# MS ACCESS

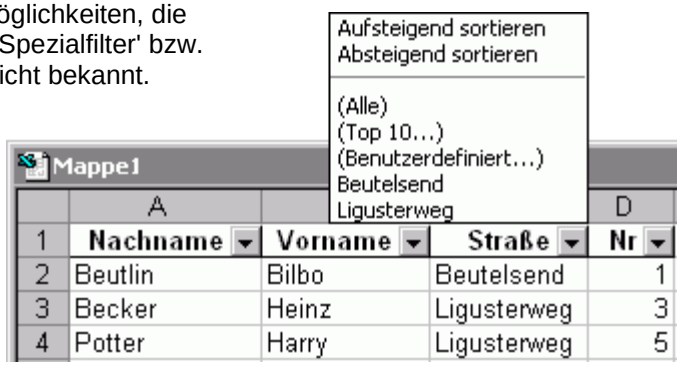
## Inhaltsverzeichnis

# Wofür sich Microsoft Access eignet und wofür nicht

Bevor man ein Projekt in ***Microsoft Access*** realisiert, sollte man prüfen, ob Access dafür geeignet ist.

## Mit Kanonen auf Spatzen schießen?

Vielleicht reicht ja eine Tabellenkalkulation. Die Möglichkeiten, die Microsoft Excel mit 'Autofilter' (siehe Bild rechts) ‚Spezialfilter' bzw. ‚Tabellen (Listen)' bietet, sind vielen Anwendern nicht bekannt.




| | A | | | D |
|---|---|---|---|---|
| 1 | Nachname | Vorname | Straße | Nr |
| 2 | Beutlin | Bilbo | Beutelsend | 1 |
| 3 | Becker | Heinz | Ligusterweg | 3 |
| 4 | Potter | Harry | Ligusterweg | 5 |

Wenn eine oder mehrere der folgenden Bedingungen zutreffen, ist eine Tabellenkalkulation allerdings nicht mehr zu empfehlen:

- Ein einzelnes Tabellenblatt genügt nicht (mehr), um den komplexen Sachverhalt zusammenhängend darzustellen. Bei der Verteilung der Daten auf mehrere Tabellenblätter sind Informationen aber nicht mehr vollständig darstellbar.
- Häufige Verwendung von Matrixfunktionen, z.B. `SVerweis()`.
- Die Tabelle ist sehr umfangreich und unübersichtlich geworden.
- Der Tabelle müssen regelmäßig (z.B. monatlich) neue Spalten angefügt werden.
- Es sind Tausende von Datensätzen (Zeilen) zu erwarten
- Die Anwendung soll vielen Anwendern zur Verfügung stehen. Die Anmeldeprozeduren von Excel sind dafür aber nicht ausreichend bzw. komfortabel genug.
- Für eine vernünftige Benutzerführung sollen komplexe Formulare eingesetzt werden.
- Die Anwendung benötigt Auswertungen, die auch 'stilistisch gut' ausgedruckt werden sollen.

Nach meiner -sehr persönlichen- Einschätzung ist Excel geeignet, solange man alle Daten eines Tabellenblattes auf dem Bildschirm sieht. Nur ‚Tabellen' (ehemals ‚Listen') sollte man nach unten, aber nicht zur Seite scrollen müssen.

## Den Missisippi mit einem Löschblatt trockenlegen?

Microsoft Access sind auch Grenzen gesetzt. Wer das erste Mal mit einer Datenbankanwendung zu tun hat, neigt unter Umständen dazu, Microsoft Access zu überschätzen. Umgekehrt wird Microsoft Access übrigens von professionellen Datenbankprogrammierern auch immer wieder *unter*schätzt. Sie übersehen zumeist, dass Access eine Lücke schließt, die zwischen Tabellenkalkulationen und ganz großen Datenbankanwendungen liegt.

Unter bestimmten Voraussetzungen sollte Access bestenfalls noch als Frontend eingesetzt werden, z.B.:

- Normalerweise ist stets mehr als eine bestimmte Anzahl Benutzer *gleichzeitig* angemeldet. Je komplexer die Anwendung und je mehr Netzwerktraffic, desto kleiner sollte die Anzahl gleichzeitiger Zugriffe sein; auch eine allereinfachste Anwendung sollte nicht mehr als ca. 50 gleichzeitige Benutzer haben.
- Es ist sehr wahrscheinlich, dass mindestens eine der Tabellen größer als 2GB wird.

Auch dann kann Access aber durchaus noch als Frontend, also Benutzeroberfläche, sinnvoll sein.

## Ins Schwarze getroffen

Die folgenden Punkte geben Hinweise, wann der Einsatz von Microsoft Access ideal ist:

- Microsoft Access ist beim Benutzer schon vorhanden.
- Die Anwendung soll auf einem einzelnen Rechner oder einem sehr kleinen Netzwerk laufen.
- Es werden nur einige (hundert)tausend Datensätze anfallen.

# Namenskonventionen

Beim Erstellen einer Datenbank legt man zumeist viele Objekte an, die alle benannt werden müssen. Dabei sollte man einfache, 'sprechende' Namen finden.
*Ein Name sollte nicht ÄÖÜß und keine Leerzeichen enthalten (sonst muss man ihn fast überall in [eckigen Klammern] schreiben). Mehrere Worte schreibt man einfach 'am Stück', mit Großbuchstaben als optische Trennung (z.B.* `KundeTelefonNr`*).*

Oft muss man rund um eine Sache mehrere Dinge benennen.
Dann ist ein einheitliches Namensschema praktisch, so dass man Zusammengehöriges erkennt, aber auch eine eindeutige Unterscheidbarkeit gegeben ist. Eine Namenskonvention hilft auch, eine Anwendung zu verstehen, die man nicht selbst geschrieben hat.

Hierzu hat sich in der Praxis ein System aus Präfixen etabliert:
**Die ungarische Notation.**

Natürlich ist dieses System nicht verbindlich und wird in der Praxis oft nur teilweise umgesetzt. Es spricht auch nichts gegen ein eigenes Namenssystem. Dieses Tutorial bezieht sich stets auf die folgende Systematik. Während der Lektüre der nachfolgenden Seiten sollte man immer mal wieder hier nachschauen:

**Formulare: Objekte**

| Objekt | Präfix | Beispiel |
|---|---|---|
| **Bezeichnungsfeld** | lbl | lblName |
| **Textfeld** | txt | txtName |
| **Optionsfeld** | opt | optAuswahlAlle |
| **Kontrollkästchen** | chk | chkBearbeitet |
| **Kombinationsfeld** | cbo | cboAuswahl |
| **Listenfeld** | lst | lstAuswahl |
| **Befehlsschaltfläche** | cmd | cmdOK |

**Datenbankobjekte**

| Objekt | Präfix | Beispiel |
|---|---|---|
| **Tabellen** | tab | tabKunden |
| **Abfragen** | qry | qryBestelldetails |
| **Formulare** | frm | frmKunden |
| **Berichte** | rpt | rptKatalog |
| **Module** | mod | modFunktionen |
| **Klassenmodule** | cls | clsDateiDialog |

**Tabellen: Feldnamen**

| Objekt | Präfix | Beispiel |
|---|---|---|
| **AutoWert** | ID | IDObjekt |
| **Boolean** | bln | blnEingabeOK |
| **Byte** | byt | bytAbteilung |
| **Integer** | int | intLagerbestand |
| **Long Int.** | lng | lngKunde |
| **Double** | dbl | dblKurs |
| **Dezimal** | dec | decRabatt |
| **Datum/Zeit** | dat | datBestellung |
| **Text** | txt | txtNachname |
| **Währung** | cur | curPreis |

| Tabellen: Feldnamen | | |
|---|---|---|
| **Objekt** | **Präfix** | **Beispiel** |
| **AutoWert** | ID | IDObjekt |
| **Boolean** | bln | blnEingabeOK |
| **Byte** | byt | bytAbteilung |
| **Integer** | int | intLagerbestand |
| **Long Int.** | lng | lngKunde |
| **Double** | dbl | dblKurs |
| **Dezimal** | dec | decRabatt |
| **Datum/Zeit** | dat | datBestellung |
| **Text** | txt | txtNachname |
| **Währung** | cur | curPreis |

| Datenbankobjekte | | |
|---|---|---|
| **Objekt** | **Präfix** | **Beispiel** |
| **Tabellen** | tab | tabKunden |
| **Abfragen** | qry | qryBestelldetails |
| **Formulare** | frm | frmKunden |
| **Berichte** | rpt | rptKatalog |
| **Module** | mod | modFunktionen |
| **Klassenmodule** | cls | clsDateiDialog |

| Formulare: Obj |
|---|
| **Objekt** |
| **Bezeichnungsf** |
| **Textfeld** |
| **Optionsfeld** |
| **Kontrollkästch** |
| **Kombinationsf** |
| **Listenfeld** |
| **Befehlsschaltfl** |

# Tabellen

In den Tabellen einer Datenbank werden Daten gespeichert. Nur die 'Rohdaten', keine Berechnungen oder Formatierungen. Außerdem bekommt nur der Programmierer, nie der Benutzer die Tabellen zu sehen.

Im Gegensatz zu einer Tabellenkalkulation besteht eine Datenbank üblicherweise nicht aus nur einer oder sehr wenigen Tabellenblättern, sondern es fallen meistens ziemlich viele Tabellen an, zwischen denen Beziehungen bestehen. Das ist keineswegs 'unnötig' oder 'eigentlich doch überflüssig'.
Im Gegenteil: Gerade die Verteilung der Daten auf mehrere Tabellen stellt die eigentliche Stärke einer Datenbank dar! **So soll, vereinfacht gesagt, verhindert werden, dass Informationen mehrfach abgespeichert werden ( keine *redundante Daten*).**

Wofür jeweils Tabellen bzw. Felder gebraucht werden, legt die Datenbanktheorie fest, auf die ebenfalls eingegangen wird. Dabei handelt es sich um die ***Normalisierung der Datenbank***, und eine gute Datenbank entspricht den ***Normalformen***. In der Praxis ist es sinnvoll, die Normalisierung***vorab***, am besten auf Papier(!) zu erarbeiten. Solange das nicht geschehen ist, wird man keine gute Datenbankanwendung zustande bringen.

Merke: ***Eine gute Datenbank entsteht zuerst im Kopf!***

## Aufbau einer Tabelle

Eine Datenbanktabelle besteht aus ***Feldern*** und ***Datensätzen***(nicht etwa aus 'Spalten' und 'Zeilen'). Bevor in einer Tabelle der erste Datensatz eingegeben werden kann, müssen erst die Tabellenfelder definiert werden.

Wie alle Datenbankobjekte, werden neue Tabellen mit  ***Neu*** angelegt. Das machen wir in der ***Entwurfsansicht***. Anfängern helfen die verschiedenen Assistenten nämlich nicht, die Funktionsweise einer Datenbank zu erlernen. Auch, dass Microsoft Access 2007 eine fertige Tabelle vorgaukelt, in die man einfach hineinschreiben kann, ist nicht gerade hilfreich. Bei existierenden Objekten kommt man über  ***Entwurf*** in die Entwurfsansicht und mit  ***Öffnen*** in die Datenblattansicht, wo man Daten eingeben könnte.

| **Tabelle1 : Tabelle** | | |
|---|---|---|
| | **Feldname** | **Felddatentyp** |
| ■ | txtNachname | Text |
| | | |

In der Entwurfsansicht legt man neue Felder an, indem man in der Spalte 'Feldname' einen Namen einträgt und einen ***Felddatentyp*** auswählt. In Felder eines bestimmten Felddatentyps können nur Werte diesen Typs eingetragen werden. Hier die wichtigsten Felddatentypen im Überblick:

| Felddatentypen | | |
|---|---|---|
| **Typ** | **Größe** | **Erläuterung** |
| Text | max 255 Byte | für Fließtexte |
| Zahl | siehe unten | speichert Zahlen. Dazu sind Einstellungen der Feldgröße nötig, die unten erläutert werden |
| Autowert | 4 Byte | füllt das Feld automatisch mit Ganzzahlen |
| Währung | 8 Byte | für Geldbeträge |
| Ja/Nein | 1 Bit | wird auch 'Boolean' genannt |
| Datum/Uhrzeit | 8 Byte | Dabei wird intern 'Datum' als Ganzzahl, 'Uhrzeit' als Nachkommastelle gespeichert |

| Tabelle1 : Tabelle |
|---|
| txtNachname |
| |

Jetzt könnte man die Tabelle auch schon speichern, in die***Datenblattansicht*** wechseln und Daten erfassen. Üblicherweise nimmt man aber für jedes Feld noch im unteren Teil des Entwurfsfensters weitere Einstellungen vor. Zum Beispiel kann man bestimmen, dass ein Feld zwingend gefüllt wird, oder der Inhalt bestimmte Anforderungen erfüllen muss.

Für den Felddatentyp ***Zahl*** ist dabei die ***Feldgröße*** besonders wichtig: Zahlen können natürliche Zahlen sein oder Nachkommastellen haben. Da eine Datenbank große Datenmengen mit möglichst wenig Speicherplatz und möglichst schnell verwalten soll, muss der Programmierer entscheiden, wie Zahlen gespeichert werden. Hier die wichtigsten Feldgrößen im Überblick:

| Feldgrößen für Zahlenfelder | | |
|---|---|---|
| **Typ** | **Größe** | **Erläuterung** |
| Byte | 1 Byte | Ganzzahlen von 0 bis 255 |
| Long Integer | 4 Byte | Ganzzahlen von -2.147.483.648 bis 2.147.483.647 |
| Double | 8 Byte | Fließkommazahlen |
| Dezimal | 12 Byte | Fließkommazahlen mit bis zu 28 Nachkommastellen mit hoher Genauigkeit |

# Zusammenspiel mehrerer Tabellen

Stellen wir uns vor, wir wollen in einer Datenbank die Namen aller Orte in Deutschland speichern. Dabei interessiert uns auch, in welchem Bundesland ein Ort liegt. Würden wir nun in einer Tabelle ein Feld `txtOrt` und ein Feld`txtLand` anlegen, wäre es nur eine Frage der Zeit, bis verschiedene Schreibweisen ein und desselben Bundeslands in der Tabelle auftauchen.

Statt dessen erstellen wir eine Tabelle namens `tabOrte` und eine weitere Tabelle `tabLaender`.

Hier die Tabelle `tabLaender`:

**tabLaender**: Tabelle

| IDLand | txtLand |
|---|---|
| 1 | Baden-Württemberg |
| 2 | Bayern |
| 3 | Berlin |
| … | … |
| 13 | Sachsen |
| … | … |
| 16 | Thüringen |

In **tabOrte** tragen wir die Orte und die Nummer des Bundeslandes ein.

**tabOrte** : Tabelle

| IDOrt | txtOrt | lngLand |
|---|---|---|
| 1 | München | 2 |
| 2 | Bamberg | 2 |
| 3 | Berlin | 3 |
| 4 | Stuttgart | 1 |
| 5 | Bayreuth | 2 |
| 6 | Dresden | 13 |

Dem aufmerksamen Leser wird auffallen, dass die Tabelle `tabLaender`ein Feld `IDLand` enthält, `tabOrte` dagegen ein Feld`lngLand`. Das Nummerieren überlassen wir nämlich einfach der Datenbank:`IDLand` ist vom Typ Autowert. Trägt man im Feld `txtLand`einen neuen Datensatz ein, wird `IDLand` automatisch gefüllt. In der Tabelle `tabOrte` muss dann das Feld, in dem das Bundesland gespeichert wird, den gleichen Zahlenumfang speichern können. Autowert entspricht insofern dem Typ Long Integer.

Man mag sich fragen, ob man bei der Datenerfassung immer nach der richtigen Nummer des Bundeslands suchen muss. Keine Panik!
Später werden wir noch sehen, dass nur der Programmierer, aber nicht der Anwender Tabellen zu Gesicht bekommt. Die Nummer sucht Microsoft Access (in einem Formular) automatisch.

Je nach Zweck einer Datenbank können noch viele Tabellen hinzukommen. In unserem Beispiel könnte man z.B. noch Tabellen benötigen, um Straßen zu erfassen. Oder man will nicht nur deutsche Orte abbilden; dann bräuchte man noch eine Tabelle für die Nationen.

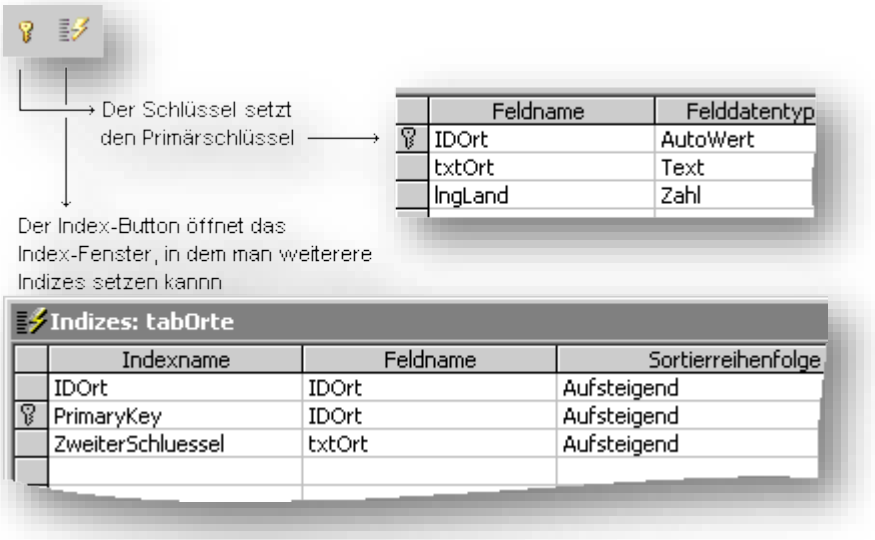
Würden wir weitere Angaben zum Bundesland benötigen (z.B. die Landeshauptstadt) würde das ebenfalls in `tabLaender` gespeichert.***Jede Tabelle ist also eine Informationssammlung gleichartiger Objekte***, und jedes Feld speichert eine Eigenschaft des Objekts.

## Schlüssel

Schauen wir uns die Tabelle `tabOrte` noch genauer an: Wäre`IDOrt` nicht vom Typ Autowert, sondern Zahl, könnte die gleiche Zahl mehrfach vorkommen! Und in `txtOrt` kann man mehrfach die gleiche Stadt eintragen.

Hier kommen ***Schlüssel*** oder auch ***Indizes*** ins Spiel: Ein indiziertes Feld wird schneller durchsucht und sortiert als ein nicht indiziertes Feld. Zudem lässt sich einstellen, ob in einem Index Duplikate erlaubt sind. Wir brauchen also je einen Index ohne Duplikate für die Felder `IDOrt` und`txtOrt`. Jetzt sind Duplikate ausgeschlossen! Will man jetzt z.B. mehrfach 'Frankfurt' eingeben, muss man zwischen 'Frankfurt am Main' und 'Frankfurt an der Oder' unterscheiden.

Jetzt noch einmal zu `tabLaender`: `IDLand` enthält Werte, die wir auch in der Tabelle `tabOrte` brauchen. Daher ist dies kein normaler Index, sondern der ***Primärschlüssel*** der Tabelle. Jede Tabelle sollte einen Primärschlüssel haben. Er bezeichnet eindeutig einen bestimmten Datensatz. Deswegen gibt es auch das Feld `IDOrt` in der Tabelle `tabOrte`: Wir brauchen es in unserem Beispiel zwar nicht, aber mit dem Primärschlüssel wird unsere Datenbank insgesamt schneller.

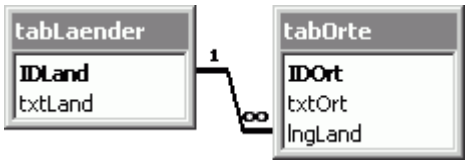



Es sind auch ***zusammengesetzte Schlüssel*** möglich, also Schlüssel, die sich über mehrere Felder erstrecken. Dazu wird ein Index auf das erste Feld erstellt, und in der nächsten Zeile im Index-Fenster in der Spalte***Feldname*** das zweite Feld ausgewählt, ohne die Spalte mit dem Indexnamen auszufüllen.

# Beziehungen

Beim Beispiel mit den Orten und Bundesländernsollen im Feld `lngLand` in `tabOrte` ausschließlich Werte aus `tabLaender` stehen. Andere Werte wären inhaltlich sinnlos - sogenannte ***Dateninkonsistenzen***. Microsoft Access kann das verhindern.

Dafür gibt es ***Beziehungen***. Eventuell muss man im sich öffnenden Fenster ***Tabellen anzeigen*** wählen, um aus einer Liste aller Tabellen `tabOrte` und `tabLaender`auswählen zu können und so im bisher leeren Beziehungsfenster einzufügen. Dort sieht man dann die beiden Tabellen; die Primärschlüssel sind **fett**dargestellt. Nun ziehen wir aus `tabLaender` das Feld`IDLand` auf `lngLand` in `tabOrte`. Damit wird zwischen diesen Feldern eine Beziehung hergestellt:




Man spricht von einer ***1:n-Beziehung***, oder auch 1:∞ (eins zu unendlich)-Beziehung. Nun darf ein Wert auf der 1−Seite der Beziehung nur ein einziges Mal vorkommen, auf der n−Seite dagegen beliebig oft.

Für Beziehungen gibt es weitere Einstellungen.Besonders wichtig ist die ***referentielle Integrität***, die auf der n−Seite nur Werte erlaubt, die es auch auf der 1−Seite gibt. Inkonsistente[1] Daten sind nicht mehr möglich.

Zur referentiellen Integrität[2] kann man auch einstellen, was passiert, wenn man auf der 1−Seite Daten ändert oder löscht: Löscht man z.B. bei erlaubter***Löschweitergabe*** einen Datensatz auf der 1−Seite, werden auch alle auf der n−Seite zugehörigen Datensätze gelöscht. Würde man in unserem Beispiel also ein Bundesland löschen, würden zugleich alle zugehörigen Orte gelöscht. Ohne erlaubte Löschweitergabe kann ein Datensatz auf der 1−Seite nur gelöscht werden, wenn es auf der n−Seite keinen zugehörigen Datensatz (mehr) gibt.

---

1 Inkonsistenz bedeutet insbesondere Widersprüchlichkeit oder Unbeständigkeit (Zusammenhanglosigkeit)

2 Die referentielle Integrität – oft auch als RI abgekürzt – ist eine Form der Datenintegrität.

## n:m-Beziehung

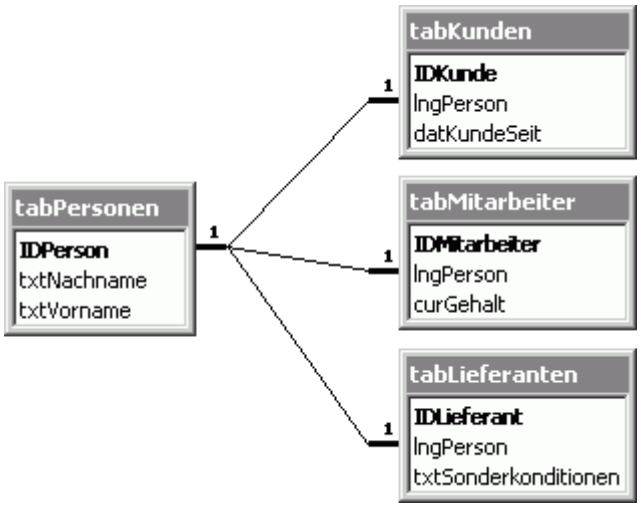
Wir haben gerade gelernt, wie Beziehungen zwischen zwei Tabellen aussehen, wenn auf der einen Seite der Beziehung nur ein, auf der anderen Seite dagegen beliebig viele Datensätze stehen. Es gibt aber auch den Fall, dass es auf beiden Seiten beliebig viele Datensätze gibt. Dies nennt man ***n:m−Beziehung***. Ein einfaches Beispiel:

Ein Unternehmen hat Kunden, die Produkte bestellen. Ein Kunde kann mehrere Produkte bestellen, und ein Produkt kann von mehreren Kunden bestellt worden sein.

Dafür benötigt man neben den Tabellen `tabKunden` und`tabProdukte` noch eine 'Zwischentabelle', nennen wir sie`tabBestellungen`. Damit kann ein Kunde beliebig viele Bestellungen haben, und ebenso kann ein Produkt beliebig oft bestellt worden sein. Die n:m−Beziehung ist somit in zwei 1:n−Beziehungen aufgelöst:


## 1:1-Beziehung

Nehmen wir an, ein Unternehmen hat eine Tabelle mit Adressen aller Personen, mit denen es zu tun hat. Eine Person kann Kunde, Mitarbeiter oder Lieferant sein - oder mehreres davon.




Hier liegen drei 1:1-Beziehungen vor.

Für 1:1-Beziehungen müssen die zu verknüpfenden Felder auf beiden Seiten einen eindeutigen Schlüssel haben.`lngPerson` braucht also in allen Tabellen, in denen es vorkommt, einen Index ohne Duplikate.

Man sieht hier, warum man Personendaten separat speichern sollte: Das Gehalt der Kollegen oder die Sonderkonditionen bestimmter Lieferanten sollten vielleicht nicht allen Mitarbeiter zugänglich sein. Wären umgekehrt die Adressen jeweils in den drei anderen Tabellen gespeichert, hätte man mehrfach gespeicherte (redundante) Daten und die Gefahr unterschiedlicher (inkonsistenter) Angaben. Man könnte auch nur schwer prüfen, ob z.B. ein Mitarbeiter auch Lieferant ist.

# Normalformen

Bisher haben wir uns mit dem 'Handwerkszeug' einer relationalen Datenbank beschäftigt. Jetzt lernen wir etwas Datenbanktheorie, die bisher zu kurz gekommen ist. Das ***Normalisieren einer Datenbank*** verhindert doppelte (redundante) Daten. Eine Datenbank soll gemäß folgender sog. 'Normalformen' Schritt für Schritt geplant werden. Die Datenbanktheorie kennt noch weitere Normalformen, aber in der Praxis kommen Datenstrukturen, die eine weitergehende Normalisierung erforderlich machen, weniger häufig vor.

## Nullte Normalform

Eine Tabelle soll keine Felder enthalten, die aus anderen Feldern berechnet werden können (Dafür sind Abfragen zuständig).

Sehen wir uns folgende Tabelle an:

| **Verkaeufe** | | | | |
|---|---|---|---|---|
| **VerkaufsNr** | **Verkaufsdatum** | **Nettopreis** | **MwSt** | **Bruttopreis** |
| 1 | 19.12.2007 | 100 | 19% | 119 |
| 2 | 05.12.2011 | 50 | 12% | 60 |

Während der erste Datensatz in sich stimmig scheint, ist der Mehrwertsteuersatz des zweiten Datensatzes definitiv falsch. Und egal, wie man rechnet, der Bruttopreis kann auch nicht stimmen.

MwSt und Bruttopreis gehören auch gar nicht erst in die Tabelle, denn anhand des Datums kann jederzeit der Mehrwertsteuersatz und damit auch der Bruttopreis errechnet werden.

## Erste Normalform

Die erste Normalform bestimmt, wofür jeweils ein Feld benötigt wird: Eine Tabelle befindet sich in der ersten Normalform, wenn die Werte in jedem Feld und in jedem Datensatz ***atomar*** sind, d.h. sich nicht mehr in kleinere Einheiten zerlegen lassen.

Hier ein Negativbeispiel:

| **Personen** | | |
|---|---|---|
| **IDKunde** | **Name** | **Anschrift** |
| 1 | Bilbo Beutlin | Beutelsend 1; 12345 Hobbingen |
| 2 | Petra Roth | Berliner Str. 5a), 60123 Frankfurt am Main |

Name und Anschrift lassen sich noch weiter zerlegen, nämlich in Vorname, Nachname, Straße, HausNr, PLZ und Ort. Wollte man aus dieser Tabelle alle Personen mit einer bestimmten Postleitzahl abfragen, wäre man vor eine unlösbare Aufgabe gestellt: Mal befindet sich vor der Postleitzahl ein Semikolon, mal ein Punkt, und den Benutzern würden bestimmt noch ein paar andere lustige Trennzeichen einfallen - falls sie überhaupt an eine PLZ denken...

## Zweite Normalform

Eine Tabelle befindet sich in der zweiten Normalform, wenn sie der ersten Normalform entspricht und darüber hinaus jeder Datensatz nur Felder enthält, die sich auf das Objekt beziehen, das durch den ***Primärschlüssel*** dargestellt wird.

In der folgenden Beispieltabelle gibt es zwar ein Primärschlüsselfeld (IDKunde), das aber keinen Bezug zu den Autokennzeichen hat.

| **Werkstattkunden** | | | |
|---|---|---|---|
| **IDKunde** | **Nachname** | **Auto-KZ1** | **Auto-KZ2** |
| 1 | Junker | F J-2000 | F J-2001 |
| 2 | Becker | SB AF-123 | |

Bei den Kunden haben die Auto-KZ nichts zu suchen. Gemäß der zweiten Normalform müssen sie aus obiger Tabelle entfernt werden. Sie gehören in eine eigene Tabelle. Dann können auch mehr als zwei Kennzeichen pro Kunde dargestellt werden.`Auto-KZ` ist dann das Primärschlüsselfeld der neuen Tabelle.

| **PKWs** | |
|---|---|
| **IDKunde** | **Auto-KZ** |
| 1 | F J-2000 |
| 1 | F J-2001 |
| 2 | SB AF-123 |

## Dritte Normalform

Eine Tabelle befindet sich in der dritten Normalform, wenn sie der zweiten Normalform entspricht und darüber hinaus alle Felder nur einmal vorkommen und alle Felder, die nicht den Primärschlüssel bilden, voneinander ***unabhängig*** sind.

Auch hier wieder ein Beispiel, wie es nicht sein sollte:

| **Orte** | | | |
|---|---|---|---|
| **IDOrt** | **txtOrt** | **lngLand** | **txtLand** |
| 1 | München | 2 | Bayern |
| 2 | Bamberg | 2 | Baiern |
| 3 | Berlin | 3 | Berlin |

Hier wurde nicht nur die Nummer des Bundeslandes, sondern auch noch der Klartext dazu abgelegt. `txtLand` ist aber von `lngLand`abhängig, also wurde gegen die dritte Normalform verstoßen. So konnte sich auch der Tippfehler 'Baiern' einschleichen. Mehr als `lngLand` hätte nicht in diese Tabelle gehört.

# Tipps & Tricks zu Tabellen

Zu Tabellen noch ein paar allgemeine, unzusammenhängende Tipps und Tricks. Einiges davon geht ein wenig über das Tutorial hinaus.

Üblicherweise sollte eine Microsoft-Access-Datenbank aus zwei Dateien bestehen:
Einer ***Backend-Datei*** mit den Tabellen und Beziehungen,
und einer***Frontend-Datei*** mit den übrigen Objekten.
Wenn man die Anwendung später einmal weiterentwickelt, betrifft das selten das Backend. Ist eine neue Version fertig, wird nur die Frontend-Datei ersetzt; die Daten stehen weiter zur Verfügung. In einer Mehrbenutzerumgebung bekommt jeder Benutzer eine Kopie des Frontends, dadurch wird die Anwendung schneller.

***Autowert*** ist für interne Zwecke nützlich, aber selten geeignet, um ihn dem Benutzer zu zeigen. Wenn man nämlich das Anlegen eines neuen Datensatzes abbricht, so wurde dafür schon eine Nummer angelegt, die nicht ohne weiteres wieder verwendet werden kann. Es kann also zu Lücken in der Nummerierung kommen.

Im Datentyp ***OLE-Objekt*** kann Microsoft Access auch beliebige Dateien, wie z.B. Bilder, in einer Tabelle speichern. Das bläht aber die Datenbank enorm auf und ist nicht allzu perfomant. Statt dessen sollte man die Dateien in einem Unterverzeichnis speichern und in der Tabelle nur ein Textfeld mit dem Speicherort anlegen.

Der Datentyp ***Nachschlagefeld*** ist überflüssig, entspricht nicht derDatenbanklehre und sollte niemals verwendet werden.

Tabellenfelder sollte man ***nicht formatieren***. Die Formatierung erfolgt erst später in den Formularen und Berichten. Gerade bei Werten, mit denen später noch gerechnet werden soll, kann es sonst nämlich beim Betrachten der Werte zu Verwirrung kommen.

# Abfragen

Während in Tabellen die Daten gespeichert werden, sind Abfragen für die Darstellung und Auswertung der Daten zuständig.

Das Ergebnis einer Abfrage sieht wie eine Tabelle aus. Aber ***eine Abfrage speichert keine Daten***, sondern bezieht ihre Daten stets aus einer oder mehreren Tabellen. Trotzdem kann man meistens in einer Abfrage Daten eingeben und ändern. Das wird dann in der zugrundeliegenden Tabelle gespeichert. Wir werden im folgenden auch Abfragen kennenlernen, bei denen Access nicht wissen kann, wohin genau die Daten gespeichert werden sollen. Dann ist die Dateneingabe automatisch gesperrt.

Datenbanken nutzen für Abfragen eine eigene Sprache namens ***SQL***(Structured Query Language). Mit ***Microsoft Access*** kann man Abfragen in der 'Entwurfsansicht' komfortabel grafisch generieren, ohne SQL lernen zu müssen. Natürlich kann man, wenn man mit SQL vertraut ist, direkt damit arbeiten, oder beide Arbeitsweisen mischen.

SQL-Code
Anzeigefenster

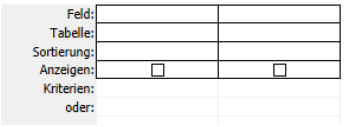
Im folgenden lernen wir, Abfragen in der Entwurfsansicht zu erstellen. Informativ wird auch der SQL-Code dargestellt - allerdings in einer vereinfachten und leichter verständlichen Form. Microsoft Access muss den Code ja automatisch generieren und legt daher im Zweifel lieber etwas zuviel Text an. Durch dieses 'Mehr' an Text wird die Abfrage aber weder besser noch schlechter, schneller oder langsamer als die hier verwendete Form.

## Einfache Abfragen

Wie alle Datenbankobjekte, werden in Microsoft Access auch neue Abfragen mit***Neu*** angelegt. Das sollte auch hier wieder über die ***Entwurfsansicht*** passieren. Einem Anfänger helfen auch hier die Assistenten nicht, die Funktionsweise einer Datenbank zu lernen.

Bei vorhandenen Abfragen kommt man mit***Entwurf*** wieder zur Entwurfsansicht und mit ***Öffnen***zur Datenblattansicht. Außerdem gibt es in einer geöffneten Abfrage einen Button**SQL**, mit dem man zur SQL-Ansicht gelangt. Beim Erstellen einer Abfrage öffnet sich das Fenster***Tabelle anzeigen***, mit dem man die als Datenquelle(n) benötigten Tabellen oder Abfragen hinzufügen kann. Dieses Fenster kann man natürlich auch nachträglich öffnen.

Im oberen Teil des Entwurfsfensters sieht man die ausgewählten Tabellen und Abfragen. Die Darstellung kennen wir schon vom Beziehungsfenster. Neu ist nur das Sternchen vor dem ersten Feldnamen. Im Folgenden beschäftigen wir uns mit der Tabelle `tabOrte`, die wir ja schon ausfrüheren Beispielen kennen.

Klickt man in die Zeile ***Feld***, öffnet sich eine Auswahlliste mit allen Feldnamen, die in in den Datenquellen vorhanden sind. Wir wählen jetzt einmal`txtOrt` aus (Man könnte auch aus der Feldliste im oberen Teil des Fensters einen Feldnamen herunterziehen).

| | | |
|---|---|---|
| Feld: | txtOrt | |
| Tabelle: | tabOrte | |
| Sortierung: | | |
| Anzeigen: | ☑ | ☐ |
| Kriterien: | | |
| oder: | | |

Das Feld ***Tabelle*** wurde automatisch gefüllt und in ***Anzeigen***automatisch ein Häkchen gesetzt. Schon haben wir eine vollständige Abfrage erstellt, die wir ausführen können (Im Menü/Ribbon ***Ansicht/Datenblattansicht***auswählen).

| **Abfrage1 : Auswahlabfrage** |
|---|
| **txtOrt** |
| München |
| Bamberg |
| Berlin |
| Stuttgart |
| Bayreuth |

```
SELECT txtOrt
FROM tabOrte;
```

Wie wir sehen, wird das Feld `txtOrt` aus der Tabelle`tabOrte` angezeigt, und zwar so, als wenn `tabOrte` nur aus diesem Feld bestehen würde.

Probieren wir einmal folgende Einstellung:

| | | |
|---|---|---|
| Feld: | txtOrte | IDOrt |
| Tabelle: | tabOrte | tabOrte |
| Sortierung: | | |
| Anzeigen: | ☑ | ☑ |
| Kriterien: | | |
| oder: | | |

```
SELECT txtOrt, IDOrt
FROM tabOrte;
```

**Abfrage1 : Auswahlabfrage**

| **txtOrt** | **IDOrt** |
|---|---|
| München | 1 |
| Bamberg | 2 |
| Berlin | 3 |
| Stuttgart | 4 |
| Bayreuth | 5 |

Die Reihenfolge der Spalten in der Abfrage ist also abhängig von der Reihenfolge, in der sie in der Entwurfsansicht angeordnet sind.

Zurück in der Entwurfsansicht, wählen wir jetzt in der Zeile ***Feld***einmal den Wert ***tabOrte.****.

| | | |
|---|---|---|
| Feld: | tabOrte.* | |
| Tabelle: | tabOrte | |
| Sortierung: | | |
| Anzeigen: | ☑ | ☐ |
| Kriterien: | | |
| oder: | | |

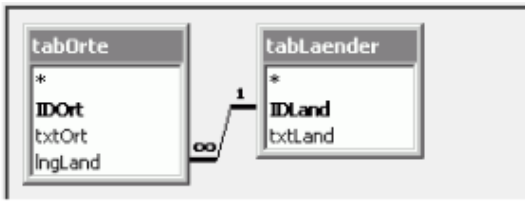
```
SELECT *
FROM tabOrte;
```

| Abfrage1 : Auswahlabfrage | | |
|---|---|---|
| **IDOrt** | **txtOrt** | **lngLand** |
| 1 | München | 2 |
| 2 | Bamberg | 2 |
| 3 | Berlin | 3 |
| 4 | Stuttgart | 1 |
| 5 | Bayreuth | 2 |

Das Sternchen, das ja auch im oberen Teil des Entwurfsfensters zu sehen ist, ist also einfach eine Abkürzung für 'alle Felder'.

## Quellenübergreifende Abfragen

Durch die Tabellennormalisierung können wir Daten effektiv speichern. Wie schon erwähnt, sind aber Daten in den Tabellen oft nicht gut lesbar. Daher erstellen wir jetzt eine Abfrage, die dieses Problem löst. Über ***Tabelle anzeigen***fügen wir die Tabellen `tabOrte` und `tabLaender` zu einer Abfrage hinzu. Wir kennen sie ja schon ausfrüheren Beispielen.




Wie man sieht, wird auch unsere Beziehung automatisch angezeigt. Im unteren Teil des Abfragefensters nehmen wir folgende Einstellung vor:

| | | |
|---|---|---|
| Feld: | txtOrt | txtLand |
| Tabelle: | tabOrte | tabLaender |
| Sortierung: | | |
| Anzeigen: | ☑ | ☑ |
| Kriterien: | | |
| oder: | | |

```
SELECT txtOrt, txtLand
FROM tabLaender INNER JOIN tabOrte ON
tabLaender.IDLand = tabOrte.lngLand;
```

| Abfrage1 : Auswahlabfrage | |
|---|---|
| **txtOrt** | **txtLand** |
| München | Bayern |
| Bamberg | Bayern |
| Berlin | Berlin |
| Stuttgart | Baden-Württemberg |
| Bayreuth | Bayern |

München Bayern

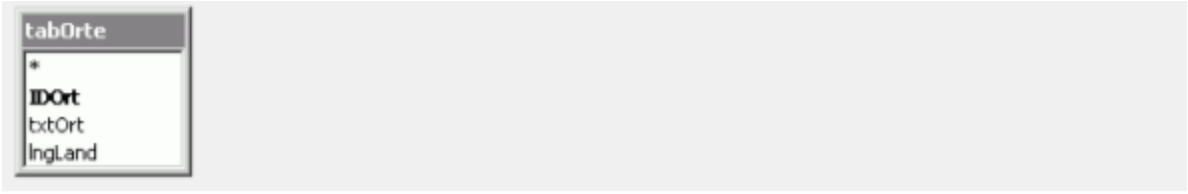
Bamberg Bayern

Wir haben also Daten aus verschiedenen Tabellen zusammengeführt.

## Sortieren und Anzeigen

Klickt man in die Zeile ***Sortierung***, bekommt man ***Aufsteigend, Absteigend*** oder ***(nicht sortiert)*** vorgeschlagen. Dies ist soweit selbsterklärend. Wenn die Abfrage nicht sortieren soll (was übrigens performanter ist), richtet sich die Sortierung nach denSchlüsseln der Tabelle.

Nimmt man das Häkchen ☑in der Zeile***Anzeigen*** heraus, wird das zugehörige Feld nicht in der Datenblattansicht ausgegeben. Folgendes Beispiel soll den Sinn verdeutlichen:




| Feld: | txtOrt | lngLand | txtOrt |
|---|---|---|---|
| Tabelle: | tabOrte | tabOrte | tabOrte |
| Sortierung: | | Aufsteigend | Aufsteigend |
| Anzeigen: | ☑ | ☑ | ☐ |
| Kriterien: | | | |
| oder: | | | |

Nun wird zuerst nach der Nummer des Landes, und innerhalb dessen nach dem Ortsnamen sortiert. Trotzdem wird zuerst der Ort angezeigt.

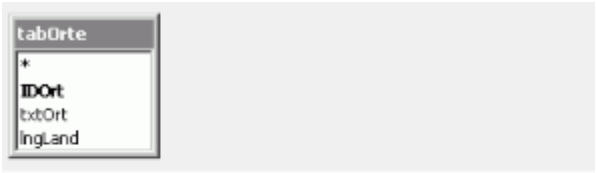
| **Abfrage1 : Auswahlabfrage** | |
|---|---|
| **txtOrt** | **lngLand** |
| Stuttgart | 1 |
| Bamberg | 2 |
| Bayreuth | 2 |
| München | 2 |
| Berlin | 3 |

```
SELECT txtOrt, lngLand
FROM tabOrte
ORDER BY lngLand, txtOrt;
```

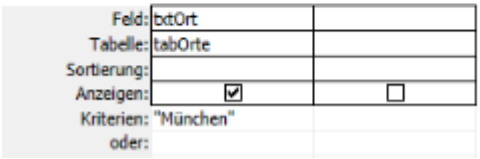
# Kriterien

## Ausdrücke

Eine der wichtigsten Fähigkeiten von Abfragen ist, Datensätze nach beliebigen Kriterien bzw. Bedingungen zu filtern. Hier ein einfaches Beispiel:



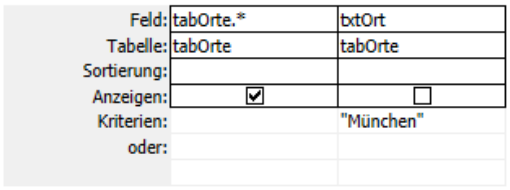

```
SELECT txtOrt
FROM tabOrte
WHERE txtOrt="München";
```




Die Abfrage gibt jetzt nur noch Datensätze aus, bei denen im Feld`txtOrt` 'München' eingetragen ist:

| **Abfrage1 : Auswahlabfrage** |
|---|
| **txtOrt** |
| München |

Auch im Zusammenhang mit Kriterien ist es manchmal sinnvoll, Felder nicht anzuzeigen:




```
SELECT *
FROM tabOrte
WHERE txtOrt="München";
```

## Operatoren

Es ist möglich, als Kriterium nicht nur einen festen Ausdruck, sondern auch ***Operatoren*** einzugeben:

| Feld: | tabOrte.* | lngLand |
|---|---|---|
| Tabelle: | tabOrte | tabOrte |
| Sortierung: | | |
| Anzeigen: | ☑ | ☐ |
| Kriterien: | | >2 |
| oder: | | |

```
SELECT *
FROM tabOrte
WHERE lngLand>2;
```

So werden alle Datensätze ausgegeben, bei denen im Feld `lngLand`ein Wert größer als 2 steht:

**Abfrage1 : Auswahlabfrage**

| IDOrt | txtOrt | lngLand |
|---|---|---|
| 3 | Berlin | 3 |

Hier eine Auflistung von Operatoren:

| **Operatoren** | | |
|---|---|---|
| **Operator** | **Beispiel** | **Bedeutung** |
| **=** | =1 | gleich |
| **<>** | <>1 | ungleich |
| **<** | <1 | kleiner als |
| **>** | >1 | größer als |
| **<=** | <=1 | kleiner oder gleich |
| **>=** | >=1 | größer oder gleich |
| **Zwischen ... und** | Zwischen 1 und 3 | Zwischen zwei Werten (jeweils einschließlich) |
| **Wie** | Wie "X*" | entspricht einem Textmuster |
| **In** | In ('Müller', 'Mayer', 'Schulze') | in einer Liste enthaltene Werte |

Darüber hinaus gibt es noch den ***Null-Operator***, mit dem man nach leeren Feldern suchen kann. Dies ist nicht zu verwechseln mit einer ***leeren Zeichenfolge*** oder dem Zahlenwert ***0***:
In einem Zahlenfeld kann entweder den Wert 0 stehen oder, wenn es in den Feldeinstellungen erlaubt ist, gar nichts.
Um nach einer 0 zu suchen, hieße das Kriterium ***=0***,
um nach einem leeren Feld zu suchen, würde es ***Ist Null*** heißen.
Eine leere Zeichenfolge hingegen sucht man mit ***""*** (ohne Leerzeichen dazwischen).

Operatoren funktionieren auch für Textfelder:

| Feld: | txtOrt | |
|---|---|---|
| Tabelle: | tabOrte | |
| Sortierung: | | |
| Anzeigen: | ☑ | ☐ |
| Kriterien: | >"C" | |
| oder: | | |

```
SELECT txtOrt
FROM tabOrte
WHERE txtOrt>"C";
```

Nun erhält man alle Ortsnamen, die alphabetisch nach "C" kommen:

**Abfrage1 : Auswahlabfrage**

| txtOrt |
|---|
| München |
| Stuttgart |

### *Der Wie-Operator*

Für Textfelder gibt es zusätzlich noch den ***Wie−Operator***. Mit ihm können ***Platzhalterzeichen*** verwendet werden, um alle Elemente zu finden, die einem Muster entsprechen:

| | | |
|---|---|---|
| Feld: | tabOrte.* | txtOrt |
| Tabelle: | tabOrte | tabOrte |
| Sortierung: | | |
| Anzeigen: | ☑ | ☐ |
| Kriterien: | | Wie "B*" |
| oder: | | |

```
SELECT *
FROM tabOrte
WHERE txtOrt Like "B*";
```

Die Abfrage gibt jetzt alle Datensätze aus, bei denen der Ortsname mit 'B' beginnt:

**Abfrage1 : Auswahlabfrage**

| IDOrt | txtOrt | lngLand |
|---|---|---|
| 2 | Bamberg | 2 |
| 5 | Bayreuth | 2 |
| 3 | Berlin | 3 |

Folgende Platzhalterzeichen sind möglich:

**Platzhalterzeichen**

| Symbol | Beispiel | Ergebnis | Verwendung |
|---|---|---|---|
| * | Wie "*er" | findet Maier, Müller, Junker | Entspricht einer beliebigen Anzahl Zeichen |
| ? | Wie "Ma?er" | findet Maier, Majer und Mayer | Entspricht einem beliebigen einzelnen Zeichen |
| # | Wie "1#3" | findet 103, 113, 123 | Entspricht einer beliebigen einzelnen Ziffer im Text |
| [ ] | Wie "Ma[iy]er" | findet Maier und Mayer, aber nicht Majer | Entspricht einem einzelnen Zeichen innerhalb der eckigen Klammern |
| ! | Wie "Ma[!iy]er" | findet Majer, aber nicht Maier oder Mayer | Entspricht einem einzelnen, beliebigen, nicht aufgelisteten Zeichen |
| - | Wie "b[a-c]d" | findet bad, bbd und bcd | Entspricht einem einzelnen, beliebigen Zeichen innerhalb des angegebenen Bereichs |

### *Der In-Operator*

Der oben aufgefürte ***in***-Operator ist eine Besonderheit: Man könnte mit***In ('Müller', 'Mayer', 'Schulze')*** nach Datensätzen suchen, die in einer Liste enthalten sind. So etwas ist normalerweise mit 'oder' sinnvoller (sh. mehrere Kriterien). Spannend wird es erst im Zusammenhang mit ***Unterabfragen***: In der folgenden Abfrage steckt im In-Operator selbst wieder SQL-Code. Sie gibt alle Orte aus, die in einem Bundesland liegen, in denen es auch einen Ort namens 'Neustadt' gibt - ohne, dass man wissen muss, in welchen Bundesländern ein solcher Ort existiert.

| | | |
|---|---|---|
| Feld: | txtOrt | lngLand |
| Tabelle: | tabOrte | tabOrte |
| Sortierung: | | |
| Anzeigen: | ☑ | ☑ |
| Kriterien: | | In (SELECT lngLand FROM tabOrte WHERE txtOrt = "Neustadt") |
| oder: | | |

Übrigens können Unterabfragen auch in der Zeile 'Feld' stehen: Um einen Feldnamen überhaupt erst per Abfrage zu ermitteln. Unterabfragen sind also recht mächtig -führen allerdings eigentlich für dieses Tutorial erst einmal zu weit...

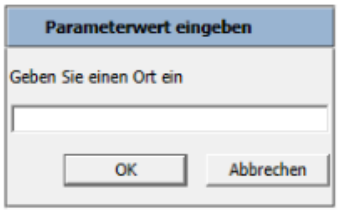
## Parameterabfragen

Es können auch manuell Kriterien, sogenannte ***Parameterwerte***, an eine Abfrage übergeben werden:

| | | |
|---|---|---|
| Feld: | tabOrte.* | txtOrt |
| Tabelle: | tabOrte | tabOrte |
| Sortierung: | | |
| Anzeigen: | ☑ | ☐ |
| Kriterien: | | [Geben Sie einen Ort ein] |
| oder: | | |

```
SELECT *
FROM tabOrte
WHERE txtOrt=[Geben Sie einen Ort ein];
```

Führt man diese Abfrage aus, erscheint zunächst eine Eingabeaufforderung ...




... anhand derer man einen Wert an die Abfrage übergeben kann.
*Im Allgemeinen sind Abfrageparameter auf diese Art nicht zu empfehlen. Interessant werden sie erst im Zusammenspiel mit Formularen, da so die Standardeingabeaufforderung durch ein individuelles Formular ersetzt wird:*

| | | |
|---|---|---|
| Feld: | tabOrte.* | txtOrt |
| Tabelle: | tabOrte | tabOrte |
| Sortierung: | | |
| Anzeigen: | ☑ | ☐ |
| Kriterien: | | [Forms]![Formularname]![Steuerelementname] |
| oder: | | |

## Mehrere Kriterien

Natürlich kann man nicht nur ein Kriterium eingeben, sondern beliebig viele, die dann mit ***und*** bzw. ***oder*** zueinander in Bezug stehen. Die folgende Einstellung gibt alle Ortsnamen , die mit 'S' oder 'M' beginnen:


| Feld: | tabOrte.* | txtOrt |
|---|---|---|
| Tabelle: | tabOrte | tabOrte |
| Sortierung: | | |
| Anzeigen: | ☑ | ☐ |
| Kriterien: | | "S*" |
| oder: | | "M*" |

```
SELECT txtOrt
FROM tabOrte
WHERE (txtOrt Like "S*") OR (txtOrt Like "M*");
```

**Abfrage1 : Auswahlabfrage**

| txtOrt |
|---|
| München |
| Stuttgart |

Jetzt die Kriterien für alle Datensätze, die mit 'B' beginnen und in Bayern liegen:

| Feld: | tabOrte.* | txtOrt | lngLand |
|---|---|---|---|
| Tabelle: | tabOrte | tabOrte | tabOrte |
| Sortierung: | | | |
| Anzeigen: | ☑ | ☐ | ☐ |
| Kriterien: | | "B*" | 2 |
| oder: | | | |

```
SELECT *
FROM tabOrte
WHERE (txtOrt Like "B*") AND (lngLand=2);
```

**Abfrage1 : Auswahlabfrage**

| IDOrt | txtOrt | lngLand |
|---|---|---|
| 2 | Bamberg | 2 |
| 5 | Bayreuth | 2 |

Es ist auch möglich, mit ***Nicht*** eine Bedingung zu verneinen:

| Feld: | tabOrte.* | txtOrt |
|---|---|---|
| Tabelle: | tabOrte | tabOrte |
| Sortierung: | | |
| Anzeigen: | ☑ | ☐ |
| Kriterien: | | Nicht "B*" |
| oder: | | |

```
SELECT *
FROM tabOrte
WHERE txtOrt Not Like "B*";
```

**Abfrage1 : Auswahlabfrage**

| IDOrt | txtOrt | lngLand |
|---|---|---|
| 1 | München | 2 |
| 4 | Stuttgart | 1 |

# Abfragefunktionen

Abfragen können auch für verschiedene Berechnungen genutzt werden. Dazu kann mit dem ∑***Funktionen-Button*** eine weitere Zeile im unteren Bereich des Entwurfsfensters eingeblendet werden.


| | | |
|---|---|---|
| Feld: | | |
| Tabelle: | | |
| Funktion: | | |
| Sortierung: | | |
| Anzeigen: | ☐ | ☐ |
| Kriterien: | | |
| oder: | | |

Klickt man in die Zeile ***Funktion***, öffnet sich eine Auswahlliste. Wir wählen zunächst ***Gruppierung*** und in der Zeile 'Feld' ***lngLand***.

| | | |
|---|---|---|
| Feld: | lngLand | |
| Tabelle: | tabOrte | |
| Funktion: | Gruppierung | |
| Sortierung: | | |
| Anzeigen: | ☑ | ☐ |
| Kriterien: | | |
| oder: | | |

```
SELECT lngLand
FROM tabOrte
GROUP BY lngLand;
```

| **Abfrage1 : Auswahlabfrage** |
|---|
| **lngLand** |
| 1 |
| 2 |
| 3 |

Die Zahl '2' wird nur noch einmal angezeigt, obwohl sie im zugrundeliegenden Tabellenfeld mehrfach vorkommt. Die Gruppierung fasst also gleiche Datensätze zusammen.

Jetzt interessiert uns natürlich, wieviele Datensätze jeweils zusammengefasst wurden:

| | | |
|---|---|---|
| Feld: | lngLand | lngLand |
| Tabelle: | tabOrte | tabOrte |
| Funktion: | Gruppierung | Anzahl |
| Sortierung: | | |
| Anzeigen: | ☑ | ☑ |
| Kriterien: | | |
| oder: | | |

```
SELECT lngLand, Count(lngLand) AS AnzahlvonlngLand
FROM tabOrte
GROUP BY lngLand;
```

| **Abfrage1 : Auswahlabfrage** | |
|---|---|
| **lngLand** | **AnzahlvonlngLand** |
| 1 | 1 |
| 2 | 3 |
| 3 | 1 |

In der zugrundeliegenden Tabelle gibt es also dreimal den Eintrag '2', und je einmal eine '1' und eine '3'.
Die übrigen Abfragefunktionen dürften selbsterklärend sein:

***Summe*** ***Min*** ***Anzahl*** ***Varianz*** ***Letzter Wert***
***Mittelwert*** ***Max*** ***StAbw*** ***Erster Wert***

# Berechnungen

Nehmen wir an, wir haben folgende Tabelle:

| tabVerkaeufe : Tabelle | | |
|---|---|---|
| **IDVerkauf** | **datVerkauf** | **curNettopreis** |
| 1 | 19.10.2011 | 100 |
| 2 | 20.10.2011 | 50 |

Jetzt wollen wir mit einer Abfrage den Bruttopreis herausfinden. Dazu erstellen wir folgende Abfrage:


| Feld: | tabVerkaeufe.* | Bruttopreis: [curNettopreis]*1,19 |
|---|---|---|
| Tabelle: | tabVerkaeufe | |
| Sortierung: | | |
| Anzeigen: | ☑ | ☑ |
| Kriterien: | | |
| oder: | | |
| | | |

```
SELECT tabVerkaeufe.*, [curNettopreis]*1.19 AS Bruttopreis
FROM tabVerkaeufe;
```

Damit haben wir ein neues, berechnetes Feld erzeugt, welches das Gewünschte anzeigt:

| Abfrage1 : Auswahlabfrage | | | |
|---|---|---|---|
| **IDVerkauf** | **datVerkauf** | **curNettopreis** | **Bruttopreis** |
| 1 | 19.10.2011 | 100 | 119 |
| 2 | 20.10.2011 | 50 | 59,5 |

Der Feldname `Bruttopreis` selbst wurde erst in der Abfrage erzeugt. Auf diese Weise kann man auch ansonsten Felder beliebig (um)benennen.

Natürlich kann man nicht nur mit festen Werten rechnen, sondern auch mit mehreren Feldern.
Es wäre z.B. auch `[IDVerkauf] + [curNettopreis]` möglich (Der Sinn im obigen Beispiel mag dahingestellt bleiben).

Neben Plus (+), Minus (-), Mal (*), Geteilt (/) und Exponent (^) ist der ***Verkettungsoperator*** (&) wichtig, um Werte hintereinander zu setzen.
*Sind beide Operanden **Texte**, liefert Plus zwar in der Regel ein identisches Ergebnis.*
*Wenn aber beide Operanden aus Zahlen bestehen, liefern*
`[Feld1] + [Feld2]` *und*
`[Feld1] & [Feld2]` *unterschiedliche Ergebnisse.*
Da man sich oft nicht darauf verlassen kann, was User in Textfeldern eingeben, ist bei Texten der Verkettungsoperator vorzuziehen.

An dieser Stelle sei noch auf die schon unter Kriterienvorgestellten Operatoren verwiesen, da Berechnungen auch in den Kriterien stattfinden können.
Es gibt noch weitere Operatoren (wer sich mit binärer Logik auskennt, wird mit den Stichworten `And, Or, Not, Imp, Xor, Eqv` zurechtkommen), und zahlreiche Funktionen.
Z.B. kann man mit `Right([Name], 1)` das letzte Zeichen eines Textes ermitteln.

# Aktionsabfragen

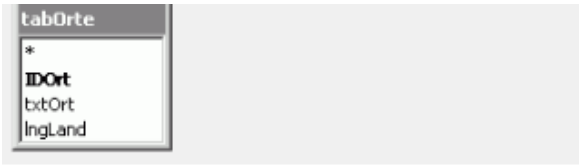
Nehmen wir einmal an, ein alter, urbayrischer Wunsch ginge in Erfüllung, und im Rahmen einer Förderalismusreform würde Baden-Württemberg zu Bayern hinzugeschlagen.
Zur Erinnerung; wir haben folgende Tabelle:

tabOrte : Tabelle

| IDOrt | txtOrt | lngLand |
|---|---|---|
| 1 | München | 2 |
| 2 | Bamberg | 2 |
| 3 | Berlin | 3 |
| 4 | Stuttgart | 1 |
| 5 | Bayreuth | 2 |

Jetzt müssen also alle Datensätze, die eine '1' im Feld `lngLand`stehen haben, auf '2' geändert werden. In unserem Beispiel ist das nur ein einziger Datensatz, aber nur, weil wir zu faul waren, für dieses Tutorial Hunderte von Daten einzugeben. Stellen wir uns also eben vor, es wäre nicht nur einer, sondern Hunderte von Datensätzen zugleich zu ändern. Wie dies nun umsetzen?

Zunächst erstellen wir eine Abfrage auf das zu ändernde Feld`lngLand`, und grenzen es auf die zu ändernden Datensätze ein:




```
SELECT lngLand
FROM tabOrte
WHERE lngLand=1;
```

| | | |
|---|---|---|
| Feld: | lngLand | |
| Tabelle: | tabOrte | |
| Sortierung: | | |
| Anzeigen: | ☑ | ☐ |
| Kriterien: | 1 | |
| oder: | | |

Dies sind die zu ändernden Werte. Bisher hatten wir es ausschließlich mit sogenannten ***Auswahlabfragen*** zu tun. Jetzt wird es Zeit für eine***Aktualisierungsabfrage***.
Dazu wählen wir in der Entwurfsansicht den***Abfragetyp*** aus, und ändern die Einstellung auf ***Aktualisierungsabfrage***. Der untere Teil des Entwurfsfensters verändert sich dadurch:

| | | |
|---|---|---|
| Feld: | lngLand | |
| Tabelle: | tabOrte | |
| Aktualisieren: | | |
| Kriterien: | 1 | |
| oder: | | |

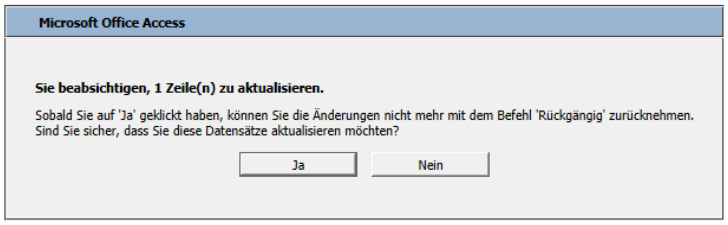
Unser Kriterium '1' ist erhalten geblieben, in der Zeile ***Aktualisieren*** tragen wir jetzt eine '2' ein:

| | | |
|---|---|---|
| Feld: | lngLand | |
| Tabelle: | tabOrte | |
| Aktualisieren: | 2 | |
| Kriterien: | 1 | |
| oder: | | |

Wenn wir jetzt einfach wieder in die Datenblattansicht schalten, sehen wir noch immer unsere '1', wie schon zuvor. Obwohl wir jetzt einen anderen SQL-Code haben, hat sich in der Datenblattansicht (außer der Überschrift) nichts verändert.


```
UPDATE tabOrte SET lngLand = 2
WHERE lngLand=1;
```

**Jetzt probieren wir den Button !*Ausführen* aus**




Wir wählen 'Ja', öffnen danach die Tabelle `tabOrte` und sehen, dass im Feld `lngLand` überall dort, wo bisher eine '1' stand, jetzt eine '2' steht. Unsere Aktualisierungsabfrage hat also genau das gemacht, was wir wollten!

Folgende Arten von Aktionsabfragen gibt es:

***Tabellenerstellungsabfragen*** speichern das Abfrageergebnis in einer neuen Tabelle. Sinnvoll z.B. zum Export in Fremdprogramme oder, um einen aktuellen, sich später ändernden Zustand der Daten festzuhalten.

***Aktualisierungsabfragen*** verändern, wie eben gezeigt, bestehende Daten.

***Anfügeabfragen*** fügen an eine bestehende Tabelle die Datensätze an, die von der Abfrage geliefert werden. Natürlich müssen dafür die Felder in die Tabelle passen.

***Löschabfragen*** löschen unwiederbringlich die Datensätze, die von der Abfrage geliefert werden.

# Kreuztabellenabfragen

***Beim folgenden Beispiel gehen wir davon aus, dass die Datenquelle keine Tabelle, sondern eine Abfrage sei.*** Einerseits um zu zeigen, dass das geht, zum anderen, weil die folgenden Daten in einer Tabelle kaum sinnvoll normalisiert wären:

qryVerkaeufe : Auswahlabfrage

| IDVerkauf | datVerkaufsDatum | txtProdukt | lngStueck | curEinnahme |
|---|---|---|---|---|
| 1 | 01.02.2010 | Kaugummi | 10 | 5,00 |
| 2 | 01.02.2010 | Lutscher | 15 | 1,50 |
| 3 | 01.02.2010 | Paprikachips | 5 | 10,00 |
| 4 | 02.02.2010 | Kaugummi | 5 | 2,50 |
| 5 | 02.02.2010 | Paprikachips | 5 | 10,00 |


Nun wollen wir die Verkäufe eines Tages in einer Zeile sehen. Die Produkte sollen spaltenweise dargestellt werden. Dafür wählen wir den ***Abfragetyp***'Kreuztabelle' aus. Der untere Teil des Abfragefensters verändert sich daraufhin: Der uns schon bekannte ∑***Funktionen-Button*** wird aktiviert, eine neue Zeile 'Kreuztabelle' kommt hinzu, und die Zeile 'Anzeigen' verschwindet. Wir füllen dies wie folgt aus:

| | |
|---|---|
| Feld: | datVerkauf |
| Tabelle: | tabVerkaeu |
| Funktion: | Gruppierun |
| Kreuztabelle: | Zeilenüber |
| Sortierung: | Aufsteigen |
| Kriterien: | |
| oder: | |

```
TRANSFORM Sum(tabVerkaeufe.lngStueck)
SELECT tabVerkaeufe.datVerkaufsDatum
FROM tabVerkaeufe
GROUP BY tabVerkaeufe.datVerkaufsDatum
PIVOT tabVerkaeufe.txtProdukt;
```

**Abfrage1 : Kreuztabellenabfrage**

| datVerkaufsDatum | Kaugummi | Lutscher | Paprikachips |
|---|---|---|---|
| 01.02.2010 | 10 | 15 | 5 |
| 02.02.2010 | 5 | | 5 |

Die Abfrage zeigt, wieviele Produkte pro Tag verkauft wurden. Wer das Beispiel am eigenen PC nachverfolgt, sollte auch einmal mit anderen Funktionen in der Spalte 'lngStueck' experimentieren, oder 'lngStueck' durch 'curEinnahme' ersetzen.

# Tipps & Tricks zu Abfragen

Auch zum Kapitel Abfragen noch ein paar allgemeine Tipps und Tricks zur Performancesteigerung - auch, wenn nicht alles, was hier erwähnt wird, im Tutorial vorkam.

Oft sind verschiedene Lösungen möglich, damit eine Abfrage das Gewünschte ausgibt. Unter bestimmten Umständen testet Access eigenständig, ob ein anderer Lösungsweg als der, den man gerade vorgegeben hat, schneller ist (in diesem Zusammenhang kann man über Begriffe wie 'Rushmore' oder 'Showplan' stoßen; darauf wird hier nicht näher eingegangen).

Die wichtigste Voraussetzung für eine interne Optimierung ist, dass die ***Abfrage auch als Abfrage gespeichert*** ist. Enthält ein Formular als Datenquelle einfach SQL, oder 'bastelt' man SQL per VBA zur Laufzeit, kann Access nicht intern optimieren.

Vor dem Speichern sollte man eine Abfrage außerdem mindestens einmal ***ausführen***. Dabei findet dann die Optimierung statt, die dann unsichtbar, zusammen mit der Abfrage gespeichert wird.

In einer Abfrage sollten so wenig ***Beziehungen*** wie möglich vorkommen. Im Allgemeinen sollte nicht mehr als eine Beziehung zwischen zwei Tabellen vorhanden sein.

Für ***Kreuztabellenabfragen*** empfehlen sich fixierte Spaltenüberschriften, also die Begrenzung nur auf die benötigten Spalten.

***Aktionsabfragen*** *sollten unter VBA* weder mit OpenQuery noch mit RunSQL ausgeführt werden, sondern mit DB.Execute.

Werden Abfragen ziemlich komplex, kann es helfen, ***die Abfrage zu splitten***, also aus einer Abfrage mehrere zu machen. Häufig stellt sich dabei heraus, dass gleich mehrere Abfragen auf eine 'Basisabfrage' zugreifen können.

Beim Splitten der Abfrage sollten folgende Überlegungen greifen:

> Die 'Basisabfrage' sollte die Datenmenge eingrenzen. Das geschieht über die Bedingungen (WHERE) sowie die Eingrenzung der benötigten Felder.
>
> Die 'Endabfrage' ist dann zuständig für zeitaufwendige Operationen.
> Dies sind insbesondere: Gruppieren, Sortieren und berechnete Felder.
> Auf je weniger Daten diese Abfrage zugreifen muss, desto schneller kann sie arbeiten.

# Formulare

Formulare dienen der Interaktion des Benutzers; sie sind die***Benutzerschnittstelle***. In Formularen werden Daten eingegeben und ausgewertet. Ein Benutzer sollte möglichst nie Tabellen oder Abfragen zu sehen bekommen, sondern immer nur Formulare.

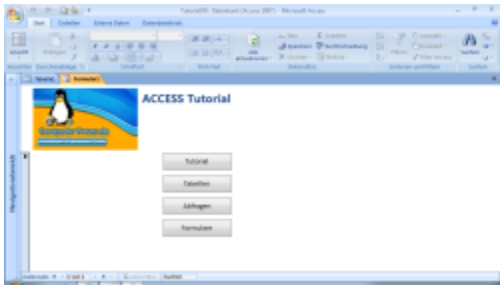
Wie wir gelernt haben, müssen Tabellen aus Sicht einer effektiven Datenhaltung streng systematisch geplant werden. Im Gegensatz dazu sollten Formulare einen Benutzer einfach und intuitiv leiten. Zwischen diesen beiden Welten vermitteln Abfragen.

Obwohl man natürlich schon vorher ungefähr wissen sollte, wofür man ein Formular entwirft, ist das Erstellen eines Formulars eine Sache, bei der man fleißig herumprobieren kann und sollte. Ein wesentlicher Unterschied zum Entwerfen von Tabellen!

## Ein erstes Formular

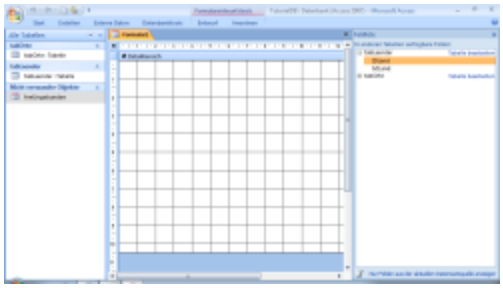
Wie alle Datenbankobjekte, werden auch neue Formulare mit ***Neu*** angelegt. Wie üblich, machen wir das über die Entwurfsansicht (obwohl die Formularassistenten von Microsoft Access gute Formulare erstellen können). Bei vorhandenen Formularen kommt man mit ***Entwurf*** in die Entwurfsansicht und mit ***Öffnen*** in die Formularansicht.

Beim Anlegen eines neuen Formular kann man wählen, aus welcher Tabelle oder Abfrage die Daten für das Formular stammen sollen. Diese Datenherkunft kann man später noch ändern. In der Praxis bekommt meistens jedes Formular eine eigene ***Abfrage*** als Datenherkunft. Formulare mit einer ***Tabelle*** als Datenherkunft sind eher selten: Beim Erstellen eines Formulars zeigt sich oft, dass das Formular Kriterien benötigt, die am besten mit einer Abfrage verwirklicht werden.



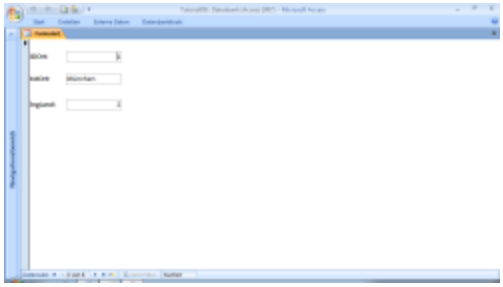

Ein Formular kann aber auch ohne Datenherkunft auskommen - ein sogenanntes ***ungebundenes Formular***.
Typisches Beispiel ist eine Benutzersteuerung, in der der User wählt, was er machen will, z.B. andere Formulare öffnen.

Im Formularentwurf bekommt man zunächst ein leeres Formular zu sehen, das nur ein paar Hilfslinien und Lineale enthält, mit denen man die einzufügenden Objekte ausrichten kann.



Bei einem ***gebundenen Formular*** (also ein Formular mit Datenherkunft) kann man mittels ***Feldliste***die Felder der Datenherkunft einblenden. Das Aussehen der Feldliste kennen wir schon von Beziehungen undAbfragen.

Zieht man Formula- Textfelder Entwurfs- Daten der oder neue erstes



mit der Maus einen oder mehrere Feldnamen aus der Feldliste auf den rentwurf, werden die Felder dort als angezeigt. Schaltet man nun von der in die ***Formularansicht***, können die zugrundeliegenden Tabellen geändert Datensätze erfasst werden. Unser Formular ist also einsatzbereit!

## Steuerelemente

Formulare enthalten meist verschiedene Arten von Steuerelementtypen. Diese wählt man mit der ***Toolbox***aus (Entwurfsansicht des Formulares / ‚Entwurf'). Klickt man in der Toolbox einen Steuerelementtyp an und zieht danach ein Feld aus der Feldliste ins Formular, bekommt das Feld eben diesen Steuerelementtyp. Hier die wichtigsten Steuerelementtypen:

### Textfeld

In ein Textfeld können beliebige Texte und Zahlen geschrieben werden.

### Kontrollkästchen

Kontrollkästchen können per Mausklick mit einem Häkchen versehen bzw. das Häkchen wieder entfernt werden. Gedacht sind sie für Ja/Nein-Werte (Boolean). Das ist aber nicht zwingend so. Liegt einem Kontrollkästchen z.B. ein Feld vom Typ 'Zahl' zugrunde, so wird der Feldwert auf '0' gesetzt, wenn das Kästchen geleert, und auf '-1', wenn ein Häkchen gesetzt wird. '0' steht also für 'Nein' und '-1' für 'Ja'.

### Optionsfeld, Optionsgruppe

Optionsfelder werden in Optionsgruppen eingesetzt. Eine Optionsgruppe ist ein Rahmen, der die darin befindlichen Optionsfelder als zusammengehörig definiert. Außerdem ist die Optionsgruppe mit dem zugrundeliegenden Datenfeld verknüpft. Optionsfelder sind Knöpfe, aus denen ein Benutzer immer nur genau einen auswählen kann. Der Wert des aktivierten Optionsfelds wird in das der Optionsgruppe zugeordnete Datenfeld geschrieben.

### Kombinationsfeld


Kombinationsfelder zeigen Werte aus einer frei definierbaren Liste oder einer Tabelle/Abfrage an. Der Benutzer kann einen davon auswählen und ein damit in Zusammenhang stehender Wert wird eingetragen.

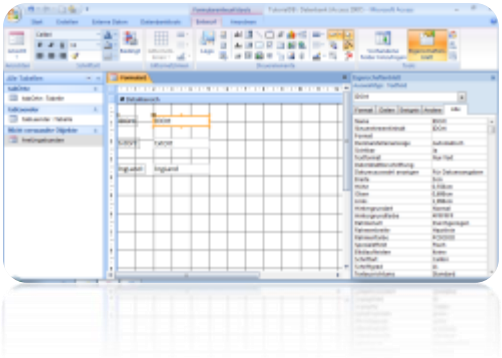
### Bezeichnungsfeld

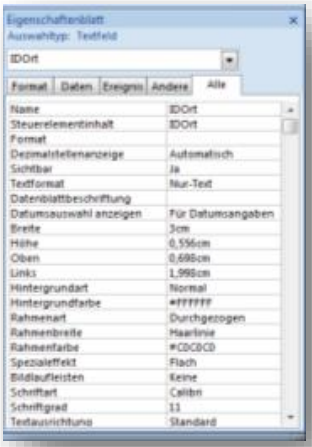
Bezeichnungsfelder enthalten freien Text zur Information des Benutzers. Man kann beliebige Bezeichnungsfelder einzufügen.

*Meistens sollte man assoziierte Bezeichnungsfelder noch umbenennen: Fügt man in der Entwurfsansicht in ein Bezeichnungsfeld ein '&' ein, wird das darauffolgende Zeichen des Bezeichnungsfeldes in der Formularansicht unterstrichen dargestellt. Damit bekommt das assoziierte Steuerelement ein entsprechendes Tastaturkürzel und kann mit* `[ALT]-[Taste]` *angesprochen werden.*

## Eigenschaften

Alle Steuerelemente - und auch das Formular selbst - können über das  ***Eigenschaftenfenster*** angepasst werden. Natürlich sind die Eigenschaften nicht für alle Objekte gleich, aber sie sind sich doch recht ähnlich. Zur besseren Orientierung ist das Eigenschaftenfenster in Karteiregister unterteilt.






Unter ***Format*** finden sich Eigenschaften, die das Aussehen des Elements bestimmen, z.B. Text- und Hintergrundfarbe oder den Rahmen.

Unter ***Daten*** wird insbesondere die Datenherkunft des Objekts festgelegt.

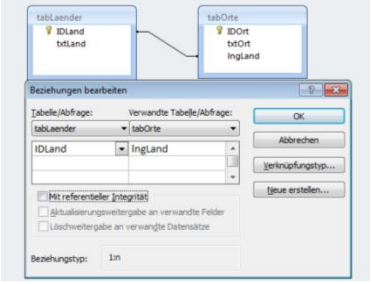
***Ereignis*** ist im Zusammenhang mit Automatisierung (Visual Basic) interessant. Es kann z.B. bestimmt werden, dass beim Klicken oder beim Verlassen eines Elements bestimmte Aktionen passieren, wie z.B. das Ausdrucken eines Berichts.

Unter ***Andere*** finden sich weitere Eigenschaften. Wichtig ist z.B. der Name eines Steuerelements, aber auch die Reihenfolge der Steuerelemente wird hier festgelegt.

Wem es optisch lieber ist, findet unter ***Alle*** alle Eigenschaften zugleich.

# Ein Beispielformular

Genug der Theorie, Zeit für die Praxis! Dafür nutzen wir wieder dasBeispiel, das uns schon geraume Zeit verfolgt: Die Erfassung von Orten samt Bundesländern. Zur Erinnerung, wir haben zwei Tabellen:



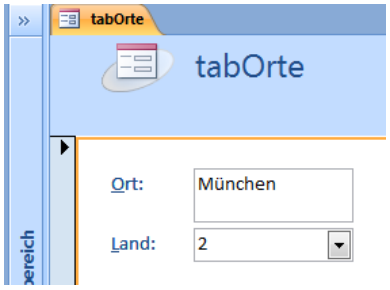

Mit ***Neu*** (Erstellen ‚Formular') erstellen wir ein neues Formular und wählen `tabOrte` als Datenherkunft. In den Formulareigenschaften (Register 'Daten', Eigenschaft 'Datenherkunft') kann man jede vorhandene Tabelle oder Abfrage als Datenquelle auswählen. Man könnte sogar eine neue Abfrage kreieren, die nicht als Abfrage, sondern im Formular selbst gespeichert wird. Davon ist aber abzuraten! So eine Abfrage ist stets langsamer als eine Abfrage, die auch als Abfrage gespeichert ist.

In der Entwurfsansicht sollten ***Feldliste***, ***Eigenschaftenfenster*** und ***Toolbox*** eingeschaltet sein. Der ***Steuerelement-Assistent*** in der Toolbox sollte ausgeschaltet sein. Das ist zwar u.U. eine nützliche Hilfe, aber für den Lerneffekt eines Tutorials kontraproduktiv.

Zuerst ziehen wir `txtOrt` aus der Feldliste in den Formularentwurf. Wir sehen dort jetzt ein Textfeld, in dem `txtOrt` steht, und davor ein Bezeichnungsfeld, das mit `txtOrt:` beschriftet ist. Die Beschriftung ändern wir, indem wir darauf klicken, um es auszuwählen, und dann im Eigenschaftenfenster das Register 'Format' wählen. Dort findet sich die Eigenschaft 'Beschriftung', die wir von `txtOrt:` auf `&Ort:` (mit einem '&' am Anfang) ändern.

Kleine Anmerkung: Im Tabellenentwurf von `tabOrte` hätten wir im unteren Bereich des Feldes `txtOrt` unter 'Beschriftung' schon `&Ort:` eingeben können. Dann wäre beim Einfügen des Feldes ins Formular gleich diese Bezeichnung eingefügt worden. Die Nutzung ist Geschmackssache.

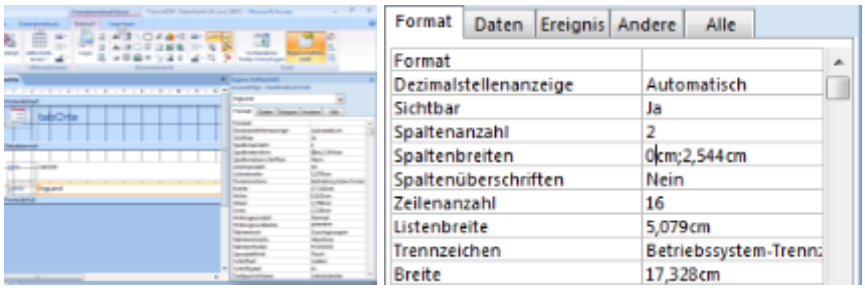
Nun klicken wir in der Toolbox auf ***Kombinationsfeld***, und ziehen aus der Feldliste `lngLand` ins Formular. Auch hier ändern wir die Bezeichnung, nämlich auf `&Land:`. Außerdem ändern wir den Namen des Kombinationsfeldes im Register 'Andere' auf `cboLand`. Dann legen wir fest, was das Kombinationsfeld eigentlich anzeigen soll. Dazu wählen wir im Eigenschaftenfenster das Register 'Daten'.




Der Steuerelementinhalt `lngLand` stimmt ja schon (das haben wir schließlich ins Formular gezogen), die Eigenschaft 'Herkunftstyp' muss auf `Tabelle/Abfrage` stehen, die Datensatzherkunft auf `tabLaender`, und 'gebundene Spalte' auf `1`. Jetzt schalten wir probeweise in die Formularansicht.

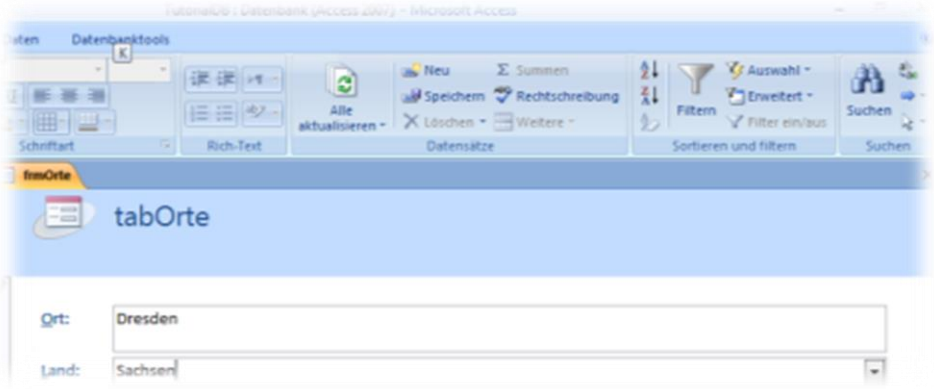
Jetzt ist der Buchstabe 'O' im Wort 'Ort' und das 'L' im Wort 'Land' unterstrichen. Wir können also mit den Tastenkombinationen `[ALT]-O` bzw. `[ALT]-L` zwischen den beiden Steuerelementen hin- und herspringen.

Was das Formular nicht anzeigt, ist der ID des Ortes. Aber muss ein Benutzer den ID wirklich sehen? Das Feld ist vom Typ Autowert und füllt sich selbst. Normalerweise kann auf derartige Angaben in Formularen getrost verzichtet werden. Unser Formular ist also fertig, und wir speichern es unter `frmOrte` ab.

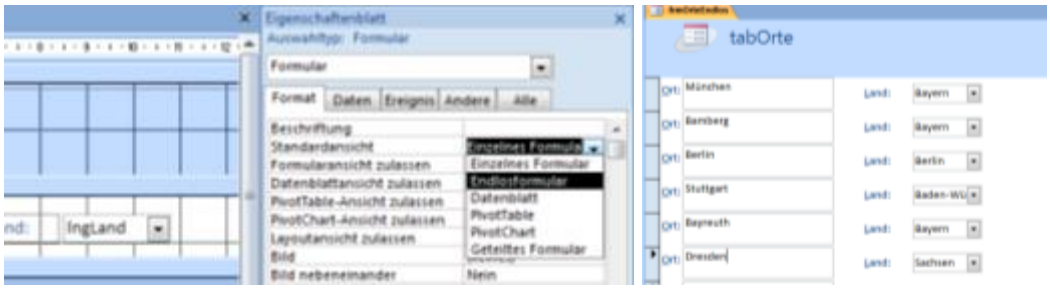



## Endlosformular

Mit dem soeben erstellten Formular`frmOrte` beschäftigen wir uns noch eingehender: Das Formular hat nur zwei Steuerelemente. In der Praxis werden darüber wohl ziemlich viele Datensätze eingegeben, von denen man aber immer nur einen sieht. Es wäre übersichtlicher, eine Liste zu haben.




Kein Problem für Access:
Zuerst verschieben wir die beiden Steuerelemente so, dass sie nebeneinander liegen. Beide sollten ziemlich weit oben, knapp unter dem Wort 'Detailbereich' liegen. Die Größe des Detailbereichs selbst kann man ebenfalls mit der Maus anpassen, wir sorgen für eine geringe Höhe, z.B. 1cm oder weniger.
Dann wählen wir im Eigenschaftsfenster das Formular aus und stellen im Register 'Format' die Eigenschaft 'Standardansicht' auf `Endlosformular`.
Zurück in der Formularansicht haben wir unser Ziel schon erreicht!



Es ist unnötig, in jedem Datensatz die Beschriftungen 'Ort' und 'Land' zu sehen. Deswegen löschen wir in der Entwurfsansicht die Bezeichnungsfelder und wählen im Menü `Ansicht - Formularkopf/-fuß`. Die Entwurfsansicht ist jetzt unterteilt in `Formularkopf, Detailbereich, Formularfuß`.

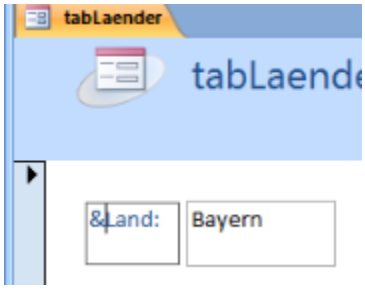

Im Formularkopf fügen wir zwei Bezeichnungsfelder ein, von denen wir eines mit `Ort` beschriften und über dem Textfeld `txtOrt` im Detailbereich ausrichten. Ein weiteres Bezeichnungsfeld beschriften wir mit `Bundesland` und richten es über `cboLand` aus. Dem Formularfuß verpassen wir abschließend noch eine Höhe von 0.

Nun haben wir eine 'Überschriftenzeile', die auch beim Scrollen stehen bleibt.

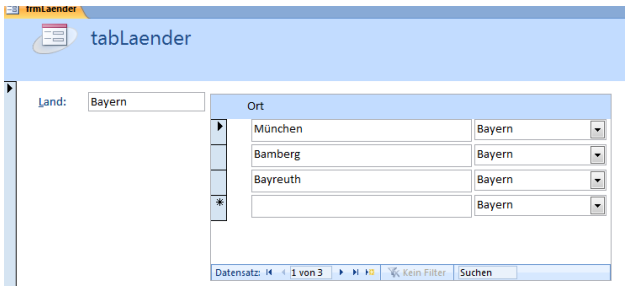
## Unterformular

Das Endlosformular `frmOrte`zeigt nun alle Orte aus `tabOrte` an. Es wäre aber praktisch, immer man nur die Orte eines bestimmten Bundeslandes zu sehen.

Dazu erstellen wir ein neues Formular, das wir `frmLaender` nennen und das auf `tabLaender` zugreift. Für unser Beispiel genügt es, wenn ein einfaches Textfeld darin enthalten ist, das `txtLand` anzeigt.




Nun fügen wir aus der Toolbox (Entwurf ‚steuerelemente') ein ***Unterformular*** ein. Diesem neuen Steuerelement weisen wir im Eigenschaftsfenster als Herkunftsobjekt (in der Registerkarte 'Daten') unser Endlosformular `frmOrte` zu. Und falls das nicht automatisch passieren sollte, stellen wir die Eigenschaften 'Verknüpfen von' sowie 'Verknüpfen nach' auf`lngLand` bzw. `IDLand`.

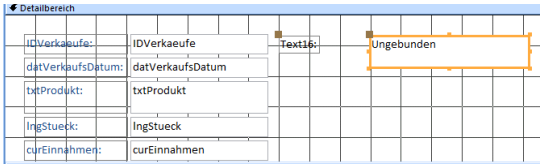


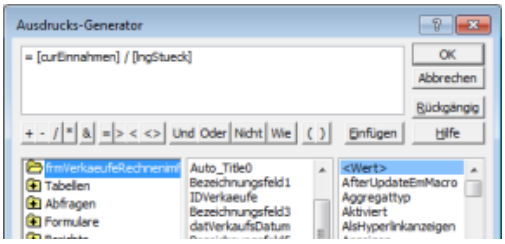

Damit wird in `frmLaender` in einem 'Fensterchen' `frmOrte`angezeigt - aber nur noch mit den Orten, die im gerade angezeigten Bundesland liegen *(Man kann also in `frmOrte` auch die Anzeige der Bundesländer löschen).* Geht man jetzt im Hauptformular zu einem anderen Datensatz, werden auch im Unterformular die dazu passenden Datensätze angezeigt.

## „freies Rechnen“ im Formular


Kleine Berechnungen können sich quasi en passent in das fertige Formular integrieren lassen, ohne dass man dafür die Abfragen bemühen muss!

Man wechselt in die Entwurfsansicht und zieht (als Ergebnisfeld) ein **ungebundenes** **Textfeld** auf. (Entwurf ‚Steuerelemente‘)




Eigenschaften des Feldes aufrufen und ‚Steuerelementeinhalt‘ ‘…‘ aktivieren

Im ‚Ausdrucks-Generator‘ wählt man die zu berechnenden Felder mit den dazugehörigen Rechenoperationen aus.

Formatierung des Ausgabeformates im ‚Eingabeformat‘ – fertig, das war’s.


# Tipps & Tricks zu Formularen

Beim Entwerfen von Formularen führen meistens viele Wege zum Ziel. Ein paar Dinge sollte man aber beachten:

Was ist mit der ***Intuition des Benutzers***:
Was erwartet ein Benutzer vom Formular, und wie kommt er mit den Elementen des Formulars zurecht? Technisch sind zahllose Möglichkeiten vorhanden, und es lassen sich faszinierende Lösungen im Internet finden. **Aber in der Praxis ist meistens das einfache Formular dem technisch verliebten überlegen.** Schon alleine, weil es nicht lange erklärt werden muss.

Was ist mit der ***Übersichtlichkeit*** des Formulars:
Überfrachtet man ein Fomular mit zu viel Information, sieht der Benutzer den Wald vor lauter Bäumen nicht mehr.
Dann sollte man entweder (oft nicht so gut) die Daten auf mehreren Registern verteilen, oder (oft besser) mehrere Formulare verwenden.

Gibt es ***Wiedererkennungseffekte***?
Darunter kann man verschiedenes verstehen:

1. Wenn die Nutzer z.B. vorher schon ein anderes System hatten- egal, ob Papier oder EDV - erleichtert es die Einarbeitung, wenn das Formular ähnlich aussieht.
2. Wenn die Anwendung in mehreren Formularen z.T. gleiche Informationen zeigt, sollten diese Teile gleich aussehen.
3. Wenn es an verschiedenen Stellen Steuerelemente mit gleichen oder sehr ähnlichen Zwecken gibt, sollten sie gleich formatiert werden
   (Z.B. nicht einmal einen Button mit 'Drucken' beschriften, und einmal mit einem Bild).

Die Datenherkunft eines Formulars kann eine Tabelle oder eine Abfrage sein.

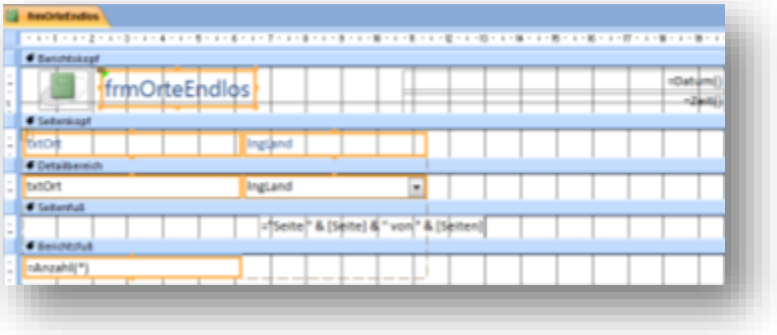
# Berichte

Berichte dienen dem Ausdruck auf Papier. Technisch unterscheiden sie sich fast nicht von Formularen. Nur, dass man in den Steuerelementen keine Daten pflegen, sondern nur ansehen kann. Wegen dieser Ähnlichkeit zu Formularen wird hier nicht mehr darauf eingegangen, wie man einen Bericht anlegt und welche Steuerelemente es gibt.

Manche der im folgenden beschriebenen Möglichkeiten sind - besonders in neueren Accessversionen - auch in Formularen anwendbar. In gewisser Weise handelt es sich also auch um ein vertiefendes Kapitel. Es lohnt sich, das ein oder andere einfach mal in einem Formular auszuprobieren!

## Berichtsbereiche

Wie wir schon bei den Formularen gelernt haben, kann ein Formular einenFormularkopf/-fuß haben. Ein Bericht kann darüber hinaus auch noch einen ***Seitenkopf/-fuß*** haben. Beim Ausdruck des Berichts wird der Berichtskopf nur auf der ersten Seite des Berichts ausgedruckt, der Seitenkopf hingegen oben - bei der ersten Seite unterhalb des Berichtskopfs - auf jeder Seite des Berichts.




Dieser Beispielbericht enthält im Berichtskopf ein Bezeichnungsfeld mit der Aufschrift 'frmOrteEndlos“ und daneben die Textfelder 'Datum' und ‚Zeit‘. Der Steuerelementinhalt dieser Felder ist aber nicht an ein Feld der zugrundeliegenden Tabelle oder Abfrage gebunden, sondern enthält die Funktionen `=Datum()` bzw. `=Zeit()`.


Ähnlich ist es mit den Textfeldern 'Seite' / ‚Seiten‘ im Seitenfuß. Außerdem enthält der Seitenkopf noch unter der Seitenzahl eine Linie. Da der Seitenkopf - anders als der Berichtskopf -auf jeder Seite gedruckt wird, erscheinen beim Ausdruck die Seitenzahl und die Linie oben auf jeder Seite.

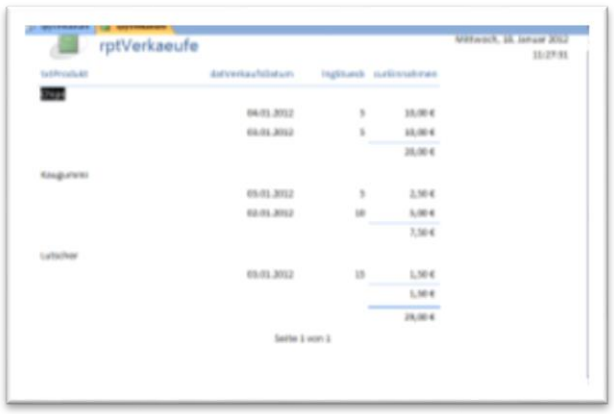
## Gruppieren

Nehmen wir an, wir haben folgende Abfrage:

qryVerkaeufe

| datVerkaufsDatum | txtProdukt | lngStueck | curEinnahm |
|---|---|---|---|
| 02.01.2012 | Kaugummi | 10 | 5,00 € |
| 03.01.2012 | Lutscher | 15 | 1,50 € |
| 03.01.2012 | Chips | 5 | 10,00 € |
| 03.01.2012 | Kaugummi | 5 | 2,50 € |
| 04.01.2012 | Chips | 5 | 10,00 € |

Daraus wollen wir einen Bericht erstellen, der uns die Verkäufe nach Produkten ausgibt.
Das wäre auch mit einem Formular bzw. einer Abfrage kein Problem - wir wollen aber auch Zwischensummen einfügen.
Hierfür gibt es im Berichtsentwurf die Möglichkeit, zu ***gruppieren***.
Gruppierungen bringen noch zusätzliche Berichtsbereiche.


# Tipps & Tricks zu Berichten

Häufig werden bestimmte Berichte nur selten bzw. nur von bestimmten Personen benötigt, z.B. ein Quartalsbericht an den Vorgesetzten.
Warum also nicht solche Berichte in ein eigenes Frontend packen, das dann auch nur dem Vorgesetzten zur Verfügung steht? So bleibt das Frontend des 'Normalanwenders' für den Programmierer übersichtlicher, und der Vorgesetzte kann sich seine Berichte jederzeit selbst abrufen.

Wer mit Access umgehen kann, kann leicht einen Bericht seinen Bedürfnissen anpassen. Aber meistens ist es nicht gewollt, dass jeder Benutzer auch die Datenbankobjekte verändern kann oder die Benutzer sind dazu nicht in der Lage.

In solchen Fällen kann man auch ***Microsoft Word*** für Berichte nutzen: Man reicht die Access-Daten an Word weiter und fügt sie entweder an Textmarken ein, oder nutzt die Serienbrieffunktionen von ***Microsoft Word***.