstelltes Formular auf Basis einer Tabelle oder Abfrage

In den meisten Fällen wird dies nicht Ihren Anforderungen entsprechen, aber wenn Sie keinen Aufwand investieren und die Daten schnell einmal einsehen wollen, ist dies eine gute Möglichkeit für die Formularansicht der Daten einer Tabelle oder Abfrage.

3 Formulare entwerfen

Am Anfang leisten die Assistenten und Vorlagen für Formulare, die Sie im Ribbon finden, gute Unterstützung beim Erstellen der ersten Formulare. Sie werden aber schnell bemerken, dass Sie die besten Resultate erzielen, wenn Sie die Formulare manuell erstellen. Dazu sind zwar meist ein paar Mausklicks mehr nötig, aber Access unterstützt Sie mit einigen Hilfsmitteln bei der Erstellung Ihrer Formulare.

3.1 Allgemeine Eigenschaften

In den folgenden Abschnitten stellen wir einige allgemeine Eigenschaften für Formulare vor.

3.1.1 Titel festlegen

Beginnen wir mit ein wenig Optik: Normalerweise zeigt das Formular in der Titelleiste den Namen des Formulars an, also etwa *frmBeispiel*. Dies können Sie mit der Eigenschaft *Beschriftung* ändern. Der Wert, den Sie hier eintragen, erscheint dann in der Formularansicht in der Titelleiste (siehe Abbildung 3.1).

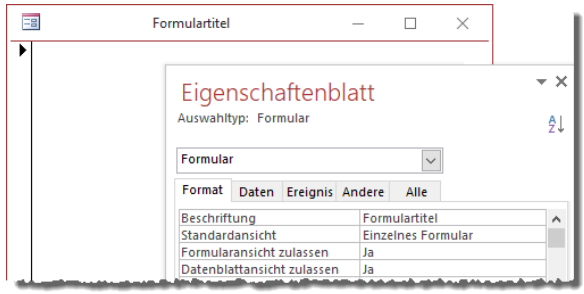


Abbildung 3.1: Formulartitel per Eigenschaft anpassen

3.1.2 Hintergrundbild festlegen

Sie können über die Eigenschaft *Bild* ein Hintergrundbild für den Detailbereich des Formulars angeben. Dazu öffnen Sie über die Schaltfläche mit den drei Punkten (...) in der Eigenschaft *Bild* einen *Datei öffnen*-Dialog und wählen eine beliebige Bilddatei aus, die dann in der Datenbank gespeichert und als Hintergrund angezeigt wird. Neben der Eigenschaft *Bild* gibt es noch weitere Eigenschaften, die für die Darstellung wichtig sind:

» *Bildtyp*: Legt fest, ob das Bild im Formular eingebettet werden soll (*Eingebettet*), über den Pfad verknüpft (*Verknüpft*) oder in einer speziellen Tabelle gespeichert (*Freigegeben*). Im letzteren Fall können Sie das Bild noch an anderen Stellen nutzen und brauchen es nur einmal in der Datenbank zu speichern. Mehr dazu unter »Bilder in Formularen« ab Seite 367.

» *Bild nebeneinander*: Stellt für den Wert *Ja* das Bild mehrfach über- oder nebeneinander dar.

» *Bildausrichtung*: Stellt ein, wie das Bild im Detailbereich ausgerichtet werden soll.

» *Bildgrößenmodus*: Legt fest, wie das Bild in den Detailbereich eingepasst wird. *Abschneiden* zeigt das Bild in der Originalgröße an, *Dehnen* dehnt das Bild so, dass es sich dem Detailbereich in horizontaler und vertikaler Richtung anpasst und *Zoomen* passt das Bild in den Detailbereich ein, behält aber die Proportionen bei.

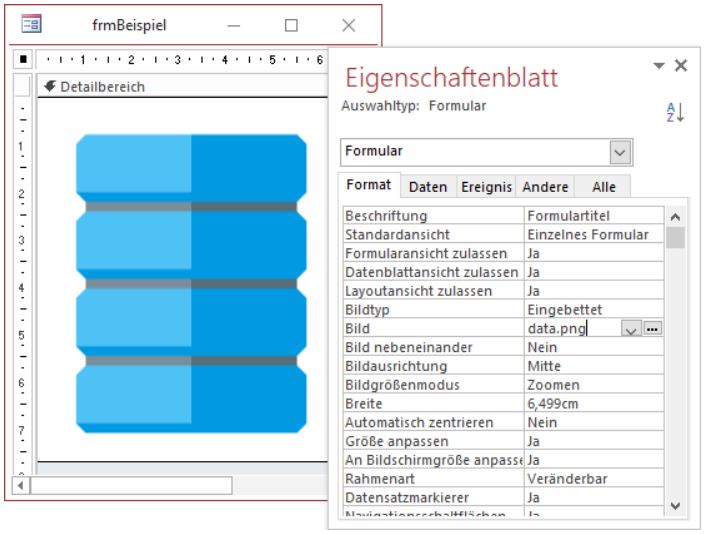


Abbildung 3.2: Einfügen eines Hintergrundbilds

3.1.3 Breite und Höhe festlegen

Die Breite und die Höhe eines Formulars hängt von den Eigenschaften unterschiedlicher Formularbereiche ab. Die Eigenschaft *Breite* finden Sie beispielsweise nur, wenn Sie das Formular selbst markiert haben, also etwa auf den grauen Hintergrund des Formulars geklickt haben oder auf das kleine Kästchen, das sich im Schnittpunkt der Lineale befindet.

Die Eigenschaft *Höhe* können Sie für alle Formularbereiche einstellen, also etwa für den Formularkopf, den Detailbereich und den Formularfuß. Um die Eigenschaft *Höhe* für den jeweiligen Bereich zu aktivieren, klicken Sie auf den Kopf des gewünschten Bereichs. Die Summe ergibt letztlich die Höhe des angezeigten Formularbereichs (ohne Titelleiste und Rahmen).

Tatsächlich sich die Größe eines Formulars, wenn Sie es schließen und dann wieder öffnen, aus der Summe der Höhe der einzelnen Bereiche mal der Breite plus Titelleiste und Rahmen.

Wenn Sie das Formular allerdings direkt aus der Entwurfsansicht heraus etwa in der Formularansicht anzeigen, behält es die vorherige Größe bei. Dies gilt jedoch auch nur, wenn die Eigenschaft *Größe anpassen* den Wert *Ja* enthält.

Nur dann wird der Rahmen des Formulars an die Größe der Formularebereiche angepasst. Wenn Sie *Größe anpassen* auf *Nein* einstellen und das Formular öffnen, wird es in der Größe angezeigt, die es beim letzten Speichern hatte.

3.1.4 Formular beim Öffnen zentrieren

Wenn Sie für ein Formular festlegen wollen, dass es es beim Öffnen zentriert im Anwendungsfenster angezeigt werden soll, stellen Sie die Eigenschaft *Automatisch zentrieren* auf den Wert *Ja* ein. Unter »Formulare positionieren« ab Seite 511 zeigen wir, welche Optionen zur Positionierung es noch gibt. Diese erfordern allerdings alle den Einsatz von VBA.

3.2 Entwurf per Entwurfsansicht oder Layoutansicht

Access bietet zwei Ansichten an, in denen Sie den Entwurf von Formularen beeinflussen können:

» Die erste gibt es seit der ersten Version von Access – die Entwurfsansicht. Hier werden wir die meisten der in diesem Buch beschriebenen Schritte durchführen.

» Die zweite Ansicht ist die Layoutansicht. Die Layoutansicht ist ein Hybrid aus der Formularansicht und der Entwurfsansicht. Da wir uns später noch ausgiebig mit der Entwurfsansicht beschäftigen werden, schauen wir uns hier kurz die Eigenschaften der Layoutansicht an.

3.2.1 Formular in der Layoutansicht öffnen

Weiter oben haben wir uns die verschiedenen Möglichkeiten angesehen, wie Sie ein Formular erstellen können. Dieses konnten Sie in der Entwurfsansicht erstellen oder gleich als fertiges Formular, das dann in der Formularansicht angezeigt wird.

Das Erstellen eines Formulars in der Layoutansicht vermissen wir dort, und so können Sie diese Ansicht auch erst für bereits erstellte Formulare auswählen. Wenn das Formular bereits gespeichert und im Navigationsbereich aufgelistet ist, können Sie es mit der rechten Maustaste anklicken und den Eintrag *Layoutansicht* auswählen. Dann erscheint das Formular etwa wie in Abbildung 3.3. Sie können diese Ansicht auch für ein bereits geöffnetes Formular auswählen, indem Sie den entsprechenden Kontextmenü-Befehl für dieses Formular aufrufen.

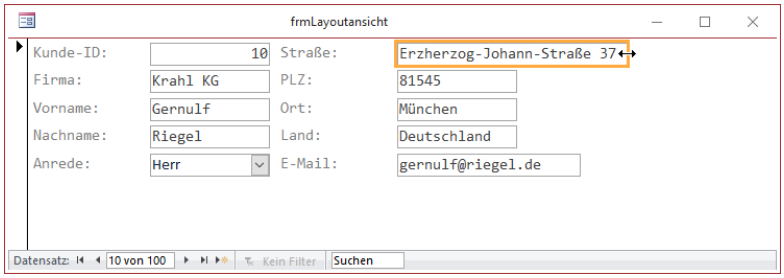


Abbildung 3.3: Ein Formular in der Layoutansicht

3.2.2 Vorteile und Nachteile der Layoutansicht

In der Abbildung erkennen Sie bereits den Unterschied zur Entwurfsansicht: Das Formular wird nämlich direkt mit den zugrunde liegenden Daten gefüllt und Sie können durch die Datensätze blättern. Das ist sehr hilfreich, um zu prüfen, ob die Steuerelemente groß genug für die Inhalte der gebundenen Felder ausgelegt sind. Falls nicht – und das ist die Abgrenzung der Layoutansicht zur Formularansicht –, können Sie die Größe der Steuerelemente anpassen, wie hier im Falle der Breite des Textfeldes zur Anzeige der Straße.

Sie können in der Layoutansicht auch gebundene und ungebundene Steuerelemente hinzufügen, aber die Ausrichtung der Steuerelemente ist in der Entwurfsansicht doch wesentlich genauer und einfacher als in der Layoutansicht. Daher würden wir diese Ansicht ausschließlich dazu nutzen, die Größe von Steuerelementen an die anzuzeigenden Inhalte anzupassen.

3.2.3 Layoutansicht sperren

Wenn Sie nicht wollen, dass der Benutzer das Formular in der Layoutansicht anzeigt, können Sie dies durch Einstellen der Eigenschaft *Layoutansicht zulassen* auf den Wert *Nein* deaktivieren (siehe Abbildung 3.4).

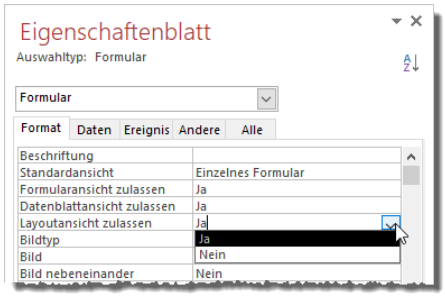


Abbildung 3.4: Eigenschaft zum Zulassen der Layoutansicht

Sie können die Layoutansicht auch für die komplette Datenbank deaktivieren. Dazu öffnen Sie die Access-Optionen und wählen im Bereich *Aktuelle Datenbank* unter *Anwendungsoptionen* die Option *Layoutansicht aktivieren* ab (siehe Abbildung 3.5).

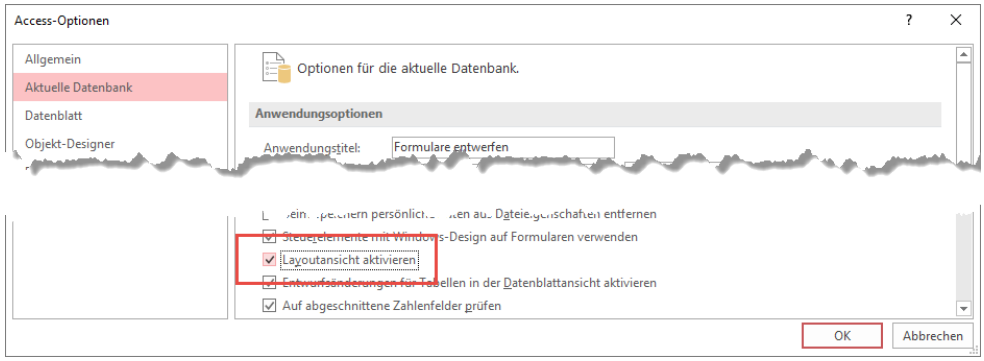


Abbildung 3.5: Deaktivieren der Layoutansicht

3.3 Steuerelemente hinzufügen

Zum Hinzufügen von Steuerelementen verwenden Sie zwei Möglichkeiten:

» über die Elemente des Ribbons unter *Entwurf|Steuerelemente* oder

» über die Feldliste, nachdem Sie einem Formular eine Datensatzquelle zugewiesen haben.

Die erstgenannte Variante bietet die Steuerelemente aus Abbildung 3.6.

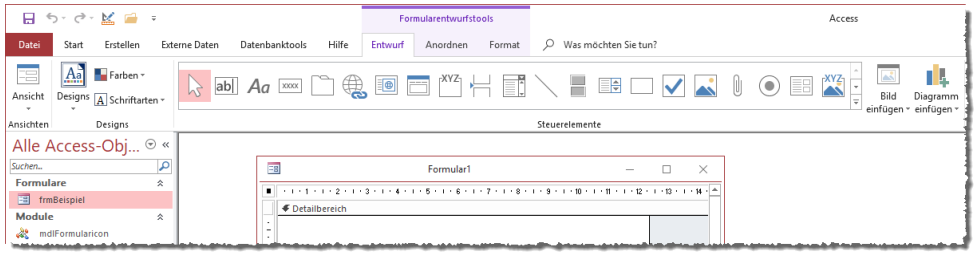


Abbildung 3.6: Die Leiste mit der Auswahl der verfügbaren Steuerelemente

3.3.1 Ungebundenes Steuerelement hinzufügen

Um ein Steuerelement aus dieser Liste zum Formularentwurf hinzuzufügen, klicken Sie den entsprechenden Befehl im Ribbon an und platzieren dann die Maus an der Stelle im Formular, wo

Sie das Steuerelement anlegen wollen. Der Mauszeiger zeigt dann den Typ des einzufügenden Steuerelements an – in Abbildung 3.7 beispielsweise für ein Textfeld. Betätigen Sie dann die linke Maustaste, wird das Steuerelement in der Standardgröße an der gewünschten Stelle angelegt.

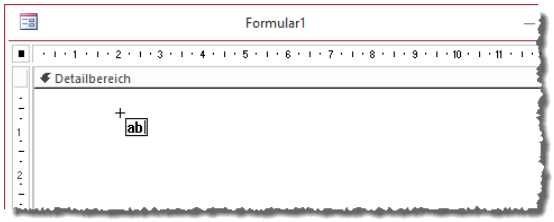


Abbildung 3.7: Einfügen eines Textfeld-Steuerelements in ein Formular

Sie können aber auch einen Rahmen in der gewünschten Größe aufziehen, um das Steuerelement gleich in einer anderen Größe anzulegen als in der Standardgröße.

Nach dem Einfügen erscheint das markierte Steuerelement dann im Entwurf des Formulars (siehe Abbildung 3.8).

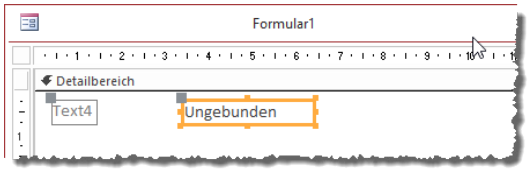


Abbildung 3.8: Ein frisch eingefügtes ungebundenes Steuerelement

Das eingefügte Steuerelement ist, unabhängig vom Steuerelementtyp, ungebunden. Ungebunden bedeutet, dass es nicht an ein Feld der Datensatzquelle des Formulars gebunden ist. Mit den Worten des Eigenschaftenblatts gesprochen: Die Eigenschaft *Steuerelementinhalt* ist leer.

3.3.2 Ungebundenes Steuerelement nachträglich binden

Wenn Sie ein Steuerelement, das sie auf diese Weise zum Formular hinzugefügt haben, vom ungebundenen in den gebundenen Zustand überführen wollen, müssen Sie für seine Eigenschaft *Steuerelementinhalt* eines der Felder der Datensatzquelle des Formulars auswählen. Dazu muss diese erst einmal gefüllt sein. Wir gehen davon aus, dass sich die Tabelle *tblKunden* wie in der Beispieltabelle in der Datenbank befindet. Dann führen Sie nun zwei Schritte für ein neues, leeres Formular durch. Der erste ist das Auswählen der Tabelle *tblKunden* als Datensatzquelle für das Formular (siehe Abbildung 3.9).

Abbildung 3.9: Auswahl der Datensatzquelle für ein Formular

Danach können Sie dann für ein neu hinzugefügtes, ungebundenes Textfeld die Eigenschaft *Steuerelementinhalt* auf eines der Felder der als Datensatzquelle angegebenen Tabelle einstellen. Das Auswahlfeld der Eigenschaft zeigt alle möglichen Felder an. Wir wählen hier das Feld *KundeID* aus (siehe Abbildung 3.10).

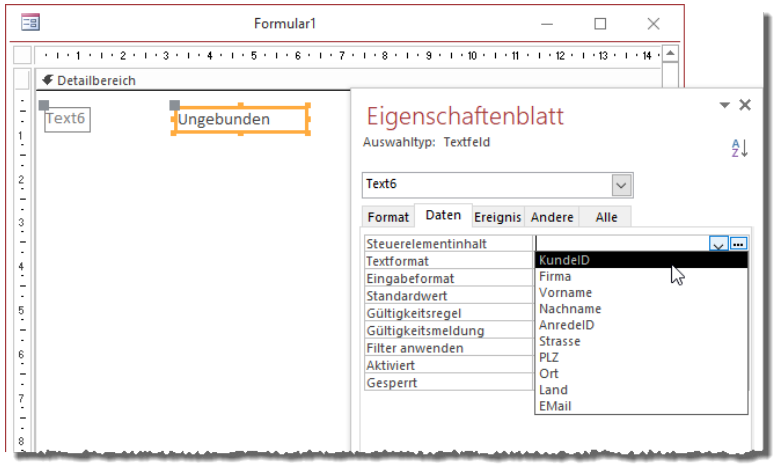


Abbildung 3.10: Auswahl eines Feldes für die Eigenschaft *Steuerelementinhalt* des Textfeldes

Wenn Sie nun in die Formularansicht wechseln, zeigt das Textfeld den Wert des Feldes *KundeID* für den aktuell ausgewählten Datensatz an (siehe Abbildung 3.11). Wenn Sie nun mit den Navigationsschaltflächen durch die Datensätze blättern, sehen Sie, wie sich der Wert dieses Feldes ändert.

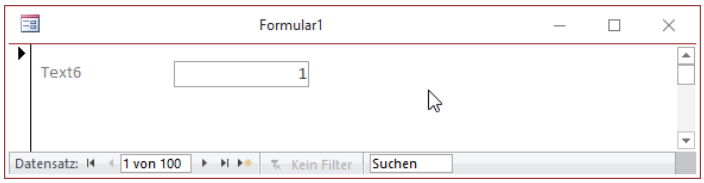


Abbildung 3.11: Formular mit einem gebundenen Textfeld

3.3.3 Gebundenes Steuerelement hinzufügen

Ein als ungebundenes Steuerelement eingefügtes Textfeld können Sie natürlich wie oben ohne weiteres in ein gebundenes Steuerelement umwandeln, indem Sie seine Eigenschaft *Steuerelementinhalt* mit dem Namen eines der Felder der Datensatzquelle des Formulars füllen. Dazu muss natürlich zunächst einmal eine Datensatzquelle angegeben sein. Es gibt jedoch noch einen viel eleganteren Weg, um ein gebundenes Steuerelement, etwa ein Textfeld, zu einem Formular hinzuzufügen.

Dazu bleiben wir bei dem Formular, dem wir gerade schon ein Textfeld hinzugefügt haben, das wir nachträglich an ein Feld der Datensatzquelle des Formulars gebunden haben.

Wir wechseln wieder in die Entwurfsansicht. Dann rufen wir im Ribbon den folgenden Eintrag auf: *Entwurf|Tools|Vorhandene Felder hinzufügen* (siehe Abbildung 3.12).

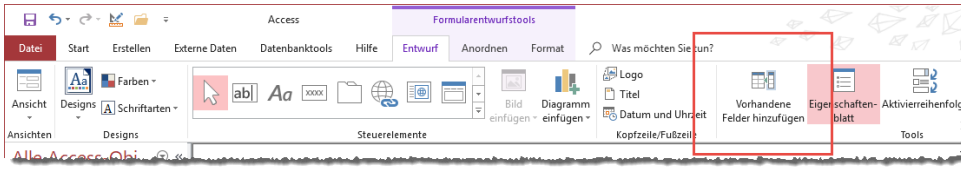


Abbildung 3.12: Der Befehl *Vorhandene Felder hinzufügen* im Ribbon

Das Eigenschaftenblatt wandelt sich daraufhin in die Feldliste um. Diese enthält im unteren Bereich zunächst einmal alle Felder der über die Eigenschaft *Datensatzquelle* gebundenen Tabelle *tblKunden*. Sie können nun ein oder mehrere Felder aus dieser Liste markieren (siehe Abbildung 3.13).

Zum Markieren mehrerer Felder klicken Sie das erste Feld an und markieren dann bei gedrückter *Strg*-Taste weitere einzelne Felder oder Sie markieren das erste von mehreren zusammenhängenden Einträgen und dann bei gedrückter *Umschalt*-Taste das letzte Element. Dann ziehen Sie die markierten Einträge per Drag and Drop an die gewünschte Stelle im Detailbereich des Formularentwurfs (siehe Abbildung 3.14).

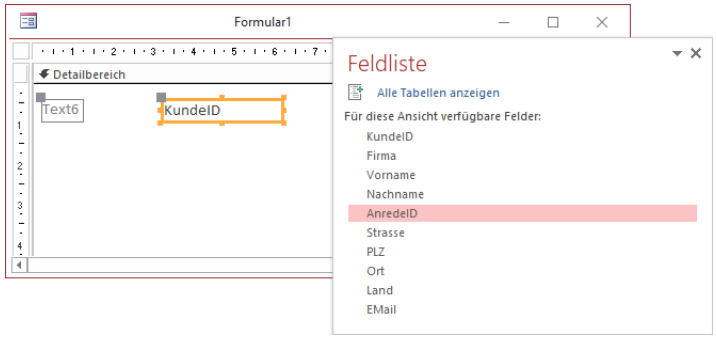


Abbildung 3.13: Die Feldliste eines Formulars

Abbildung 3.14: Hinzufügen mehrerer Felder per Drag and Drop

Wenn Sie danach in die Formularansicht wechseln, zeigt dieses alle in den Detailbereich gezogenen gebundenen Steuerelemente mit den damit verknüpften Daten aus der Tabelle *tblKunden* an (siehe Abbildung 3.15).

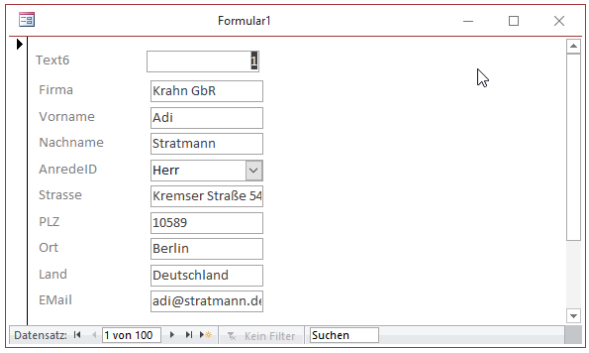


Abbildung 3.15: Formular mit frisch hinzugefügten gebundenen Steuerelementen

Welche Vorteile bringt nun das Hinzufügen der gebundenen Steuerelemente aus der Feldliste gegenüber dem Einfügen eines ungebundenen Steuerelements, das wir anschließend an das gewünschte Feld binden? Der erste Vorteil ist offensichtlich: Das Hinzufügen der Felder aus der Feldliste ist mit einfachem Drag and Drop getan und das Steuerelement ist dann bereits an das entsprechende Feld gebunden. Den zweiten können Sie auch gleich erkennen: Das mit dem Textfeld automatisch hinzugefügte Bezeichnungsfeld enthält den Feldnamen als Beschriftung, den Sie in den meisten Fällen auch gleich beibehalten können. Lediglich bei Feldern wie *Strasse* (in *Straße*) oder *AnredeID* (in *Anrede*) sind Korrekturen nötig.

Der nächste Vorteil: Wenn in der zugrunde liegenden Tabelle ein Nachschlagefeld für das Feld hinterlegt wurde, legt Access beim Hinzufügen des Feldes über die Feldliste auch automatisch ein Kombinationsfeld für dieses Feld an – wie hier beim Feld *AnredeID* der Fall.

Und Sie können über die Definitionen der Felder der Tabellen und ihrer Eigenschaften noch weitere Vereinfachungen schaffen. Mehr dazu in »Tabellenentwurf für Formulare optimieren« ab Seite 52.

3.3.4 Einstellungen für das Hinzufügen gebundener Steuerelemente

Wenn Sie gebundene Steuerelemente zu einem Formular hinzufügen, geschieht das nach bestimmten Vorgaben. Ein paar davon beruhen auf den Einstellungen, die im Tabellenentwurf für die hinzugefügten Felder vorgenommen wurden. Andere können Sie in den Eigenschaften der Vorlage für ein Steuerelement für ein Formular definieren. Wie geht das? Dazu markieren Sie, während sich das Formular in der Entwurfsansicht befindet, das Steuerelement, dessen Eigenschaften Sie anpassen möchten – beispielsweise ein Textfeld. Das Eigenschaftenblatt zeigt dann ein paar zusätzliche Eigenschaften an, die dort nicht erscheinen, wenn Sie die Eigenschaften für eines der bereits vorhandenen Textfelder anzeigen. Stattdessen erscheinen die Eigenschaften, die man grundsätzlich für alle einzufügenden Steuerelemente, in diesem Fall Textfelder, definieren kann und die dann nach dem Einfügen eines neuen Textfeldes für dieses übernommen werden. Es handelt sich also um die Eigenschaften der Vorlage für ein bestimmtes Steuerelement für dieses Formular.

Grundsätzlich gibt es also eine ganze Reihe von Eigenschaften, die Sie individuell einstellen können, um diese als Voreinstellung für Steuerelemente des aktuellen Formulars festzulegen. Es gibt jedoch auch ein paar Eigenschaften, die sich nur über die Voreinstellung realisieren lassen. Dabei handelt es sich um die folgenden Eigenschaften:

» *Mit Bezeichnungsfeld*: Legt fest, ob mit dem Steuerelement automatisch ein Bezeichnungsfeld angelegt werden soll.

» *Mit Doppelpunkt*: Gibt an, ob der Text des Bezeichnungsfelds am Ende einen Doppelpunkt enthalten soll.

» *Bezeichnungsfeld X*: Gibt die horizontale Position des Bezeichnungsfeldes relativ zum Steuerelement an. Bei Textfeldern lautet dieser Wert beispielsweise -3cm, sodass die Bezeichnungsfelder hier drei Zentimeter links vom Steuerelement angelegt werden.

» *Bezeichnungsfeld Y*: Gibt die vertikale Position des Bezeichnungsfeldes relativ zum Steuerelement an. Bei Textfeldern beispielsweise *0*, da die Bezeichnungsfelder hier links neben dem Textfeld angezeigt werden sollen.

» *Bezeichnungsausrichtung*: Ausrichtung des Textes im Bezeichnungsfeld. Hier sind die üblichen Werte verfügbar.

Wenn Sie eine oder mehrere dieser Eigenschaften ändern, damit sich diese auf die Neuerstellung von Steuerelementen im aktuellen Formular auswirken, dann müssen Sie die Einstellungen speichern, bevor Sie ein Steuerelement auf Basis der geänderten Vorlage anlegen. Das erledigen Sie am einfachsten mit der Tastenkombination *Strg + S*. Angenommen, Sie stellen *Mit Doppelpunkt* auf *Ja*, *Bezeichnungsfeld X* auf *0cm*, *Bezeichnungsfeld Y* auf *0,6cm* und *Bezeichnungsausrichtung* auf *Rechts* ein und speichern diese Einstellung dann mit *Strg + S*. Dann wird ein neues, aus der Feldliste in den Formularentwurf gezogenes Steuerelement wie in Abbildung 3.16 eingefügt.

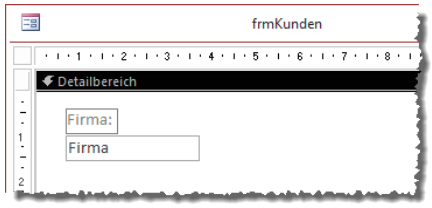


Abbildung 3.16: Text- und Bezeichnungsfeld mit alternativen Voreinstellungen

Das sieht nicht allzu schlecht aus, wenn Sie es so benötigen. Wenn Sie nun allerdings mehrere Felder gleichzeitig aus der Feldliste in den Entwurf ziehen, überlappen die Bezeichnungsfelder leider die jeweils darüber liegenden Steuerelemente (siehe Abbildung 3.17).

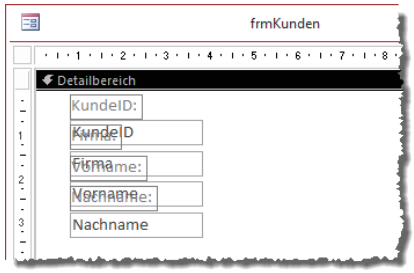


Abbildung 3.17: Überlappende Bezeichnungsfelder und Steuerelemente

Bedenken Sie, dass die Standardwerte für die Formulare immer nur für das aktuelle Formular gespeichert werden können, also normalerweise nicht auf andere Formulare übertragen werden können. Sie haben allerdings zwei Möglichkeiten, um dies zu umgehen:

» Sie können die Einstellungen für ein Formular festlegen und dieses dann jeweils als Vorlage für neue Formulare verwenden. Dazu kopieren Sie das als Vorlage definierte Formular und fügen es als neues Formular unter dem gewünschten Namen wieder ein.

» Oder Sie speichern das als Vorlage gestaltete Formular unter dem Namen *Normal* und geben diesen, sofern noch nicht so angegeben, für die Eigenschaft *Formularvorlage* in den Access-Optionen an. Mehr dazu erläutern wir unter »Formular-Vorlage« ab Seite 133.

3.3.5 Tabellenentwurf für Formulare optimieren

Wenn Sie mit Access-Tabellen arbeiten und nicht etwa mit eingebundenen SQL Server-Tabellen, können Sie über die Definition der Access-Tabellen einige Vorbereitungen treffen, die Ihnen die folgende Arbeit beim Erstellen von Formularen erleichtern. Das betrifft allerdings ausschließlich das Hinzufügen von Steuerelementen aus der Feldliste bei einem an eine Datensatzquelle gebundenen Formular.

Normalerweise legen Sie Tabellen in der Entwurfsansicht vielleicht nur mit den nötigsten Informationen an – so wie wir es bei der Tabelle *tblArtikel* aus der Beispieldatenbank zu diesem Ka pitel getan haben (siehe Abbildung 3.18).

Abbildung 3.18: Tabellenentwurf mit den nötigsten Informationen

Sie können aber schon hier einige Schritte durchführen, die Ihnen später viel Zeit beim Hinzufügen der Felder aus der Feldliste in den Formularentwurf sparen können.

3.3.6 Feldbeschreibungen anpassen

Der erste Schritt ist das Anpassen der Feldbeschreibungen. Hier können Sie Texte anzeigen, die dem Benutzer in der Statusleiste angezeigt werden sollen, wenn dieser den Fokus auf eines der Felder verschiebt (siehe Abbildung 3.19).

Abbildung 3.19: Hinzufügen von Feldbeschreibungen zu den Feldern der Tabelle

Wenn sie die Feldbeschreibung angepasst haben, fragt Access Sie, ob es den Statusleistentext überall dort aktualisieren soll, wo dieses Feld verwendet wird (siehe Abbildung 3.19). Dies sollten Sie akzeptieren.

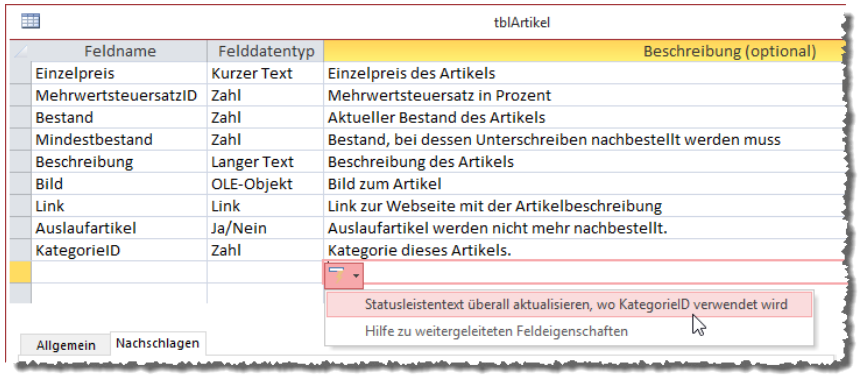


Abbildung 3.20: Angebot, die Statusleistentexte zu aktualisieren

3.3.7 Beschriftung anpassen

Eine weitere Möglichkeit, das Anlegen von Steuerelementen auf Basis der Felder einer Tabelle optimal vorzubereiten, ist das Einstellen der Eigenschaft *Beschriftung* der Felder. Dies ist genau da sinnvoll, wo Sie aus Gründen der Konventionen für die Benennung von Feldern (keine

Leerzeichen, Umlaute et cetera) nicht den Feldnamen vergeben haben, den Sie gern auch als Feldbeschriftung genutzt hätten. In der Tabelle *tblArtikel* finden wir zwei Beispiele: Das Primärschlüsselfeld *ArtikelID* würde man vermutlich eher als *Artikel-ID* schreiben und das Fremdschlüsselfeld *MehrwertsteuersatzID* soll als Feldbeschriftung auch eher als *MwSt.-Satz* erscheinen. Genau das erreichen wir, indem wir ein Feld markieren und dann unten in den Eigenschaften auf der Seite *Allgemein* einen neuen Wert für das Feld *Beschriftung* eintragen (siehe Abbildung 3.21).

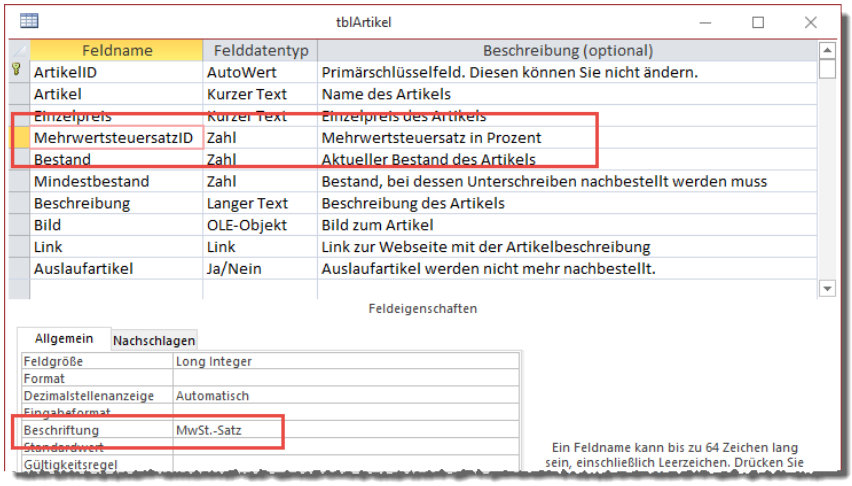


Abbildung 3.21: Anpassung der Beschriftung eines Feldes

Beschriftung in der Abfrage anpassen

In den meisten Fällen werden Sie nicht direkt eine Tabelle, sondern eine Abfrage als Datensatzquelle eines Formulars verwenden. In diesem Fall haben Sie in der Abfrage nochmals die Möglichkeit, die Beschriftung des Bezeichnungsfeldes für Steuerelemente, die sie aus der Feldliste in den Formularentwurf ziehen, anzupassen. Markieren Sie das gewünschte Feld und blenden Sie mit der Taste *F4* das Eigenschaftsfenster ein, um die Eigenschaften *Beschreibung* und *Beschriftung* anzupassen (siehe Abbildung 3.22).

3.3.8 Nachschlagefelder

Es gibt Programmierer, die die Einrichtung von Nachschlagefeldern nicht gutheißen, weil man zum Beispiel für diese etwa in der Datenblattansicht nicht mehr die eigentlichen Feldwerte vorfindet, sondern immer die entsprechenden Daten aus der verknüpften Tabelle angezeigt werden.

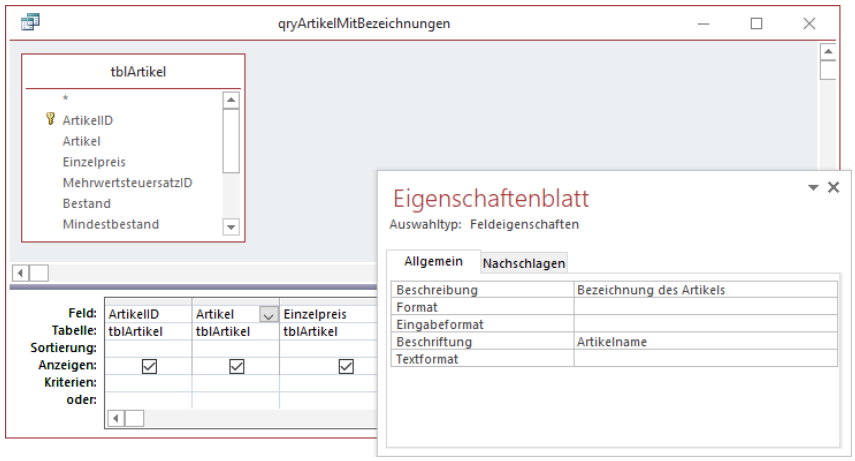


Abbildung 3.22: Einstellen von Beschreibung und Beschriftung über die Eigenschaften eines Abfragefeldes

Im Hinblick auf die anschließende Gestaltung von Formularen auf Basis von Tabellen mit solchen Feldern muss man allerdings die Vorteile hervorheben: Wenn Sie ein Nachschlagefeld für ein Feld definieren, das im Prinzip die gleichen Eigenschaften wie ein Kombinationsfeld erwartet, dann wird ein solches Feld beim Hineinziehen aus der Feldliste in den Formularentwurf automatisch als Kombinationsfeld angelegt.

Um im Tabellenentwurf ein Nachschlagefeld anzulegen, wie wir es etwa in der Tabelle *tblArtikel* für das Feld *KategorieID* gemacht haben, sind nur wenige Schritte nötig. In diesem Fall gilt die Voraussetzung, dass unsere Tabelle ein Fremdschlüsselfeld erhalten soll, das zur Auswahl der Daten einer anderen Tabelle verwendet werden soll – wie in diesem Fall bei der Tabelle *tblArtikel*, dem Fremdschlüsselfeld *KategorieID* und der Tabelle *tblKategorien* der Fall:

» Sie legen das Feld an, also beispielsweise *KategorieID*.

» Sie wählen als Datentyp *Nachschlage-Assistent* aus.

» Im ersten Schritt des Assistenten behalten Sie die Option *Das Nachschlagefeld soll die Werte aus einer Tabelle oder Abfrage abrufen* bei.

» Im zweiten Schritt wählen Sie die Tabelle aus, aus der die zu verknüpfenden Daten stammen, hier also *tblKategorien*.

» Im dritten Schritt wählen Sie die Felder aus, wobei Sie darauf achten sollten, dass das Primärschlüsselfeld als erstes Feld ausgewählt wird und als zweites ein Feld das Feld der Tabelle, das den im Kombinationsfeld anzuzeigenden Wert liefert – hier das Feld *Kategorie*.

» Im vierten Schritt legen Sie für das Feld *Kategorie* eine aufsteigende Sortierung fest.

» Im fünften Schritt stellen behalten Sie die Einstellung bei, dass die Schlüsselspalte ausgeblendet werden soll. Das sorgt hier dafür, dass das Primärschlüsselfeld nicht im Nachschlagefeld angezeigt wird.

» Im sechsten Schritt stellen Sie bei Bedarf ein, dass Datenintegrität aktiviert werden soll und ob eine Löschweitergabe oder Löschbeschränkung gewünscht ist.

» Schließlich klicken Sie auf *Fertigstellen*, um das Nachschlagefeld mit den gewählten Einstellungen anzulegen.

Wenn Sie nun für das neue Feld unten die Registerkarte *Nachschlagen* der Eigenschaften aktivieren, erhalten Sie die Ansicht aus Abbildung 3.23. Diese Eigenschaften entsprechen, wie Sie später bei der Beschreibung der Kombinationsfelder unter »Das Kombinationsfeld« ab Seite 281 erfahren werden, genau den dort verwendeten Eigenschaften. Und diese werden beim Anlegen eines Feldes aus der Feldliste, das als Nachschlagefeld eingerichtet wurde, aus dem Tabellenentwurf übernommen. Sie sorgen also durch Einrichten eines Feldes als Nachschlagefeld dafür, dass Sie ein gebundenes Kombinationsfeld zum Formular hinzufügen können, ohne nochmals die notwendigen Einstellungen vornehmen zu müssen.

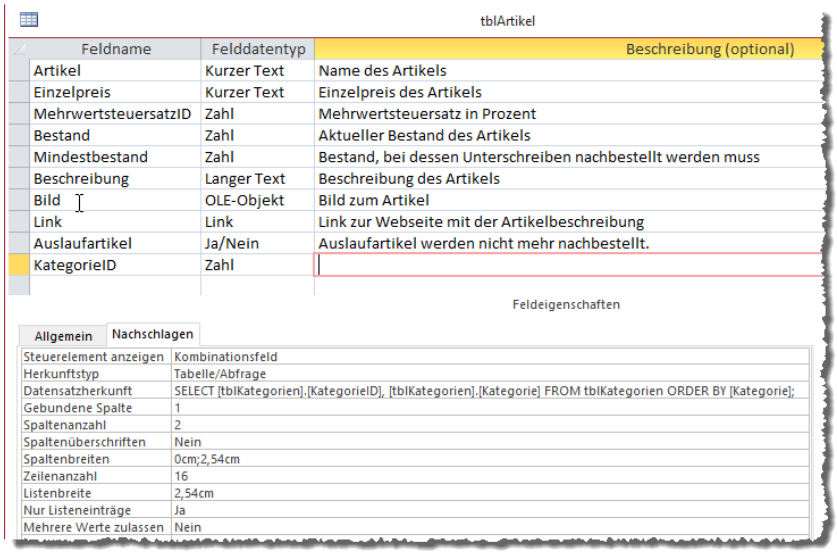


Abbildung 3.23: Eigenschaften, die durch den Nachschlage-Assistenten eingerichtet wurden

3.3.9 Kontrollkästchen

Bei *Ja/Nein*-Felder sieht es ähnlich aus. Mit dem Unterschied, dass die notwendigen Einstellungen im Bereich *Nachschlagen* der Feldeigenschaften automatisch beim Einfügen eines Feldes mit dem Datentyp *Ja/Nein* vorgenommen werden – in diesem Fall durch die Auswahl des Wertes

Kontrollkästchen für die Eigenschaft *Steuerelement anzeigen* (siehe Abbildung 3.24). Dadurch wird beim Anlegen eines Steuerelements auf Basis dieses Feldes aus der Feldliste auch direkt ein Kontrollkästchen zum Formular hinzugefügt, das an das Feld *Auslaufartikel* gebunden ist.

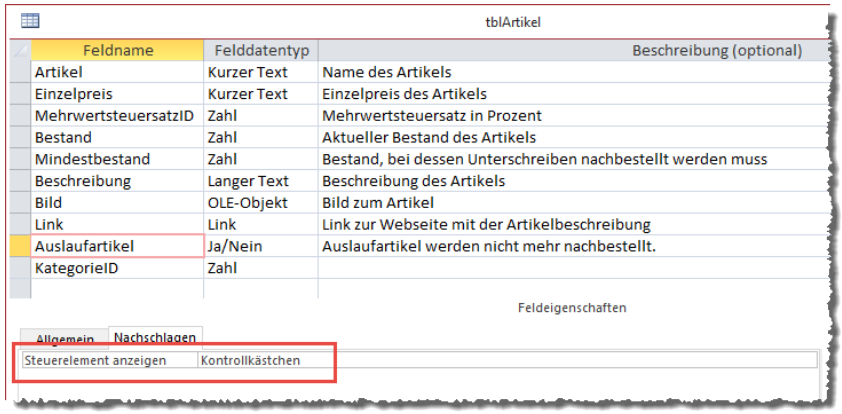


Abbildung 3.24: Steuerelement-Vorbelegung für *Ja/Nein*-Felder

Für die übrigen Felddatentypen wie *Kurzer Text*, *Zahl* et cetera hat diese Eigenschaft übrigens den Wert *Textfeld*. Sie können hier aber bei Bedarf auch einen der anderen Werte auswählen. Im Falle eines Feldes mit dem Datentyp *Kurzer Text* oder *Zahl* haben Sie die Wahl zwischen *Textfeld*, *Kombinationsfeld* und *Listenfeld*. Für *Langer Text* und *OLE-Objekt* gibt es diese Eigenschaft nicht. Für *Ja/Nein*-Felder können Sie alternativ zu *Kontrollkästchen* auch *Textfeld* oder *Kombinationsfeld* auswählen.

Bei den Kontrollkästchen können Sie noch eine weitere Optimierung vornehmen: Standardmäßig werden die Bezeichnungsfelder von Kontrollkästchen nämlich rechts neben dem Kontrollkästchen angelegt und nicht wie bei Textfeldern, Kombinationsfeldern und so weiter rechts davon. Diese Optimierung können wir jedoch erst durch eine Formularvorlage vornehmen. Wie das gelingt, zeigen wir in »Formular-Vorlage« ab Seite 133.

3.3.10 Ergebnisse der Optimierung

Wenn Sie nun ein Formular in der Entwurfsansicht anlegen, seiner Eigenschaft *Datensatzquelle* die Tabelle mit den Beschriftungen, Beschreibungen und weiteren Anpassungen hinzufügen und dann die Felder der Datensatzquelle aus der Feldliste einfügen, erhalten Sie durch die obigen Maßnahmen ein anderes Ergebnis als ohne Festlegung dieser Eigenschaften. Schauen wir uns das doch einmal im Detail an.

Wenn Sie das Formular in der Entwurfsansicht öffnen, finden Sie zum Beispiel in der Eigenschaft *Statusleistentext* den als Beschreibung des Feldes im Tabellenentwurf angegebenen Text wie-

der. Außerdem wird, wie Sie am Feld *MehrwertsteuersatzID* sehen, der Text aus der Eigenschaft *Beschriftung* als Text des Bezeichnungsfeldes verwendet, hier *MwSt.-Satz*.

Abbildung 3.25: Übernahme der Beschreibung als Statusleistentext

Beim Feld *MehrwertsteuersatzID* hat Access, wie angekündigt, die Nachschlagefeld-Eigenschaften für das Kombinationsfeld übernommen (siehe Abbildung 3.26).

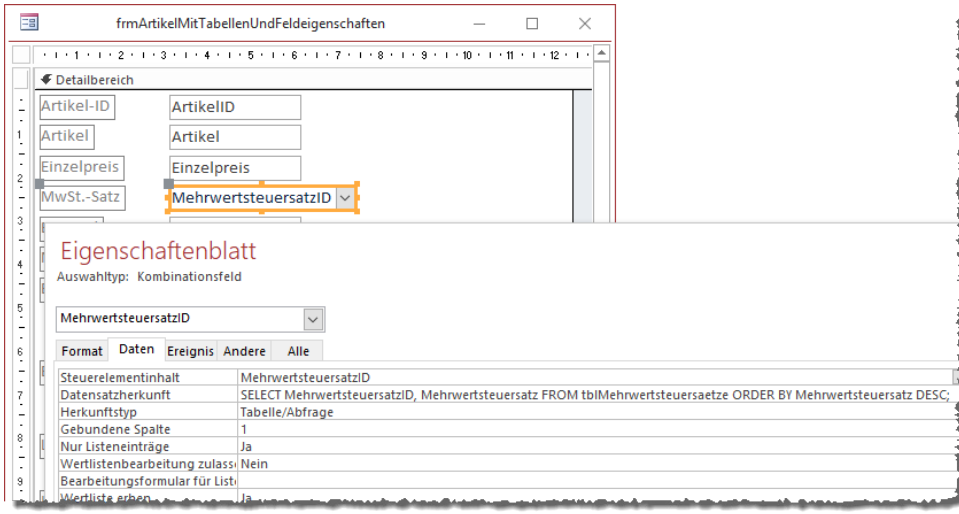


Abbildung 3.26: Kombinationsfeld-Einstellungen

Wenn Sie in die Formularansicht wechseln, finden Sie den ursprüngliche Beschreibungstext nach der Markierung eines der Steuerelemente in der Statusleiste vor (siehe Abbildung 3.27).

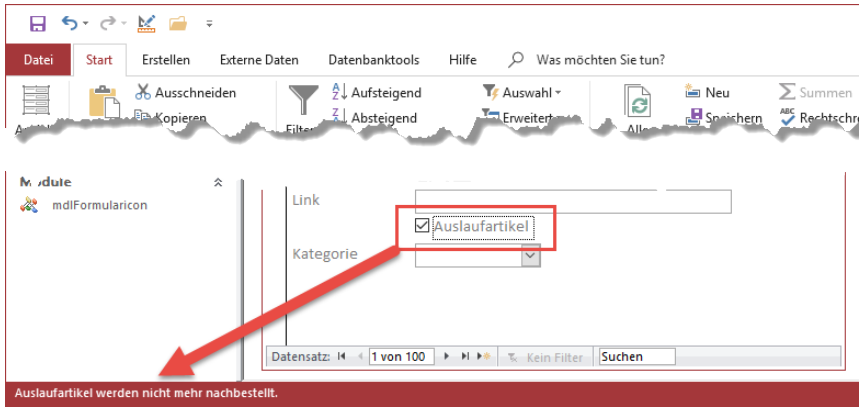


Abbildung 3.27: Feldbeschreibung als Statusleistentext

Noch toller wäre es, wenn der Text aus der Statusleiste auch gleich in die Eigenschaft *SteuerelementTip-Text* eingetragen würde – diesen würde der Benutzer vermutlich auch wesentlich schneller entdecken. In »Statusleistentext in ToolTip-Text umwandeln« ab Seite 513 zeigen wir, wie Sie mit einer einfachen VBA-Prozedur den Inhalt der Eigenschaft *Statusleistentext* für alle Steuerelemente *SteuerelementTip-Text* übertragen.

3.4 Steuerelemente positionieren und anpassen

Wenn Sie die gewünschten Steuerelemente zum Formularentwurf hinzugefügt haben, möchten Sie diese vermutlich schön ordentlich ausrichten und auf eine zu den Inhalten passende Größe bringen. Wir schauen uns erst einmal an, wie Sie grundsätzlich die Größe und die Position von Steuerelementen beeinflussen können. Danach gehen wir auf spezielle Möglichkeiten ein.

3.4.1 Größe und Position mit der Maus einstellen

Wenn Sie gerade ein frisches Textfeld über das Ribbon oder aus der Feldliste zum Formularentwurf hinzugefügt haben, sieht dieses wie in Abbildung 3.28 aus.

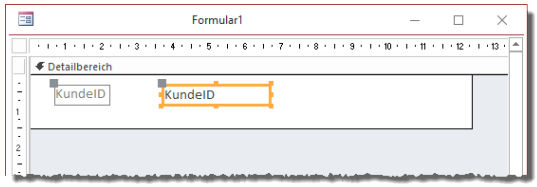


Abbildung 3.28: Ein frisch angelegtes Textfeld

Sie haben dann verschiedene Möglichkeiten, dieses Textfeld anzupassen. Die offensichtlichen sind die sichtbaren Varianten. Sie sehen am Bezeichnungsfeld ein graues Kästchen, das zum Ziehen mit der Maus einlädt. Ein solches graues Kästchen weist auch das Textfeld auf. Zusätzlich finden Sie hier noch acht Griffe in Rahmenfarbe, an denen Sie die Größe des Textfeldes ändern können. Sie können auch das Bezeichnungsfeld aktivieren – dann bleiben die grauen Kästchen links oben bei beiden Steuerelementen erhalten, aber die Markierung wechselt zum Bezeichnungsfeld (siehe Abbildung 3.29). Hier erkennen Sie auch schon, wie sich der Mauszeiger verändert, wenn Sie mit diesem über den grauen Kasten links oben über dem Steuerelement fahren. Dieses Symbol bedeutet, dass Sie nun bei gedrückter Maustaste das Steuerelement bewegen können.

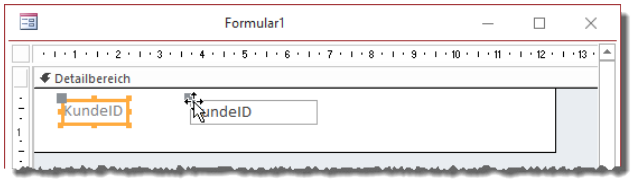


Abbildung 3.29: Wechsel der Markierung zum Bezeichnungsfeld

Das können Sie auch erreichen, wenn Sie mit dem Mauszeiger über den Rand des aktuell markierten Steuerelements fahren – aber nicht, wenn Sie den Mauszeiger über eines der übrigen Kästchen bewegen (siehe Abbildung 3.30).

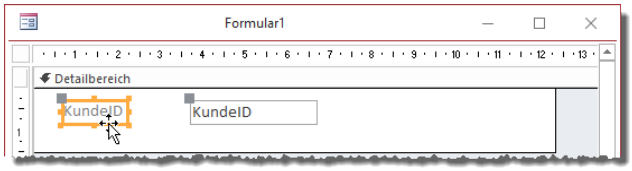


Abbildung 3.30: Platzieren des Mauszeigers zum Verschieben des Steuerelements

Es gibt jedoch einen gravierenden Unterschied, ob Sie ein Steuerelement am Rand greifen und es bewegen oder ob Sie das Kästchen oben links dazu verwenden:

» Am Rand greifen: Das Steuerelement bewegt sich samt dem verknüpften Steuerelement dorthin, wohin Sie es mit der Maus ziehen. Wenn Sie also das Textfeld markieren und am Rand mit der Maus greifen, um es nach unten zu ziehen, ziehen Sie das Bezeichnungsfeld mit nach unten.

» Am Kästchen oben links greifen: Das Steuerelement bewegt sich unabhängig von anderen Steuerelemente in die gewünschte Richtung. Die Verknüpfung etwa von Textfeld und Bezeichnungsfeld bleibt erhalten.

4 Gebundene Formulare

Grundsätzlich lässt sich ein Formular immer nur an eine Tabelle oder Abfrage (als gespeicherte Abfrage oder als SQL-Abfrage) binden. Es kann dann entweder in der Formularansicht einen Datensatz gleichzeitig anzeigen und Steuerelemente zum Navigieren zwischen den Datensätzen anbieten. Oder es zeigt mehrere Datensätze gleichzeitig in der Datenblattansicht oder der Endlosansicht an. Aber wie können wir es bei dieser Einschränkung von einer Tabelle oder Abfrage je Formular schaffen, Daten aus verknüpften Tabellen etwa aus einer 1:n-Beziehung oder einer m:n-Beziehung darzustellen? Ganz einfach: Die verknüpften Daten werden dann entweder in speziellen Steuerelementen wir dem Kombinationsfeld oder dem Listenfeld oder in einem Unterformular abgebildet. Die folgenden Abschnitte erläutern die gängigen Konstellationen für die üblichen Beziehungstypen wie 1:n, m:n oder 1:1.

4.1 Bindung an einfache Tabelle oder Abfrage

Um ein Formular an eine Tabelle oder Abfrage zu binden, stellen Sie die Eigenschaft *Datenherkunft* des Formulars auf einen der folgenden Werte ein:

» Name einer Tabelle

» Name einer Abfrage

» Speziell für dieses Formular zusammengestellte Abfrage

» SQL-Ausdruck

Wenn Sie eine Tabelle als Datenherkunft angeben, enthält die Datenherkunft des Formulars alle Datensätze der angegebenen Tabelle und Sie können beim Füllen der Steuerelemente auf alle Felder dieser Tabelle zugreifen. Wenn Sie eine Abfrage als Datenherkunft angeben, können Sie damit sowohl die Daten nur einer einzigen Tabelle, aber auch die Daten aus mehreren Tabellen berücksichtigen. Beim Zugriff nur auf die Daten einer Tabelle können Sie auch nur einige Felder dieser Tabelle abfragen. Oder Sie schränken einfach die Anzahl der anzuzeigenden Datensätze ein – zum Beispiel, indem Sie für ein Unterformular in der Datenblatt nur die Datensätze anzeigen, die einer im Hauptformular ausgewählten Kategorie entsprechen.

4.1.1 Bindung an gespeicherte Tabelle oder Abfrage

Die Bindung an eine bereits angelegte und gespeicherte Tabelle oder Abfrage (also eine, die im Navigationsbereich angezeigt wird) stellen Sie in der Regel über die Eigenschaft *Datensatzquelle* her. Diese zeigt im Auswahlfeld alle in der Datenbank vorhandenen Tabellen und Abfragen an (siehe Abbildung 4.1).

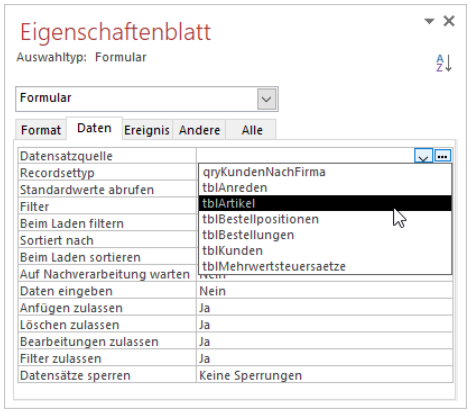


Abbildung 4.1: Auswahl der Datensatzquelle

Hier wählen Sie einen der Einträge aus, dessen Felder dann als Steuerelementinhalt für die verschiedenen Steuerelemente wie Textfeld, Kontrollkästchen, Kombinationsfeld, Listenfeld und so weiter genutzt werden können.

4.1.2 Bindung an komplett neue Abfrage

Sie können aber auch auf die Schaltfläche mit den drei Punkten (...) rechts im Eigenschaftenfeld *Datensatzquelle* klicken, ohne zuvor einen Eintrag ausgewählt zu haben. Dann erscheint eine neue, leere Abfrage mit der Überschrift *Abfragegenerator* (siehe Abbildung 4.2).

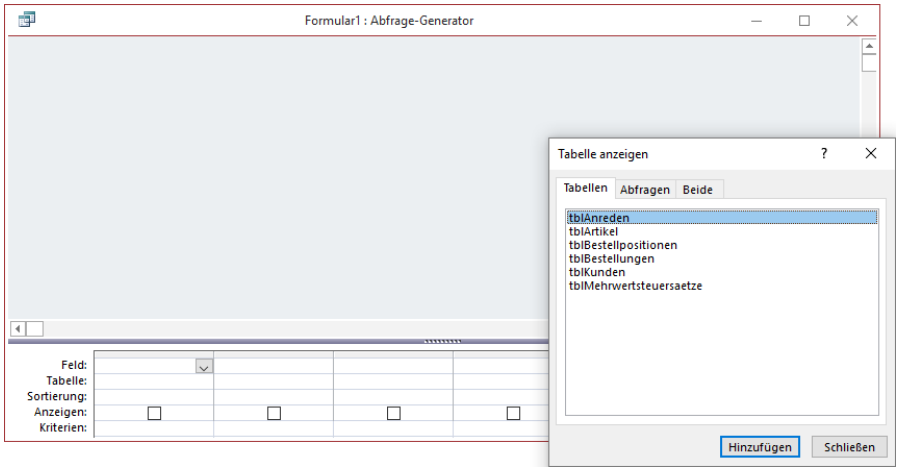


Abbildung 4.2: Neue, leere Abfrage als Datensatzquelle

Dies, weil die Abfrage nur temporär im Entwurf angezeigt wird, aber die Abfrage nicht als eigenes Objekt gespeichert wird. Wenn Sie die gewünschte Tabelle oder Abfrage hinzugefügt und die benötigten Felder in das Entwurfsraster gezogen haben und den Abfragegenerator dann schließen, erscheint die Meldung aus Abbildung 4.3. Mit dem Klick auf *OK* übernehmen Sie den Abfrageentwurf in die Eigenschaft *Datensatzquelle*.

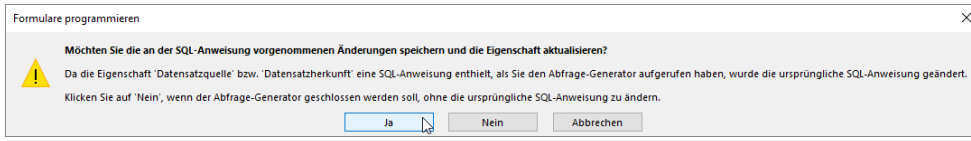


Abbildung 4.3: Meldung vor dem Schließen der als Datensatzquelle angelegten Abfrage

Nach dem Bestätigen der Meldung erscheint demzufolge nicht der Name einer der gespeicherten Tabellen oder Abfragen in der Eigenschaft *Datensatzquelle*, sondern der SQL-Ausdruck, welcher der zuvor zusammengestellten Abfrage entspricht (siehe Abbildung 4.4).

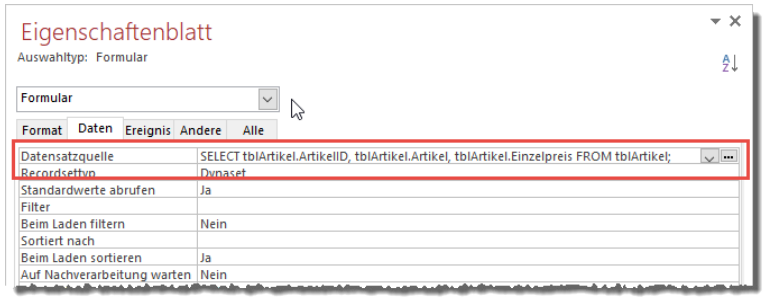


Abbildung 4.4: Die neue Datensatzquelle

Klicken Sie nun erneut auf die Schaltfläche mit den drei Punkten, wird die Abfrage wieder im Abfrage-Generator angezeigt. Nun schauen wir uns an, welche Optionen das Ribbon für diesen Fall bietet (siehe Abbildung 4.5).

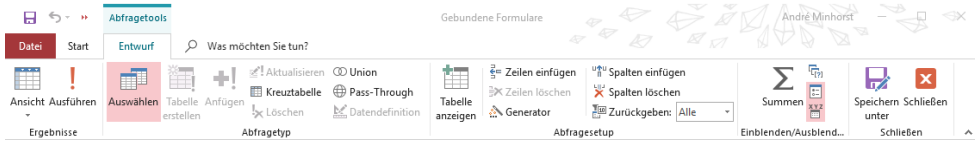


Abbildung 4.5: Ribbon für den Entwurf einer Abfrage im Abfrage-Generator

Hier finden wir am rechten Rand einen interessanten Befehl, nämlich *Entwurf|Schließen|Speichern unter*. Damit können Sie auch eine Abfrage, die Sie mit dem Abfrage-Generator eigentlich nur für die Datensatzquelle des Formulars definiert haben, auch als eigenes Objekt speichern.

Klicken Sie auf diese Schaltfläche, erscheint der Dialog aus Abbildung 4.6, mit dem Sie den Namen für die zu speichernde Abfrage angeben können.

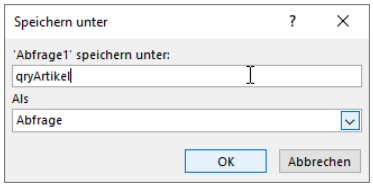


Abbildung 4.6: *Speichern unter*-Dialog

Vorsicht beim Ändern von als Objekt gespeicherten Abfragen

Wenn Sie eine als eigenes Objekt gespeicherte Abfrage als Datensatzquelle verwenden und dann auf die Schaltfläche mit den drei Punkten (...) klicken, gelangen Sie in den Entwurf dieser Abfrage. Wenn Sie diese ändern, werden die Änderungen in das Abfrage-Objekt übertragen. Manchmal verwendet man aber die gleiche gespeicherte Abfrage als Datensatzquelle für verschiedene Formulare oder Steuerelemente wie Kombinationsfelder oder Listenfelder. Die Änderungen wirken sich somit auch auf die anderen Objekte aus, welche diese Abfrage als Datensatzquelle verwenden. Sie sollten also im Zweifel eher eine neue Abfrage auf Basis der vorhandenen Abfrage erstellen.

4.1.3 Auswahl der Datensatzquelle über die Feldliste

Früher zeigte die Feldliste ausschließlich die Felder der Datensatzquelle des Objekts an. Heute zeigt die Feldliste aber den unscheinbaren Eintrag namens *Alle Tabellen anzeigen* an – egal, ob Sie bereits eine Datensatzquelle ausgewählt haben oder nicht (siehe Abbildung 4.7).

Abbildung 4.7: Feldliste ohne vorausgewählte Datensatzquelle

Klicken Sie auf diesen Eintrag, erscheint eine Liste aller Tabellen der Datenbank. Sie können den Eintrag zu jeder einzelnen Tabelle per Klick auf das Plus-Zeichen erweitern und so die Felder dieser Tabelle anzeigen (siehe Abbildung 4.8).

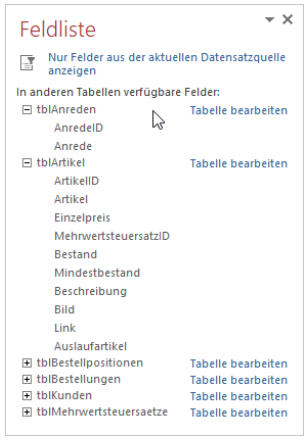


Abbildung 4.8: Auswahl von Feldern aus allen Tabellen

Wenn wir hier jetzt beispielsweise das Feld *ArtikelID* aus der Tabelle *tblArtikel* in das Formular ziehen, erscheint nicht nur dieses Feld im Formularentwurf, sondern es wird auch die Eigenschaft *Datensatzquelle* des Formulars auf diese Tabelle eingestellt. Ziehen wir noch die Felder *Artikel* und *Einzelpreis* in den Detailbereich, wird die Datensatzquelle etwa auf die folgende Abfrage eingestellt:

```
SELECT tblArtikel.ArtikelID, tblArtikel.Artikel, tblArtikel.Einzelpreis
```

Ziehen wir nun etwa das Feld *Mehrwertsteuersatz* aus der Tabelle *tblMehrwertsteuersaetze* in den Formularentwurf, um den entsprechenden Wert aus dieser Tabelle für den aktuell angezeigten Artikel hinzuzufügen, gelingt das nicht ohne Weiteres. Es erscheint nämlich der Dialog aus Abbildung 4.9, der nach der Angabe einer Beziehung zwischen den Tabellen mit den hinzugefügten Feldern verlangt.

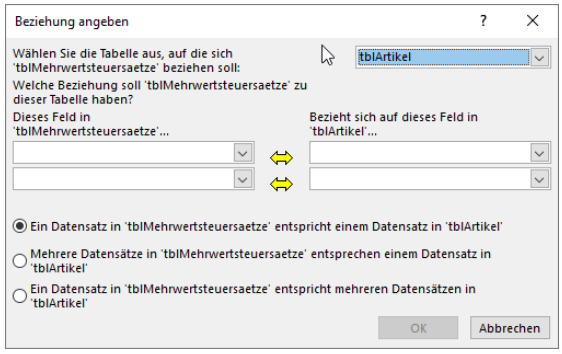


Abbildung 4.9: Hinzufügen der Beziehung zwischen den Tabellen, deren Felder zum Formular hinzugefügt wurden

Man hätte erwarten können, dass Access versucht, die Beziehungen automatisch zu erkennen, aber das ist nicht der Fall. Also tragen wir die Beziehungseigenschaften wie in Abbildung 4.10 noch ein. Neben dem Primärschlüsselfeld der Tabelle *tblMehrwertsteuersaetze* und dem Fremdschlüsselfeld der Tabelle *tblArtikel* stellen wir auch noch ein, dass ein Datensatz der Tabelle *tblMehrwertsteuersaetze* mehreren Datensätzen der Tabelle *tblArtikel* zugeordnet werden kann.

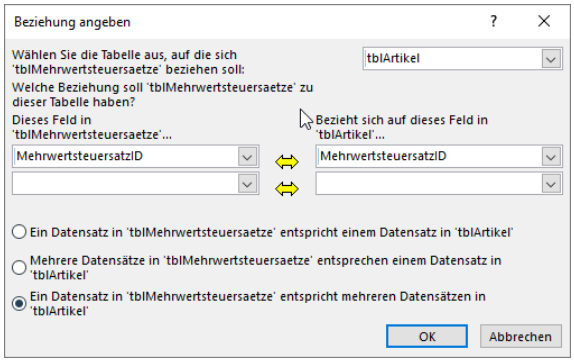


Abbildung 4.10: Nachträgliches Einfügen der Beziehung für die beteiligten Tabellen

Wenn wir den Dialog schließen und einen erneuten Blick auf die Feldliste werfen, finden wir dort nun im oberen Bereich die beiden beteiligten Tabellen. Die nicht beteiligten Tabellen werden im unteren Bereich der Feldliste abgebildet. Schauen wir auf die Eigenschaft *Datensatzquelle* des Formulars, erhalten wir die folgende Abfrage als Inhalt:

```
SELECT tblArtikel.ArtikelID, tblArtikel.Artikel, tblArtikel.Einzelpreis, tblMehrwert-
steuersaetze.Mehrwertsteuersatz
FROM tblMehrwertsteuersaetze INNER JOIN tblArtikel ON (tblMehrwertsteuersaetze.
MehrwertsteuersatzID = tblArtikel.MehrwertsteuersatzID);
```

Die Feldliste sieht nun wie in Abbildung 4.11 aus. Unter *Für diese Ansicht verfügbare Felder:* befinden sich nun Felder aus den beiden Tabellen *tblArtikel* und *tblMehrwertsteuersaetze*.

Wer nicht so oft mit SQL-Code arbeitet, der kann sich die automatisch über die Feldliste erstellte Abfrage natürlich auch wieder in der Abfrage-Entwurfsansicht anschauen. Dazu wechseln wir zurück zum Eigenschaftenblatt, und zwar am schnellsten durch einen Doppelklick auf den grauen, leeren Bereich im Formularentwurf.

Dort wechseln wir zur Registerseite *Daten* und klicken auf die Eigenschaft *Datensatzherkunft*. Ein weiterer Klick auf die nun erscheinende Schaltfläche mit den drei Punkten liefert uns die Entwurfsansicht der von uns zusammengestellten Abfrage (siehe Abbildung 4.12). Und das Ergebnis sieht genau wie gewünscht aus – es enthält die beiden verknüpften Tabellen sowie die Felder, die wir aus diesen beiden Tabellen ausgewählt haben.

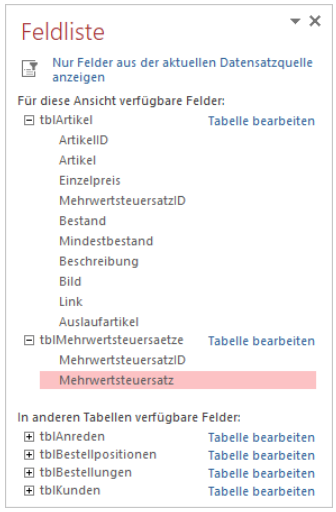


Abbildung 4.11: Die beteiligten Tabellen erscheinen nun im oberen Bereich der Feldliste.

Die Feldliste ist also ein weiteres interessantes Tool gerade für Einsteiger, um die Informationen für die Datensatzquelle eines Formulars zusammenzustellen.

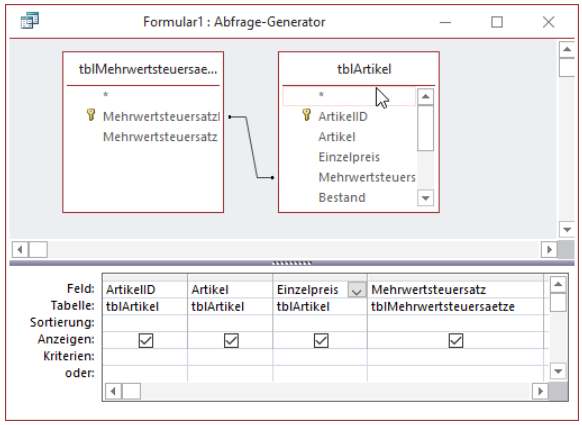


Abbildung 4.12: Entwurf der per Feldliste zusammengestellten Abfrage

4.1.4 Einfache Tabelle oder Abfrage in der Formularansicht

Im ersten Teil dieses Kapitels haben wir uns angesehen, wie Sie die Tabelle oder Abfrage für das Formular verfügbar machen. Ein Ausdruck davon ist, dass die Felder in der Feldliste erscheinen. Aus der Feldliste können Sie die Felder dann einfach in den Detailbereich des Formularentwurfs ziehen.

Wenn Sie nach dem Binden des Formular an eine Tabelle oder Abfrage die Felder dieser Datensatzquelle, deren Daten Sie im Formular anzeigen wollen, etwa aus der Feldliste in den Detailbereich des Formularentwurfs ziehen, zeigen die daraus resultierenden Steuerelemente, in der Regel Textfelder, Kombinationsfelder oder Kontrollkästchen, dann jeweils den Inhalt des daran gebundenen Feldes der Datensatzquelle an.

Das schauen wir uns nun im Detail an. Dazu legen Sie ein neues, leeres Formular in der Entwurfsansicht an und speichern dieses unter dem Namen *frmKundendetails*. Danach stellen wir die Eigenschaft *Datensatzquelle* auf den Wert *tblKunden* ein.

Wählen Sie dann den Ribbon-Befehl *Entwurf|Tools|Vorhandene Felder* hinzufügen, sodass die Feldliste erscheint (siehe Abbildung 4.13).

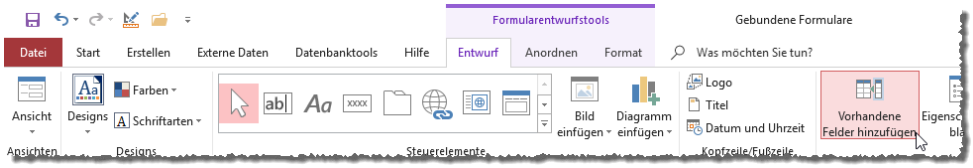


Abbildung 4.13: Schaltfläche zum Einblenden der Feldliste

Diese sollte sich dann neben dem Formular befinden, sodass Sie die Einträge aus der Feldliste direkt in das Formular ziehen können. Dann markieren Sie alle Einträge, indem Sie beispielsweise einen Eintrag anklicken und dann die Tastenkombination *Strg + A* betätigen. Anschließend ziehen Sie alle Einträge per Drag and Drop in den Detailbereich des Formularentwurfs (siehe Abbildung 4.14).

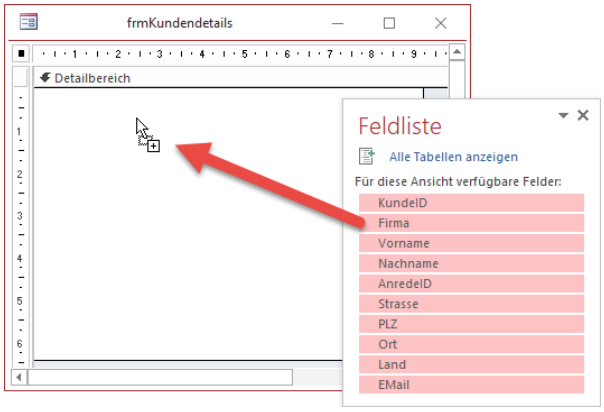


Abbildung 4.14: Ziehen der Einträge der Feldliste in den Formularentwurf

Nach dem Fallenlassen der Elemente im Formularentwurfs sieht das Ergebnis wie in Abbildung 4.15 aus. Hier können wir noch kleinere Anpassungen vornehmen, beispielsweise das Ändern der Texte einiger Bezeichnungsfelder (*KundeID* in *Kunde-ID*, *AnredeID* in *Anrede*, *Strasse* in *Straße* und *EMail* in *E-Mail*). Außerdem macht es Sinn, die Breite der Textfelder anzupassen, also etwa *KundeID*, *AnredeID* und *PLZ* etwas schmaler zu machen und die übrigen Textfelder etwas breiter.

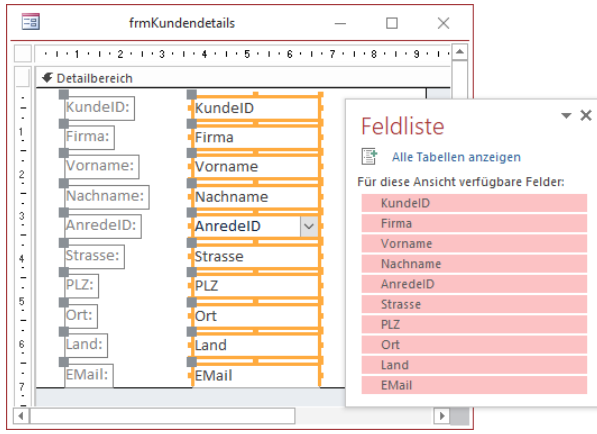


Abbildung 4.15: Die Einträge der Feldliste werden zu gebundenen Steuerelementen mit Bezeichnungsfeldern

Wenn Sie das erledigt haben, können Sie endlich in die Formularansicht wechseln (am schnellsten mit der Tastenkombination *Strg + .*). Das Ergebnis sieht dann wie in Abbildung 4.16 aus.

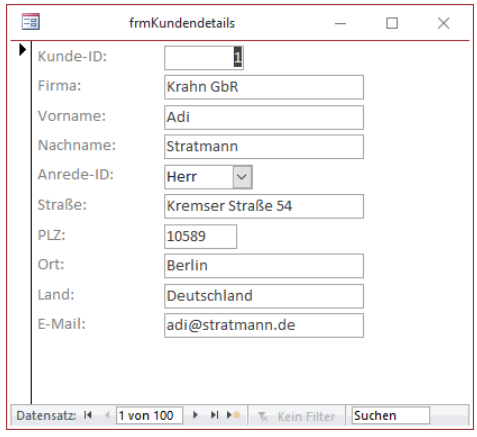


Abbildung 4.16: Das Formular in der Formularansicht

Wenn Sie ein solches Formular erstellt haben, finden Sie dort immer die üblichen Elemente eines Formulars vor – also Datensatzmarkierer, Navigationsschaltflächen und Bildlaufleisten. Es gibt sicher Formulare, bei denen Sie dem Benutzer die Möglichkeit bieten wollen, in den Datensätzen zu navigieren und vor- und zurückzublättern.

Ab einer bestimmten Menge von Datensätzen macht das aber keinen Sinn mehr, denn dann würde es zu lange dauern, einen bestimmten Datensatz zu finden. Und wenn man es eingehender betrachtet, benötigt man in einem Formular, das einen einzelnen Datensatz anzeigt, überhaupt Navigationsschaltflächen, Datensatzmarkierer und Bildlaufleisten? Wir meinen nicht.

Damit überfordern Sie den Benutzer dann auch nicht mit lauter Elementen, die er gar nicht benötigt. Und Möglichkeiten, zum gewünschten Datensatz zu steuern, gibt es genügend – zum Beispiel Formulare in der Datenblattansicht oder Listenfelder oder auch eine der beiden Varianten in Kombination mit einen oder mehreren Suchfeldern.

Also wechseln wir nun einmal zurück in die Entwurfsansicht und stellen einige Eigenschaften auf der Registerseite *Format* im Eigenschaftenblatt um. Die Eigenschaft *Automatisch zentrieren* erhält den Wert *Ja*, damit das Formular automatisch in der Mitte des Access-Fensters geöffnet wird. Die Eigenschaften *Datensatzmarkierer*, *Navigationsschaltflächen* und *Bildlaufleisten* stellen wir auf *Nein* ein, damit diese Elemente wegfallen (siehe Abbildung 4.17).

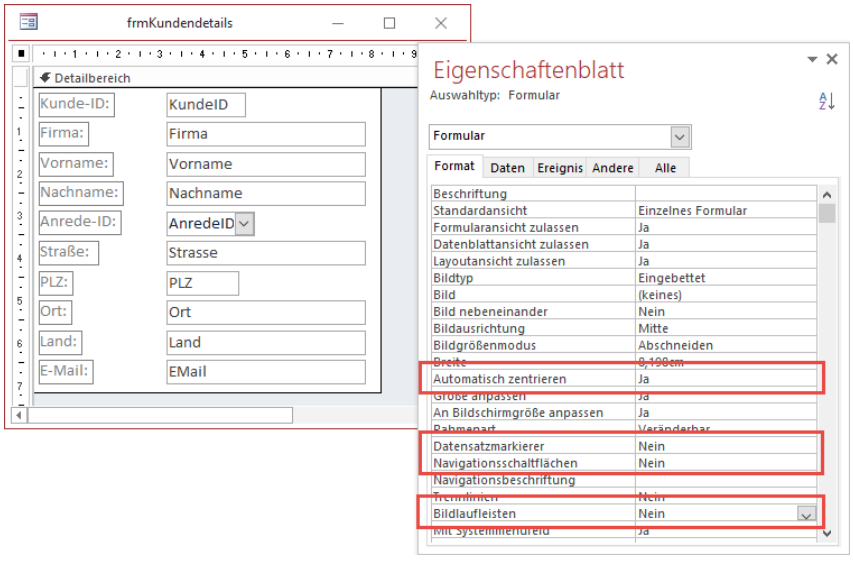


Abbildung 4.17: Einstellen wichtiger Eigenschaften des Formulars

Wechseln Sie nun zurück in die Formularansicht, sieht das Formular schon viel übersichtlicher aus. Hier kann der Kunde sich nun auf die Eingabe oder das Bearbeiten des angezeigten

Datensatzes konzentrieren und wird nicht von Bildlaufleisten, Datensatzmarkierer und Navigationsschaltflächen abgelenkt (siehe Abbildung 4.18).

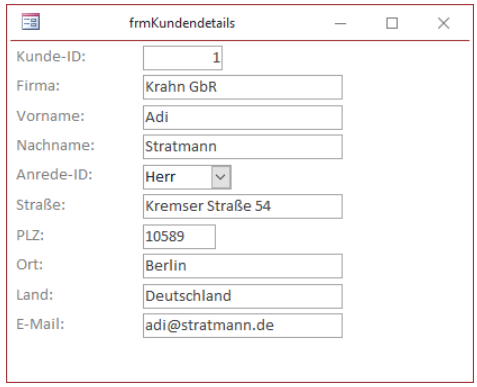


Abbildung 4.18: Formular mit angepassten Eigenschaften

In diesem Formular können Sie nun, auch wenn die Steuerelemente zur Navigation nicht mehr sichtbar sind, immer noch zum nächsten und zum vorherigen Datensatz wechseln. Dazu bewegen Sie die Einfügemarke einfach zum letzten Element in der Aktivierreihenfolge (mehr dazu siehe »Aktivierreihenfolge« ab Seite 81) verschieben und dann nochmals die Tabulator-Taste betätigen. Dann springt das Formular zum nächsten Datensatz oder, wenn es bereits den letzten Datensatz anzeigt, zu einem neuen, leeren Datensatz.

Wenn Sie dies verhindern wollen, können Sie einfach die Eigenschaft *Zyklus* auf *Aktueller Datensatz* einstellen. Beim Betätigen der Tabulator-Taste für das letzte Steuerelement der Aktivierreihenfolge springt der Fokus einfach wieder zum ersten Steuerelement der Aktivierreihenfolge.

4.1.5 Einfache Tabelle oder Abfrage in der Datenblattansicht

Der Unterschied zwischen dem Formular in der Formularansicht, wie wir es in den vorherigen Abschnitten erstellt haben, ist der Wert genau einer Eigenschaft – nämlich der Eigenschaft *Standardansicht*. Der lautete im Beispiel von oben *Einzelnes Formular* und wird für die Datenblattansicht auf *Datenblatt* eingestellt.

Wir kopieren das Formular von oben, indem wir es schließen, im Navigationsbereich markieren, die Tastenkombinationen *Strg + V* und *Strg + C* betätigen und im nun erscheinenden Fenster den Namen des neuen Formulars eingeben, und zwar *frmKundenDatenblatt*. Öffnen Sie das Formular in der Entwurfsansicht und stellen Sie die Eigenschaft *Standardansicht* auf *Datenblatt* ein (siehe Abbildung 4.19).

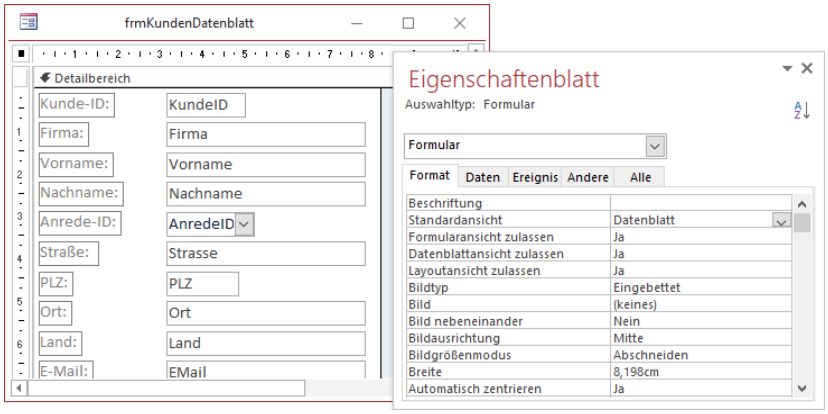


Abbildung 4.19: Einstellen der Standardansicht

Schließen Sie das Formular nun und öffnen es erneut, erscheint es ohne weitere Änderungen in der Datenblattansicht. Als Spaltenüberschriften verwendet Access die Texte der Bezeichnungsfelder aus dem Formularentwurf, allerdings ohne Doppelpunkt (siehe Abbildung 4.20).

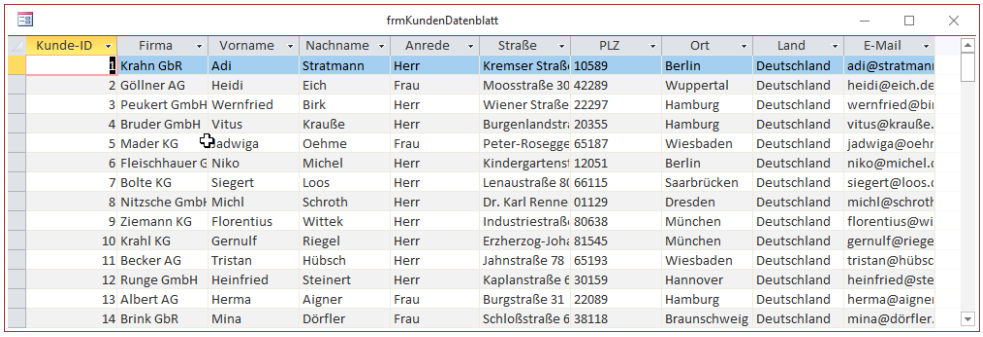


Abbildung 4.20: Das Formular in der Datenblattansicht

Zusammengefasst können wir festhalten: Die Datenblattansicht zeigt für alle gebundenen Steuerelemente der Entwurfsansicht jeweils eine Spalte an, wobei die Spaltenüberschrift aus dem Text des entsprechenden Bezeichnungsfeldes übernommen wird. Es bietet eine Menge interessanter Features wie die Möglichkeit zum Sortieren oder Filtern – diese sehen wir uns im Detail unter »Die Datenblattansicht« ab Seite 187 an.

Die Datenblattansicht würde sich beispielsweise vorzüglich eignen, um eine Übersicht der Datensätze einer Tabelle zu liefern und dem Benutzer dann etwa durch einen Doppelklick auf einen der Einträge die Anzeige in einem Detailformular zu ermöglichen. Wie das gelingt, zeigen wir später unter »Übersichtsformular mit Detailformular« ab Seite 467.

4.1.6 Einfache Tabelle oder Abfrage in der Endlosansicht

Die Endlosansicht ist eine Art Hybrid zwischen der Formularansicht und der Datenblattansicht. Sie erlaubt im Gegensatz zur Datenblattansicht das freie Anordnen der Steuerelemente wie in der Formularansicht.

Anders als in der Formularansicht können dabei jedoch auch mehrere Datensätze untereinander angezeigt werden. Auf diese Weise können Sie dann auch beispielsweise eine oder mehrere Schaltflächen zu jedem Datensatz hinzufügen, über die der Benutzer etwa die Detailansicht eines Datensatzes öffnen oder diesen löschen kann.

Details zur Endlosansicht finden Sie unter »Die Endlosansicht« ab Seite 209.

4.2 Geteilte Formulare

Geteilte Formulare sind eine Sonderform von Formularen, die Sie per Assistent herstellen können, aber deren Eigenschaften Sie auch individuell einstellen können.

Die Besonderheit dieses Formulartyps ist, dass er die Daten der gleichen Datensatzquelle prinzipiell gleich zweimal im gleichen Formular anzeigt – einmal in der Datenblattansicht und einmal in der Formularansicht.

Die Formularansicht zeigt dabei die Details des aktuell in der Datenblattansicht ausgewählten Datensatzes an. Wenn Sie über die Navigationsschaltflächen den Datensatz wechseln, wird in die Detailansicht auf den neuen markierten Datensatz eingestellt und in der Datenblattansicht verschiebt sich der Datensatzzeiger ebenfalls auf diesen Datensatz.

Wenn Sie ein geteiltes Formular über den Assistenten anlegen wollen, verwenden Sie den Ribbon-Eintrag *Erstellen|Formulare|Weitere Formulare|Geteiltes Formular*. Das Ergebnis sieht wie in Abbildung 4.21 aus.

Wie Sie hier erkennen, enthält das Formular nur eine Leiste mit Navigationsschaltflächen, obwohl es scheinbar zwei Formulare enthält. Diese Navigationsschaltflächen steuern beide Ansichten synchron.

Wenn Sie versuchen, etwa den oberen Teil über das Kontextmenü ebenfalls in der Datenblattansicht anzuzeigen, schlägt dies fehl – es gibt nur die drei Ansichten *Formularansicht*, *Layoutansicht* und *Entwurfsansicht* (siehe Abbildung 4.22).

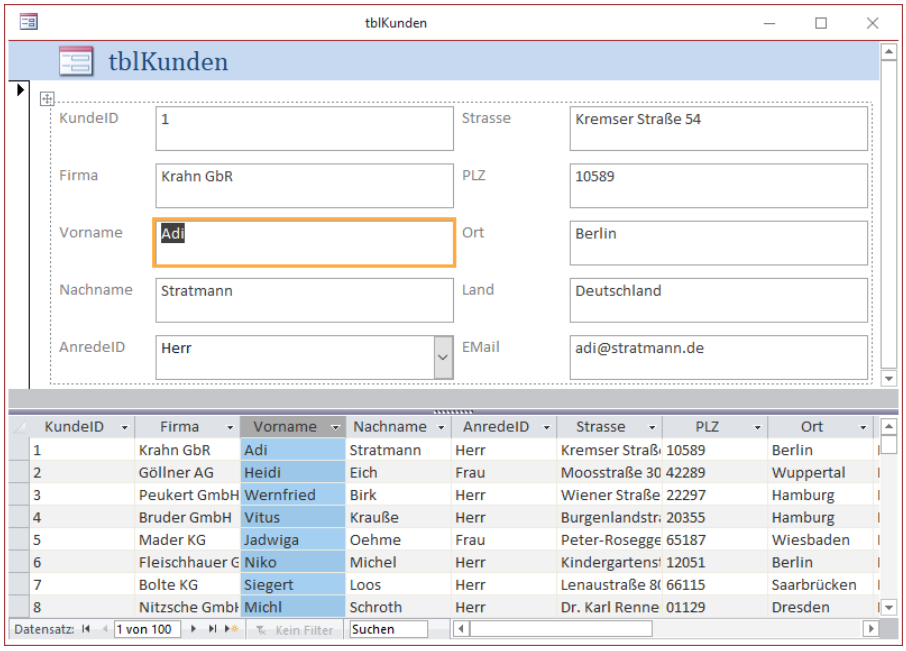


Abbildung 4.21: Formular als geteiltes Formular

Abbildung 4.22: Die für das geteilte Formular verfügbaren Ansichten

Wenn wir hier in die Entwurfsansicht wechseln und das Eigenschaftenblatt aktivieren, finden wir auf der Registerseite *Format* einige Eigenschaften speziell für die geteilte Ansicht vor (siehe Abbildung 4.23).

Weiter oben auf dieser Seite des Eigenschaftenblatts fällt außerdem noch auf, dass die Eigenschaft *Standardansicht* auf den Wert *Geteiltes Formular* eingestellt ist. Stellen Sie diese auf *Einfaches Formular* ein, verschwindet der Datenblattansicht-Teil des Formulars wieder. Sie können also auch herkömmliche Formulare durch Einstellen der Eigenschaft *Standardansicht* auf *Geteiltes Formular* in ein geteiltes Formular umwandeln.

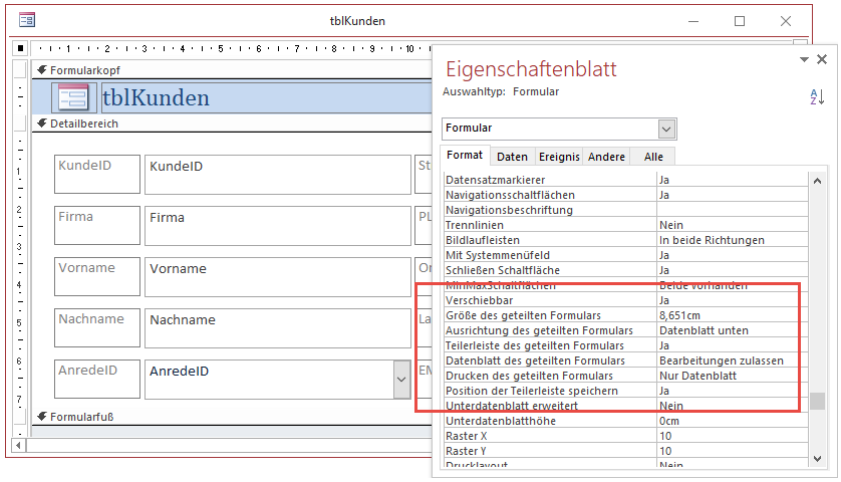


Abbildung 4.23: Eigenschaften für das geteilte Formular

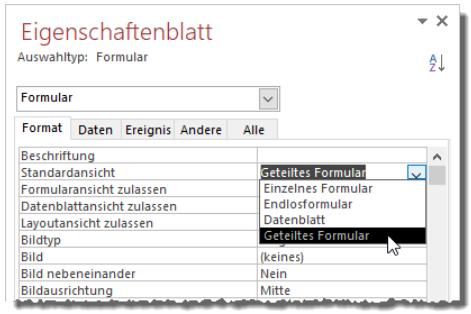


Abbildung 4.24: Die Standardansicht lautet *Geteiltes Formular*.

4.2.1 Eigenschaften für das geteilte Formular

Für das geteilte Formular können wir die folgenden Eigenschaften verwenden:

» *Größe des geteilten Formulars* (VBA: *SplitFormSize*): Diese Eigenschaft legt fest, welche Höhe der Teil des Formulars in der Formularansicht aufweist.

» *Ausrichtung des geteilten Formulars* (VBA: *SplitFormOrientation*): Diese Eigenschaft gibt die Ausrichtung an. Wenn man sich die VBA-Konstanten für die Eigenschaft *SplitFormOrientation* ansieht, findet man heraus, dass es sich hierbei um die Ausrichtung des Datenblatt-Teils des geteilten Formulars handelt. Es gibt die Werte *acDataSheetOnBottom*, *acDataSheetOnLeft*, *acDataSheetOnRight* und *acDataSheetOnTop*. Diese Eigenschaft lässt sich übrigens nur in der Entwurfsansicht einstellen, also nicht dynamisch zur Laufzeit.

» *Teilerleiste des geteilten Formulars* (VBA: *SplitFormSplitterBar*): Mit dieser Eigenschaft geben Sie an, ob die Teilerleiste, mit der Sie das Verhältnis der beiden Bereiche mit der Maus einstellen können, angezeigt werden soll oder nicht.

» *Datenblatt des geteilten Formulars* (VBA: *SplitFormDatasheet*): Diese Eigenschaft hat einen etwas ungünstig gewählten Namen, denn eigentlich geben Sie damit an, ob der Datenblatt-Teil der geteilten Ansicht schreibgeschützt dargestellt werden soll oder nicht.

» *Drucken des geteilten Formulars* (VBA: *SplitFormPrinting*): Normalerweise sollte Sie Berichte zum Drucken von Daten verwenden. Wenn Sie allerdings ein geteiltes Formular ausdrucken wollen, legt diese Eigenschaft mit den beiden Werten *Nur Formular* oder *Nur Datenblatt* fest, welcher der beiden Teile gedruckt wird.

» *Position der Teilerleiste speichern* (VBA: *SplitFormSplitterBarSave*): Diese Eigenschaft legt fest, ob die Position der Teilerleiste, sofern diese überhaupt angezeigt wird, nach dem Schließen des Formulars gespeichert werden soll, damit diese beim nächsten Öffnen wiederhergestellt werden kann.

4.2.2 Beispiele für den Einsatz der geteilten Formulare

Ein schönes Beispiel für den Einsatz dieser Ansicht ist die mit der Einstellung des Wertes *Datenblatt links* für die Eigenschaft *Ausrichtung des geteilten Formulars* (siehe Abbildung 4.25). Mit dieser Ansicht können Sie schön durch die vorhandenen Datensätze scrollen und nach dem Auffinden den gewünschten Datensatz anklicken, dessen Details dann in der Formularansicht auf der rechten Seite angezeigt werden.

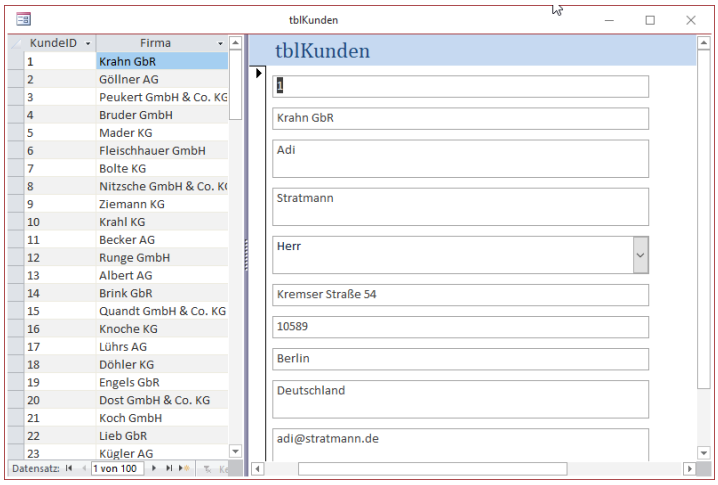


Abbildung 4.25: Praktische Ansicht mit dem Datenblatt als Auswahlwerkzeug

5 Gebundene Steuerelemente

Im vorherigen Kapitel haben wir erläutert, wie Formulare an eine Datensatzquelle gebunden werden. Dabei haben wir schon die wichtigste Möglichkeit der Bindung von Steuerelementen kennengelernt – nämlich die Bindung von Textfeldern, Kombinationsfeldern und Kontrollkästchen an die Felder der Datensatzquelle eines Formulars.

Wir sollten allerdings nicht vergessen, dass es noch eine ganze Menge weiterer Bindungsmöglichkeiten gibt, die wir der Eigenschaft *Steuerelementinhalt* zuweisen können.

5.1 Steuerelemente mit Steuerelementinhalt

Zunächst schauen wir uns an, welche Steuerelemente überhaupt die Eigenschaft *Steuerelementinhalt* (unter VBA: *ControlSource*) aufweisen und die wir so an ein Feld oder an einen Ausdruck binden können. Dabei handelt es sich um die folgenden Steuerelemente:

- Gebundenes Objektfeld (*BoundObjectFrame*)
- Anlage (*Attachment*)
- Kontrollkästchen (*CheckBox*)
- Kombinationsfeld (*ComboBox*)
- Custom Control (*CustomControl*)
- Listenfeld (*ListBox*)
- Optionsfeld (*OptionButton*)
- Optionsgruppe (*OptionGroup*)
- Textfeld (*TextBox*)
- Umschaltfläche (*ToggleButton*)
- Webbrowsersteuerelement (*WebBrowserControl*)
- BildSteuerelement (*Image*)

Hier gibt es kleine Besonderheiten: Das Custom Control ist ein Container etwa für ActiveX-Steuerelemente. Wenn Sie hier die Eigenschaft *Steuerelementinhalt* einstellen, wirkt sich dies etwa für die ebenfalls in diesem Buch beschriebenen Steuerelemente *ImageList*, *TreeView* und *ListView* nicht aus. Und das Bildsteuerelement weist die Eigenschaft *Steuerelementinhalt* nur in den Eigenschaften im Formularentwurf auf, aber nicht unter VBA.

Wenn Sie zu einem bestimmten Zweck einmal selbst herausfinden wollen, welche Steuerelemente (oder Klassen im Allgemeinen) eine bestimmte Eigenschaft aufweisen, können Sie diese Eigenschaft im Suchfeld des Objektkatalogs im VBA-Editor eingeben und erhalten etwa das Ergebnis aus Abbildung 5.1.

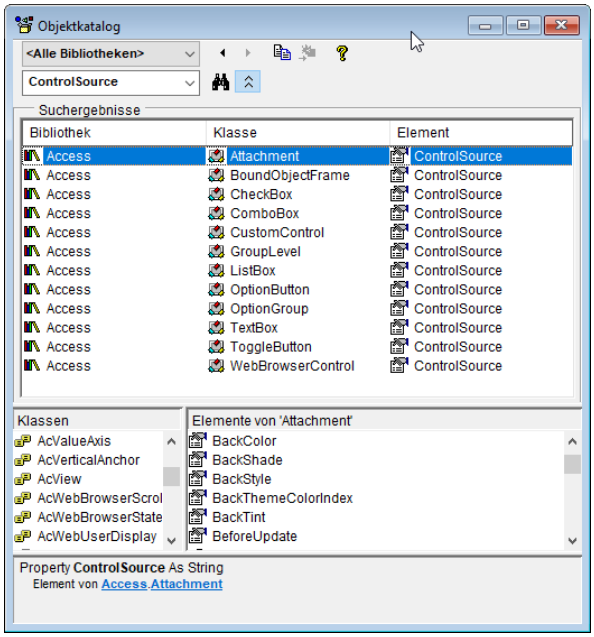


Abbildung 5.1: Steuerelemente mit der Eigenschaft *ControlSource* beziehungsweise *Steuerelementinhalt*

5.2 Beispiele für gebundene Felder

Die folgenden Abschnitte stellen die verschiedenen Möglichkeiten für das Anzeigen von Werten über die Eigenschaft *Steuerelementinhalt* vor.

5.2.1 Werte als Literal oder Standardwert vorgeben

Die einfachste Art, der Eigenschaft *Steuerelementinhalt* eines Steuerelements einen Wert zuzuweisen, ist das Eintragen dieses Wertes in die Eigenschaft *Steuerelementinhalt* – und zwar mit vorangestelltem Gleichheitszeichen und – bei einem String-Literal – mit Anführungszeichen (siehe Abbildung 5.2):

```
="Literal"
```

Sie können auch eine Zahl angeben, die Sie dann allerdings nicht in Anführungszeichen einschließen müssen:

```
=123
```

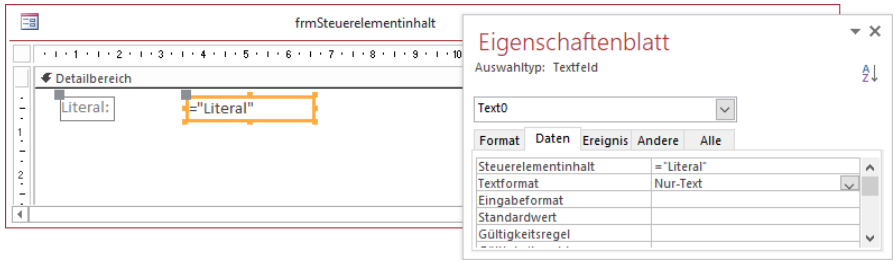


Abbildung 5.2: Einstellen eines Literals als Steuerelementinhalt

Wenn Sie ein solches Formular in der Formularansicht öffnen und versuchen, den enthaltenen Text zu ändern, gelingt dies nicht – siehe Abbildung 5.3. Da wir einen festen Ausdruck zugewiesen haben, ist eine Änderung nicht möglich.

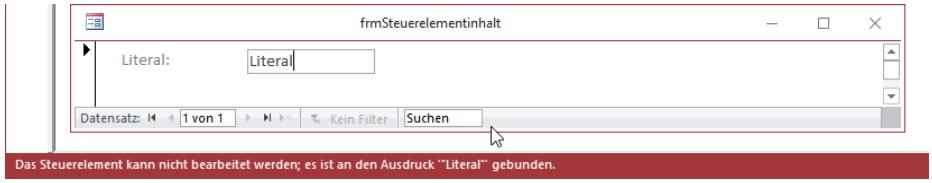


Abbildung 5.3: Feste Ausdrücke in gebundenen Steuerelementen können nicht geändert werden.

Solche Steuerelemente, die Ausdrücke enthalten, die nicht durch den Benutzer geändert werden können, sollten Sie direkt sperren. Dazu stellen Sie die Eigenschaft *Aktiviert* auf *Nein* und *Gesperrt* auf *Ja* ein. Dadurch erscheint das Steuerelement genau wie das aktivierte und nicht gesperrte Steuerelement, aber kann nicht mehr durch den Benutzer bearbeitet werden.

Wert vorgeben

Wenn Sie ein Steuerelement mit einem Wert vorbelegen wollen, den der Benutzer anschließend ändern können soll, können Sie dies auf zwei Arten tun:

» Sie stellen die Eigenschaft *Standardwert* auf diesen Wert ein oder

» Sie legen den Wert beim Öffnen des Formulars per VBA fest (mehr dazu unter »Standardwert festlegen« ab Seite 265).

Wenn Sie die Eigenschaft *Standardwert* auf die Zeichenkette *Standardwert* einstellen, wird dieser genau wie das Literal im Textfeld angezeigt. Dieser wird dann, wenn es sich um ein an

ein Feld der Datensatzquelle gebundenes Steuerelement handelt, übernommen, wenn der Benutzer den Datensatz bearbeitet und speichert.

5.2.2 An VBA-Funktionen binden

Die Programmiersprache bietet eine Reihe von Funktionen an, die Sie als Wert der Eigenschaft *Steuerelementinhalt* eines Steuerelements festlegen können. Die Steuerelemente zeigen das Ergebnis dieser Funktionen dann an. Eines der bekanntesten Beispiele dürfte die Anzeige des aktuellen Datums, der aktuellen Uhrzeit oder auch von beidem sein. Dazu hinterlegen Sie für das Textfeld etwa Ausdrücke wie die folgenden:

```
=Datum()
=Zeit()
=Jetzt()
```

Das Ergebnis für die drei Steuerelemente *txtDatum*, *txtUhrzeit* und *txtDatumUndZeit* zeigt Abbildung 5.4.

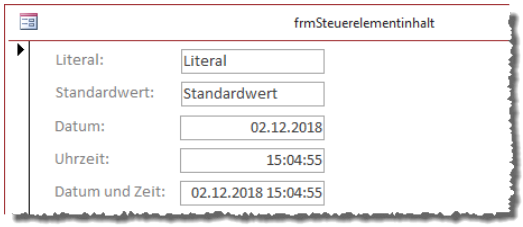


Abbildung 5.4: Datum, Uhrzeit und beides

Wie Sie sehen, werden die VBA-Funktionen hier nicht in der englischen, sondern in der deutschen Schreibweise abgebildet. Woher wissen wir, wie die Funktionen auf Deutsch benannt werden – und welche Funktionen überhaupt zur Verfügung stehen?

Dazu können Sie den Ausdrucks-Generator nutzen, den Access anzeigt, wenn Sie etwa auf die Schaltfläche mit den drei Punkten (...) rechts im Feld der Eigenschaft *Steuerelementinhalt* des Steuerelements klicken.

Hier finden im linken Listenfeld den Bereich *Funktionen|Integrierte Funktionen*. Nach der Auswahl zeigt das mittlere Listenfeld die Kategorien an. Klicken Sie etwa auf die Kategorie *Datum/Uhrzeit*, finden Sie im rechten Listenfeld die passenden Funktionen, zum Beispiel die oben verwendete Funktion *Jetzt*. Ein Doppelklick auf diesen Eintrag fügt die Funktion dann zum Textfeld im oberen Bereich des Ausdrucks-Generators hinzu, dessen Inhalt nach einem Mausklick auf die Schaltfläche *OK* in die Eigenschaft übernommen wird, von der Sie den Dialog aufgerufen haben (siehe Abbildung 5.5).

Access fügt dann sogar noch das führende Gleichheitszeichen zum neuen Wert der Eigenschaft hinzu.

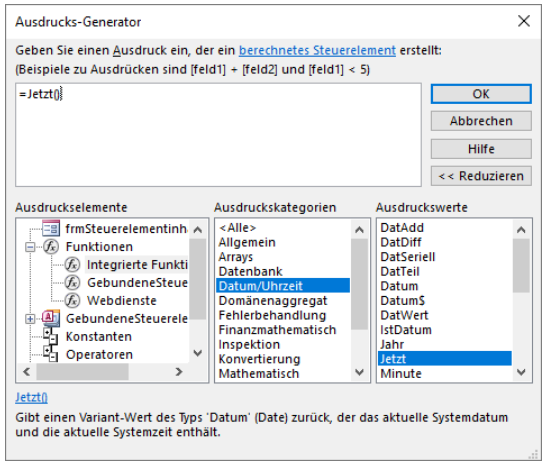


Abbildung 5.5: Auswahl der VBA-Funktionen als Steuerelementinhalt

Wenn wir nun noch ein weiteres Textfeld namens *txtAktuellesJahr* hinzufügen, dem wir über den Ausdrucks-Generator die Funktion *Jahr* hinzufügen, landet dieser Ausdruck in der Eigenschaft:

```
=Jahr([«Datum»])
```

Logisch: Die Funktion benötigt noch ein Datum, aus dem es das Jahr herausfiltern soll. Hier haben wir nun die Möglichkeit, einfach die Funktion *Datum* einzufügen:

```
=Jahr(Datum())
```

Hier dürfen Sie nicht das Klammernpaar hinter *Datum()* weglassen, da Datum sonst als Feld interpretiert und automatisch in eckige Klammern eingefasst wird:

```
=Jahr([Datum]) 'so nicht!
```

Wenn Sie wie im Beispiel schon ein Steuerelement verwenden, welches das Datum anzeigt, können Sie auch den Inhalt dieses Textfeldes als Argument für die *Jahr*-Funktion verwenden:

```
=Jahr([txtDatum])
```

5.3 Andere Steuerelemente im gleichen Formular referenzieren

Im obigen Beispiel haben Sie schon auf den Inhalt des Steuerelements *txtDatum* zugegriffen, um den Wert als Parameter einer Funktion zu nutzen. Das ist auch ohne Funktion sinnvoll.

5.3.1 Anderes Steuerelement referenzieren

Sie können der Eigenschaft *Steuerelementinhalt* zum Beispiel auch einfach so den Inhalt eines anderen Feldes zuweisen. Wir wollen im Formular *frmSteuerelementinhalt* ein neues Textfeld namens *txtQuelle* und eines namens *txtZiel* anlegen. *txtQuelle* soll keinen Wert im Feld *Steuerelementinhalt* aufweisen, also ungebunden und zunächst leer sein.

txtZiel statten wir für die Eigenschaft *Steuerelementinhalt* mit dem Ausdruck *=txtQuelle* aus. Nach dem Wechsel in die Formularansicht sind zunächst beide Felder leer. Geben wir dann einen Wert in das Textfeld *txtQuelle* ein und bestätigen die Eingabe etwa durch das Betätigen der *Eingabe*-Taste oder der *Tabulator*-Taste, wird der Inhalt von *txtQuelle* auch in *txtZiel* angezeigt (siehe Abbildung 5.6).

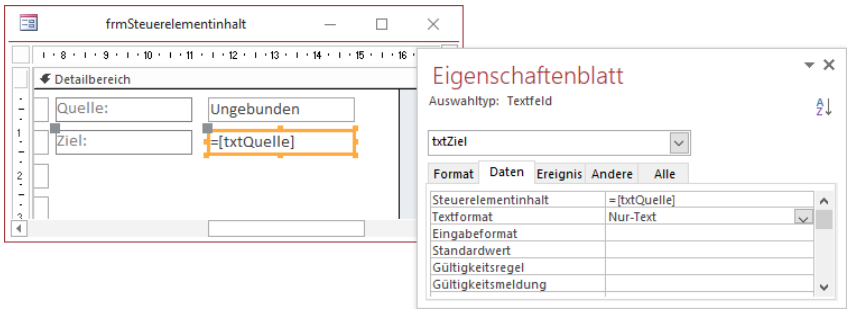


Abbildung 5.6: Verweis auf ein anderes Steuerelement

5.3.2 Zusammengesetzter Verweis

Auf diese Weise können wir in einem Steuerelement auch Inhalte mehrerer anderer Steuerelemente zusammenfügen. Dazu legen wir ein Textfeld namens *txtVorname* und eines namens *txtNachname* an, beide als ungebundene Steuerelemente.

Ein drittes Textfeld namens *txtVollerName* erhält zunächst den folgenden Ausdruck für die Eigenschaft *Steuerelementinhalt*:

```
=[txtVorname] & [txtNachname]
```

6 Formular-Features

In diesem Kapitel betrachten wir einige spezielle Features für Formulare, beispielsweise die Möglichkeit, eine Vorlage zu erstellen, Unterdatenblätter zu nutzen oder Inhalte mit der bedingten Formatierung zu formatieren.

6.1 Formular-Vorlage

Wie wir schon in »Einstellungen für das Hinzufügen gebundener Steuerelemente« ab Seite 50 beschrieben haben, können Sie für die Steuerelemente eines Formulars Standardwerte für die Eigenschaften festlegen, die dann beim Neuanlegen aller Steuerelemente des jeweiligen Typs übernommen werden. Auf diese Weise können Sie sich auch eine Formularvorlage erstellen, die Sie unter dem Namen *Normal* speichern, der normalerweise als Name des Standardformulars in den Access-Optionen angegeben ist.

Diese Einstellung prüfen wir nun als Erstes. Dazu öffnen Sie die Access-Optionen, indem Sie auf den Ribbon-Reiter *Datei* klicken und aus dem nun erscheinenden Backstage-Bereich den Eintrag *Optionen* auswählen. Es erscheint der Dialog *Access-Optionen*, in dem Sie im Bereich *Objekt-Designer* unter *Entwurfsansicht für Formulare/Berichte* die Eigenschaft *Formularvorlage* finden. Hier sollte der Wert *Normal* gespeichert sein (siehe Abbildung 6.1).

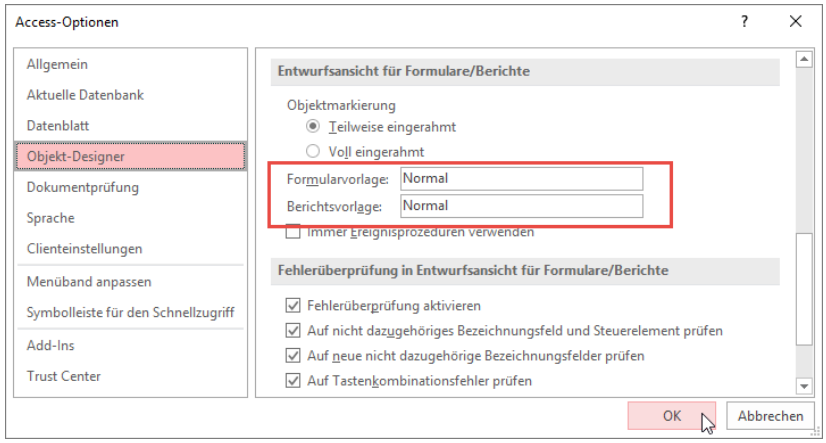


Abbildung 6.1: Einstellen des Namens für die Formularvorlage

Nun machen wir uns an die Erstellung einer Formularvorlage, indem wir ein neues, leeres Formular anlegen. Dieses speichern wir unter dem Namen *Normal*.

Dabei können wir nun zwei verschiedene Dinge durchführen:

» Wir können das grundlegende Aussehen des Formulars so definieren, wie wir es für weitere Formulare wünschen. Wir können also etwa einen Formularkopf anlegen, der eine bestimmte Hintergrundfarbe hat oder wir können im Formularfuß eine *OK*- und eine *Abbrechen*-Schaltfläche unterbringen, weil wir denken, dass wir diese in den meisten Formularen, die wir neu anlegen, benötigen werden.

» Außerdem können wir die Eigenschaften für die Steuerelemente, die wir in unseren Formularen auf Basis dieser Formularvorlage erstellen wollen, vordefinieren. Das gelingt im Gegensatz zum Vorbereiten konkreter Elemente der Benutzeroberfläche oder Steuerelemente ohne das Anlegen eines einzigen Steuerelements.

6.1.1 Steuerelement-Eigenschaften vordefinieren

Wir beginnen mit dem zweiten Punkt, da wir auf dessen Basis auch gleich die beispielhaft zur Formularvorlage hinzugefügten Schaltflächen erstellen können. Nehmen wir zum Beispiel an, wir wollen eine Anwendung für ein Unternehmen programmieren, das strenge Vorschriften bezüglich des Corporate Design hat. So soll beispielsweise immer die gleiche Schriftart verwendet werden.

Um dies für unsere Anwendung zumindest in den Formularen zu gewährleisten, müssten wir also normalerweise immer für alle Steuerelemente die Schriftart ändern. Damit wir uns diese Arbeit nur einmal machen müssen, legen wir unser Formular namens *Normal* an und speichern es. Dann können wir uns an die Einstellung der gewünschten Eigenschaften machen.

Um beispielsweise einzustellen, dass die Schriftart in Textfeldern immer *Consolas* sein soll. Dann gehen wir wie folgt vor:

» Öffnen Sie das Formular *Normal* in der Entwurfsansicht.

» Klicken Sie in der Liste der Steuerelemente im Ribbon unter *Entwurf|Steuerelemente* auf das Textfeld, aber klicken Sie dann nicht in den Formularentwurf – wir wollen nur die Eigenschaften ändern, aber kein Textfeld anlegen!

» Wechseln Sie zum Eigenschaftenblatt. Betätigen Sie die Taste *F4*, falls das Eigenschaftenblatt aktuell nicht angezeigt wird.

» Stellen Sie dort auf der Seite Format die Eigenschaft *Schriftart* auf *Consolas* ein. Um diese schnell zu finden, geben Sie einfach die Anfangsbuchstaben ein, statt in der Liste zu suchen.

» Betätigen Sie die Tastenkombination *Strg + S*, um die Änderung zu speichern.

Legen Sie nun versuchsweise ein neues Textfeld im Formular an. Dieses sollte nun die Schriftart *Consolas* aufweisen. Allerdings erkennen wir schnell, dass dies nur für das Textfeld gilt, nicht

jedoch für das automatisch mit angelegte Bezeichnungsfeld. Für dieses müssen wir die Schriftart separat einstellen.

Eigenschaften für das Bezeichnungsfeld einstellen

Also stellen wir nun die Schriftart für das Bezeichnungsfeld auf die gleiche Art und Weise ein wie zuvor für das Textfeld: Wir markieren also die Schaltfläche zum Hinzufügen eines Bezeichnungsfeldes im Ribbon, ohne dieses tatsächlich hinzuzufügen, und stellen die Eigenschaft Schriftart auf *Consolas* ein.

Außerdem wollen wir dafür sorgen, dass das Bezeichnungsfeld etwa beim Hinzufügen eines Textfeldes aus dem Ribbon oder aus der Feldliste automatisch einen Doppelpunkt hinter dem Text erhält. Das wiederum können wir nicht direkt für das Bezeichnungsfeld einstellen: Diese Eigenschaft entsteht erst im Kontext des Bezeichnungsfeldes als Teil des Textfeldes (oder eines anderen Steuerelements, dem automatisch ein Bezeichnungsfeld hinzugefügt wird) und muss somit für das Textfeld festgelegt werden. Die dazu notwendige Eigenschaft des Textfeldes bekommen wir im Gegensatz zu den bisher geänderten Eigenschaften nur zu sehen, wenn wir es im Ribbon anklicken – aber nicht, wenn wir das Textfeld bereits in den Formularentwurf eingefügt haben. Der Name der Eigenschaft lautet *Mit Doppelpunkt* und wir müssen den standardmäßig eingestellten Wert auf *Ja* ändern (siehe Abbildung 6.2).

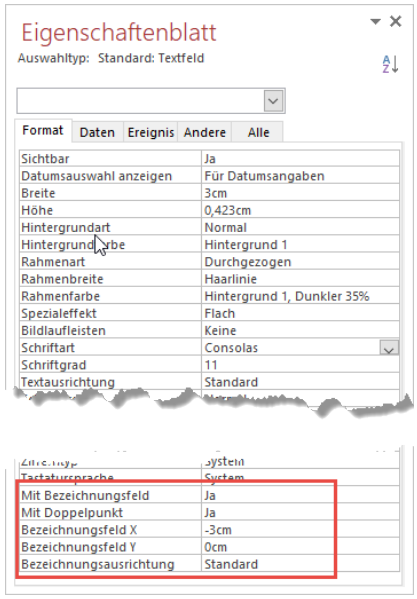


Abbildung 6.2: Eigenschaften, die nur für das noch nicht angelegte Textfeld angezeigt werden

Wenn wir das Formular nun speichern und ein neues Textfeld hinzufügen, zeigen sowohl das Textfeld als auch das automatisch hinzugefügte Bezeichnungsfeld ihre Texte in der Schriftart *Consolas* an. Außerdem finden wir auch einen Doppelpunkt hinter dem Text des Bezeichnungsfeldes vor und brauchen diesen nicht mehr manuell hinzuzufügen.

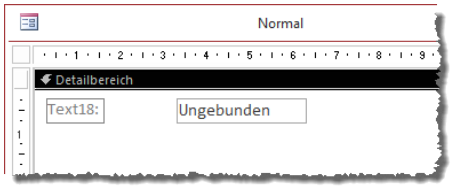


Abbildung 6.3: Textfeld in der gewünschten Schrift mit Doppelpunkt hinter dem Text des Bezeichnungsfeldes

Wenn Sie nicht sicher sind, ob Sie gerade die Eigenschaften eines bereits eingefügten Steuerelements bearbeiten oder die der Steuerelementvorlage, schauen Sie sich einfach das Auswahlfeld oben im Eigenschaftenblatt an: Zeigt dieses keinen Eintrag an, bearbeiten Sie gerade den Entwurf eines der Steuerelemente. Anderenfalls würde dort der Name des markierten Steuerelements oder Formularbereichs erscheinen.

Schriftart für weitere Steuerelemente einstellen

Damit die Schriftart auch der übrigen Steuerelemente zum Corporate Identity unserer fiktiven Firma passt, müssen wir diese nach und nach in der Toolbox im Ribbon anklicken, die Schriftart auf *Consolas* einstellen und den Formularentwurf speichern. Diese Änderung müssen wir für alle Steuerelemente vornehmen, die über die Eigenschaft *Schriftart* verfügen. Außerdem müssen Sie, wenn Sie wollen, dass das Bezeichnungsfeld auch für diese Steuerelemente automatisch den Doppelpunkt einfügt, wenn Sie es aus der Feldliste hinzufügen, auch für diese Steuerelemente die Eigenschaft *Mit Doppelpunkt* auf den Wert *Ja* einstellen.

Kontrollkästchen vorbereiten

Auch das Kontrollkästchen können Sie auf die gleiche Art anpassen, sodass Sie verschiedene Änderungen nicht mehr von Hand durchführen müssen. Hier gibt es allerdings eine Besonderheit: Wenn Sie ein *Ja/Nein*-Feld aus der Feldliste zum Formular hinzufügen, wird das Bezeichnungsfeld rechts vom Kontrollkästchen eingefügt (siehe Abbildung 6.4). Das ist nicht sinnvoll, denn Sie wollen Kontrollkästchen ja genauso anordnen, wie Sie es mit einem Textfeld oder einem Kombinationsfeld erledigen. Der Grund für diese Platzierung ist, dass die Eigenschaft *Bezeichnungsfeld X* hier den Wert *0,406cm* aufweist. Wenn Sie diesen wie bei den anderen Steuerelementen wie etwa beim Textfeld auf *-3cm* einstellen, landet das Bezeichnungsfeld wie die der anderen Steuerelemente drei Zentimeter links vom Kontrollkästchen.

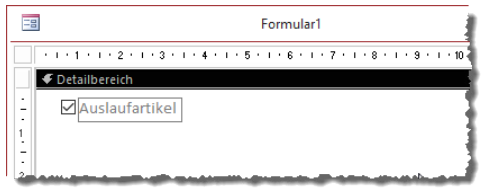


Abbildung 6.4: Das Bezeichnungsfeld eines Kontrollkästchens landet standardmäßig rechts vom Kontrollkästchen.

Schaltfläche vorbereiten

Mit dem Textfeld haben wir bereits eines der am meisten verwendeten Steuerelemente angepasst. Nun kümmern wir uns noch um die Schaltfläche. Diese soll immer ein Icon in der Größe 24 x 24 anzeigen sowie den Text als Link markiert präsentieren, also mit blauer Textfarbe und unterstrichen. Davon ab soll es keinen Rahmen aufweisen und einen transparenten Hintergrund besitzen. Das erledigen wir in den folgenden Schritten:

» Öffnen Sie das Formular *Normal* in der Entwurfsansicht.

» Klicken Sie im Ribbon auf das Steuerelement *Schaltfläche*.

» Aktivieren Sie mit *F4* das Eigenschaftenblatt.

» Stellen Sie die Eigenschaft *Hintergrundart* auf *Transparent* ein, die *Schriftart* auf *Consolas*, die *Schriftfarbe* auf *Akzent 1, Dunkler 25%* sowie *Unterstrichen* auf *Ja*.

» Außerdem können Sie die Eigenschaft *Bildbeschriftung* schon auf *Rechts* einstellen.

Damit sind die Möglichkeiten bezüglich der Steuerelementvorlage ausgeschöpft – wenn Sie tatsächlich Schaltfläche mit Icons vorab in einem Formular zur Verfügung stellen wollen, müssen Sie ein herkömmliches Formular als Vorlage erstellen, die Sie dann bei Bedarf einfach kopieren (siehe weiter unten).

Formularvorlage ausprobieren

Speichern Sie das Formular und schließen Sie es. Nun wollen wir ausprobieren, ob die Vorlage so funktioniert wie gewünscht. Also legen Sie über den Ribbon-Befehl *Erstellen|Formulare|Formularentwurf* ein neues Formular in der Entwurfsansicht an. Ziehen Sie dann beispielsweise ein Textfeld und eine Schaltfläche in den Detailbereich des Formularentwurfs. Das Ergebnis sieht genau wie gewünscht aus (siehe Abbildung 6.5).

Abbildung 6.5: Ein neues Formular mit Steuerelementen auf Basis der im Formular *Normal* angelegten Formate

6.1.2 Steuerelemente und Formulareigenschaften vordefinieren

Nun nehmen wir noch einige weitere Änderungen am Formular vor. Wir wollen beispielsweise einen Kopf- und Fußbereich einblenden sowie zwei Schaltflächen mit den Beschriftungen *OK* und *Abbrechen* und entsprechenden Icons im Fußbereich anzeigen.

Dazu blenden wir zunächst Kopf- und Fußbereich mit dem Kontextmenü-Befehl *Formularkopf/-fuß* des Detailbereichs ein. Dann fügen wir im Fußbereich eine neue Schaltfläche hinzu und ändern den Text auf *OK*. Außerdem wählen wir, während diese Schaltfläche den Fokus hat, den Ribbon-Befehl *Entwurf|Steuerelemente|Bild einfügen* aus.

Mit dem nun erscheinenden Befehl *Durchsuchen* öffnen wir den *Grafik einfügen*-Dialog und wählen eine geeignete Bilddatei aus – in unserem Fall in der Größe 24 x 24 und mit einem grün hinterlegten Haken als Motiv. Dieses Icon wird dann in die Schaltfläche eingefügt, deren Größe wir entsprechend der Bildgröße anpassen. Außerdem stellen wir den Namen der Schaltfläche auf *cmdOK* ein.

Das Gleiche erledigen wir noch für die Schaltfläche *cmdAbbrechen*, die lediglich eine andere Beschriftung (*Abbrechen*) und ein anderes Icon erhält.

Zu guter Letzt stellen wir noch die Eigenschaft *Automatisch zentrieren* des Formulars auf den Wert *Ja* ein. Das Ergebnis sieht zunächst wie in Abbildung 6.6 aus.

Nun fehlt noch die Probe aufs Exempel: Wir speichern und schließen die Formularvorlage und legen ein neues Formular in der Entwurfsansicht an. Das Ergebnis ernüchtert ein wenig: Das Formular erscheint zwar zentriert und zeigt den Kopf- und den Fußbereich an, aber die beiden von uns angelegten Schaltflächen werden nicht übernommen.

Letztlich ist das aber kein Problem: Schließlich ist es kaum mehr Aufwand, einfach das Formular *Normal* zu kopieren und unter einem neuen Namen, zum Beispiel *frmKunden*, in die Datenbank einzufügen. Und auf diese Weise werden auch alle bereits angelegten Steuerelemente übernommen.

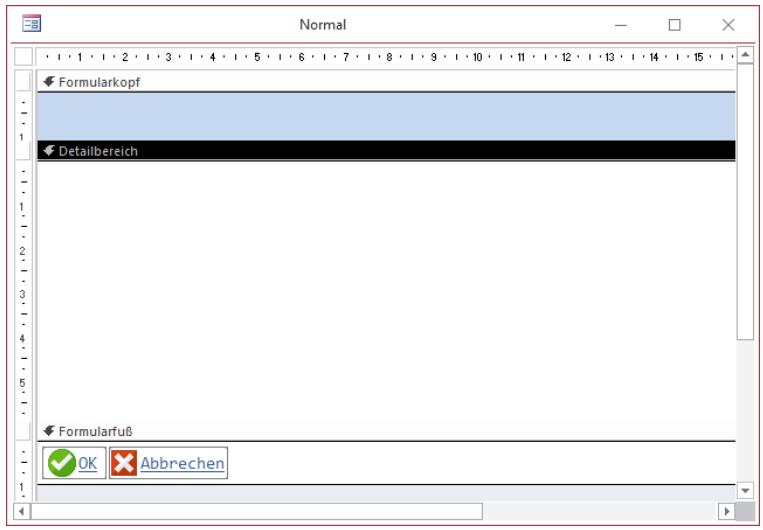


Abbildung 6.6: Formularvorlage mit Schaltflächen

Außerdem werden Sie ohnehin nicht mit einer Vorlage auskommen, sondern zum Beispiel eine Formularvorlage für die Datenblattansicht und eine für die Detailansicht der Daten aus verschiedenen Tabellen benötigen.

6.2 Unterdatenblätter

Unterdatenblätter sind eines der Features, die mit Access 2007 eingeführt wurden. Diese sind in allen Datenblattansichten von Access verfügbar, aber nur in der Formularansicht können sie ihr volles Potenzial entfalten.

Die Datenblattansicht zeigt normalerweise immer nur die Daten einer Tabelle oder Abfrage an, wobei die Spalten für jede Zeile gleich aussehen. Mit Unterdatenblättern jedoch können Sie zu jedem Datensatz einer Tabelle noch die Datensätze einer damit verknüpften Tabelle anzeigen.

Sie können also beispielsweise alle Datensätze der Tabelle *tblKunden* in der obersten Ebene anzeigen und darunter alle Datensätze aus der Tabelle *tblBestellungen*, die zu dem jeweiligen Kunden gehören.

Und zu diesen Bestellungen können Sie dann sogar noch alle Bestellpositionen ausgeben (siehe Abbildung 6.7).

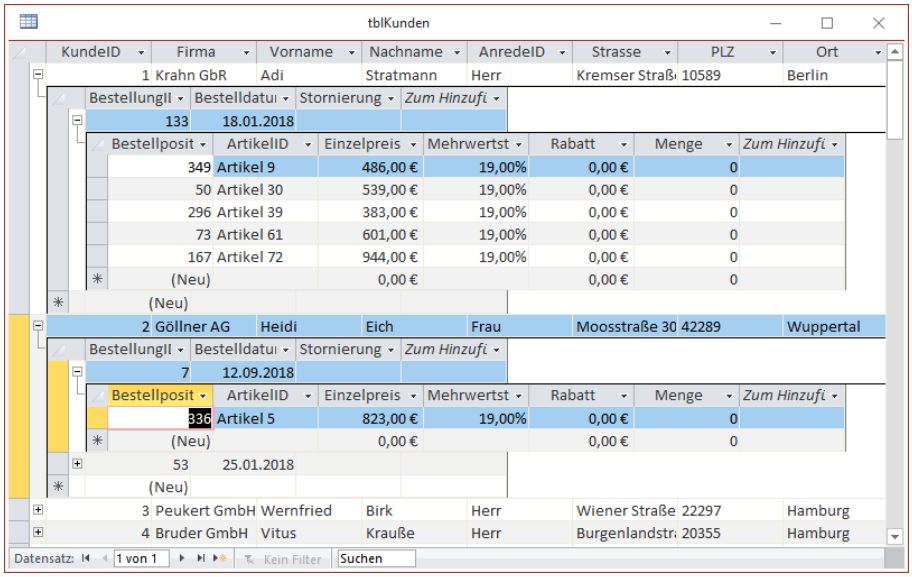


Abbildung 6.7: Unterdatenblätter in der Datenblattansicht einer Tabelle

Sollte dies nicht der Fall sein, können Sie zunächst prüfen, ob die Tabellen die notwendigen Beziehungen aufweisen. Außerdem werfen Sie einen Blick auf die Eigenschaft *Unterdatenblattname*, der normalerweise den Wert *[Automatisch]* aufweisen sollte (siehe Abbildung 6.8).

Wenn Sie hier einen anderen Wert auswählen, also etwa *Tabelle.tblBestellungen*, liefern die beiden weiteren Eigenschaften *Verknüpfen von* und *Verknüpfen nach* noch eine Liste der Felder der beiden Tabellen. Diese Eigenschaften enthalten die Namen der beiden an der Beziehung beteiligten Felder, also das Primärschlüsselfeld der einen Tabelle und das Fremdschlüsselfeld der anderen Tabelle.

Wir schauen uns in den folgenden Abschnitten an, wie wir das auch in Formularen realisieren können.

6.2.1 Neues Formular anlegen

Als Erstes legen wir dazu ein neues Formular namens *frmUnterdatenblatt* in der Entwurfsansicht an und wählen als Datensatzquelle die Tabelle *tblKunden* aus. Wir wechseln zur Feldliste und ziehen alle Felder der Tabelle in den Detailbereich des Formularentwurfs.

Damit die Daten gleich in der Datenblattansicht erscheinen, stellen wir die Eigenschaft *Standardansicht* auf den Wert *Datenblatt* ein.

Das Ergebnis nach dem Wechsel in die Datenblattansicht sieht dann wie in Abbildung 6.9 aus. Hier gibt es noch keine Spur von einem Unterdatenblatt.

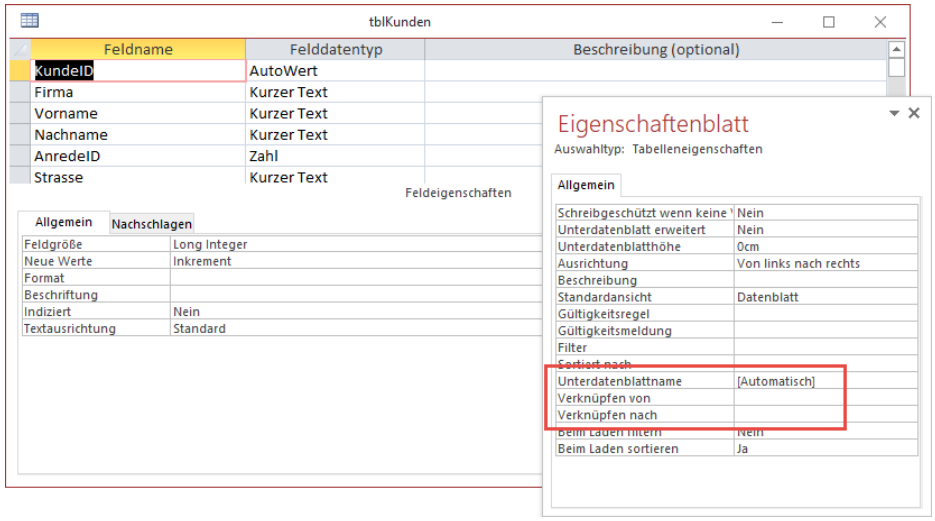


Abbildung 6.8: Einstellungen für Unterdatenblätter

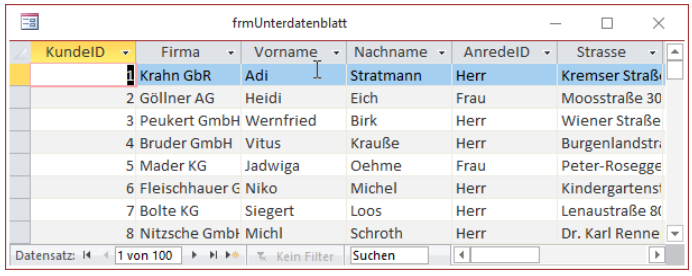


Abbildung 6.9: Datenblatt ohne Unterdatenblatt

Damit sich dies ändert, zeigen wir das Eigenschaftenblatt des Formulars in der Datenblattansicht an. Dies erreichen Sie durch einen rechten Mausklick in das Formular und Auswahl des Kontextmenü-Eintrags *Formulareigenschaften* (siehe Abbildung 6.10).

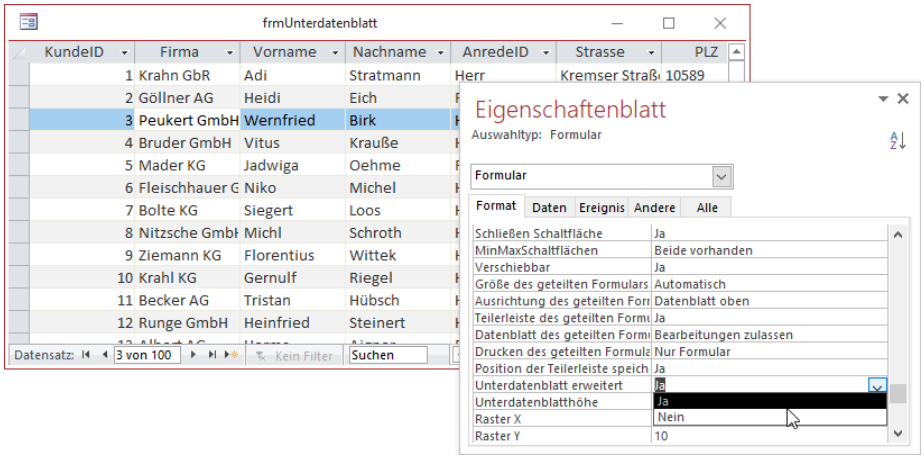


Abbildung 6.10: Aktivieren des Unterdatenblatts

Nachdem Sie hier den Wert *Ja* für die Eigenschaft *Unterdatenblatt erweitert* ausgewählt haben, erscheint der Dialog aus Abbildung 6.11.

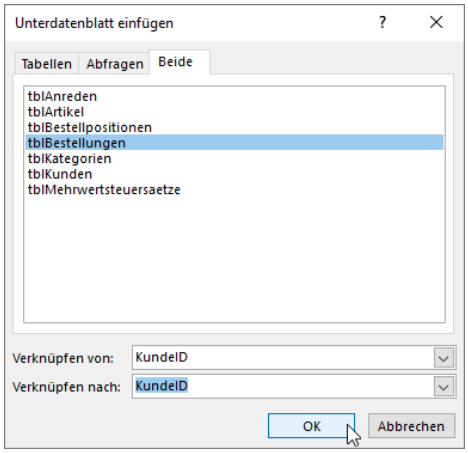


Abbildung 6.11: Einstellen der Eigenschaften für das Unterdatenblatt

Hier müssen Sie nun drei Dinge einstellen: Als Erstes wählen Sie oben die Tabelle aus, deren Daten im Unterdatenblatt angezeigt werden sollen. Danach legen Sie fest, von welchem Feld der hier gewählten Tabelle, also *tblBestellungen*, zu welchem Feld der Tabelle im Formuar, also *tblKunden*, die Verknüpfung für die Datensätze des Unterdatenblatts hergestellt werden soll – in diesem Fall zweimal das Feld *KundeID*.Interessanterweise ändert sich ... nichts! Wir können die Datensatzquelle mit *F5* aktualisieren und das Formular schließen und wieder öffnen,

7 Formulare programmieren

Formulare können schon durch das Füllen mit einer Datensatzquelle und die Anzeige der Daten in gebundenen Steuerelementen eine Menge Funktionen abbilden. Richtig interessant wird es aber erst, wenn wir die Programmiersprache VBA hinzunehmen und die Formular- und Steuerelementereignisse nutzen, um benutzerdefinierte Aktionen auszuführen. In diesem Kapitel stellen wir Ihnen die Grundlagen zur Programmierung von Formularen mit VBA vor.

Einen Grundkurs in VBA können wir aus Platzgründen leider nicht anbieten, aber wir werden einführend alle wichtigen Begriffe und Elemente detailliert erläutern.

7.1 Objektorientierter Ansatz

Die Verwendung von VBA-Code in Formularen und Steuerelementen nutzt einige Konzepte der objektorientierten Programmierung – aber nicht alle, denn dazu ist die Programmiersprache VBA gar nicht ausgelegt. An dieser Stelle reicht das Wissen aus, dass sowohl die Formulare als auch die Steuerelemente (genauso wie einige andere Elemente in der VBA-Programmierung) als Objekte angesehen werden und dass diese drei verschiedenen Möglichkeiten zur Programmierung bieten:

» *Eigenschaften*: Eigenschaften kennen Sie bereits aus den vorherigen Kapiteln. Dort haben wir die Eigenschaften verschiedener Elemente wie dem Formular selbst oder auch den darin enthaltenen Steuerelementen über das Eigenschaftenblatt zugewiesen. Dies gelingt auch per VBA. Auf diese Weise können Sie beispielsweise den Titel eines Formulars ändern, den Inhalt von Steuerelemente einstellen und vieles mehr.

» *Methoden*: Methoden sind VBA-Befehle, die von den jeweiligen Objekten bereitgestellt werden. Ein Formular stellt beispielsweise eine Methode bereit, mit der Sie die enthaltenen Daten aktualisieren können. Sie könnten dies über die Benutzeroberfläche auslösen, indem Sie beispielsweise die Taste *F5* drücken, aber es gelingt auch per VBA über die Methode *Requery*. Solche Methoden gibt es für das Formular selbst, aber auch für die einzelnen Steuerelemente.

» *Ereignisse*: Ereignisse werden durch verschiedene Aktionen ausgelöst, beispielsweise durch das Anklicken einer Schaltfläche, durch das Wechseln zu einem neuen Datensatz, durch das Löschen eines Datensatzes et cetera. Sie können diese Ereignisse nutzen, um speziell für ein solches Ereignis zugeschnittene Prozeduren zu definieren. Wenn ein Ereignis ausgelöst wird, für das Sie eine Ereignisprozedur definiert haben, wird diese automatisch auch ausgelöst. Sie können so beispielsweise festlegen, dass beim Anklicken einer Schaltfläche mit der Beschriftung *OK* eine Prozedur ausgelöst wird, die das aktuelle Formular schließt.

7.2 Der VBA-Editor

Bevor wir gleich mit den ersten Beispielen beginnen, wollen wir uns kurz ansehen, wo Sie die ersten Befehle absetzen können und wo Sie beim Programmieren Ihrer Anwendungen viel Zeit verbringen werden. Wir reden über den VBA-Editor, welcher neben der Benutzeroberfläche von Access das zweite Element ist.

Den VBA-Editor bekommen in der Regel nur die Entwickler zu sehen, der Benutzer sollte keinen Zugriff auf den Code einer Anwendung erhalten. Den VBA-Editor öffnen Sie für eine Anwendung mit einer der Tastenkombinationen *Alt + F11* oder *Strg + G*, wobei letztere direkt den sogenannten Direktbereich aktiviert. Der VBA-Editor sieht wie in Abbildung 7.1 aus.

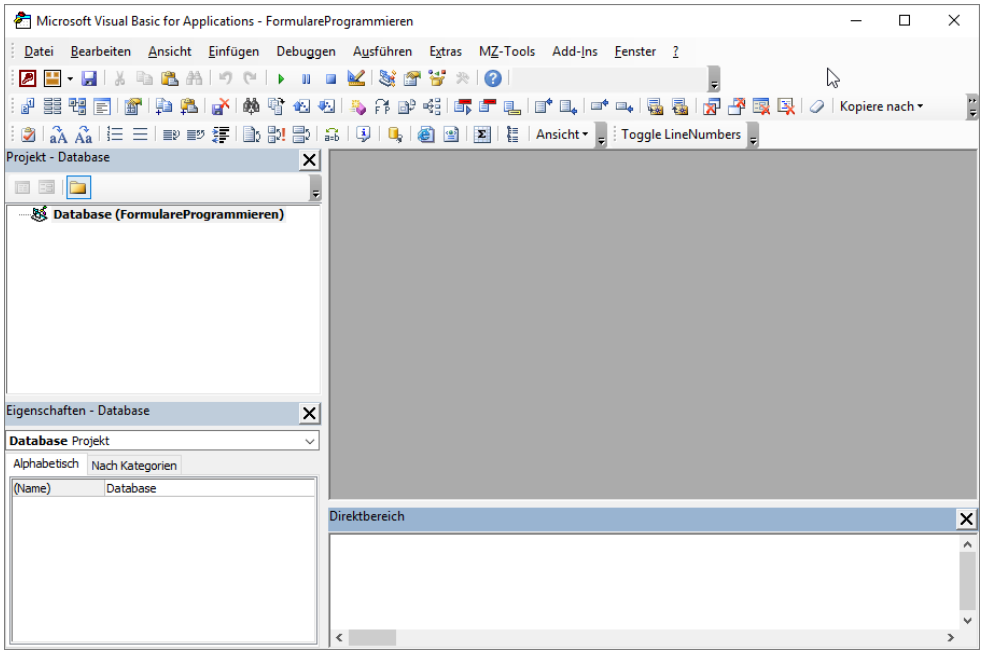


Abbildung 7.1: Der VBA-Editor eines jungfräulichen VBA-Projekts

Oben finden Sie die eingebauten Menü- und Symbolleisten. Diese sind in diesem Fall ergänzt um weitere Elemente, die durch das Add-In MZ-Tools bereitgestellt werden. Auf der linken Seite finden Sie den Projekt-Explorer, der aktuell nur den Eintrag für das VBA-Projekt der aktuell geöffneten Datenbank anzeigt, das bisher noch keine Objekte enthält.

Darunter sehen Sie das Eigenschaften-Fenster, das die Eigenschaften des aktuell markierten Elements anzeigt. Mit diesem werden Sie keinen großen Kontakt haben, außer Sie möchten

den Namen des aktuellen VBA-Projekts ändern oder den Namen des aktuell im Projekt-Explorer markierten Elements.

Unten finden Sie den Direktbereich. Dieser ist ein sehr hilfreiches Tool bei der Entwicklung von Access-Anwendungen und auch somit natürlich auch bei der Programmierung von Formularen. Damit können Sie folgende wichtige Dinge erledigen:

» Im Code Ihrer Prozeduren und Funktionen zu Debugging-Zwecken Inhalte von Variablen und anderen Elementen ausgeben,

» Befehle eingeben, um Aktionen auszuführen – wie beispielsweise gleich im Anschluss zum Öffnen von Formularen und

» mit der *Debug.Print*-Anweisung die Inhalte von Variablen oder die Werte von Eigenschaften der Elemente der Benutzeroberfläche, also von Formularen und Steuerelementen, ausgeben.

Geben Sie dort beispielsweise einmal die folgende Anweisung ein und betätigen die Eingabetaste, erhalten Sie die folgende Ausgabe:

```
Debug.Print CurrentProject.Path
C:\Users\User\Dropbox\Daten\Buecher\AFOR\Beispieldateien\FormulareProgrammieren
```

Sie können so also beispielsweise Eigenschaften des aktuellen Projekts ausgeben, hier den Pfad zur Datenbankdatei.

Den Direktbereich werden wir im Anschluss für die ersten Aufrufe von Formularen einsetzen.

Einen Bereich haben wir noch nicht angesprochen, weil dieser aktuell noch leer ist: Der graue Bereich ist den Code-Fenstern vorbehalten. Hier werden die Inhalte der Klassenmodule von Formularen, alleinstehenden Klassenmodulen und Standardmodulen angezeigt und bearbeitet. Wie Sie solche Formulare anlegen, erläutern wir, wenn es soweit ist.

7.3 Formulare öffnen

Wie Sie ein Formular über die Benutzeroberfläche öffnen, wissen Sie ja bereits. Sie klicken doppelt auf den gewünschten Eintrag im Navigationsbereich und das Formular erscheint. Die zweite Möglichkeit ist, in den Access-Optionen ein Formular anzugeben, das beim Starten der Anwendung automatisch angezeigt wird (siehe »Ein Formular beim Start der Anwendung öffnen« ab Seite 36).

Es gibt aber natürlich auch die Möglichkeit, ein Formular per Code zu öffnen. Das ist allein schon deshalb wichtig, weil Sie ja auch einmal von einem Formular aus etwa per Mausklick auf eine Schaltfläche ein weiteres Formular öffnen wollen. Oder Sie wollen benutzerdefinierte Einträge im Ribbon anlegen, mit denen Sie Formulare öffnen können.

Es gibt natürlich auch noch die Möglichkeit, ein Formular per Makro zu öffnen, aber Makros wollen wir in diesem Buch nicht beschreiben, da man alle Aufgaben, die man mit Makros löst, genauso gut oder besser mit VBA lösen kann.

Es gibt zwei Ausnahmen, dabei handelt es sich um das Makro *AutoExec*, das automatisch beim Starten von Access aufgerufen wird, und das Makro *AutoKeys*, mit dem Sie anwendungsweit gültige Tastenkombinationen definieren können.

7.3.1 Die DoCmd.OpenForm-Methode

Die gängigste Methode, ein Formular unter VBA zu öffnen, ist die Methode *OpenForm* des *DoCmd*-Objekts. *DoCmd* ist ein Objekt der Klasse *Application*, das Sie einfach so im Code verwenden können.

Um die Methoden dieses Objekts zu nutzen, geben Sie einfach den Objektnamen ein, also *DoCmd*, gefolgt von einem Punkt. Eine Einrichtung namens IntelliSense zeigt dann alle möglichen Methoden dieses Objekts an. Wir suchen in diesem Fall nach der Methode *OpenForm* (siehe Abbildung 7.2).

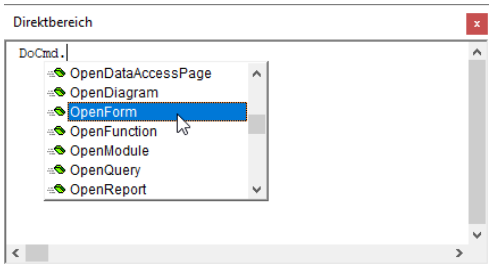


Abbildung 7.2: Eingeben des *DoCmd.OpenForm*-Befehls

Wählen Sie diese aus und geben Sie ein Leerzeichen ein, erscheint eine Liste mit den möglichen Parametern dieses Befehls (siehe Abbildung 7.3).

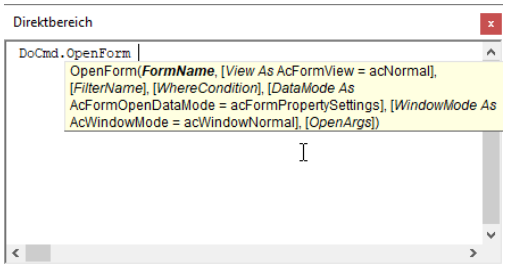


Abbildung 7.3: Parameter des *DoCmd.OpenForm*-Befehls

Diese schauen wir uns nun an:

» *FormName*: Der Name des Formulars, das geöffnet werden soll, zum Beispiel *frmKunden*. Dieser Parameter wird in Anführungszeichen angegeben – außer, Sie speichern den Namen des zu öffnenden Formulars in einer Variablen und geben diese als Wert für den Parameter *FormName* an.

» *View*: Ansicht, in welcher das Formular geöffnet werden soll. Es gibt die folgenden Werte: *acDesign* – Entwurfsansicht, *acFormDS* – Datenblattansicht, *acFormPivotChart* – wird nicht mehr unterstützt, *acFormPivotTable* – wird nicht mehr unterstützt, *acLayout* – Layoutansicht, *acNormal* (Standardwert) – Formularansicht, *acPreview* – Vorschau auf die Druckversion

» *FilterName*: Name einer Abfrage, von der dann die *WHERE*-Klausel als Filter verwendet wird

» *WhereCondition*: Bedingung für die Auswahl der Datensätze im Format der *WHERE*-Klausel einer Abfrage wie beispielsweise *Firma = "A*"* für alle Kunden, deren Firma mit *A* beginnt

» *DataMode*: Gibt an, in welchem Modus bezüglich der Daten das Formular geöffnet wird. Es gibt folgende Werte: *acFormAdd* öffnet das Formular mit einem leeren, neuen Datensatz, *acFormEdit* öffnet das Formular zum Bearbeiten von Datensätzen, *acFormPropertySettings* (Standardwert) übernimmt die mit speziellen Eigenschaften eingestellten Optionen für die Anzeige und Bearbeitbarkeit der Daten, *acFormReadOnly* öffnet das Formular im schreibgeschützten Modus

» *WindowMode*: *acDialog* öffnet das Formular als modalen Dialog, *acHidden* öffnet das Formular und blendet es aus, *acIcon* öffnet das Formular minimiert, *acWindowNormal* (Standardwert) öffnet das Formular normal.

» *OpenArgs*: Hiermit übergeben Sie einen Wert als Öffnungsargument an das zu öffnende Formular, das dann von diesem ausgelesen und verarbeitet werden kann.

7.3.2 Formular in verschiedenen Ansichten öffnen

Wenn wir alle optionalen Parameter weglassen und die Methode *DoCmd.OpenForm* nur mit dem Namen des Formulars als ersten Parameter öffnen, wird dieses in der Formularansicht geöffnet:

```
DoCmd.OpenForm "frmKunden"
```

Wenn Sie ein Formular gezielt in der Datenblattansicht öffnen wollen, müssen Sie also den Parameter *View* angeben. Für diesen Parameter, der in der Liste der Parameter direkt hinter dem Namen des zu öffnenden Formulars folgt, geben wir den Wert *acFormDS* an:

```
DoCmd.OpenForm "frmKundenUebersicht", acFormDS
```

Wenn das Formular in der Layoutansicht geöffnet werden soll, verwenden wir den Wert *acLayout*.

Wenn Sie ein Formular in der Endlosansicht öffnen wollen, müssen Sie seine Eigenschaft *Standardansicht* auf den Wert *Endlosformular* einstellen und dieses dann als Standardformular öffnen.

7.3.3 Formular mit FilterName öffnen

Für *FilterName* geben Sie den Namen einer Abfrage an, aus der das Formular dann beim Öffnen den für die *WHERE*-Klausel der Abfrage angegebenen Ausdruck ausliest und diesen dann als Filter für das Formular einsetzt.

Angenommen, Sie haben eine Abfrage wie die aus Abbildung 7.4 angelegt, die alle Kunden ausgeben soll, deren Firmenbezeichnung mit dem Buchstaben *A* beginnt.

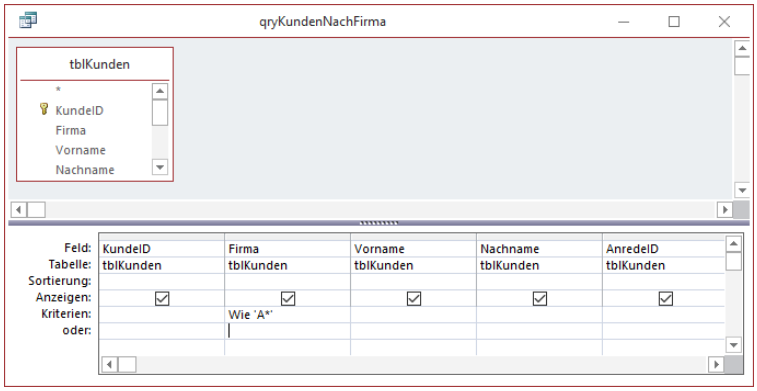


Abbildung 7.4: Abfrage mit einem Kriterium, das alle Kunden liefert, deren Firma mit *A* beginnt

Außerdem haben Sie ein Formular namens *frmKundenUebersicht* in der Datenblattansicht definiert, das alle Kunden ausgibt. Dieses öffnen wir nun mit dem folgenden Befehl:

```
DoCmd.OpenForm "frmKundenUebersicht", acFormDS, "qryKundenNachFirma"
```

Wir geben also für den ersten Parameter namens *View* den Wert *acFormDS* an, was das Formular in der Datenblattansicht öffnet und den Namen der Abfrage *qryKundenNachFirma* als zweiten Parameter, um das Kriterium aus dieser Abfrage als Kriterium für die Ausgabe der Daten zu nutzen. Dies öffnet das Formular wie in Abbildung 7.5.

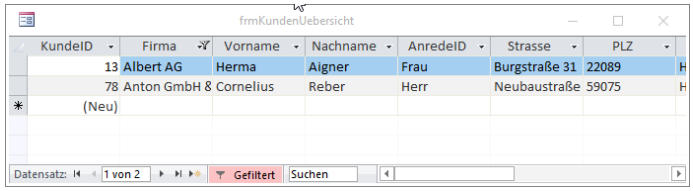


Abbildung 7.5: Formular, das mit dem *FilterName*-Parameter geöffnet wurde

Würden Sie folgenden Aufruf verwenden, würden Sie das Formular mit dem gleichen Kriterium öffnen, aber in der Formularansicht, da beim Wegfallen des Parameters View der Standardwert zum Einsatz kommt – hier (siehe Abbildung 7.6).

```
DoCmd.OpenForm "frmKundenUebersicht", , "qryKundenNachFirma"
```

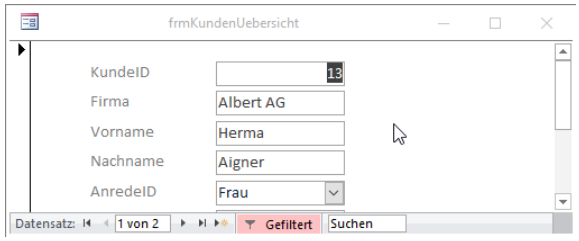


Abbildung 7.6: Formular mit Kriterium in der Formularansicht

7.3.4 Formular mit benannten Parametern öffnen

Sie sehen schon, dass es eventuell etwas unübersichtlich werden könnte, wenn Sie nur einige Parameter der *DoCmd.OpenForm*-Methode nutzen – zum Beispiel nur die erste und die letzte. Das liegt daran, dass die Kommata dazwischen immer noch angegeben werden müssen. Es gibt aber auch eine vereinfachte Variante: Dabei brauchen Sie die Parameter nicht in der richtigen Reihenfolge anzugeben, sondern benennen Sie mit ihrem Parameternamen. In der Praxis hat es sich bewährt, den ersten Parameter, also den Namen des zu öffnenden Formulars, immer noch als ohne Benennung zu schreiben und die übrigen dann mit den Parameternamen zu versehen. Im vorherigen Beispiel würde dies dann etwa wie folgt aussehen. Beachten Sie, dass die Zuweisung immer mit *:=* und nicht etwa nur mit dem Gleichheitszeichen erfolgt:

```
DoCmd.OpenForm "frmKundenUebersicht", FilterName:="qryKundenNachFirma"
```

7.3.5 Formular mit WhereCondition öffnen

Das Öffnen eines Formulars mit dem Parameter *WhereCondition* ist die fortgeschrittene Variante des Parameters *FilterName*. Für *WhereCondition* müssen Sie das Kriterium nämlich als Text an-

geben und können es nicht einfach aus einer Abfrage auslesen. Wenn Sie also das zuvor bereits mit *FilterName* aus der Abfrage *qryKundenNachFirma* ermittelte Kriterium direkt angeben wollen, nutzen Sie dazu den Parameter *WhereConditon*. Der Aufruf sieht dann wie folgt aus:

```
DoCmd.OpenForm "frmKundenUebersicht", WhereCondition:="Firma LIKE 'A*'"
```

Und wenn Sie das Formular wieder als Datenblatt öffnen wollen, sieht der Aufruf so aus:

```
DoCmd.OpenForm "frmKundenUebersicht", View:=acFormDS, WhereCondition:="Firma LIKE 'A*'"
```

Sie können die benannten Parameter übrigens auch vertauschen und müssen nicht die Reihenfolge einhalten wie bei der Auflistung der Parameter ohne Benennung:

```
DoCmd.OpenForm "frmKundenUebersicht", WhereCondition:="Firma LIKE 'A*'", View:=acFormDS
```

Das Ergebnis ist das Gleiche wie beim vorherigen Aufruf des Formulars.

7.3.6 Aufruf für verschiedene Daten-Operationen

Der Parameter *DataMode* ermöglicht es, ein Formular direkt für verschiedene Aktionen aufzurufen, beispielsweise zum Anlegen eines neuen Datensatzes oder zum Bearbeiten vorhandener Datensätze. Wenn der Benutzer mit einem Formular gar nicht auf die vorhandenen Datensätze zugreifen soll, sondern einfach nur einen neuen Datensatz anlegen, verwenden Sie beispielsweise den Wert *acFormAdd* als Wert für diesen Parameter:

```
DoCmd.OpenForm "frmKundeDetails", DataMode:=acFormAdd
```

Dies öffnet das Formular *frmKundeDetails* in der Ansicht aus Abbildung 7.7. Hier können Sie also lediglich einen neuen Datensatz eingeben und das Formular wieder schließen.

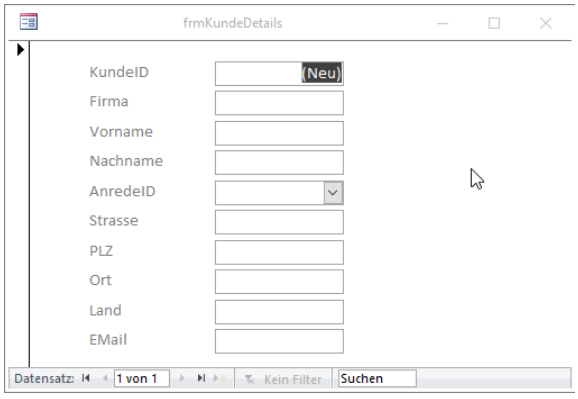


Abbildung 7.7: Ein zum Anlegen eines Datensatzes vorbereitetes Formular

Die zweite Variante für diesen Parameter ist der Wert *acFormEdit*. Damit öffnet *DoCmd.OpenForm* das Formular in einem Modus, in dem der Benutzer neue Datensätze anlegen, aber auch vorhandene Datensätze bearbeiten kann:

```
DoCmd.OpenForm "frmKundeDetails", DataMode:=acFormEdit
```

Die dritte Variante mit dem Wert *acFormReadOnly* für den Parameter *DataMode* sorgt dafür, dass das Formular im schreibgeschützten Zustand geöffnet wird. Der Benutzer kann in diesem Modus nur Daten betrachten, aber nicht bearbeiten, löschen oder neu anlegen:

```
DoCmd.OpenForm "frmKundeDetails", DataMode:=acFormReadOnly
```

Dies erkennen Sie schon daran, dass die die Navigationsschaltfläche zum Anlegen eines neuen Datensatzes deaktiviert ist (siehe Abbildung 7.8). Der Versuch, die Daten des aktuell angezeigten Datensatzes zu überschreiben, scheitert ebenfalls.

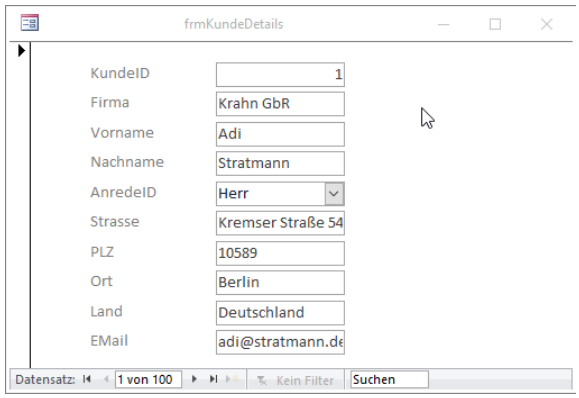


Abbildung 7.8: In diesem Modus können die Daten nicht verändert und auch keine neuen Datensätze angelegt werden.

7.3.7 Aufruf mit verschiedenen Fenstereinstellungen

Der Parameter *WindowMode* gibt an, in welchem Fenster-Modus das Formular geöffnet werden soll. Es gibt die folgenden Werte:

» *acDialog*: Öffnet das Formular als modalen Dialog.

» *acHidden*: Öffnet das Formular im ausgeblendeten Zustand.

» *acIcon*: Öffnet das Formular in einer minimierten Ansicht.

» *acWindowNormal* (Standardwert): Öffnet das Formular im normalen Zustand.

Wenn Sie wie folgt die Einstellung *acDialog* für den Parameter *WindowMode* nutzen, erscheint das Formular wie in Abbildung 7.9:

```
DoCmd.OpenForm "frmKundenUebersicht", WindowMode:=acDialog
```

Die erste Besonderheit ist hier schon ersichtlich: Sie können das als modaler Dialog geöffnete Formular aus dem Arbeitsbereich von Access herausbewegen. Genau genommen können Sie als modale Dialoge geöffnete Formulare überall auf Ihrem Bildschirm platzieren.

Die nächste Besonderheit ist, dass Sie, wenn Sie ein Formular als modalen Dialog geöffnet haben, nicht mehr auf die anderen Bedienelemente und Objekte von Access zugreifen können. Das gelingt erst wieder, wenn Sie den modalen Dialog geschlossen haben.

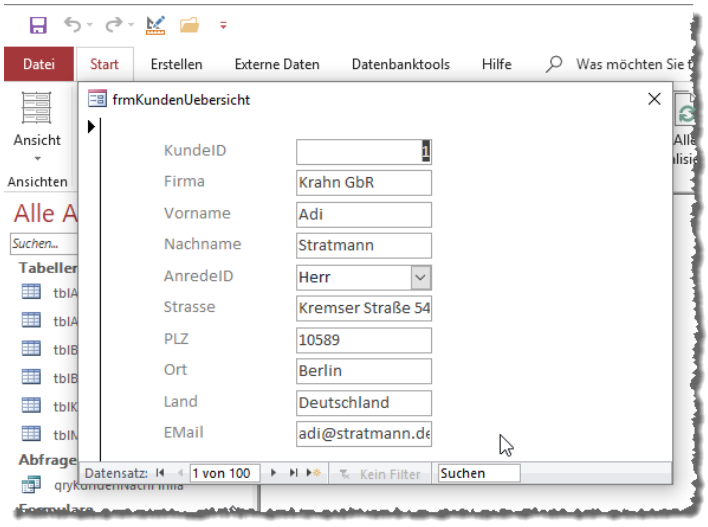


Abbildung 7.9: Formular mit *acDialog* als modalen Dialog öffnen

Welchen Vorteil bringt das? Sie können damit sicherstellen, dass ein Benutzer erst die in diesem Formular durchzuführenden Aufgaben erledigt, bevor er wieder mit anderen Elementen dieser Anwendung arbeiten kann. Das machen wir uns beispielsweise zunutze, wenn wir von einem Übersichtsformular ein Detailformular öffnen.

Wir können dann den Aufruf so gestalten, dass nach dem Schließen des Detailformulars dort durchgeführte Änderungen am Datensatz auch im aufrufenden Formular aktualisiert werden. Mehr dazu finden Sie unter »Daten von Formular zu Formular« ab Seite 219.

Wenn Sie den Wert *acIcon* für den Parameter *WindowMode* verwenden, wird das Formular als minimiertes Element geöffnet. Das sieht dann wie in Abbildung 7.10 aus

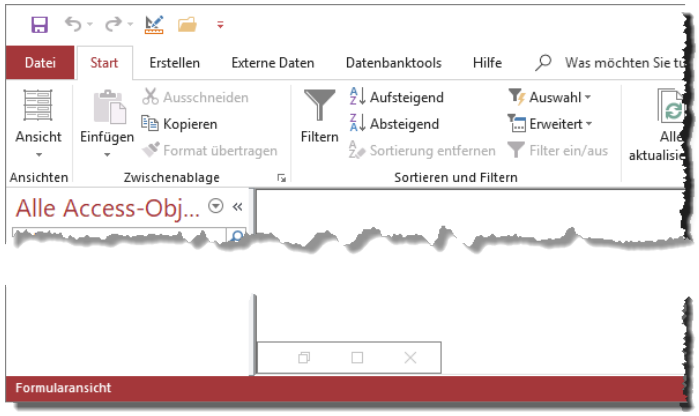


Abbildung 7.10: Ein mit dem Wert *acIcon* für den Parameter *WindowMode* geöffnetes Formular

Schließlich gibt es noch den Wert *acHidden* für den Parameter *WindowMode*. In diesem Fall wird das Formular im ausgeblendeten Modus geöffnet:

```
DoCmd.OpenForm "frmKundenUebersicht", WindowMode:=acHidden
```

Wir würden gern einen Screenshot liefern, wie das aussieht, aber das macht keinen Sinn, denn das Formular ist halt unsichtbar. Welchen Zweck hat das? Es gibt genau einen, der uns eingefallen ist: Sie können mit einem Trick dafür sorgen, dass Sie eine Ereignisprozedur programmieren, die beim Schließen der Access-Datenbank ausgelöst wird.

Diesen Trick beschreiben wir unter »Ereignis beim Schließen der Anwendung« ab Seite 503.

7.3.8 Übergabe von Argumenten beim Öffnen

Den letzten Parameter namens *OpenArgs* können Sie nutzen, um einem Formular beim Aufrufen eine Zeichenkette als Parameter zu übergeben. Ein solcher Aufruf sieht beispielsweise wie folgt aus:

```
DoCmd.OpenForm "frmKundeDetails", OpenArgs:="Öffnungsargument"
```

Wenn Sie im aufgerufenen Formular keinen weiteren Maßnahmen ergreifen, geschieht nichts weiter. Aber der Sinn ist ja gerade, den übergebenen Parameter im aufgerufenen Formular auszulesen und weiterzuverarbeiten.

Ein Beispiel für den Einsatz des Öffnungsarguments finden Sie in »Daten per Öffnungsargument übergeben« ab Seite 220.

7.4 Eigenschaften für den Bearbeitungsmodus

Wenn Sie ein Formular mit der *DoCmd.OpenForm* öffnen, können Sie diesem Befehl einen Wert für den Parameter *DataMode* übergeben. Damit stellen Sie implizit verschiedene Eigenschaften des Formulars ein, die festlegen, wie Sie die Daten im Formular bearbeiten können.

Wenn Sie hier beispielsweise den Wert *acFormReadOnly* angeben, werden die Eigenschaften *Anfügen zulassen*, *Löschen zulassen* und *Bearbeitungen zulassen* auf den Wert *Nein* eingestellt.

Es kann aber durchaus auch vorkommen, dass Sie dem Benutzer zwar das Bearbeiten vorhandener Datensätze und das Löschen von Datensätzen erlauben wollen, aber dieser keine neuen Datensätze anlegen können soll. Dann können Sie die Eigenschaften auch separat einstellen, und zwar auf zwei Arten: über das Eigenschaftenblatt und per VBA etwa beim Öffnen des Formulars.

Mit der Eigenschaft *Anfügen zulassen* können Sie einstellen, ob der Benutzer neue Datensätze im aktuellen Formular einfügen kann. Wenn Sie als dafür sorgen wollen, dass dies nicht möglich ist, stellen Sie diese Eigenschaft gleich im Eigenschaftenblatt auf *Nein* ein.

Mit Löschen zulassen geben Sie an, ob der Benutzer vorhandene Datensätze löschen kann oder nicht. Und mit Bearbeitungen zulassen legen Sie entsprechend fest, ob der Benutzer die vorhandenen Datensätze bearbeiten können soll oder nicht.

Während Sie die Einstellungen über das Eigenschaftenblatt vorab vornehmen können, ist dies natürlich auch dynamisch möglich. So kann es etwa sein, dass ein Benutzer einen Datensatz nur noch ändern können darf, wenn dieser nicht gedruckt ist und nach dem Drucken über eine bestimmte Schaltfläche die Eigenschaft *Bearbeitungen zulassen* auf Nein eingestellt werden soll. Dies können Sie in diesem Fall natürlich nur per VBA einstellen.

7.5 Formulare schließen

Viel einfacher und mit weniger Optionen behaftet ist das Schließen eines Formulars. Dabei benötigen wir im einfachsten Fall einen Klick auf die eingebaute *Schließen*-Schaltfläche oder, wenn es per VBA geschehen soll, einen Aufruf per *DoCmd.Close*-Anweisung. Aber es gibt

7.5.1 Formular per Schaltfläche schließen

Mit der *Schließen*-Schaltfläche rechts oben oder über den entsprechenden Eintrag des Systemmenüfelds hat der Benutzer auch ohne Ihr Zutun zwei Möglichkeiten, ein Formular zu schließen (siehe Abbildung 7.11).

Abbildung 7.11: Schließen eines Formulars

7.5.2 Formular per Tastenkombination schließen

Sie können ein Formular natürlich auch mit der Tastatur schließen. Dazu verwenden Sie die Tastenkombination *Strg + F4*.

7.5.3 Die DoCmd.Close-Methode

Die einfachste Methode, ein Formular per VBA zu schließen, ist die *Close*-Methode des *DoCmd*-Objekts. Dabei müssen Sie zwei Parameter angeben, denn diese Methode dient nicht zur zum Schließen von Formularen, sondern zum Schließen aller Arten von Access-Objekten. Also übergeben wir mit dem ersten Parameter den Typ des zu schließenden Objekts und mit dem zweiten den Namen des zu schließenden Formulars. Wenn Sie also etwa das Formular *frmKundenUebersicht* geöffnet haben und dieses durch die Eingabe der *DoCmd.OpenForm*-Methode im Direktbereich schließen wollen, verwenden Sie die folgende Anweisung:

```
DoCmd.Close acForm, "frmKundenUebersicht"
```

Für den zweiten Parameter dieser Methode können Sie auch einen Ausdruck angeben, der das zu schließende Formular angibt. Wenn zum Beispiel aktuell nur ein Formular geöffnet ist, dann können Sie den ersten Eintrag der *Forms*-Auflistung referenzieren:

```
DoCmd.Close acForm, Forms.Item(0).Name
```

Oder kurzgefasst:

```
DoCmd.Close acForm, Forms(0).Name
```

Sie werden aber nur in den seltensten Fällen in die Verlegenheit kommen, ein Formular über den Direktbereich des VBA-Editors schließen zu müssen – das ist vielleicht mal der Fall, wenn Sie ein Formular unsichtbar geschaltet haben und dieses schließen wollen, ohne es zuvor wieder per VBA oder auf andere Weise einzublenden.

In den meisten Fällen werden Sie ein Formular entweder über eine Schaltfläche schließen, die sich im Formular selbst befindet – beispielsweise mit der Aufschrift *OK* oder *Schließen*. Dazu benötigen wir eine Ereignisprozedur, die durch einen Mausklick auf diese Schaltfläche ausgelöst

wird. Da wir Ereignisprozeduren noch nicht besprochen haben, erklären wir das Schließen des Formulars per benutzerdefinierter Schaltfläche weiter unten unter »Formular per Schaltfläche schließen« ab Seite 168.

Der zweite Einsatzzweck ist das Schließen von anderen Formularen aus, beispielsweise von dem Formular aus, dass das Formular aufgerufen hat. Ein Einsatzzweck, den wir uns ebenfalls weiter unten unter »Daten vom geöffneten Formular aus auslesen« ab Seite 224 ansehen, sieht etwa so aus: Sie öffnen von einem Formular aus ein Detailformular.

Nachdem der Benutzer dieses mit der *OK*-Schaltfläche schließt (beziehungsweise in diesem Fall ausblendet), möchten Sie vom aufrufenden Formular noch Werte aus dem bereits ausgeblendeten Detailformular auslesen. Erst danach soll das ausgeblendete Detailformular endgültig geschlossen werden, was wir dann mit *DoCmd.Close* erledigen.

7.6 Formularereignisse

Eine sehr wichtige Einrichtung für die Programmierung von Formularen sind die Ereignisse. Ereignisse werden durch verschiedene Aktionen ausgelöst – zum Beispiel das Öffnen eines Formulars, das Schließen, das Wechseln des Datensatzes, das Ändern eines Datensatzes, das Löschen eines Datensatzes, ein Mausklick auf eine bestimmte Stelle im Formular und viele mehr. Die Ereignisse schauen wir uns im Detail in »Formularereignisse« ab Seite 235 an.

Das Schöne ist, dass diese Ereignisse nicht einfach nur ausgelöst werden, sondern dass wir diese Ereignisse auch aufgreifen können und sogenannte Ereignisprozeduren anlegen können, die beim Auslösen dieser Ereignisse aufgerufen werden. Dazu wählen wir eines der Ereignisse aus dem Eigenschaftenblatt aus, hinterlegen wir dieses den Wert *[Ereignisprozedur]* und legen dann mit einem Klick auf die Schaltfläche mit den drei Punkten rechts die passende Ereignisprozedur an.

An dieser Stelle ist für uns interessant, wie wir ein solches Formularereignis überhaupt nutzen können. Also legen wir einfach einmal ein solches Ereignis an und überzeugen uns davon, dass es tatsächlich funktioniert.

7.6.1 Formularereignis anlegen

Formularereignisse legen Sie ganz einfach ausgehend vom Eigenschaftenblatt eines Formulars aus an. Bevor wir damit beginnen, erstellen Sie ein neues, leeres Formular in der Entwurfsansicht und speichern dieses unter dem Namen *frmEreignisse*.

Hier wechseln Sie zur Registerseite *Ereignis*. Hier finden Sie alle zur Verfügung stehenden Ereigniseigenschaften für ein Formular. Wir wollen an dieser Stelle einmal ein Ereignis anlegen, das beim Öffnen eines Formulars ausgelöst wird. Dazu klicken wir in die entsprechen-

8 Die Datenblattansicht

Die Datenblattansicht, die Sie für ein Formular durch die Einstellung *Datenblatt* für die Eigenschaft *Standardansicht* erreichen oder durch das Öffnen über den Eintrag *Datenblattansicht* des Kontextmenüs der Entwurfsansicht, bietet einen mächtigen Funktionsumfang. Sie können in dieser Ansicht verschiedene Sortierungen durchführen, Filter setzen, die Reihenfolge der angezeigten Spalten verändern oder Spalten komplett ein- oder ausblenden.

Dieses Kapitel beschreibt die Datenblattansicht der Formulare von Access im Detail. Im Vergleich zur Datenblattansicht von Tabellen oder Abfragen bieten sich eine Menge Vorteile, zum Beispiel durch die Programmierbarkeit per VBA.

8.1 Erstellen eines Formulars in der Datenblattansicht

Das Erstellen eines solchen Formulars ist einfach: Sie legen ein neues Formular an, fügen diesem über die Eigenschaft *Datensatzquelle* eine Tabelle, eine Abfrage oder einen SQL-Ausdruck hinzu und ziehen die gewünschten Felder für die Datenblattansicht in den Detailbereich der Entwurfsansicht.

Außerdem stellen Sie für die Eigenschaft *Standardansicht* den Wert *Datenblatt* ein. Danach brauchen Sie nur noch das Formular zu schließen und erneut zu öffnen, um dieses in der Datenblattansicht zu betrachten.

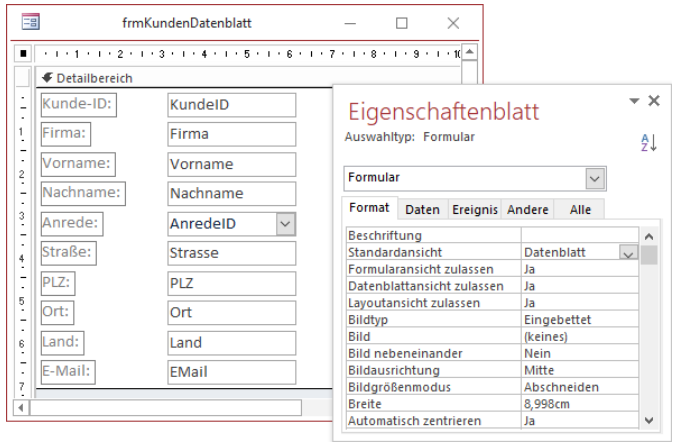


Abbildung 8.1: Einstellen der wichtigsten Eigenschaft für die Datenblattansicht

Danach finden Sie das Formular in der Datenblattansicht vor (siehe Abbildung 8.2). Das Datenblatt hat im Wesentlichen die gleichen Elemente wie die Formularansicht ohne benutzerdefinierte Veränderungen – unten finden Sie die Navigationsschaltflächen, mit denen Sie zum ersten, letzten, vorherigen und nächsten sowie zu einem neuen Datensatz wechseln können.

Durch Eingabe einer Zahl in das Textfeld können Sie auch zu einer bestimmten Position im Datenblatt springen. Rechts daneben zeigt das Formular den aktuellen Filter an und noch ein Stück weiter rechts finden Sie ein *Suchen*-Feld, mit dem Sie einen Suchbegriff eingeben können (siehe »Schnellsuche im Datenblatt« ab Seite 194).

Rechts daneben und am rechten Rand finden Sie die Bildlaufleisten zum Scrollen. Auf der linken Seite ist im Vergleich zum Formular in der Formularansicht nicht nur ein Feld für den Datensatzmarkierer, sondern Sie finden dort für jeden Datensatz eines.

8.2 Funktionen des Datenblatts

Wenn Sie oben auf den nach unten zeigenden Pfeil rechts der Spaltenüberschriften klicken, klappt ein Menü auf, das diverse Such- und Sortiermöglichkeiten anbietet (siehe »Sortieren im Datenblatt« ab Seite 188 und »Filtern im Datenblatt« ab Seite 189).

Sie können aber auch mit der rechten Maustaste auf einen Spaltenkopf klicken und erhalten weitere Optionen (siehe »Kontextmenü der Spaltenköpfe« ab Seite 192).

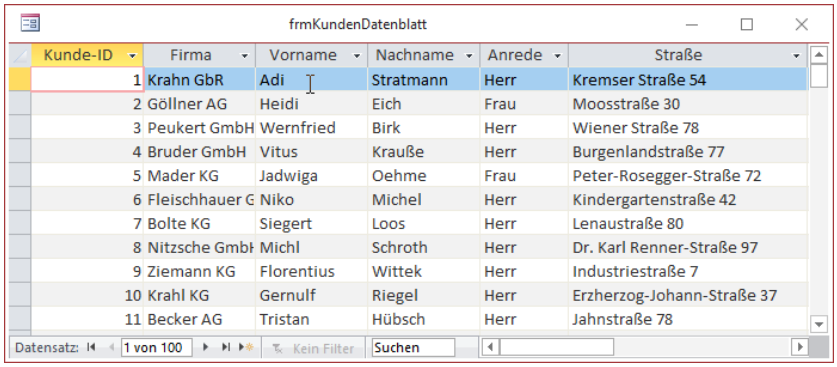


Abbildung 8.2: Das Formular in der Datenblattansicht

8.3 Sortieren im Datenblatt

Wenn Sie mit der rechten Maustaste rechts neben einer Spaltenüberschrift auf die Taste mit dem Pfeil nach unten drücken, erscheint ein Menü mit Befehlen zum Sortieren und Filtern der

Datensätze des Formulars. Die beiden Befehle *Von A bis Z sortieren* und *Von Z bis A sortieren* sollten selbsterklärend sein (siehe Abbildung 8.3). Interessant ist allerdings, was geschieht, wenn Sie die Sortierung für mehrere Felder aktivieren.

Wenn wir also zuerst etwa den Befehl *Von A bis Z sortieren* für das Feld *Firma* auswählen und dann für das Feld *Nachname*, wir wirkt sich dies dann auf die Sortierung aus? Dann sortiert das Formular die Datensätze zuerst aufsteigend nach dem zuletzt gewählten Feld und erst dann aufsteigend nach dem zuerst gewählten Feld. Wie intern die Sortierreihenfolge festgelegt wird, können Sie auch per VBA herausfinden. Dafür geben Sie im Direktbereich den folgenden Befehl ein:

```
? Screen.ActiveForm.OrderBy
[tblKunden].[Nachname], [tblKunden].[Firma]
```

Mit *Screen.ActiveForm* referenzieren wir das aktive Formular und *OrderBy* enthält die Definition der Sortierreihenfolge für das Formular. Diese liefert einen Ausdruck, der in einer SQL-SELECT-Anweisung für die *ORDER BY*-Klausel eingesetzt werden könnte.

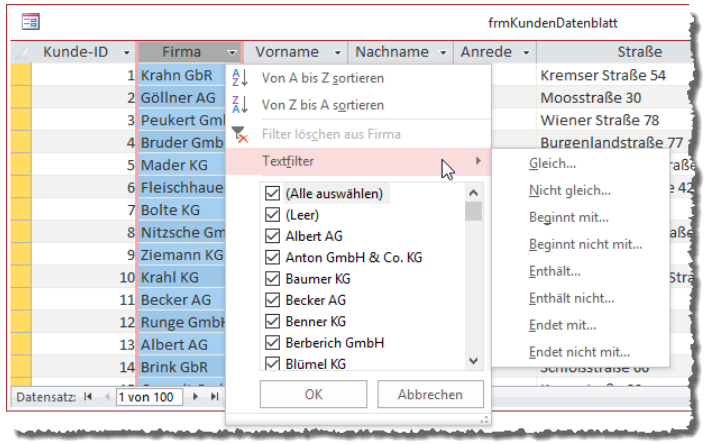


Abbildung 8.3: Befehle zum Sortieren und Filtern des Datenblatts

8.4 Filtern im Datenblatt

Außerdem finden wir im gleichen Menü noch weitere Einträge, die alle zum Filtern der Datensätze vorgesehen sind. Der oberste Eintrag namens *Filter löschen aus <Feldname>* entfernt die vorhandenen Filter für dieses Feld. Aber zuerst wollen wir ja erst einmal einen Filter anlegen. Dazu gibt es verschiedene Möglichkeiten. Über das Untermenü *Textfilter* können Sie einen passenden Eintrag auswählen, der angibt, wie Sie in dem markierten Feld nach dem anschließend einzugebenden Filterausdruck suchen:

» *Gleich...*: Liefert alle Datensätze, in denen der Feldinhalt exakt mit dem Vergleichsausdruck übereinstimmt.

» *Nicht gleich...*: Liefert alle Datensätze, die einen anderen als den angegebenen Ausdruck enthalten.

» *Beginnt mit.../Endet mit...*: Liefert alle Datensätze, deren Feldinhalt mit dem angegebenen Ausdruck beginnt oder endet.

» *Beginnt nicht mit.../Endet nicht mit...*: Liefert alle Datensätze, deren Feldinhalt nicht mit dem angegebenen Ausdruck beginnt oder endet.

» *Enthält.../Enthält nicht...*: Liefert alle Datensätze deren Feldinhalt den Ausdruck an irgendeiner Stelle enthält oder an keiner Stelle.

In Abbildung 8.4 suchen wir zum Beispiel nach allen Datensätzen, deren Feld *Firma* den Ausdruck *GmbH* enthält.

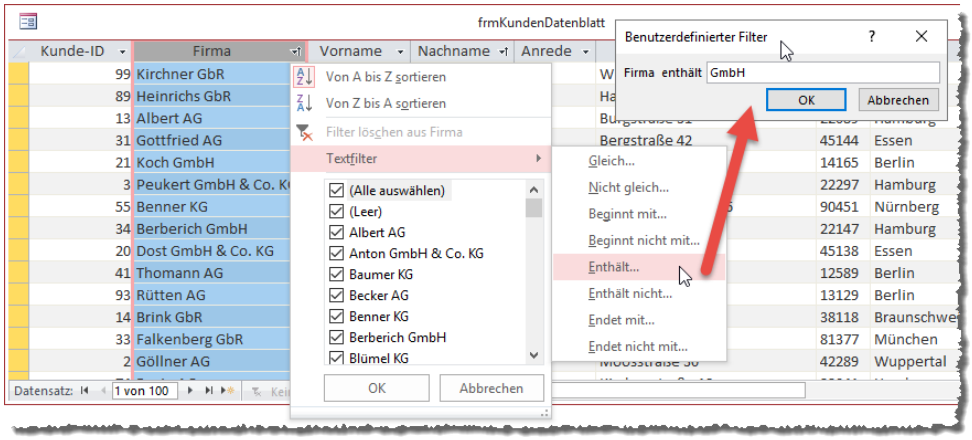


Abbildung 8.4: Eingabe eines Textfilters

Auch hier können wir uns den Filterausdruck, den Access zusammensetzt, im VBA-Editor im Direktbereich ansehen. Dazu geben wir den folgenden Befehl ein:

```
? Screen.ActiveForm.Filter
([tblKunden].[Firma] Like "*GmbH*")
```

Was geschieht, wenn wir noch einen Filter für ein weiteres Feld festlegen? Wird dieser mit *Und* oder mit *Oder* verknüpft? Wir fügen noch einen Filter für das Feld *Vorname* hinzu, der alle Datensätze liefern soll, die mit dem Buchstaben *A* beginnen.

Wie in Abbildung 8.5 zu sehen, werden nun zwei Felder mit dem Filter-Symbol in der Spaltenüberschrift markiert. Außerdem zeigt ein Bereich neben den Navigationsschaltflächen nun an, dass es sich um eine gefilterte Ansicht handelt.

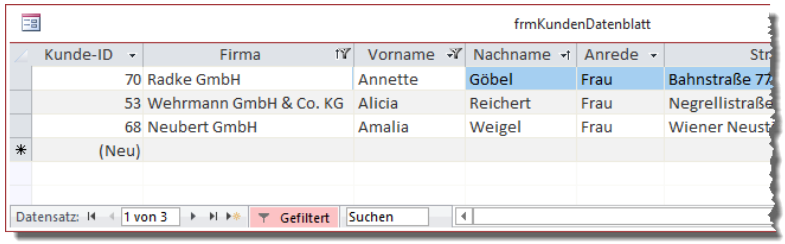


Abbildung 8.5: Datenblatt mit zwei Filtern

Die Abfrage des aktuellen Filterausdrucks liefert nun folgendes Ergebnis:

```
? Screen.ActiveForm.Filter
(([tblKunden].[Firma] Like "*GmbH*")) AND ([tblKunden].[Vorname] Like "A*")
```

Die Filterausdrücke werden also mit *Und* verknüpft. Wir können auch zwei oder mehr Filterbedingungen für ein einziges Feld anlegen. Wenn wir noch einen zweiten Filter für das Feld *Vorname* definieren, sieht das etwa wie folgt aus:

```
? Screen.ActiveForm.Filter
((([tblKunden].[Firma] Like "*GmbH*")) AND ([tblKunden].[Vorname] Like "A*")) AND
([tblKunden].[Vorname] Like "*a")
```

8.4.1 Auswahl bestimmter Werte

Im unteren Bereich des Menüs, das nach einem Mausklick auf den *Nach unten*-Pfeil aufgeklappt wird, finden Sie noch einige Einträge mit einem Kontrollkästchen (siehe Abbildung 8.6). Hier können Sie einzelne Einträge abwählen oder auch alle Einträge auf einen Schlag aus der Auswahl nehmen – und zwar durch Deaktivieren der Option *(Alle auswählen)*. Danach könnten Sie einzelne Einträge wieder hinzufügen. Die Option *(Leer)* dient dazu, die Anzeige von leeren Einträgen entweder zu aktivieren oder auch dazu, diese herauszufiltern und nur Datensätze mit Werten in diesem Feld anzuzeigen.

Die Zusammenstellung der resultierenden Filterausdrücke erfolgt dabei durchaus effizient. Wenn Sie etwa alle Einträge deaktivieren und fünf Werte aktivieren, sieht der Filterausdruck wie folgt aus:

```
? Screen.ActiveForm.Filter
([tblKunden].[Firma] In ("Albert AG","Anton GmbH & Co. KG","Baumer KG","Becker
AG","Benner KG"))
```

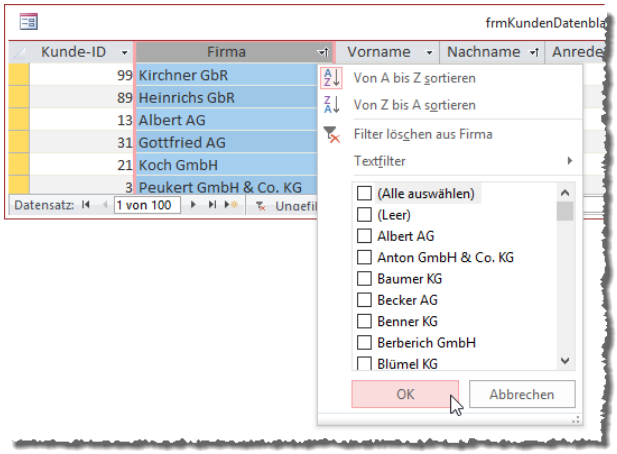


Abbildung 8.6: Abwählen aller Einträge

Hier werden also nicht etwa blind mit *AND* verknüpfte Kriterien aneinandergehängt, sondern auch einmal die *IN*-Klausel verwendet.

Wenn Sie für ein Feld einen solchen Filter definiert haben und dann das Menü für ein anderes Feld öffnen, finden Sie dort zur Auswahl nur noch solche Werte vor, die noch in den angezeigten Elementen vorkommen.

8.5 Kontextmenü der Spaltenköpfe

Wenn Sie mit der rechten Maustaste auf einen der Spaltenköpfe klicken, erscheint das Kontextmenü aus Abbildung 8.7. Hier finden Sie die bereits bekannten Befehle zum Einstellen der Sortierung nach den Werten der aktuellen Spalte, aber auch noch weitere Befehle, die das zuvor beschriebene Menü nicht enthält.

Damit können Sie etwa eine komplette Spalte löschen. Damit sollten Sie vorsichtig sein, denn diese Funktion arbeitet ohne vorherige Rücksprache und löscht das Feld aus dem zugrunde liegenden Formularentwurf.

Wenn Sie eine Spalte nur temporär ausblenden möchten, um beispielsweise die Übersicht zu optimieren, verwenden Sie den Befehl *Felder ausblenden*. Wenn Sie das Feld wieder einblenden möchten, können Sie dann einfach mit dem Befehl Felder wieder einblenden den Dialog aus Abbildung 8.8 öffnen. Hier können Sie mehrere Felder gleichzeitig ein- oder ausblenden.

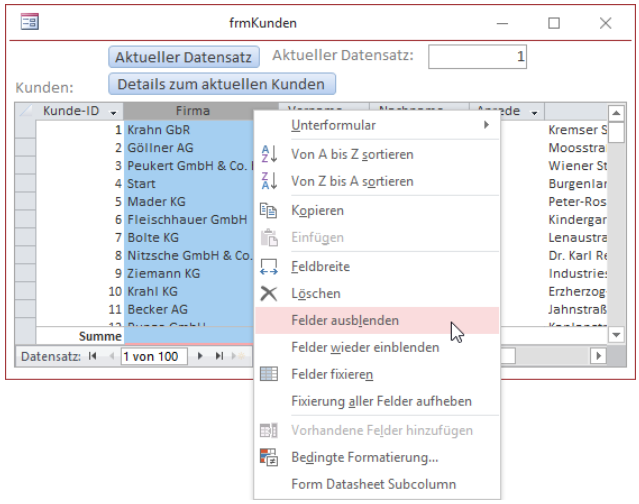


Abbildung 8.7: Kontextmenü eines Spaltenkopfes

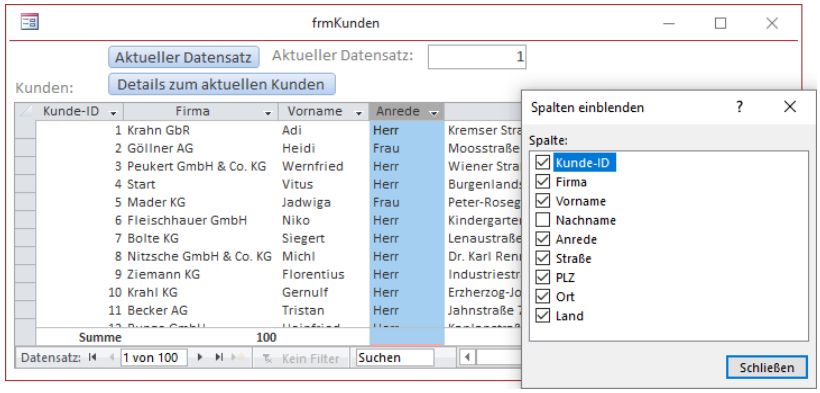


Abbildung 8.8: Ein- und Ausblenden von Feldern

Mit dem Befehl *Felder fixieren* können Sie die aktuell markierten Felder auf der linken Seite des Formulars fixieren. Das bedeutet zwei Dinge: Erstens werden diese Felder dann ganz nach links verschoben. Und zweitens werden diese dann durch Betätigen der horizontalen Bildlaufleiste nicht mehr nach links aus dem sichtbaren Bereich geschoben, sondern die rechts neben den nun fixierten Spalten verschwinden dann aus dem sichtbaren Bereich, werden also quasi »unter« die fixierten Spalten geschoben. Die Fixierung heben Sie mit dem Befehl *Fixierung aller Felder aufheben* wieder auf. Die fixierten Felder werden dann allerdings nicht wieder an die vorherige Position verschoben.

8.6 Schnellsuche im Datenblatt

Wenn Sie einen Suchbegriff in das Textfeld *Suchen* rechts neben dem Navigationsschaltflächen eingeben, startet Access direkt nach Eingabe des ersten Buchstaben mit der Suche und markiert direkt die erste Fundstelle (siehe Abbildung 8.9).

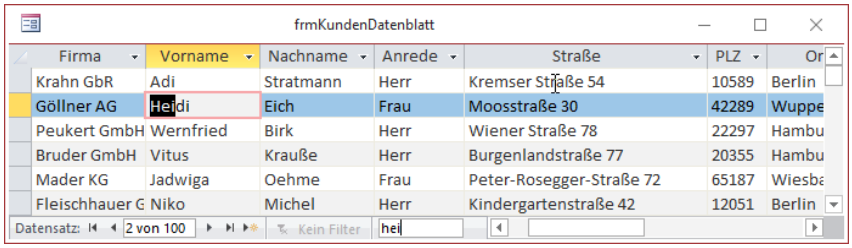


Abbildung 8.9: Schnelle Suche im Datenblatt

Auf diese Weise können Sie dann durch die Einträge navigieren, die einen bestimmten Suchbegriff enthalten. Sie können auch noch einen Suchen-Dialog öffnen – mehr dazu gleich unter »Ribbon-Befehle für das Datenblatt« ab Seite 194.

8.7 Ribbon-Befehle für das Datenblatt

Wenn Sie ein Formular in der Datenblattansicht öffnen, aktiviert das Ribbon auf der Seite *Start* einige Schaltflächen, die sonst nicht aktiviert sind (siehe Abbildung 8.10). Wir schauen uns einmal an, welche Sie davon sinnvoll nutzen können.

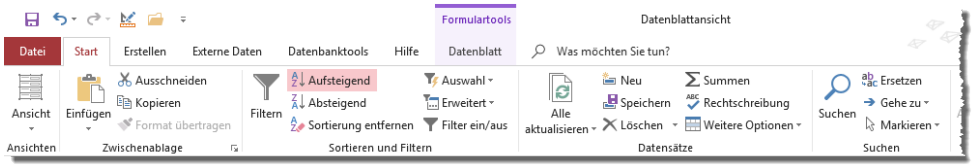


Abbildung 8.10: Ribbon-Befehle für das Datenblatt

8.7.1 Sortieren per Ribbon

Zunächst einmal finden Sie im Bereich *Sortieren und Filtern* einige Befehle, die Sie auch über die *Nach unten*-Schaltfläche neben den Spaltenüberschriften finden können. Die Schaltfläche *Filtern* etwa blendet für die aktuell markierte Spalte den gleichen Dialog ein, den Sie auch per Klick auf die *Nach unten*-Schaltfläche aktivieren können. Mit *Sortierung entfernen* löschen Sie

die Sortierung aus der aktuell markierten Spalte. Die Schaltflächen *Aufsteigend* und *Absteigend* sortieren das aktuelle Feld in der angegebenen Reihenfolge.

8.7.2 Auswahl basierend auf der aktuellen Auswahl

Wenn Sie im Datenblatt einen bestimmten Teil des Textes markiert haben und dann im Ribbon auf die Schaltfläche *Auswahl* klicken, bietet Access einige Vorschläge für Abfragen an, beispielsweise *Beginnt mit '<Markierter Text>'* (siehe Abbildung 8.11).

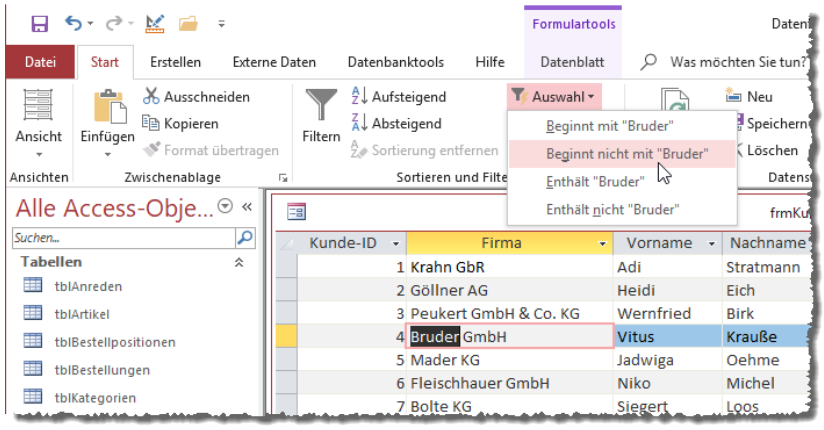


Abbildung 8.11: Auswahlfilter für den aktuell markierten Wert im Datenblatt

8.7.3 Formularbasierter Filter

Sehr praktisch ist auch der Befehl *Erweitert|Formularbasierter Filter*. Nach der Auswahl dieses Eintrags zeigt Access die Datenblattansicht des Formulars wie in Abbildung 8.12 an. Hier können Sie aus der ersten Zeile für alle Felder einen der vorhandenen Werte auswählen oder auch einen neuen Vergleichswert eingeben, gegebenenfalls auch mit einem Platzhalter wie dem Sternchen (*).

Sie können für jedes Feld einen Vergleichswert auswählen und auch noch per *Oder* verknüpft weitere Vergleiche anlegen, indem Sie unten auf die Registerlasche *Oder* klicken. Wenn Sie den Filter ausführen wollen, wählen Sie den Eintrag *Filter/Sortierung anwenden* aus dem Kontextmenü aus (siehe Abbildung 8.13).

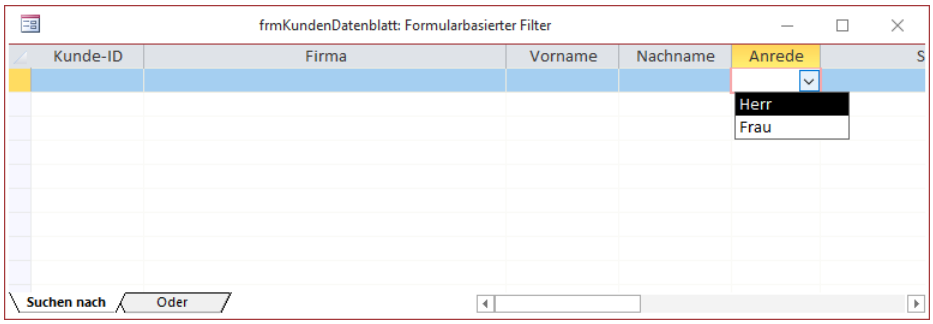


Abbildung 8.12: Eingeben eines formularbasierten Filters

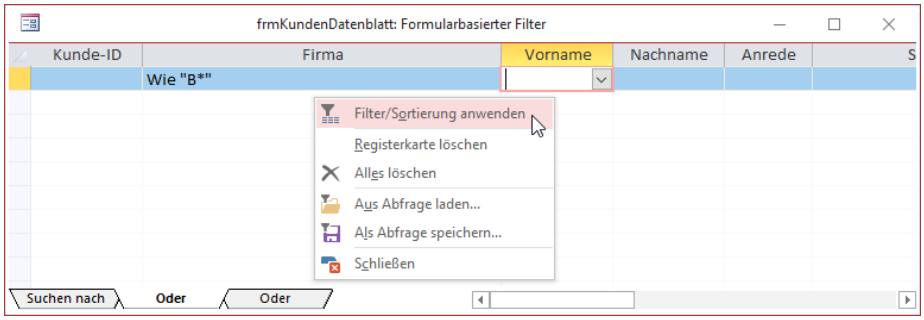


Abbildung 8.13: Anwenden des Filters

Wenn Sie den Ribbon-Befehl *Spezialfilter/Sortierung* wählen, wird die Entwurfsansicht einer Abfrage angezeigt. Sie können dann nach Belieben Kriterien festlegen, die auf das aktuell geöffnete Formular in der Datenblattansicht angewendet werden.

Hier werden die zuvor über die anderen Methoden angelegten Filter direkt berücksichtigt und können durch weitere Kriterien ergänzt werden (siehe Abbildung 8.14).

Über den Kontextmenü-Eintrag *Aus Abfrage laden* können Sie auch die Kriterien, die Sie in einer gespeicherten Abfrage festgelegt haben, auf die Datenblattansicht anwenden. Nach dem Aufruf dieses Befehls zeigt Access einen Dialog an, mit dem Sie eine der gespeicherten Abfragen auswählen können.

Access wendet dann den für den *WHERE*-Teil der gespeicherten Abfrage angegebenen Ausdruck auf die Datenblattansicht an.

9 Die Endlosansicht

Die Endlosansicht unterscheidet sich von der Datenblattansicht und der Formularansicht, indem es eine Mischung aus beiden abbildet. Während die Datenblattansicht zwar mehrere Datensätze untereinander abbildet, gibt es dort einige Einschränkung etwa was das Design angeht – die Daten werden automatisch strikt in Zeilen und Spalten aufgeteilt.

Außerdem können gibt es keinen Kopf- oder Fußbereich. In der Formularansicht hingegen haben Sie viel mehr Freiheiten und können die Steuerelemente anordnen, wie Sie möchten, aber dafür zeigt diese Ansicht immer nur einen Datensatz gleichzeitig an. Die Endlosansicht kombiniert beide in der Art, dass es alle Möglichkeiten der Formularansicht bietet, aber den Detailbereich für jeden Datensatz der Datensatzquelle je einmal abbildet und somit ein Scrollen durch die einzelnen Datensätze erlaubt.

Das heißt, Sie können dort sowohl einen größeren Raum als nur eine Zeile für die Daten einräumen als auch auf alle Steuerelemente zugreifen, die Sie auch in der Formularansicht einsetzen können – also beispielsweise auch auf die Schaltfläche. Das bedeutet allerdings nicht, dass Sie nicht versuchen sollten, die Höhe des Detailbereichs zu beschränken, denn sonst sehen Sie ja dennoch nicht viel mehr Daten als in der Formularansicht.

9.1 Kunden im Endlosformular

Für dieses Beispiel sollen die Datensätze der Tabelle *tblKunden* in der Endlosansicht eines Formulars angezeigt werden. Dazu führen Sie zunächst die folgenden Schritte durch:

» Legen Sie ein neues, leeres Formular an.

» Stellen Sie die Eigenschaft *Datenherkunft* auf *tblKunden* ein.

» Ändern Sie die Eigenschaft *Standardansicht* auf *Endlosformular* (siehe Abbildung 9.1).

Das waren die Vorbereitungen. Nun geht es an die eigentliche Arbeit: das Einfügen und Ausrichten der Steuerelemente. Dazu müssen Sie grundsätzlich wissen, dass ein Formular in der Endlosansicht den Detailbereich und die darin enthaltenen Steuerelemente für jeden Datensatz einmal anzeigt. Wie viele Datensätze gleichzeitig sichtbar sind, ohne dass Sie scrollen müssen, hängt von mehreren Faktoren ab.

Der erste ist die Höhe des für die Anzeige der Detailbereiche für die einzelnen Datensätze verfügbare Platz, der zweite die Höhe des Detailbereichs für einen Datensatz. Die Höhe für die Detailbereiche entspricht der Höhe des Formulars minus der Höhe von Formularkopf und Formularfuß, sofern diese beiden Bereiche eingeblendet sind.

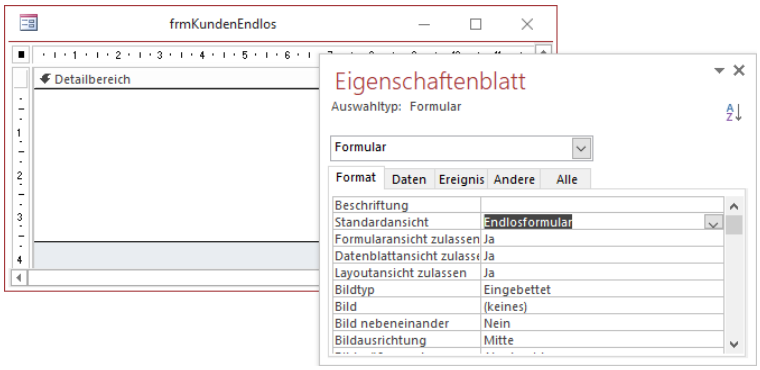


Abbildung 9.1: Einstellen der Eigenschaft *Standardansicht* auf *Endlosformular*

Um den Formularkopf und -fuß einzublenden, klicken Sie mit der rechten Maustaste auf die Titelzeile des Detailbereichs (siehe Abbildung 9.2).

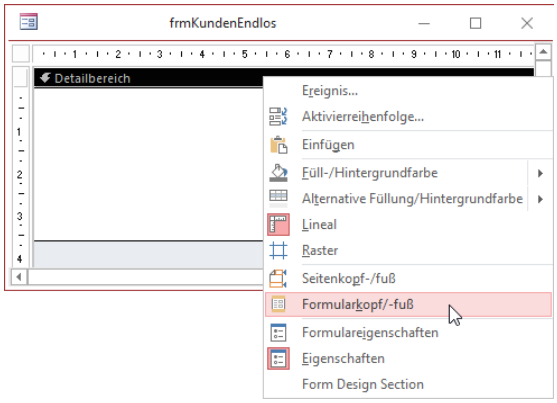


Abbildung 9.2: Aktivieren des Kopf- und Fußbereichs

Danach erscheinen die beiden Bereiche über- und unterhalb des Detailbereichs (siehe Abbildung 9.3).

Damit Sie das Zusammenspiel der einzelnen Bereiche genauer kennenlernen, führen Sie nun die folgenden Schritte aus:

» Fügen Sie im Formularkopf ein Beschriftungsfeld mit dem Text *Kundenübersicht* ein (die Schriftgröße können Sie auf einen größeren Wert einstellen).

» Legen Sie im Formularfuß eine Schaltfläche mit der Beschriftung *OK* und dem Namen *cmdOK* an.

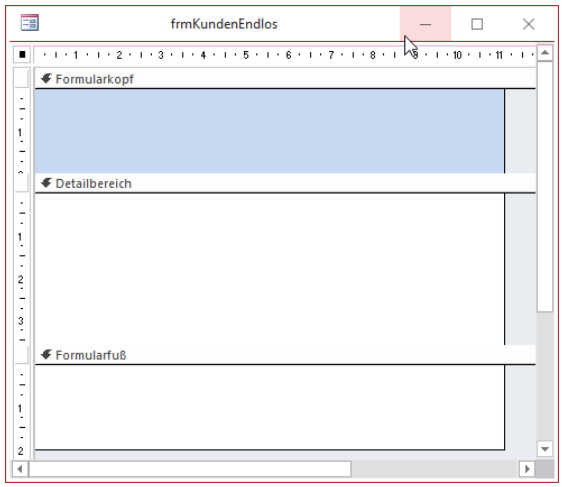


Abbildung 9.3: Formular mit Kopf- und Fußbereich

Fügen Sie im Detailbereich vorerst zwei Felder der in der Datenherkunft angegebenen Tabelle hinzu. Dazu aktivieren Sie am einfachsten die Feldliste mit der Schaltfläche mit der Tastenkombination *Alt + F8* und ziehen die zwei Felder *KundeID* und *Firma* in den Detailbereich (siehe Abbildung 9.4).

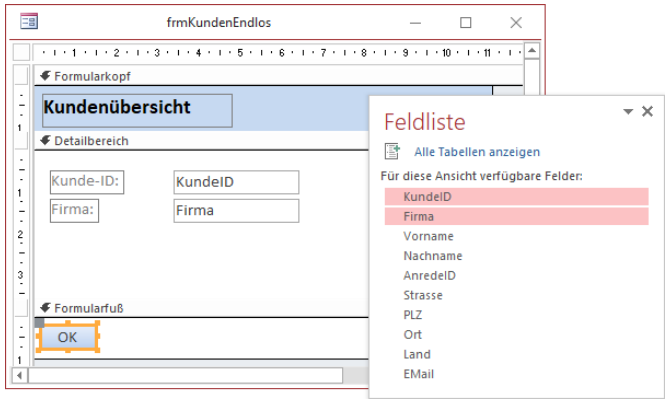


Abbildung 9.4: Einfügen der Felder in den Detailbereich

Nach einem Wechsel in die Formularansicht sieht das Formular wie in Abbildung 9.5 aus. Die Inhalte von Formularkopf und Formularfuß werden oben und unten angezeigt, dazwischen erscheint der Detailbereich so oft, wie es der verbleibende Platz zulässt.

Abbildung 9.5: Formular in der Endlosansicht

Wenn Sie einen Datensatz etwa zum Löschen markieren möchten, klicken Sie wie in der Datenblattansicht auf den Datensatzmarkierer links und betätigen die *Entfernen*-Taste. Zwischen den Datensätzen können Sie wie gewohnt mit den Navigationsschaltflächen im unteren Bereich navigieren.

Hier tritt auch gleich der entscheidende Unterschied zur Datenblattansicht zutage: Sie können die Steuerelemente zur Anzeige der Feldinhalte des Datensatzes beliebig innerhalb des Detailbereichs anordnen. Und es wird noch besser: Sie können sogar Schaltflächen im Detailbereich unterbringen, die für jeden einzelnen Datensatz angezeigt werden. Damit können Sie dem Benutzer eine *Löschen*-Schaltfläche anbieten, mit der ein Datensatz ohne vorheriges Selektieren mit dem Datensatzmarkierer gelöscht werden kann.

9.2 Daten in Tabellenform im Endlosformular

Wenn Sie die übrigen Felder der Tabelle *tblKunden* zum Detailbereich der Entwurfsansicht hinzufügen, wird es dort etwas unübersichtlich (wir beschränken uns aus Gründen der Übersichtlichkeit auf die beiden Felder *KundeID* und *Firma*). Das ist logisch, denn die Beschriftungsfelder und die eigentlichen Steuerelemente zur Anzeige der Feldinhalte nehmen Platz weg. Zumindest die Beschriftungsfelder können wir aber viel besser anordnen – nämlich im Formularkopf.

Leider lassen sich die Beschriftungsfelder, beispielsweise das mit der Beschriftung *KundeID*, nicht ohne das dazugehörige gebundene Steuerelement in den Formularkopf verschieben. Es gibt jedoch einen Trick, wie Sie es doch schaffen: Markieren Sie eines oder mehrere Beschriftungsfelder. Schneiden Sie diese mit der Tastenkombination *Strg + X* aus, klicken Sie in den Formularkopf-Bereich und fügen Sie die Steuerelemente dort mit *Strg + V* wieder ein (eine noch elegantere Methode, um diese Arbeit zu erledigen, finden Sie unter »Steuerelemente mit »Anordnen« positionieren« ab Seite 70).

Danach ist ein wenig Feinarbeit nötig, um die Steuerelemente anzuordnen. Tipps zu dieser Aufgabe finden Sie unter »Steuerelemente positionieren und anpassen« ab Seite 59. Anschließend könnte das Formular in der Entwurfsansicht wie in Abbildung 9.6 aussehen.

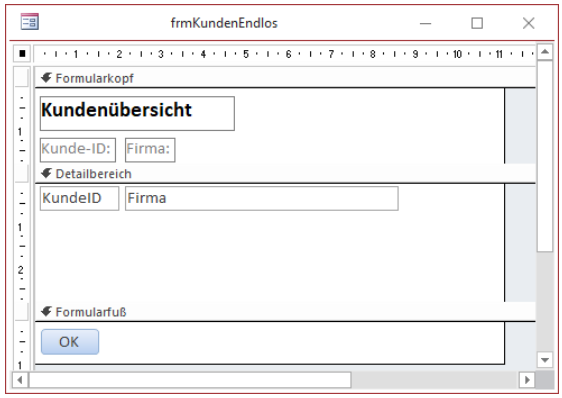


Abbildung 9.6: Aufteilung von Bezeichnungsfeldern und gebundenen Steuerelementen auf Formularkopf und Detailbereich

Wenn Sie die Steuerelemente angeordnet haben, optimieren Sie noch die Höhe des Detailbereichs. Dies erledigen Sie, indem Sie entweder die Eigenschaft *Höhe* auf den gewünschten Wert einstellen oder indem Sie die Maus genau am unteren Rand des Detailbereichs platzieren und diesen nach dem Erscheinen des Icons aus Abbildung 9.7 nach oben oder unten verschieben.

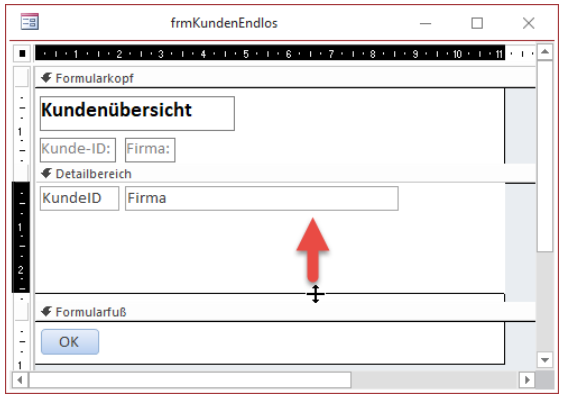


Abbildung 9.7: Verkleinern des Detailbereichs

Ein Wechsel in die Formularansicht liefert das Formular aus Abbildung 9.8.

10 Daten von Formular zu Formular

Beispieldatenbank: *FormulareProgrammieren\FormulareProgrammieren.accdb*

Ein wichtiges Feature für die Programmierung von Datenbankanwendungen ist die Übergabe von Daten zwischen verschiedenen Formularen. Das einfachste Beispiel ist ein Übersichtsformular, das alle Datensätze einer Tabelle anzeigt – beispielsweise die Kunden –, und mit einem Doppelklick auf einen der Kunden das Formular mit den Details zu diesem Kunden öffnet.

Wir müssen in diesem Fall die ID des Kunden vom Übersichtsformular an das Detailformular übergeben, damit dieses weiß, zu welchem Kunden es die Detaildaten anzeigen soll. Oder vielleicht möchten Sie einen neuen Artikel für eine bestimmte Kategorie anlegen. Dann wollen Sie vom Übersichtsformular die ID der Kategorie an das Detailformular zum Anlegen des neuen Artikels übergeben, damit dort die Kategorie schon voreingestellt werden kann.

Auch andersherum sollen manchmal Daten übergeben werden. So möchten Sie nach dem Anlegen eines neuen Artikels vielleicht wissen, welcher Artikel angelegt wurde und diesen gleich im Übersichtsformular markieren.

Oder Sie haben einen neuen Eintrag für ein Kombinationsfeld in einem separaten Formular angelegt und möchten diesen nun direkt im Kombinationsfeld auswählen. Hierzu benötigen Sie genau wie beim Übergeben an das aufgerufene Formular spezielle Techniken. All diese Techniken stellen wir Ihnen in diesem Kapitel vor.

10.1 Daten per WhereCondition übergeben

Eine einfache Möglichkeit, eine Information von einem Formular zum anderen zu übergeben, ist das Festlegen einer Bedingung für die Festlegung der anzuzeigenden Daten mit dem Parameter *WhereCondition* der *DoCmd.OpenForm*-Methode (siehe auch »Formular mit WhereCondition öffnen« ab Seite 163).

Das Formular aus Abbildung 10.1 enthält ein Unterformular, das die Einträge der Tabelle *tblKunden* in der Datenblattansicht anzeigt. Darunter finden wir eine Schaltfläche, welche die Details zum aktuell markierten Datensatz in einem neuen Formular anzeigen soll.

Für das Formular hinterlegen Sie die folgende Ereignisprozedur. Diese ermittelt zunächst den Wert des Feldes *KundeID* für den aktuellen Datensatz im Unterformular und speichert diesen in der Variablen *lngKundeID*. Dabei verwenden wir die *Nz*-Funktion, um den Fehler abzufangen, der auftritt, wenn der Datensatzzeiger auf dem neuen, leeren Datensatz steht, in dem das Feld *KundeID* noch keinen Wert enthält, also *Null* ist.

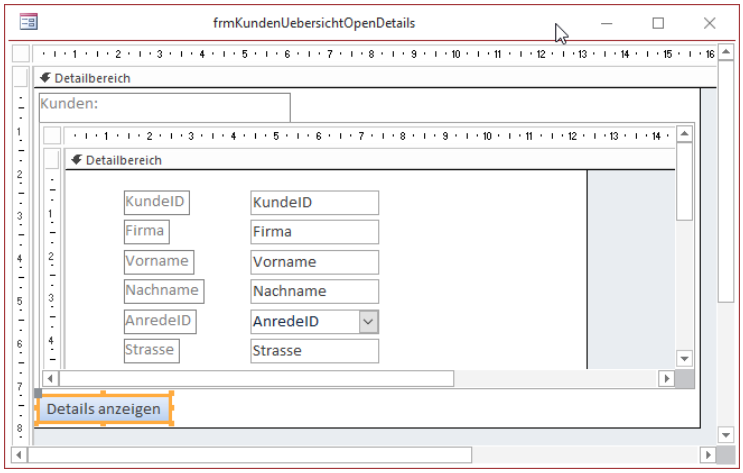


Abbildung 10.1: Formular mit Unterformular zur Anzeige der Kundenübersicht und mit Schaltfläche zum Öffnen der Kundendetails

In diesem Fall gibt *Nz* den Wert des zweiten Parameters zurück, also *0*. Ist *lngKundeID* danach ungleich *0*, ruft die Prozedur das Formular *frmKundeDetails* auf und übergibt mit dem Parameter *WhereCondition* eine Bedingung, die etwa *KundeID = 123* lauten könnte:

```
Private Sub cmdDetailsAnzeigen_Click()
    Dim lngKundeID As Long
    lngKundeID = Nz(Me!sfmKundenUebersichtOpenDetails.Form!KundeID, 0)
    If Not lngKundeID = 0 Then
        DoCmd.OpenForm "frmKundeDetails", WhereCondition:="KundeID = " & lngKundeID
    End If
End Sub
```

Damit wird das Formluar *frmKundeDetails* dann so geöffnet, dass es ausschließlich den Datensatz mit dem mit *lngKunde* übergebenen Primärschlüsselwert anzeigt (siehe Abbildung 10.2).

10.2 Daten per Öffnungsargument übergeben

Der Parameter *OpenArgs* der *DoCmd.OpenForm*-Methode erlaubt die Übergabe einer Zeichenkette als Parameter an das aufgerufene Formular.

Von diesem Formular aus können Sie den übergebenen Wert mit der VBA-Eigenschaft *OpenArgs* des Formular-Objekts selbst auslesen.

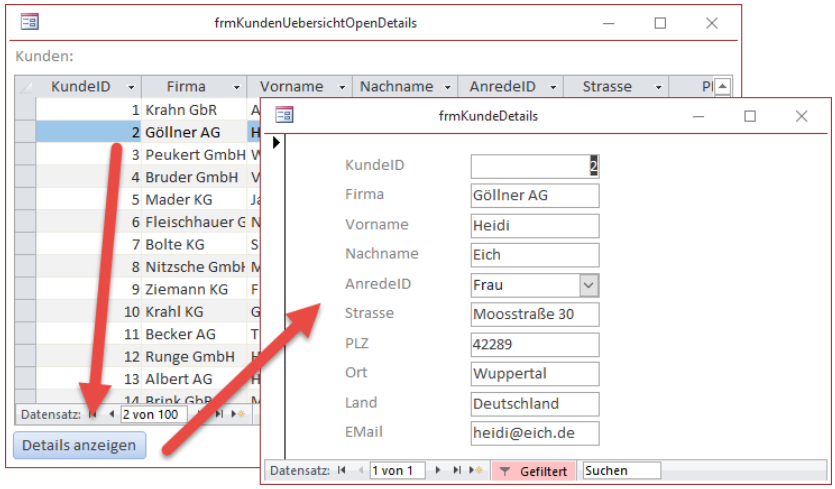


Abbildung 10.2: Aufruf eines Detailformulars mit *WhereCondition*

10.2.1 Einen Wert per Öffnungsargument übergeben

Um dies zu demonstrieren, erstellen wir zwei Formulare. Das erste heißt *frmOpenArgsSender*, das zweite *frmOpenArgsEmpfaenger*. Im Formular *frmOpenArgsSender* legen wir das Textfeld *txtOpenArgs* an und eine Schaltfläche namens *cmdOpenArgsUebergeben* (siehe Abbildung 10.3). Für die Schaltfläche hinterlegen wir die folgende Ereignisprozedur:

```
Private Sub cmdOpenArgsUebergeben_Click()
    DoCmd.OpenForm "frmOpenArgsEmpfaenger", OpenArgs:=Me!txtOpenArgs
End Sub
```

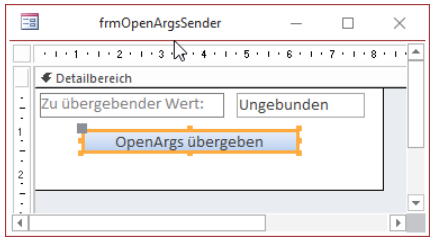


Abbildung 10.3: Formular zum Übergeben eines Öffnungsarguments

Das Formular aus *frmOpenArgsEmpfaenger* soll das Öffnungsargument schließlich entgegennehmen und in das Textfeld *txtOpenArgs* schreiben (siehe Abbildung 10.4).

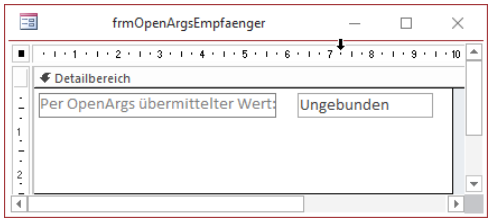


Abbildung 10.4: Formular zum Empfangen des Öffnungsarguments

Damit das gelingt, fügen wir dem Formular *frmOpenArgsEmpfaenger* ein Ereignis für die Ereignisprozedur *Beim Öffnen* hinzu.

Diese soll das übergebene Argument auslesen. Das gelingt mit der Eigenschaft *OpenArgs* des Formular-Objekts selbst, das wir mit dem Schlüsselwort *Me* referenzieren können. Die Prozedur sieht wie folgt aus:

```
Private Sub Form_Open(Cancel As Integer)
    Me!txtOpenArgs = Me.OpenArgs
End Sub
```

Wenn wir nun einen Text in das Textfeld *txtOpenArgsSender* schreiben und auf die Schaltfläche *cmdOpenArgsUebergeben* klicken, sieht das Ergebnis wie in Abbildung 10.5 aus.

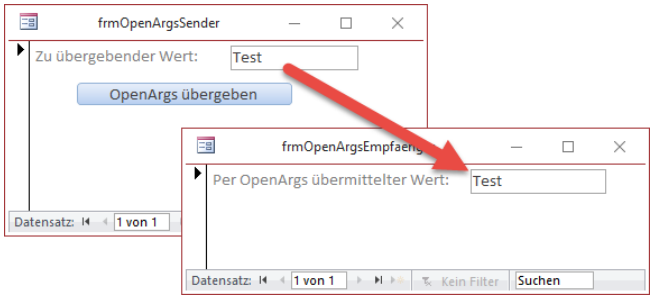


Abbildung 10.5: Übergabe eines Textes als Öffnungsparameter

Es gibt allerdings einen kleinen Haken: Wenn wir das Textfeld im aufrufenden Formular leer lassen, erhalten wir beim Öffnen des zweiten Formulars eine Fehlermeldung (siehe Abbildung 10.6).

Der Fehler tritt auf, weil der Parameter *OpenArgs* den Datentyp *Text* hat, wir aber den Inhalt eines leeren Textfeldes übergeben wollen, also *Null*. Damit das funktioniert, müsste der Parameter *OpenArgs* den Datentyp *Variant* haben. Da das nicht geht, müssen wir den eventuell auftretenden *Null*-Wert in eine leere Zeichenkette umwandeln.

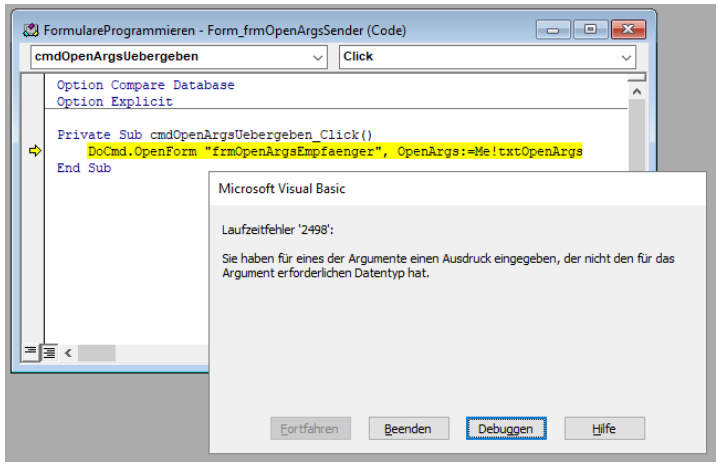


Abbildung 10.6: Fehler beim Versuch, einen *Null*-Wert als Öffnungsargument zu übergeben

Das erledigen wir durch die Funktion *Nz* mit einer leeren Zeichenkette als zweiten Parameter:

```
Private Sub cmdOpenArgsUebergeben_Click()
    DoCmd.OpenForm "frmOpenArgsEmpfaenger", OpenArgs:=Nz(Me!txtOpenArgs, "")
End Sub
```

Auf diese Weise wird im Falle eines leeren Textfeldes der enthaltene *Null*-Wert durch die *Nz*-Funktion in eine leere Zeichenkette umgewandelt und an das aufgerufene Formular übergeben.

10.2.2 Mehrere Werte per Öffnungsargument übergeben

Gegebenenfalls wollen Sie auch mehrere Werte per Öffnungsargument übergeben. In diesem Fall müssten wir diese zu einer Zeichenkette zusammenfassen, wobei die einzelnen Werte durch ein Zeichen voneinander getrennt sind, das sonst üblicherweise nicht verwendet wird – zum Beispiel das Zeichen mit dem ASCII-Code 166 (¦).

Zu Beispielzwecken fügen wir dem Formular *frmOpenArgsSender* noch drei weitere Textfelder namens *txtArgument1*, *txtArgument2* und *txtArgument3* hinzu sowie eine Schaltfläche namens *cmdOpenArgsMehrereArgumente*. Die Schaltfläche *cmdOpenArgsMehrere* Argumente löst die folgende Ereignisprozedur aus:

```
Private Sub cmdOpenArgsMehrereArgumente_Click()
    Dim strArgumente As String
    strArgumente = Nz(Me!txtArgument1, "") & "¦" & Nz(Me!txtArgument2, "") & "¦" _
        & Nz(Me!txtArgument3, "")
    DoCmd.OpenForm "frmOpenArgsMehrere", OpenArgs:=strArgumente
End Sub
```

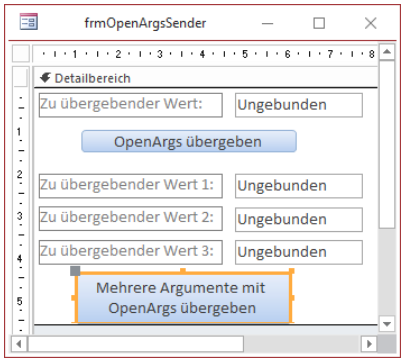


Abbildung 10.7: Formular zur Übergabe mehrerer Parameter per Öffnungsargument

Das Zielformular *frmOpenArgsMehrere* statten wir mit den drei Textfeldern *txtParameter1*, *txtParameter2* und *txtParameter3* aus. Außerdem legen wir für dieses Formular die folgende Ereignisprozedur an, die durch das Ereignis *Beim Öffnen* ausgelöst wird:

```
Private Sub Form_Open(Cancel As Integer)
    Dim strOpenArgs As String
    strOpenArgs = Me.OpenArgs
    Me!txtParameter1 = Split(strOpenArgs, "¦")(0)
    Me!txtParameter2 = Split(strOpenArgs, "¦")(1)
    Me!txtParameter3 = Split(strOpenArgs, "¦")(2)
End Sub
```

Die Prozedur speichert das mit *OpenArgs* gelieferte Öffnungsargument in der Variablen *strOpenArgs* zwischen. Dann greifen wir mit der *Split*-Funktion auf die einzelnen, durch das Trennzeichen (¦) voneinander getrennten Elemente zu. *Split* erstellt dabei ein Array aus den Elementen, auf das wir über den Index zugreifen können, also etwa per *Split(strOpenArgs, "¦")(0)* für das erste Element. Das Ergebnis sieht dann wie in Abbildung 10.8 aus. Wir können auch ein oder mehrere Textfelder im aufrufenden Formular leer lassen. Durch die *Nz*-Funktion wird dann eine leere Zeichenfolge in die Variable *strArgumente* eingefügt.

10.3 Daten vom geöffneten Formular aus auslesen

Wenn Sie Daten von einem Formular zum nächsten übergeben wollen, gibt es auch die Möglichkeit, dass der Benutzer per Mausklick vom ersten Formular das zweite Formular öffnet und dass das geöffnete Formular dann auf das öffnende Formular zugreift, um sich die benötigten Informationen zu holen. In diesem Fall müsste das geöffnete Formular allerdings prüfen, ob das vermeintlich öffnende Formular auch offen ist – gegebenenfalls wurde es ja auch auf andere Art geöffnet, zum Beispiel vom Navigationsbereich aus.

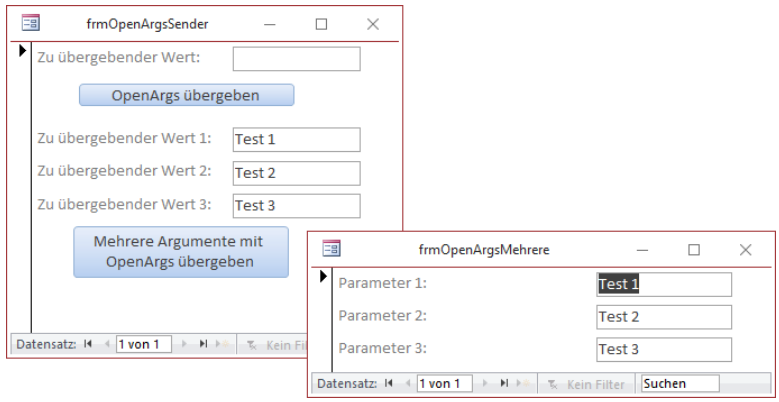


Abbildung 10.8: Übergabe von drei Parametern gleichzeitig per Öffnungsargument

Als Erstes schauen wir uns also die Funktion an, mit der Sie prüfen, ob ein Formular bereits geöffnet ist. Dieses enthält nur eine einzige Zeile:

```
Public Function IstFormularGeoeffnet(strFormular As String) As Boolean
    IstFormularGeoeffnet = SysCmd(acSysCmdGetObjectState, acForm, strFormular) > 0
End Function
```

Die Funktion prüft mit der *SysCmd*-Funktion den Objekt-Zustand des mit *strFormular* übergebenen Elements. *SysCmd* liefert einen Wert größer als *0* zurück, wenn das Formular geöffnet ist.

Den genauen Öffnungszustand wollen wir nicht abfragen, denn wir gehen davon aus, dass der Benutzer die Formulare nicht in der Entwurfsansicht öffnet, was zu Problem führt, wenn Sie auf die Werte von Steuerelementen in diesem Formular zugreifen wollen.

Das öffnende Formular dieses Beispiels heißt *frmOeffnendesFormular* und enthält drei Textfelder namens *txtWert1*, *txtWert2* und *txtWert3* sowie eine Schaltfläche namens *cmdAnderesFormularOeffnen*.

Das von dieser Schaltfläche geöffnete Formular heißt *frmDatenOeffnendemFormularAuslesen* und enthält ebenfalls drei Textfelder namens *txtWert1*, *txtWert2* und *txtWert3* (siehe Abbildung 10.9).

Die Schaltfläche *cmdAnderesFormularOeffnen* löst die folgende Ereignisprozedur aus:

```
Private Sub cmdAnderesFormularOeffnen_Click()
    DoCmd.OpenForm "frmDatenVonOeffnendemFormularAuslesen"
End Sub
```

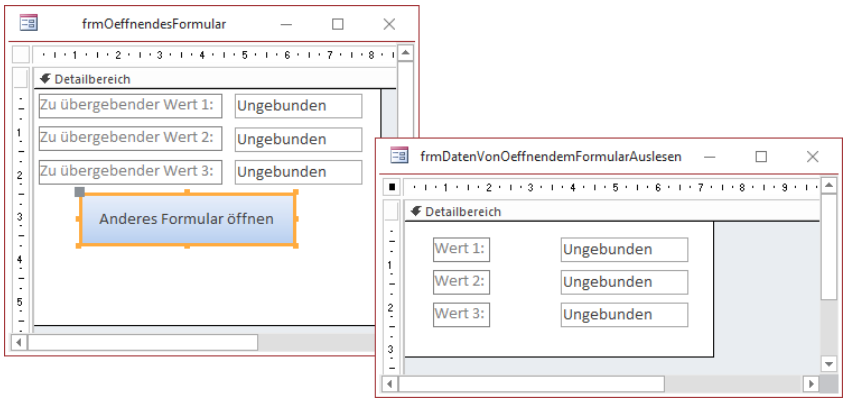


Abbildung 10.9: Öffnendes und geöffnetes Formular in der Entwurfsansicht

Dies öffnet das Formular *frmDatenVonOeffnendemFormularAuslesen*. Damit dieses auf das öffnende Formular zugreift und die enthaltenen Werte einliest, fügen wir ihm eine Ereignisprozedur hinzu, die durch das Ereignis *Beim Laden* ausgelöst wird und wie folgt aussieht:

```
Private Sub Form_Load()
    If IstFormularGeoeffnet("frmOeffnendesFormular") Then
        Me!txtWert1 = Nz(Forms!frmOeffnendesFormular!txtWert1, "")
        Me!txtWert2 = Nz(Forms!frmOeffnendesFormular!txtWert2, "")
        Me!txtWert3 = Nz(Forms!frmOeffnendesFormular!txtWert3, "")
    End If
End Sub
```

Wenn wir nun einige der Felder im aufrufenden Formular füllen und das Zielformular aufrufen, werden die Werte in den Textfeldern des aufrufenden Formulars in die Textfelder des Zielformulars geschrieben (siehe Abbildung 10.10).

10.4 Daten vom öffnenden Formular aus auslesen

Ein weiterer interessanter Anwendungsfall für das Auslesen von Werten von einem an das andere Formular erfolgt in die entgegengesetzte Richtung. Dabei ruft ein Formular ein anderes auf, der Benutzer trägt dort Werte in Steuerelemente ein und nach dem Schließen des aufgerufenen Formulars soll das aufrufende Formular die eingegebenen Werte auslesen.

Dazu müssen Sie wissen, dass man die Werte in einem Formular nach dem Schließen nicht mehr auslesen kann. Das heißt, dass wir den Moment, in dem der Benutzer die *OK*-Schaltfläche anklickt, um das Formular zu schließen, bis zum tatsächlichen Schließen des Formulars ausnutzen müssen.

11 Formularereignisse

Formulare bieten Ereignisse für alles an, was in der Praxis nötig ist. Von den Ereignissen, die beim Öffnen und Schließen eines Formulars ausgelöst werden über solche, die durch das Anzeigen eines neuen Datensatzes feuern bis hin zu Ereignissen für die verschiedenen Datensatzänderungen und Maus- und Tastaturaktionen.

Die folgenden Abschnitte stellen die Ereignisse und gängige Praxisbeispiele vor.

11.1 Ereigniseigenschaften

Formulare bieten die folgenden Ereigniseigenschaften. Sie finden den Namen im Eigenschaftenblatt des Formulars in der Entwurfsansicht, in Klammern den Namen des Ereignisses unter VBA und dahinter eine kurze Beschreibung und Hinweise, wo wir dieses Ereignis mit einem Beispiel beschreiben.

Wir geben die Ereignisse hier in der Reihenfolge an, in der sie auch im Eigenschaftenblatt aufgeführt werden. Ereignisse, die in der Regel nicht zum Einsatz kommen oder die nur beim Drucken eines Formulars verwendet werden, haben wir ausgelassen:

» *Beim Anzeigen* (VBA: *OnCurrent*): Wird beim Anzeigen eines Datensatzes ausgelöst. Siehe zum Beispiel unter »Ereignisse beim Öffnen eines Formulars« ab Seite 237.

» *Bei Laden* (VBA: *OnLoad*): Wird beim Öffnen eines Formulars ausgelöst. Siehe »Ereignisse beim Öffnen eines Formulars« ab Seite 237.

» *Beim Klicken* (VBA: *OnClick*): Wird beim Anklicken des Formulars mit der Maus ausgelöst.

» *Nach Aktualisierung* (VBA: *AfterUpdate*): Wird bei Änderungen am aktuellen Datensatz ausgelöst. Siehe »Ereignisse beim Bearbeiten von Datensätzen« ab Seite 242.

» *Vor Aktualisierung* (VBA: *BeforeUpdate*): Wird bei Änderungen am aktuellen Datensatz ausgelöst. Siehe »Ereignisse beim Bearbeiten von Datensätzen« ab Seite 242.

» *Vor Eingabe* (VBA: *BeforeInsert*): Wird beim Anlegen eines neuen Datensatzes ausgelöst. Siehe »Ereignisse beim Bearbeiten von Datensätzen« ab Seite 242.

» *Nach Einfügung* (VBA: *AfterInsert*): Wird beim Anlegen eines neuen Datensatzes ausgelöst. Siehe »Ereignisse beim Bearbeiten von Datensätzen« ab Seite 242.

» *Vor Löschbestätigung* (VBA: *BeforeDelConfirm*): Wird vor dem Löschen eines Datensatzes ausgelöst. Siehe »Ereignisse beim Löschen von Datensätzen« ab Seite 244.

- *Beim Löschen* (VBA: *OnDelete*): Wird beim Löschen eines Datensatzes ausgelöst. Siehe »Ereignisse beim Löschen von Datensätzen« ab Seite 244.
- *Nach Löschbestätigung* (VBA: *AfterDelConfirm*): Wird nach dem Löschen eines Datensatzes ausgelöst. Siehe »Ereignisse beim Löschen von Datensätzen« ab Seite 244.
- *Bei Geändert* (VBA: *OnDirty*): Wird ausgelöst, sobald der Benutzer einen neu geladenen oder soeben gespeicherten Datensatz ändert.
- *Bei Fokuserhalt* (VBA: *OnGotFocus*): Wird beim Öffnen eines Formulars ausgelöst, aber nur, wenn das Formular keine Steuerelemente aufweist, die den Fokus erhalten könnten. Siehe »Ereignisse beim Öffnen eines Formulars« ab Seite 237.
- *Bei Fokusverlust* (VBA: *OnLostFocus*): Wird beim Schließen eines Formulars ausgelöst, aber nur, wenn das Formular keine Steuerelemente aufweist, die den Fokus besessen haben könnten. Siehe »Ereignisse beim Schließen eines Formulars« ab Seite 241.
- *Beim Doppelklicken* (VBA: *OnDblClick*): Wird beim Doppelklick mit einer Maustaste ausgelöst.
- *Bei Maustaste ab* (VBA: *OnMouseDown*): Wird beim Herunterdrücken der Maustaste ausgelöst.
- *Bei Maustaste auf* (VBA: *OnMouseUp*): Wird beim Loslassen der Maustaste ausgelöst.
- *Bei Mausbewegung* (VBA: *OnMouseMove*): Wird beim Bewegen der Maus in kurzen zeitlichen Abständen ausgelöst.
- *Bei Taste auf* (VBA: *OnKeyUp*): Wird beim Loslassen einer Taste ausgelöst.
- *Bei Taste ab* (VBA: *OnKeyDown*): Wird beim Herunterdrücken einer Taste ausgelöst.
- *Bei Taste* (VBA: *OnKeyPress*): Wird beim Betätigen einer Taste ausgelöst.
- *Bei Rückgängig* (VBA: *OnUndo*): Wird ausgelöst, wenn der Benutzer die aktuellen und ungespeicherten Änderungen durch Betätigung der *Esc*-Taste verwirft.
- *Beim Öffnen* (VBA: *OnOpen*): Wird beim Öffnen eines Formulars ausgelöst. Siehe »Ereignisse beim Öffnen eines Formulars« ab Seite 237.
- *Beim Schließen* (VBA: *OnClose*): Wird beim Schließen eines Formulars ausgelöst. Siehe »Ereignisse beim Schließen eines Formulars« ab Seite 241.
- *Bei Größenänderung* (VBA: *OnResize*): Wird beim Ändern der Größe des Formulars ausgelöst, aber auch etwa beim Öffnen eines Formulars. Siehe etwa »Ereignisse beim Öffnen eines Formulars« ab Seite 237.
- *Bei Aktivierung* (VBA: *OnActivate*): Wird bei Aktivieren eines Formulars ausgelöst, also etwa auch beim Öffnen. Siehe »Ereignisse beim Öffnen eines Formulars« ab Seite 237.

- *Bei Deaktivierung* (VBA: *OnDeactivate*): Wird beim Aktvieren eines anderen Elements, aber auch beim Schließen eines Formulars ausgelöst. Siehe »Ereignisse beim Schließen eines Formulars« ab Seite 241.
- *Bei Entladen* (VBA: *OnUnload*): Wird beim Schließen eines Formulars ausgelöst. Siehe »Ereignisse beim Schließen eines Formulars« ab Seite 241.
- *Bei Fehler* (VBA: *OnError*): Dieses Ereignis wird ausgelöst, wenn ein Fehler in einem Formular auftritt, der nicht durch eine VBA-Anweisung verursacht wird, sondern durch ein Problem beim Speichern eines Datensatzes – also beispielsweise, wenn ein bereits vorhandener Wert in einem Feld mit eindeutigem Index vorliegt. Mehr dazu unter »Ereignisse beim Öffnen eines Formulars« ab Seite 237.
- *Bei Mausrad* (VBA: *MouseWheel*): Wird beim Betätigen des Mausrades ausgelöst.
- *Bei Filter* (VBA: *OnFilter*): Wird ausgelöst, wenn der Benutzer etwa einen formularbasierten Filter oder einen Spezialfilter anwendet.
- *Bei angewendetem Filter* (VBA: *OnApplyFilter*): Wird ausgelöst, wenn der Benutzer das Formular filtert.
- *Bei Zeitgeber* (VBA: *OnTimer*): Wird immer dann ausgelöst, nachdem der für die Eigenschaft Bei Zeitgeber angegebene Zeitraum abgelaufen ist (siehe »Zeitgeber für Formulare programmieren« ab Seite 250).

11.2 Ereignisse beim Öffnen eines Formulars

Wenn Sie ein Formular öffnen, werden direkt einige Ereignisse ausgeführt. Wie aber finden wir heraus, um welche Ereignisse es sich dabei handelt und in welcher Reihenfolge diese ausgelöst werden?

Dazu gibt es eine einfache Methode: Sie legen einfach für alle Ereigniseigenschaften, die Sie prüfen wollen, jeweils eine Ereignisprozedur an. Dieser fügen Sie dann jeweils eine *Debug.Print*-Anweisung hinzu, welche den Namen des Ereignisses im Direktbereich des VBA-Editors ausgibt. Dazu legen wir einfach ein neues, leeres Formular an.

Diesem fügen wir für alle Ereigniseigenschaften des Formulars den Wert *[Ereignisprozedur]* hinzu (siehe Abbildung 11.1). Wir wollen bei unserem Experiment im Anschluss einfach keine Ereigniseigenschaft auslassen.

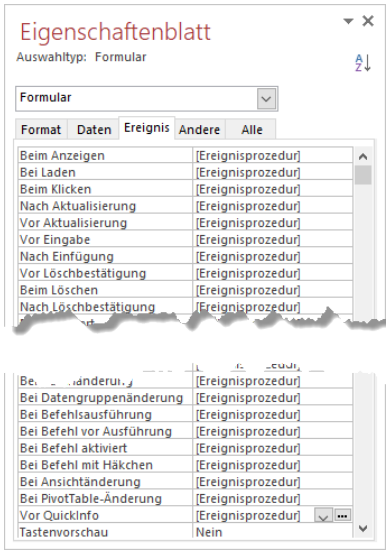


Abbildung 11.1: Ereignisprozeduren für alle Ereigniseigenschaften

Außerdem füllen wir die Ereignisprozedur im VBA-Editor jeweils mit einer *Debug.Print*-Anweisung, die den Namen der Ereignisprozedur samt der Parameterliste ausgibt. Für uns sind vor allem die folgenden Ereignisprozeduren interessant:

```
Private Sub Form_Activate()
    Debug.Print "Form_Activate"
End Sub

Private Sub Form_Current()
    Debug.Print "Form_Current()"
End Sub

Private Sub Form_GotFocus()
    Debug.Print "Form_GotFocus()"
End Sub

Private Sub Form_Load()
    Debug.Print "Form_Load()"
End Sub

Private Sub Form_Open(Cancel As Integer)
    Debug.Print "Form_Open(Cancel As Integer)"
End Sub
```

12 Steuerelemente

In diesem Kapitel schauen wir uns die in Access eingebauten Steuerelemente an. In einem weiteren Kapitel sehen wir uns dann noch die wichtigsten ActiveX-Steuerelemente wie das *TreeView*-, das *ListView*- und das *ImageList*-Steuerelement an.

12.1 Steuerelemente hinzufügen

Wenn Sie ein Steuerelement zum Formular hinzufügen wollen, klicken Sie in der Regel einen der Einträge im Ribbon unter *Entwurf|Steuerelemente* an, zum Beispiel das Textfeld (siehe Abbildung 12.1). Die überschaubare Menge von eingebauten Steuerelementen finden Sie, wenn Sie die Liste der Steuerelemente wie in der Abbildung aufklappen.

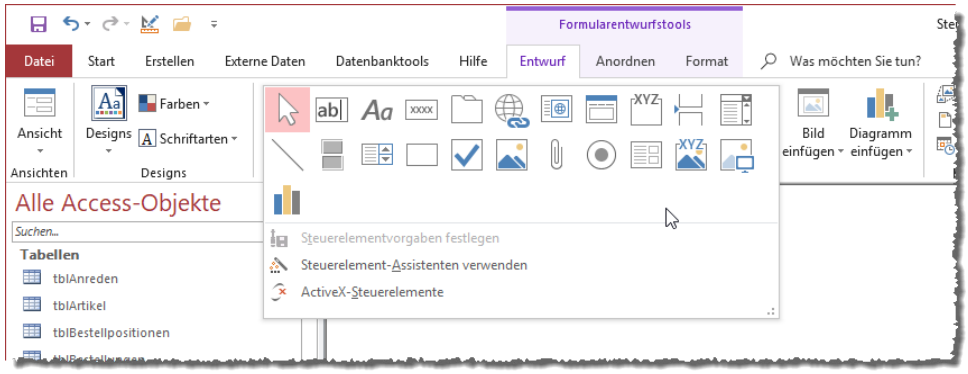


Abbildung 12.1: Auswahl von Steuerelementen

Darunter finden Sie noch drei weitere Einträge. Der erste heißt *Steuerelementvorgaben festlegen*. Damit können Sie die Einstellungen, die Sie für das aktuell im Formular markierte Steuerelement vorgenommen haben, als Standard für alle von nun an in diesem Formular anzulegenden Steuerelemente übernehmen. Mehr dazu erfahren Sie unter »Formular-Vorlage« ab Seite 133.

Der zweite Eintrag heißt *Steuerelement-Assistenten verwenden*. Damit aktivieren oder deaktivieren Sie die Verwendung von Assistenten, wenn Sie ein Steuerelement anlegen wollen. Einen Assistenten gibt es beispielsweise für Schaltflächen. Wenn Sie also den Eintrag *Steuerelement-Assistenten verwenden* aktiviert haben und dann eine Schaltfläche zum Formular hinzufügen, erscheint der Schaltflächen-Assistent (siehe Abbildung 12.2). Wir wollen Ihnen allerdings in die-

sem Buch lieber die manuellen Techniken zeigen, um Steuerelemente zu Formularen hinzuzufügen und diese mit den gewünschten Eigenschaften auszustatten.

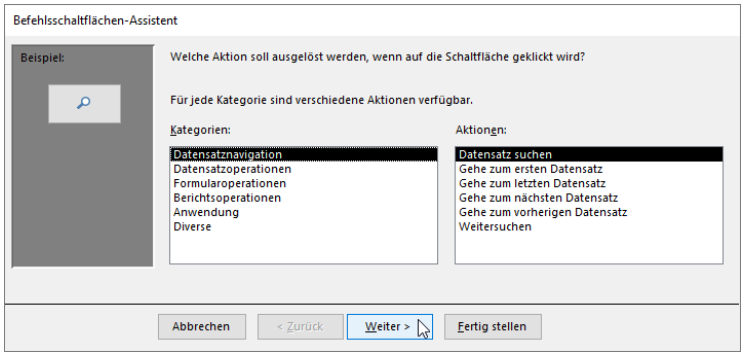


Abbildung 12.2: Der Schaltflächen-Assistent

Der letzte Eintrag schließlich heißt *ActiveX-Steuerelemente*. Mit diesem öffnen Sie den Dialog zum Auswählen weiterer Steuerelemente. Mehr über die interessantesten dieser Steuerelemente erfahren Sie unter »ActiveX-Steuerelemente« ab Seite 379.

12.2 Das Textfeld

Das Textfeld hat die Aufgabe, Texte anzuzeigen und ihre Bearbeitung zu ermöglichen. Diese stammen entweder aus dem Feld der an das Formular gebundenen Datensatzquelle, an welches das Textfeld gebunden ist, oder das Feld enthält einen berechneten Wert auf Basis andere Steuerelemente oder es ist schlicht ungebunden und soll nur zur Eingabe von Informationen dienen, die dann anderweitig verarbeitet werden können – beispielsweise, um einen Suchbegriff zum Suchen nach bestimmten Datensätzen zu ermitteln.

Damit steht das Textfeld im Formular im Gegensatz zu dem Textfeld in einem Bericht, denn dort ist es nur zur Anzeige von Text vorgesehen.

12.2.1 Gebundene Textfelder

Wie oben bereits angedeutet, kann ein Textfeld gebunden oder ungebunden sein, oder anders formuliert: einen Wert in der Eigenschaft *Steuerelementinhalt* enthalten oder auch nicht. Letzteres entspricht einem ungebundenen Textfeld. Der Anteil der gebundenen Textfelder in Ihrer Anwendung wird tendenziell den der ungebundenen Textfelder bei weitem übersteigen. Textfelder werden in der Regel über die Eigenschaft *Steuerelementinhalt* an ein Feld gebunden, das von der Tabelle oder Abfrage bereitgestellt wird, die für die Eigenschaft *Datensatzquelle* des Formulars angegeben ist.

Die erste Möglichkeit, ein Textfeld anzulegen, das an ein Feld der Datensatzquelle gebunden ist, sieht so aus – hier am Beispiel des Feldes *Firma* der Tabelle *tblKunden*: Sie öffnen das betroffene Formular in der Entwurfsansicht und stellen seine Eigenschaft *Datensatzquelle* etwa auf die Tabelle *tblKunden* ein. Dann fügen ein Textfeld über den Ribbon-Befehl *Entwurf|Steuerelemente|Textfeld* zum Detailbereich des Formularentwurfs hinzu. Nach dem Einfügen des Textfeldes markieren Sie dieses, wechseln zum Eigenschaftenblatt und wählen für die Eigenschaft *Steuerelementinhalt* das Feld *Firma* aus (siehe Abbildung 12.3). In diesem Fall müssten Sie noch die Beschriftung des Bezeichnungsfeldes auf *Firma:* anpassen.

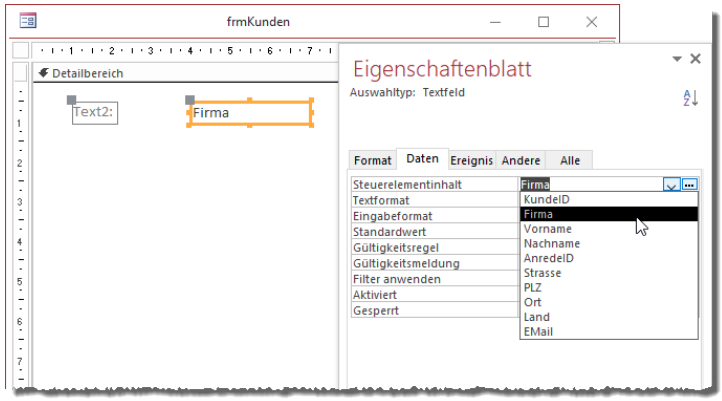


Abbildung 12.3: Binden eines Textfeldes an ein Feld einer Tabelle

Einfacher gelingt dies, indem Sie die Feldliste etwa mit der Tastenkombination *Alt + F8* aktivieren und dann einfach das Feld *Firma* aus der Feldliste in den Detailbereich des Formulars ziehen. Vorteil: So erhält auch das Bezeichnungsfeld gleich den richtigen Text (siehe Abbildung 12.4).

Abbildung 12.4: Anlegen eines gebundenen Feldes über die Feldliste

Wenn Sie nun in die Formularansicht wechseln und das Formular den ersten Datensatz der für die Datensatzquelle angegebenen Tabelle oder Abfrage anzeigt, liefert das an das Feld *Firma* gebundene Textfeld den Wert dieses Feldes für den aktuellen Datensatz (siehe Abbildung 12.5).

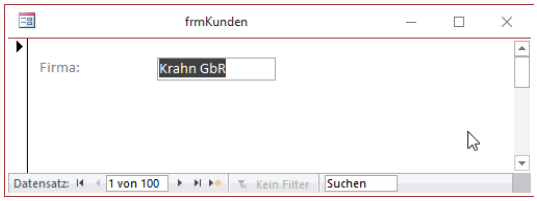


Abbildung 12.5: Anzeigen des gebundenen Steuerelements in der Formularansicht

Ändern des Wertes eines gebundenen Textfeldes

Wenn der Benutzer nun einen neuen Wert für dieses Textfeld einträgt und den Datensatz speichert, wird der neue Wert automatisch in der dem Textfeld zugrundeliegenden Datensatzquelle gespeichert.

Vorgaben für das Bezeichnungsfeld

Das automatisch hinzugefügte Bezeichnungsfeld eines aus der Feldliste hinzugefügten Steuerelements erhält automatisch einen bestimmten Namen. Wenn Sie keine weiteren Einstellungen treffen, wird hier der Name des Textfeldes aus der zugrunde liegenden Tabelle oder Abfrage verwendet, also der Datensatzquelle. Sie können jedoch, wie in »Tabellenentwurf für Formulare optimieren« ab Seite 52 beschrieben, einen alternativen Beschriftungstext definieren, und zwar indem Sie die Eigenschaft *Beschriftung* des Feldes im Entwurf der zugrundeliegenden Tabelle einstellen.

Es gibt noch eine Möglichkeit, die Beschriftung im Vorfeld anzupassen, und zwar wenn Sie eine Abfrage als Datensatzquelle des Formulars nutzen. In diesem Fall können Sie auch für jedes Feld der Abfrage nochmals die Eigenschaft *Beschriftung* einstellen. Diese finden Sie im Entwurf der Abfrage, wenn Sie ein Feld markieren und dann mit F4 das Eigenschaftenblatt aktivieren (siehe Abbildung 12.6).

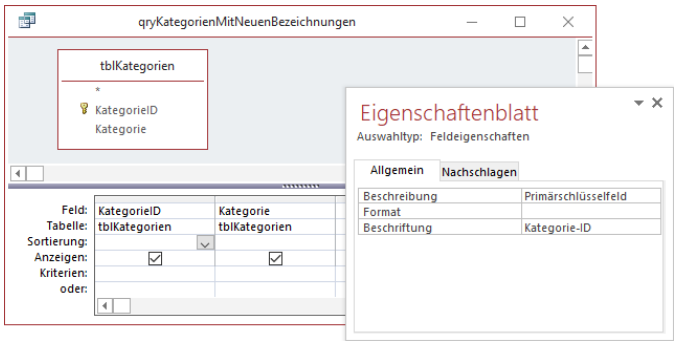


Abbildung 12.6: Individuelle Beschriftung per Abfrage-Eigenschaft

12.2.2 Ungebundene Textfelder

Sie können natürlich auch Textfelder zu einem Formular hinzufügen, die nicht an die Felder der verwendeten Datensatzquelle gebunden sind. Sie können ja sogar auch Formulare verwenden, die gar keine Datensatzquelle enthalten. Wozu aber sollte man ungebundene Textfelder brauchen?

Die Frage ist einfach zu beantworten: Zum Beispiel könnten Sie ein solches Textfeld nutzen, um den Suchbegriff für eine Suchfunktion abzufragen. Oder damit der Benutzer den Pfad zu einer Datei eingeben kann, aus der Daten eingelesen werden sollen oder die mit Daten gefüllt werden soll.

Die Benutzung solcher Textfelder erfordert daher im Gegensatz zu den gebundenen Feldern, die ja zum Anzeigen und Bearbeiten der Inhalte der Datensatzquelle dienen und auch ohne Einsatz von VBA auskommen, in den meisten Fällen eine benutzerdefinierte Programmierung beispielsweise zum Einstellen oder Auslesen dieser Felder. Wie dies gelingt, schauen wir uns in den folgenden Abschnitten an.

12.2.3 Standardwert festlegen

Für ein Textfeld können Sie einen Standardwert festlegen. Dieser wird dann für einen neuen, leeren Datensatz automatisch angezeigt und übernommen, wenn der Benutzer den Datensatz bearbeitet und gespeichert hat. Diese Eigenschaft gibt es auch schon im Tabellenentwurf.

Wenn Sie ein gebundenes Textfeld auf Basis eines Tabellenfeldes mit einem Standardwert anlegen, dann verwendet das Textfeld den im Tabellenentwurf angelegten Standardwert – allerdings ohne diesen explizit auch für die gleichnamige Eigenschaft des Textfeldes anzulegen.

Wenn Sie für ein Textfeld, das an ein Tabellenfeld mit einem Standardwert gebunden ist, auch noch die Eigenschaft *Standardwert* mit einem Wert versehen, verwendet das Textfeld diesen Standardwert.

Für die Zuweisung des Standardwerts über die VBA-Eigenschaft *DefaultValue* müssen Sie den zu übergebenden Wert mit zusätzlichen Anführungszeichen versehen. Wenn Sie etwa für das Feld Land des Formulars *frmTextfeld* den Standardwert auf den zuletzt angegebenen Wert festlegen wollen, hinterlegen Sie die folgende Ereignisprozedur für das Ereignis *Nach Aktualisierung* des Textfeldes *Land*:

```
Private Sub Land_AfterUpdate()
    Me!Land.DefaultValue = Chr(34) & Me!Land & Chr(34)
End Sub
```

Mit *Chr(34)* geben Sie das notwendige öffnende und schließende Anführungszeichen an.

Das ist nur für Steuerelemente nötig, die an Textfelder gebunden sind. Für Zahlenfelder wie etwa das Fremdschlüsselfeld *AnredeID* wäre folgende Variante möglich:

```
Private Sub AnredeID_AfterUpdate()
    Me!AnredeID.DefaultValue = Me!AnredeID
End Sub
```

12.2.4 Gültigkeitsregel und Gültigkeitsmeldung

Hier gilt nur teilweise das Gleiche wie beim Standardwert: Sollten die Eigenschaften *Gültigkeitsregel* und *Gültigkeitsmeldung* für ein Feld im Tabellenentwurf festgelegt worden sein, werden diese auch bei der Eingabe der Daten für dieses Feld im Formular berücksichtigt. Allerdings bieten die gebundenen Steuerelemente wie auch das Textfeld auch noch eigene, gleichnamige Eigenschaften an, mit denen Sie – im Gegensatz zum Standardwert – eine zusätzliche Gültigkeitsregel angeben können. Wir können also in der Tabellendefinition etwa für das Feld *Firma* festlegen, dass die Werte länger als zwei Zeichen lang sein müssen. Dazu verwenden wir die folgenden Eigenschaftswerte:

» *Gültigkeitsregel*: *Länge([Firma])>2*

» *Gültigkeitsmeldung*: *Der Firmenname muss mehr als zwei Zeichen enthalten.*

In den gleichnamigen Eigenschaften für das an das Feld *Firma* gebundene Steuerelement legen wir nun eine weitere Gültigkeitsregel fest (siehe Abbildung 12.7).

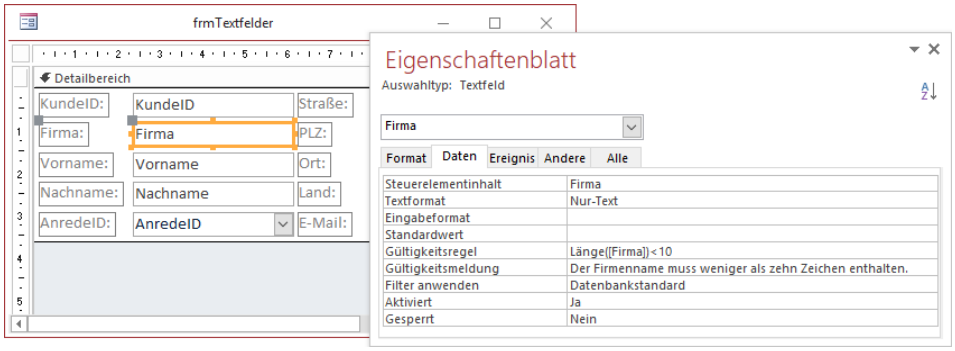


Abbildung 12.7: Einstellen der Gültigkeitsregel für ein Steuerelement

Wenn Sie nun eine Zeichenkette eingeben, die kürzer als drei Zeichen ist, erscheint die Gültigkeitsmeldung aus der Tabellendefinition, wenn Sie eine Zeichenkette eingeben, die länger als zehn Zeichen ist, erscheint die Meldung, die wir im Formularentwurf für das Textfeld angegeben haben, das an dieses Tabellenfeld gebunden ist.

12.2.5 Richtext in Textfeldern

Für Felder mit dem Datentyp *Langer Text* (früher Memotext genannt) können Sie die Eigenschaft *Textformat* neben *Nur Text* auch auf *Rich-Text* einstellen (siehe Abbildung 12.8). Das ist nötig, damit Sie auch in Formular-Steuerelementen, die an dieses Feld gebunden sind, die Rich-Text-Funktionen nutzen können.

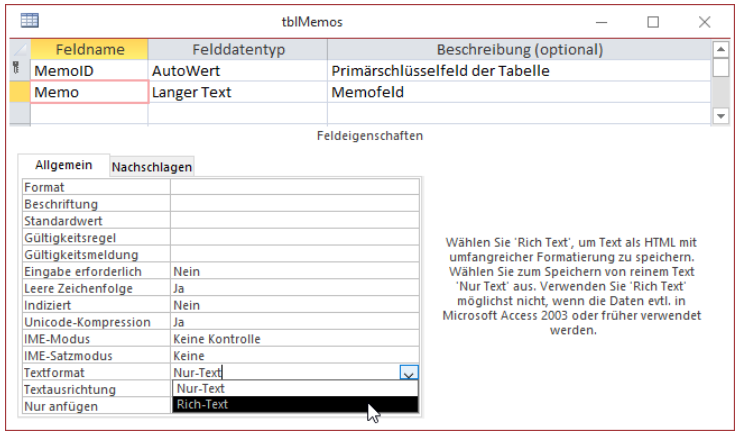


Abbildung 12.8: Rich-Text-Einstellung für ein langes Textfeld

Wenn Sie dann ein Formular an diese Tabelle und ein Textfeld an das Memofeld binden, erhält dieses für die Eigenschaft *Textformat* ebenfalls den Wert *Rich-Text* (siehe Abbildung 12.9).

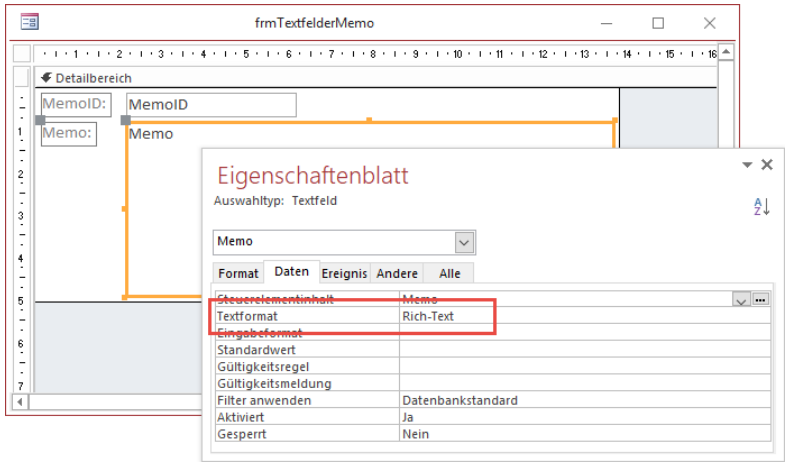


Abbildung 12.9: Rich-Text-Format für ein Textfeld

Wenn Sie nun in die Formularansicht wechseln und einen Text im Memofeld markieren, zeigt Access das Popup aus Abbildung 12.10 an, mit dem Sie die Formatierungen einstellen können.

Abbildung 12.10: Popup-Element mit Formatierungsbefehlen

Wenn Sie den Inhalt des Steuerelements mit Formatierungstags ausgeben wollen, setzen Sie im Direktbereich etwa den Befehl *Screen.ActiveControl.Value* ab. Dazu muss das Memofeld den Fokus haben. Das Ergebnis sieht wie in Abbildung 12.11 aus.

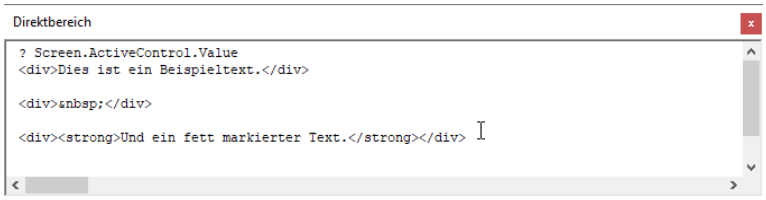


Abbildung 12.11: Ausgabe des Rich-Textes mit Formatangaben im Direktbereich

Um gezielt den Text ohne Formatierungsangaben zu erhalten, verwenden Sie die *PlainText*-Funktion:

```
? PlainText(Screen.ActiveControl.Value)
Dies ist ein Beispieltext.

Und ein fett markierter Text.
```

12.2.6 Datumsangaben in Textfeldern

Für Textfelder, die an ein Datumsfeld gebunden sind, können Sie einen Datepicker aktivieren. Bevor wir uns ansehen, wie dieser aussieht, schauen wir uns erst die Eigenschaft an (siehe Abbildung 12.12).

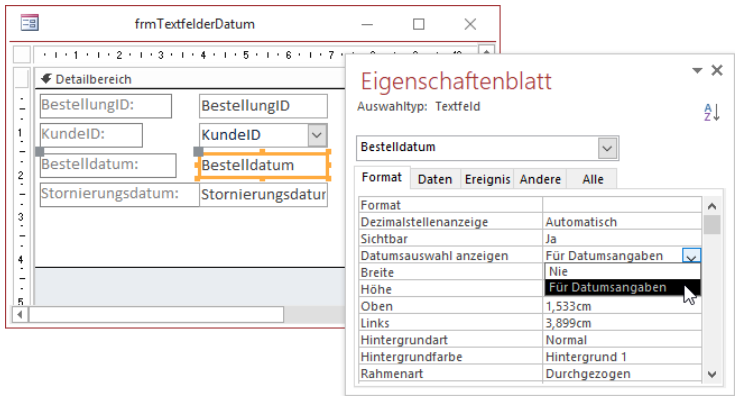


Abbildung 12.12: Aktivieren des Datepickers

Wenn Sie dann in die Formularansicht wechseln und das Datumsfeld anklicken, erscheint rechts das Kalendersymbol. Dieses klicken Sie an und erhalten dann die Auswahl des Datums per Kalenderblatt (siehe Abbildung 12.13).

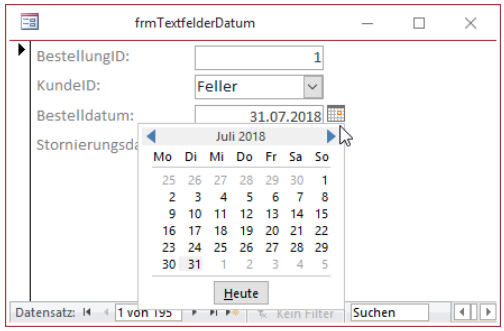


Abbildung 12.13: Der Datepicker in Aktion

Den Datepicker können Sie auch per VBA aufrufen. Dazu fügen Sie etwa eine Schaltfläche rechts vom Textfeld ein und fügen dieser die folgende Ereignisprozedur für das Ereignis *Beim Klicken* hinzu:

```
Private Sub cmdDatepicker_Click()
    Me!Stornierungsdatum.SetFocus
    RunCommand acCmdShowDatePicker
End Sub
```

Dies setzt den Fokus auf das betroffene Steuerelement und zeigt dann den Datepicker an.

12.2.7 Markierungen in Textfeldern

Wenn Sie den Fokus auf ein Textfeld verschieben, zeigt Access die Einfügemarke auf eine von drei Arten an, deren Standard Sie in den Access-Optionen festlegen können. Dazu öffnen Sie die Access-Optionen über den Ribbon-Eintrag *Datei|Optionen*. Hier finden Sie unter *Clienteinstellungen* im Bereich *Bearbeiten* die Option *Cursorverhalten bei Eintritt ins Feld* (siehe Abbildung 12.14).

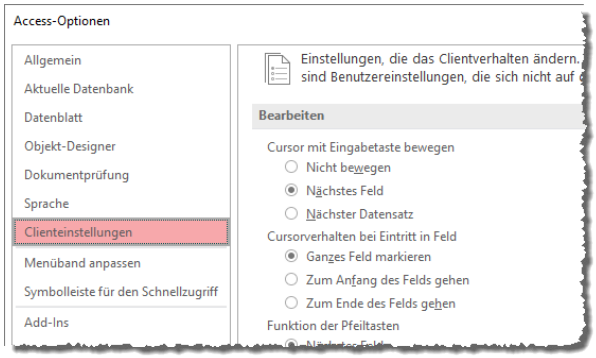


Abbildung 12.14: Optionen für das Eintrittsverhalten in ein Textfeld

Die Optionen sind selbsterklärend. Wenn Sie die Einstellung ändern, wird diese direkt auch für aktuell geöffnete Formulare übernommen.

Eintrittsverhalten per VBA einstellen

Wenn es für einen bestimmten Zweck sinnvoll sein sollte, ein spezielles Verhalten für den Eintritt in ein Feld zu wählen, können Sie dies auch beim Öffnen des Formulars per VBA festlegen. Zu Beispielzwecken legen wir ein Formular namens *frmTextfelderEintrittsverhalten* an, das eine Optiongruppe enthält (mehr dazu siehe weiter unten unter »Die Optionsgruppe« ab Seite 320) sowie ein Textfeld namens *txtTest*.

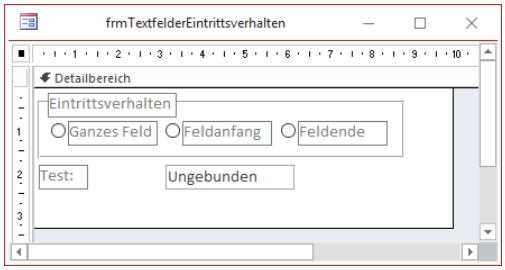


Abbildung 12.15: Beispiel für das Eintrittsverhalten

Für das Ereignis *Beim Laden* haben wir die folgende Ereignisprozedur hinterlegt:

```
Private Sub Form_Load()
    Me!ogrEintrittsverhalten = GetOption("Behavior Entering Field")
    Me!txtTest = "Beispieltext"
End Sub
```

Dies wählt auf Basis der Access-Einstellung *Behavior Entering Field*, die wir mit der Funktion *GetOption* einlesen, die passende Option in der Optionsgruppe aus und füllt das Textfeld. Die folgende Prozedur wird durch das Ändern des Wertes der Optionsgruppe ausgelöst und stellt die Option *Behavior Entering Field* auf den Wert der Optionsgruppe ein.

Danach verschieben wir den Fokus auf das Textfeld, damit sie gleich das aktuelle Verhalten erkennen können:

```
Private Sub ogrEintrittsverhalten_AfterUpdate()
    SetOption "Behavior Entering Field", Me!ogrEintrittsverhalten
    Me!txtTest.SetFocus
End Sub
```

Allerdings gibt es einen Haken an dieser Einstellung: Sie gilt für alle Access-Anwendungen. Wenn Sie also nur für die Benutzung eines bestimmten Textfeldes die Einstellung ändern, müssen Sie diese auch anschließend wieder rückgängig machen.

Die Alternative: Der Benutzer markiert etwa ein ganzes Wort durch einen Doppelklick auf den Inhalt oder markiert den gesamten Text, indem er diesen bei gedrückter Maustaste überfährt. Auch mit der Tastatur gelingt dies, indem Sie etwa zur Erweiterung der Markierung um ein Zeichen bei gedrückter *Umschalt*-Taste die Einfügemarke mit den Pfeil-Tasten nach rechts oder links bewegen. Oder Sie drücken zusätzlich die *Strg*-Taste, um mit jedem Betätigen von *Nach links* oder *Nach rechts* ein zusätzliches Wort zu markieren.

12.2.8 Markierungen per VBA setzen

Mit den Eigenschaften *SelLength*, *SelStart* und *SelText* können Sie die aktuelle Markierung auslesen und auch einstellen. *SelLength* liefert die Länge der Markierung, *SelStart* das erste Zeichen und *SelText* den markierten Text.

SelStart ist 0-basierend und enthält folglich für eine Markierung, die vor dem ersten Zeichen beginnt, den Wert *0*. Im Formular *frmTextfelderMarkierungen* können Sie sich anschauen, wie verschiedene Markierungen wirken. Dazu haben wir das Feld *txtBeispiel* zum Experimentieren sowie die drei Felder *txtSelLength*, *txtSelStart* und *txtSelText* zum Anzeigen der Werte genutzt.

Für das Feld *txtBeispiel* haben wir für die Ereignisse *Bei Taste auf* und *Bei Maustaste ab* die folgenden beiden Ereignisprozeduren hinterlegt:

```
Private Sub txtBeispiel_KeyUp(KeyCode As Integer, Shift As Integer)
    TextfelderAktualisieren
End Sub

Private Sub txtBeispiel_MouseUp(Button As Integer, Shift As Integer, X As Single, _
        Y As Single)
    TextfelderAktualisieren
End Sub
```

Diese rufen die folgende Prozedur auf, welche die Werte der drei Eigenschaften *SelLength*, *SelStart* und *SelText* in die drei Textfelder mit den entsprechenden Namen schreibt:

```
Private Sub TextfelderAktualisieren()
    Me!txtSelLength = Me!txtBeispiel.SelLength
    Me!txtSelStart = Me!txtBeispiel.SelStart
    Me!txtSelText = Me!txtBeispiel.SelText
End Sub
```

Andersherum können Sie die Werte auch per VBA einstellen. Auch dafür haben wir ein Beispiel im Formular vorgesehen. Wenn der Benutzer die Werte der beiden Textfelder *txtSelLength* oder *txtSelStart* ändert, werden die Änderungen auf dasTextfeld *txtBeispiel* übertragen. Dazu rufen die beiden Textfelder für das Ereignis *Nach Aktualisierung* jeweils die Prozedur *MarkierungAktualisieren* auf:

```
Private Sub txtSelLength_AfterUpdate()
    MarkierungAktualisieren
End Sub

Private Sub txtSelStart_AfterUpdate()
    MarkierungAktualisieren
End Sub
```

Diese Prozedur sieht wie folgt aus:

```
Private Sub MarkierungAktualisieren()
    Me!txtBeispiel.SetFocus
    Me!txtBeispiel.SelStart = Me!txtSelStart
    Me!txtBeispiel.SelLength = Me!txtSelLength
End Sub
```

Die Prozedur verschiebt erst den Fokus auf das Textfeld *txtBeispiel*, da Sie die Eigenschaften *SelStart* und *SelLength* nicht ändern können, wenn das Textfeld nicht den Fokus hat.

12.2.9 Markierung bei Fokuserhalt

Wir können das Markierungsverhalten, das wir weiter oben in »Markierungen in Textfeldern« ab Seite 270 kennengelernt haben, auch per VBA abbilden. Zu Beispielzwecken haben wir ein Formular mit den drei Textfeldern *txtKomplett*, *txtStart* und *txtEnde* erstellt (siehe Abbildung 12.16). Der Inhalt des ersten Textfeldes soll beim Fokuserhalt komplett markiert werden, beim zweiten Textfeld soll die Einfügemarke ganz vorn erscheinen und beim dritten Textfeld ganz hinten.

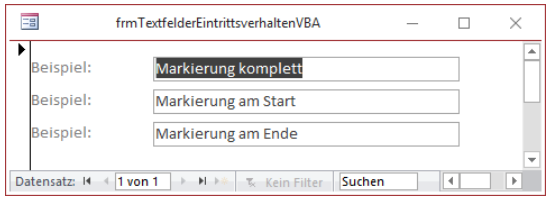


Abbildung 12.16: Markieren von Textfeldern per VBA

Wenn Sie etwa den Feldinhalt bei Fokuserhalt des Feldes komplett markieren wollen, hinterlegen Sie eine entsprechende Ereignisprozedur für das Ereignis *Bei Fokuserhalt*. Diese sieht wie folgt aus:

```
Private Sub txtKomplett_GotFocus()
    Me!txtKomplett.SelStart = 0
    Me!txtKomplett.SelLength = 1000
End Sub
```

Der Wert *0* für *SelStart* stellt den Start der Markierung vor dem ersten Zeichen ein. Der Wert *1000* für *SelLength* sorgt für eine Markierung von maximal 1.000 Zeichen (wenn das Feld mehr als 1.000 Zeichen aufnehmen kann, wählen Sie einen größeren Wert). Wenn Sie wollen, dass die Einfügemarke am Anfang des Textes erscheint, verwenden Sie die folgende Ereignisprozedur. Sie stellt *SelStart* auf den Wert *0* ein:

```
Private Sub txtStart_GotFocus()
    Me!txtStart.SelStart = 0
End Sub
```

Fehlt noch die Prozedur, die dafür sorgt, dass die Einfügemarke am Ende des Textes erscheint:

```
Private Sub txtEnde_GotFocus()
    Me!txtEnde.SelStart = 1000
End Sub
```

Wenn Sie die drei Textfelder etwa mit der *Tabulator*-Taste durchlaufen, sehen Sie die Wirkung dieser Prozeduren.

12.2.10 Value, Text und OldValue

Diese drei Eigenschaften beziehen sich alle auf den Inhalt eines Textfeldes. Sinnvoll ist der Einsatz dieser drei Eigenschaften vor allem für gebundene Textfelder. *Value* liefert dabei den aktuellen Wert des Feldes, also den Wert, der beim Anzeigen des Datensatzes in einem gebundenen Textfeld enthalten ist. Die *Text*-Eigenschaft liefert zu diesem Zeitpunkt den gleichen Wert. Das ändert sich, wenn der Benutzer den Inhalt des Textfeldes ändert, aber diesen noch nicht aktualisiert, indem er etwa das Feld verlässt oder den Datensatz speichert. Dann enthält *Text* nämlich den aktuell angezeigten Wert. *Value* liefert nach wie vor den Wert, der beim Anzeigen des Datensatzes vorhanden war. Wenn Sie dann noch den geänderten Wert übernehmen, indem Sie etwa zum nächsten Steuerelement wechseln, kommt die *OldValue*-Eigenschaft ins Spiel. *Value* enthält nämlich nun den neuen Wert des Textfeldes (genau wie *Text*) und mit *OldValue* können Sie auf den Wert des Textfeldes vor der Änderung zugreifen.

Wenn Sie also einmal den vorherigen Wert eines Feldes des aktuellen Datensatzes benötigen, können Sie diesen über die Eigenschaft *OldValue* ermitteln. Diese liefert solange den vorherigen Wert, bis Sie den Datensatz erneut speichern.

Im Formular *frmTextfelderValueOldValueText* zeigen wir, wie diese Eigenschaften funktionieren (siehe Abbildung 12.17).

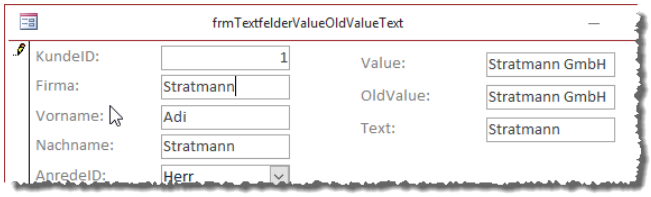


Abbildung 12.17: Demonstration der Eigenschaften *Value*, *OldValue* und *Text*

Für die Ereignisse *Bei Änderung*, *Bei Fokuserhalt* und *Bei Fokusverlust* haben wir jeweils eine Ereignisprozedur angelegt. Alle drei Ereignisprozeduren rufen die folgende Prozedur auf:

```
Private Sub Aktualisieren()
    Me!txtValue = Me!Firma.Value
    Me!txtOldValue = Me!Firma.OldValue
    Me!txtText = Me!Firma.Text
End Sub
```

Diese stellt jeweils die drei Textfelder rechts auf die Werte der Eigenschaften *Value*, *OldValue* und *Text* ein.

Sie können die Eigenschaften *Value* und *OldValue* beispielsweise nutzen, um beim Protokollieren von Änderungen den vorherigen und den aktuellen Wert eines Feldes zu speichern.

12.2.11 Weitere wichtige Eigenschaften

Weitere Eigenschaften sind für verschiedene Einsatzzwecke wichtig:

» *Gesperrt*: Gibt an, ob die Daten im Feld geändert werden können. Die Einstellung *Ja* erlaubt zwar das Verschieben des Fokus in das Textfeld, aber der Benutzer kann den Inhalt nicht verändern. Kombinieren Sie die Einstellung Nein für die Eigenschaft *Aktiviert* und *Ja* für die Eigenschaft *Gesperrt*, erhalten Sie ein Feld, das zwar deaktiviert ist und dementsprechend nicht mehr durch den Benutzer angesteuert werden kann. Allerdings fällt die für deaktivierte Steuerelemente typische optische Hervorhebung weg.

» *Eingabetastenverhalten*: Diese Eigenschaft legt fest, was beim Betätigen der Eingabetaste geschieht. Die Einstellung *Standard* sorgt für das Übernehmen der eingegebenen Daten und das Verschieben des Fokus zum nächsten Steuerelement. Die Einstellung *Neue Zeile* hingegen sorgt bei Betätigung der Eingabetaste dafür, dass an der Stelle der Einfügemarke ein Zeilenumbruch zum Feldinhalt hinzugefügt wird. Das ist sinnvoll, wenn der Benutzer mehrzeilige Texte in ein Textfeld eingeben können soll. Dies ist allerdings auch bei der Einstellung *Standard* möglich: Der Benutzer muss die Eingabetaste dann bei gedrückter *Strg*-Taste betätigen, um einen Zeilenumbruch hinzuzufügen.

» *SteuerelementTip-Text*: Wenn das Textfeld erklärungsbedürftig ist, können Sie für die Eigenschaft *SteuerelementTip-Text* einen Text hinzufügen. Dieser wird beim Platzieren der Maus über dem Textfeld eingeblendet.

» *Reihenfolgenposition* und *In Reihenfolge*: Wenn das Formular mehrere Steuerelemente besitzt, können Sie über diese beiden Eigenschaften festlegen, welches Steuerelement gleich beim Öffnen des Formulars den Fokus erhalten soll und in welcher Reihenfolge die einzelnen Steuerelemente beim Betätigen von Tasten wie der *Tabulator*-Taste, der Eingabetaste et cetera durchlaufen werden sollen (siehe auch »Aktivierreihenfolge« ab Seite 81).

12.3 Das Bezeichnungsfeld

Bezeichnungsfelder sind Steuerelemente, die Texte anzeigen und die nicht zur Laufzeit durch den Benutzer anpassbar sind wie etwa die Inhalte von Textfeldern. Sie können zwar den Text eines Bezeichnungsfeldes zur Laufzeit anpassen, indem Sie die *Caption*-Eigenschaft per VBA ändern, aber das passiert in der Regel selten.

Bezeichnungsfelder werden meist zusammen mit einem Steuerelement angelegt, zum Beispiel mit Textfeldern, Kombinationsfeldern, Listenfeldern und anderen Steuerelementen, die selbst keine Möglichkeit zur Anzeige einer Beschriftung anbieten. Zu dieser Kategorie gehören die Schaltfläche oder die Umschaltfläche, die im Gegensatz zu Textfeld und Co. eine Eigenschaft namens *Beschriftung* anbieten.

Die Beschriftung können Sie über die entsprechende Eigenschaft im Eigenschaftenblatt ändern, aber Sie können auch das Bezeichnungsfeld markieren und dann nochmals auf den angezeigten Text klicken oder die Taste *F2* betätigen. In beiden Fällen wird der Bearbeitungsmodus für die Beschriftung aktiviert.

Im Gegensatz zu den übrigen Steuerelementen werden Sie ein Bezeichnungsfeld in den seltensten Fällen per VBA referenzieren, daher ist es nicht nötig, ein Bezeichnungsfeld mit einem besonderen Namen zu versehen. Sie können die von Access vergebenen Namen einfach beibehalten.

12.4 Die Schaltfläche

Die Funktion einer Schaltfläche dürfte geläufig sein. In der Regel kommen Sie bei Schaltflächen mit den folgenden Features hin:

» Einstellen der Beschriftung, etwa auf *OK*

» Festlegen des Namens, zum Beispiel *cmdOK*

» Einstellen der Eigenschaft *Beim Klicken* auf den Wert *[Ereignisprozedur]*

» Anlegen einer passenden Ereignisprozedur durch Anklicken der Schaltfläche mit den drei Punkten rechts im Eigenschaftsfeld

Diese Prozedur sieht dann etwa wie folgt aus, wenn Sie damit das aktuelle Formular schließen wollen:

```
Private Sub cmdOK_Click()
    DoCmd.Close acForm, Me.Name
End Sub
```

In der einfachsten Fassung sieht eine Schaltfläche schließlich wie in Abbildung 12.18 aus.

Abbildung 12.18: Eine einfache Schaltfläche

12.4.1 Bilder auf Schaltflächen

Wenn Sie einer Schaltfläche ein Icon hinzufügen wollen, können Sie das unter Access 2016 problemlos erledigen (genau genommen geht es auf die nachfolgend beschriebene Art sogar

schon ab Access 2010). Dazu fügen Sie zunächst eine Schaltfläche zum Formular hinzu. Dann wählen Sie den Ribbon-Eintrag *Entwurf|Steuerelemente|Bild einfügen* aus und klicken entweder die Schaltfläche *Durchsuchen* an oder wählen, wenn Sie bereits Bilder auf diese Weise zur Anwendung hinzugefügt haben, eines der vorhandenen Bilder aus der Liste aus. Dieses wird dann als Icon zur Schaltfläche hinzugefügt (siehe Abbildung 12.19).

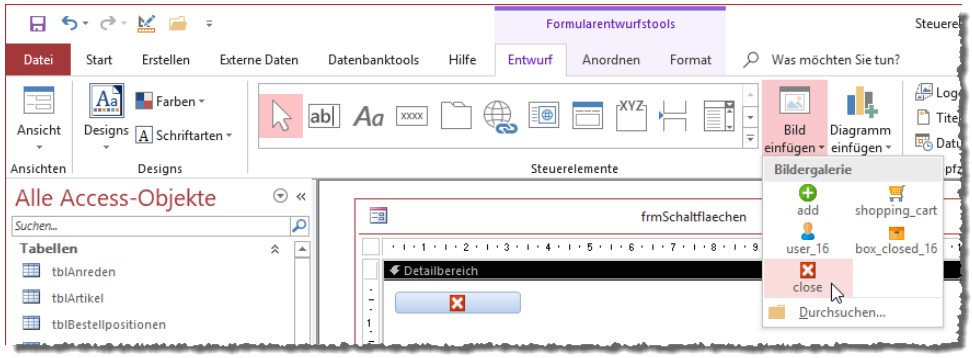


Abbildung 12.19: Nach dem Markieren der Zielschaltfläche und der Auswahl eines der Bilder der Liste erscheint dieses auf der Schaltfläche.

Die Bilder, die Sie in diesem Bereich des Ribbons vorfinden, werden in einer Systemtabelle namens *MSysResources* gespeichert. Sie können einmal hinzugefügte Bilder an verschiebenen Stellen in der Anwendung nutzen (siehe auch »Bilder in Formularen« ab Seite 367).

In dem Moment, indem Sie der Schaltfläche ein Bild zuweisen, verschwindet allerdings seine Beschriftung. Dies können Sie mit der Eigenschaft *Anordnung der Bildbeschriftung* ändern. Normalerweise werden Sie die Beschriftung wohl rechts vom Icon anzeigen wollen, wozu die Eigenschaft entweder auf den Wert *Allgemein* oder *Rechts* einstellen.

Wir finden übrigens, dass zwischen Icon und Beschriftung standardmäßig zu wenig Platz ist. Deshalb schlagen wir vor, die Beschriftung mit einem Leerzeichen zu beginnen. In Abbildung 12.20 sehen Sie oben zwei Schaltflächen ohne Leerzeichen und unten mit einem Leerzeichen zwischen Bild und Text.

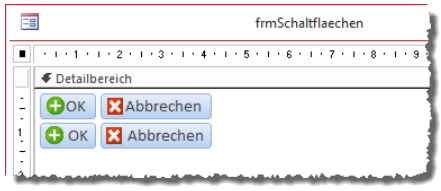


Abbildung 12.20: Schaltflächen mit Bildern

Optisch ebenfalls ansprechend ist es, wenn Sie die Eigenschaft *Hintergrundart* für eine solche Schaltfläche auf *Transparent* einstellen wie es in Abbildung 12.21 der Fall ist.

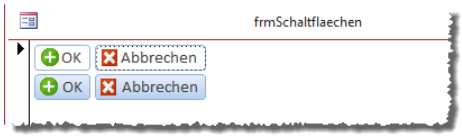


Abbildung 12.21: Schaltflächen mit transparentem Hintergrund

12.4.2 Schaltflächen aktivieren und deaktivieren

Schaltflächen sind nicht die flexibelsten Steuerelemente. Sie können sie anklicken und die dahinterliegende Ereignisprozedur auslösen. Das war es weitestgehend. Gelegentlich werden sie allerdings eine Schaltfläche deaktivieren wollen, damit der Benutzer zwar noch sieht, dass die Funktion grundsätzlich verfügbar ist, aber die Funktion gerade nicht genutzt werden kann. Eine Schaltfläche aktivieren oder deaktivieren Sie im Entwurf, indem Sie die Eigenschaft *Aktiviert* auf den Wert *Ja* oder *Nein* einstellen. Das hilft aber in den meisten Fällen nur bedingt weiter, da Sie die Schaltfläche ja vermutlich aufgrund bestimmter Bedingungen aktivieren oder deaktivieren wollen. Deshalb zeigen wir Ihnen noch, wie Sie eine Schaltfläche per VBA aktivieren oder deaktivieren.

Die erste Schaltfläche im Formular aus Abbildung 12.22 heißt *cmdAktivieren*. Sie löst die folgende Ereignisprozedur aus:

```
Private Sub cmdAktivieren_Click()
    Me!cmdZuAktivierendeSchaltflaeche.Enabled = True
End Sub
```

Die zweite löst diese Prozedur aus und deaktiviert eine weitere Schaltfläche:

```
Private Sub cmdDeaktivieren_Click()
    Me!cmdZuAktivierendeSchaltflaeche.Enabled = False
End Sub
```

Abbildung 12.22: Aktivieren und Deaktivieren einer Schaltfläche

Schaltflächen können Sie auch etwa in Abhängigkeit der Werte der angezeigten Daten aktivieren oder deaktivieren. Ein Beispiel dafür finden Sie unter »Übersichtsformular mit Detailformular« ab Seite 467.

12.5 Die Umschaltfläche

Die Umschaltfläche ist eine Schaltfläche, die zwei Zustände anzeigen kann, zum Beispiel gedrückt und nicht gedrückt. Dabei können Sie diese Schaltfläche sogar an ein Feld der Datensatzquelle des Formulars binden, beispielsweise an ein *Ja*/*Nein*-Feld.

Wenn Sie eine neue Umschaltfläche zum Entwurf eines Formulars hinzufügen, sieht dieses zunächst einmal etwas merkwürdig aus. Abbildung 12.23 zeigt den Unterschied zu einer normalen Schaltfläche.

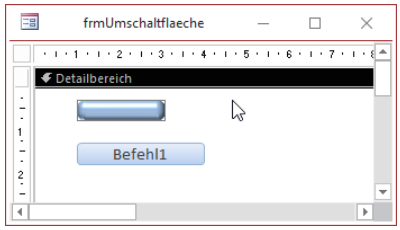


Abbildung 12.23: Ungewöhnliches Design der Umschaltfläche (oben)

12.5.1 Design der Umschaltfläche anpassen

Damit das Design zu dem einer herkömmlichen Schaltfläche passt, fügen Sie eine solche zum Entwurf des Formulars hinzu (wenn auch nur temporär), markieren dieses, wählen den Ribbon-Befehl *Start|Zwischenablage|Format übertragen* aus und klicken dann die Umschaltfläche an. Fügen Sie dann noch eine Beschriftung hinzu, sieht die Umschaltfläche wie eine Schaltfläche aus. Wir fügen dieser noch eine Ereignisprozedur für das Ereignis *Beim Klicken* hinzu:

```
Private Sub tglUmschaltflaeche_Click()
    MsgBox Me!tglUmschaltflaeche.Value
End Sub
```

Danach erscheint jeweils eine Meldung wie in Abbildung 12.24 mit dem aktuellen Wert der Umschaltfläche.

Da die Umschaltfläche auch an ein Feld einer Tabelle gebunden werden kann (vorzugsweise ein *Ja*/*Nein*-Feld), bietet es auch die beiden Ereignisse *Vor Aktualisierung* und *Nach Aktualisierung* an.

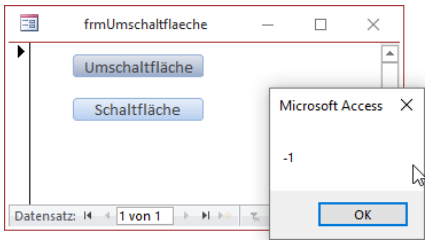


Abbildung 12.24: Wert einer Umschaltfläche

12.5.2 Beschriftung anpassen

Interessant ist für dieses Steuerelement die Anpassung der Beschriftung, um dem Benutzer mitzuteilen, was geschieht, wenn er auf die Taste klickt. Hier soll die Umschaltfläche den Wert *Anschalten* anzeigen, wenn die Taste nicht niedergedrückt ist und *Ausschalten*, wenn sie niedergedrückt ist:

```
Private Sub tglAnAus_AfterUpdate()
    If Me!tglAnAus = True Then
        Me!tglAnAus.Caption = "Ausschalten"
    Else
        Me!tglAnAus.Caption = "Anschalten"
    End If
End Sub
```

Hierbei ist noch zu beachten, dass Sie den initialen Wert manuell eintragen – oder per VBA, wenn die Umschaltfläche an ein Feld der Datensatzquelle gebunden ist.

12.5.3 Umschaltfläche als Element einer Optionsgruppe

Sie können einer Optionsgruppe (siehe weiter hinten unter »Die Optionsgruppe« ab Seite 320) statt Optionsfelder auch Umschaltflächen zuweisen. Das sieht im Entwurf etwa wie in Abbildung 12.25 aus.

In der Formularansicht sieht die Optionsgruppe dann wie in Abbildung 12.26 aus. Der Clou ist, dass bei Betätigen einer der Umschaltflächen die bis dahin gedrückte Umschaltfläche als nicht gedrückt dargestellt wird. Wichtig ist, dass Sie für die Eigenschaft *Optionswert* der drei Schaltflächen verschiedene Werte eingeben, also beispielsweise *1*, *2* und *3* oder Werte, die Sie für die Weiterverarbeitung der gewählten Einstellung benötigen.

Wenn Sie den Rahmen der Optionsgruppe drumherum als störend empfinden, können Sie ihn durch Einstellen der Eigenschaft *Rahmenart* auf den Wert *Transparent* einfach ausblenden – das Bezeichnungsfeld können Sie bei Bedarf direkt löschen.

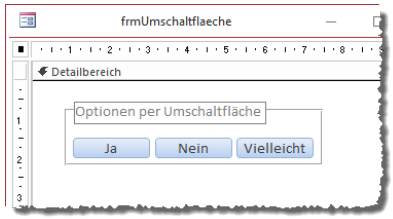


Abbildung 12.25: Umschaltflächen in einer Optionsgruppe

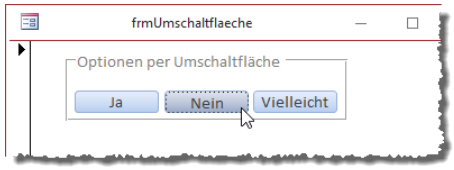


Abbildung 12.26: Umschaltfläche in einer Optionsgruppe in Aktion

12.6 Das Kombinationsfeld

Das Kombinationsfeld ist eines der Steuerelemente mit den meisten Funktionen. Sie können damit nicht nur Werte aus einer Liste auswählen, sondern auch noch per Eingabe von Buchstaben schnell zu dem gesuchten Listeneintrag springen oder sogar neue Einträge zu einer Liste hinzufügen. Wir schauen uns in den folgenden Abschnitten an, welche Möglichkeiten das Kombinationsfeld bietet.

Gebunden oder ungebunden

Kombinationsfelder können Sie als gebundene und als ungebundene Steuerelemente nutzen. Die Bindung bietet sich beispielsweise an, wenn Sie damit eine 1:n-Beziehung abbilden wollen, bei der Sie mit dem Kombinationsfeld die Werte der Tabelle mit dem Primärschlüsselfeld der Beziehung auswählen wollen. Ungebundene Kombinationsfelder funktionieren auch – Sie könnten damit beispielsweise in einem Formular eine Liste von Optionen anbieten.

Datensatzherkunft eines Kombinationsfeldes

Ein Kombinationsfeld müssen Sie genau wie ein Formular mit einer Datensatzherkunft ausstatten. Dabei haben Sie ähnliche Möglichkeiten:

» Die Werte kommen aus einer Tabelle, Abfrage oder aus einem SQL-Ausdruck.

» Die Werte kommen aus einer Wertliste, die in der Regel beim Entwurf festgelegt wird.

» Die Werte werden über eine spezielle Funktion eingefügt (dieser Fall kommt sehr selten vor und daher behandeln wir ihn nicht).

Lookupdaten auswählen

Wir starten mit einem kleinen Beispiel. Wenn Sie etwa die Einträge der Tabelle *tblAnreden* in einem Kombinationsfeld anlegen wollen, gehen Sie wie folgt vor:

Legen Sie ein neues Kombinationsfeld im Formularentwurf an. Legen Sie dann einen Namen für das Kombinationsfeld fest, zum Beispiel *cboAnredeID*. Stellen Sie die Eigenschaft *Datensatz* auf eine Abfrage ein, die genau zwei Felder enthält: das Primärschlüsselfeld der Tabelle sowie das Feld, dessen Inhalt Sie im Kombinationsfeld anzeigen wollen. In diesem Fall soll die Datensatzquelle *SELECT AnredeID, Anrede FROM tblAnreden* lauten.

Wechseln Sie nun in die Formularansicht und öffnen das Kombinationsfeld, erhalten Sie die Liste aus Abbildung 12.27. Das soll nicht so aussehen, denn wir wollen die Primärschlüsselwerte eher nicht sehen, dafür aber die Inhalte des Feldes *Anrede*.

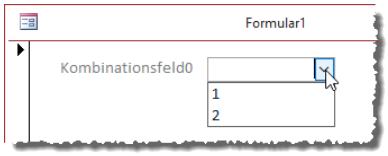


Abbildung 12.27: Das Kombinationsfeld zeigt die Primärschlüsselwerte an.

Um dies zu erreichen, benötigen wir zwei Eigenschaften: Die Eigenschaft *Spaltenanzahl* stellt ein, wie viele Spalten im Kombinationsfeld angezeigt werden sollen. Diese stellen wir auf den Wert *2* ein. Warum das – wir wollen doch nur eine Spalte anzeigen? Doch eins nach dem anderen. Wenn wir nun auch noch die Eigenschaft *Spaltenbreiten* auf *0cm* einstellen, geschieht folgendes: Es wird nur noch der Inhalt des Feldes *Anrede* angezeigt (siehe Abbildung 12.28).

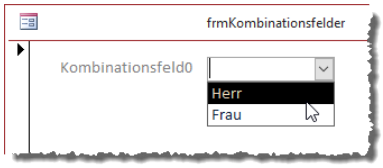


Abbildung 12.28: Das Kombinationsfeld zeigt die gewünschten Werte an.

Das geschieht deswegen, weil das Kombinationsfeld nun zwar zwei Spalten anzeigt, aber die erste mit einer Breite von *0cm*, also quasi ausgeblendet. Dadurch erscheint nun nur noch die Spalte mit den gewünschten Werten.

Wozu eine gebundene Spalte?

In diesem Beispiel benötigen Sie die gebunden Spalte nicht. Diese ist meist erforderlich, wenn Sie das Formular auch an eine Datensatzquelle binden und das Kombinationsfeld entsprechend an das Fremdschlüsselfeld der Datensatzquelle des Formulars gebunden ist, damit es den aus der verknüpften Tabelle ausgewählten Datensatz anzeigen kann. Um dies zu testen, binden wir das Formular nun über seine Eigenschaft *Datensatzquelle* an die Tabelle *tblKunden*. Damit das Kombinationsfeld jeweils den Datensatz der Tabelle *tblAnreden* anzeigt, der für das Feld *AnredeID* in der Tabelle *tblKunden* ausgewählt ist, binden wir das Kombinationsfeld auch noch über die Eigenschaft *Steuerelementinhalt* an das Feld *AnredeID* der Datensatzquelle des Formulars (siehe Abbildung 12.29).

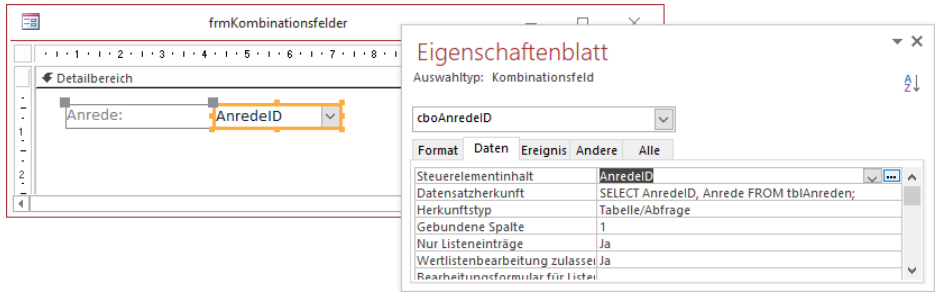


Abbildung 12.29: Einstellen des Steuerelementinhalts des Kombinationsfeldes

Wenn Sie nun in die Formularansicht wechseln, zeigt das Kombinationsfeld gleich die Anrede für den ersten Datensatz der Tabelle *tblKunden* an (siehe Abbildung 12.30). Das Kombinationsfeld ist also nun an das Feld *AnredeID* der Tabelle *tblKunden* des Formulars gebunden. Damit können wir uns nun die Eigenschaft *Gebundene Spalte* ansehen. Diese hat in der Regel den Wert *1*, was bedeutet, dass der anzuzeigende Inhalt der Datensatzquelle des Kombinationsfeldes über die erste Spalte der Datensatzherkunft an das als Steuerelementinhalt angegebene Feld gebunden wird. Das heißt, das Kombinationsfeld zeigt immer den Datensatz der Tabelle *tblAnreden* an, dessen Primärschlüsselwert mit dem Wert des Feldes *AnredeID* des Formulars übereinstimmt.

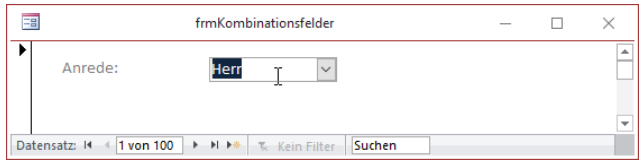


Abbildung 12.30: Gebundenes Kombinationsfeld

12.6.1 Herkunftstypen eines Kombinationsfeldes

Ein Kombinationsfeld bietet in seiner Eigenschaft *Herkunftstyp* drei Werte an, die angeben, welchen Typ das mit der Eigenschaft *Datensatzquelle* angegebene Element aufweisen muss:

» *Tabelle/Abfrage*: Hier geben Sie den Namen einer Tabelle oder Abfrage an oder auch einer SQL-WHERE-Anweisung. Ein Beispiel haben Sie weiter oben schon für die Abfrage basierend auf der Tabelle *tblAnreden* gesehen.

» *Wertliste*: Eine Wertliste ist eine durch Semikola getrennte Liste, die Sie bei dieser Einstellung für die Eigenschaft *Datensatzherkunft* angeben (zum Beispiel *"Herr";"Frau"*)

» *Feldliste*: Eine Feldliste erwartet die Angabe des Namens einer Tabelle der aktuellen Datenbank, deren Felder dann über das Kombinationsfeld ausgewählt werden können.

12.6.2 Datensatzherkunft eines Kombinationsfeldes

Für das Kombinationsfeld gibt es drei verschiedene Möglichkeiten für die Angabe der Datensatzherkunft, die von der Auswahl der oben beschriebenen Eigenschaft *Herkunftsart* abhängig sind.

Tabelle oder Abfrage als Datensatzherkunft

Oben haben Sie mit der Abfrage *SELECT AnredeID, Anrede FROM tblAnreden* bereits ein Beispiel für eine Datensatzherkunft einer Abfrage kennengelernt. Sie hätten in diesem Fall auch einfach die Tabelle *tblAnreden* als Datensatzherkunft angeben können, da diese ja nicht mehr Felder oder Datensätze als die angegebene Abfrage enthält.

Grundsätzlich gilt hier wie auch bei der Datensatzquelle von Formularen, dass Sie mit der Formulierung der Datensatzquelle nicht mehr Felder und Datensätze als nötig festlegen sollten. Wenn das Kombinationsfeld also etwa nur zwei Felder enthalten soll, die Tabelle aber mehr Felder enthält, sollten Sie eine Abfrage erstellen und speichern und diese als Datensatzherkunft angeben oder aber die Abfrage direkt als SQL-Ausdruck angeben.

Wobei es performanter wäre, eine gespeicherte Abfrage zu nutzen, da diese nicht immer wieder neu kompiliert werden muss.

Wertliste als Datensatzherkunft

Bei einer Wertliste stellen Sie entweder zur Entwurfszeit die Daten zusammen, die im Kombinationsfeld angezeigt werden sollen oder Sie füllen dieses zur Laufzeit per VBA. Zur Entwurfszeit würde dies beispielsweise wie in Abbildung 12.31 aussehen.

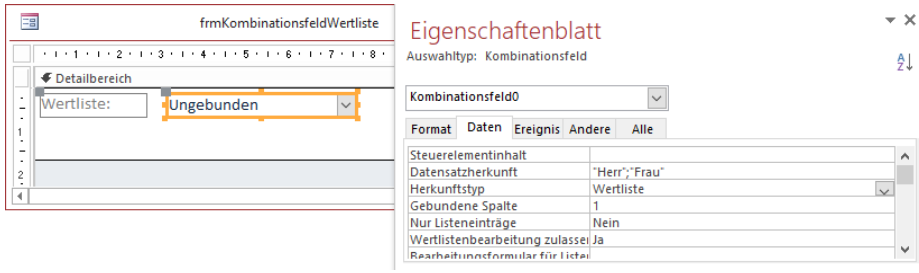


Abbildung 12.31: Wertliste als Datensatzherkunft

Wenn Sie auf die Schaltfläche mit den drei Punkten (...) klicken, die beim Aktivieren der Eigenschaft *Datensatzherkunft* erscheint, öffnet Access einen Dialog, der die einzelnen Einträge der Wertliste untereinander anzeigt. Dadurch vereinfacht sich die Bearbeitung natürlich wesentlich (siehe Abbildung 12.32). Sie können hier auch einen der vorhandenen Einträge als Standardwert festlegen, der dann in der Eigenschaft *Standardwert* hinterlegt wird. Das Kombinationsfeld zur Auswahl des Standardwertes enthält alle bisher eingegebenen Werte und einen leeren Eintrag.

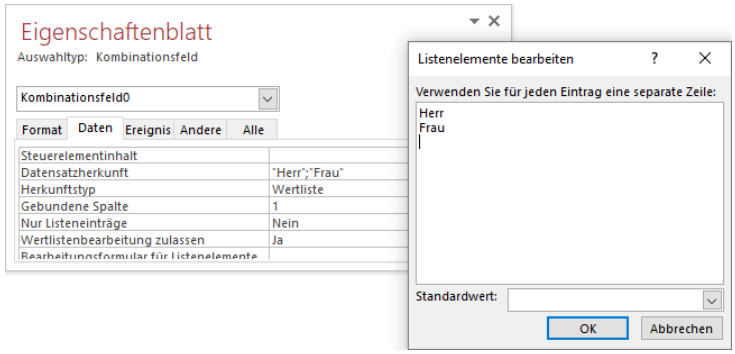


Abbildung 12.32: Dialog zum Bearbeiten der Eigenschaft *Datensatzherkunft* für eine Wertliste

Wenn Sie nun in die Formularansicht wechseln, stehen die beiden per Semikolon getrennten Einträge zur Auswahl im Kombinationsfeld bereit (siehe Abbildung 12.33). Darüber hinaus erscheint darunter auch noch eine Schaltfläche, mit der Sie die Listenelemente bearbeiten können.

Wenn Sie auf diesen Dialog klicken, erscheint der gleiche Dialog, den Sie auch aus der Entwurfsansicht öffnen konnten. Wenn Sie nicht möchten, dass der Benutzer die Werte des Kombinationsfeldes bearbeiten kann, stellen Sie einfach die Eigenschaft *Wertlistenbearbeitung zulassen* auf den Wert *Nein* ein. Die Schaltfläche wird dann in der Formularansicht nicht mehr eingeblendet.

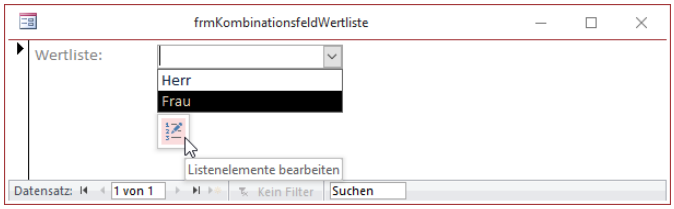


Abbildung 12.33: Kombinationsfeld mit Wertliste in der Formularansicht

Feldliste als Datensatzquelle

Wenn Sie den Eintrag *Feldliste* für die Eigenschaft *Herkunftstyp* auswählen und dann die Eigenschaft *Datensatzherkunft* aktivieren, erscheint eine Schaltfläche zum Aufklappen eines Auswahlfeldes. Dieses bietet dann alle Tabellen der aktuellen Datenbank zur Auswahl an (siehe Abbildung 12.34).

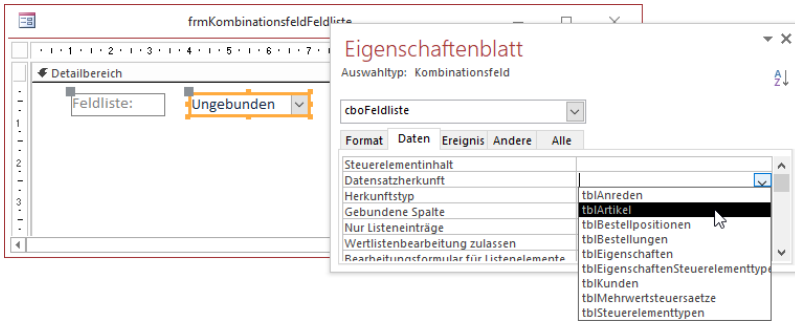


Abbildung 12.34: Auswahl einer Tabelle für die Feldliste

Wenn Sie eine Tabelle ausgewählt haben und in die Formularansicht wechseln, können Sie mit dem so definierten Kombinationsfeld eines der Felder dieser Tabelle auswählen (siehe Abbildung 12.35).

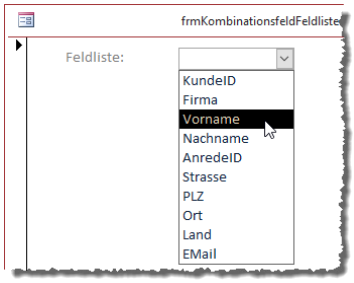


Abbildung 12.35: Auswahl eines Feldnamens aus einer Feldliste

12.6.3 Spaltenanzahl und Spaltenbreiten

Bei Kombinationsfeldern sollte man annehmen, dass man immer nur eine oder zwei Spalten benötigt – eine, falls es sich um ein ungebundenes Feld handelt und zwei für gebundene Felder, damit man das Primärschlüsselfeld noch in einer Spalte mit einer Breite von *0cm* ausblenden und das eigentlich gewünschte Feld anzeigen kann. Allerdings kann man auch noch mehr Spalten angeben. Wozu aber sollte man das tun? Nun: Manchmal, wenn der Benutzer einen der Einträge aus dem Kombinationsfeld auswählt, möchte man vielleicht noch auf weitere Werte aus dem gewählten Datensatz zugreifen. Dann hat man die Wahl, per *DLookup* oder, wenn es mehrere Felder sind, auch per Recordset auf diese Daten zuzugreifen. Man könnte die Daten aber auch gleich im Kombinationsfeld unterbringen, und zwar in weiteren, nicht sichtbaren Spalten.

Also erstellen wir als Erstes mit dem Abfrage-Generator eine Datensatzherkunft für ein Kombinationsfeld, mit dem wir beispielsweise einen Artikel für eine Bestellposition auswählen wollen. Diese hat vier Felder, wobei eines der Felder sogar noch aus einer verknüpften Tabelle gefüllt wird (siehe Abbildung 12.36).

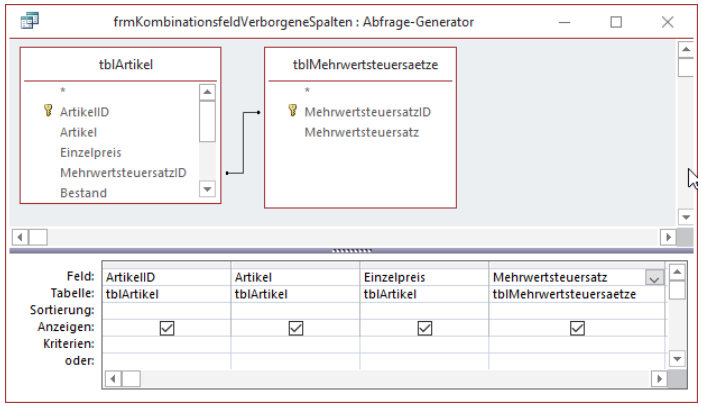


Abbildung 12.36: Datensatzquelle für ein Kombinationsfeld mit mehr als zwei Spalten

Die Eigenschaft *Spaltenanzahl* stellen wir dann auf den Wert *4* ein. Spalten ohne explizite Angabe der Breite nehmen den kompletten Platz ein. Wenn wir also keine Breite angeben, zeigt das Kombinationsfeld den Wert der gebundenen Spalte an und beim Aufklappen erscheinen alle Werte gleichmäßig aufgeteilt über die Breite des Kombinationsfeldes (siehe Abbildung 12.37).

Wie aber bekommen wir es hin, dass wir nur den Inhalt der zweiten Spalte zu sehen bekommen? Dazu haben wie zwei Möglichkeiten: Die erste ist, die Breite für alle Spalten explizit anzugeben. Das heißt, wir geben für die erste und die beiden letzten Spalten die Breite *0cm* an und für die zweite Spalte die Breite des Kombinationsfeldes, die wir aus der Eigenschaft *Breite* auslesen können – also etwa *0cm;3,768cm;0cm;0cm*.

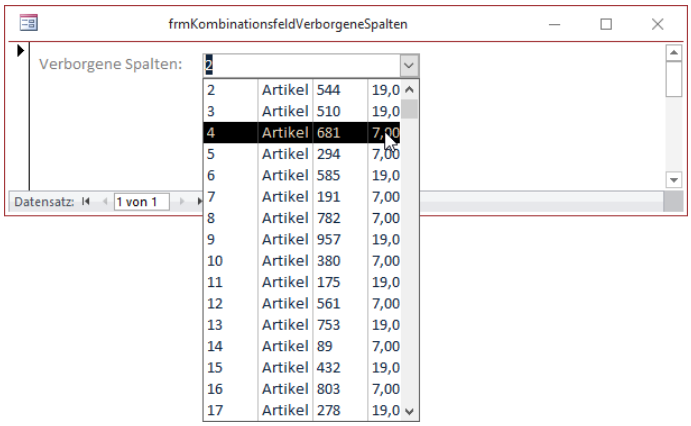


Abbildung 12.37: Gleichmäßige Aufteilung der vier Spalten

Sie können aber auch einen Trick nutzen, der dafür sorgt, dass die zweite Spalte immer über die komplette Breite angezeigt wird. Dazu geben Sie den Wert *0cm;;0cm;0cm* für die Eigenschaft *Spaltenbreiten* ein.

In diesem Fall sehen wir nur das Feld, das wir auch sehen wollen (Abbildung 12.38).

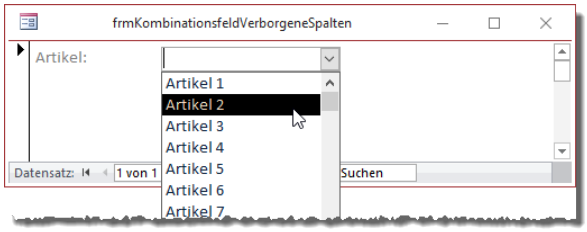


Abbildung 12.38: Kombinationsfeld mit einem sichtbaren von vier Feldern

Wie wir auf die Inhalte der übrigen Spalten zugreifen können, erfahren Sie weiter unten unter »Werte aller Spalten eines Eintrags ermitteln« ab Seite 295.

Anzeige mehrerer Felder bei der Auswahl nutzen

Natürlich kann es auch hilfreich sein, wenn Sie zumindest bei der Auswahl im Kombinationsfeld den Inhalt weiterer Felder angezeigt bekommen. Dann sollten wir allerdings auch die Breite der aufgeklappten Liste etwas großzügiger gestalten. Nehmen wir beispielsweise an, wir wollen im ausgewählten Zustand nur die Firma eines Kunden als Wert anzeigen, beim Aufklappen der Liste aber zusätzlich noch den Vornamen, den Nachnamen und die E-Mail-Adresse des Kunden. Dann

würden wir die Eigenschaft *Datensatzherkunft* zunächst eine Abfrage mit all diesen Werten angeben:

```
SELECT tblKunden.KundeID, tblKunden.Firma, tblKunden.Vorname, tblKunden.Nachname,
tblKunden.EMail FROM tblKunden;
```

Damit als Wert nur die Firma angezeigt wird, stellen wir die Eigenschaft *Spaltenanzahl* auf *5* und *Spaltenbreiten* auf *0cm;5cm;2cm;2cm;4cm* ein (Sie werden hier ein wenig experimentieren müssen, um eine passende Einstellung zu finden, bei der anschließend alle Einträge gut sichtbar sind). Das allein hilft jedoch nicht weiter, da die Breite des aufgeklappten Kombinationsfeldes mit der Eigenschaft *Listenbreite* auf den Wert *Automatisch* eingestellt ist. Dadurch ist die aufgeklappte Liste genauso breit wie das Steuerelement.

Wenn Sie wollen, dass alle Einträge sichtbar sind, stellen Sie die Eigenschaft *Listenbreite* auf einen Wert ein, welcher der Gesamtbreite aller Spalten entspricht – in diesem Fall also *16cm*. Das Ergebnis sieht dann wie in Abbildung 12.39 aus.

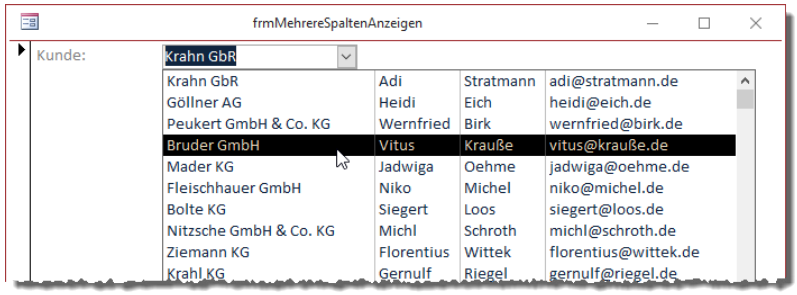


Abbildung 12.39: Anzeige mehrerer Felder beim Aufklappen des Kombinationsfeldes

12.6.4 Anzeige mehrerer Felder als Ausdruck

Manchmal reicht es nicht aus, die Detailinformationen erst im ausgeklappten Kombinationsfeld anzuzeigen. Dann möchte der Benutzer vielleicht schon als Wert des Kombinationsfeldes die Informationen aus mehreren Feldern sehen. Auch das ist kein Problem, denn Sie können ja eine beliebige Abfrage als Datensatzherkunft angeben.

Also auch eine solche, die in einem Feld einen Ausdruck mit dem Inhalt mehrerer Felder zusammenstellt. Und solange Sie für die gebundene Spalte den Primärschlüsselwert des Datensatzes verwenden, aus dem die Daten stammen, können Sie mit dem Kombinationsfeld ja auch gebundene Felder darstellen. Dazu definieren wir zunächst einmal eine passende Datensatzherkunft, welche für die zweite Spalte den Nachnamen, ein Komma und den Vornamen zusammenstellt (siehe Abbildung 12.40).

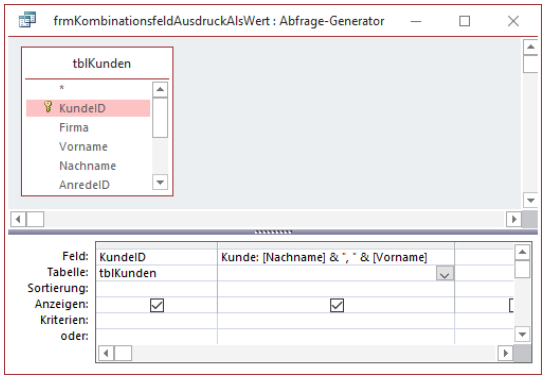


Abbildung 12.40: Datensatzherkunft mit Ausdruck als zweites Feld

Danach brauchen Sie nur noch die Eigenschaft *Spaltenanzahl* auf *2* und *Spaltenbreiten* auf *0cm* einzustellen, um das Ergebnis aus zu Abbildung 12.41 erhalten.

Abbildung 12.41: Anzeige eines Ausdrucks aus Nachname und Vorname

12.6.5 Anzahl der Einträge ermitteln

Die Anzahl der Einträge eines Kombinationsfeldes ermitteln Sie mit der Eigenschaft *ListCount*. Wenn Sie ein Formular namens *frmKombinationsfeldListCount* geöffnet haben und die Anzahl der Datensätze im Kombinationsfeld *cboListcount* ermitteln wollen, verwenden Sie etwa folgende Anweisung:

```
MsgBox "Das Kombinationsfeld enthält " & Me!cboListCount.ListCount & " Einträge."
```

12.6.6 Wertlisteneinträge per VBA als RowSource einfügen

Um Werte zu einem Kombinationsfeld mit der Einstellung *Wertliste* für die Eigenschaft *Herkunftsart* zuzuweisen, gibt es zwei Möglichkeiten. Die erste stellt die Liste in einem String zusammen und weist diesen einfach dem Wert *RowSource* des Kombinationsfeldes zu:

```
Private Sub cmdRowSource_Click()
    Dim strRowSource As String
    Dim i As Integer
    For i = 1 To 10
        strRowSource = strRowSource & "Eintrag " & i & ";"
    Next i
    strRowSource = Left(strRowSource, Len(strRowSource) - 1)
    Me!cboRowsource.RowSource = strRowSource
End Sub
```

Wir durchlaufen hier eine Schleife, in der wir jedes Mal einen neuen Eintrag namens *Eintrag x;* zur Variablen *strRowSource* hinzufügen, wobei *x* dem Wert der Zählervariablen entspricht. Dann trennen wir das hintere Semikolon ab, indem wir die Länge der Zeichenkette ermitteln, den Wert *1* abziehen und dann mit der *Left*-Funktion den Teil der Zeichenkette ohne das letzte Zeichen ermitteln. Diese Zeichenkette weisen wir dann der Eigenschaft *RowSource* zu, was im Ergebnis wie in Abbildung 12.42 aussieht.

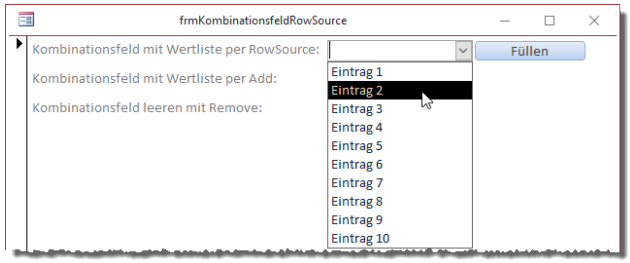


Abbildung 12.42: Per *RowSource* eingestellte Datensatzherkunft

12.6.7 Herkunftsart per VBA auf Wertliste einstellen

Achtung: Damit dies klappt, muss die Eigenschaft *Herkunftsart* des Kombinationsfeldes auf *Wertliste* eingestellt sein. Dies können Sie auch erledigen, indem Sie die Eigenschaft vorsichtshalber per VBA einstellen. Dazu würden Sie diese Anweisung vor den anderen Befehlen in der Prozedur einfügen:

```
Me!cboAdd.RowSourceType = "Value List"
```

12.6.8 Kombinationsfeld per Code aufklappen

Manchmal kann es sehr ergonomisch sein, wenn Sie ein Kombinationsfeld automatisch für den Benutzer aufklappen. Zum Beispiel, wenn er einen Datensatz speichern will, aber noch keinen Wert für dieses Kombinationsfeld ausgewählt hat. Dann können Sie ihm eine Meldung anzeigen und danach gleich das noch leere Kombinationsfeld aufklappen. Das erledigen wir mit der

Methode *DropDown* des Kombinationsfeldes. Hierbei ist es wichtig zu wissen, dass Sie dazu zuvor den Fokus auf dieses Kombinationsfeld verschieben müssen, was Sie wiederum mit der *SetFocus*-Methode erledigen.

Wenn der Benutzer im Formular *frmKombinationsfeldDropDown* auf die Schaltfläche klicke, ohne einen Eintrag im Kombinationsfeld auszuwählen, erscheint eine Meldung. Nach Bestätigung durch den Benutzer verschieben wir den Fokus auf das Kombinationsfeld und klappen dieses auf:

```
Private Sub cmdEingabeAbschliessen_Click()
    If IsNull(Me!cboAnredeID) Then
        MsgBox "Bitte wählen Sie eine Anrede aus."
        Me!cboAnredeID.SetFocus
        Me!cboAnredeID.Dropdown
    End If
End Sub
```

Das Ergebnis sieht wie in Abbildung 12.43 aus.

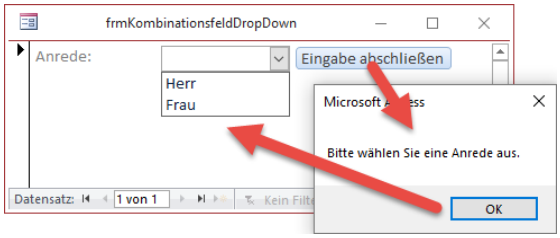


Abbildung 12.43: Aufklappen des Kombinationsfeldes

12.6.9 Wertlisteneinträge per AddItem hinzufügen

Bei der zweiten verwenden wir die Methode *AddItem*, um jeweils einen Eintrag zum Kombinationsfeld hinzuzufügen. Damit Sie direkt sehen, wie sich die Änderung auswirkt, ohne dass Sie das Kombinationsfeld immer erst manuell aufklappen müssen, erledigen wir das hier per Code:

```
Private Sub cmdEintragHinzufuegen_Click()
    Me!cboAdd.AddItem "Eintrag " & Me!cboAdd.ListCount + 1
    Me.cboAdd.SetFocus
    Me!cboAdd.Dropdown
End Sub
```

Den Namen des jeweils neuesten Eintrags stellen wir aus dem Text *Eintrag* und der aktuellen Anzahl der Listeneinträge plus dem Wert *1* zusammen. Das sieht nach drei Klicks auf die Schaltfläche *cmdEintragHinzufuegen* dann wie in Abbildung 12.44 aus.

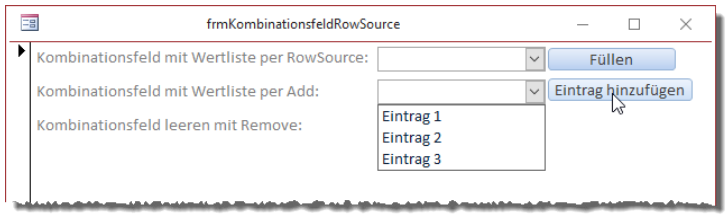


Abbildung 12.44: Hinzufügen von Einträgen per *Add*-Methode

12.6.10 Wertlisteneinträge per RemoveItem entfernen

Wo es eine *AddItem*-Methode gibt, gibt es auch eine *RemoveItem*-Methode. Damit können Sie zur Laufzeit Einträge aus der Wertliste eines Kombinationsfeldes entfernen. Das gelingt beispielsweise wie folgt:

```
Private Sub cmdEintragEntfernen_Click()
    Me!cboRemove.RemoveItem 0
    Me!cboRemove.SetFocus
    Me!cboRemove.Dropdown
End Sub
```

Hier entfernen wir immer den ersten Eintrag, also den mit dem Index *0*. Wenn Sie den letzten Eintrag entfernen wollen, ermitteln wir den Index dieses Eintrags mit der Eigenschaft *ListCount* und entfernen dann den Eintrag mit diesem Wert minus *1* als Index:

```
Me!cboRemove.RemoveItem Me!cboRemove.ListCount - 1
```

Wenn Sie einen Eintrag mit einem Indexwert entfernen wollen, der nicht vorhanden ist, erhalten Sie eine Fehlermeldung. Wenn Sie diese verhindern wollen, können Sie so etwas machen:

```
Dim intEntfernen As Integer
intEntfernen = Me!cboRemove.ListCount - 1
If Not intEntfernen < 0 Then
    Me!cboRemove.RemoveItem
End If
```

12.6.11 Bestimmten Eintrag auswählen

Es kommt relativ häufig vor, dass Sie einen bestimmten Eintrag in einem Kombinationsfeld vorauswählen wollen. Das dient dann beispielsweise dazu, zu verhindern, dass eine nachfolgende Funktion komplett ohne eine Auswahl ausgeführt werden kann. Wenn Sie in einem Kombinationsfeld etwa mit den Datensätzen der Tabelle *tblKategorien* gleich beim Öffnen eines Formulars den ersten Eintrag auswählen wollen, müssten Sie dem Kombinationsfeld als

Wert den Primärschlüsselwert des ersten Eintrags zuweisen. Den müssen wir zuvor ermitteln. Den Wert eines Eintrags mit einem bestimmten Indexwert ermitteln Sie mit der *ItemData*-Eigenschaft. Wenn Sie den Wert der gebundenen Spalte für den ersten Eintrag ermitteln wollen, der ja den Index *0* besitzt, würden Sie diesen mit der Eigenschaft *ItemData(0)* ermitteln. Das Ergebnis weisen wir dann dem Steuerelement als Wert zu. Das sieht dann etwa beim Öffnen des Formulars wie folgt aus:

```
Private Sub Form_Load()
    Me!cboKategorieID = Me!cboKategorieID.ItemData(0)
End Sub
```

Das Ergebnis sehen Sie in Abbildung 12.45.

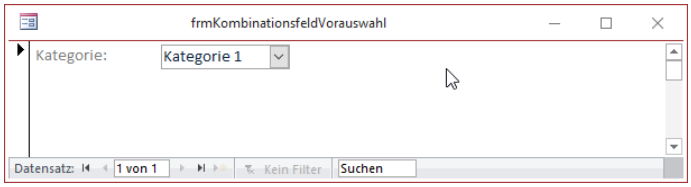


Abbildung 12.45: Voreinstellen eines Wertes für ein Kombinationsfeld

Wenn Sie den Wert der gebundenen Spalte für den auszuwählenden Eintrag kennen und es nicht beispielsweise der erste Eintrag sein soll, ist es noch einfacher: Sie brauchen der Eigenschaft *Value* (die nicht explizit angegeben werden muss, da es die Standardeigenschaft des Steuerelements ist) dann nur noch den Wert der gewünschten Kategorie zu übergeben (siehe Abbildung 12.46):

```
Private Sub cmdEintragAuswaehlen_Click()
    Me!cboKategorieMitWertID = 4
End Sub
```

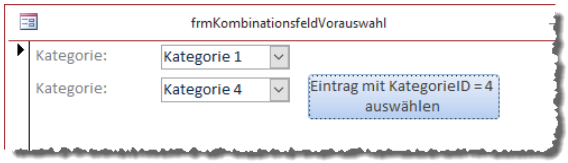


Abbildung 12.46: Direkte Auswahl eines Wertes

12.6.12 Wert und Index des aktuellen Eintrags ermitteln

Wenn das Kombinationsfeld nicht gerade ein gebundenes Feld ist, mit dem der Benutzer den Wert für ein Fremdschlüsselfeld auswählt, möchten Sie vielleicht nach der Auswahl den Wert

des Eintrags ermitteln, den der Benutzer gewählt hat. Oder Sie wollen wissen, welchen Index der gewählte Eintrag hat. Das fragen wir im folgenden Beispiel in einer Ereignisprozedur ab, die durch das Ereignis *Nach Aktualisierung* aufgerufen wird und die Werte in zwei dafür vorgesehene Textfelder einträgt (siehe Abbildung 12.47):

```
Private Sub cboArtikelID_AfterUpdate()
    Me!txtIndex = Me!cboArtikelID.ListIndex
    Me!txtWert = Me!cboArtikelID.Value
End Sub
```

Wir erhalten den Index also über die Eigenschaft *ListIndex*. Den Wert können wir über die Eigenschaft *Value* ermitteln, aber auch durch Angabe des Steuerelements allein (also nur *Me!cboArtikelID*).

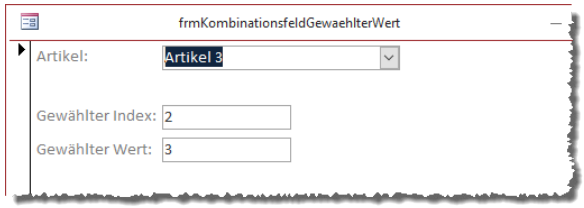


Abbildung 12.47: Index und Primärschlüsselwert des gewählten Eintrags

12.6.13 Werte aller Spalten eines Eintrags ermitteln

Wir kommen noch einmal auf das Beispiel mit dem Formular *frmKombinationsfeldVerborgene-Spalten* zurück. Wir möchten uns anschauen, wie Sie auf die Werte der einzelnen Spalten des aktuell ausgewählten Datensatzes zugreifen können.

Zur Erinnerung: Die Datensatzherkunft des Kombinationsfeldes enthält vier Spalten, von denen allerdings nur die zweite angezeigt wird (siehe weiter oben unter »Spaltenanzahl und Spaltenbreiten« ab Seite 287). An diese Werte kommen Sie über die *Colum*-Eigenschaft heran, der Sie den Index der gewünschten Spalte als Parameter übergeben. Der Index ist wie die meisten 0-basiert, *0* liefert also den Inhalt der ersten Spalte. Den bekommen Sie aber auch über die *Value*-Eigenschaft oder, wie in folgendem Beispiel, wieder über die Referenz auf das Steuerelement selbst:

```
Private Sub cboArtikel_AfterUpdate()
    Me!txtArtikelID = Me!cboArtikel
    Me!txtArtikel = Me!cboArtikel.Column(1)
    Me!txtEinzelpreis = Me!cboArtikel.Column(2)
    Me!txtMwStSatz = Me!cboArtikel.Column(3)
End Sub
```

Die Prozedur, die nach der Aktualisierung des Kombinationsfeldes ausgelöst wird, schreibt die vier Werte in entsprechende Textfelder (siehe Abbildung 12.48).

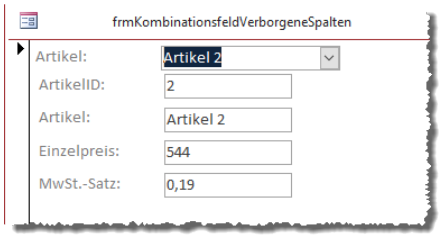


Abbildung 12.48: Ermitteln der Werte verschiedener Spalten des gewählten Eintrags

12.6.14 Abhängige Kombinationsfelder

Eine oft gestellt Frage ist, wie man nach der Auswahl eines Eintrags eines ersten Kombinationsfeldes die zur Verfügung stehenden Datensätze eines zweiten Kombinationsfeldes beeinflussen kann.

Wenn der Benutzer also beispielsweise eine Kategorie aus dem ersten Kombinationsfeld auswählt, wie kann man dann erreichen, dass das zweite Kombinationsfeld nur noch die Artikel anzeigt, welche der im ersten Kombinationsfeld gewählten Kategorie entsprechen.

Für dieses Beispiel fügen wir zwei Kombinationsfelder namens *cboKategorien* und *cboArtikel* zu einem neuen Formular hinzu. Für die erste stellen wir als Datenherkunft die Tabelle *tblKategorien* ein. Für die zweite legen wir fürs Erste die folgende SQL-Abfrage als Datensatzherkunft ein:

```
SELECT ArtikelID, Artikel FROM tblArtikel
```

Für beide Kombinationsfelder stellen wir die Eigenschaft *Spaltenanzahl* auf *2* und *Spaltenbreiten* auf *0cm* ein, damit nur jeweils das zweite Feld der Datensatzherkunft angezeigt wird.

Nun hinterlegen wir für die Ereigniseigenschaft *Nach Aktualisierung* des ersten Kombinationsfeldes die folgende Prozedur:

```
Private Sub cboKategorien_AfterUpdate()
    Me!cboArtikel.RowSource = _
        "SELECT ArtikelID, Artikel FROM tblArtikel WHERE KategorieID = " & Me!cboKategorien
End Sub
```

In dieser ermitteln wir mit *Me!cboKategorien* den Wert, den der Benutzer für das erste Kombinationsfeld ausgewählt hat. Dann fügen wir diesen Wert als Vergleichswert der *WHERE*-Bedingung der Abfrage ein und weisen diese der Eigenschaft *RowSource* des zweiten Kombinationsfeldes zu. Das Ergebnis sieht wie in Abbildung 12.49 aus.

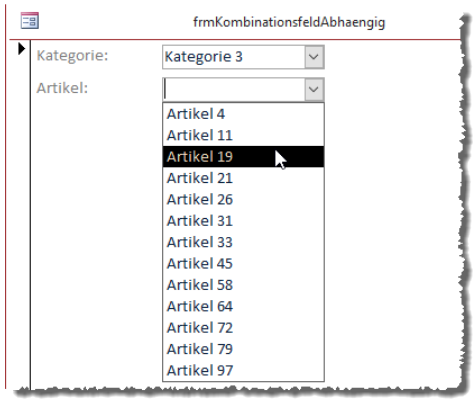


Abbildung 12.49: Nach dem ersten Kombinationsfeld gefiltertes zweites Kombinationsfeld

12.6.15 Auswählen-Eintrag hinzufügen

Vielleicht wollen Sie den Benutzer explizit darauf aufmerksam machen, dass er für ein Kombinationsfeld einen Wert auswählen soll. Das ginge beispielsweise durch die Anzeige eines Eintrags, der beispielsweise den Text *<Auswählen>* anzeigt.

Um dies zu realisieren, ohne diesen Eintrag dauerhaft in der zugrunde liegenden Tabelle zu speichern, müssen wir eine spezielle Form einer *SELECT*-Abfrage wählen. In dieser fügen wir den Eintrag mit dem Text *<Auswählen>* und mit dem Wert *0* im Feld *ArtikelID* über den ersten Teil einer *UNION*-Abfrage zur Datensatzherkunft hinzu:

```
SELECT 0 AS ArtikelID, '<Auswählen>' AS Artikel FROM tblArtikel
UNION
SELECT ArtikelID, Artikel FROM tblArtikel ORDER BY Artikel
```

Nun müssen wir nur noch sicherstellen, dass der *<Auswählen>*-Eintrag auch beim Öffnen des Formulars im Kombinationsfeld angezeigt wird. Dazu nutzen wir die folgende Prozedur, die durch das Ereignis beim Laden ausgelöst wird:

```
Private Sub Form_Load()
    Me!cboArtikel = Me!cboArtikel.ItemData(0)
End Sub
```

Das Ergebnis sieht nach dem Öffnen des Formulars und nach dem Aufklappen des Kombinationsfeldes wie in Abbildung 12.50 aus.

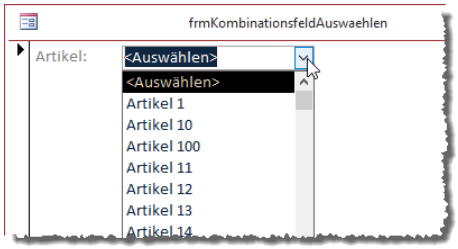


Abbildung 12.50: Kombinationsfeld mit *<Auswählen>*-Eintrag

12.6.16 Neuer Datensatz im Kombinationsfeld

Einer der praktischsten Einsatzfälle für das Kombinationsfeld ist es, einen neuen Eintrag über das Kombinationsfeld einzugeben und diesen direkt in der zugrunde liegenden Tabelle zu speichern. Dazu benötigen wir nun ein Kombinationsfeld, das beispielsweise an eine einfache Tabelle gebunden ist – also eine Tabelle, bei der man für das Anlegen eines neuen Datensatzes nur den Wert des im Kombinationsfeld anzuzeigenden Feldes eingeben muss. Beispiele dafür sind *tblAnreden* oder *tblKategorien*. Wobei man Anreden wohl eher weniger ständig ergänzt – also schauen wir uns das Beispiel anhand der Tabelle *tblKategorien* an.

Als Datensatzherkunft des Kombinationsfeldes geben wir die folgende SQL-Abfrage an:

```
SELECT KategorieID, Kategorie FROM tblKategorien ORDER BYKategorie;
```

Damit es nur den Kategorienamen anzeigt, stellen wir wieder die Eigenschaft *Spaltenanzahl* auf *2* und *Spaltenbreiten* auf *0cm* ein.

Ob das Kombinationsfeld an ein Feld des gegebenenfalls mit einer Datensatzquelle versehenen Formulars gebunden ist, spielt in diesem Beispiel keine Rolle, daher haben wir das Beispielformular *frmKombinationsfeldNeuerEintrag* als ungebundenes Formular erstellt.

Für die Ereignisprozedur *Bei nicht in Liste* hinterlegen wir nun die folgende Ereignisprozedur:

```
Private Sub cboKategorien_NotInList(NewData As String, Response As Integer)
    Dim db As DAO.Database
    Set db = CurrentDb
    db.Execute "INSERT INTO tblKategorien(Kategorie) VALUES('" & NewData & "')", _
        dbFailOnError
    Response = acDataErrAdded
End Sub
```

Diese Prozedur referenziert mit der Variablen *db* das *Database*-Objekt der aktuellen Datenbank und führt für dieses eine *INSERT INTO*-Abfrage aus, die einen neuen Datensatz zur Tabelle *tblKa-*

tegorien hinzufügt und dieser für das Feld *Kategorie* den mit dem Parameter *NewData* übergibt. Der neue Eintrag bleibt danach direkt im Kombinationsfeld stehen (siehe Abbildung 12.51).

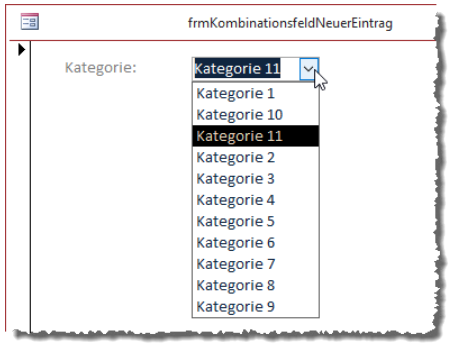


Abbildung 12.51: Neuer Eintrag im Kombinationsfeld

Neuer Datensatz mit Rückfrage

Gegebenenfalls möchten Sie verhindern, dass der Benutzer unbewusst neue Datensätze zur Datensatzherkunft des Kombinationsfeldes hinzufügt. In diesem Fall schalten wir eine *MsgBox*-Anweisung vor das Anlegen des Datensatzes. Die Ereignisprozedur sieht nun wie folgt aus:

```
Private Sub cboKategorien_NotInList(NewData As String, Response As Integer)
    Dim db As DAO.Database
    If MsgBox("Eintrag '" & NewData & "' zu den Kategorien hinzufügen?", vbYesNo) _
            = vbYes Then
        Set db = CurrentDb
        db.Execute "INSERT INTO tblKategorien(Kategorie) VALUES('" & NewData & "')", _
            dbFailOnError
        Response = acDataErrAdded
    Else
        Response = acDataErrContinue
    End If
End Sub
```

Wir fragen also per *MsgBox* ab, ob der Benutzer den Eintrag aus *NewData* als neuen Datensatz zur Tabelle *tblKategorien* hinzufügen möchte. Falls ja, wird dieser wie zuvor in die Tabelle eingetragen und der neue Wert mit dem Wert *acDataErrAdded* für den Parameter *Response* übernommen. Falls der Benutzer in der *MsgBox* auf *Nein* klickt, stellen wir nur den Parameter *Response* auf den Wert *acDataErrContinue* ein, damit die Access-eigene Meldung unterdrückt wird, die sonst nun noch erscheinen würde. Die von der Prozedur generierte Meldung sieht wie in Abbildung 12.52 aus.

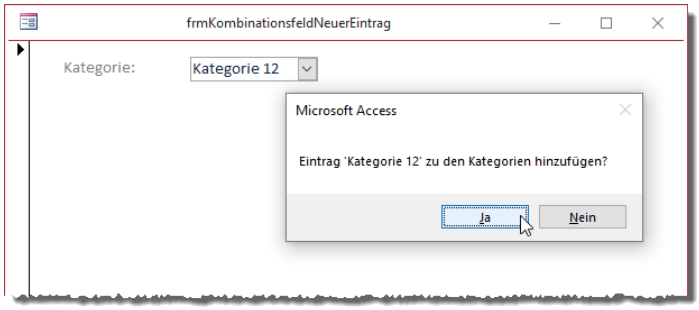


Abbildung 12.52: Rückfrage vor dem Anlegen eines neuen Eintrags in der Tabelle *tblKategorien*

12.7 Das Listenfeld

Das Listenfeld hat sehr viele Gemeinsamkeiten mit dem Kombinationsfeld, aber diese treten meist im Hintergrund zutage. Optisch sieht ein Listenfeld ganz anders aus als ein Kombinationsfeld, denn es zeigt erstens normalerweise mehrere Einträge der Datensatzherkunft an, während das Kombinationsfeld immer nur den ausgewählten Eintrag liefert und die übrigen Einträge der Datensatzherkunft nur beim Aufklappen angezeigt werden. Das Listenfeld kann außerdem gleichzeitig mehrere Spalten anzeigen und sogar Spaltenüberschriften anzeigen. Ein weiterer Vorteil des Listenfeldes ist, dass Sie damit nicht nur einen, sondern auf Wunsch auch mehrere Einträge gleichzeitig auswählen können. Die folgenden Abschnitte zeigen, wie Sie ein Listenfeld mit Daten füllen, die Anzeige der Daten beeinflussen und Einträge auswählen oder die Auswahl auslesen können.

12.7.1 Listenfeld anlegen

Wenn Sie ein Listenfeld anlegen, indem Sie dieses in der Entwurfsansicht eines Formulars im Ribbon unter *Entwurf|Steuerelemente|Listenfeld* markieren und es dann per Mausklick im Detailbereich ablegen, landet das Beschriftungsfeld in der Regel links neben dem Listenfeld. Bei einem Listenfeld soll die Beschriftung allerdings meist eher darüber angelegt werden. Das können Sie erreichen, indem Sie, wie weiter oben unter »Formular-Vorlage« ab Seite 133 beschrieben, die Standardeigenschaften dieses Steuerelements in der Formularvorlage *Normal* anpassen. Dazu stellen wir die Eigenschaft *Bezeichnungsfeld X* auf den Wert *0cm* und den Wert *Bezeichnungsfeld Y* auf den Wert *-0,6cm* ein. Wenn Sie dies im Standardformular *Normal* erledigen und dann ein neues Formular etwa über den Ribbon-Eintrag *Erstellen|Formulare|Formularentwurf* anlegen, können Sie das Listenfeld bereits in der neuen Konfiguration hinzufügen. Das Ergebnis sieht dann etwa wie in Abbildung 12.53 aus.

13 Validierung in Formularen

Zur ergonomischen Programmierung gehört, dass Sie keine ungültigen Daten zulassen. Das fängt bei der Eingabe von Daten in Formularen an. Dieser Teil beschreibt die unterschiedlichen Möglichkeiten zur Validierung der Eingabe. Die wichtigste Frage bei der Validierung ist nicht, wie diese erfolgen soll, sondern zu welchem Zeitpunkt. Die folgenden drei Regeln sind praxiserprobt:

» Fehlerhafte Eingaben wie Zeichenketten in Datumsfeldern werden sofort geahndet.

» Fehlende Eingaben werden erst beim Bestätigen des Datensatzes bemängelt.

» Die Handhabung von abhängigen Feldern ist Geschmackssache – Beispiel *Startdatum* und *Enddatum*. Am einfachsten ist eine Prüfung beim Bestätigen des Datensatzes.

13.1 Validieren direkt bei der Eingabe

Das Validieren unmittelbar nach der Eingabe erfolgt in einer Prozedur, die durch die Ereigniseigenschaft *Vor Aktualisierung* des jeweiligen Steuerelements ausgeführt wird. Eine solche Routine könnte beispielsweise wie folgt aussehen:

```
Private Sub Vorname_BeforeUpdate(Cancel As Integer)
    If IsNumeric(Left(Me!Vorname, 1)) Then
        MsgBox "Der Vorname muss mit einem Buchstaben beginnen.", _
            vbOKOnly + vbExclamation, "Eingabefehler"
        Cancel = True
        Exit Sub
    End If
End Sub
```

Die Routine prüft, ob die in das Feld *Vorname* eingegebenen Werte nicht mit einer Zahl anfangen. Sollte dies doch der Fall sein, erscheint eine entsprechende Meldung (siehe Abbildung 13.1).

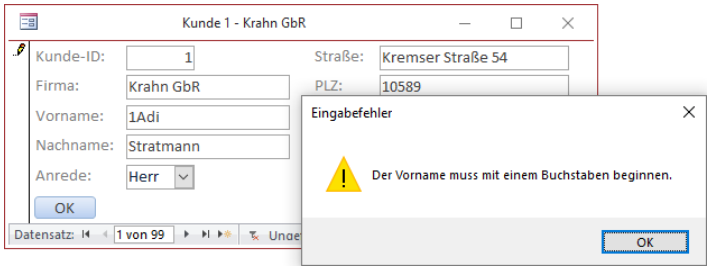


Abbildung 13.1: Vornamen, die mit einer Zahl beginnen, sind nicht erlaubt (*frmKundendetails*)

13.2 Validieren vor dem Speichern

Vor dem Speichern werden vor allem Pflichtfelder geprüft. Falls im Formular *frmBestellungen* die Felder *KundeID* und *Bestelldatum* Pflichtfelder wären, würde die Prüfung wie folgt aussehen:

```
Private Function Validierung() As Boolean
    If IsNull(Me!KundeID) Then
        MsgBox "Bitte wählen Sie einen Kunden aus.", _
            vbOKOnly + vbExclamation, "Fehlende Eingabe"
        Me!KundeID.SetFocus
        Exit Function
    End If
    If IsNull(Me!Bestelldatum) Then
        MsgBox "Bitte geben Sie das Bestelldatum ein.", _
            vbOKOnly + vbExclamation, "Fehlende Eingabe"
        Me!Bestelldatum.SetFocus
        Exit Function
    End If
    If Not IsNull(Me!Bestelldatum) And Not IsNull(Me!Stornierungsdatum) Then
        If Me!Bestelldatum > Me!Stornierungsdatum Then
            MsgBox "Das Bestelldatum darf nicht hinter " _
                & "dem Stornierungsdatum liegen.", _
                vbOKOnly + vbExclamation, "Falsche Datumsangabe"
            Me!Stornierungsdatum.SetFocus
            Exit Function
        End If
    End If
    Validierung = True
End Function
```

Die Prozedur prüft von oben nach unten alle Pflichtfelder. Sobald sie auf eines trifft, das leer ist, gibt sie eine Fehlermeldung aus, setzt den Fokus auf das Feld mit dem fehlenden Inhalt und bricht die Prozedur ab. Außerdem wird der Funktionswert nur dann auf *True* gesetzt, wenn alle Validierungen erfolgreich verlaufen sind.

13.2.1 Validieren abhängiger Felder

Im letzten Teil der Routine *Validieren* finden Sie ein Beispiel, wie sich abhängige Felder prüfen lassen. In diesem Beispiel wird sichergestellt, dass das Bestelldatum nicht hinter dem Stornierungsdatum liegt. Das Ergebnis sieht etwa wie in Abbildung 13.2 aus.

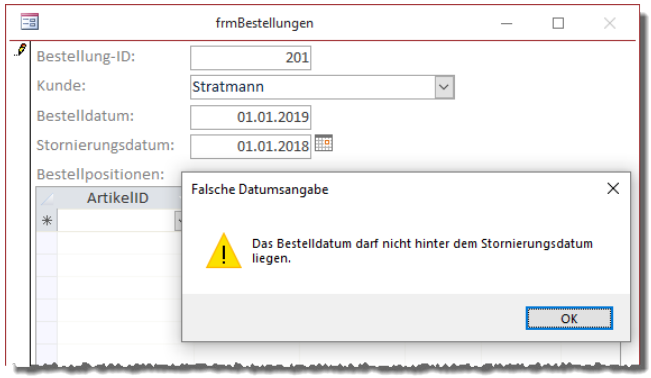


Abbildung 13.2: Validierungsmeldung bei abhängigen Datumsangaben (Formular *frmBestellungen*)

13.2.2 Aufruf der Validierung vor dem Speichern

Nun müssen Sie diese Funktion nur noch von geeigneter Stelle aus aufrufen. Leider reicht es nicht, dies mit der Ereigniseigenschaft *Vor Aktualisierung* abzudecken. Die dazugehörende Ereignisprozedur enthält auch den passenden *Cancel*-Parameter, der das Speichern des Datensatzes bei falschen oder fehlenden Daten abbrechen könnte.

Es gibt allerdings eine Schwachstelle: Theoretisch wird diese Routine zwar durch alle relevanten Vorgänge wie Wechsel des Datensatzes oder Schließen des Formulars aufgerufen – aber wenn das Schließen einmal ausgelöst wurde, hält auch das Setzen des *Cancel*-Parameters der *Form_BeforeUpdate*-Prozedur das Schließen nicht mehr auf.

Somit würde in diesem Fall ein Datensatz mit falschen oder fehlenden Daten gespeichert. Daher rufen Sie die Funktion *Validierung* von zwei Ereignisprozeduren aus auf: Von der *Form_BeforeUpdate*-Prozedur sowie von der *cmdOK_Click*-Prozedur aus. Um auszuschließen, dass das Formular anders als mit der *OK*-Schaltfläche geschlossen wird, stellen Sie noch die Eigenschaft *Schließen*-Schaltfläche auf den Wert *Nein* ein.

14 Bilder in Formularen

Seit einigen Access-Versionen, genau gesagt seit Access 2010, ist die Verwendung von Bildern in Formularen etwa zur Anzeige als Hintergrund, als eigenes Bild oder auch als Icon einer Schaltfläche erheblich einfacher geworden. Wir schauen uns in diesem Kapitel an, wie Sie Bilder in Schaltflächen oder einfach als festes Bild zu einem Formular hinzufügen, aber auch, wie Sie diese in Tabellen speichern und dann mit den entsprechenden Datensätzen im Formular präsentieren.

14.1 Bilder in Formularen oder auf Schaltflächen

Die Vereinfachung beim Umgang mit Bildern kommt dadurch, dass Bilder nun nicht mehr etwa im OLE-Format direkt im Code des Formulars integriert werden, sondern in einer eigens dafür vorgesehenen Tabelle landen (die Tabelle kann auch noch für andere Zwecke eingesetzt werden, aber für das Speichern von Bildern ist sie prädestiniert).

14.1.1 Bild hinzufügen

Schauen wir uns an, wie einem Formular ein Bild hinzufügen. Dazu beginnen wir mit einem leeren Formular. Für dieses rufen wir nun im Ribbon den Befehl *Entwurf|Steuerelemente|Bild einfügen|Durchsuchen...* auf (siehe Abbildung 14.1).

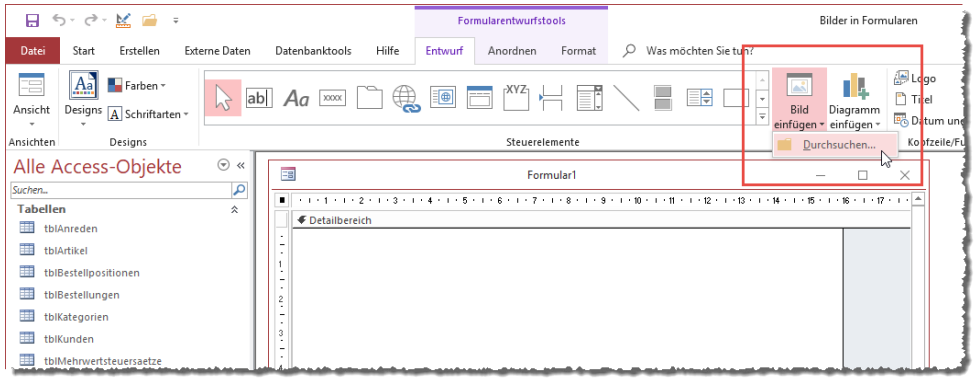


Abbildung 14.1: Hinzufügen von Bildern

Es erscheint ein *Grafik einfügen*-Dialog, mit dem wir eine geeignete Bilddatei auswählen. PNG ist günstig, da es normalerweise etwa einen transparenten Anteil mitliefert und die Icons so gut etwa auf Schaltflächen abgebildet werden können. Eine gute Icon-Sammlung gibt es bei-

spielsweise unter *www.iconexperience.de*. Diese verwenden wir im Rahmen der Beispiele dieses Buchs. Wir wählen eine Bilddatei mit der Größe 32x32 Pixel aus (siehe Abbildung 14.2).

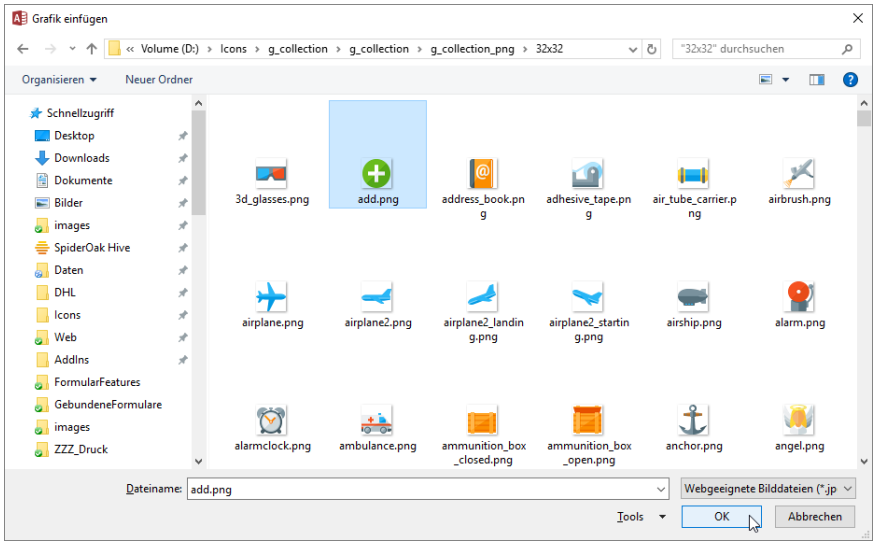


Abbildung 14.2: Auswählen einer Bilddatei

Danach bewegen wir den Mauszeiger in den Formularentwurf, wo die Maus sich in ein Symbol zum Einfügen des Bildes verwandelt. Klicken Sie an die gewünschte Stelle, wird die Bilddatei eingefügt (siehe Abbildung 14.3).

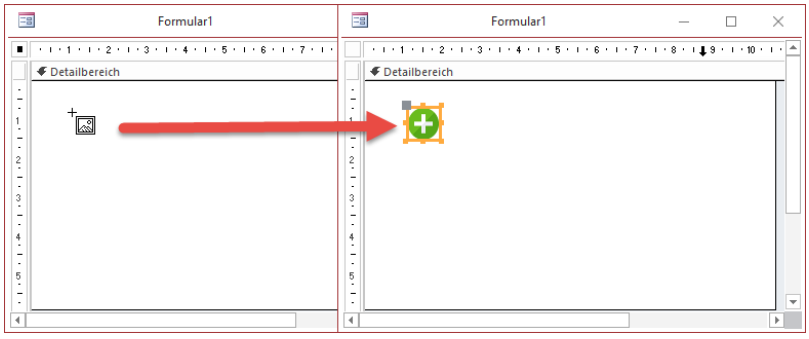


Abbildung 14.3: Einfügen der Bilddatei im Formular

14.1.2 Speicherort des Bildes

Was ist nun geschehen? Wurde das Bild einfach nur im Formular angelegt? Nein, es ist noch einiges mehr passiert. Davon können Sie sich beispielsweise überzeugen, wenn Sie nochmal

auf den Ribbon-Eintrag *Entwurf|Steuerelemente|Bild einfügen* klicken. Genau wie zuvor wird das Untermenü aufgeklappt, aber diesmal finden Sie dort einen neuen Eintrag – nämlich unser neues Bild (siehe Abbildung 14.4).

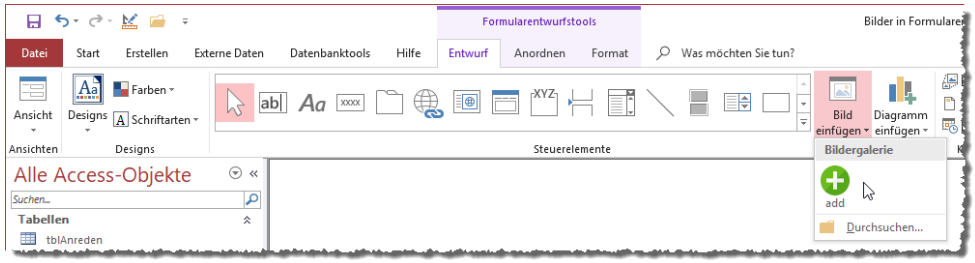


Abbildung 14.4: Die neue Bilddatei ist nun allgemein in der Anwendung einsetzbar.

Aber wo ist dieses gespeichert? Um dies zu zeigen, müssen wir zunächst die Anzeige der Systemtabellen im Navigationsbereich aktivieren. Dazu wählen Sie den Eintrag *Navigationsoptionen...* des Kontextmenüs des Titels des Navigationsbereichs aus (siehe Abbildung 14.5).

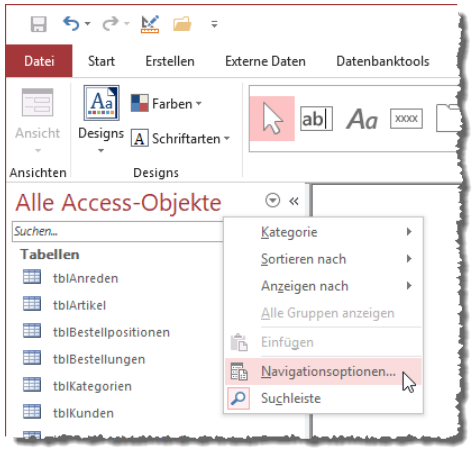


Abbildung 14.5: Öffnen der Navigationsoptionen

Dies zeigt den Dialog aus Abbildung 14.6 an. Der Dialog enthält zwei Optionen namens *Ausgeblendete Objekte anzeigen* und *Systemobjekte anzeigen*. Um ehrlich zu sein: Bei manchen Spezialtabellen wir der Tabelle, die wir hier suchen oder etwa der Tabellen zum Speichern von Ribbon-Definitionen kann ich mir nicht merken, ob es sich um Systemtabellen oder versteckte Tabellen handelt, daher aktiviere ich immer beide Optionen, wenn ich einmal eine dieser Tabellen benötige. Danach können Sie den Dialog wieder schließen.

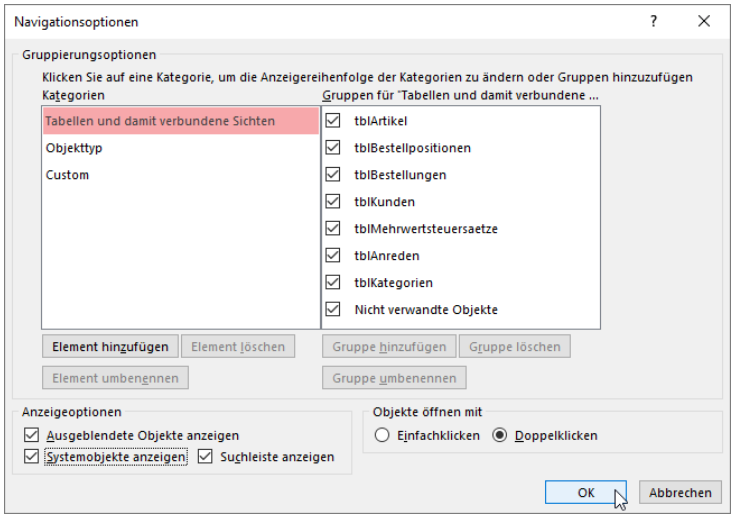


Abbildung 14.6: Aktivieren der Optionen zur Anzeige der ausgeblendeten Objekte und der Systemobjekte

Nach dem Schließen dieses Dialogs erscheinen einige ausgegraute Tabellen, die mit *MSys...* beginnen. Wir suchen die Tabelle *MSysResources*, die nach dem Öffnen wie in Abbildung 14.7 aussieht.

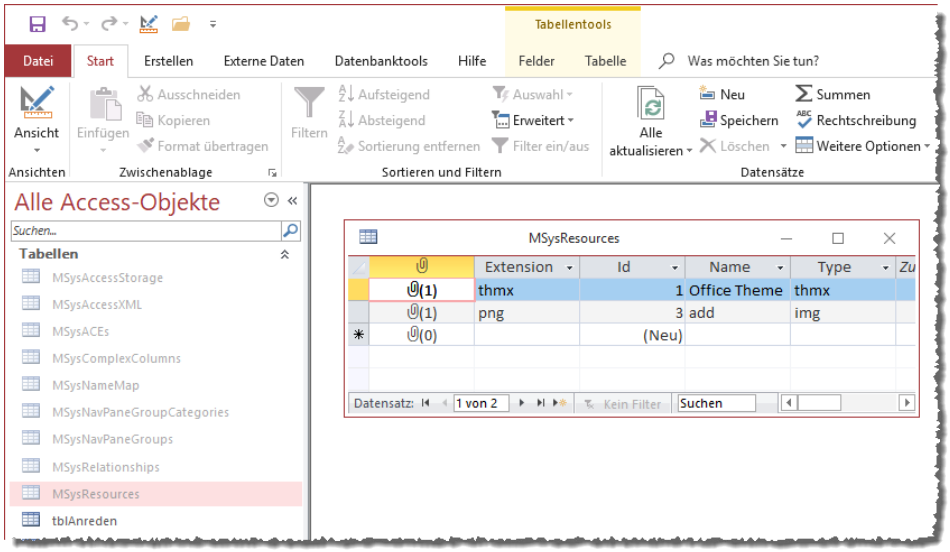


Abbildung 14.7: Öffnen und Anzeigen der Tabelle *MSysResources*

Die Tabelle *MSysResources* speichert Ressourcen für die aktuelle Datenbank, darunter auch die Bilddateien, die in Bildsteuerelementen (je nach Einstellung) oder auf Schaltflächen verwendet werden. Die Tabelle speichert die Bilddateien in einem Anlagefeld. Zusätzlich legt es die Dateinamen-Erweiterung, einen Primärschlüsselwert (*id*), den Namen ohne Dateiendung (hier *add*) und den Typ (für Bilder *img*) an.

Was können wir mit dieser Information anfangen? Wir könnten zum Beispiel, da Microsoft den Dialog zum Hinzufügen von Bilddateien leider nur für das gleichzeitige Hinzufügen einer einzigen Bilddatei ausgelegt hat, eine kleine VBA-Routine programmieren, die viele Bilddateien auf einen Rutsch zu dieser Tabelle hinzufügt. Eine solche Routine finden Sie weiter hinten unter »Bilder zur Datenbank hinzufügen« ab Seite 507.

Außerdem können wir diese Vorgehensweise aber auch für eigene Zwecke nutzen – etwa um einer Tabelle ein Feld zum Speichern von Bildern hinzuzufügen, die wir dann in Formularen anzeigen. Und für die wir im Formular die Möglichkeit anbieten, Bilder hinzuzufügen oder zu entfernen.

14.1.3 Bilder für Schaltflächen et cetera auswählen

Praktisch ist, dass Sie nun über das Ribbon auf die in der Tabelle *MSysResources* gespeicherten Bilddateien zugreifen können. Aber es gibt noch eine weitere nützliche Möglichkeit. Wenn Sie ein Bild-Steuerelement aus der Liste der Steuerelemente im Ribbon zum Formular hinzugefügt haben, können Sie diesem nun über die Eigenschaft *Bild* auswählen (siehe Abbildung 14.8).

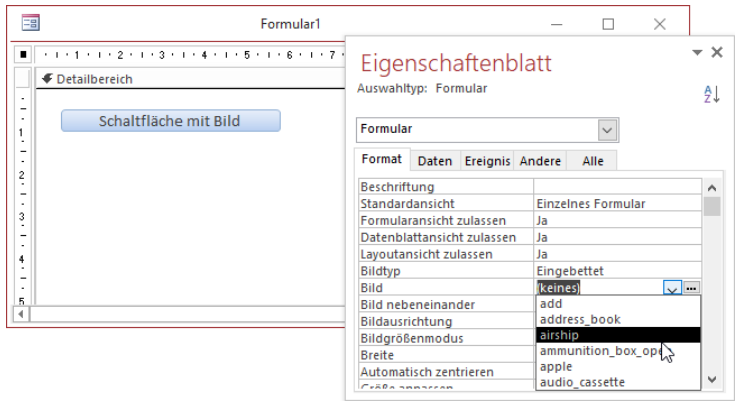


Abbildung 14.8: Bilder für eine Schaltfläche auswählen

Weiter hinten unter »Bilder zur Datenbank hinzufügen« ab Seite 507 zeigen wir, wie Sie schnell viele Bilder zur Tabelle *MSysResources* hinzufügen können.

15 ActiveX-Steuerelemente

Die Bibliothek *MSCOMCTL.ocx* bietet dem Access-Entwickler einige Steuerelemente an, von denen drei besonders beliebt sind: das *TreeView*-Steuerelement, das *ListView*-Steuerelement und das *ImageList*-Steuerelement, das die beiden zuvor genannten mit Bildern für die Icons der Einträge versorgt. Wir schauen uns diese im Detail an.

15.1 Das ImageList-Steuerelement

Wer das *TreeView*- und das *ListView*-Steuerelement nutzt, möchte in den meisten Fällen auch Icons zu den enthaltenen Einträgen anzeigen. Damit wir gleich bei der Beschreibung dieser beiden Steuerelemente aus dem Vollen schöpfen können, stellen wir dieses Steuerelement als Erstes vor. Das *ImageList*-Steuerelement arbeitet dabei als eine Art Container für die Bilddateien. Und auch wenn es mittlerweile Alternativen gibt, um Bilddateien auf einfache Weise in der Datenbank zu speichern, so kommen Sie beim *TreeView*- und beim *ListView*-Steuerelement nicht ohne das *ImageList*-Steuerelement aus.

15.1.1 ImageList-Steuerelement anlegen

Wir fügen einem neuen Formular namens *frmImageList* ein *ImageList*-Steuerelement hinzu, in dem wir zunächst über den Ribbon-Eintrag *Entwurf|Steuerelemente|ActiveX-Steuerelemente* den Dialog *ActiveX-Steuerelemente einfügen* aufrufen (siehe Abbildung 15.1).

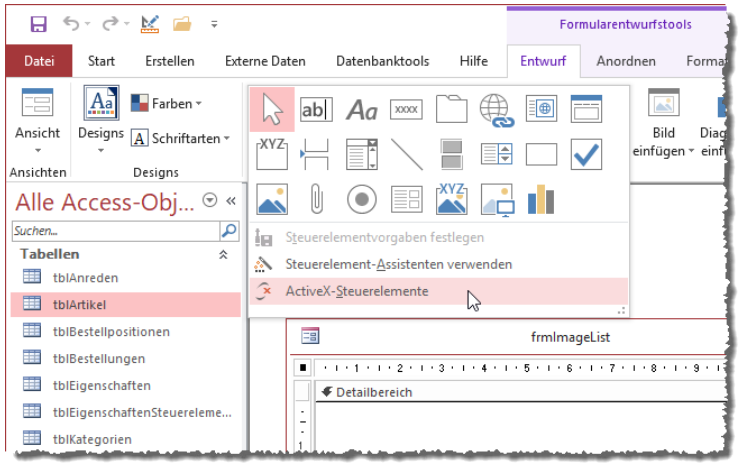


Abbildung 15.1: Aufruf des Dialogs zur Auswahl von ActiveX-Steuerelementen

Im nun erscheinenden Dialog *ActiveX-Steuerelemente einfügen* finden wir schnell den Eintrag *Microsoft ImageList Control, version 6.0*, den wir markieren und mit einem Klick auf die *OK*-Schaltfläche zum Einfügen vorbereiten (siehe Abbildung 15.2).

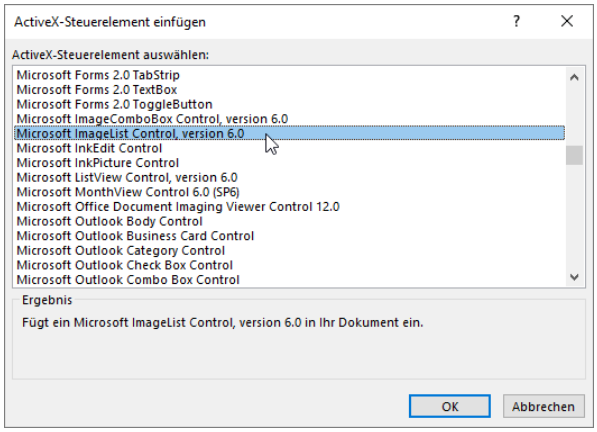


Abbildung 15.2: Auswahl des *ImageList*-Steuerelements

Danach finden Sie das neu eingefügte *ImageList*-Steuerelement in der linken oberen Ecke des Formulars wieder (siehe Abbildung 15.3). Wir legen an dieser Stelle gleich den Namen *ctlImageList* für dieses Steuerelement fest.

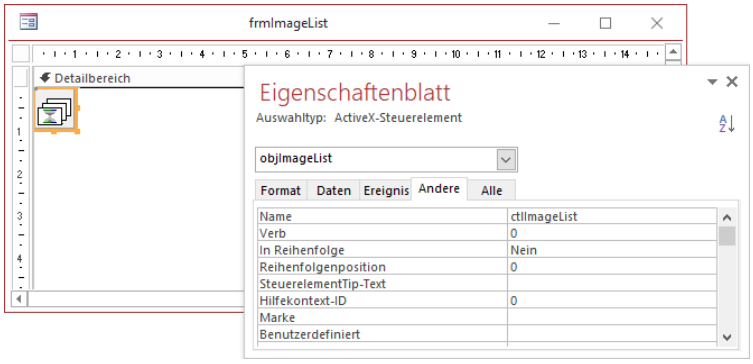


Abbildung 15.3: *ImageList*-Steuerelement im Formularentwurf

Seine Position brauchen Sie gar nicht erst zu ändern, da es nur als Container für die Bilder fungiert und in der Formularansicht nicht sichtbar ist.

15.1.2 ImageList-Steuerelement füllen

Zum Füllen des *ImageList*-Steuerelements gibt es verschiedene Wege. Wir schauen uns zwei davon an: den manuellen Weg, der empfehlenswert ist, wenn Sie eine überschaubare Anzahl von Icons im *TreeView*- und im *ListView*-Steuerelement verwenden wollen.

Alternativ können wir die Bilder auch per VBA in das *ImageList*-Steuerelement einfügen. Auch diesen Weg schauen wir uns an.

Manuelles Füllen des ImageList-Steuerelements

Wenn Sie das Steuerelement manuell mit einigen Images füllen wollen, klicken Sie doppelt auf das *ImageList*-Steuerelement. Es erscheint ein Dialog namens *Eigenschaften von ImageListCtrl*. Hier können Sie auf der ersten Registerseite zunächst die Größe der in diesem Steuerelement gespeicherten *Images* einstellen.

Die Images müssen alle die gleiche Größe aufweisen. Wenn Sie verschiedene Größe in einem Formulare benötigen, müssen Sie für jede gewünschte Größe ein eigenes *ImageList*-Steuerelement anlegen (siehe Abbildung 15.4).

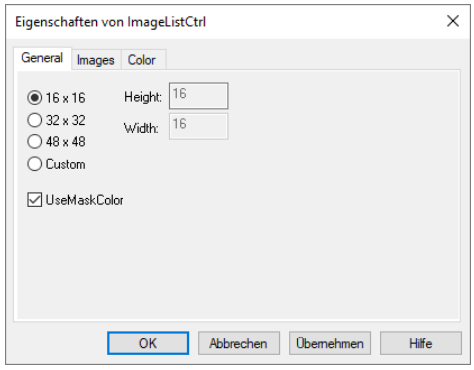


Abbildung 15.4: Einstellen der Größe der Images, die im *ImageList*-Steuerelement gespeichert werden sollen.

Auf der zweiten Registerseite finden Sie eine Schaltfläche mit dem Text *Insert Picture...* vor. Diese können Sie zum Hinzufügen von Bilddateien über einen Dateiauswahl-Dialog nutzen.

Wir fügen auf diese Weise zunächst vier Bilder mit dem Format *.ico* zum *ImageList*-Steuerelement hinzu. Dieses zeigt die Images anschließend wie in Abbildung 15.6 an.

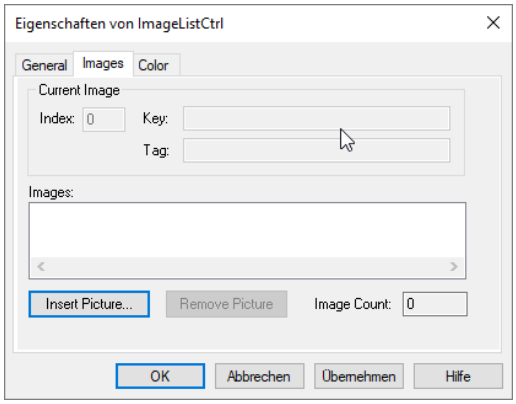


Abbildung 15.5: Dialog mit den Images

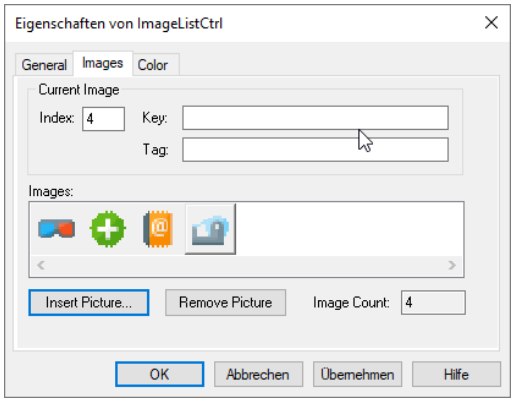


Abbildung 15.6: *ImageList*-Steuerelement mit Images

Sie können über die Eigenschaften *Key* und *Tag* für jedes Bild entsprechende Werte festlegen, über die Sie die Bilder später referenzieren. Wichtig: Sie können mehrere Bilder gleichzeitig zum *ImageList*-Steuerelement hinzufügen.

Wenn Sie die Bilder einzeln hinzufügen, weil diese sich beispielsweise in verschiedenen Verzeichnissen befinden, beachten Sie, dass neue Bilder immer rechts neben dem aktuell markierten Bild eingefügt werden.

Nachteil: Auf diese Weise können wir nur Bilder mit bestimmten Dateiendungen hinzufügen, also beispielsweise *.ico*-Dateien. *.png*-Dateien funktionieren nicht und führen zu einem Fehler (*Invalid Picture*).

Einfügen von Bildern per VBA

Per VBA ergeben sich andere Möglichkeiten. Hier verwenden wir eine Reihe von API-Funktionen, die Sascha Trowitzsch in zwei Modulen mit verschiedene Funktionen für den Einsatz von Bildern unter Access und VBA zusammengestellt hat und die seit Jahren in vielen Datenbankanwendungen im Einsatz sind.

Für die nachfolgend beschriebene Vorgehensweise gehen wir davon aus, dass Sie wie in »Bilder in Formularen« ab Seite 367 beschrieben einige Bilder in die Tabelle *MSysResources* eingefügt haben, denn auf diese greifen wir auch in diesem Abschnitt zu.

In einem Modul namens *mdlImages* haben wir die folgende Funktion angelegt, die zwei Parameter erwartet: einen Verweis auf das *ImageList*-Steuerelement sowie den Namen des Bildes in der Tabelle *MSysResources*. Die Funktion fügt das genannte Bild in das angegebene *ImageList*-Steuerelement ein.

```
Public Sub ImageHinzufuegen(objImageList As MSComctlLib.ImageList, strImage As String)
    Dim objPicture As StdPicture
    Dim lngID As Long
    lngID = BildIDErmitteln(strImage)
    Set objPicture = GetPicture(lngID)
    objImageList.ListImages.Add , strImage, objPicture
End Sub
```

Dabei verwendet diese Prozedur zunächst die Funktion *BildIDErmitteln*, welche den Namen des Bildes in der Tabelle *MSysResources* erwartet. Diese Funktion sieht wie folgt aus:

```
Public Function BildIDErmitteln(strImage As String) As Long
    Dim lngID As Long
    lngID = DLookup("Id", "MSysResources", "Name='" & strImage & "'")
    BildIDErmitteln = lngID
End Function
```

Die Funktion erwartet den Namen des Bildes in der Tabelle *MSysResources* und ermittelt per *DLookup*-Funktion den Primärschlüsselwert des entsprechenden Datensatzes, der dann auch als Ergebnis der Funktion zurückgegeben wird. Die aufrufende Prozedur *ImageHinzufuegen* ruft dann noch die Funktion *GetPicture* auf, die wir ebenfalls im gleichen Standardmodul unterbringen und die wie folgt aussieht:

```
Public Function GetPicture(lngID As Long) As StdPicture
    Dim objPicture As StdPicture
    Set objPicture = ArrayToPicture(mdlBLOBs0710.BLOB2Binary0710(CurrentDb, _
        "MSysResources", "Data", "Id", lngID, True), &HFFFFFFFF)
    Set GetPicture = objPicture
End Function
```

Mit dem hier angegebenen Aufruf der Funktion *ArrayToPicture* ermittelt die Funktion *GetPicture* mit den API-Funktionen aus den beiden Modulen *mdlBLOBS0710* und *mdlOGL0710* ein Objekt des Typs *StdPicture*, was ein gültiger Typ für die Zuweisung zu dem *ImageList*-Steuerelement ist. In den Code der beiden genannten Module wollen wir nicht tiefer einsteigen.

Wichtig ist, dass wir die im Modul *mdlImages* gespeicherten Funktionen zusammen dafür sorgen, dass die Prozedur *ImageHinzufuegen* eine in der Tabelle *MSysResources* gespeichertes Bild im *ImageList*-Steuerelement speichert.

Wir gehen an dieser Stelle davon aus, dass wir beispielsweise eine Bilddatei namens *add* in der Größe 16x16 Pixel in der Tabelle *MSysResources* gespeichert haben, die wir beim Laden des Formulars durch die folgende Ereignisprozedur in das *ImageList*-Steuerelement laden können. Zuvor deklarieren wir noch eine Objektvariable, mit der wir einen Verweis auf das *ImageList*-Steuerelement speichern:

```
Dim objImageList As MSComctlLib.ImageList
```

Damit können wir dann in der Prozedur *Form_Load* das *ImageList*-Steuerelement referenzieren, was bei ActiveX-Steuerelementen immer über die Eigenschaft *Object* des jeweiligen Steuerelements erfolgt. Mit der Prozedur *ImageHinzufuegen* fügen wir dann das Bild *add* hinzu und geben anschließend mit der *Count*-Eigenschaft der *ListImages*-Auflistung die neue Anzahl der enthaltenen Bilder aus:

```
Private Sub Form_Load()
    Set objImageList = Me!ctlImageList.Object
    ImageHinzufuegen objImageList, "add"
    Debug.Print objImageList.ListImages.count
End Sub
```

Bei einem noch leeren *ImageList*-Steuerelement sollte dies den Wert *1* im Direktbereich ausgeben. Allerdings können wir uns die neue Bilddatei noch nicht im *ImageList*-Steuerelement ansehen, da dieses hier nicht gespeichert wird. Wir schauen uns gleich unter »Images im TreeView« ab Seite 395 in den Beispielen zum *TreeView*-Steuerelement an, wie wir die hinzugefügten Images im *TreeView*-Steuerelement nutzen können.

15.2 Das TreeView-Steuerelement

Das *TreeView*-Steuerelement erlaubt die Anzeige und das Bearbeiten von hierarchischen Daten, die beispielsweise aus mehreren verknüpften Tabellen stammen können wie Kunden/Bestellungen/Bestellpositionen. Oder die Daten stammen aus reflexiv verknüpften Tabellen, die etwa die Hierarchie von Mitarbeitern über die Tabelle *tblMitarbeiter* darstellen. Das *TreeView*-Steuerelement bietet unter anderem die folgenden Design-Elemente:

» Text

» Symbole

» Plus/Minus-Zeichen zum Auf- und Zuklappen von untergeordneten Elementen

» Linien

» Kontrollkästchen zum Anhaken von Elementen

15.2.1 TreeView-Steuerelement anlegen

Das TreeView-Steuerelement fügen Sie in ein Formular ein, indem Sie zunächst über den Ribbon-Eintrag *Entwurf|Steuerelemente|ActiveX-Steuerelemente* den Dialog *ActiveX-Steuerelemente einfügen* den Dialog zur Auswahl von ActiveX-Steuerelementen öffnen. Hier finden Sie den Eintrag *Microsoft TreeView Control 6.0 (SP6)* – siehe Abbildung 15.7.

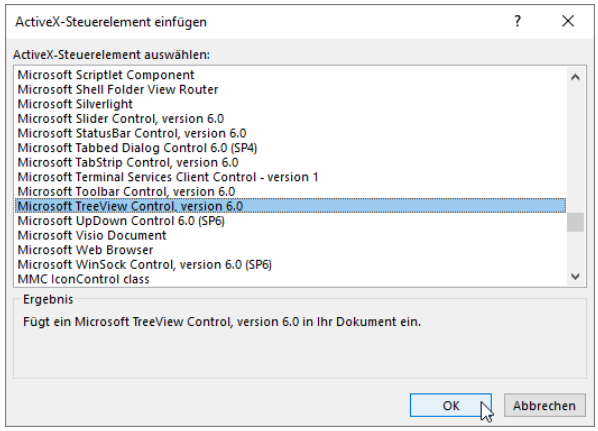


Abbildung 15.7: Einfügen eines *TreeView*-Steuerelements

Das so hinzugefügte *TreeView*-Steuerelement, das zunächst sehr klein angelegt wurde, benennen wir zunächst mit *ctlTreeView* (siehe Abbildung 15.8).

15.2.2 Verweis auf die MSCOMCTL-Bibliothek

Anschließend vergrößern wir zunächst das neue, noch leere *TreeView*-Steuerelement. Gleichzeitig mit dem Anlegen des *TreeView*-Steuerelement wurde dem VBA-Projekt der Anwendung ein Verweis auf die Bibliothek *Microsoft Windows Common Controls 6.0 (SP6)* hinzugefügt (siehe Abbildung 15.9).

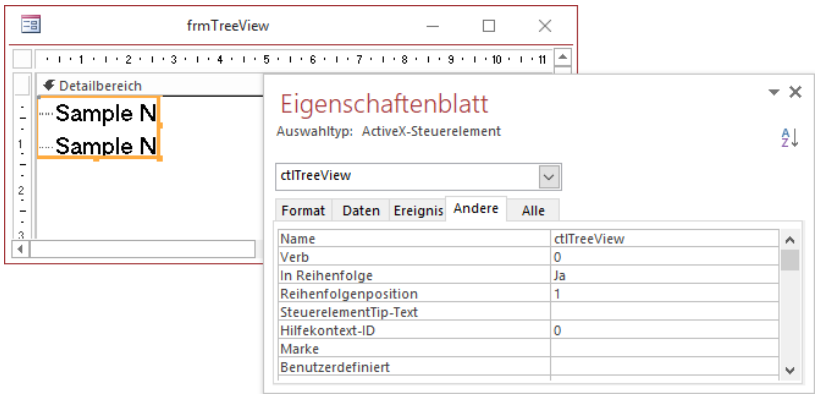


Abbildung 15.8: Das neu angelegte *TreeView*-Steuerelement

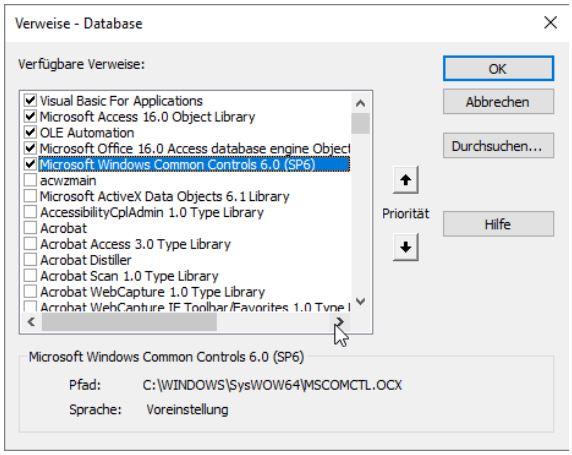


Abbildung 15.9: Verweis auf die *MSCOMCTL*-Bibliothek

15.2.3 TreeView-Element im Objektkatalog

Wenn Sie sich einen Überblick über die Methoden, Eigenschaften und Ereignisse des *TreeView*-Steuerelements verschaffen wollen, können Sie dies im Objektkatalog des VBA-Editors erledigen.

Den Objektkatalog öffnen Sie über den Menübefehl *Ansicht|Objektkatalog* oder einfach mit der Taste *F2*. Dazu stellen Sie die Bibliothek *MSComctlLib* ein und suchen beispielsweise nach *TreeView*. Unten finden Sie dann das gefundene Objekt und die zugehörigen Methoden, Eigenschaften und Ereignisse (siehe Abbildung 15.10).

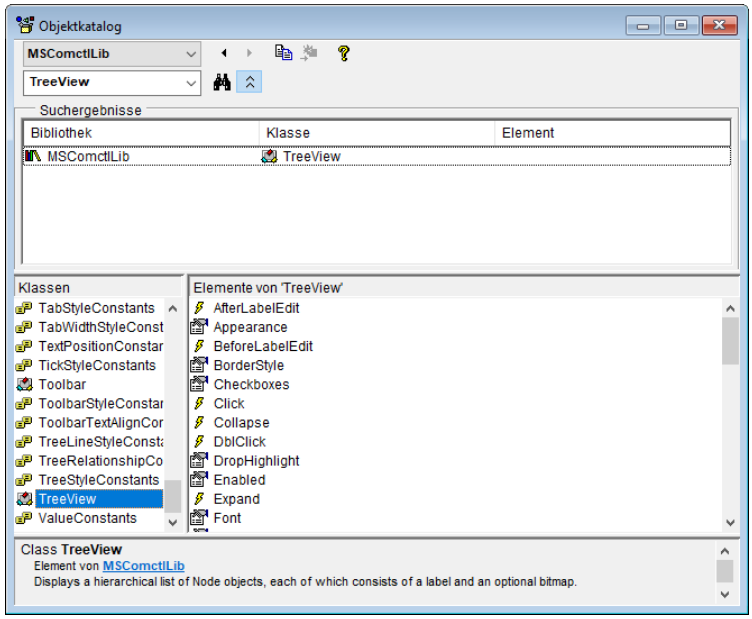


Abbildung 15.10: Methoden, Eigenschaften und Ereignisse des *TreeView*-Steuerelements

15.2.4 IntelliSense bei der TreeView-Programmierung

Wenn Sie IntelliSense verwenden möchten, was empfehlenswert ist, deklarieren Sie eine Objektvariable für das *TreeView*-Steuerelement im Modulkopf des Formulars und instanzieren diese an geeigneter Stelle. Die Deklaration sieht etwa wie folgt aus:

```
Dim objTreeView As MSComctlLib.TreeView
```

Zum Instanzieren eignet sich am besten eine der Ereignisprozeduren, die beim Öffnen des Formulars ausgeführt werden. Da mit dem *Beim Öffnen*-Ereignis Probleme mit manchen ActiveX-Steuerelementen auftreten können, verwenden Sie am besten immer das *Beim Laden*-Ereignis:

```
Private Sub Form_Load()
    Set objTreeView = Me!ctlTreeView.Object
End Sub
```

Wenn Sie in weiteren Routinen auf das Objekt *objTreeView* zugreifen, statt das Steuerelement *ctlTreeView* zu referenzieren, stellt IntelliSense die *TreeView*-spezifischen Elemente zur Verfügung (siehe Abbildung 15.11).

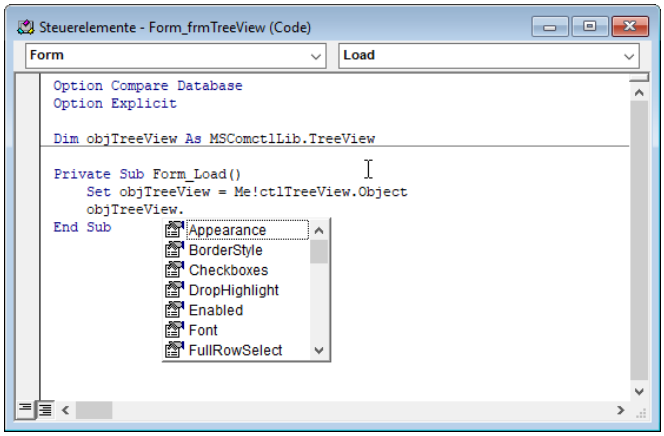


Abbildung 15.11: IntelliSense für das *TreeView*-Steuerelement

Profitipp zum TreeView

Gerade wenn Sie die Formulare mit dem *TreeView*-Steuerelement programmieren, werden Sie immer wieder Laufzeitfehler produzieren, die dafür sorgen, dass die Variable *objTreeView* nicht mehr das Steuerelement *ctlTreeView* referenziert. Solche Objektvariablen werden durch Laufzeitfehler gelöscht.

Wir können allerdings mit einer kleinen *Property Get*-Prozedur dafür sorgen, dass Sie über *objTreeView* immer auf das *TreeView*-Steuerelement zugreifen – auch wenn zwischendurch ein Laufzeitfehler aufgetreten ist. Dazu kommentieren wir erst einmal die bisher verwendete Variable *objTreeView* aus:

```
'Dim objTreeView As MSComctlLib.TreeView
```

Dann deklarieren wir stattdessen die folgende Variable:

```
Dim m_TreeView As MSComctlLib.TreeView
```

Auf diese greifen wir aber nicht von den Ereignisprozeduren aus zu, sondern nur von unserer *Property Get*-Prozedur. In dieser prüfen wir, ob *m_TreeView* leer ist, was entweder der Fall ist, wenn das Formular gerade erst geöffnet wurde oder wenn die Objektvariable durch einen nicht behandelten Laufzeitfehler geleert wurde.

Ist *m_TreeView* leer, füllen wir es auf jeden Fall wieder mit einem Verweis auf *Me!ctlTreeView.Object*. Den Inhalt von *m_TreeView* übergeben wir dann als Wert von *objTreeView*:

```
Private Property Get objTreeView() As MSComctlLib.TreeView
    If m_TreeView Is Nothing Then
        Set m_TreeView = Me!ctlTreeView.Object
```

```
    End If
    Set objTreeView = m_TreeView
End Property
```

Der Clou ist, dass Sie alle Stellen, an denen *objTreeView* verwendet wird, nicht ändern müssen. Sie müssen nur die Zeilen, in denen *objTreeView* mit einem Verweis auf *ctlTreeView* gefüllt wird, auskommentieren oder löschen. Anderenfalls löst diese Zuweisung einen Fehler aus, da *objTreeView* ja nun keine Variable mehr ist, sondern eine Eigenschaft, auf die nur lesend zugegriffen werden kann.

15.2.5 Eigenschaften des TreeView-Steuerelements

Wenn Sie mit der rechten Maustaste auf das *TreeView*-Steuerelement im Formularentwurf klicken, finden Sie dort den Eintrag *TreeCtrl-Objekt|Properties* vor (siehe Abbildung 15.12).

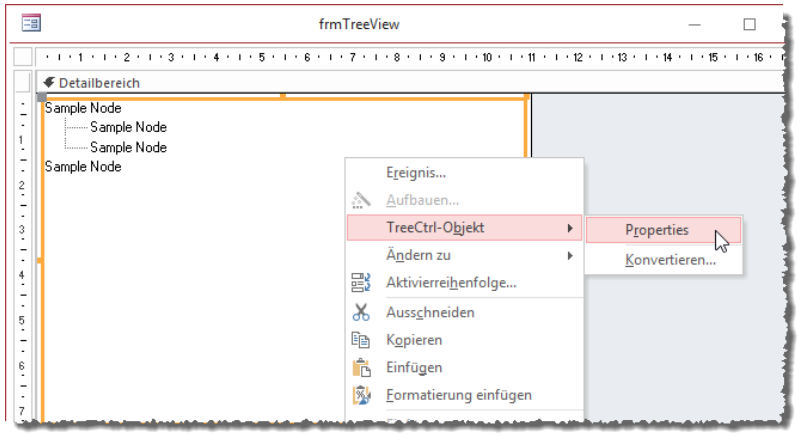


Abbildung 15.12: Aufruf des Eigenschaften-Dialogs für das *TreeView*-Steuerelement

Dieser Befehl öffnet den *Eigenschaften*-Dialog aus Abbildung 15.13. Hier können Sie fast alle Eigenschaften für das *TreeView*-Steuerelement einstellen. Den Dialog wollen wir allerdings nur dazu nutzen, uns einem Überblick über die Eigenschaften zu verschaffen. Die Einstellungen selbst nehmen wir später per VBA vor.

Auf diese Weise können Sie die Einstellungen, die vermutlich immer ähnlich aussehen werden, durch das Kopieren von einer Anwendung zur anderen oder von einem Formular zum anderen übertragen, ohne immer manuell die Einstellungen im *Eigenschaften*-Fenster anpassen zu müssen.

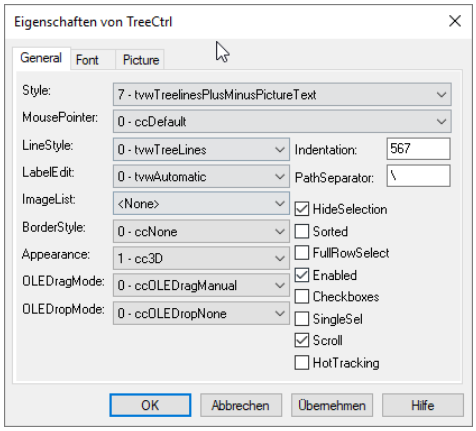


Abbildung 15.13: Eigenschaften des *TreeView*-Steuerelements

Hier sind die Eigenschaften und die VBA-Bezeichnungen samt den Werten:

» *Style*: Gibt den Stil an, also beispielsweise, ob Sie nur Text anzeigen wollen oder zusätzlich Icons, Plus/Minus-Zeichen oder Linien einblenden wollen. Wenn Sie nur Text anzeigen wollen, geben Sie den Wert *tvwTextOnly* an, für Text und Icon *tvwPictureText* und so weiter.

» *MousePointer*: Hiermit können Sie verschiedene Mauszeiger auswählen, die beim Überfahren des *TreeView*-Steuerelements den Standard-Mauszeiger ersetzen. Uns fällt aber kein Grund ein, den Standard-Mauszeiger zu ersetzen.

» *LineStyle*: Legt fest, wie die Linien angezeigt werden, wenn Sie für die Eigenschaft *Style* einen Eintrag ausgewählt haben, der die Linien einblendet. Hier gibt es die Werte *tvwTreeLines* oder *tvwRootLines*.

» *LabelEdit*: Gibt an, ob man den Text der Elemente editieren kann.

» *ImageList*: Legt das als *ImageList* zu verwendende Steuerelement fest.

» *BorderStyle*: Legt die Art des Rahmens fest.

» *Appearance*: Erscheinungsbild – flach oder dreidimensional.

» *OLEDragMode, OLEDropMode*: Einstellungen für Drag and Drop, siehe »Drag and Drop im TreeView-Steuerelement« ab Seite 420.

» *Indentation*: Einrückungstiefe der Ebenen

» *PathSeparator*: Zeichen, das beim Abfragen des Pfades zu einem Element mit der Eigenschaft *FullPath* verwendet werden soll.

» *HideSelection*: Gibt an, ob das markierte Element auch markiert sein soll, wenn das *TreeView*-Steuerelement den Fokus verliert.

» *FullRowSelect*: Markiert, wenn aktiviert, das aktive Element und den Rest der Zeile rechts davon.

» *Enabled*: Aktiviert oder deaktiviert das Element.

» *Checkboxes*: Gibt an, ob das *TreeView*-Steuerelement Kontrollkästchen vor jedem Element platzieren soll.

» *SingleSel*: Wenn aktiviert, werden die Unterelemente bei einem einfachen Mausklick statt bei einem doppelten Klick auf ein Element angezeigt.

» *HotTracking*: Wenn aktiviert, wird das aktuell mit der Maus überfahrene Element unterstrichen dargestellt.

15.2.6 Elemente zum TreeView-Steuerelement hinzufügen

Wenn Sie nun in die Formularansicht wechseln, sieht das *TreeView*-Steuerelement noch ziemlich trostlos aus (siehe Abbildung 15.14).

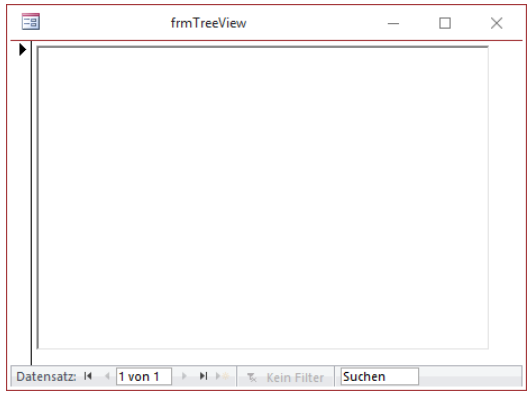


Abbildung 15.14: Leeres *TreeView*-Steuerelement

Daran wollen wir schnell etwas ändern. Leider bietet das *TreeView*-Steuerelement keine Möglichkeit, einfach eine Tabelle oder Abfrage als Datensatzherkunft zu verwenden. Stattdessen müssen wir die Elemente alle einzeln per VBA anlegen. Das ist aber auch kein Problem, denn wir können die Tabellen oder Abfragen ja per Schleife durchlaufen und das Anlegen in diesen Schleifen erledigen.

Die Add-Methode und ihre Parameter

Beim Anlegen der Elemente verwenden wir die *Add*-Methode der *Nodes*-Auflistung, die wir per IntelliSense schnell zur Prozedur *Form_Load* hinzufügen. Genau beim Laden des Formulars ist nämlich der richtige Zeitpunkt, um das *TreeView*-Steuerelement initial mit Daten zu füllen. Die *Add*-Methode erwartet die folgenden Parameter:

» *Relative*: Gibt den Index im Format Integer oder Schlüssel als Zeichenkette eines Knotens an, zu dem der neue Knoten angelegt werden soll – und zwar unter der mit dem folgenden Parameter angegebenen Beziehung.

» *Relationship*: Gibt die Beziehung an, unter der das neue Element mit dem unter *Relative* angegebenen Element angelegt werden soll. Dieser Parameter kann die Werte *tvwChild* (dem unter *Relative* angegebenen Element untergeordnet), *tvwFirst* (als erstes in der Ebene von *Relative*), *tvwLast* (als letztes in der Ebene von *Relative*), *tvwNext* (hinter *Relative*) oder *tvwPrevious* (vor *Relative*) annehmen.

» *Key*: Gibt die Zeichenfolge an, unter der das neue Element referenziert werden kann. Der Wert muss mit einem Buchstaben beginnen.

» *Text*: Text, der für das neue Element angezeigt werden soll.

» *Image*: *Index*- oder *Key*-Wert eines in der für dieses *TreeView*-Steuerelement angegebene *ImageList*-Steuerelement enthaltenen Images, das für dieses Element angezeigt werden soll.

» *SelectedImage*: *Index*- oder *Key*-Wert eines in der für dieses *TreeView*-Steuerelement angegebene *ImageList*-Steuerelement enthaltenen Images, das für dieses Element angezeigt werden soll, wenn es markiert ist.

Die hier genannten Eigenschaften können Sie auch noch nach dem Erstellen des Elements zum TreeView-Steuerelement anpassen.

Erstes Element anlegen

Um das erste Element anzulegen, das einfach nur die Beschriftung *Neues Element* trägt, fügen wir nach der Zuweisung des *TreeView*-Objekts eine neue Anweisung hinzu, welche mit der *Add*-Methode ein neues Element zur *Nodes*-Auflistung hinzufügt:

```
Private Sub Form_Load()
    Set objTreeView = Me!ctlTreeView.Object
    objTreeView.Nodes.Add , , "a1", "Neues Element"
End Sub
```

Die Eigenschaft *LineStyle* steht standardmäßig auf *tvwTreeLines*, sodass unser *TreeView*-Steuerelement nun beim Wechsel in die Formularansicht wie in Abbildung 15.15 aussieht.

Abbildung 15.15: *TreeView*-Steuerelement mit dem ersten Element

Nun wollen wir eine erste Eigenschaft für das Layout des *TreeView*-Steuerelements anpassen und die Linienart ändern – und zwar durch das Setzen der Eigenschaft *LineStyle* auf *tvwRootLines* (siehe Abbildung 15.16).

```
objTreeView.LineStyle = tvwRootLines
```

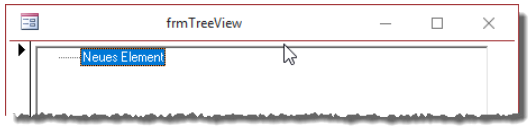


Abbildung 15.16: Einstellung *tvwRootLines* für die Eigenschaft *LineStyle*

Als nächstes wollen wir das erste Element mit ein paar Unterelementen anlegen. Der Code dafür sieht so aus:

```
Private Sub Form_Load()
    Set objTreeView = Me!ctlTreeView.Object
    objTreeView.LineStyle = tvwRootLines
    With objTreeView
        .Nodes.Clear
        .Nodes.Add , , "a1", "Erster Knoten"
        .Nodes.Add "a1", tvwChild, "a11", "Erster Unterknoten"
        .Nodes.Add "a11", tvwNext, "a12", "Zweiter Unterknoten"
        .Nodes.Add "a11", tvwFirst, "a10", "Nullter Unterknoten"
    End With
End Sub
```

Die erste *Add*-Methode fügt das Hauptelement ein:

```
.Nodes.Add , , "a1", "Erster Knoten"
```

Die zweite legt ein erstes Element unterhalb des Hauptelements an. Dazu legen wir mit den ersten beiden Parametern fest, dass das Element als Kind-Element (*tvwChild*) für das Element mit dem Key *a1* angelegt werden soll:

```
.Nodes.Add "a1", tvwChild, "a11", "Erster Unterknoten"
```

Die dritte Anweisung legt ein Element an, das auf der gleichen Ebene wie das zuletzt angelegte Element *a11* liegt, und zwar als nächstes Element, also direkt darunter:

```
.Nodes.Add "a11", tvwNext, "a12", "Zweiter Unterknoten"
```

Die vierte Anweisung fügt ein Element hinzu, das auf der gleichen Ebene wie das Element *a11* angelegt wird, das allerdings als erstes in die Reihe eingefügt wird:

```
.Nodes.Add "a11", tvwFirst, "a10", "Nullter Unterknoten"
```

Das Ergebnis sieht, nachdem Sie in die Formularansicht gewechselt sind und das oberste Element mit einem Klick auf das Plus-Zeichen aufgeklappt haben, wie in Abbildung 15.17 aus.

Abbildung 15.17: Element mit drei Unterelementen

Wenn Sie möchten, dass das Element der ersten Ebene immer direkt aufgeklappt dargestellt wird, legen Sie noch eine zusätzliche Eigenschaft fest. Dafür müsse Sie das Element allerdings zuerst referenzieren, und zwar mit einer Objektvariablen des Typs *MSCOMCTL.Node*:

```
Dim objNode As MSComctlLib.Node
Set objTreeView = Me!ctlTreeView.Object
With objTreeView
    .Nodes.Clear
    Set objNode = .Nodes.Add(, , "a1", "Erster Knoten")
    objNode.Expanded = True
    ...
End With
```

15.2.7 Stil für das TreeView-Steuerelement einstellen

Danach stellen wir den Stil für das *TreeView*-Steuerelement ein, und zwar durch die Zuweisung des gewünschten Wertes an die Eigenschaft *Style*.

Im folgenden Beispiel verwenden wir die Einstellung *tvwPictureText*, wodurch Sie zusätzlich zum Text noch ein Image zu jedem Element anzeigen können:

```
objTreeView.Style = tvwTreelinesPlusMinusPictureText
```

Wenn wir nun in die Formularansicht wechseln, ohne direkt Images zuzuweisen, sieht das Steuerelement wie in Abbildung 15.18 aus.

Abbildung 15.18: Einstellung mit Bildern, aber ohne Zuweisung von Images

15.2.8 Eigenschaften eines Elements per VBA einstellen

Sie können nicht nur die Eigenschaften für das *TreeView*-Steuerelement per VBA einstellen, sondern auch für jedes einzelne Element. Das sieht dann etwa wie folgt aus:

```
Set objNode = .Nodes.Add(, , "a1", "Erster Knoten")
With objNode
    .Expanded = True
    .Bold = True
    .ForeColor = 255
    .Text = "Neuer Text"
End With
```

Wir erweitern hier die untergeordneten Elemente, stellen die Schriftbreite auf Fett ein, legen als Schriftfarbe Rot fest und weisen dem Element einen neuen Text zu. Das sieht dann wie in Abbildung 15.19 aus.

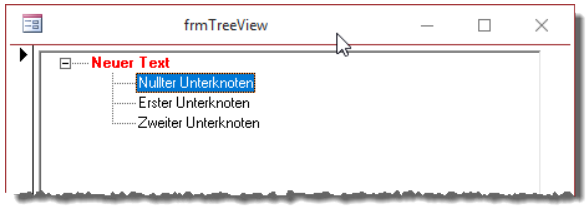


Abbildung 15.19: Node mit individuellen Eigenschaften

15.2.9 Images im TreeView

Nachdem wir weiter oben schon gezeigt haben, wie wir ein *ImageList*-Steuerelement mit einem oder mehreren Bildern füllen, wollen wir uns nun auch noch ansehen, wie wir die Bilder in das *TreeView*-Steuerelement bringen. Dazu fügen Sie das *ImageList*-Steuerelement wie oben unter

»Das ImageList-Steuerelement« ab Seite 379 beschrieben zum Formular hinzu. Dann erweitern wir den Code unserer *Form_Load*-Prozedur um ein paar Zeilen, zunächst um eine weitere Deklaration für das *ImageList*-Element im Kopf des Klassenmoduls:

```
Dim objTreeView As MSComctlLib.TreeView
Dim objImageList As MSComctlLib.ImageList
```

Die Prozedur erweitern wir dann um die Zuweisung des *ImageList*-Steuerelements zur Variablen *objImageList*:

```
Private Sub Form_Load()
    Dim objNode As MSComctlLib.Node
    Set objTreeView = Me!ctlTreeView.Object
    Set objImageList = Me!ctlImageList.Object
```

Dann folgt der Aufruf der Prozedur, um dem *ImageList*-Steuerelement das Bild *add* aus der Tabelle *MSysResources* hinzuzufügen:

```
    ImageHinzufuegen objImageList, "add"
    objTreeView.LineStyle = tvwRootLines
    objTreeView.Style = tvwTreelinesPlusMinusPictureText
```

Der Eigenschaft *ImageList* der *TreeView*-Objektvariablen *objTreeView* weisen wir dann das *ImageList*-Steuerelement aus *objImageList* zu:

```
    With objTreeView
        .ImageList = objImageList
        .Nodes.Clear
```

Schließlich legen wir mit dem fünften Parameter der *Add*-Methode der *Nodes*-Auflistung jeweils das unter dem Schlüssel *add* abgelegte Image zu:

```
        Set objNode = .Nodes.Add(, , "a1", "Erster Knoten", "add")
        ...
        .Nodes.Add "a1", tvwChild, "a11", "Erster Unterknoten", "add"
        .Nodes.Add "a11", tvwNext, "a12", "Zweiter Unterknoten", "add"
        .Nodes.Add "a11", tvwFirst, "a10", "Nullter Unterknoten", "add"
    End With
End Sub
```

Das Ergebnis sieht schließlich wie in Abbildung 15.20 aus. Verschiedene Images für verschiedene Elemente legen wir später fest, wenn wir Daten aus den Tabellen der Datenbank zum *TreeView*-Steuerelement hinzufügen.

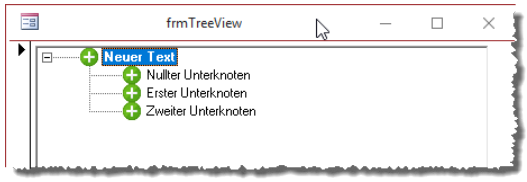


Abbildung 15.20: *TreeView*-Einträge mit Image

15.2.10 TreeView-Steuerelement mit Daten aus verknüpften Tabellen füllen

Beim Anzeigen von Daten aus verknüpften Tabellen durchlaufen wir die auf diesen Tabellen basierenden Recordsets in zwei verschachtelten *Do While*-Schleifen. Bei zwei Tabellen bedeutet dies, dass in einer äußeren Schleife die Datensätze der Haupttabelle und in einer inneren Schleife die Datensätze der untergeordneten Tabelle durchlaufen werden.

Das Füllen des *TreeView*-Steuerelements fügen wir der Ereignisprozedur hinzu, die beim Laden des Formulars ausgelöst wird. Als Basis für das folgende Beispiel dienen die Tabellen *tblKunden* und *tblBestellungen* sowie ein Formular mit einem *TreeView*-Steuerelement namens *ctlTreeView*. Damit Sie das *TreeView*-Steuerelement jederzeit mit dem Ausdruck *objTreeView* referenzieren können, fügen Sie wieder folgende Zeilen in den Kopf des Formular-Moduls ein:

```
Dim objTreeView As MSComctlLib.TreeView
```

Die Prozedur, die durch das Ereignis *Beim Laden* ausgelöst wird, füllen wir wie folgt:

```
Private Sub Form_Load()
    Dim db As DAO.Database
    Dim rstKunden As DAO.Recordset2
    Dim rstBestellungen As DAO.Recordset2
    Dim objNode As MSComctlLib.Node
    Set db = CurrentDb
    Set objTreeView = Me!ctlTreeView.Object
    With objTreeView
        .Nodes.Clear
        .Style = tvwTreelinesPlusMinusText
        .LineStyle = tvwRootLines
    End With
```

Nach dem Zuweisen des *TreeView*-Objekts zur Variablen *objTreeView* und dem Einstellen der grundlegenden Eigenschaften erstellen wir das Recordset, das die Daten für die erste Ebene des

TreeView-Steuerelements enthalten soll. Wir benötigen nur die beiden Felder *KundeID* (für den Key) und *Firma* als Anzeigetext:

```
Set rstKunden = db.OpenRecordset("SELECT KundeID, Firma FROM tblKunden", dbOpenDynaset)
```

Dieses durchlaufen wir dann in einer *Do While*-Schleife. Dabei legen wir zuerst jeweils für den aktuellen Datensatz ein neues Element im *TreeView*-Steuerelement an. Da alle in der Reihenfolge der Datensätze im Recordset einfach in der ersten Ebene angelegt werden sollen, benötigen wir die ersten beiden Parameter nicht. Der dritte Parameter soll den Schlüssel aufnehmen, den wir später benötigen, um dem Element die Bestellungen unterzuordnen. Diesem stellen wir als Präfix den Buchstaben *k* (für *Kunde*) voran. Dem vierten Parameter weisen wir den anzuzeigenden Text zu, in diesem Fall den Inhalt des Feldes *Firma* des Recordsets:

```
Do While Not rstKunden.EOF
    Set objNode = objTreeView.Nodes.Add(, , "k" & rstKunden!KundeID, _
        rstKunden!Firma)
```

Innerhalb der Schleife öffnen wir für jeden Datensatz des Recordsets *rstKunden* ein Recordset mit den Bestelldatensätzen zu diesem Kunden. Dieses Recordset heißt *rstBestellungen*. Es soll die beiden Felder *BestellungID* und *Bestelldatum* der Tabelle *tblBestellungen* aufnehmen, für die das Feld *KundeID* dem Wert des gleichnamigen Feldes des aktuell durchlaufenen Datensatzes des Recordsets *rstKunden* entspricht:

```
Set rstBestellungen = db.OpenRecordset("SELECT BestellungID, Bestelldatum " _
    & "FROM tblBestellungen WHERE KundeID = " & rstKunden!KundeID, dbOpenDynaset)
```

Auch die Datensätze dieses Recordsets durchlaufen wir in einer *Do While*-Schleife. In dieser fügen wir dem *TreeView*-Steuerelement jeweils ein neues Element hinzu. Diesmal finden die ersten beiden Parameter jedoch Verwendung, denn wir wollen das neue Element ja dem Kunden-Element aus der ersten Ebene unterordnen, zu dem die Bestellung gehört. Also geben wir als ersten Parameter den aus dem Buchstaben *k* und dem Primärschlüsselwert des Kundendatensatzes bestehenden Wert als Referenz hinzu. Damit das neue Element als Unterelement dieses Elements angelegt wird, legen wir für den zweiten Parameter den Wert *tvwChild* fest. Der dritte Parameter nimmt wieder den eindeutigen Schlüssel für das anzulegende Objekt fest, der vierte den anzuzeigenden Text. Dieser setzt sich zusammen aus dem Text *Bestellung*, der Bestellnummer und dem in Klammern eingefassten Bestelldatum, also beispielsweise *Bestellung 123 (1.2.2019)*:

```
Do While Not rstBestellungen.EOF
    objTreeView.Nodes.Add "k" & rstKunden!KundeID, tvwChild, "b" &
        rstBestellungen!BestellungID, "Bestellung " _
            & rstBestellungen!BestellungID _
            & " (" & rstBestellungen!Bestelldatum & ")"
    rstBestellungen.MoveNext
Loop
```

Nachdem alle dem aktuellen Kunden zugeordneten Bestellungen durchlaufen und so die entsprechenden Einträge zum *TreeView*-Steuerelement angelegt worden sind, können wir zum nächsten Element des Recordsets *rstKunden* übergehen und dieses und die folgenden abarbeiten:

```
            rstKunden.MoveNext
        Loop
        rstKunden.Close
        rstBestellungen.Close
        Set rstKunden = Nothing
        Set rstBestellungen = Nothing
        Set db = Nothing
    End Sub
```

Das Ergebnis sieht nach dem Aufklappen eines der Einträge wie in Abbildung 15.21 aus.

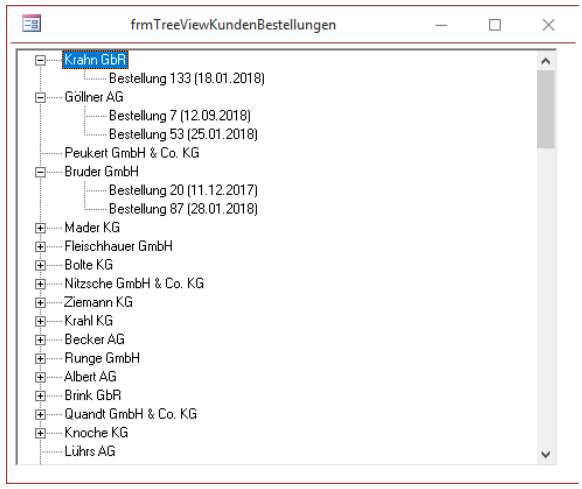


Abbildung 15.21: Kunden und Bestellungen im *TreeView*-Steuerelement

15.2.11 TreeView-Steuerelement mit Daten aus reflexiv verknüpften Tabellen füllen

Um die Daten einer über eine Hilfstabelle mit sich selbst verknüpften Tabelle im *TreeView*-Steuerelement darzustellen, haben wir die Tabellen *tblMitarbeiter* und *tblVorgesetzterMitarbeiter* wie in Abbildung 15.22 erstellt.

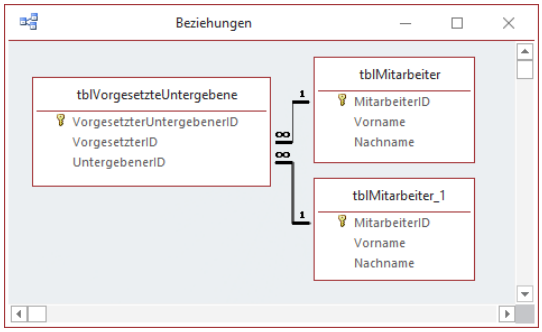


Abbildung 15.22: Reflexible Beziehung

Die Tabelle *tblMitarbeiter* enthält die Mitarbeiterdaten, in diesem Fall lediglich *MitarbeiterID*, *Vorname* und *Nachname*. Die Tabelle *tblVorgesetzteUntergebene* verknüpft jeweils einen Eintrag der Tabelle *tblMitarbeiter* als Vorgesetzten mit einem Eintrag dieser Tabelle als Untergebenen.

In der Tabelle *tblVorgesetzteUntergebene* legen wir für die beiden Fremdschlüsselfelder jeweils ein Nachschlagefeld an, das als Datensatzherkunft die folgende Abfrage verwendet:

```
SELECT [tblMitarbeiter].[MitarbeiterID], [tblMitarbeiter].[Nachname] & ", " & [tblMit-
arbeiter].[Vorname] AS Mitarbeiter FROM tblMitarbeiter;
```

Damit können wir dann in der Tabelle die Mitarbeiterhierarchie wie in Abbildung 15.23 aufbauen.

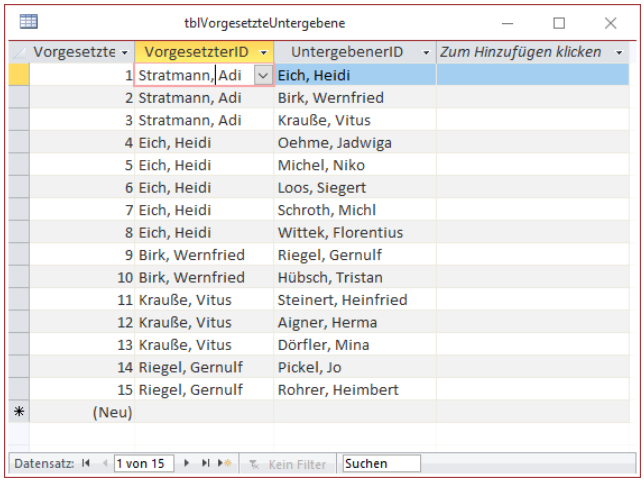


Abbildung 15.23: Zuordnung von Vorgesetzten und Untergebenen in der Verknüpfungstabelle *tblVorgesetzteUntergebene*

Nun wollen wir diese Daten nutzen, um ein *TreeView*-Steuerelement damit zu füllen.

Rekursive Funktion

Zum Füllen des *TreeView*-Steuerelements mit den Daten aus den reflexiv verknüpften Tabellen ist eine rekursiv definierte Funktion erforderlich. Diese Funktion muss sich selbst immer wieder aufrufen und dabei die untergeordneten Datensätze ermitteln, bis keine untergeordneten Datensätze mehr vorhanden sind.

Den ersten Aufruf dieser Funktion fügen wir allerdings zur Ereignisprozedur *Form_Load* hinzu. Diese referenziert wieder das *TreeView*-Steuerelement und stellt einige grundlegende Eigenschaften dieses Steuerelements ein. Dann ruft es die Funktion *TreeViewRekursiv* auf und übergibt dieser den Wert *0* als Parameter:

```
Private Sub Form_Load()
    Set objTreeView = Me!ctlTreeView.Object
    With objTreeView
        .Nodes.Clear
        .Style = tvwTreelinesPlusMinusText
        .LineStyle = tvwRootLines
    End With
    TreeViewRekursivFuellen 0
End Sub
```

Bevor wir uns diese Prozedur ansehen, wollen wir noch eine Abfrage erstellen, mit der wir die Tabellen *tblMitarbeiter* und *tblVorgesetzteUntergebene* zusammenführen. Diese Abfrage sieht im Entwurf wie in Abbildung 15.24 aus. Sie verwendet die beiden genannten Tabellen als Datenquelle und erhält den Namen *qryVorgesetzte*.

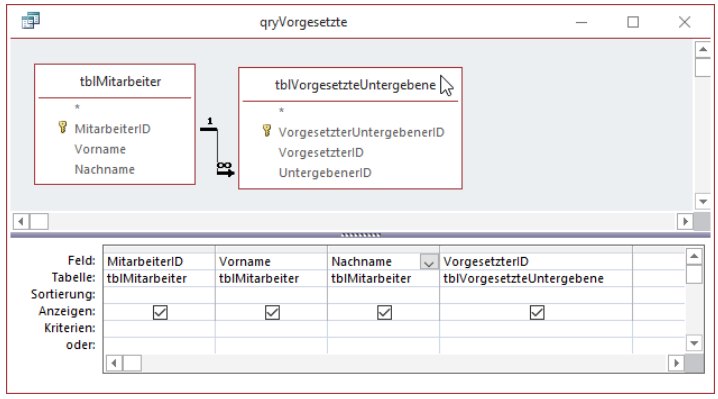


Abbildung 15.24: Abfrage zum Zusammenführen von Vorgesetzten und Untergebenen

Die Beziehung zwischen den beiden Tabellen passen wir noch an. Wenn wir das nicht tun, wird diese Abfrage alle Datensätze der beiden Tabellen bringen, bei denen es in jeder Tabelle einen passenden Datensatz gibt.

In diesem Fall fällt aber genau der Datensatz der Tabelle *tblMitarbeiter* heraus, dem kein Vorgesetzter über die Tabelle *tblVorgesetzteUntergebene* zugeordnet ist. Definitionsgemäß sind das aber genau die Mitglieder der obersten Ebene der Hierarchie im Unternehmen. Und in diesem Fall könnten wir auch die Hierarchie nicht im *TreeView* abbilden.

Aber es gibt eine Möglichkeit, auch die Kombinationen aus den Datensätzen der Tabellen *tblMitarbeiter* und *tblVorgesetzteUntergebene* anzuzeigen, für die kein entsprechender Datensatz in der Tabelle *tblVorgesetzteUntergebene* vorliegt. Dazu klicken Sie doppelt auf den Beziehungspfeil zwischen den beiden Tabellen. Wählen Sie dort die Option aus, bei der *Beinhaltet ALLE Datensätze aus 'tblMitarbeiter' ...* steht (siehe Abbildung 15.25).

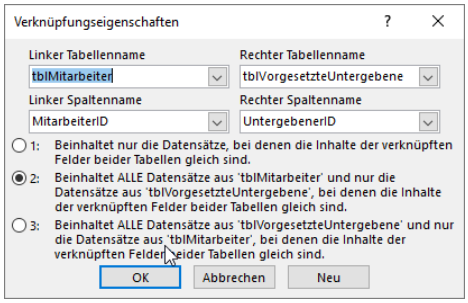


Abbildung 15.25: Anpassen der Verknüpfungseigenschaften

Wenn Sie nun in die Datenblattansicht der Abfrage wechseln, finden Sie auch den Mitarbeiter vor, der keinen Vorgesetzten hat – offensichtlich der Chef des Unternehmens (siehe Abbildung 15.26).

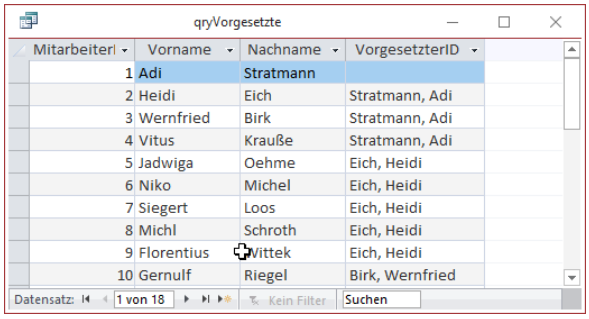


Abbildung 15.26: Mitarbeiter mit und ohne Vorgesetzten

Damit können wir uns nun der rekursiven Funktion zuwenden. Dies erwartet als Parameter die Nummer des Mitarbeiters, dessen Untergebene sie verarbeiten soll. Beim ersten Aufruf von der Prozedur *Form_Load* aus ist das beispielsweise der Wert 0:

```
Private Sub TreeViewRekursivFuellen(lngMitarbeiterID As Long)
    Dim db As DAO.Database
    Dim rst As DAO.Recordset2
    Dim objNode As MSComctlLib.Node
    Set db = CurrentDb
```

Nach dem Deklarationsteil prüft die Funktion in einer *If...Then*-Bedingung den Wert des Parameters *lngMitarbeiterID* ab. Hat dieser den Wert *0*, füllt die Funktion die *Recordset*-Variable *rst* mit den Daten der Abfrage *qryVorgesetzte*, für die das Feld *VorgesetzterID* leer ist – also für alle Mitarbeiter der obersten Ebene:

```
    If lngMitarbeiterID = 0 Then
        Set rst = db.OpenRecordset( _
            "SELECT * FROM qryVorgesetzte WHERE VorgesetzterID IS NULL")
```

Im *Else*-Teil der Abfrage, der frühestens bei ersten Aufruf der rekursiven Funktion über sich selbst angesteuert wird (dort ist *lngMitarbeiterID* dann nicht mehr *0*), stellen wir für das Recordset rst die Daten der Abfrage *qryVorgesetzte* zusammen, deren Wert im Feld *VorgesetzterID* dem übergeordneten Mitarbeiter entspricht:

```
    Else
        Set rst = db.OpenRecordset("SELECT * FROM qryVorgesetzte " _
            & "WHERE VorgesetzterID = " & lngMitarbeiterID)
    End If
```

Danach durchläuft die Funktion in einer *Do While*-Schleife alle Datensätze des Recordsets *rst*. Darin prüft sie wiederum, ob *lngMitarbeiterID* den Wert *0* hat (für alle Elemente der obersten Ebene).

In diesem Fall fügt sie ein Element zum *TreeView*-Steuerelement hinzu, das keinem anderen Element untergeordnet ist und für das wir die ersten beiden Parameter leer lassen. Den dritten Parameter füllen wir mit dem Schlüssel, der sich aus dem Buchstaben *m* und dem Primärschlüsselwert der Tabelle *tblMitarbeiter* zusammensetzt. Als Text fügen wir eine aus dem Vornamen, einem Leerzeichen und dem Nachnamen zusammengesetzte Zeichenkette hinzu:

```
    Do While Not rst.EOF
        If lngMitarbeiterID = 0 Then
            Set objNode = objTreeView.Nodes.Add(, , "m" & rst!MitarbeiterID, _
                rst!Vorname & " " & rst!Nachname)
```

Anderenfalls handelt es sich um ein untergeordnetes Element. Dann nehmen wir noch die ersten beiden Parameter hinzu. Den ersten Parameter füllen wir mit dem Schlüssel, der dem übergeordneten Mitarbeiter entspricht und der sich aus dem Buchstaben m und dem mit dem Parameter *lngMitarbeiter* gelieferten Primärschlüsselwert der Tabelle *tblMitarbeiter* zusammensetzt. Zu diesem Element soll das neue Element für den aktuellen Mitarbeiter als Kind verhalten, also erhält der zweite Parameter den Wert *tvwChild*. Die übrigen beiden Parameter füllen wir wie im zuvor beschriebenen Fall:

```
        Else
            Set objNode = objTreeView.Nodes.Add("m" & lngMitarbeiterID, tvwChild, _
                "m" & rst!MitarbeiterID, rst!Vorname & " " & rst!Nachname)
        End If
```

Schließlich rufen die Funktion *TreeViewRekursivfuellen* mit dem Primärschlüsselwert für den aktuellen Mitarbeiter als Parameter selbst auf und verarbeitet so die dem aktuellen Mitarbeiter untergeordneten Mitarbeiter:

```
        TreeViewRekursivFuellen rst!MitarbeiterID
```

Wenn der Fokus wieder zu diesem Aufruf der Funktion zurückgekehrt ist, bearbeitet diese den nächsten Datensatz und kehrt nach dem Durchlaufen aller Datensätze wieder zum übergeordneten Aufruf zurück:

```
            rst.MoveNext
        Loop
        Set objNode = Nothing
        rst.Close
        Set rst = Nothing
        Set db = Nothing
    End Sub
```

Das Ergebnis sieht dann im Ribbon wie in Abbildung 15.27 aus.

15.2.12 Viele Daten im TreeView-Steuerelement

Das Füllen des *TreeView*-Steuerelements dauert mit wachsender Datenmenge natürlich entsprechend länger. Für verschiedene Szenarien gibt es allerdings auch noch Möglichkeiten, das Füllen des TreeViews zu beschleunigen. Das ist zum Beispiel der Fall, wenn die Verschachtelungstiefe bekannt ist – also etwa wie im ersten Beispiel, wo wir nur die Ebenen mit den Kunden und den Bestellungen füllen mussten.

Der Performance-Nachteil bei der oben verwendeten Methode lag darin, dass wir für jedes der untergeordneten Elemente ein neues Recordset geöffnet haben.

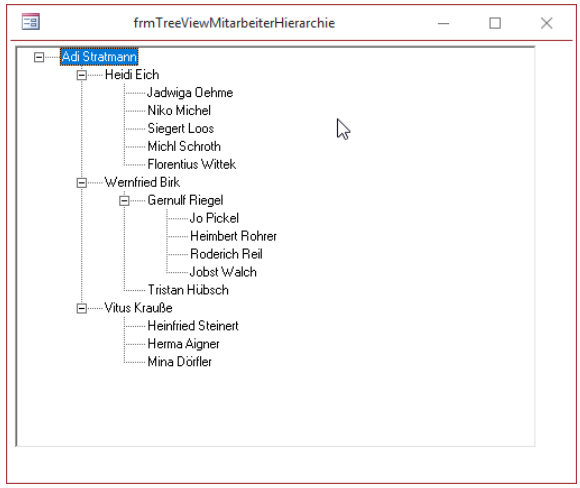


Abbildung 15.27: Mitarbeiterhierarchie im *TreeView*-Steuerelement

Das können wir auch besser lösen, wenn wir alle benötigten Daten direkt in einer einzigen Abfrage zusammenfassen. In diesem Fall brauchen wir eine Abfrage, welche die beiden beteiligten und verknüpften Tabellen enthält. Diese sieht in unserem Beispiel wie in Abbildung 15.28 aus. Sie enthält die beiden Abfragen *tblKunden* und *tblBestellungen*, die über die Felder namens *KundeID* miteinander verknüpft sind.

Aus diesen Tabellen ziehen wir alle Felder in das Entwurfsraster, die wir erstens für die Anzeige der gewünschten Texte im TreeView-Steuerelement benötigen (das Feld *Firma* der Tabelle *tblKunden* und das Feld *Bestelldatum* der Tabelle *tblBestellungen*) und zweitens für die eindeutige Benennung der Elemente – also die Primärschlüsselfelder der beiden Tabellen, *KundeID* und *BestellungID*. Wichtig ist außerdem, dass wir die Datensätze der Abfrage aufsteigend nach den beiden Primärschlüsselfeldern sortieren.

Wenn wir in die Datenblattansicht dieser Abfrage wechseln, erhalten wir das Ergebnis aus Abbildung 15.29. Die folgende Prozedur legt dann für jeden Datensatz immer entweder ein Element an oder zwei.

Wenn der Kunde das erste Mal auftaucht, legt sie für diesen ein neues Element an und direkt danach ein Element für die Bestellung, die im ersten Datensatz zu diesem Kunden aufgeführt ist. Wenn der Kunde das zweite Mal hintereinander in einem Datensatz auftaucht, legt die Prozedur kein neues Kundenelement an, sondern fügt nur das Bestellung-Element dieses Datensatzes zu dem zuletzt angelegten Kunden hinzu.

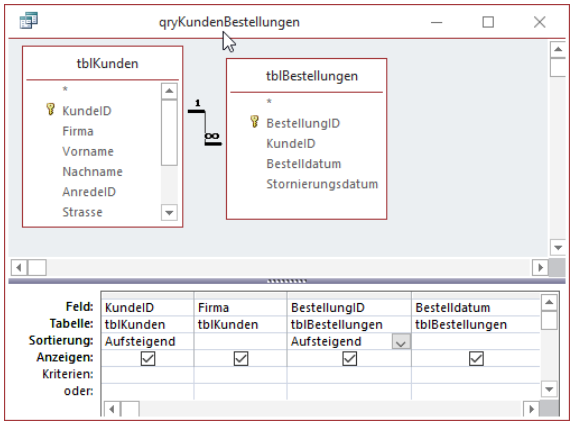


Abbildung 15.28: Zusammenfassende Datenquelle für schnelles Füllen des *TreeView*-Steuerelements

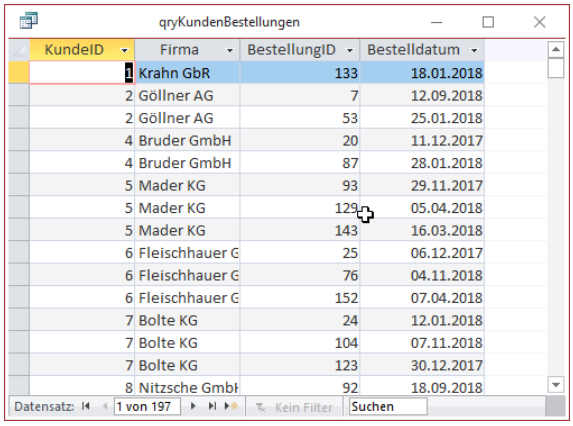


Abbildung 15.29: Ergebnis der Abfrage *qryKundenBestellungen* mit den Daten für den Kunden und die Bestellung

Im Kopf des Klassenmoduls des für dieses Beispiel neu eingerichteten Klassenformulars *frmTreeViewKundenBestellungen_Schnell* deklarieren wir wieder die Objektvariable für das *TreeView*-Steuerelement:

```
Dim objTreeView As MSComctlLib.TreeView
```

Die Ereignisprozedur *Form_Load* wird diesmal jedoch etwas anders aufgebaut. Der Deklarationsteil und die Anweisungen zum Zuweisen und Einrichten des *TreeView*-Steuerelements ähneln noch weitgehend denen des ersten Formulars zur Ausgabe der Kunden und Bestellungen im *TreeView*-Steuerelement. Wir kommen jedoch mit einem einzigen Recordset aus, das die Daten der Abfrage *qryKundenBestellungen* verwendet:

```
Private Sub Form_Load()
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Dim objNode As MSComctlLib.Node
    Dim lngVorherigerKunde As Long
    Set db = CurrentDb
    Set rst = db.OpenRecordset("qryKundenBestellungen")
    Set objTreeView = Me!ctlTreeView.Object
    With objTreeView
        .Nodes.Clear
        .Style = tvwTreelinesPlusMinusText
        .LineStyle = tvwRootLines
    End With
    objTreeView.Nodes.Clear
```

Danach durchlaufen wir die einzige *Do While*-Schleife dieser Prozedur. Dabei prüfen wir in einer *If...Then*-Bedingung, ob der Primärschlüsselwert des Kunden aus dem aktuellen Datensatz mit dem in der Variablen *lngVorherigerKunde* gespeicherten Wert übereinstimmt. Das ist beim ersten Durchlauf nicht der Fall, denn *lngVorherigerKunde* ist ja noch leer. Also wird der Teil in der *If...Then*-Bedingung ausgeführt. Hier wird dann das erste Element zum *TreeView*-Steuerelement hinzugefügt, und zwar auf der ersten Ebene für den aktuellen Kunden. Dafür verwenden wir dann einen Schlüssel als dritten Parameter, der aus dem Buchstaben *k* für *Kunde* und dem Primärschlüsselwert zusammengesetzt wird sowie den Wert des Feldes *Firma* als Beschriftung:

```
    Do While Not rst.EOF
        If Not rst!KundeID = lngVorherigerKunde Then
            Set objNode = objTreeView.Nodes.Add(, , "k" & rst!KundeID, rst!Firma)
        End If
```

Danach wird ein Element unterhalb des *Kunden*-Elements des aktuellen Datensatzes angelegt. Deshalb geben wir als Referenz im ersten Parameter den Schlüssel des übergeordneten Kunden an, den wir aus dem Buchstaben *k* und dem Primärschlüsselwert für den aktuellen Kunden anlegen. Das neue Element soll als Kind-Element angelegt werden, also geben wir für den zweiten Parameter den Wert *tvwChild* an. Für den Schlüssel des neu anzulegenden Elements geben wir den Buchstaben *b* für *Bestellung* und den Primärschlüsselwert der Bestellung an, den wir ja im gleichen Datensatz der Abfrage finden. Außerdem tragen wir als Text des neuen Elements einen Ausdruck ein, der aus dem Text *Bestellung*, der Bestell-ID sowie dem Bestelldatum in Klammern besteht:

```
        Set objNode = objTreeView.Nodes.Add("k" & rst!KundeID, tvwChild, _
            "b" & rst!BestellungID, "Bestellung " & rst!BestellungID _
            & " (" & rst!Bestelldatum)
```

Damit wir beim nächsten Durchlauf erkennen können, ob wir noch einen weiteren Datensatz für den gleichen Kunden durchlaufen und somit keinen neuen Kunden, sondern nur eine Bestellung als neues Element zum *TreeView*-Steuerelement hinzufügen müssen, stellen wir den Wert der Variablen *lngVorherigerKunde* auf den Wert des Feldes *KundeID* des Recordsets ein. Danach wechseln wir dann auch mit *rst.MoveNext* zum nächsten Datensatz und beginnen die Schleife von vorn:

```
            lngVorherigerKunde = rst!KundeID
            rst.MoveNext
        Loop
        Set objNode = Nothing
        rst.Close
        Set rst = Nothing
        Set db = Nothing
    End Sub
```

15.2.13 Elemente erst bei Bedarf anlegen

Wenn wir bei dem vorherigen Beispiel noch eine Ebene weitergehen, also auch noch die Bestelldetails unterhalb der Bestellung einfügen würden, würde sich das nochmal negativ auf die Performance beim Füllen des *TreeView*-Steuerelements auswirken. Noch gravierender wird dies, wenn das *TreeView*-Steuerelement auch noch Images anzeigen soll. Wir schauen uns das am Beispiel des Formulars *frmTreeViewKundenBestellungenPositionen* an, das am Ende wie in Abbildung 15.30 aussehen soll.

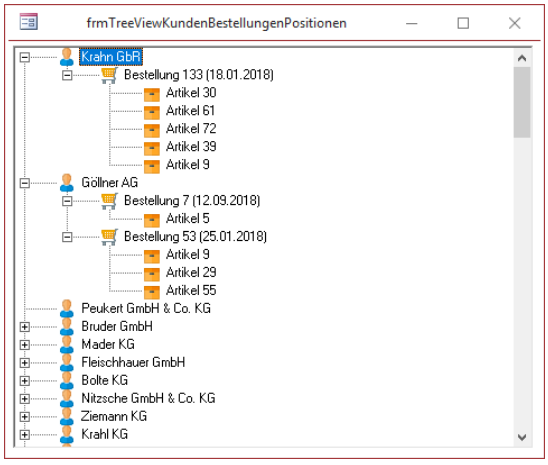


Abbildung 15.30: *TreeView*-Steuerelement mit drei Ebenen und Images

Damit wir die Artikel der einzelnen Positionen noch zu den Bestellungen hinzufügen können, benötigen wir eine weitere Abfrage, die uns die Datensätze der Tabellen *tblBestellpositionen* und *tblArtikel* zusammenführt. Diese sieht wie in Abbildung 15.31 aus.

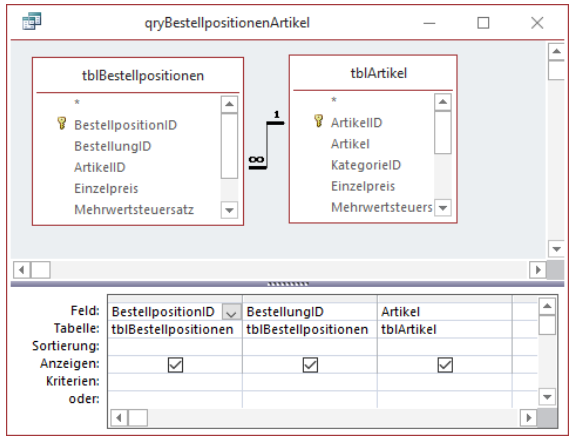


Abbildung 15.31: Abfrage mit den Tabellen *tblBestellpositionen* und *tblArtikel*

Da wir nun auch noch drei verschiedene Arten von Images für die drei Ebenen anzeigen wollen, müssen wir diese zunächst über den Dialog *Bild einfügen* hinzufügen, der im Formularentwurf angezeigt wird (siehe weiter oben unter »Bilder in Formularen« ab Seite 367).

Wir haben drei Bilder namens *shopping_cart*, *box_closed_16* und *user_16* hinzugefügt. Im Klassenmodul des Formulars deklarieren wir wieder die beiden Objektvariablen für das *TreeView*- und das *ImageList*-Steuerelement:

```
Dim objTreeView As MSComctlLib.TreeView
Dim objImageList As MSComctlLib.ImageList
```

Die Prozedur *Form_Load* sieht wie folgt aus. Neu hinzugekommen ist das Füllen des *ImageList*-Steuerelements mit den drei Images sowie das dritte Recordset zum Füllen der dritten Ebene im *TreeView*-Steuerelement:

```
Private Sub Form_Load()
    Dim db As DAO.Database
    Dim rstKunden As DAO.Recordset2
    Dim rstBestellungen As DAO.Recordset2
    Dim rstBestellpositionen As DAO.Recordset2
    Dim objNode As MSComctlLib.Node
    Set objImageList = Me!ctlImageList.Object
    ImageHinzufuegen objImageList, "shopping_cart"
    ImageHinzufuegen objImageList, "box_closed_16"
```

```
    ImageHinzufuegen objImageList, "user_16"
    Set db = CurrentDb
    Set objTreeView = Me!ctlTreeView.Object
    Set rstKunden = db.OpenRecordset("SELECT KundeID, Firma FROM tblKunden", dbOpenDynaset)
    With objTreeView
        .ImageList = objImageList
        .Nodes.Clear
        .Style = tvwTreelinesPlusMinusPictureText
        .LineStyle = tvwRootLines
    End With
    Do While Not rstKunden.EOF
        Set objNode = objTreeView.Nodes.Add(, , "k" & rstKunden!KundeID, _
            rstKunden!Firma, "user_16")
        Set rstBestellungen = db.OpenRecordset("SELECT BestellungID, Bestelldatum FROM " _
            & "tblBestellungen WHERE KundeID = " & rstKunden!KundeID, dbOpenDynaset)
        Do While Not rstBestellungen.EOF
            objTreeView.Nodes.Add "k" & rstKunden!KundeID, tvwChild, "b" _
                & rstBestellungen!BestellungID, "Bestellung " & rstBestellungen!BestellungID _
                & " (" & rstBestellungen!Bestelldatum & ")", "shopping_cart"
            Set rstBestellpositionen = db.OpenRecordset("SELECT * FROM " _
                & "qryBestellpositionenArtikel WHERE BestellungID = " _
                & rstBestellungen!BestellungID, dbOpenDynaset)
                Do While Not rstBestellpositionen.EOF
                    objTreeView.Nodes.Add "b" & rstBestellungen!BestellungID, tvwChild, _
                        "p" & rstBestellpositionen!BestellpositionID, _
                        rstBestellpositionen!Artikel, "box_closed_16"
                    rstBestellpositionen.MoveNext
                Loop
            rstBestellungen.MoveNext
        Loop
        rstKunden.MoveNext
    Loop
    rstKunden.Close
    rstBestellungen.Close
    Set rstKunden = Nothing
    Set rstBestellungen = Nothing
    Set db = Nothing
End Sub
```

Das nur zum Aufwärmen. Nun wollen wir ja einen Weg finden, wie wir Daten nachladen können – etwa die der dritten Ebene. Diese wollen wir nur nachladen, wenn der Benutzer den Eintrag

16 Kontextmenüs

Kontextmenüs sind Menüs, die beim Rechtsklick mit der Maus auf ein bestimmtes Element angezeigt werden. Das beste Beispiel sind die Befehle, die Sie beim Anklicken einer Datei mit der rechten Maustaste im Windows Explorer erhalten. Auch unter Access gibt es für die meisten Elemente Kontextmenüs. Diese wollen wir gar nicht ausschweifend beschreiben, denn Sie kennen diese sicher bereits.

Interessant ist die Tatsache, dass Sie Kontextmenüs auch selbst definieren und auch bestehende Kontextmenüs erweitern können. In alten Versionen von Access bis zur Version 2003 ging das über die Benutzeroberfläche – dort konnten Sie sich mit ein paar Mausklicks neue Kontextmenüs zusammenklicken und diese dann in verschiedenen Kontexten zur Anzeige freigeben. Ab Access 2007 sind die Elemente der Benutzeroberfläche zum Erstellen von Kontextmenüs jedoch entfallen und Sie müssen Kontextmenüs komplett per VBA programmieren.

Wie das geht, erläutern wir in diesem Kapitel. Grundsätzlich sind Kontextmenüs zwar kein Element von Formularen, jedoch lässt sich damit die Bedienung von Formularen an vielen Stellen vereinfachen. Nach dem theoretischen Teil zeigen wir Ihnen daher auch noch ein Praxisbeispiel für den Einsatz eines Kontextmenüs.

16.1 Die CommandBars-Auflistung

Wie für fast alle Elemente gibt es unter Access auch eine Auflistung für die Menüleisten. Auch hier ein kurzer geschichtlicher Hintergrund: Bis Access 2003 gab es nicht nur Elemente in der Benutzeroberfläche zum Zusammenklicken von Kontextmenüs, sondern bis dahin gab es auch noch die sogenannten Menüleisten und Symbolleisten. Diese wurden dann mit Access 2007 durch das Ribbon abgelöst. Menüleisten, Symbolleisten und Kontextmenüs teilten sich das gleiche Objektmodell und die Instanzen ihrer Hauptklasse namens *CommandBar* wurde in einer Auflistung namens *CommandBars* bereitgestellt.

16.1.1 Verweis auf die Office-Bibliothek

Um auf die Elemente wie die *CommandBars*-Auflistung, die *CommandBar*-Klasse und die darin enthaltenen Steuerelemente zuzugreifen, benötigen Sie einen Verweis auf die Bibliothek namens *Microsoft Office x.0 Object Library*. Die Objekte für die Programmierung der Kontextmenüs sind nämlich kein Bestandteil der bereits standardmäßig in einem VBA-Projekt eingebundenen Elemente, sondern stehen in allen Office-Anwendungen zur Verfügung und werden daher in einer allgemeinen Bibliothek bereitgestellt.

Diese fügen Sie zum aktuellen VBA-Projekt hinzu, indem Sie dort den Menüeintrag *Extras|Verweise* betätigen. Der nun erscheinende Dialog erlaubt normalerweise die Auswahl des Eintrags *Microsoft Office x.0 Object Library*. Sollte dieser Eintrag bei Ihnen nicht erscheinen, klicken Sie in dem Dialog auf die Schaltfläche *Durchsuchen...* (siehe Abbildung 16.1).

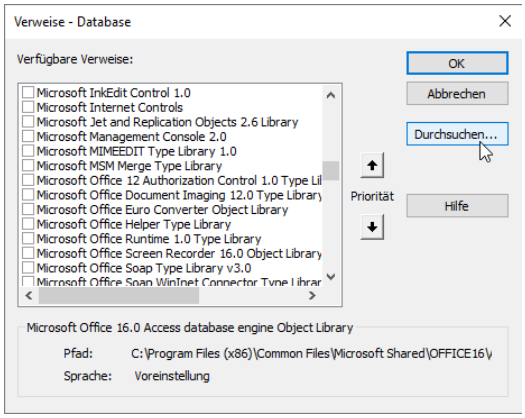


Abbildung 16.1: Der Eintrag *Microsoft Office x.0 Object Library* fehlt in dieser Installation.

Unter Office 16 in der 32bit-Version finden Sie diese Datei beispielsweise in folgendem Ordner:

```
C:\Program Files (x86)\Microsoft Office\root\vfs\ProgramFilesCommonX86\Microsoft
Shared\OFFICE16
```

Klicken Sie die Datei doppelt an, um sie zur Liste der Verweise hinzuzufügen. Anschließend finden Sie den Eintrag ganz oben im *Verweise*-Dialog vor (siehe Abbildung 16.2).

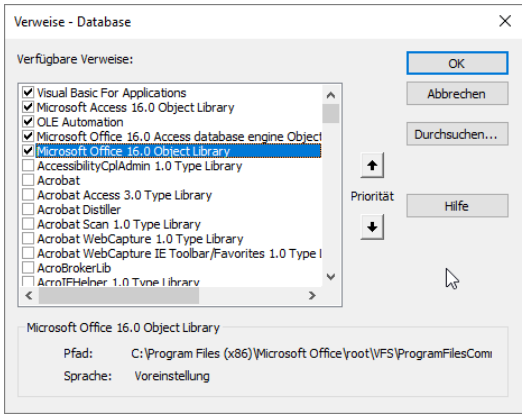


Abbildung 16.2: Die Office-Bibliothek im *Verweise*-Dialog

Danach können Sie mit der folgenden Prozedur, die Sie in einem neuen Standardmodul anlegen, alle vorhandenen Kontextmenüs ausgeben:

```
Public Sub KontextmenuesAusgeben()
    Dim cbr As CommandBar
    For Each cbr In CommandBars
        If cbr.Type = MsoBarType.msoBarTypePopup Then
            Debug.Print cbr.Name
        End If
    Next cbr
End Sub
```

In dieser Prozedur verwenden wir die *CommandBars*-Auflistung, deren *CommandBar*-Elemente wir über die Variable *cbr* in einer *For Each*-Schleife durchlaufen.

Damit Sie nicht alle *CommandBar*-Objekte ausgeben, denn die nicht mehr verwendeten Menüleisten und Symbolleisten sind auch noch über diese Auflistung verfügbar, filtern wir in der *If... Then*-Bedingung nach den Einträgen, deren Eigenschaft *Type* dem Wert *MsoBarType.msoBarTypePopup* entsprechen.

Praktisch neben dem Namen einer Kontextmenüleiste ist noch die Eigenschaft *BuiltIn*. Mit dieser können wir herausfinden, ob es sich um ein eingebautes Kontextmenü oder um ein benutzerdefiniertes handelt. Wenn Sie die *Debug.Print*-Anweisung der vorherigen Prozedur wie folgt erweitern, erhalten Sie den Wert *True* für eingebaute Elemente und *False* für benutzerdefinierte Elemente:

```
Debug.Print cbr.Name, cbr.BuiltIn
```

Wir wollen uns in den folgenden Abschnitten sowohl um die Erweiterung eingebauter Kontextmenüs als auch um die Erstellung benutzerdefinierter Kontextmenüs kümmern.

16.1.2 Steuerelemente ausgeben

Mit einer erweiterten Version der vorherigen Prozedur können wir auch noch die in einem Kontextmenü enthaltenen Steuerelemente referenzieren und im Direktbereich ausgeben. Dazu verwenden wir eine Laufvariable des Typs *CommandBarControl*. Die Steuerelemente können wir wiederum über eine Auflistung des *CommandBar*-Elements namens *Controls* referenzieren. Die enthaltenen Elemente durchlaufen wir in einer *For Each*-Schleife und geben darin jeweils den Wert der Eigenschaft *Caption*, also Beschriftung, im Direktbereich aus:

```
Public Sub KontextmenuesUndSteuerelementeAusgeben()
    Dim cbr As CommandBar
    Dim cbc As CommandBarControl
    For Each cbr In CommandBars
```

```
            If cbr.Type = MsoBarType.msoBarTypePopup Then
                Debug.Print cbr.Name
                For Each cbc In cbr.Controls
                    Debug.Print , cbc.Caption, cbc.Type
                Next cbc
            End If
        Next cbr
    End Sub
```

16.1.3 Steuerelementtypen von CommandBarControl-Elementen

Genau wie bei den Steuerelementen in Formularen gibt es auch im Kontextmenü verschiedene Steuerelementtypen. Diesen finden wir über die Eigenschaft *Type* des *CommandBarControl*-Elements heraus, das wir in der obigen Prozedur auch schon mit ausgeben.

Hier taucht im Direktfenster meist der Wert *1* auf, aber auch der Wert *10* ist vorhanden. Um zu erfahren, welche Konstanten dahinterstecken, öffnen wir den Objektkatalog des VBA-Editors mit der Taste *F2* und suchen dort nach *msoControlType*.

Als Ergebnis erhalten wir den gewünschten Eintrag sowie eine Liste der enthaltenen Elemente (siehe Abbildung 16.3). Der Wert *1* entspricht also etwa der Konstanten *msoControlButton*. Für den Wert *10* finden wir die Konstante *msoControlPopup*, was einem Untermenü entspricht.

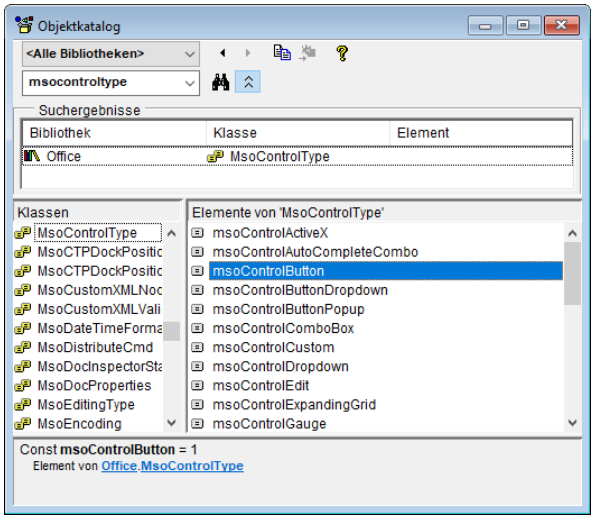


Abbildung 16.3: Auflistung der verschiedenen Typen der Kontextmenü-Steuerelemente

16.2 Benutzerdefiniertes Kontextmenü erstellen

Wenn Sie ein benutzerdefiniertes Kontextmenü erstellen wollen, verwenden Sie etwa die folgende Prozedur:

```
Public Sub KontextmenueHinzufuegen()
    Dim cbr As CommandBar
    Dim cbb As CommandBarButton
    On Error Resume Next
    CommandBars("Benutzerdefiniertes Kontextmenü").Delete
    On Error GoTo 0
    Set cbr = CommandBars.Add("Benutzerdefiniertes Kontextmenü", msoBarPopup)
    With cbr
        Set cbb = cbr.Controls.Add(msoControlButton)
        With cbb
            .Caption = "Beispielbefehl"
            .OnAction = "Beispielfunktion"
        End With
    End With
    cbr.ShowPopup
End Sub
```

Die Prozedur schaltet zunächst die eingebaute Fehlerbehandlung von Access aus und löscht eine eventuell bereits vorhandene Instanz des Kontextmenüs namens *Benutzerdefiniertes Kontextmenü* mit der Methode *Delete* für den gleichnamigen Eintrag der *CommandBars*-Auflistung. Danach schaltet sie die Fehlerbehandlung wieder ein.

Danach erstellen wir mit der Methode *Add* der *CommandBars*-Auflistung ein neues *CommandBar*-Element. Dabei übergeben wir als Parameter den Namen für das Element sowie den gewünschten Typ, in diesem Fall *msoBarPopup* für ein Kontextmenü. Dieses Objekt referenzieren wir mit der Variablen *cbo*, damit wir gleich im Anschluss noch eine Schaltfläche zu diesem Menü hinzufügen können.

Diese Schaltfläche fügen wir über die *Add*-Methode der *Controls*-Auflistung von *cbr* hinzu und geben dabei den Typ *msoControlButton* an. Das neue Objekt referenzieren wir mit der Variablen *cbb* des Typs *CommandBarButton*. Für dieses Objekt legen wir nun noch zwei Eigenschaften fest. Für *Caption*, also für die Beschriftung, geben wir den Wert *Beispielbefehl* an. Für die Eigenschaft *OnAction* legen wir den Wert *Beispielfunktion* fest. Dabei handelt es sich um den Namen der VBA-Funktion, den dieser Kontextmenü-Befehl aufrufen soll.

Schließlich kommt der entscheidende Schritt: Die Methode *ShowPopup* des *CommandBar*-Objekts zeigt das Kontextmenü an. Dieses ist natürlich nicht besonders umfangreich, denn es

enthält nur eine einzige Anweisung, aber es funktioniert. Dafür legen wir noch die aufzurufende Funktion an, die wie folgt aussieht und eine schlichte Meldung anzeigen soll:

```
Public Function Beispielfunktion()
    MsgBox "Kontextmenübefehl aufgerufen"
End Function
```

Der Aufruf dieser Prozedur zeigt schließlich das Kontextmenü an, und nach dem Anklicken des einzigen Befehls erscheint die gewünschte Meldung (siehe Abbildung 16.4).

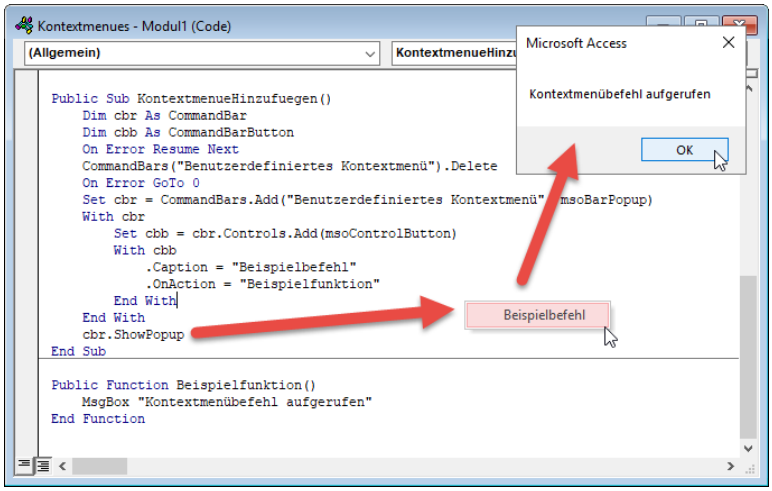


Abbildung 16.4: Anzeigen eines Kontextmenüs mit Meldungsfenster als Funktion

16.3 Kontextmenü per Mausklick

Entscheidend ist allerdings nun nicht, dass wir ein Kontextmenü durch den Aufruf einer VBA-Prozedur anzeigen können, sondern das soll natürlich per Rechtsklick auf ein Element der Benutzeroberfläche geschehen. Dafür gibt es für Formulare und Steuerelemente Ereignisse, die beim Herunterdrücken und beim Loslassen der Maustaste ausgelöst werden. Welches Ereignis wir benötigen, finden wir durch kurzes Experimentieren heraus.

Das Ergebnis: Ein Kontextmenü erscheint in dem Moment, in dem wir die rechte Maustaste wieder loslassen. Also benötigen wir die Ereignisprozedur *Bei Maustaste* auf. Diese implementieren wir für ein noch leeres Formular, indem wir für die Eigenschaft *Bei Maustaste auf* den Wert *[Ereignisprozedur]* eintragen und auf die Schaltflächen mit den drei Punkten rechts im Eigenschaftsfeld klicken.

17 Praktische Beispiele

In diesem Kapitel finden Sie einige praktische Beispiele für den Einsatz von Formularen.

17.1 Übersichtsformular mit Detailformular

Eine der meistgenutzten Kombinationen ist die Anzeige einer Übersicht in einem Formular und die Möglichkeit, aus diesem Formular heraus ein Detailformular zum Anzeigen des ausgewählten Datensatzes sowie zum Anlegen eines neuen Datensatzes zu öffnen. Außerdem können Sie noch eine Schaltfläche zum Löschen des aktuell markierten Datensatzes unterbringen. Im Entwurf sieht das hier verwendete Formular *frmKundenVerwaltenUebersicht* mit seinem Unterformular *sfmKundenVerwaltenUebersicht* wie in Abbildung 17.1 aus.

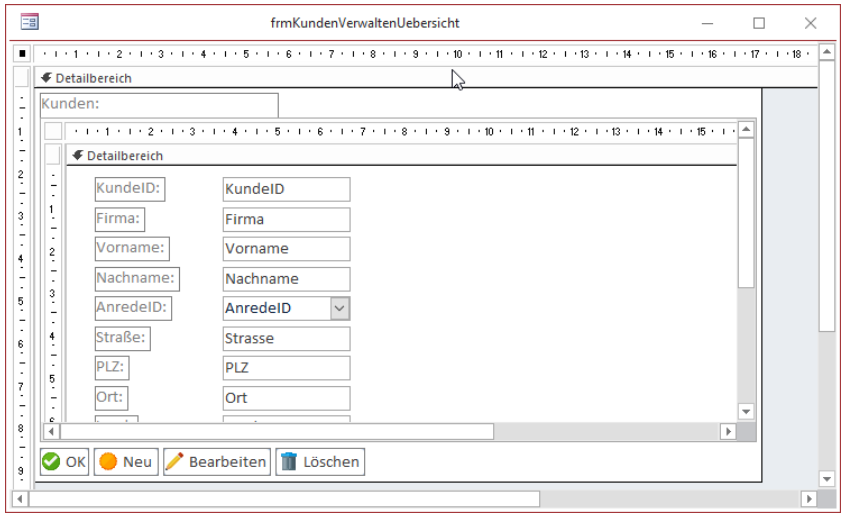


Abbildung 17.1: Formular zum Verwalten von Kunden

Wir haben hier vier Schaltflächen namens *cmdOK*, *cmdNeu*, *cmdBearbeiten* und *cmdLoeschen* untergebracht.

17.1.1 Datensatz anlegen

Die Schaltfläche *cmdNeu* löst die folgende Ereignisprozedur aus:

```
Private Sub cmdNeu_Click()
    Dim lngKundeID As Long
```

```
    DoCmd.OpenForm "frmKundenVerwaltenDetail", WindowMode:=acDialog, DataMode:=acFormAdd
    If IstFormularGeoeffnet("frmKundenVerwaltenDetail") Then
        lngKundeID = Forms!frmKundenVerwaltenDetail!KundeID
        DoCmd.Close acForm, "frmKundenVerwaltenDetail" 'muss hierhin!
        Me!sfmKundenVerwaltenUebersicht.Form.Requery
        Me!sfmKundenVerwaltenUebersicht.Form.Recordset.FindFirst "KundeID = " _
            & lngKundeID
    End If
End Sub
```

Die Prozedur öffnet das Formular mit der *DoCmd.OpenForm*-Methode und sorgt dabei mit dem Parameter *WindowMode* dafür, dass das Formular als modaler Dialog geöffnet wird und mit *DataMode* dafür, dass es einen neuen, leeren Datensatz anzeigt. Die Prozedur wird angehalten, bis der Benutzer das Detailformular *frmKundenVerwaltenDetail* entweder mit der *OK*- oder mit der *Abbrechen*-Schaltfläche schließt. Im ersten Fall ist das Formular noch geöffnet, aber unsichtbar. Dann lesen wir den Wert des Feldes *KundeID* des neuen Datensatzes aus und speichern diesen in der Variablen *lngKundeID*. Dann schließen wir das Formular, bevor wir die Datensatzquelle des Unterformulars aktualisieren und den Datensatz mit dem neuen Wert im Feld *KundeID* markieren. Das Schließen des Detailformulars muss vorher erfolgen, weil der Datensatz sonst nicht gespeichert ist und das Übersichtsformular nicht den neuen Datensatz anzeigen kann.

17.1.2 Datensatz bearbeiten

Um den aktuell im Datenblatt des Unterformulars markierten Kunden zu bearbeiten, klicken Sie auf die Schaltfläche *cmdBearbeiten*. Diese löst die folgende Prozedur aus:

```
Private Sub cmdBearbeiten_Click()
    Dim lngKundeID As Long
    lngKundeID = Nz(Me!sfmKundenVerwaltenUebersicht.Form.KundeID, 0)
    If Not lngKundeID = 0 Then
        DoCmd.OpenForm "frmKundenVerwaltenDetail", WindowMode:=acDialog, _
            WhereCondition:="KundeID = " & lngKundeID, DataMode:=acFormEdit
        If IstFormularGeoeffnet("frmKundenVerwaltenDetail") Then
            DoCmd.Close acForm, "frmKundenVerwaltenDetail" 'muss hierhin!
        End If
    End If
End Sub
```

Im ersten Schritt speichern wir den Primärschlüsselwert des aktuell markierten Datensatzes in der Variablen *lngKundeID*. Diese kann dann auch den Wert *0* aufweisen, wenn zum Beispiel aktuell ein neuer, leerer Datensatz markiert ist. Wir öffnen nur dann das Formular *frmKunden-VerwaltenDetail*, wenn *lngKundeID* nicht *0* ist. Beim Öffnen übergeben wir Parameter, die dafür

sorgen, dass das Formular als modaler Dialog geöffnet wird (*WindowMode*), dass der richtige Datensatz erscheint (*WhereCondition*) und dass das Formular zum Bearbeiten des Datensatzes geöffnet wird (*DataMode*). Die Prozedur wird an dieser Stelle angehalten, bis der Benutzer das geöffnete Formular wieder schließt. Danach prüfen wir, ob das Formular noch geöffnet (der Benutzer hat auf *OK* geklickt) oder geschlossen ist (Formular wurde mit *Abbrechen* geschlossen). Im ersteren Fall schließen wir das noch geöffnete Formular. Änderungen am Datensatz werden in reinen Access-Datenbanken in der Regel direkt in das aufrufende Formular übertragen und aktualisiert.

17.1.3 Datensatz löschen

Die nächste Schaltfläche mit dem Text *Löschen* soll den aktuell markierten Datensatz aus dem Unterformular löschen. Dazu ermitteln wir wieder den Wert des Primärschlüsselfeldes *KundeID* des aktuell markierten Datensatzes. Ist dieser ungleich *0*, löschen wir den Datensatz durch den Aufruf einer *DELETE*-Anweisung, welche den Wert aus *lngKundeID* als Vergleichswert des Kriteriums für den zu löschenden Datensatz verwendet. Nach dem Löschen aktualisieren wir mit der *Requery*-Methode das Unterformular, damit der gelöscht Datensatz nicht mehr angezeigt wird:

```
Private Sub cmdLoeschen_Click()
    Dim lngKundeID As Long
    Dim db As DAO.Database
    lngKundeID = Nz(Me!sfmKundenVerwaltenUebersicht.Form.KundeID, 0)
    If Not lngKundeID = 0 Then
        Set db = CurrentDb
        db.Execute "DELETE FROM tblKunden WHERE KundeID = " & lngKundeID, dbFailOnError
        Me!sfmKundenVerwaltenUebersicht.Form.Requery
    End If
End Sub
```

Warum nutzen wir hier nicht etwa die Methoden der Benutzeroberfläche, um das Unterformular zu markieren und den Datensatz dann etwa mit *RunCommand.DeleteRecord* zu löschen? Weil wir uns gegebenenfalls noch um Hindernisse kümmern müssen. Eines tritt zum Beispiel auf, wenn wir einen Kunden löschen wollen, zu dem es schon Bestelldatensätze gibt. Dann müssten wir eine entsprechende Meldung einbauen, um dem Kunden mitzuteilen, dass der Kunde nicht gelöscht werden kann.

Dafür können Sie den Code beispielsweise wie folgt erweitern:

```
If IsNull(DLookup("BestellungID", "tblBestellungen", "KundeID = " & lngKundeID)) Then
    Set db = CurrentDb
    db.Execute "DELETE FROM tblKunden WHERE KundeID = " & lngKundeID, dbFailOnError
    Me!sfmKundenVerwaltenUebersicht.Form.Requery
```

```
Else
    MsgBox "Der Kunde kann nicht gelöscht werden, " _
        & "da bereits Bestellungen für ihn vorhanden sind."
End If
```

Dadurch wird bei Vorhandensein von verknüpften Datensätzen eine Meldung ausgegeben, dass der Datensatz nicht gelöscht werden kann.

17.1.4 Formular schließen

Die Schaltfläche unten links mit dem Text *OK* soll das Formular schließen. Das erledigt die folgende Ereignisprozedur durch den Aufruf der *DoCmd.Close*-Methode mit den entsprechenden Parametern:

```
Private Sub cmdOK_Click()
    DoCmd.Close acForm, Me.Name
End Sub
```

17.1.5 Schaltflächen im Hauptformular aktivieren und deaktivieren

Wenn die Datenblattansicht im Unterformular einen neuen, leeren Datensatz anzeigt, macht es keinen Sinn, die Schaltflächen *Bearbeiten* oder *Löschen* zu betätigen. Die obigen Ereignisprozeduren prüfen daher, ob aktuell ein Datensatz markiert ist oder nicht. Noch schöner für den Benutzer wäre es allerdings, wenn die Schaltflächen deaktiviert werden, wenn ihre Funktion nicht verfügbar ist. Das sähe dann etwa wie in Abbildung 17.2 aus.

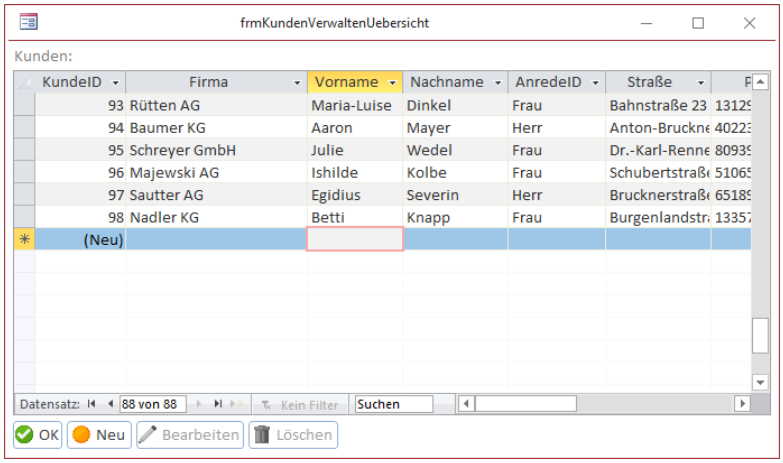


Abbildung 17.2: Bei leerem Datensatz werden die Schaltflächen *Bearbeiten* und *Löschen* deaktiviert

Um dies zu erreichen, müssen wir eine Ereignisprozedur für das Ereignis *Beim Anzeigen* des Unterformulars anlegen.

Dieses wird dann bei jedem Datensatzwechsel ausgelöst, also auch direkt beim Öffnen des Formulars, wenn der erste Datensatz angezeigt wird. Diese Prozedur füllen wir mit den folgenden Anweisungen:

```
Private Sub Form_Current()
    If IsNull(Me!KundeID) Or Me.NewRecord Then
        Me.Parent!cmdBearbeiten.Enabled = False
        Me.Parent!cmdLoeschen.Enabled = False
    Else
        Me.Parent!cmdBearbeiten.Enabled = True
        Me.Parent!cmdLoeschen.Enabled = True
    End If
End Sub
```

Die *If...Then*-Bedingung prüft, ob der Datensatzzeiger gerade auf einen vorhandenen Datensatz zeigt oder auf einen neuen, leeren Datensatz. Im ersteren Fall werden die beiden Schaltflächen *cmdBearbeiten* und *cmdLoeschen* aktiviert, sonst werden sie deaktiviert.

17.2 Schnelle Suche mit Listenfeld

Die Anzeige von Daten in einem Listenfeld als Übersichtsformular ist praktisch, wenn Sie etwa per Doppelklick auf einen der Einträge ein Detailformular zum jeweiligen Datensatz öffnen wollen. Je nach der Menge der angezeigten Datensätze ist das Auffinden des gewünschten Datensatzes jedoch mitunter recht anstrengend. Also fügen wir eine kleine Suchfunktion hinzu, mit der Sie die Datensätze im Listenfeld schnell filtern können.

Das Listenfeld namens *lstKunden* im Formular *frmListenfeldsuche* soll die folgende Abfrage für die Eigenschaft *Datensatzherkunft* verwenden:

```
SELECT tblKunden.KundeID, tblKunden.Firma, tblKunden.Vorname, tblKunden.Nachname FROM
tblKunden ORDER BY tblKunden.Firma;
```

Diese Abfrage haben wir mit dem Abfrage-Generator auf Basis der Tabelle *tblKunden* zusammengestellt. Damit das Listenfeld das Primärschlüsselfeld nicht anzeigt und die drei übrigen Felder sich gleichmäßig auf die Listenfeldbreite verteilen, stellen wir die Eigenschaft *Spaltenanzahl* auf *4* und die Eigenschaft *Spaltenbreiten* auf *0cm* ein.

Das Textfeld zur Eingabe der Suchbegriffe erhält den Namen *txtSuche* und sieht wie in Abbildung 17.3 aus.

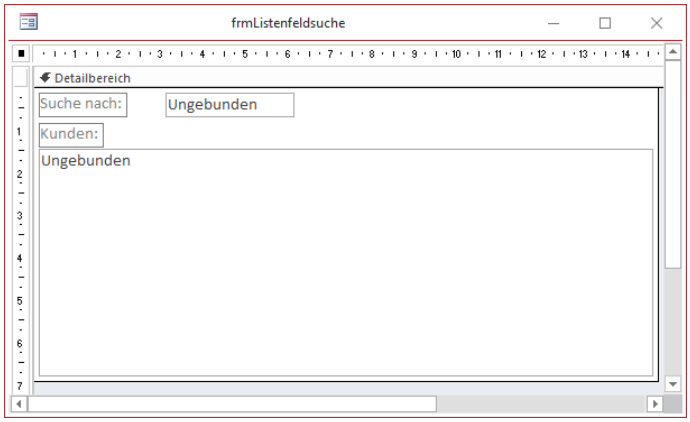


Abbildung 17.3: Das Formular mit dem zu durchsuchenden Listenfeld in der Entwurfsansicht

17.2.1 Suche nach der Firma direkt nach Eingabe eines jeden Zeichens

Als Erstes wollen wir eine Suche implementieren, deren Ergebnis nach Eingabe eines jeden Zeichens aktualisiert wird. Dazu hinterlegen wir für die Ereigniseigenschaft *Bei Änderung* des Textfeldes die folgende Ereignisprozedur:

```
Private Sub txtSuche_Change()
    Dim strSuchbegriff As String
    strSuchbegriff = Me!txtSuche.Text
    Me!lstKunden.RowSource = "SELECT KundeID, Firma, Vorname, Nachname FROM tblKunden " _
        & "WHERE Firma LIKE '*" & strSuchbegriff & "*' ORDER BY Firma;"
End Sub
```

Die Prozedur ermittelt über die Eigenschaft *Text* des Textfeldes den aktuell im Textfeld angezeigten Text. Es gibt einen Unterschied zwischen den Eigenschaften *Text* und der Standardeigenschaft *Value* des Textfeld-Steuerelements. *Value* liefert den zuletzt gespeicherten Wert des Textfeldes, *Text* den aktuellen Inhalt, auch wenn dieser nach dem letzten Speichern oder Füllen geändert wurde.

Den aktuellen Text speichert die Prozedur in der Variablen *strSuchbegriff*, der dann als Vergleichswert der *WHERE*-Klausel der Abfrage verwendet wird, die wir der Eigenschaft *RowSource* des Listenfeldes zuweisen (im Eigenschaftsfenster *Datensatzquelle*). Dabei fügen wir vorn und hinten zum Suchbegriff noch jeweils das Sternchen (*) als Platzhalter hinzu. Der Suchbegriff lautet dann etwa **a**, wenn der Benutzer den Buchstaben *a* eingegeben hat.

Warum geben wir hier direkt Platzhalter vor, statt dies dem Benutzer zu überlassen? Weil es sonst eine Weile dauern kann, bis das Listenfeld ein Ergebnis anzeigt – nämlich so lange, bis der Benutzer den kompletten Wert eines der Werte im Feld *Firma* eingetragen hat.

Geben wir beispielsweise die beiden Buchstaben *be* ein, liefert das Listenfeld alle Einträge, deren Feld *Firma* an beliebiger Stelle die Zeichenkfolge *be* aufweist (siehe Abbildung 17.4).

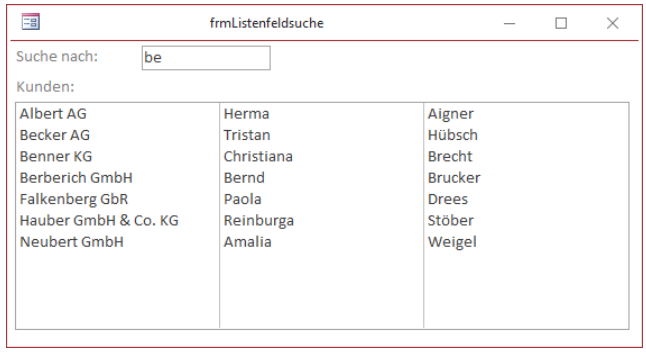


Abbildung 17.4: Suchergebnisse mit der Zeichenkette *be*

Variante: Firmen, die mit dem Suchbegriff beginnen

Eine Variante wäre es, nur hinten an den Suchbegriff das Sternchen als Platzhalter anzufügen. Dazu fügen wir ein weiteres Suchfeld namens *txtSucheStart*. Die Ereignisprozedur *txtSucheStart_Change* würde dann so aussehen:

```
Private Sub txtSucheStart_Change()
    Dim strSuchbegriff As String
    strSuchbegriff = Me!txtSucheStart.Text
    Me!lstKunden.RowSource = "SELECT KundeID, Firma, Vorname, Nachname " _
        & "FROM tblKunden WHERE Firma LIKE '" & strSuchbegriff & "*' ORDER BY Firma;"
End Sub
```

Damit erscheinen nun Einträge wie in Abbildung 17.5.

Abbildung 17.5: Ergebnisse mit der gesuchten Zeichenkette am Anfang

17.2.2 Mehrere Felder gleichzeitig nach dem gleichen Wert durchsuchen

Die nächste Variante soll gleich alle angezeigten Felder durchsuchen, also *Firma*, *Vorname* und *Nachname*. Dabei soll sich das eingegebene Zeichen am Anfang der durchsuchten Feldinhalte befinden.

Der Code der Prozedur sieht wie folgt aus:

```
Private Sub txtSucheAlle_Change()
    Dim strSuchbegriff As String
    strSuchbegriff = Me!txtSucheAlle.Text
    Me!lstKunden.RowSource = "SELECT KundeID, Firma, Vorname, Nachname " _
        & "FROM tblKunden WHERE Firma LIKE '" & strSuchbegriff & "*' " _
        & "OR Vorname LIKE '" & strSuchbegriff & "*' " _
        & "OR Nachname LIKE '" & strSuchbegriff & "*' ORDER BY Firma;"
End Sub
```

Nun liefert das Listenfeld Einträge, die in einem der drei Felder den Wert *Be* am Wortanfang enthalten (siehe Abbildung 17.6).

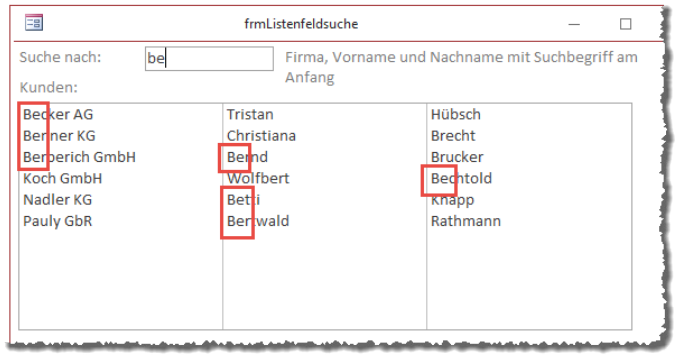


Abbildung 17.6: Suchbegriffe in allen drei Spalten

17.2.3 Mehrere Felder nach verschiedenen Werten durchsuchen

Dass man Datensätze sucht, die in einem von mehreren Feldern eine bestimmte Zeichenkette enthalten, ist eher selten. Öfter wird man vielleicht Werte für die drei Felder *Firma*, *Vorname* und *Nachname* in verschiedene Textfelder eingeben wollen.

Das lässt sich ebenfalls mit einer Schnellsuche realisieren, allerdings ist hier etwas mehr Aufwand nötig.

Dazu legen wir ein neues Formular auf Basis des vorherigen Formulars an, indem wir dieses kopieren und unter dem Namen *frmListenfeldsucheAnd* speichern. Dann ändern wir die Bezeichnung der Textfelder für die Suche auf *txtSucheFirma*, *txtSucheVorname* und *txtSucheNachname* (siehe Abbildung 17.7).

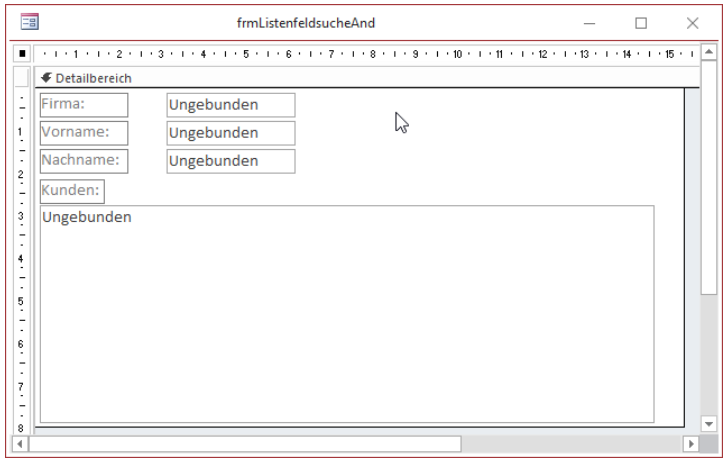


Abbildung 17.7: Suche mit drei Kriterien, die per *AND* verknüpft werden sollen

Wir wollen immer auf den Text der jeweiligen Steuerelemente zugreifen. Damit wir nicht immer neu den Inhalt auslesen müssen (für das aktuelle Textfeld mit der *Text*-Eigenschaft, für die übrigen Textfelder mit der *Value*-Eigenschaft), speichern wir die Werte nach der jeweiligen Änderung in den folgenden Variablen:

```
Dim strSucheFirma As String
Dim strSucheNachname As String
Dim strSucheVorname As String
```

Wenn der Benutzer beispielsweise einen Wert in das Textfeld *txtSucheFirma* eingibt, wird der Wert der Eigenschaft *Text* immer direkt in die Variable *strSucheFirma* geschrieben.

Danach rufen wir dann die Prozedur *KundenFiltern* auf, die wir weiter unten beschreiben und die die eigentliche Arbeit macht:

```
Private Sub txtSucheFirma_Change()
    strSucheFirma = Me!txtSucheFirma.Text
    KundenFiltern
End Sub
```

Auf die gleiche Weise gehen wir vor, wenn der Benutzer einen Text in eines der beiden Textfelder *txtSucheVorname* und *txtSucheNachname* eingibt:

```
Private Sub txtSucheVorname_Change()
    strSucheVorname = Me!txtSucheVorname.Text
    KundenFiltern
End Sub

Private Sub txtSucheNachname_Change()
    strSucheNachname = Me!txtSucheNachname.Text
    KundenFiltern
End Sub
```

Die Prozedur *KundenFiltern* wird das erste Mal aufgerufen, wenn zumindest in einer der Variablen *strSucheFirma*, *strSucheVorname* oder *strSucheNachname* ein Wert eingegeben wurde. Sie prüft jeweils, ob eine der Variablen einen Wert von der Länge größer als 0 enthält.

Falls ja, fügt sie zu einer Variablen namens *strWhere* einen Ausdruck wie *AND Firma LIKE 'b*'* hinzu. Hat der Benutzer für alle drei Textfelder einen Suchbegriff eingegeben, besteht *strWhere* entsprechend aus drei solcher Elemente:

```
Private Sub KundenFiltern()
    Dim strSQL As String
    Dim strWhere As String
    If Len(strSucheFirma) > 0 Then
        strWhere = strWhere & " AND Firma LIKE '" & strSucheFirma & "*' "
    End If
    If Len(strSucheVorname) > 0 Then
        strWhere = strWhere & " AND Vorname LIKE '" & strSucheVorname & "*' "
    End If
    If Len(strSucheNachname) > 0 Then
        strWhere = strWhere & " AND Nachname LIKE '" & strSucheNachname & "*' "
    End If
```

Enthält *strWhere* nach den drei *If...Then*-Bedingungen mindestens eine Bedingung, wird das führende *AND* mit der *Mid*-Funktion abgeschnitten. Außerdem stellt die Prozedur dann das Schlüsselwort *WHERE* voran, sodass wir etwa *WHERE Firma LIKE 'b*'* erhalten:

```
    If Len(strWhere) > 0 Then
        strWhere = Mid(strWhere, 5)
        strWhere = " WHERE " & strWhere
    End If
```

Diesen Teil fügen wir dann in die *SELECT*-Anweisung zwischen der *FROM*- und der *ORDER BY*-Klausel ein und stellen *RowSource* auf *strSQL* ein:

```
    strSQL = "SELECT KundeID, Firma, Vorname, Nachname FROM tblKunden" & strWhere _
```

```
        & " ORDER BY Firma"
    Me!lstKunden.RowSource = strSQL
End Sub
```

Damit können wir dann nach dem Wert eines jeden Feldes filtern und erhalten etwa ein Ergebnis wie in Abbildung 17.8.

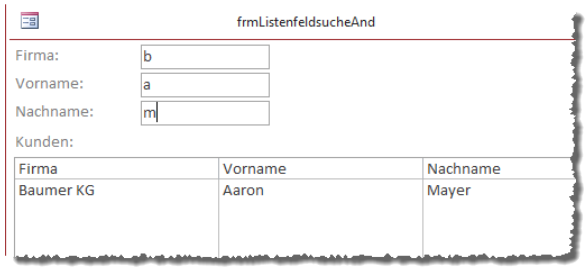


Abbildung 17.8: Suche nach Einträgen, die alle genannten Bedingungen erfüllen

Wichtig ist auch, dass wir wieder alle Datensätze erhalten, wenn der Benutzer alle drei Textfelder leert. Dann entfällt die *WHERE*-Klausel und die ursprüngliche *SELECT*-Anweisung wird wieder als Datensatzherkunft eingestellt.

Das können wir noch mit einer Schaltfläche namens *cmdFilterLeeren* vereinfachen, deren Ereignisprozedur alle drei Textfelder leert und die Datensatzherkunft auf den Anfangszustand einstellt:

```
Private Sub cmdFilterLeeren_Click()
    Me!txtSucheFirma = ""
    Me!txtSucheNachname = ""
    Me!txtSucheVorname = ""
    strSucheFirma = ""
    strSucheNachname = ""
    strSucheVorname = ""
    KundenFiltern
End Sub
```

Wir müssen hier nicht nur die Textfelder leeren, sondern auch die Variablen für die Suchbegriffe, da diese nicht automatisch beim Einstellen der Inhalte der Textfelder aktualisiert werden.

17.2.4 Schnellsuche für die Datenblattansicht

Die gleiche Suchfunktion können wir auch für ein Unterformular in der Datenblattansicht programmieren. Hier gibt es nur kleine Unterschiede. Wir erstellen dazu ein Unterformular namens

frmKundensuche, welches die gleiche Abfrage wie das Listenfeld der vorherigen Beispiele als Datensatzquelle verwendet (siehe Abbildung 17.9).

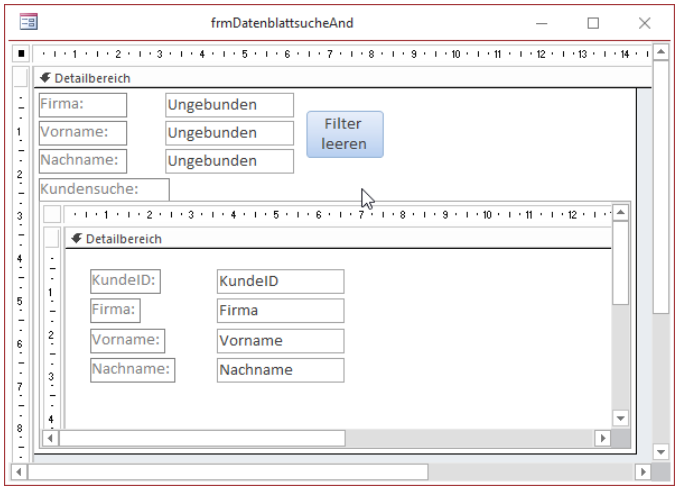


Abbildung 17.9: Suche in einem Unterformular in der Datenblattansicht

Die Textfelder und die Schaltfläche zum Zurücksetzen des Filters können wir beibehalten. Auch die Ereignisprozeduren für die Änderungen am Inhalt der Textfelder bleiben bestehen. Die einzigen Änderungen sind in der Prozedur *KundenFiltern* notwendig. Diese sieht nun so aus:

```
Private Sub KundenFiltern()
    Dim strWhere As String
    If Len(strSucheFirma) > 0 Then
        strWhere = strWhere & " AND Firma LIKE '" & strSucheFirma & "*' "
    End If
    If Len(strSucheVorname) > 0 Then
        strWhere = strWhere & " AND Vorname LIKE '" & strSucheVorname & "*' "
    End If
    If Len(strSucheNachname) > 0 Then
        strWhere = strWhere & " AND Nachname LIKE '" & strSucheNachname & "*' "
    End If
```

Interessant wird es an dieser Stelle. Hier trennen wir nur noch das eventuell vorhandene erste *AND* ab, fügen aber nicht mehr das Schlüsselwort *WHERE* vorne an den Filterausdruck an:

```
    If Len(strWhere) > 0 Then
        strWhere = Mid(strWhere, 5)
    End If
```

Der Grund: Wir müssen dem Unterformular nur den Kriterienausdruck übergeben, nicht die vollständige Abfrage. Also stellen wir nur noch die *Filter*-Eigenschaft des Unterformulars im Unterformular-Steuerelement auf den Wert aus *strWhere* ein. Außerdem müssen wir den Filter mit dem Wert *True* für die Eigenschaft *FilterOn* aktivieren:

```
    Me!sfmKundensuche.Form.Filter = strWhere
    Me!sfmKundensuche.Form.FilterOn = True
End Sub
```

Das Formular sieht dann in Aktion wie in Abbildung 17.10 aus.

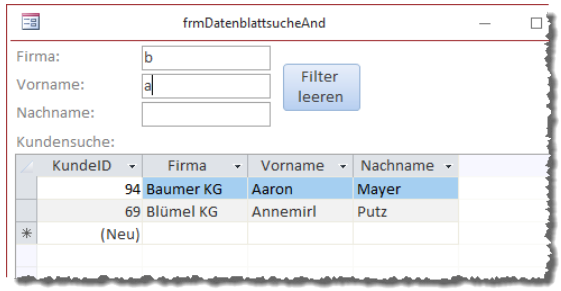


Abbildung 17.10: Gefiltertes Unterformular

17.2.5 Andere Vergleichswerte

Wenn Sie bei einem der vorherigen Beispiele etwa nach Einträgen suchen wollen, die an beliebiger Stelle eine bestimmte Zeichenfolge aufweist, können Sie dem Benutzer den Hinweis geben, dass er selbständig das Sternchen als Platzhalter eingibt. Mit **A** suchen die Formulare dann beispielsweise nach allen Vorkommen von A in den betroffenen Feldern.

17.3 m:n-Daten im Bestellformular

Weiter oben haben wir unter »m:n-Beziehung mit Verknüpfungsdaten« ab Seite 114 bereits begonnen, ein Bestellformular zu programmieren. Dieses hatte allerdings noch ein paar Schwachstellen, zum Beispiel wurden beim Hinzufügen eines neuen Artikels noch nicht der Einzelpreis und der Mehrwertsteuersatz automatisch zur Bestellposition hinzugefügt. Dazu ist der Einsatz von VBA nötig, was wir im oben genannten Abschnitt noch nicht besprochen hatten.

Nun sind Sie jedoch für diese Aufgabe gerüstet, sodass wir uns das hier nun ansehen können.

Wenn Sie in diesem Formular einen neuen Artikel auswählen und somit eine Bestellposition anlegen, sollen nun der Einzelpreis für den hinzugefügten Artikel sowie der Mehrwertsteuersatz hinzugefügt werden. Und auch das Feld *Menge* soll mit dem Wert *1* vorbelegt werden. Nach

dem Auswählen des Artikels können wir ein Ereignis des Kombinationsfeldes namens *Nach Aktualisierung* nutzen. Bevor wir dieses anlegen, stellen wir jedoch noch den Namen des Kombinationsfeld-Steuerelements auf *cboArtikelID* ein. So erkennen wir auch später im Code, dass es sich bei den Ereignisprozeduren um solche für ein Kombinationsfeld handelt. Anschließend hinterlegen wir die Ereignisprozedur, die wie folgt aussieht:

```
Private Sub cboArtikelID_AfterUpdate()
    Dim lngMehrwertsteuersatzID As Long
    Me!Einzelpreis = DLookup("Einzelpreis", "tblArtikel", "ArtikelID = " & Me!ArtikelID)
    lngMehrwertsteuersatzID = DLookup("MehrwertsteuersatzID", "tblArtikel", _
        "ArtikelID = " & Me!ArtikelID)
    Me!Mehrwertsteuersatz = DLookup("Mehrwertsteuersatz", "tblMehrwertsteuersaetze", _
        "MehrwertsteuersatzID = " & lngMehrwertsteuersatzID)
    Me!Menge = 1
End Sub
```

Die erste Anweisung weist dem Feld *Einzelpreis* den Wert des Feldes *Einzelpreis* für den Datensatz der Tabelle *tblArtikel* zu, der dem ausgewählten Artikel entspricht. Die zweite ermittelt den Wert des Fremdschlüsselfeldes *MehrwertsteuersatzID* des Datensatzes in der Tabelle *tblArtikel*.

Diesen speichern wir in der Variablen *lngMehrwertsteuersatzID*. Da wir im Feld *Mehrwertsteuersatz* der aktuellen Bestellposition direkt dem Mehrwertsteuersatz und nicht den Wert des Primärschlüsselfeldes des passenden Datensatzes der Tabelle *tblMehrwertsteuersaetze* eintragen wollen, ermitteln wir diesen wiederum mit einer *DLookup*-Funktion. Und schließlich stellen wir noch den Wert des Feldes *Menge* auf den Wert *1* ein.

Wenn Sie danach für eine vorhandene Bestellung durch Auswahl eines Artikels eine neue Bestellposition hinzufügen, werden die Daten aus der zugrunde liegenden Tabelle *tblArtikel* direkt in den Datensatz der Tabelle *tblBestellpositionen* im Unterformular eingetragen (siehe Abbildung 17.11).

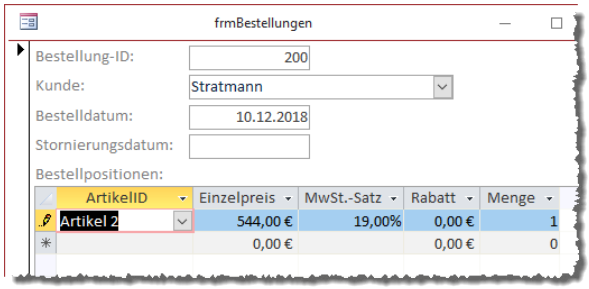


Abbildung 17.11: *Einzelpreis*, *Mehrwertsteuersatz* und *Menge* werden nach Auswahl des Artikels automatisch eingestellt

17.3.1 Fehlermeldung bei fehlendem verknüpftem Datensatz im Hauptformular

Weiter oben unter »Hinzufügen von Datensätzen in Unterformular bei leerem Hauptformular verhindern« ab Seite 340 haben wir schon einmal gezeigt, dass es sein kann, dass ein im Unterformular ausgewählter Datensatz im Nirvana verschwindet, wenn das Hauptformular keinen passenden Datensatz enthält. Dies lief im dort beschriebenen Beispiel allerdings ohne Fehler ab. Wenn wir im Formular *frmBestellungen* probieren, einen neuen Datensatz im Unterformular anzulegen, ohne dass ein Datensatz im Hauptformular vorliegt, erhalten wir sogar eine Fehlermeldung (siehe Abbildung 17.12).

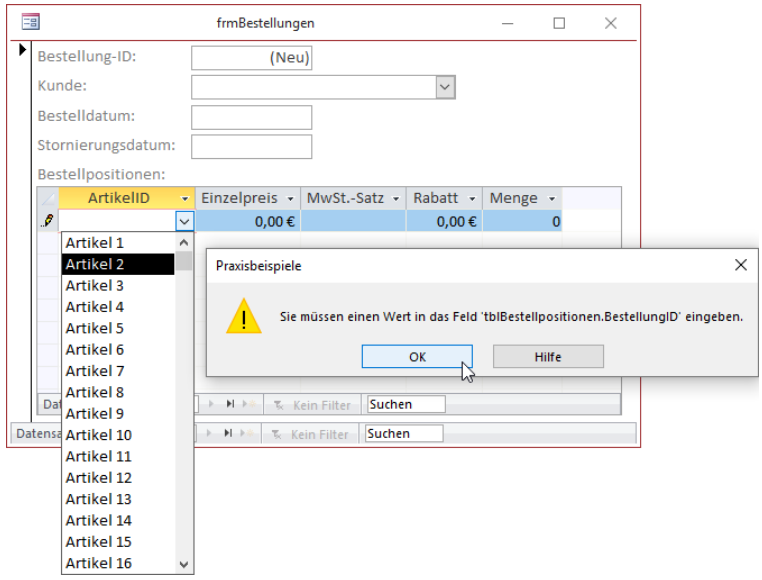


Abbildung 17.12: Fehler beim Anlegen eines Datensatzes im Unterformular ohne passenden Datensatz im Hauptformular

Hier können wir allerdings die gleiche Lösung verwenden, die wir bereits vorgestellt haben. Dazu fügen wir für das Unterformular-Steuerelement eine Ereignisprozedur für das Ereignis *Bei Aktivierung* hinzu:

```
Private Sub sfmBestellungen_Enter()
    If Me.NewRecord Then
        If Not Me.Dirty Then
            MsgBox "Bitte legen Sie zuerst eine Bestellung an."
            Me!KundeID.SetFocus
            Me!KundeID.Dropdown
```

```
            End If
        End If
    End Sub
```

In diesem Fall verschieben wir den Fokus nicht einfach auf ein Feld des Hauptformulars, sondern verwenden dazu das Kombinationsfeld zur Auswahl des Kunden und klappen dieses auch gleich auf.

17.4 Formular mehrfach öffnen

Wenn Sie eine Kundenübersicht verwenden, welche die Kunden in der Datenblattansicht oder in einem Listenfeld anzeigt, möchten Sie die Kundendaten vermutlich in einem Detailformular anzeigen. Manchmal wünscht man sich jedoch, die Datensätze gleich mehrerer Kunden gleichzeitig anzuzeigen. ir bauen in diesem Abschnitt auf dem Beispiel aus »Details per Doppelklick« ab Seite 206 auf, wo wir per Doppelklick auf einen Eintrag im Datenblatt im Unterformular das Detailformular mit den Daten des aktuell markierten Datensatzes angezeigt haben.

Dieses Beispiel wollen wir nun so anpassen, dass Sie mehrmals hintereinander doppelt auf einen Eintrag im Datenblatt im Unterformular klicken können und damit nicht nur immer wieder das gleiche Formular mit verschiedenen Datensätzen anzeigen, sondern jeweils ein neues Formular mit dem jeweiligen Datensatz anzeigen.

17.4.1 Kundendaten im Formulartitel anzeigen

Damit wir schneller erkennen können, zu welchem Kunden das jeweilige Detailformular gehört, schreiben wir in der Ereignisprozedur *Form_Current*, die durch das Ereignis *Beim Anzeigen* ausgelöst wird, die Werte der Felder *KundeID* und *Firma* in die Titelzeile des Formulars:

```
Private Sub Form_Current()
    Me.Caption = "Kunde " & Me!KundeID & " - " & Me!Firma
End Sub
```

Das Detailformular sieht dann wie in Abbildung 17.13 aus.

17.4.2 Formularinstanz erzeugen

Bisher haben wir in allen Beispielen die Methode *DoCmd.OpenForm* verwendet, um ein Formular zu öffnen. Das ist allerdings nicht die einzige Methode, um diese Aufgabe zu erledigen. Genau genommen ist *DoCmd.OpenForm* eine etwas untypische Möglichkeit, ein Objekt zu erstellen – wenn Sie sich einmal ansehen, wie Sie das sonst in objektorientierten Sprachen erledigen.

18 Programmiertricks

Dieses Kapitel zeigt verschiedene Programmiertricks, die Sie bei der Programmierung von Formularen einsetzen können.

18.1 Ereignis beim Schließen der Anwendung

Sie können Access zwar mitteilen, dass es beim Starten der Anwendung bestimmte Befehle ausführen soll. Das können Sie auf verschiedene Arten erledigen, die beste ist jedoch die folgende:

Sie verwenden das Makro *AutoExec*, das beim Starten ausgeführt wird, und geben dort als Parameter des Befehls *AusführenCode* den Namen der VBA-Funktion ein, die ausgeführt werden soll (siehe Abbildung 18.1).

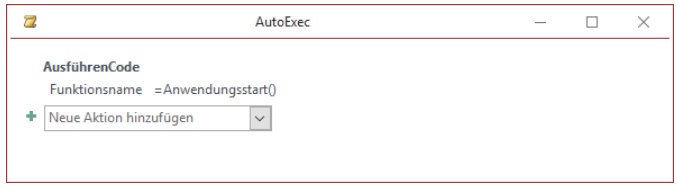


Abbildung 18.1: Das Makro *AutoExec* mit dem Aufruf einer VBA-Prozedur

In einem Standardmodul können Sie dann beispielsweise die folgende Funktion anlegen, um beim Anwendungsstart Befehle auszuführen – hier das Anzeigen einer Meldung:

```
Public Function Anwendungsstart()
    MsgBox "Anwendung gestartet"
End Function
```

Wie aber können wir das erledigen, wenn die Anwendung geschlossen wird? Es gibt schließlich nur ein Makro, das automatisch beim Anwendungsstart ausgeführt wird, aber keines, das Access beim Beenden der Anwendung aufruft.

Also behelfen wir uns mit einem Trick. Dabei verwenden wir ein Formular, das wir gleich beim Starten der Anwendung öffnen und das wir aber auch direkt ausblenden. Dazu nutzen wir in eben jener durch das oben angegebene *AutoExec*-Makro aufgerufenen Funktion den Parameter *acHidden* für den Parameter *WindowMode*:

```
Public Function Anwendungsstart()
    DoCmd.OpenForm "frmStart", WindowMode:=acHidden
End Function
```

Dieses neue Formular wird durch den Modus als ausgeblendetes Formular unbemerkt vom Benutzer versteckt. Wenn wir nun die Anwendung schließen, werden alle noch offenen Formulare geschlossen – unabhängig davon, ob sie eingeblendet oder ausgeblendet sind.

Das gilt auch für unser Formular, das demzufolge auch alle Ereignisse auslöst, die beim Schließen eines Formulars auf der Liste stehen, als auch das Ereignis *Beim Entladen*.

Für dieses hinterlegen wir nun die folgende Ereignisprozedur:

```
Private Sub Form_Unload(Cancel As Integer)
    Dim intMessage As VbMsgBoxResult
    intMessage = MsgBox("Die Anwendung wird beendet. Fortfahren?", vbYesNo)
    If intMessage = vbNo Then
        Cancel = True
    End If
End Sub
```

Damit erreichen wir, dass beim Beenden von Access auch versucht wird, das Formular *frmStart* zu schließen. Dies löst das Ereignis *Beim Entladen* aus, was die Anzeige einer Meldung wie in Abbildung 18.2 nach sich zieht.

Hier kann der Benutzer nun angeben, ob die Anwendung tatsächlich geschlossen werden soll. Wählt er die Schaltfläche *Nein* aus, setzt die Prozedur den Parameter *Cancel* auf den Wert *True*, was dazu führt, dass das Formular nicht geschlossen werden kann. Und dadurch kann dann auch die Anwendung nicht beendet werden und bleibt geöffnet.

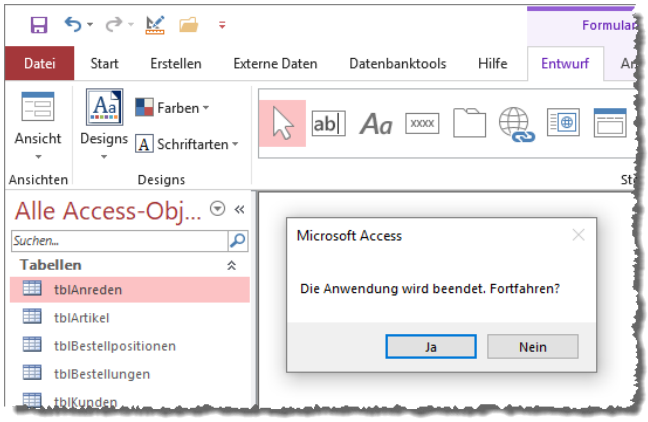


Abbildung 18.2: Meldung beim Versuch, die Anwendung zu schließen

18.2 Formular nur unter bestimmten Bedingungen öffnen

Sie können das Öffnen eines Formulars selbst nach einem Aufruf des Befehls *DoCmd.OpenForm* oder einem Doppelklick auf einen Formular-Eintrag in der Navigationsleiste von Access noch abbrechen. Das kann zum Beispiel sinnvoll sein, wenn Sie nicht wollen, dass ein Benutzer ein Formular einfach so per Doppelklick auf den Eintrag in der Navigationsleiste öffnet. Dies soll nur über eine bestimmte Schaltfläche eines speziellen Formulars möglich sein. Wie können wir das erreichen?

Dazu legen wir in dem zu öffnenden Formular eine Ereignisprozedur für das Ereignis *Beim Öffnen* an. In diesem prüfen wir, ob ein bestimmtes Öffnungsargument übergeben wurde. Ist das der Fall, soll das Formular geöffnet werden, anderenfalls soll der Vorgang unterbrochen werden und das Formular nicht erscheinen.

Unser Formular soll *frmNichtAusNavibereichOeffnen* heißen. Um zu prüfen, ob es einfach per Doppelklick auf den Eintrag im Navigationsbereich geöffnet wurde, legen wir die folgende Ereignisprozedur für das Ereignis *Beim Öffnen* des Formulars an:

```
Private Sub Form_Open(Cancel As Integer)
    If IsNull(Me.OpenArgs) Then
        MsgBox "Dieses Formular darf nicht vom Navigationsbereich aus geöffnet werden."
        Cancel = True
    End If
End Sub
```

Wenn Sie das Formular nun einfach per Doppelklick auf den Namen *frmNichtAusNavibereich-Oeffnen* öffnen wollen, erscheint die entsprechende Meldung und der Öffnen-Vorgang wird unterbrochen.

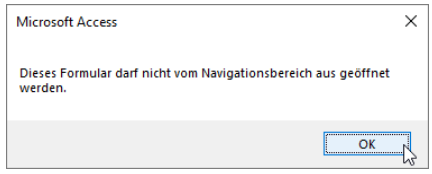


Abbildung 18.3: Meldung beim Versuch, ein Formular aus dem Navigationsbereich zu öffnen

Erst, wenn Sie es von einem anderen Formular aus öffnen und dieses einen Wert ungleich *Null* als Öffnungsargument übergibt, wird dieses Formular geöffnet. Der Aufruf dazu könnte etwa wie folgt aussehen:

```
DoCmd.OpenForm "frmNichtAusNavibereichOeffnen", OpenArgs:="Test"
```

18.3 Formularicon per VBA

Was per Eigenschaft nicht möglich ist, bekommen wir mit VBA und zwei API-Funktionen hin. Wir fügen jedem Formular ein eigenes Icon hinzu. Dieses muss in dieser einfachen Version als *.ico*-Datei im gleichen Verzeichnis wie die Datenbank liegen. Das Ergebnis sieht wie in Abbildung 18.4 aus.

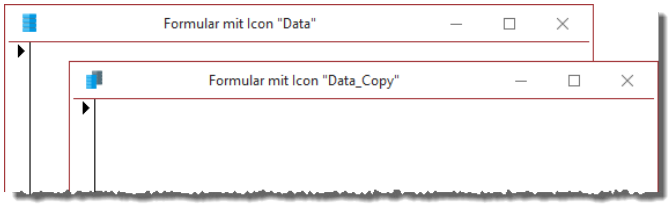


Abbildung 18.4: Formulare mit verschiedenen Icons

Um dies zu erreichen, legen Sie den folgenden Code in ein Standardmodul:

```
Private Declare Function LoadImageA Lib "user32" (ByVal hinst As Long, _
    ByVal lpszName As String, ByVal uType As Long, ByVal cxDesired As Long, _
    ByVal cyDesired As Long, ByVal fuLoad As Long) As Long
Private Declare Function SendMessageA Lib "user32" (ByVal hWnd As Long, _
    ByVal Msg As Long, ByVal wParam As Long, ByRef lParam As Any) As Long

Private Const IMAGE_ICON As Long = 1
Private Const LR_LOADFROMFILE As Long = &H10
Private Const WM_SETICON As Long = &H80
Private Const ICON_BIG As Long = 1

Public Sub SetFormIcon(hWnd As Long, IconPfad As String, Optional TaskIcon As Boolean)
    Dim hIcon As Long
    hIcon = LoadImageA(0, IconPfad, IMAGE_ICON, 0, 0, LR_LOADFROMFILE)
    If hIcon Then
        SendMessageA hWnd, WM_SETICON, ICON_BIG, ByVal hIcon
        If TaskIcon Then
            SendMessageA hWndAccessApp, WM_SETICON, ICON_BIG, ByVal hIcon
        End If
   End If
End Sub
```

Die Funktion *SetFormIcon* rufen Sie dann etwa im *Beim Öffnen*-Ereignis des Formulars auf. Dort geben Sie den Pfad zur Icon-Datei als zweiten Parameter an:

```
Private Sub Form_Open(Cancel As Integer)
    Call SetFormIcon(Me.hWnd, CurrentProject.Path & "\data.ico")
End Sub
```

Wenn Sie für den dritten Parameter den Wert *True* übergeben, wird das Icon auch noch als Icon der Access-Anwendung in der Taskleiste angezeigt:

```
Private Sub Form_Open(Cancel As Integer)
    Call SetFormIcon(Me.hWnd, CurrentProject.Path & "\data_copy.ico", True)
End Sub
```

18.4 Bilder zur Datenbank hinzufügen

Wenn Sie wie in »Bild hinzufügen« ab Seite 367 beschrieben Bilder zur Tabelle *MSysResources* der Datenbank hinzufügen wollen, müssen Sie jedes Bild einzelnen auswählen. Das ist nicht besonders benutzerfreundlich, daher erstellen wir nun ein Formular mit einer Schaltfläche und der Anzeige der bereits in der Tabelle *MSysResources* enthaltenen Bilder.

Einem neuen Formular namens *frmBilderEinfuegen* fügen wir als Datensatzquelle eine Abfrage hinzu, welche die Tabelle *MSysResources* als Datenquelle verwendet. Das Feld *Data* dieser Tabelle ist ein Anlagefeld, welches intern drei weitere Felder enthält. Davon fügen wir das Feld *Data.FileData* zum Abfrageentwurf hinzu – neben den beiden Feldern *ID* und *Name* (siehe Abbildung 18.5).

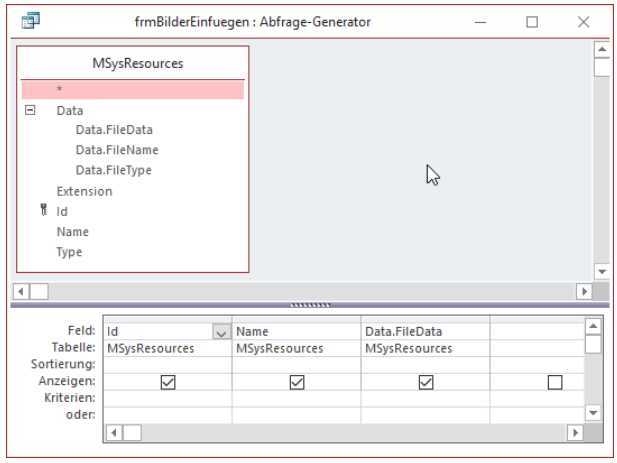


Abbildung 18.5: Datensatzquelle des Formulars

Die drei Felder dieser Abfrage fügen wir dann im Detailbereich des neuen Formulars ein. Außerdem legen wir im Formularkopf eine Schaltfläche namens *cmdBilderHinzufuegen* an (siehe

Abbildung 18.6). Für die Eigenschaft *Standardansicht* des Formulars legen wir den Wert *Endlosformular* fest.

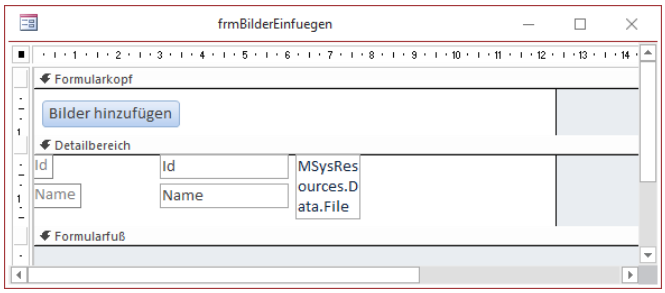


Abbildung 18.6: Entwurf des Formulars

Der wichtige Teil ist die Prozedur, die durch einen Klick auf diese Schaltfläche ausgelöst wird. Damit wollen wir einen *Datei auswählen*-Dialog öffnen, der die Auswahl der in der Tabelle *MSysResources* zu speichernden Bilder erlaubt. Die Prozedur sieht wie folgt aus:

```
Private Sub cmdBilderHinzufuegen_Click()
    Dim strStartDir As String
    Dim strPictures As String
    Dim strPicture As String
    Dim strPath As String
    Dim strFile As String
    Dim strName As String
    Dim strExtension As String
    Dim db As DAO.Database
    Dim lngPKID As Long
    Dim i As Integer
    Dim bolOverwrite As Boolean
    Dim rst As DAO.Recordset
    Dim rstAttachments As DAO.Recordset
```

Als Erstes füllen wir die Variable *db* mit einem Verweis auf das *Database*-Objekt für die aktuelle Datenbank. Dann legen wir als Startverzeichnis für den *Datei auswählen*-Dialog das aktuelle Datenbankverzeichnis fest:

```
    Set db = CurrentDb
    strStartDir = CurrentProject.Path
```

Die Liste der einzulesenden Bilder ermitteln wir durch den Aufruf eines *Datei auswählen*-Dialogs. Dies geschieht über eine VBA-Funktion namens *OpenFileNameMultiple*, die Sie im Modul *mdlTools* finden und die wir hier aus Platzgründen nicht im Detail beschreiben können. Die

Index

Symbole

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z