Ausgabe 05/2023

ACCESS BASICS

DAS ACCESS-MAGAZIN FÜR ALLE, DIE VON 0 AUF 100 WOLLEN

In diesem Heft:

EINGEBAUTE FUNKTIONEN: RUND UM DATUM UND UHRZEIT (S. 31)

André Minhorst

www.access-basic.de

ACCESS BASICS

DAS ACCESS-MAGAZIN FÜR ALLE, DIE VON 0 AUF 100 WOLLEN

Ausgabe 05/2023

Terminplanung mit Access auf einem neuen Level

In unserem Shop gibt es ein tolles, neues Produkt: den **amvCalendar**!

Diese Lösung kannst Du problemlos in alle Access-Datenbanken integrieren. Du kannst damit Termine anlegen und diese in Kalendern in verschiedenen Ansichten wie Tag, Arbeitswoche, Woche oder Monat darstellen.

Dazu sind keine externen ActiveX-Komponenten oder DLLs nötig. Du brauchst nur die Objekte der Lösung in Deine Datenbank zu importieren und kannst den Kalender direkt anzeigen und damit arbeiten.

Der Kalender hat eine für die Anzahl der angezeigten Steuerelemente nicht für möglich zu haltende Performance.

Überzeuge Dich selbst und hole Dir die Demo von unserer Seite unter folgendem Link:

https://www.amvcalendar.de

Viel Spaß beim Ausprobieren!

André Minhorst

IMPRESSUM

ACCESS [BASICS] WIRD HERAUSGEGEBEN VON:
Minhorst und Minhorst GbR - André Minhorst Verlag
Borkhofer Straße 17
47137 Duisburg

REDAKTION:
André Minhorst (V.i.S.d.P)
Telefon: 0203/4495577
E-Mail: info@access-basics.de
Internet: www.access-basics.de

Geschäftsführung, Herstellung, Text- und Schlussredaktion, Layout von Magazin und Webseite: André Minhorst
Autor: André Minhorst
Fach- und Sprachlektorat: Carsten Gromberg

ISSN: 2190-8761

Tabellen [basics]: 1:1-Beziehungen

Nachdem wir bereits die 1:n-Beziehung und die m:n-Beziehung kennengelernt haben, wobei letztere nur zwei 1:n-Beziehungen mit zwei Mastertabellen und der gleichen Detailtabelle sind, schauen wir uns nun den letzten Beziehungstyp an: die 1:1-Beziehung. Dabei legen wir eine Einschränkung fest, bei der wir einen Datensatz der einen Tabelle genau einem Datensatz der anderen Tabelle zuordnen können. Aber wozu sollte man das brauchen – und wie realisiert man eine 1:1-Beziehung? All das erläutern wir in diesem Artikel.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **TabellenBasics_11Beziehungen.accdb**.

Was sind 1:1-Beziehungen?

Bevor wir richtig einsteigen, eine kurze Beschreibung, was 1:1-Beziehungen eigentlich sind. Dabei handelt es sich eigentlich um eine 1:n-Beziehung, bei der wir in der Detailtabelle, also der Tabelle mit dem an der Beziehung beteiligten Fremdschlüsselfeld, für das Fremdschlüsselfeld einen eindeutigen Index festlegen. Das bedeutet, das wir jeden Datensatz der Mastertabelle nur einmal zu einem Datensatz der Detailtabelle zuordnen können. Wie wir die 1:1-Beziehung technisch definieren, erläutern wir weiter unten.

Warum 1:1-Beziehungen?

Nun schauen wir uns erst einmal an, warum man überhaupt 1:1-Beziehungen nutzen sollte.

- Ein zwingender Grund, eine 1:1-Beziehung zu definieren, ist eine mögliche Beschränkung in der Anzahl der Felder, die eine Tabelle aufnehmen kann. In einer Access-Datenbank sind dies 255 Felder. Danach weigert sich Access, weitere Felder anzulegen (siehe Bild 1). In SQL Server-Datenbanken sind 30.000 Felder je Tabelle möglich – aber wenn man so viele Felder benötigt, sollte man ohnehin sein Datenmodell überdenken.

- Tabellen mit vielen Feldern auf mehrere kleine Tabellen aufzuteilen kann jedoch auch zu einer besseren Performance führen. Man braucht nicht immer alle Felder einer Tabelle, und wenn man einige davon in einer per 1:1-Beziehung verknüpften Tabelle auslagert und diese nicht immer benötigt, müssen weniger Daten transportiert werden. Das ist insbesondere in Access-Datenbanken wichtig, die in Frontend und Backend

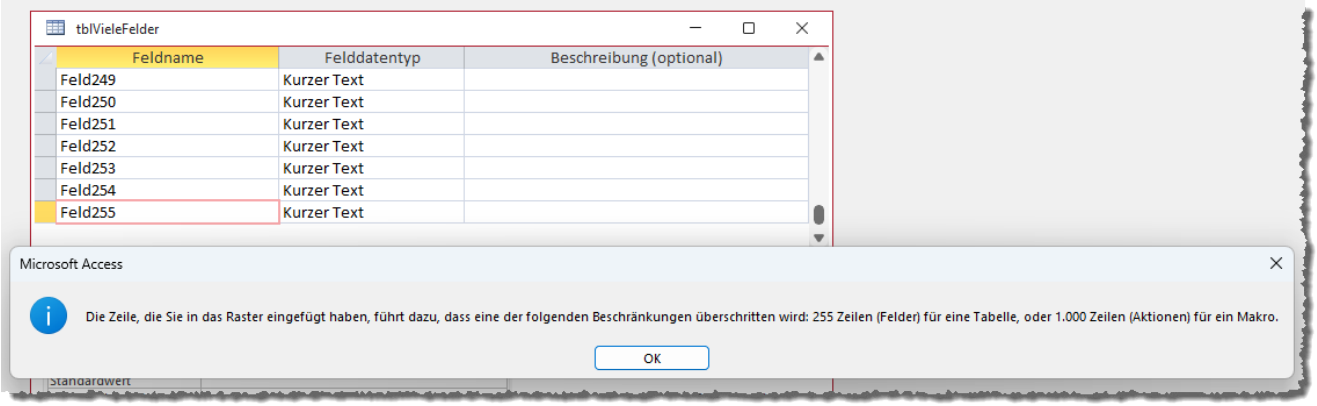

Bild 1: Nach 255 Feldern ist Schluss.

aufgeteilt sind. Hier werden im Gegensatz zu Datenbanken beispielsweise mit einem SQL Server-Backend immer alle Daten einer Tabelle übermittelt, wenn man per Abfrage auf diese zugreift.

- Hier könnte man seine Tabellen auf Felder überprüfen, die selten gefüllt werden. Ist das der Fall, könnte man diese Felder in eine Tabelle auslagern und diese per 1:1-Beziehung mit der ursprünglichen Tabelle verknüpfen.

- Eine 1:1-Beziehung kann auch zur Regelung der Zugriffskontrolle verwendet werden: In manchen Fällen ist es notwendig, bestimmte Informationen aufzuteilen und den Zugriff auf diese Informationen auf verschiedene Benutzer oder Benutzergruppen zu beschränken. Eine 1:1-Beziehung kann verwendet werden, um eine Entität mit sensiblen Daten zu erstellen und sicherzustellen, dass nur autorisierte Benutzer Zugriff auf diese Entität haben – beispielsweise, indem man in einem Frontend für Benutzer mit weniger Rechten nur den Teil der 1:1-Beziehung anzeigt, der für diese relevant ist, für Benutzer mit mehr Rechten aber auch den zweiten Teil der 1:1-Beziehung.

- Manchmal gibt es auch einfach zwei Tabellen, von denen man denkt, sie müssten in einer 1:n-Beziehung stehen. Tatsächlich ist aber dann eine 1:1-Beziehung der richtige Weg.

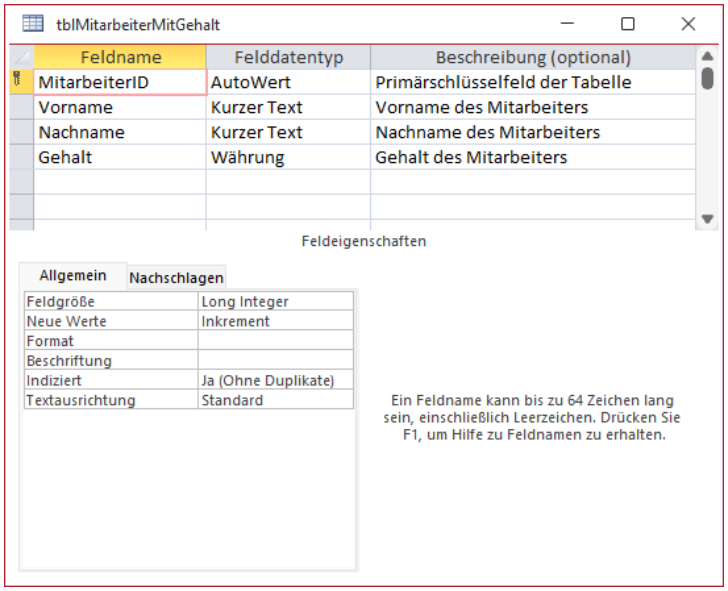

Bild 2: Diese Tabelle soll in zwei per 1:1-Beziehung verknüpfte Tabellen aufgeteilt werden.

- Gelegentlich braucht man Felder in einer Tabelle einfach nicht mehr, aber man möchte diese dennoch für den Fall der Fälle vorhalten. Dann könnte man eine Tabelle für diese Felder erstellen und sie per 1:1-Beziehung mit der eigentlichen Tabelle verknüpfen.

Tabelle in zwei per 1:1-Beziehung verknüpfte Tabellen aufteilen

Wenn wir eine 1:1-Beziehung zwischen zwei Tabellen herstellen wollen, die zuvor in einer einzigen Tabelle enthalten waren gehen wir wie folgt vor. Als Beispiel verwenden wir eine Mitarbeiter-Tabelle, die ein Feld namens **Gehalt** aufweist, das wir in eine eigene Tabelle auslagern wollen (siehe Bild 2).

Die Daten sehen aktuell wie in Bild 3 aus.

Nun sind folgende Schritte notwendig:

- Erstellen einer neuen Tabelle namens **tblGehaelter** mit den Feldern **MitarbeiterID** und **Gehalt**

- Umbenennen der Tabelle **tblMitarbeiterMitGehalt** in **tblMitarbeiter**

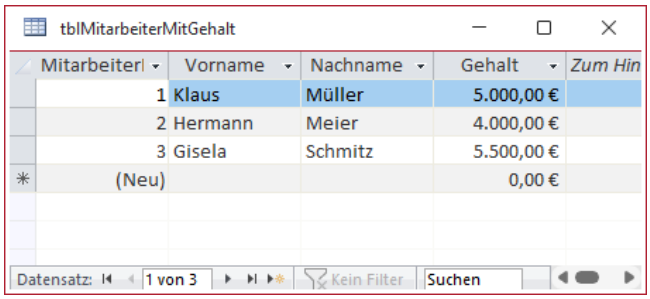

Bild 3: Daten der aufzuteilenden Tabelle

- Herstellen einer 1:1-Beziehung zwischen den Tabellen **tblMitarbeiter** und **tblGehaelter**
- Übertragen der Werte des Feldes **Gehalt** der Tabelle **tblMitarbeiter** in das gleichnamige Feld der Tabelle **tblGehaelter**
- Löschen des Feldes **Gehalt** aus der Tabelle **tblMitarbeiter**

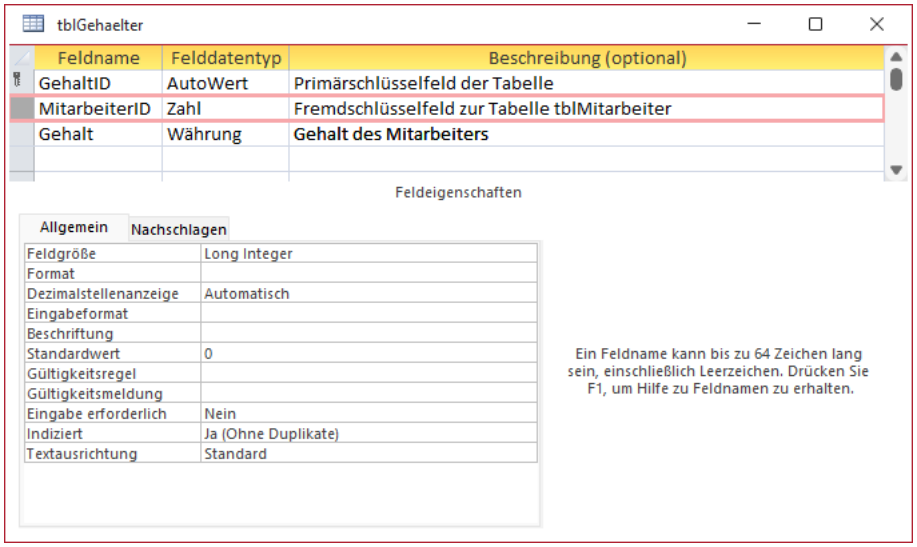

Bild 4: Tabelle mit Primärschlüsselfeld und Fremdschlüsselfeld

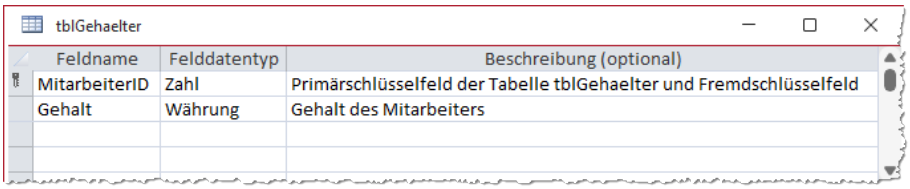

Bild 5: Tabelle mit Primärschlüsselfeld als Fremdschlüsselfeld

Neue Tabelle für die Gehälter erstellen

Beim Erstellen einer Tabelle, welche die Zusatzinformationen für eine Tabelle liefert, stellt sich eine Frage: Fügen wir dieser Tabelle einen eigenen Primärschlüssel hinzu und ein Fremdschlüsselfeld zur Zuordnung des Mitarbeiters oder soll das Primärschlüsselfeld gleichzeitig das Fremdschlüsselfeld sein?

Technisch ist es nicht relevant, zumal bei der Variante mit dem Fremdschlüsselfeld das Fremdschlüsselfeld ohnehin als eindeutiger Index definiert werden muss. Die Variante mit dem eigenen Primärschlüsselfeld finden wir in Bild 4. Das Primärschlüsselfeld wird wie gewöhnlich als Autowert ausgelegt. Für das Fremdschlüsselfeld stellen wir die Eigenschaft **Indiziert** auf **Ja (Ohne Duplikate)** ein und legen so einen eindeutigen Index für dieses Feld fest.

Wollen wir ein Feld einsparen, können wir die Variante aus Bild 5 nutzen. Hier ist das Feld **MitarbeiterID** als Primärschlüsselfeld definiert, aber nicht mit dem Datentyp **Autowert**, sondern **Zahl**. Durch die Eigenschaft als Primärschlüsselfeld ist das Feld automatisch als eindeutiges Feld definiert. Wir arbeiten im Folgenden mit der dieser Variante weiter.

Umbenennen der Tabelle tblMitarbeiterMitGehalt in tblMitarbeiter

Dieser Schritt ist schnell erledigt: Wir brauchen nur die Tabelle zu schließen, falls sie geöffnet ist, und benennen sie im Navigationsbereich von **tblMitarbeiterMitGehalt** in **tblMitarbeiter** um.

In der Beispieldatenbank haben wir einfach eine Kopie der Tabelle unter diesem Namen erstellt, damit die alte Tabelle zu Ansichtszwecken noch vorhanden ist.

Herstellen einer 1:1-Beziehung zwischen den Tabellen tblMitarbeiter und tblGehaelter

Nun öffnen wir mit dem Ribbon-Befehl **Datenbanktools|Beziehungen|Beziehungen** das **Beziehungen**-Fenster und ziehen die beiden Tabellen **tblMitarbeiter** und **tblGehaelter** hinein.

Hier haben wir zwei Möglichkeiten, um eine Beziehung zwischen dem Feld **MitarbeiterID** der Tabelle

tblMitarbeiter und dem Feld **MitarbeiterID** der Tabelle **tblGehaelter** herzustellen:

- Wir ziehen das Feld **MitarbeiterID** aus der Tabelle **tblMitarbeiter** auf das Feld **Mitarbeiter** der Tabelle **tblGehaelter** oder
- wir ziehen das Feld **MitarbeiterID** aus der Tabelle **tblGehaelter** auf das Feld **MitarbeiterID** der Tabelle **tblMitarbeiter**.

Man könnte meinen, es wäre das Gleiche, aber das ist mitnichten der Fall. Der Unterschied zeigt sich erst, wenn man eine Abfrage anlegt, um die Inhalte der beiden Tabelle in einem zu bearbeiten. Wir gehen beide Wege durch.

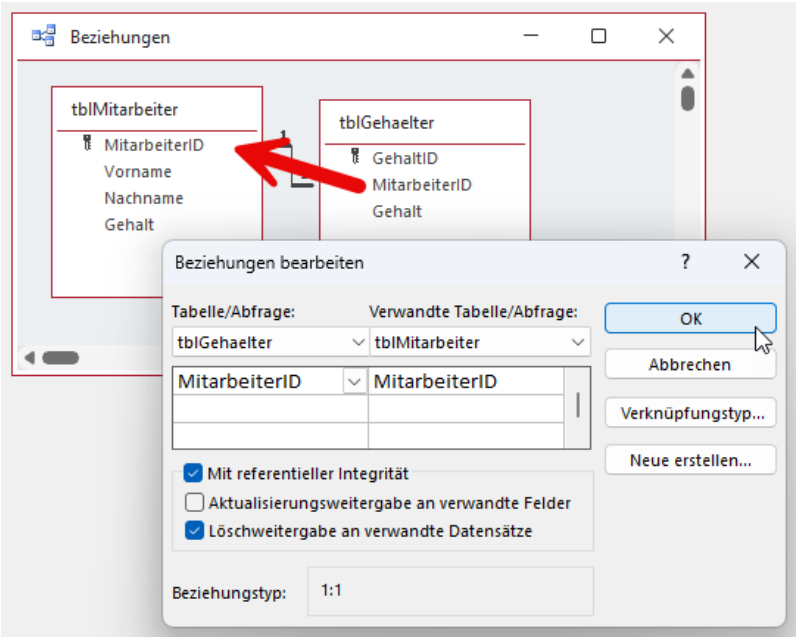

Bild 6: Herstellen der 1:1-Beziehung von **tblGehaelter** nach **tblMitarbeiter**

Beziehung herstellen von tblGehaelter nach tblMitarbeiter

Im ersten Beispiel ziehen wir den Beziehungspfeil von **tblGehaelter** nach **tblMitarbeiter** und legen dann für die Beziehung referentielle Integrität fest (siehe Bild 6). Dadurch steht im Dialog **Beziehungen bearbeiten** die Tabelle **tblGehaelter** unter **Tabelle/Abfrage** und **tblMitarbeiter** unter **Verwandte Tabelle/Abfrage**.

Wenn wir die Beziehung nun speichern wollen, erhalten wir die Fehlermeldung aus Bild 7. Was bedeutet das genau? Dazu müssen wir verstehen, was wir hier im Vergleich zu einer herkömmlichen 1:n-Beziehung erzeugt haben. Wir haben hier nämlich das Feld **MitarbeiterID** der Tabelle **tblMitarbeiter** als Fremdschlüsselfeld definiert und das Feld **MitarbeiterID** der Tabelle **tblGehaelter** nimmt hier die Rolle ein, die normalerweise das Primärschlüsselfeld in einer Beziehung einnimmt.

Diese Beziehung können wir so definieren, allerdings dürfen wir für das Feld **MitarbeiterID** der Tabelle **tblMitarbeiter** nur solche Werte anlegen, die bereits im Feld **MitarbeiterID** der Tabelle **tblGehaelter** enthalten sind. Das wollen wir nicht, also verwerfen wir diese Variante direkt.

Beziehung herstellen von tblMitarbeiter nach tblGehaelter

Nun schauen wir uns dem umgekehrten Fall an. Wir fügen die Beziehung also nun hinzu, indem wir das

Bild 7: Fehler beim Herstellen der 1:1-Beziehung

Tabellen [basics]: Anlagefelder

Ein spannender Felddatentyp sind die Anlagefelder. Sie ermöglichen eine viel einfachere Handhabung zum Speichern von Dateien in einem Feld einer Tabelle. Wir können Dateien damit über die Benutzeroberfläche in einem Datensatz einer Tabelle speichern, diesen wieder im Dateisystem speichern und wieder aus der Tabelle entfernen. Für Bilddateien offerieren Anlagefelder sogar noch eine weitere Funktion, denn diese können wir über ein entsprechendes Steuerelement im Formular anzeigen. Früher haben wir Dateien, wenn überhaupt, im OLE-Feld einer Tabelle gespeichert, aber der Umgang mit dem Anlagefeld ist wesentlich einfacher. In diesem Artikel schauen wir uns das Anlagefeld genauer an.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **TabellenBasics_Anlagefelder.accdb**.

Einsatzzwecke von Anlagefeldern

Anlagefelder können wir für verschiedene Zwecke nutzen, zum Beispiel:

- Dokumentenmanagement: Wir können das Anlage-Feld verwenden, um Dokumente, Berichte, Verträge und andere wichtige Dateien in einer Datenbank zu speichern und zu organisieren. Dies erleichtert die Suche und den Zugriff auf diese Dokumente im Zusammenhang mit den entsprechenden Datensätzen.
- Bilder und Fotos: Das Anlage-Feld ist besonders nützlich, wenn wir Bilder oder Fotos in Ihrer Datenbank speichern möchten. Dies kann in Anwendungen wie Fotogalerien, Produktkatalogen oder Kundenprofilen hilfreich sein.
- Bei der Datenverarbeitung anfallende Dokumente: In einer Rechnungsverwaltung können wir die als PDF erzeugten Rechnungen einfach in der Tabelle mit den Rechnungsdaten speichern.

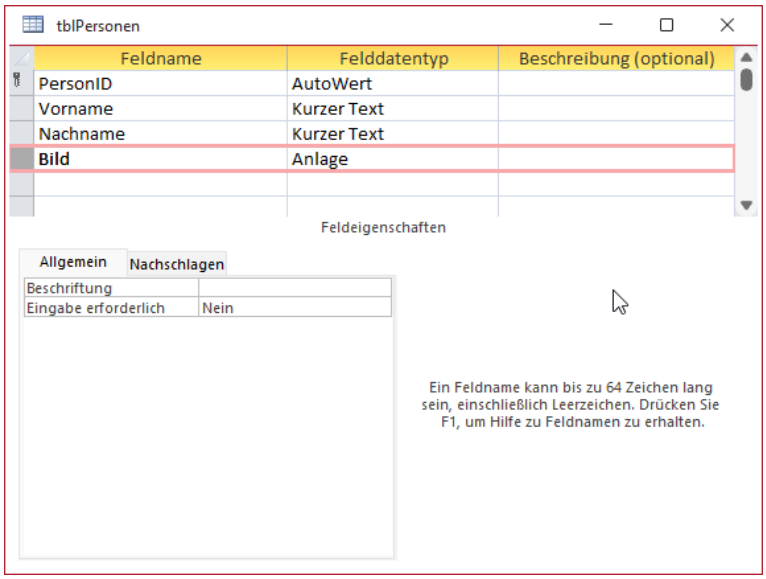

Bild 1: Ein Anlagefeld im Entwurf

Grenzen von Anlagefeldern

Auch wenn Anlagefelder im Gegensatz zu den früher verwendeten OLE-Feldern die Dateien nicht in einem speziellen Format speichern, was erheblich mehr Speicher benötigt hat als die Dateien selbst hatten, gibt es natürlich Grenzen für das Speichern von Dateien: nämlich die Zwei-Gigabyte-Größe einer Datenbank. Wenn wir planen, größere Mengen von Daten in einer Datenbank zu speichern, sollten wir also rechtzeitig darüber nachdenken, solche Dateien direkt in einem eigenen Backend zu hinterlegen.

Anlagefeld anlegen

Das Anlegen eines Anlagefeldes ist so einfach wie bei jedem anderen Feld – in diesem

Fall soll es Bild heißen und in einer Tabelle namens **tblPersonen** landen (siehe Bild 1). Wie wir sehen, hat ein Anlagefeld wesentlich weniger Eigenschaften als die Felder mit den übrigen Felddatentypen.

Wechseln wir in die Datenblattansicht, sehen wir das Anlagefeld wie in Bild 2. Das erste, was auffällt, ist die durch ein Büroklammer-Symbol ersetzte Spaltenüberschrift. Im Feld selbst sehen wie das gleiche Büroklammer-Symbol sowie die Anzeige der Zahl **0** in Klammern.

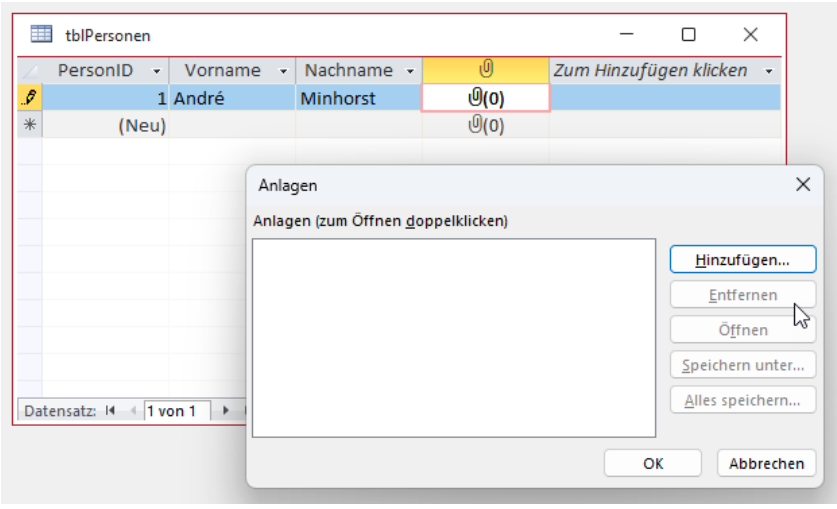

Bild 2: Ein Anlagefeld in der Datenblattansicht

Dies gibt an, dass das Feld derzeit noch keine Datei enthält. Gleichzeitig deutet dies darauf hin, dass wir hier nicht nur eine, sondern auch mehrere Dateien hinterlegen können. Klicken wir doppelt in das Feld, öffnet sich der Dialog **Anlagen**. Hier ist aktuell nur die Schaltfläche **Hinzufügen...** aktiviert. Die anderen sind deaktiviert, da sich noch keine Datei im Anlagefeld befindet.

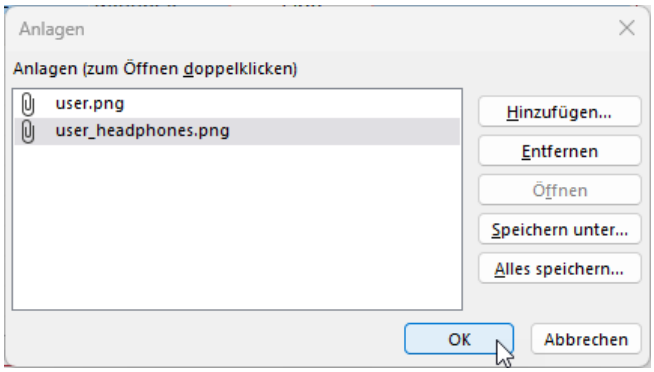

Bild 3: Hinzufügen von Bildern

Anlage hinzufügen

Klicken wir auf **Hinzufügen...**, erscheint der übliche Dateiauswahl-Dialog. Hier können wir ein oder mehrere Dateien auswählen, die zum Anlagefeld hinzugefügt werden sollen. Im Beispiel haben wir Bilddateien ausgewählt. Der Dialog sieht anschließend wie in Bild 3 aus.

- **Hinzufügen...**: Erlaubt das Hinzufügen weiterer Dateien
- **Entfernen**: Entfernt die markierte Datei

Nach dem Schließen des Dialogs mit der **OK**-Schaltfläche ändert sich die Darstellung im Anlagefeld, das nun den Wert **2** anzeigt (siehe Bild 4). Hier sehen wir auch, dass wir den Dialog zum Verwalten von Anlagen auch per Kontextmenü aufrufen können.

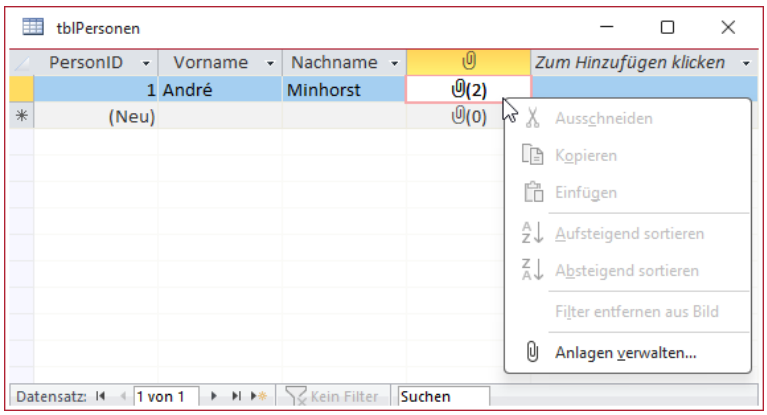

Bild 4: Anlagefeld mit zwei Anlagen

Anlagen verwalten

Öffnen wir den Dialog erneut und markieren einen der Einträge, sind nun alle Schaltflächen markiert:

Tabellen [basics]: Datum/Zeit oder Datum und Zeit

Microsoft Access bietet verschiedene Möglichkeiten, um Datums- und Uhrzeitangaben zu speichern. Da der entsprechende Datentyp im Tabellenentwurf Datum/Uhrzeit heißt, gehen wir erst einmal von der Intention aus, das beides im gleichen Feld gespeichert wird. Deshalb hat Microsoft auch eine spezielle Art vorgesehen, wie es Datums- und Uhrzeitangaben intern speichert – nämlich als Double-Wert, bei dem der ganzzahlige Anteil angibt, wie viele Tage seit dem 30.12.1899 vergangen sind und bei dem der Teil hinter dem Komma die Uhrzeit definiert. Nun stellt sich allerdings eine Frage: Kann man eigentlich Datum und Uhrzeit auch in jeweils einem eigenen Feld speichern und wann macht das im Vergleich zum Speichern beider Informationen in einem einzigen Feld überhaupt Sinn?

Beispieldatenbank

Die Beispiele dieses Artikels findest Du in der Datenbank **TabellenBasics_DatumZeitOderDatumUndZeit.accdb**.

Die Beispieldatenbank enthält eine Tabelle namens **tblDatumUndZeit**, die ein Feld namens **DatumUndZeit** zum Speichern von Datums- und Zeitangabe in einem Feld sowie zwei Felder namens **NurDatum** und **NurZeit**, die jeweils für das Speichern von Datum und Uhrzeit vorgesehen sind (siehe Bild 1).

Aktuelles Datum und Uhrzeit schnell eintragen

Falls Du diesen Trick noch nicht kennst: Es gibt zwei Tastenkombinationen, mit denen Du das aktuelle Datum und die aktuelle Uhrzeit in Access-Elemente wie Tabellenfelder oder Steuerelemente einfügen.

Mit der Tastenkombination **Strg + Umschalt + Komma** fügst Du das Datum zu dem Element hinzu, das aktuell den Fokus hat.

Mit der Tastenkombination **Strg + Umschalt + Punkt** fügst Du die aktuelle Uhrzeit zum Element mit dem Fokus hinzu.

Wenn wir schnell das aktuelle Datum und die aktuelle Uhrzeit in ein Feld einfügen wollen, könnten wir versucht sein, diese beiden Tastenkombinationen hintereinander auszuführen. Das führt allerdings zum folgenden Ergebnis:

```
30.11.202310:54:06
```

Wir müssen also dazwischen noch ein Leerzeichen einfügen. Eine Tastenkombination zum gleichzeitigen Einfügen von Datum und Uhrzeit gibt es unseres Wis-

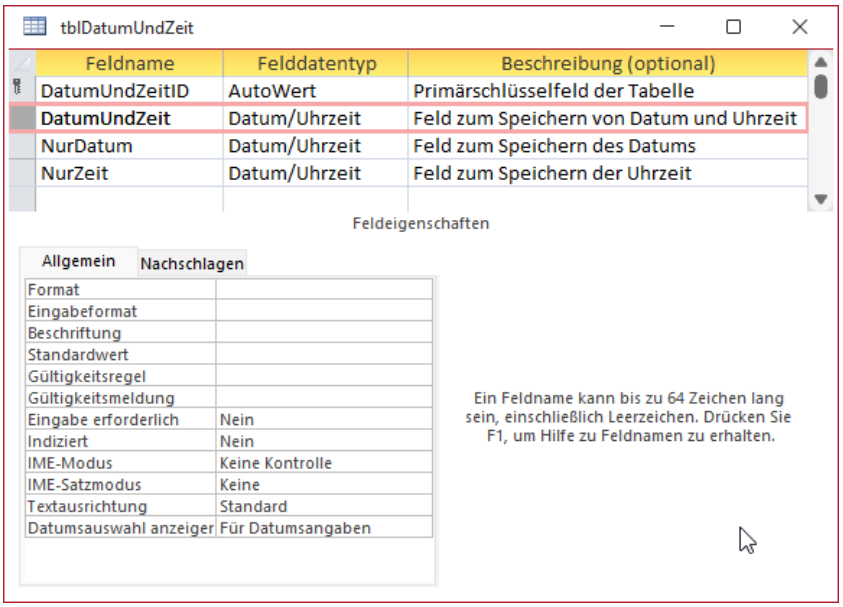

Bild 1: Tabelle mit Feldern für Datum/Uhrzeit, Datum und Uhrzeit

sens nach nicht. Außerdem funktionieren die Tastenkombinationen nicht in anderen Ansichten wie der Entwurfsansicht – hier müssen wir Datum und Uhrzeit manuell eingeben.

Datum/Zeit, Datum und Zeit in eigene Felder

Wenn wir nun Daten und Zeiten in die Tabelle **tblDatumUndZeit** eingeben, in der wir eigentlich ein Feld für Datum und Uhrzeit und jeweils eines für das Datum und die Uhrzeit vorgesehen haben, stellen wir schnell fest: Unsere Beschriftungen werden den Benutzer nicht davon abhalten, seine Daten und Zeiten auch mal in das falsche Feld zu füllen (siehe Bild 2).

Ein Datumsfeld hat eben intern den Datentyp **Double** und zeigt seine Daten nur in der entsprechenden Formatierung an. Noch einmal kurz zur Erinnerung die Art, wie Access Datum- und Zeit intern speichert: Dort landet das, was wir als **1.1.2024 12:00:00** eingeben, in einer **Double**-Zahl. Der Teil vor dem Komma entspricht der Anzahl der Tage zwischen dem eingegebenen Datum und dem **30.12.1899** und der Teil nach dem Komma entspricht dem Anteil der eingegebenen Uhrzeit an 24 Stunden. 12:00 Uhr entspricht also **0,5**, 6:00 Uhr ist **0,25** und 18:00 Uhr wäre **0,75**. Bei den Tagen würde das Datum **30.11.2023** dem Zahlenwert **45.260** entsprechen – also sind vom **30.12.1899** bis zum **30.11.2023** genau **45.260** Tage vergangen.

Und der **30.11.2023, 12:00 Uhr** würde folgerichtig dem Zahlenwert **45.260,5** entsprechen – und dieser Wert wird in der Tabelle intern abgespeichert. Dadurch können wir nun nur scheinbar Informationen wie reine Datumsangaben ohne Uhrzeit und reine Uhrzeiten ohne Datum speichern. Wenn wir beispielsweise das Datum **30.11.2023** in einem **Datum/Uhrzeit**-Feld speichern, landet dort der intern der Wert **45.260,0** – also eigentlich **30.11.2023 0:00:00**.

Und wenn wir in einem **Datum/Uhrzeit**-Feld den Wert **12:00:00** eingeben, dann landet zwar intern der Wert **0,5** in diesem Feld. Allerdings entspricht dies eigentlich dem Datum **30.11.1899 12:00:00**.

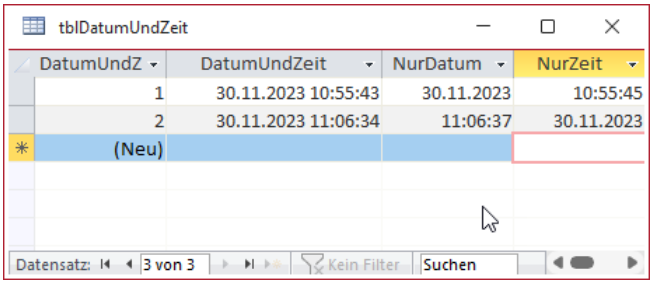

Bild 2: Tabelle mit Feldern für Datum/Uhrzeit, Datum und Uhrzeit in der Datenblattansicht

Access schneidet allerdings bei der Standardformatierung das Datum weg, wenn intern vor dem Komma der Wert **0** steht und es schneidet die Uhrzeit weg, wenn hinter dem Komma der Wert **0** steht.

Was geschieht folgerichtig, wenn wir das Datum **30.12.1899** eingeben? Dann erscheint lediglich die Uhrzeit **00:00:00** in dem Datumsfeld.

Nur Datum erlauben

Wie also können wir den Benutzer dazu bringen, dass er nur das Datum in ein Feld eingibt und er nicht versehentlich die Uhrzeit hinzufügt oder sogar nur die Uhrzeit eingibt?

Dazu können wir die beiden Eigenschaften **Gültigkeitsregel** und **Gültigkeitsmeldung** nutzen. Aber wie formulieren wir die Gültigkeitsregel, um die Eingabe von Uhrzeiten auszuschließen? Beginnen wir mit dem, was wir sicherstellen wollen: Wir wollen eine reine Datumsangabe, das bedeutet, dass unser intern gespeicherter **Double**-Wert keine Nachkommastellen haben darf. Wie können wir das in der Gültigkeitsregel formulieren? Wir prüfen einfach, ob der eingegeben Wert dem eingegebenen Wert ohne Nachkommastellen entspricht. Nur den Wert ohne Nachkommastellen einer Zahl ermitteln wir mit der Funktion **Int**. Sie liefert beispielsweise für den Wert **1,5** den Wert **1**, schneidet als die Nachkommastellen einfach ab. Also vergleichen wir im Ausdruck für die Gültigkeitsregel einfach den tatsächlichen Wert und den Wert, der nur die Stellen vor dem Komma enthält. Wir legen dafür also den folgenden Ausdruck fest:

```
=Int([NurDatum])
```

Wichtig: Der Feldname muss in eckigen Klammern eingefasst sein, weil er sonst in Anführungszeichen gesetzt wird. Nur wenn diese Regel erfüllt ist, wird keine Gültigkeitsmeldung angezeigt. Anderenfalls erscheint die Meldung, die wir wie in Bild 3 für die Eigenschaft **Gültigkeitsmeldung** hinterlegt haben.

Dies sieht beispielsweise wie in Bild 4 aus.

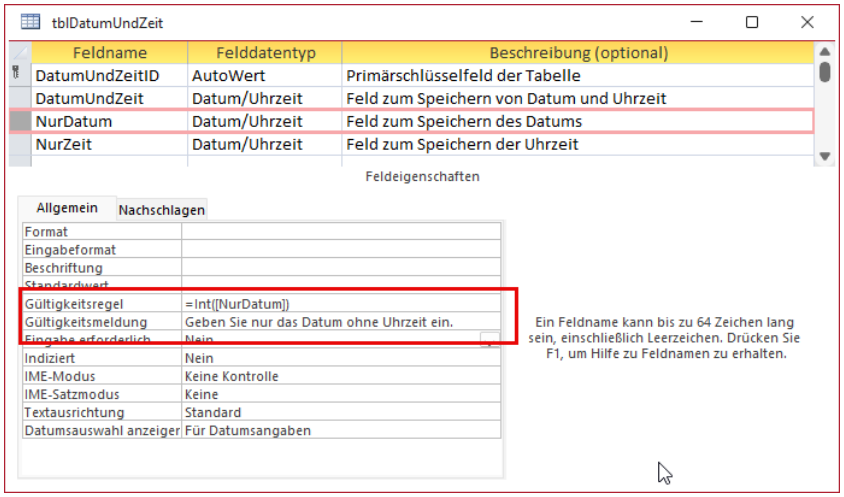

Bild 3: Gültigkeitsregel und Gültigkeitsmeldung, welche die Eingabe von Uhrzeitanteilen verhindern sollen.

Nur Uhrzeit erlauben

Wenn wir nur die Eingabe einer Uhrzeit erlauben wollen, müssten wir analog prüfen, ob der Benutzer einen Wert eingegeben hat, der intern als eine Zahl gespeichert wird, die den Wert **0** vor dem Komma aufweist. Das ist ein klein wenig komplizierter als beim Datum, aber auch leicht umzusetzen: Wir müssen die Gültigkeitsmeldung so formulieren, dass der enthaltene Wert gleich dem enthaltenen Wert minus dem Datum ist.

Die Gültigkeitsmeldung soll also angezeigt werden, wenn der eingegebene Wert minus dem ganzzahligen Anteil nicht dem eingegebenen Wert entspricht. Dies formulieren wir wie in Bild 5:

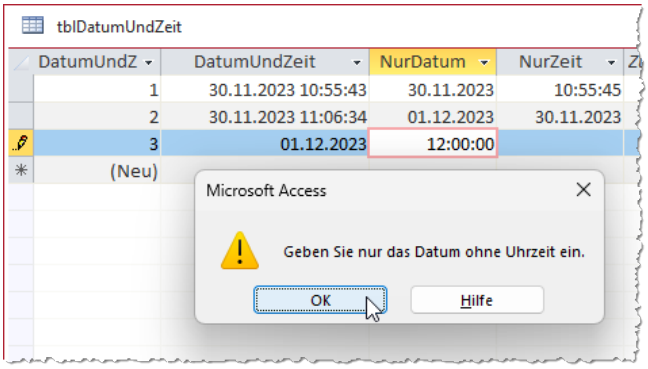

Bild 4: Meldung bei der Eingabe eines Datums mit Uhrzeit oder auch nur einer Uhrzeit

```
=[NurZeit]-Int([NurZeit])
```

Wir ziehen also von einem Wert wie **1.1.2024 12:00:00**, was der Zahl **45.292,5** entspricht, den Wert **45.292** ab und prüfen, ob das Ergebnis dem Wert entspricht, was hier offensichtlich nicht der Fall ist.

Geben wir also eine Uhrzeit ein, die auch noch ein Datum enthält, erscheint eine Meldung wie in Bild 6.

So können wir also sicherstellen, dass nur reine Datums- oder Uhrzeitangaben in den dafür vorgesehenen Feldern der Tabelle landen.

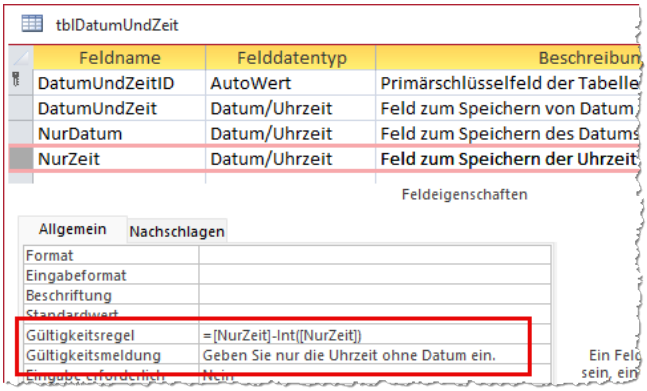

Bild 5: Gültigkeitsregel und Gültigkeitsmeldung, welche die Eingabe von Datumsanteilen verhindern sollen.

Abfragen [basics]: Daten zusammenfassen mit UNION

Die meisten Abfragetypen unter Access lassen sich in der Entwurfsansicht zusammenstellen. Einer der Exoten, für die der Entwurf keine Funktion bietet, ist die UNION-Abfrage. Warum ist das so? Weil UNION-Abfragen gleich mehrere Abfragen miteinander verbinden, und das lässt sich in der Entwurfsansicht schlicht nicht abbilden. Allerdings gibt es neben der Entwurfsansicht und der Datenblattansicht noch eine dritte Ansicht für Abfragen, nämlich die SQL-Ansicht. Diese lässt sich zu Lernzwecken übrigens auch nutzen, um einmal den SQL-Code herkömmlicher Abfragen zu untersuchen. In diesem Artikel geht es allerdings erst einmal um die UNION-Abfrage: Sie verbindet zwei oder mehr Abfragen mit einander, wobei für diese Abfragen bestimmte Regel gelten – sie müssen zum Beispiel die gleiche Anzahl Felder haben. Alles Wichtige zu UNION-Abfragen liefern die folgenden Seiten.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **AbfragenBasics_Unionabfragen.accdb**.

Einsatzzwecke von UNION-Tabellen

UNION-Tabellen setzen wir immer dann ein, wenn wir die Daten aus mehr als einer Tabelle in einer Abfrage zusammenführen wollen. Vielleicht denkst Du jetzt, dass man das ja auch ohne **UNION**-Abfragen machen kann – zum Beispiel, indem man die Kunden-Tabelle und die Anreden-Tabelle in eine Abfrage zieht, um die Kunden samt Anreden darzustellen.

Das ist jedoch nicht der Fall, für den **UNION**-Tabellen vorgesehen sind: Sie sollen nicht die Daten verknüpfter Tabellen in einem Datensatz darstellen, sondern sie sollen gleichartige Daten aus verschiedenen Tabellen zusammenführen, und zwar so, dass im einfachsten Fall alle Datensätze der ersten Tabelle und alle Datensätze der zweiten Tabelle untereinander dargestellt werden.

Wozu sollte man das nutzen? Macht es nicht viel mehr Sinn, solche Daten direkt in einer einzigen Tabelle zu speichern, statt diese auf zwei oder mehr Tabellen zu verteilen und diese anschließend mit einer UNION-Abfrage wieder zusammenzuführen? Nein, denn wir beziehen uns hier auf solche Fälle, wo die Daten aus gutem Grund auf mehrere Tabellen aufgeteilt wurden. Zum Beispiel kann es sein, dass wir eine Datenbank verwenden, um Transaktionen auf einer Handelsplattform zu verwalten. Dort gibt es zu jeder Transaktion einen Käufer und einen Verkäufer. Diese sind in verschiedenen

tblKaeufer

KaeuferID	Vorname	Nachname	AnredeID	Strasse	PLZ
1	Adi	Stratmann	Herr	Kremser Straß	10589
2	Heidi				
3	Wernfrie				
4	Vitus				
5	Jadwiga				
6	Niko				
7	Siegert				
8	Michl				
9	Florentiu				
10	Gernulf				
11	Tristan				
12	Heinfried				
13	Herma				
14	Mina				
15	Jo				
16	Heimbert				
17	Roderich				

tblVerkaeufer

VerkaeuferI	Vorname	Nachname	AnredeID
1	Miriam	Dill	Frau
2	Gislind	Claßen	Frau
3	Hildegardt	Erhardt	Frau
4	Sibyl	Spitzer	Frau
5	Stella	Heck	Frau
6	Giselbert	Kroll	Herr
7	Siegrid	Siefert	Frau
8	Traudel	Dröge	Frau
9	Louisa	Wachter	Frau
10	Schorsch	Bloch	Herr
11	Reinholdine	Urselmann	Frau
12	Dorlies	Wischnewski	Frau
13	Helbert	Matz	Herr

Bild 1: Formular-Entwurf unseres Beispielformulars

Tabellen namens **tblKaeufer** und **tblVerkaeufer** gespeichert (siehe Bild 1).

Das kann man so machen, gerade wenn man aus Performancegründen nicht alle Daten in einer Tabelle speichern möchte. Nun gibt es jedoch Fälle, in denen man dennoch bestimmte Daten von Käufern und Verkäufern in einer einzigen Abfrage sehen möchte – beispielsweise, um allen eine Information über eine neue Funktion der Handelsplattform zu übermitteln.

Genau das ist der Einsatzzweck für eine UNION-Abfrage.

SQL-Ansicht aktivieren

Wenn wir eine neue, leere Abfrage erstellt haben, können wir von der Entwurfsansicht aus zur SQL-Ansicht wechseln. Dazu klicken wir oben links im Ribbon-Tab **Abfrageentwurf** auf die Schaltfläche **SQL** (siehe Bild 2).

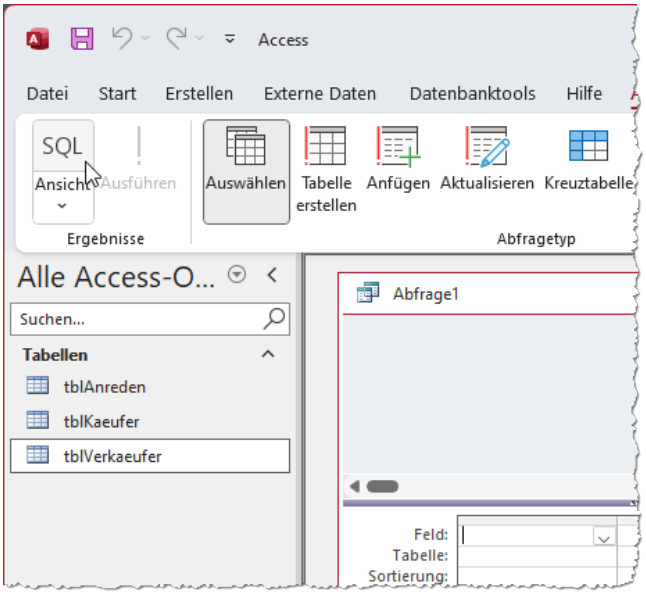

Bild 2: Wechseln zur SQL-Ansicht einer Abfrage

Damit aktivieren die SQL-Ansicht, die ziemlich karg daherkommt (siehe Bild 3).

Bild 3: SQL-Ansicht einer Abfrage

Hier würden wir nun normalerweise die bereits vorgegebene **SELECT**-Anweisung ergänzen. Allerdings müssten wir nun die Liste aller Felder, die Quelltabellen und gegebenenfalls noch weitere Daten eintragen. Einen Großteil davon können wir uns sparen.

UNION-Abfrage vorbereiten

Dazu wechseln wir zurück zur Entwurfsansicht und ziehen dort die erste der beiden Tabellen, die als Quelle der **UNION**-Abfrage dienen sollen, in den Entwurf. Aus dieser ziehen wir alle Felder, die in der **UNION**-Abfrage landen sollen, in das Entwurfsras-

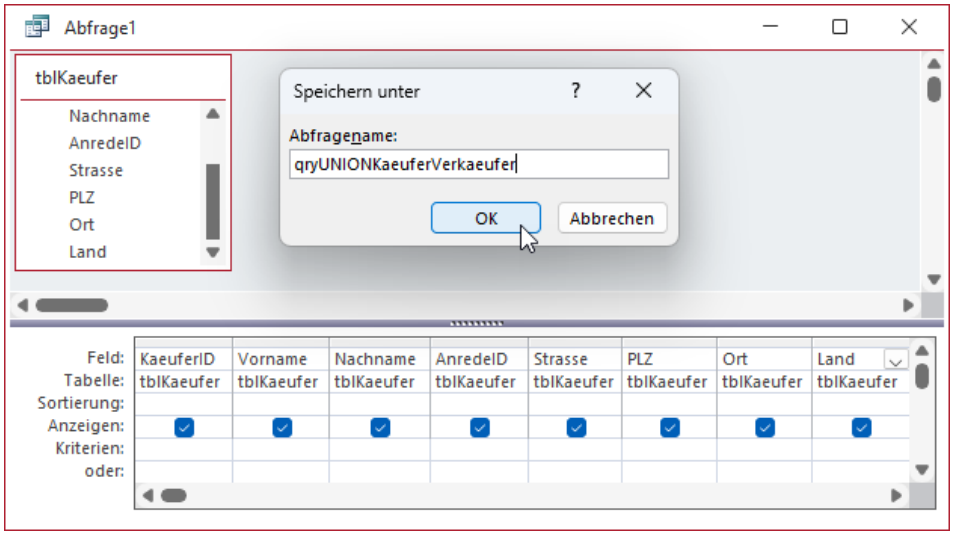

Bild 4: Vorbereiten der **UNION**-Abfrage

ter. Außerdem speichern wir die Abfrage gleich unter dem Namen **qryUNIONKaeuferVerkaeufer** (siehe Bild 4).

Wechseln wir nun erneut in die SQL-Ansicht der Abfrage, finden wir die **SELECT**-Abfrage aus Bild 5 vor.

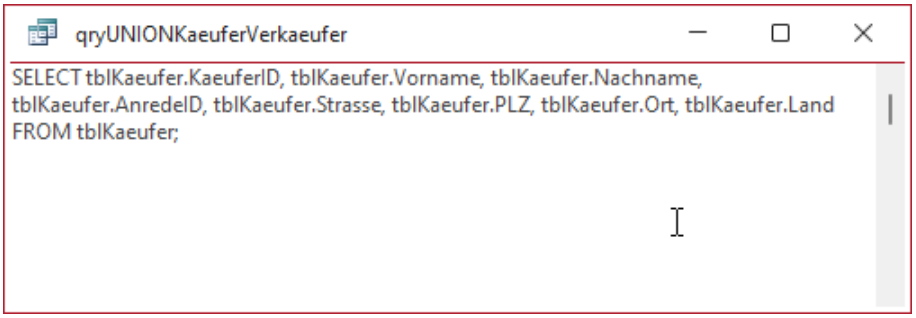

Bild 5: Erster Teil der **UNION**-Abfrage

Hier werden nach dem **SELECT**-Schlüsselwort alle Felder samt Tabelle wie **tblKaeufer.KaeuferID** aufgelistet, dann folgen das Schlüsselwort **FROM** und der Name der Quelltabelle für diese Abfrage.

Das ist eine herkömmliche Abfrage, und um diese nun in eine **UNION**-Abfrage zu verwandeln, benötigen wir zuerst noch die gleiche Abfrage für die Tabelle **tblVerkaeufer**. Wir können nun den Code der ersten SQL-Abfrage kopieren, eine neue Abfrage mit der Tabelle **tblVerkaeufer** erstellen, dort in die SQL-Ansicht wechseln und dann die erste Abfrage an die neue Abfrage anhängen – getrennt vom Schlüsselwort **UNION**.

Da beide Tabellen gleich aufgebaut sind, können wir aber auch den Code der hier vorliegenden Abfrage einfach in die Zwischenablage kopieren, das Schlüsselwort **UNION** anhängen und die Abfrage ein zweites Mal hinter **UNION** einfügen.

Gegebenenfalls muss noch das Semikolon hinter der ersten Abfrage entfernt werden. Dann brauchen wir nur noch in der eingefügten Version alle Vorkommen des Tabellennamens **tblKaeufer** durch **tblVerkaeufer** zu ersetzen. Außerdem müssen wir im zweiten Teil noch **KaeuferID** durch **VerkaeuferID** ersetzen:

```
SELECT tblKaeufer.KaeuferID, tblKaeufer.Vorname,
    tblKaeufer.Nachname, tblKaeufer.AnredeID,
    tblKaeufer.Strasse, tblKaeufer.PLZ,
    tblKaeufer.Ort, tblKaeufer.Land
FROM tblKaeufer
UNION
SELECT tblVerkaeufer.VerkaeuferID, tblVerkaeufer.Vorname,
    tblVerkaeufer.Nachname, tblVerkaeufer.AnredeID,
    tblVerkaeufer.Strasse, tblVerkaeufer.PLZ,
    tblVerkaeufer.Ort, tblVerkaeufer.Land
FROM tblVerkaeufer
```

Bevor wir uns das Ergebnis anschauen, speichern und schließen wir die Abfrage und sehen dann im Navigationsbereich, wie sich eine **UNION**-Abfrage von anderen Abfragen unterscheidet (siehe Bild 6) – hier finden wir ein anderes Icon vor als üblich.

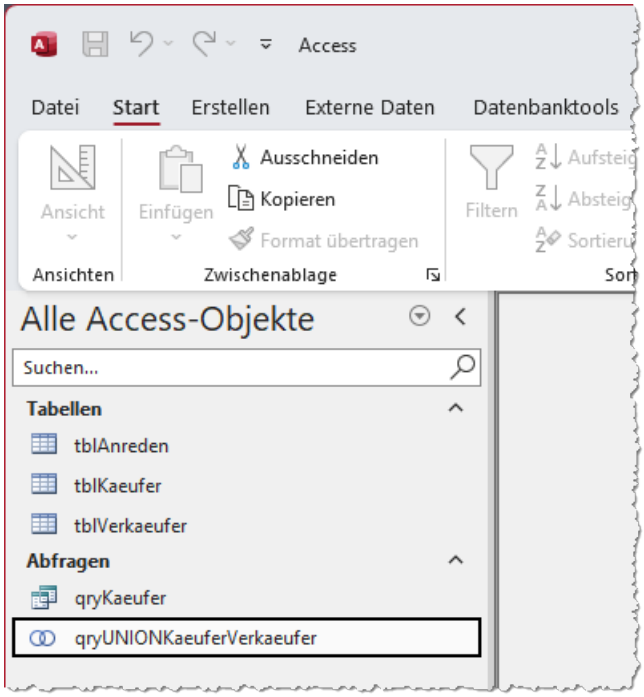

Bild 6: UNION-Abfrage im Navigationsbereich

Eingebaute Funktionen: Rund um Datum und Uhrzeit

Access stellt eine ganze Reihe von Funktionen zur Verfügung, mit denen wir mit Datums- und Uhrzeitangaben arbeiten können. Damit lassen sich das aktuelle Datum und die aktuelle Uhrzeit ermitteln, mit Datum und Uhrzeit rechnen, Bestandteile wie Tage, Monate, Jahre, Stunden, Minuten und Sekunden auslesen oder auch die Namen des aktuellen Wochentags oder Monats ermitteln. Welche Funktionen es gibt und wie wir damit in Eigenschaften von Tabellen, Abfragen, Formularen, Berichten und Steuerelementen arbeiten können, zeigen wir in diesem Artikel.

Beispieldatenbank

Die Beispiele dieses Artikels findest Du in der Datenbank **EingebauteFunktionen_RundUmDatumUnd-Uhrzeit.accdb**.

Funktionen für Datum und Uhrzeit

Nachfolgend listen wir zur besseren Übersicht zunächst einmal alle Datums- und Uhrzeitfunktionen, die wir in diesem Artikel behandeln, in alphabetischer Reihenfolge auf.

Wir geben wieder die deutsche Bezeichnung zuerst an, gefolgt von der englischen Bezeichnung, die wir auch in VBA verwenden können.

Die deutsche Bezeichnung ist notwendig, weil diese Funktionen, wenn wir sie über die Benutzeroberfläche der deutschsprachigen Version von Access eingeben, in die deutsche Version umgewandelt werden – selbst wenn wir die englische Version angeben:

- **DatAdd** (**DateAdd**): Addiert einem Teil des Datums, zum Beispiel Tag, Monat oder Jahr, einen bestimmten Wert hinzu oder zieht diesen ab.
- **DatDiff** (**DateDiff**): Ermittelt die Differenz zwischen zwei Datumsangaben, gegebenenfalls mit Uhrzeit, für das angegebene Intervall.
- **DatSeriell** (**DateSerial**): Erwartet drei Zahlenwerte für Tag, Monat und Jahr und fügt diese zu einem gültigen Datumswert zusammen.
- **DatTeil** (**DatePart**): Ermittelt den per Parameter angegebenen Teil eines Datums, zum Beispiel Tag, Monat oder Jahr.
- **Datum** (**Date**): Ermittelt das aktuelle Datum als Variant-Wert des Typs Datum.
- **Datum$** (**Date$**): Ermittelt das aktuelle Datum als String-Wert.
- **DatWert** (**DateValue**): Gibt den Datumsanteil einer Datumsangabe zurück. Die eventuell vorhandene Angabe der Uhrzeit wird abgeschnitten.
- **IstDatum** (**IsDate**): Erwartet eine Datumsangabe und gibt den Wert **Wahr**/**True** zurück, wenn diese gültig ist.
- **Jahr** (**Year**): Ermittelt die Jahre des angegebenen Datums.
- **Jetzt** (**Now**): Ermittelt das aktuelle Datum mit Uhrzeit.
- **Minute** (**Minute**): Ermittelt die Minuten des angegebenen Datums.
- **Monat** (**Month**): Ermittelt die Monate des angegebenen Datums.
- **Monatsname** (**Monthname**): Ermittelt den Namen des Monats des übergebenen Datums.

- **Sekunde** (**Second**): Ermittelt die Sekunden des angegebenen Datums.
- **Stunde** (**Hour**): Ermittelt die Stunden des angegebenen Datums.
- **Tag** (**Day**): Ermittelt die Tage des angegebenen Datums.
- **Wochentag** (**Weekday**): Ermittelt den Namen des Wochentags des angegebenen Datums.
- **Wochentagsname** (**WeekdayName**):
- **Zeit** (**Time**): Ermittelt die aktuelle Uhrzeit als Variant-Wert des Typs Datum.
- **Zeit$** (**Time$**): Ermittelt die aktuelle Uhrzeit als String-Wert.
- **ZeitSeriell** (**TimeSerial**): Erwartet drei Zahlenwerte für Stunde, Minute und Zeit und fügt diese zu einer gültigen Uhrzeit zusammen.
- **Zeitgeber** (**Timer**): Gibt die Anzahl der Sekunden seit 0:00 Uhr des aktuellen Tages zurück.

Intervalle für das Arbeiten mit Datumsfunktionen

Einige Datumsfunktionen, wie **DatDiff**, **DatTeil**, **DatAdd** oder **DatWert**, nutzen einen Parameter namens **Intervall**. Diese lauten (erst die deutsche, dann die englische Intervallangabe):

- **jjjj** (**yyyy**): Jahr
- **q** (**q**): Quartal
- **m** (**m**): Monat
- **j** (**y**): Tag des Jahres
- **t** (**d**): Tag
- **w** (**w**): Wochentag
- **ww** (**ww**): Woche
- **h** (**h**): Stunde
- **n** (**n**): Minute
- **s** (**s**): Sekunde

Aktuelles Datum ermitteln

Die erste Funktion, die wir vorstellen, ist die Funktion **Datum** (**Date**). Sie ermittelt das aktuelle Datum. Es gibt zwei Versionen dieser Funktion. Die Erste liefert einen Wert des Typs **Variant** mit dem Untertyp **Datum** zurück. Die Zweite heißt **Datum$** und liefert das Datum als **String**-Wert zurück. Für die Angabe in einem Textfeld spielt es keine Rolle, welche Version wir verwenden. Im Textfeld landet ohnehin immer nur eine Zeichenkette, also können wir direkt **die Datum$**-Funktion nutzen. Nur wenn wir noch Berechnungen auf Basis des Datums durchführen wollen, bietet sich **Datum** eher an.

Wir können die **Datum**-Funktion zum Beispiel nutzen, wenn wir in einer Tabelle das aktuelle Datum als Standardwert für ein Datumsfeld angeben wollen. Dazu

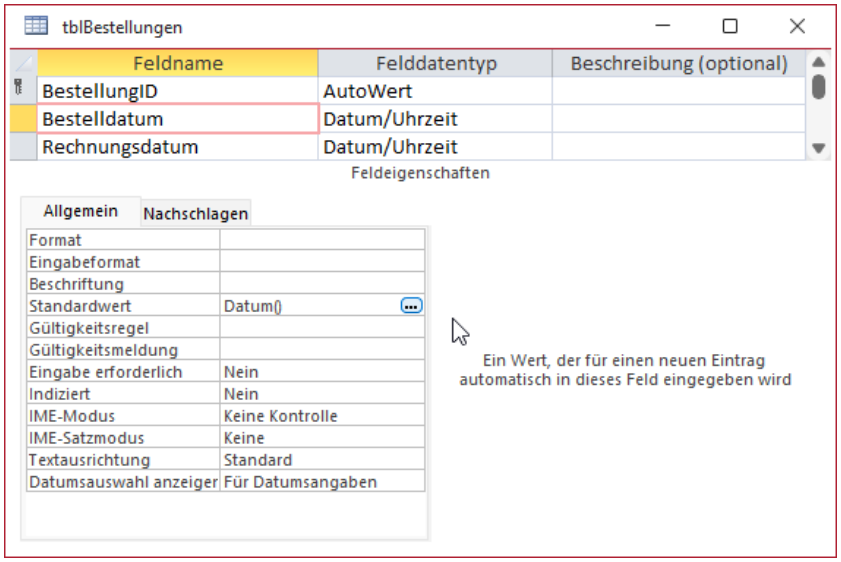

Bild 1: Datum()-Funktion als Standardwert eines Datumfeldes

hinterlegen wir die Funktion **Datum()** für die Eigenschaft **Standardwert** dieses Feldes (siehe Bild 1).

Sobald ein neuer Datensatz angelegt wird, erscheint im Feld **Bestelldatum** das aktuelle Datum als Standardwert (siehe Bild 2).

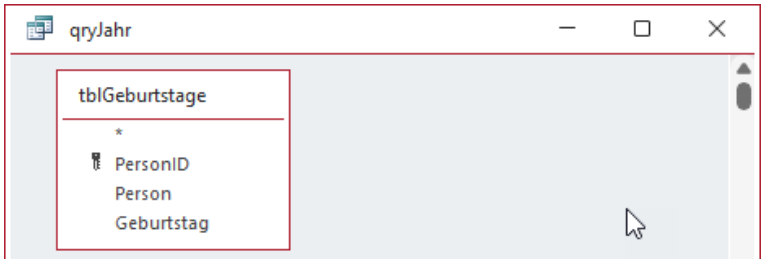

Bild 2: Ein Datumsfeld mit **Datum()**-Standardwert in Aktion

Wenn wir an dieser Stelle die Funktion **Datum$** verwendet hätten, müsste Access das Ergebnis noch in ein echtes Datum umrechnen, damit es im **Datum/Uhrzeit**-Feld gespeichert werden kann. Wobei die Bezeichnung »echtes Datum« täuscht, denn auch dahinter verbirgt sich letztlich ein Zahlenfeld, das nur auf bestimmte Art formatiert wird.

Aktuelle Uhrzeit ermitteln

Auf die gleiche Art können wir auch die aktuelle Uhrzeit ermitteln. Dazu nutzen wir eine der beiden Funktionen **Zeit** (**Time**) oder **Zeit$** (**Time$**) Letztere dient wieder dazu, eine reine **String**-Variante der Uhrzeit zu generieren. Wenn man ein Feld zum Speichern der Uhrzeit in einer Tabelle verwendet, kann man diese Funktion ebenfalls als Standardwert für dieses Feld nutzen.

Aktuelles Datum mit Uhrzeit ermitteln

Mit der Funktion **Jetzt** (**Now**) ermitteln wir das aktuelle Datum mit Uhrzeit. Auch diese können wir beispielsweise in den Standardwert für Datumsfelder integrieren, wenn zusätzlich das Speichern der Uhrzeit gewünscht ist.

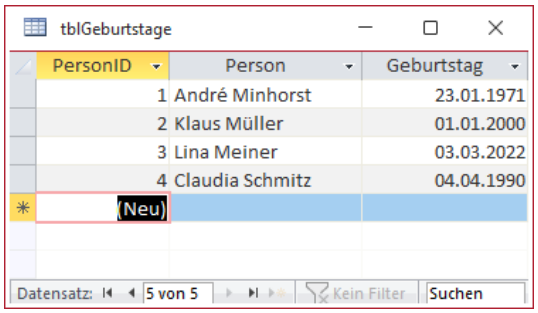

Bild 3: Beispieltabelle mit Geburtsdaten

Funktionen rund um Datumsbestandteile

Für den ersten Satz von Beispielen erstellen wir eine Tabelle namens **tblGeburtstage**, die wie in Bild 3 aussieht.

Jahr eines Datums ermitteln

Um überhaupt Bestandteile eines Datums zu ermitteln, benötigen wir eine Abfrage mit der Tabelle, die das zu untersuchende Feld enthält. Eine solche Abfrage erstellen wir wie in Bild 4, indem wir einer neuen Abfrage die Tabelle **tblGeburtstage** hinzufügen und dann das Feld **Geburtstag** in das Entwurfsraster ziehen.

Um das Geburtsjahr zu ermitteln, ist das nicht erforderlich, aber wir wollen ja sehen, für welches Datum wir das Jahr ermitteln.

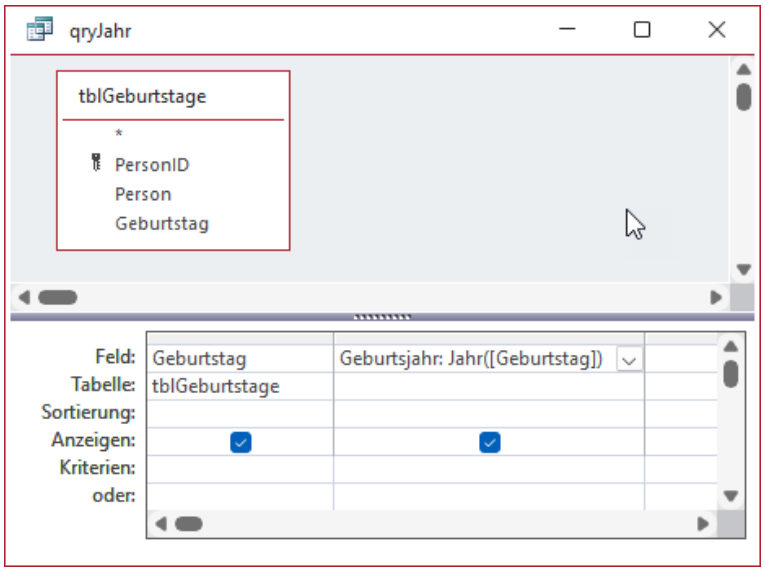

Bild 4: Beispielabfrage mit Geburtsdaten in der Entwurfsansicht

Nun fügen wir ein weiteres Feld mit dem berechneten Ausdruck hinzu, der wie folgt lautet:

```
Geburtsjahr: Jahr([Geburtstag])
```

Wechseln wir damit in die Datenblattansicht, erhalten wir die Anzeige der Geburtsjahre wie in Bild 5.

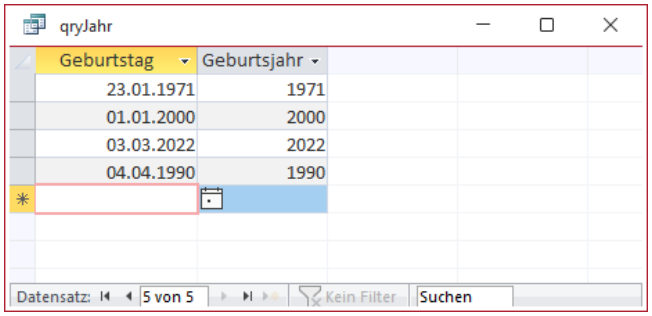

Bild 5: Die Jahre der Geburtsdaten

Monat, Tag und Wochentag, Wochentagsname eines Datums ermitteln

Die Ermittlung der weiteren Elemente eines Datums können wir in einem Abschnitt abhandeln, da wir lediglich weitere Felder zu der bestehenden Abfrage hinzufügen. Dazu müssen wir nur die verschiedenen Funktionen kennen:

- **Monat**: Liefert den Monat eines Datums als Zahl, beim **31.12.2023** also beispielsweise den Wert **12**.
- **Tag**: Liefert den Tag eines Datums als Zahl, beim **31.12.2023** also beispielsweise den Wert **31**.
- **Wochentag**: Liefert den Wochentag als Zahl. Dabei kann man als zweiten Parameter angeben, welcher Tag der Woche der erste Wochentag ist, also für welchen Tag der Wert **1** zurückgeliefert werden soll. Das ist standardmäßig der Sonntag. Das ist wichtig, wenn man einmal per VBA und speziell mit Konstanten wie **vbMonday**, **vbTuesday** und so weiter auf Datumsangaben zugreifen möchte. Normalerweise ist man aber mit dem Standardwert auf der sicheren Seite. Für den 31.12.2023, der ein Sonntag ist, erhalten wir somit den Wert 1 zurück.

Die Abfrage mit dem Entwurf aus Bild 6 fragt für die im Feld **Geburtstag** enthaltenen Daten die einzelnen Bestandteile ab.

Das Ergebnis sehen wir in Bild 7.

Wochentagsname und Monatsname ermitteln

Mit dem Zahlenwert für einen Wochentag kann man normalerweise wenig anfangen, zumal man in Deutschland nach DIN 1355-1 den Montag als ersten

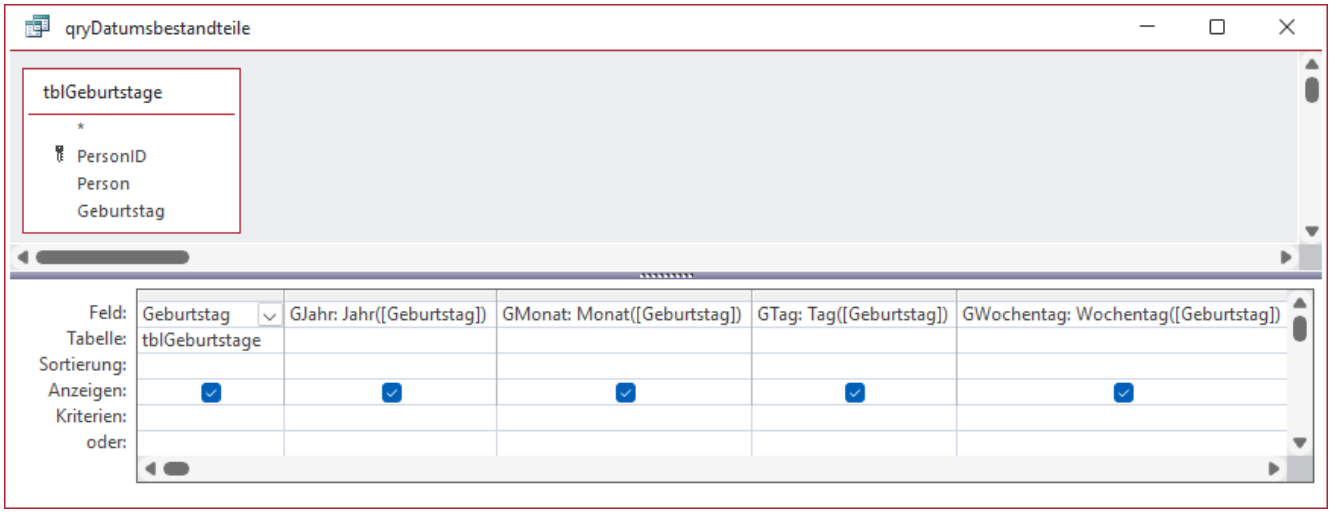

Bild 6: Verschiedene Informationen eines Datums im Entwurf