So ziemlich jede neue Datenbank enthält eine Menge Elemente, die nicht in Access enthalten sind, aber aus anderen, bereits fertigen Datenbanken herübergerettet werden. Und jedes Mal ärgert man sich über diese ganzen Vorarbeiten und dass dann doch wieder irgendwas fehlt oder vielleicht endlich schöner gelöst werden könnte.

Ich möchte hier daher einen Prototypen vorführen, der vieles von dem enthält, was ich so im Laufe der Jahre entwickelt habe. Es ist ausdrücklich kein Gestaltungs-Konzept, sondern eines für die oft unsichtbare Technik im Hintergrund.

AEKcess – die Kraft der zwei Herzen

Die vorgestellte *AEKcess*-Datenbank besteht im Grunde aus zwei Teilen: den technisch notwendigen Objekten, die im Folgenden erläutert werden, sowie den zum Vorführen hilfreichen Beispieldaten.

Damit die technisch notwendigen Objekte für normale Benutzer nicht sichtbar sind, erhalten diese das Präfix *USys_*. Der Unterstrich ist dabei nicht wirklich notwendig, sondern dient nur der besseren Lesbarkeit. Durch das Präfix werden sie bei Bedarf mit der Option *Systemobjekte anzeigen* ein- oder ausgeblendet. Das funktioniert nicht nur für Tabellen, wie es ja schon bei *USysRibbons* genutzt wird, sondern auch bei Abfragen bis hin zu Modulen. Die einzige Ausnahme in dieser Benennung sind zwangsläufig die beiden namentlich vorgegebenen Makros *AutoExec* und *AutoKeys*.

Die Beispieldatenbank mit geeigneten Testdaten stellt die stark vereinfachte Verwaltung einer Yachtcharter-Firma mit mehreren Niederlassungen am Mittelmeer dar. Die Yachten werden dabei von privaten Eignern zur Verfügung gestellt und von den Kunden für Törns gebucht.

Eigentlich lassen sich die Tabellen für diese beiden Datenbanken auch in getrennten BackEnds speichern. Allerdings bilden die (technischen) Benutzer der Datenbank fast immer sowieso eine Untermenge der (inhaltlich beteiligten) Personen. Das gilt beispielsweise für Mitarbeiter, die in einer solchen Datenbank ihre Verkäufe eintragen. Hier im Beispiel wäre das eher ausnahmsweise der Fall, wenn jemand, der die Datenbank bedient, auch eine Yacht chartern möchte. Wenn es zwischen den vielen Personen und den wenigen Benutzern aber eine 1:1-Beziehung mit Referentieller Integrität gibt, müssen diese beiden Tabellen in einer gemeinsamen Datenbank enthalten sein.

Um wirklich eine komplette Tabellentrennung durchzuführen, müssten Personen und Benutzer in eigenen Tabellen ohne Zusammenhang gespeichert werden, wodurch viele Felder doppelt wären. Da aber auch die übrigen Objekte von Abfragen bis hin zu Modulen sowieso technisch nicht ernsthaft trennbar sind, habe ich auf die Tabellentrennung auch verzichtet.

Version

Das vorliegende Beispiel ist in Access 2010/2013 erstellt. Die Ribbons sind sogar noch in der alten Version

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
```

notiert, wie sie für Access 2007 nötig sind. Da ich keine neueren Ribbon-Fähigkeiten nutze, habe ich auf die eigentlich korrektere Version

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
```

verzichtet. Die Ribbons funktionieren also ab Version 2007.

Um die Datenbank auch komplett für Access 2007 zu nutzen, müssen lediglich die berechneten Tabellenfelder (hier als *…NameAnzeigen* zu erkennen) entfernt und durch zusätzliche Abfragen mit gleichnamigen berechneten Feldern nachgebildet werden. Statt neuer Abfragen könnt Ihr auch einfach die dann fehlenden Felder in Formularen und Berichten durch vorhandene ersetzen.

BackEnd

Immer, wenn ein FrontEnd an den Kunden ausgeliefert wird, stehen in den verknüpften Tabellen noch die Pfade des Entwicklungsrechners drin. Jede neue Version löst also Irritationen bei den Endbenutzern aus, die plötzlich mit unverständlichen Fehlermeldungen konfrontiert werden.

Daher greift eine VBA-Prozedur beim Start der Datenbank auf eine beliebig ausgewählte Tabelle zu und prüft, ob die Verknüpfung in Ordnung ist. Falls nicht, wird hier nach einem Hinweis direkt der Windows-Dateidialog geöffnet, um das BackEnd auszuwählen und anschließend die Tabellen damit automatisch zu verknüpfen.

Für Datenbanken mit mehreren BackEnds lässt sich stattdessen der Tabellenverknüpfungs-Manager oder auch ein eigenes Formular aufrufen. Dazu ändert sich in der Funktion *StarteDB()* die Fehlerbehandlung von der bisherigen Version

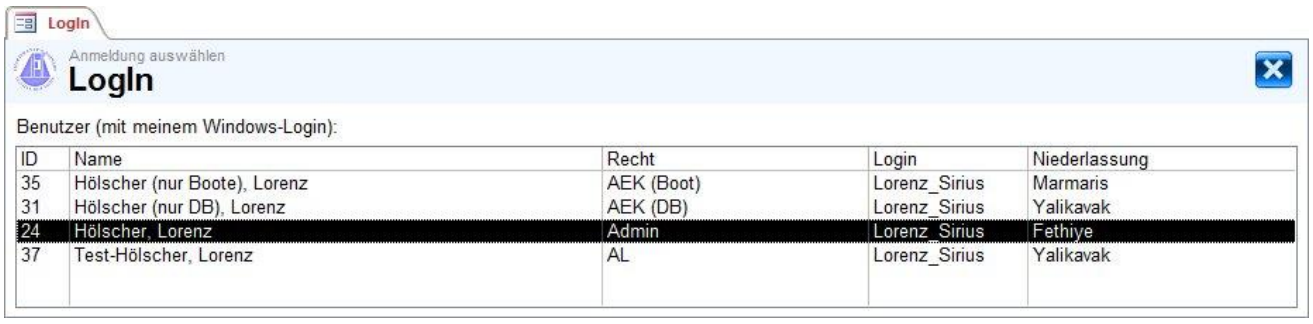
```
BackEndFehlt:
    MsgBox "Die BackEnd-Datenbank ist nicht korrekt eingebunden," & vbCrLf & _
        "bitte wählen Sie die BackEnd-Datei neu aus.", vbCritical, p_cstrTitel
    TabellenNeuVerknuepfen
    Exit Function
End Function
```

in diese Fassung:

```
BackEndFehlt:
    DoCmd.Hourglass False
    MsgBox "BackEnd-Datenbank ist nicht korrekt eingebunden." & vbCrLf & _
        "Der Tabellenverknüpfungsmanager wird aufgerufen.", vbCritical, p_cstrTitel
    DoCmd.RunCommand acCmdLinkedTableManager
    Exit Function
End Function
```

Anmeldung

AEKcess benutzt eine zweistufige Anmeldung. Der Benutzer wird zuerst eindeutig erkannt anhand seines Windows-LogIns. Damit ist er überall als Echtperson identifizierbar. Dann bietet das LogIn-Formular alle in der Datenbank (in *USys_tblPersonenBenutzer*) enthaltenen Benutzer zur Auswahl an, welche das gleiche LogIn gespeichert haben:



ID	Name	Recht	Login	Niederlassung
35	Hölscher (nur Boote), Lorenz	AEK (Boot)	Lorenz_Sirius	Marmaris
31	Hölscher (nur DB), Lorenz	AEK (DB)	Lorenz_Sirius	Yalikavak
24	Hölscher, Lorenz	Admin	Lorenz_Sirius	Fethiye
37	Test-Hölscher, Lorenz	AL	Lorenz_Sirius	Yalikavak

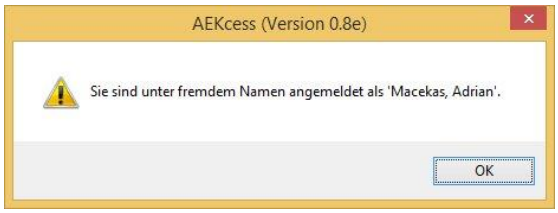
Typischerweise gibt es für normale Benutzer zu deren Windows-LogIn auch nur genau einen Datenbank-Benutzer. Häufig existieren aber Anforderungen wie in Sekretariaten, bei denen jemand für mehrere Abteilungen arbeitet und jeweils mit deren unterschiedlichen Rechten ausgestattet werden soll. Dann kann er wie in der obigen Abbildung unter mehreren DB-Benutzern auswählen und wechselt damit seine ID. In diesem Fall weiß die Datenbank, dass der Benutzer das LogIn *Lorenz_Sirius* und die ID 24 hat.

Diese erste Anmeldung ist Pflicht, um sich gegenüber der Datenbank überhaupt qualifiziert ausweisen zu können. Mit der Erstanmeldung wäre die Datenbank bereits in vollem Umfang zu benutzen. Die beiden nächsten Varianten können erst nach dieser Erstanmeldung stattfinden, weil sie die weiteren Auswahl-

möglichkeiten aus dem gewählten Recht (hier *Admin* für die ID 24) oder der Gruppe (hier die Niederlassung *Fehtiye*) ableiten.

Eine typische Anforderung im Firmen-Umfeld ist nicht nur eine (vorhersagbare) Urlaubsvertretung, sondern vor allem eine (überraschende) Krankheitsvertretung. Daher gibt es beim erneuten Aufruf des LogIn-Formulars die Auswahl, alle Benutzer meiner Gruppe (in diesem Fall meiner Niederlassung) anzuzeigen, obwohl diese nicht mein LogIn haben.

Dort kann sich ein Benutzer für einen dieser anderen Namen anmelden. Eine Freigabe durch diesen anderen Benutzer ist ausdrücklich nicht erforderlich, weil sonst ja Krankheitsvertretungen scheitern würden. Damit das nicht missbräuchlich zur Manipulation von dessen Daten genutzt wird, erzeugt *AEKcess* bei einer solchen Fremdanmeldung ausdrücklich eine Warnung an diesen und eine Nachricht an den anderen Benutzer:

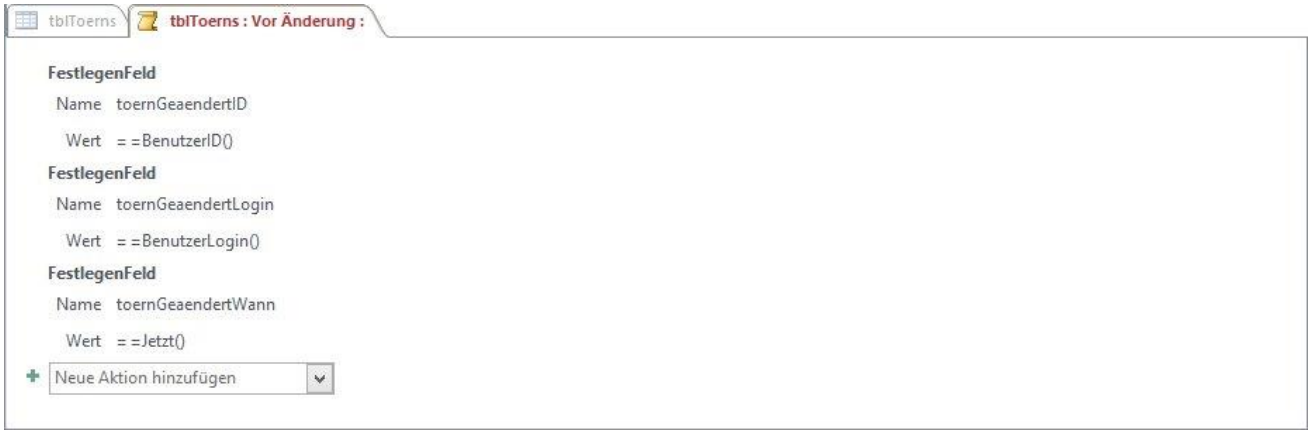


Nur für Admins (genauer gesagt, hier diejenigen mit dem Recht 1) gibt es zusätzlich die Möglichkeit, sich im Namen jedes beliebigen Benutzers anzumelden, egal aus welcher Gruppe oder mit welchem LogIn. Dadurch ist es sehr bequem und effektiv, bei der Fehlersuche mit genau den Einstellungen des jeweiligen Benutzers zu arbeiten.

Innerhalb des Nachrichtensystem erhält ein Benutzer dann für jede solche Fremdanmeldung einen Hinweis wie in der folgenden Abbildung (hier unter der ID 434), dass und welcher andere Benutzer sich in seinem Namen angemeldet hat.

Außerdem weiß die Datenbank für jeden Benutzer immer sowohl dessen Windows-LogIn (hier weiterhin *Lorenz_Sirius*) als auch dessen Benutzer-ID (hier jetzt 26 statt 24). Beide Informationen stehen bei wichtigen Datensätzen als letzte Änderung in entsprechenden Feldern, so dass also leicht herauszufinden ist, welche Daten tatsächlich von wem manipuliert wurden.

In der Beispiel-Datenbank ist das nur in der Tabelle *tblToerns* verwirklicht. Dazu schreibt ein *VorÄnderung*-Tabellenmakro diese Informationen in die Felder:



Als Besitzer solch eines fremdgeänderten Datensatzes gilt derjenige, dessen ID darin steht, so dass also im Urlaubs- oder Krankheitsfall der ursprüngliche Ersteller weiterhin den Zugriff darauf behält.

Nachrichten

Wie schon im Zusammenhang mit der Fremdanmeldung erwähnt, gibt es ein einfaches Nachrichtensystem, welches im Grunde nur aus einer Tabelle *USys_tblNachrichten* besteht. Jede Nachricht ist entweder mit einer konkreten BenutzerID verknüpft (persönliche Nachrichten) oder dieses Referenzfeld ist leer (allgemeine Nachrichten).

Beim Start der Datenbank wird noch vor der eigentlichen Benutzer-Auswahl das Nachrichten-Formular angezeigt. Dort sind alle Nachrichten sichtbar, die entweder allgemein sind oder zum jeweiligen Windows-LogIn passen. Nachrichten können sowohl ein Start- als auch ein Ablaufdatum besitzen, welches ihre Sichtbarkeit steuert.

Eine Bearbeitung von Nachrichten durch Benutzer ist so derzeit nicht vorgesehen, alle Datensätze werden hier einfach vom Administrator direkt in der Tabelle eingegeben. Dazu kommen dann noch die automatisch erstellten Fremdzugriff-Benachrichtigungen.

Auf Kundenwunsch hatte ich dieses auch schon mal erweitert (siehe folgende Abbildung), so dass jeder Benutzer eigene Nachrichten erzeugen und für diese bei Bedarf auch eine Firmen-ID angeben konnte. Das diente dort als Gesprächsnotiz, wenn Telefonate für Kollegen angenommen wurden und diese dann ohne langes Suchen von der Nachricht aus einen schnellen Zugriff auf die Firmendaten hatten.

Der Verwaltungsaufwand erhöhte sich dadurch, weil natürlich jeder Benutzer nur diejenigen Nachrichten löschen durfte, die von ihm erstellt oder an ihn gerichtet worden waren. Lesen sollte er allerdings auch allgemeine Nachrichten, deren Empfänger also leer war.

Damit ihn neue Nachrichten schnell erreichten, musste das entsprechende Formular (welches ohnehin einen zentralen Bedienungsbildschirm mit "Menüs" darstellte) nicht nur dauerhaft sichtbar sein, sondern das Nachrichten-Unterformular per Timer auch oft aktualisiert werden. Das ist nicht gerade perfekt für die Performance…

Dynamische Befehle

Abgesehen von einigen allgemeinen Befehlen (*Info*, *Datenbank komprimieren*, etc.) sowie den Formular-spezifischen Aktionen ist die wesentliche Bedienung der Datenbank in einem *dynamicMenu* organisiert. Das hat gegenüber dem Einsatz von Ribbon-Buttons mehrere Vorteile:

- Ein Menü benötigt optisch viel weniger Icons, ohne dass es deswegen so lieblos oder unfertig aussieht wie ein Ribbon voller Text-Buttons. Ich beschränke mich hier auf Icons für die erste Menü-Ebene, welche an die untergeordneten Elemente weitergereicht werden.
- Menü-Einträge vertragen sehr viel längere Texte und brauchen dabei weniger Platz. Sie lassen sich hierarchisch (in Untermenüs oder mit Trennungen) organisieren und sind dadurch übersichtlich, solange sie nicht durch schiere Menge überladen werden.
- Vor allem aber muss das *dynamicMenu*-Element erst dann seine Inhalte aktualisieren, wenn es auch wirklich ausgeklappt wird. Ein Ribbon mit seinen immer sichtbaren Buttons muss diese hingegen bei jeder Änderung in irgendeinem Formular sofort aktualisieren.

Technisch wird ein solches *dynamicMenu* mit einem XML-Baum beschrieben, der erst beim Klick auf dessen Start-Button via VBA erstellt wird. Die Prozedur liest alle Einträge aus der Tabelle *USys_tblMenues*, die in *USys_tblMenues2Rechte* mit demjenigen Recht verknüpft sind, welches der Benutzer laut *USys_tblPersonenBenutzer* hat. Das bedeutet, dass auch Administratoren nicht selbstverständlich alle Menü-Einträge sehen, sondern diese explizit zugewiesen bekommen müssen.

Für eine ähnliche Datenbank, bei der es auf Dauer nur drei Rechte (Admin, interner und externer Mitarbeiter) geben sollte, hatte ich statt dieser umfangreichen Tabellen-/Rechte-Lösung auch eine reine VBA-Version erstellt. Dabei wurde der XML-String je nach Recht einfach mit ein paar `If`-Blöcken direkt in VBA zusammengesetzt.

Das war technisch bedeutend einfacher (vor allem auch mit dynamischen Beschriftungen in Menüs oder dem Aufruf von Prozeduren samt Parametern), ist aber auf Dauer natürlich zu unflexibel. Während bei Tabellen nämlich ein erfahrener Benutzer neue Menü-Einträge hinzufügen oder wegen geänderten Rechten entziehen kann, bedeutet die reine VBA-Lösung immer einen erheblich Eingriff in den Code und entsprechende Programmierkenntnisse. Nachdem der Anfangsaufwand hier nun bereits erledigt ist, würde ich also von jener reinen VBA-Variante abraten.

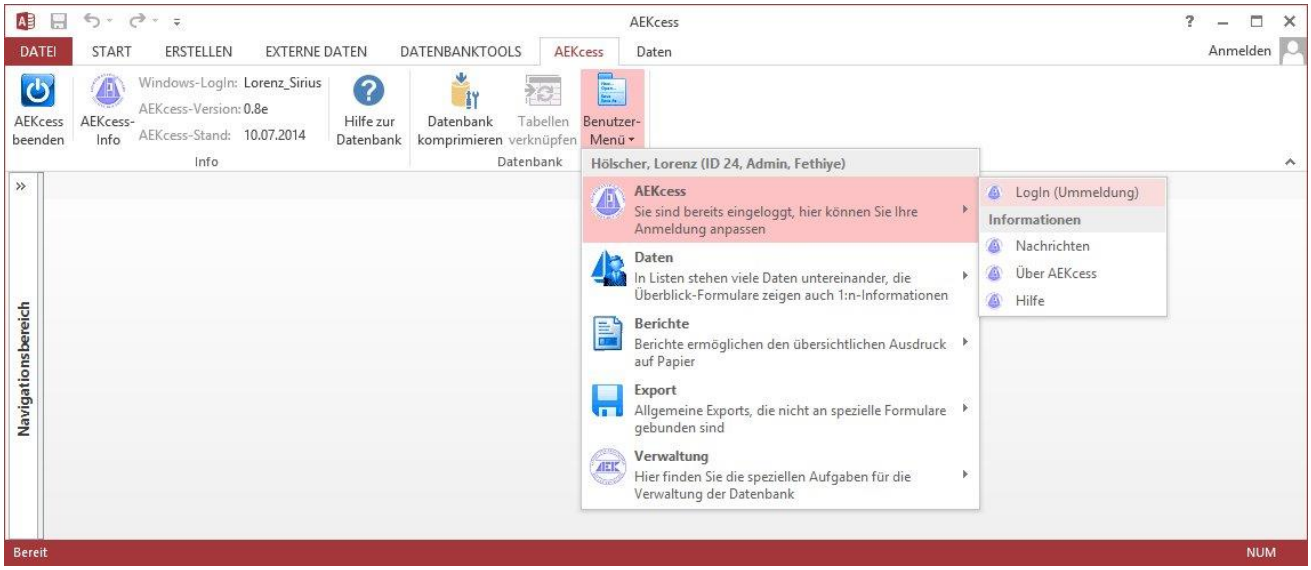
Die Tabelle *USys_tblMenues* enthält die maximale Liste aller Menü-Elemente einschließlich einer Sortierung. Jedes dieser Menü-Elemente kann von folgendem Typ sein (Beispiele siehe in den folgenden Abbildungen):

- **Überschrift**: Eintrag in der ersten Menü-Ebene (z.B. *Formulare*)
- **Eintrag**: Eintrag in der zweiten Menü-Ebene (z.B. *Nachrichten*)
- **Separator**: grauer Trennbalken mit freiem Text (z.B. *Informationen*)
- **Untermenü Start**: Beginn der dritten Menü-Ebene oder tiefer (danach folgen Eintrag-Elemente)
- **Untermenü Ende**: Abschluss der letzten Menü-Ebene
- **Liste Einträge**: automatisch erzeugte Einträge aus einer Abfrage (z.B. *Anzahlung erfolgt*, *Buchung*, etc.)

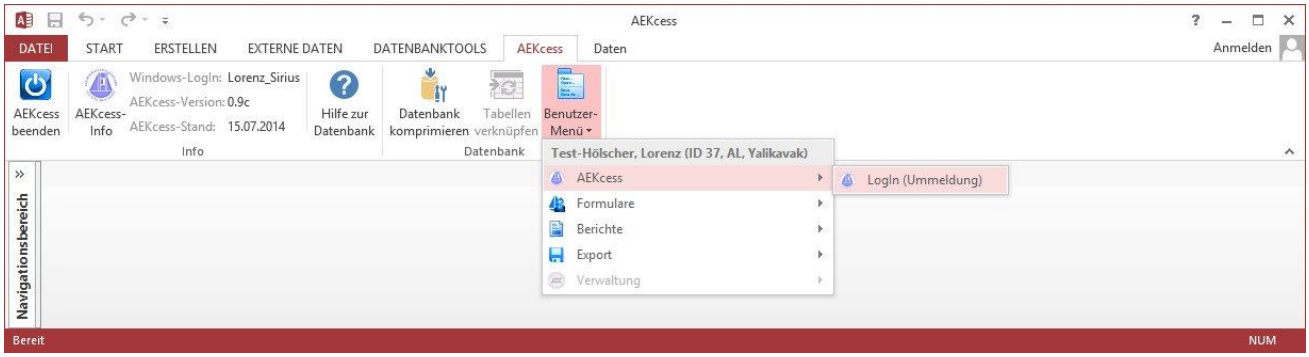
Die Position, an der ein Menü-Eintrag erscheint, ist abhängig von der Sortierung und eventuell den vorher sichtbaren Elementen. Insbesondere die unteren Menü-Ebenen erfordern, dass der Benutzer auch die Rechte an den übergeordneten Untermenü-Start-/-Ende-Elementen besitzt, weil die Einträge sonst fälschlich einer höheren Ebene zugeordnet werden.

Die folgende Abbildung zeigt das Menü noch ohne Anmeldung, also für einen Gast ohne Rechte (wie es jeweils auch im grauen Separator zu lesen ist):

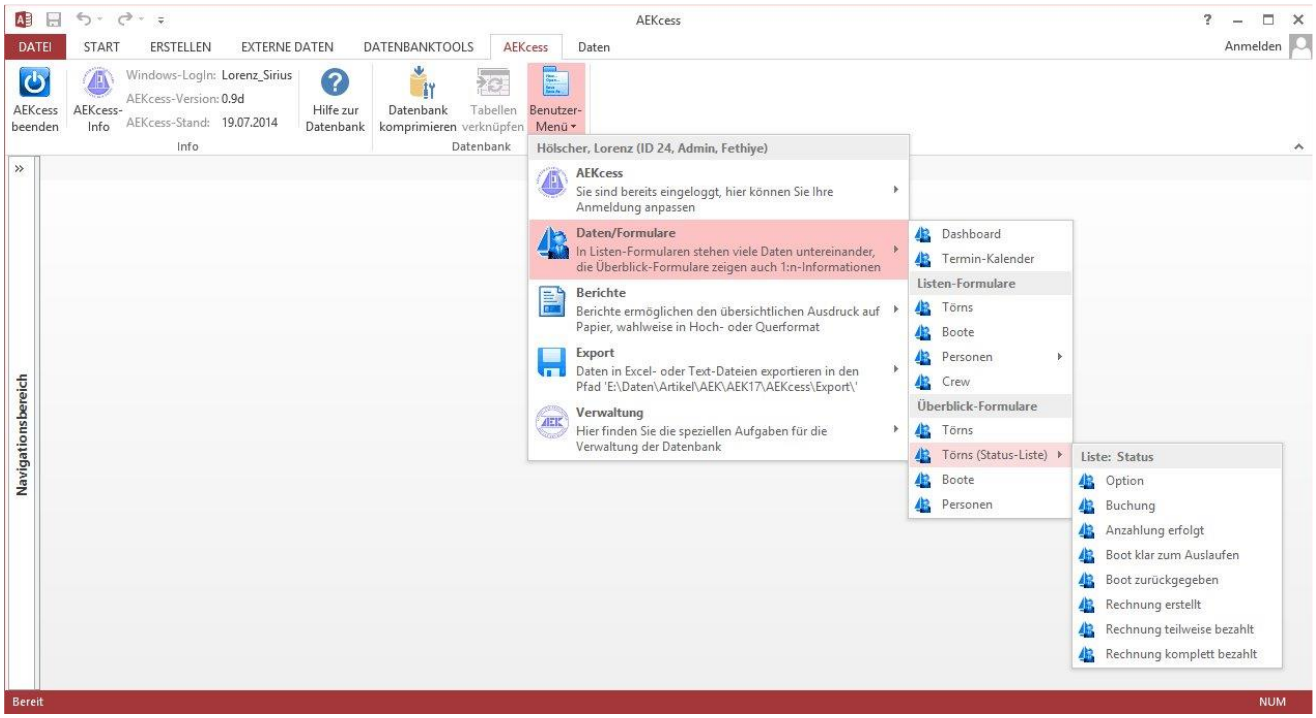
Nach der Anmeldung (hier mit der ID *24* als *Lorenz Hölscher* aus der Niederlassung *Fethiye* mit *Admin*-Rechten) zeigt das Menü deutlich mehr Einträge, weil diesem Recht praktisch alle Elemente zugeordnet wurden:



Andere Rechtegruppen (hier AL = Abteilungsleiter) sehen nach Belieben eine Auswahl aus den Menüs, wie es inhaltlich für sinnvoll erachtet wird. In der folgenden Abbildung ist zudem die Benutzer-Option *Kleines Menü* aktiv, daher sind die Menüeinträge nur noch einzeilig mit Mini-Icon dargestellt:



Am Untermenü-Eintrag *Törns (Status-Liste)* hängt in der folgenden Abbildung beispielhaft das Ergebnis einer dynamisch erzeugten Eintragsliste mit den Werten *Option* bis *Rechnung komplett bezahlt*. Diese Werte stehen nicht als einzelne Datensätze in *USys_tblMenues*, sondern sind in diesem Fall das Ergebnis der angegebenen Abfrage *qryStatustypenSortiertFuerBericht*.



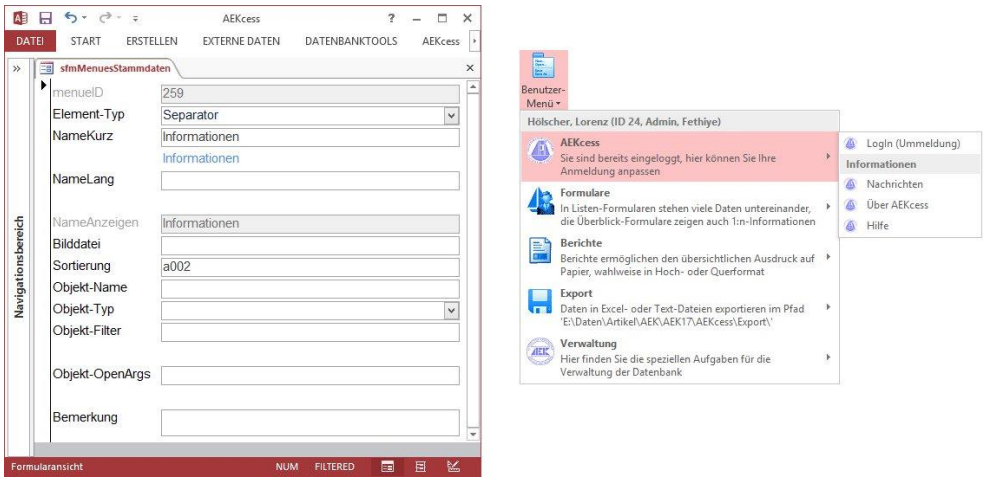
Da das Menü recht flexibel ist, gibt es viele Bedingungen und Parameter zu beachten:

- Der erste Eintrag (mit dem Recht des jeweiligen Benutzers) muss einer vom Typ *Überschrift* sein.
- Nach einem *Untermenü-Start* folgen beliebig viele *Eintrag*-Elemente, sie müssen mit einem *Untermenü-Ende*-Element abgeschlossen werden.
- Ein *Separator* ohne folgende Inhalte wird nicht angezeigt (originales Ribbon-Verhalten).
- Ein *Überschrift*-Element ohne zugehörige Einträge ist deaktiviert (originales Ribbon-Verhalten).

Die Sortierung ist für alle Rechtegruppen gleich, allerdings kann der gleiche Menü-Eintrag mehrfach genannt und mit unterschiedlichen Sortierungsangaben versehen werden. Die Art der Sortierung ist egal, ich habe der Einfachheit halber für *Überschriften* Buchstaben benutzt, für *Einträge* dreistellige Zahlen und für *Untermenüs* wiederum Buchstaben.

Separator

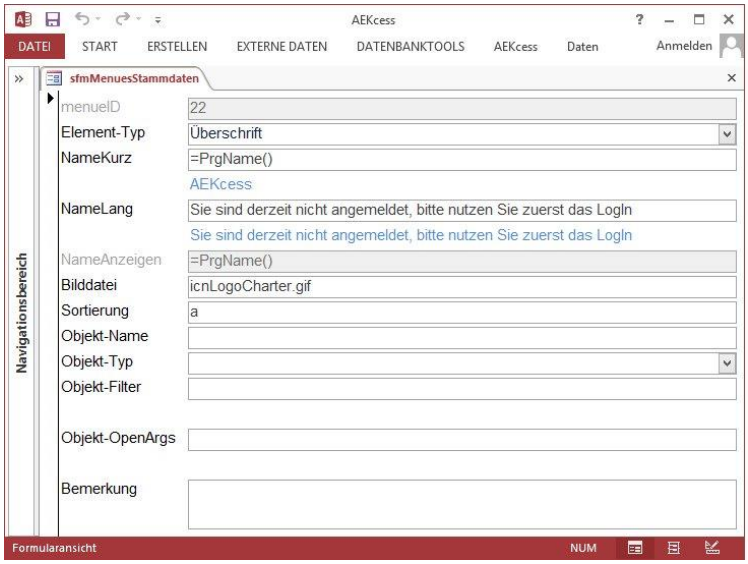
Ein Separator ist eine inaktive und grau hinterlegte Zeile mit beliebigem Text. Sie dient nur der übersichtlichen Strukturierung von längeren (Unter-)Menüs wie hier bei *Informationen*:

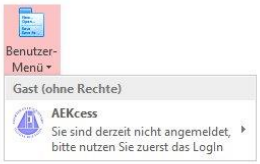


Der erste Separator ganz oben im Hauptmenü mit den Informationen zum angemeldeten Benutzer wird nicht von der Tabelle *USys_tblMenues* erzeugt, sondern automatisch immer via VBA hinzugefügt.

Überschrift

Eine Überschrift bildet einen Menüeintrag in der ersten Ebene. Der (wie in der folgenden Abbildung gezeigte und eventuell interpretierte) Inhalt von *menueNameKurz* wird als Beschriftung angezeigt. Darunter steht der Inhalt von *menueNameLang* als sogenannte *Description*. Wenn die Benutzer-Option *Kleine Menüs* ausgeschaltet ist, werden die Elemente nur einzeilig ohne Untertext angezeigt.

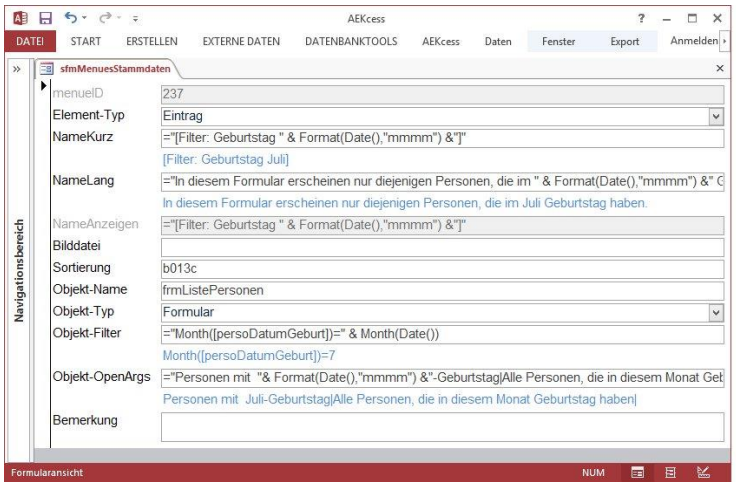


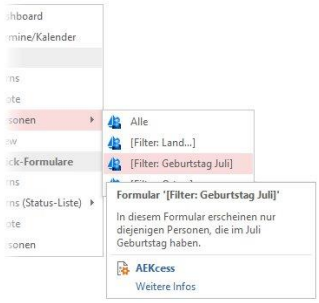


Nur *Überschrift*-Elemente benötigen den Namen einer Bilddatei (aus dem Verzeichnis *Icons*), diese wird dann auch für alle untergeordneten Elemente angezeigt.

Eintrag

Ein Menü-*Eintrag* funktioniert sehr ähnlich wie ein *Überschrift*-Element, allerdings sind Menüeinträge immer klein, besitzen also keinen Untertext. Wie die folgende Abbildung zeigt, dient *menueNameKurz* als Beschriftung und (aber nur, wenn *menueNameLang* nicht leer ist) *menueObjektTyp* plus *menueNameKurz* als Screentip sowie *menueNameLang* als SuperTip:



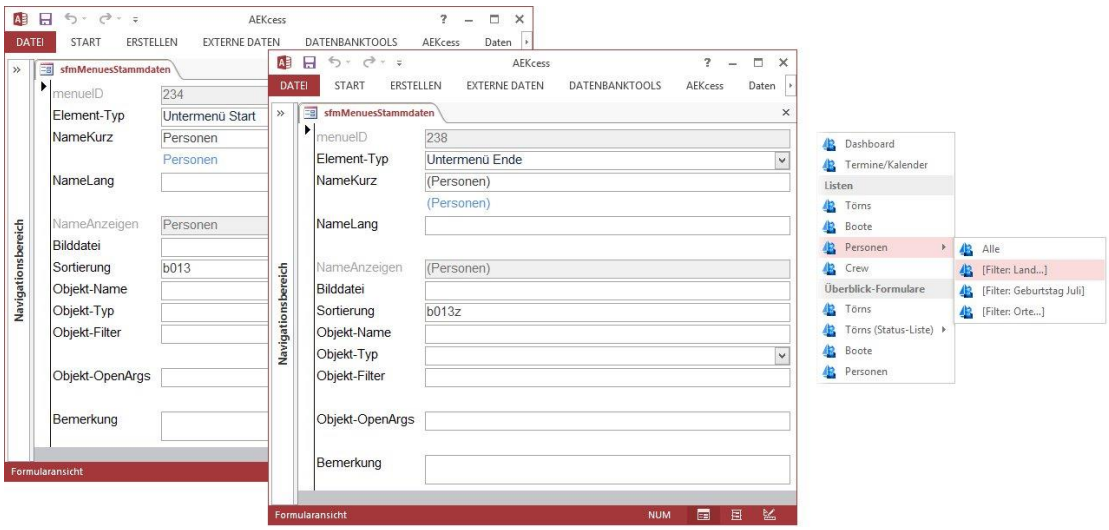


Objekt-Typ und Objekt-Name alleine reichen als Aufruf aus, bei Bedarf dürfen außerdem Objekt-Filter oder Objekt-OpenArgs übergeben werden. Die Objekt-OpenArgs werden als komplette Zeichenkette an

das Formular oder den Bericht weitergereicht, diese müssen sich dann um die (eventuelle Zerlegung mittels *Split()*-Funktion und) Interpretation kümmern.

Untermenü

Um ein *Untermenü* zu erzeugen, braucht es immer je einen *Untermenü-Start*- und *Untermenü-Ende*-Datensatz, wie in der folgenden Abbildung für die *Personen*. Dazwischen liegen beliebig viele Menü-*Eintrag*-Datensätze. Eine Angabe von *menueNameKurz* für *Untermenü Ende* ist technisch überflüssig, aber deutlich übersichtlicher:

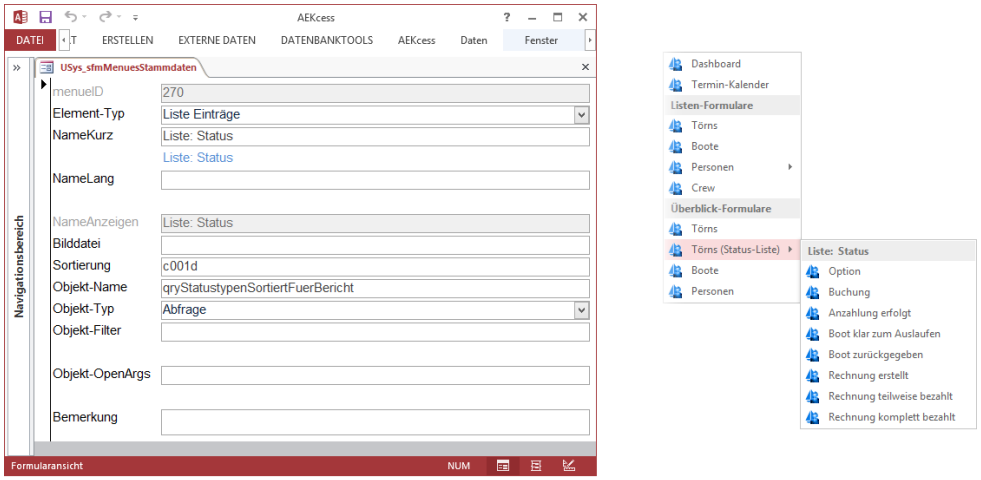


Liste

Während alle bisherigen Menüeinträge jeweils auch als ein Datensatz in *USys_tblMenues* zu finden sind, gilt das für den Sonderfall der Listen nicht. Diese entstehen aus den Inhalten einer geeigneten Abfrage, die immer ein Wert/Name-Paar erzeugen muss. Die Werte (typischerweise ein ID-Feld) werden versteckt in der *Marke*-Eigenschaft des Menüs gespeichert, die Namen erscheinen als Beschriftungen. Eine solche typische Abfrage *qryStatustypenSortiertFuerBericht* ist hier nebenstehend zu sehen, die tatsächlichen Feldnamen sind dabei unerheblich.

toernstatulDf	statuNameAnzeigen
1	Option
2	Buchung
3	Anzahlung erfolgt
4	Boot klar zum Auslaufen
5	Boot zurückgegeben
6	Rechnung erstellt
7	Rechnung teilweise bezahlt
8	Rechnung komplett bezahlt
*	(Neu)

Aus Gründen der Übersichtlichkeit sollten Listen ein eigenes Untermenü bilden, daher ist es sinnvoll die eigentliche Liste mit einem *Untermenü-Start* und einem *Untermenü-Ende* zu umrahmen. Sie können aber auch als Teil eines (Unter-)Menüs genutzt werden. Der Inhalt von *menueNameKurz* (hier *Liste: Status*) erscheint bei Listen als automatisch erstellter Separator.



Da Menüs intern eindeutige Namen haben müssen, wird zu deren Bezeichnung *btnMenue…* die *menueID* des jeweiligen Datensatzes hinzugefügt. Anhand dieser ID kann die Callback-Funktion später die notwendigen Werte zur Anzeige des Menüs ermitteln. Bei den hier gezeigten Listen gibt es aber nur eine einzige echte Datensatz-ID, daher wird dort zusätzlich die ID des Listenwerts angehängt. Das bedingt, dass in der Abfrage nur eindeutig nummerierte Werte benutzt werden können.

Objekt-Typen

Ein Aufruf eines Menüeintrags kann vier verschiedene Objekttypen auslösen:

- Abfrage
- Formular
- Bericht
- VBA-Funktion (keine Prozedur!)

Zusätzlich können, vor allem für Formulare und Berichte, Filter eingetragen werden und, auch für VBA-Funktionen, Parameter. Letztere müssen aus technischen Gründen in einer einzigen Zeichenkette (und bei Bedarf durch das |-Zeichen getrennt) vorliegen.

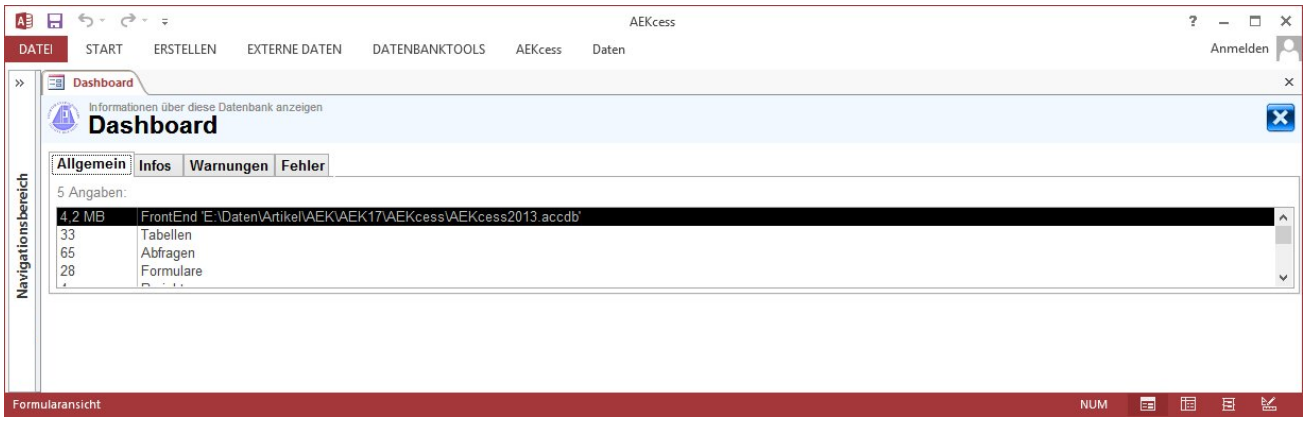
Die Felder *menueNameKurz*, *menueNameLang*, *menueObjektFilter* und *menueObjektOpenArgs* werden mit der *Eval()*-Funktion interpretiert, so dass also darin auch VBA-Funktionen genutzt werden können. In den Stammdaten des *frmMenues*-Formulars ist das Ergebnis der *Eval*-Interpretation parallel sichtbar, so dass also dort die syntaktische Korrektheit des Codes überprüft werden kann. Im Menü hingegen zeigt sich eine fehlerhafte Syntax nur durch zwei Fragezeichen, weil ansonsten der XML-Baum syntaktisch fehlerhaft sein könnte.

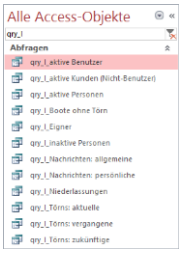
Die aufgerufenen VBA-Funktionen dürfen sowohl integrierte als auch eigene sein, sie müssen allerdings öffentlich (also nicht mit *Private* gekennzeichnet oder in einem Klassenmodul gespeichert) sein. Selbst wenn die Funktion keinen Rückgabewert hat, sind *Sub*-Prozeduren technisch nicht möglich.

Dashboard

Das Dashboard-Formular zeigt praktisch beliebige Informationen zur Datenbank an. Es löst das häufige Problem, dass Benutzer bestimmte Daten gerne auf einen Blick sehen wollen. Diese Funktion wird manchmal auch als *Ampel* bezeichnet, weil dort farblich codiert werden soll, ob ein Wert in Ordnung ist.

In dieser Version besitzt das Dashboard vier Registerkarten, von denen die erste (*Allgemein*) per VBA ermittelte Daten zur Datenbank selber anzeigt, sozusagen die Minimalversion des Info-Formulars:

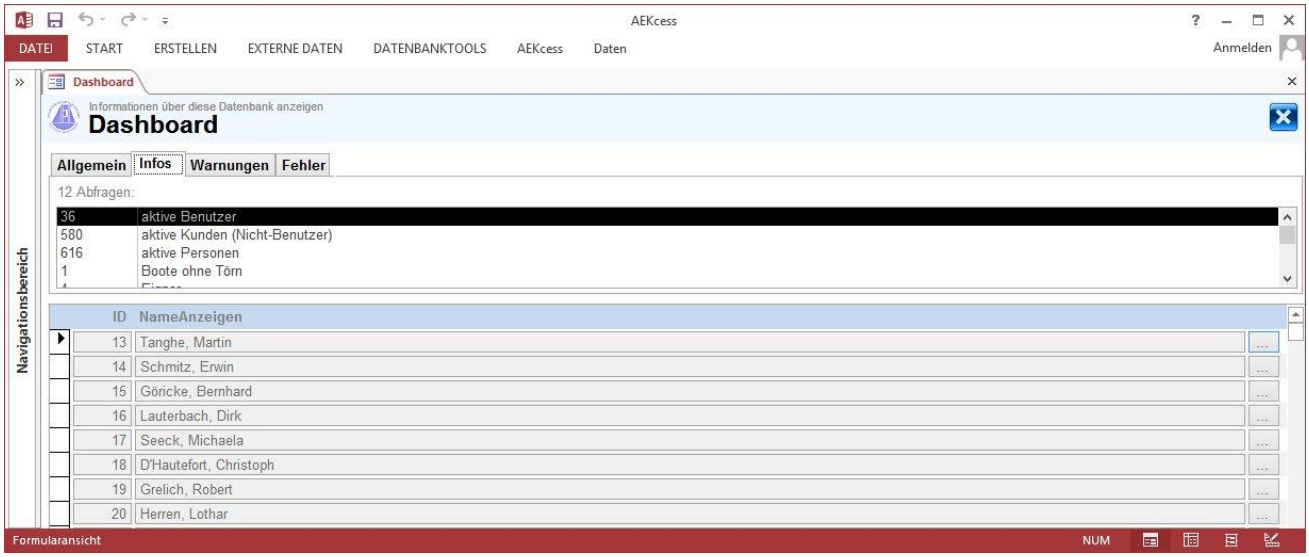




Die weiteren drei Registerkarten zeigen alle Abfragen und die Anzahl der darin enthaltenen Datensätze an, die einem bestimmten Namensschema entsprechen. Sie werden alle nach dem Muster *qry_X_BeliebigerName* erstellt, wobei *X* konkret *I* (Infos), *W* (Warnungen) oder *F* (Fehler) ist.

Zu den in der folgenden Abbildung sichtbaren 12 Zeilen für die Infos gibt es also (siehe rechts) die entsprechenden Abfragen, von denen jeweils nur der hintere Teil des Namens angezeigt wird.

Für die Abfragen werden im unteren Teil des Formulars die konkreten Datensätze sichtbar. Damit diese für beliebige Inhalte in diesem Unterformular angezeigt werden können, müssen sie reservierte Feldnamen benutzen, aus *persoID* wird hier also *ID* und aus *persoNameAnzeigen* nur *NameAnzeigen*.



Zusätzlich gibt es zwei berechnete Felder *NameFeld* und *NameFormular*, welche für den Klick auf die Schaltfläche daneben ausgewertet werden. Sie enthalten die Bezeichnung des dann aufzurufenden Formulars sowie den ursprünglichen, echten ID-Feldnamen, damit im Formular der passende Datensatz herausgefiltert werden kann.

Suchformular

Damit die Suche nach Datensätzen nicht immer wieder neu programmiert werden muss, gibt es ein zentrales Suchformular. Es zeigt immer ein Wert/Name-Paar aus einer Abfrage und gibt eine *WHERE*-Klausel als Filter-String zurück. Der Aufruf erfolgt wahlweise aus dem Menü heraus vor dem Erscheinen eines Formulars oder Berichts, über einen Ribbon-Button oder auch vom Filter-/Sortier-Control.

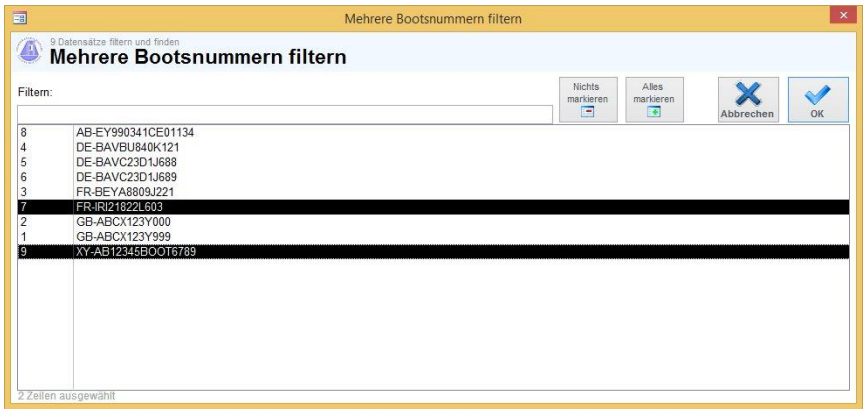
Das Formular wird modal angezeigt und verändert sich je nach übergebenen Parametern oder Daten. Es gibt je zwei wesentliche Alternativen:

1. Die zugrundeliegende Abfrage enthält echte Wert/Name-Paare, dann ist die Liste zweispaltig. Diese Variante ist typisch für Nachschlagefelder. Das Filterfeld oberhalb der Liste durchsucht beide Spalten. Das ist vor allem dann hilfreich, wenn in der ersten Spalte keine IDs stehen, sondern Nummern, die dem Benutzer geläufig sind.
2. Die beiden Felder der Abfrage sind namensgleich, dann wird die erste Spalte ausgeblendet. Das gilt vor allem für die Suche in Textfeldern.

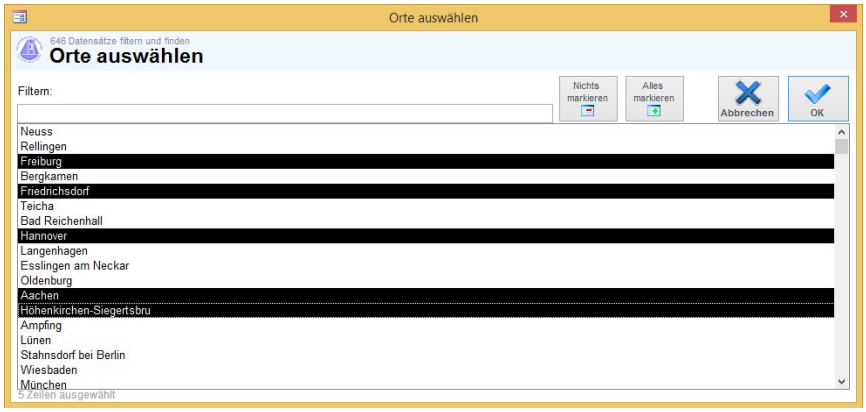
Außer in den angezeigten Spalten unterscheidet sich der Dialog auch je nach Einfach- oder Mehrfachmarkierung:

1. Wenn nur die Auswahl eines einzigen Wertes erlaubt ist, sind ausschließlich die Buttons Abbrechen und OK zu sehen.
2. Ist eine Mehrfachauswahl zulässig, gibt es zusätzlich die beiden Buttons *Nichts markieren* und *Alles markieren*.

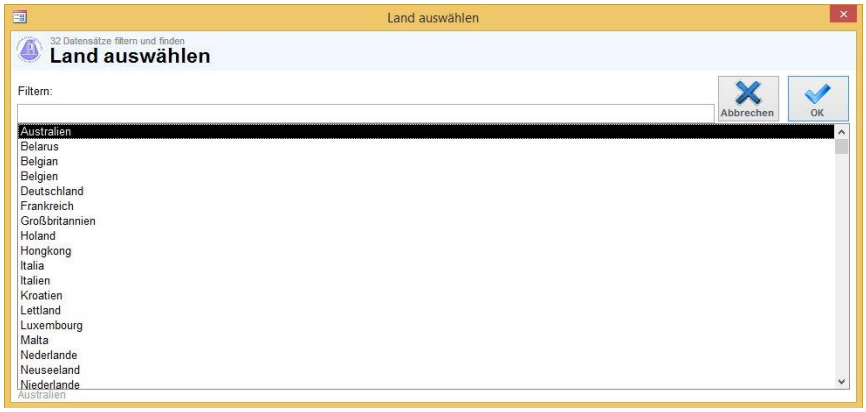
Die Unterscheidung zwischen Einfach- und Mehrfachauswahl erfolgt über einen Parameter beim Aufruf des Such-Formulars. Die folgende Abbildung zeigt das Such-Formular mit einer zweispaltigen Liste (und zulässiger Mehrfachauswahl) aus *boot_ID* und *boot_Name*:



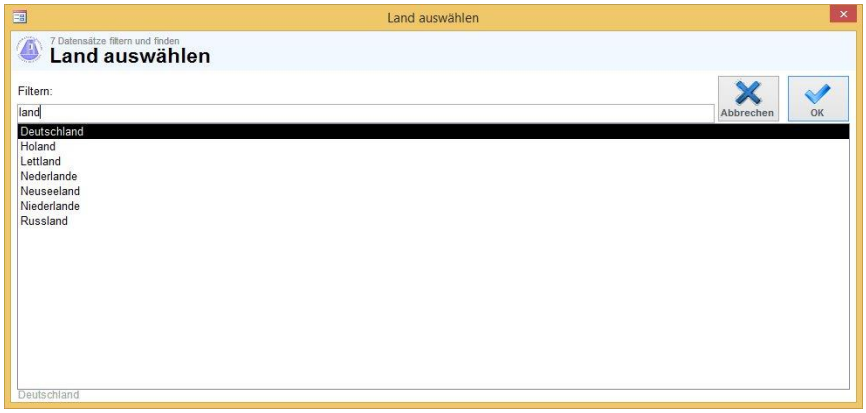
Da der Ort in *tblPersonen* kein Nachschlagefeld ist, führt die Suche danach im Formular zur einspaltigen Darstellung, wie die folgende Abbildung zeigt. Auch hier ist eine Mehrfachauswahl erlaubt, daher sind die beiden Buttons für *Nichts markieren* und *Alles markieren* sichtbar:



Wenn die Feldnamen für die anzuzeigenden Daten identisch sind, verbirgt der Dialog automatisch die erste Spalte. Das ist meistens für Felder sinnvoll, die nicht aus Nachschlagetabellen stammen und daher keine ID besitzen (hier ist nur Einfachmarkierung erlaubt):

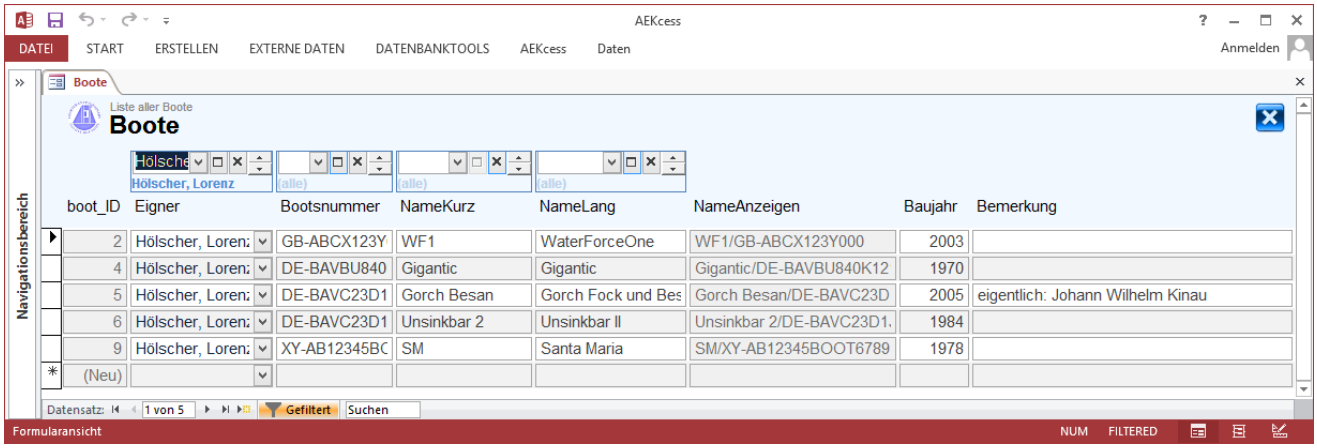


Das Eingabefeld über der Datenliste filtert die angezeigten Daten live bei der Eingabe, dabei werden beide Spalten berücksichtigt. So lässt sich die angezeigte Treffermenge deutlich einschränken und der gesuchte Inhalt schneller finden:



Filtern/Sortieren

Neben der Suche nach Datensätzen ist das Filtern und Sortieren in Formularen eine der wichtigsten Aufgaben. Damit das nicht in jedem Formular (und meistens sogar mehrfach für mehrere Felder!) neu programmiert werden muss, gibt es ein Formular *USys_sfmFilterSortierer*, welches diese Aufgabe kapselt. Dieses Formular lässt sich über der gewünschten Spalte als Unterformular einfügen und passt sich in der Breite an:

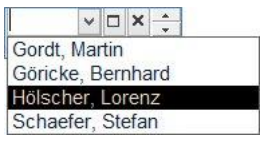


Es besteht links aus drei Controls für das Filtern und rechts aus zwei übereinanderliegenden Buttons zum Sortieren. Darunter steht ein Text, welcher den aktuellen Filter beschreibt:

Damit das Control funktioniert, benötigt es einige Zeilen VBA-Code im Elternformular, der weiter unten beschrieben wird.

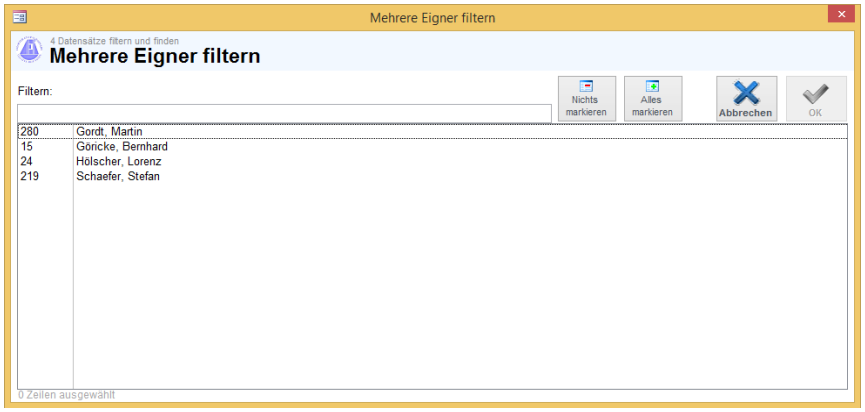
Filtern

Für eine schnelle Auswahl eines einzigen Wertes gibt es links eine Combobox, welche die per VBA übergebene Abfrage anzeigt:



Nach deren Auswahl wird das Formular sofort durch den entsprechenden lokalen Code gefiltert und das Control zeigt in der unteren Zeile den ausgewählten Wert an. Das ist vor allem für die wie auch hier oft sehr schmale Spaltenbreite im Formular gedacht, bei der die Combobox nur sehr eingeschränkt Werte anzeigen kann:

Soll aus sehr vielen Filterdaten oder mehrere Werte ausgewählt werden, zeigt ein Klick auf den Button mit dem Rechteck-Symbol das oben schon beschriebene Suchformular an:



Das ist typischerweise die gleichen Datenquelle wie in der Combobox, aber hier ist mehr Überblick, eine Vorfilterung und die Auswahl mehrerer Werte möglich.

Beide Filter werden durch Klick auf den Button mit dem X gelöscht und das Formular (für diese Spalte) wieder ungefiltert angezeigt.

Sortieren

Am rechten Rand sind im Control zwei Buttons übereinander zu sehen, welche die Sortierung dieser Spalte vorgeben. Sie kennen drei Zustände:

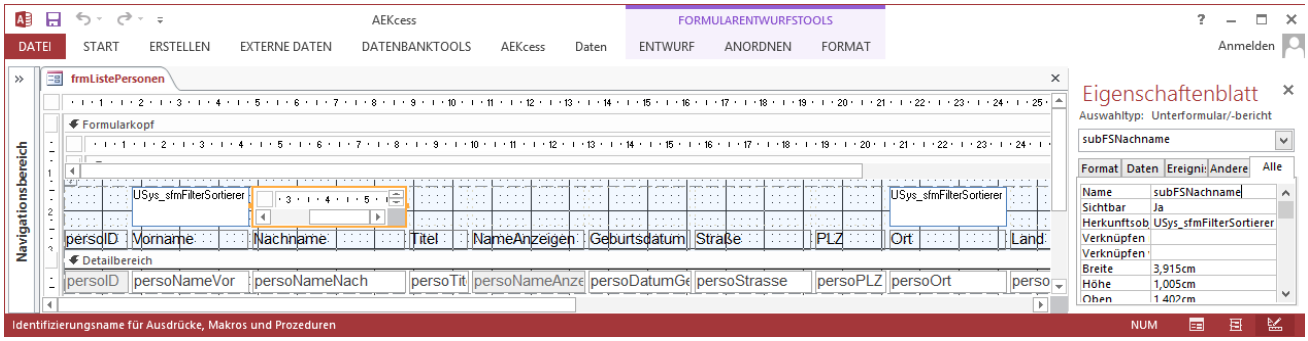
- Keiner der beiden Buttons ist gedrückt, die Spalte ist unsortiert.
- Der obere Button ist gedrückt, die Spalte wird aufsteigend sortiert.
- Der untere Button ist gedrückt, die Spalte wird absteigend sortiert.

Da es sich um Toggle-Buttons handelt, wird die Nicht-Sortierung durch erneuten Klick auf einen bereits gedrückten Button ausgewählt.

Notwendiger VBA-Code

Damit das Filter/Sortier-Control mit seinem jeweiligen Elternformular zusammenarbeitet, braucht es ein paar Zeilen VBA-Code, sowohl für die Parameter als vor allem auch für die ausgelösten Ereignisse. Am Beispiel des Personen-Listen-Formulars lässt sich das gut erläutern. Im Ergebnis sieht es wie in der folgenden Abbildung mit vier dieser Filter/Sortier-Controls aus. Eine Besonderheit in diesem Beispiel besteht darin, dass die Filterung des *persoAktiv*-Feldes gar nicht auf einer echten Abfrage basiert.

Im Entwurf sind mehrere Instanzen von *USys_sfmFilterSortierer* in die Layout-Tabelle eingefügt. Da der Controlname dabei sowieso nicht mehr gleichnamig mit dem Herkunftsobjekt sein kann, heißen diese Unterformulare einheitlich *subFS…* mit Hinweis auf die Spalte, in der Abbildung also *subFSNachname*:



Der folgende (gekürzte) Teil des VBA-Codes aus dem Elternformular *frmListePersonen* zeigt den Einsatz dieses FilterSortier-Controls. Damit die Ereignisse funktionieren, braucht es am Anfang des Moduls für jedes dieser Unterformulare eine *WithEvents*-Anweisung:

```
Public WithEvents frmFSNachname As Form_USys_sfmFilterSortierer
Public WithEvents frmFSAktiv As Form_USys_sfmFilterSortierer
```

In der *Form_Load*-Prozedur wird für jedes Unterformular das entsprechende Objekt verbunden und mit den gewünschten Parameter befüllt:

```
Set frmFSNachname = Me.subFSNachname.Form
With frmFSNachname
    .ComboAbfrage = "SELECT DISTINCT persoNameNach, persoNameNach FROM tblPersonen " & _
        "ORDER BY persoNameNach"
    .ComboListenbreiteCM = 5
    .DialogFeldID = "persoNameNach"
    .DialogFeldAnzeigen = "persoNameNach"
    .DialogTitel = "Mehrere Nachnamen filtern"
    .FormularFeldFiltern = "persoNameNach"
    .FormularFeldOrderBy = "persoNameNach"
End With
```

Dabei ist *ComboAbfrage* die Datenquelle und *ComboListenbreiteCM* die Listenbreite der Combobox. Das Suchformular wird über die *Dialog…*-Parameter gesteuert, dort müssen zwei Felder (*DialogFeldID* für die linke und wegen Gleichnamigkeit hier versteckte Spalte sowie *DialogFeldAnzeigen* für die rechte sichtbare Spalte) und der Titel des Formulars angegeben werden.

Da sich der Name des im Elternformular zum Filtern eingesetzten Feldes von demjenigen im Suchformular unterscheiden könnte, wird dies als *FormularFeldFiltern* explizit angegeben. Alleine für die Sortier-Buttons gilt das *FormularFeldOrderBy*, welches meistens mit *FormularFeldFiltern* identisch sein wird.

Das *persoAktiv*-Feld in diesem Beispiel enthält allerdings nur Ja/Nein-Werte und ist kein Nachschlagefeld mit einer Tabelle/Abfrage als Datenquelle. Damit auch für solche Fälle eine lesbare Auswahl möglich ist, täuscht die Funktion *ErzeugePseudoDaten()* eine Abfrage vor. Dabei wird aus den Wert/Name-Paaren des

Arguments mittels *SELECT*- und *UNION*-Anweisungen ein Recordset erzeugt. Schon bei kleineren Datenmengen (ca. 10-20 Datensätze) erscheint dabei jedoch sehr schnell die Fehlermeldung, dass dies zu komplex sei. Dann ist eine echte Abfrage nötig. Die bei der Pseudo-Abfrage entstehenden Felder heißen immer einheitlich *xID* und *xInhalt* und müssen genau so für *DialogFeldID* und *DialogFeldAnzeigen* angegeben werden:

```
Set frmFSAktiv = Me.subFSAktiv.Form
With frmFSAktiv
    .ComboAbfrage = ErzeugePseudoDaten("0;inaktiv;-1;aktiv")
    .ComboListenbreiteCM = 2
    .DialogFeldID = "xID"
    .DialogFeldAnzeigen = "xInhalt"
    .DialogTitel = "Mehrere Aktiv-Werte filtern"
    .FormularFeldFiltern = "persoAktiv"
    .FormularFeldOrderBy = "persoAktiv"
End With
```

Bei echten Nachschlagefeldern, die also mit Comboboxen im Elternformular dargestellt werden, würde die bisher beschriebene Technik allerdings dazu führen, dass nach deren ID und nicht nach deren angezeigtem Text sortiert würde. Daher unterscheidet sich das *FormularFeldOrderBy* dann wie im folgenden Code:

```
Set frmFSEigner = Me.subFSEigner.Form
With frmFSEigner
    .ComboAbfrage = "SELECT DISTINCT * FROM qryPersonenSortiert_Eigner ORDER BY persoNameAnzeigen"
    .ComboListenbreiteCM = 5
    .DialogFeldID = "boot_persoIDRef"
    .DialogFeldAnzeigen = "persoNameAnzeigen"
    .DialogTitel = "Mehrere Eigner filtern"
    .FormularFeldFiltern = "boot_persoIDRef"
    .FormularFeldOrderBy = "[Lookup_boot__persoIDRef].[persoNameAnzeigen]"
End With
```

Offensichtlich gibt es eine undokumentierte Eigenschaft in Access-Formularen, die ein internes Lookup-Feld enthält, wenn per Rechtsklick in einer Combobox mit Nachschlagefeld sortiert wird. Der Name des Feldes lässt sich dann in der Formular-Eigenschaft *Sortiert Nach* auslesen. Merkwürdigerweise findet sich in dessen Lookup-Feldnamen ein Unterstrich mehr als im Originalfeldnamen und es funktioniert trotzdem bzw. nur dann. Das verstehe, wer will…

Die einzelnen Elemente des Controls lassen sich deaktivieren mit

```
    .ComboAktiviert = False
    .DialogAufrufAktiviert = False
    .FilterAusAktiviert = False
```

Das mag insbesondere für die hier erkennbare Möglichkeit überraschen, auch den *FilterLöschen*-Button zu deaktivieren. Das wäre beispielsweise nötig, wenn hier alle Törns nur von denjenigen gesehen werden dürfen, welche die passenden Rechte besitzen. Das wären der Besitzer, dessen Gruppe/Niederlassung oder ein Admin. Wenn der Filter der Combobox dann löschbar wäre, hätten beliebige Benutzer plötzlich Zugriff auf alle Datensätze.

Damit in solchen Fällen die Combobox immer einen Wert und damit einen Filter für das Elternformular hat, lässt sich ein Standardwert für sie vorgeben. Das Argument gibt die Spalte (`0` für die erste, `1` für die zweite Spalte) an, wegen der meist eindeutigen ID ist die erste Spalte zu bevorzugen:

```
    .ComboValue(0) = 3    'zufälliger Wert als Beispiel, abhängig von den Inhalten
```

Bei mehreren Filter/Sortier-Controls in einem Elternformular können diese von dort aus gemeinsam auf gewünschte Standardwerte zurückgestellt werden. Dabei werden allerdings für jedes Control dessen Events einzeln ausgelöst, die wiederum eine meist langsame Sortierung/Filterung im Elternformular verursachen. Um das zu vermeiden, können die Control-Events deaktiviert werden, beispielsweise in einer solchen Prozedur des Elternformulars, um alle Filter zu leeren:

```
Sub AlleFilterLeeren()
    frmFSVorname.EventAktiv = False
    frmFSNachname.EventAktiv = False
    frmFSOrt.EventAktiv = False

    frmFSVorname.FilterAus
    frmFSNachname.FilterAus
    frmFSOrt.FilterAus

    frmFSVorname.EventAktiv = True
    frmFSNachname.EventAktiv = True
    frmFSOrt.EventAktiv = True

    frmFSAktiv.FilterAus
End Sub
```

Dabei ist das *frmFSAktiv*-Control als zuletzt aufgerufenes ausdrücklich nicht deaktiviert, sondern sorgt durch seinen "normalen" Aufruf für das Auslösen des Events, damit im Elternformular genau einmal alle Filter/Sortier-Prozeduren aufgerufen werden. Andernfalls wären zwar alle Controls geleert, aber das Elternformular selber nicht passend gefiltert/sortiert.

Egal, ob der letzte Filter aus der Combobox oder über das Suchformular kam, kann dieser manuell durch Klick auf den Button mit dem X gelöscht werden. Per VBA entspricht das dem Befehl:

```
    .FilterAus
```

Das Gegenstück für das Aufheben der Sortierung ist:

```
    .SortierungAus
```

In beiden Fällen werden die Anzeigen aktualisiert und die notwendigen Events ausgelöst, falls das nicht ausdrücklich deaktiviert wurde.

Mehrfachnutzung

Kopfzeilen in Formularen und Berichten

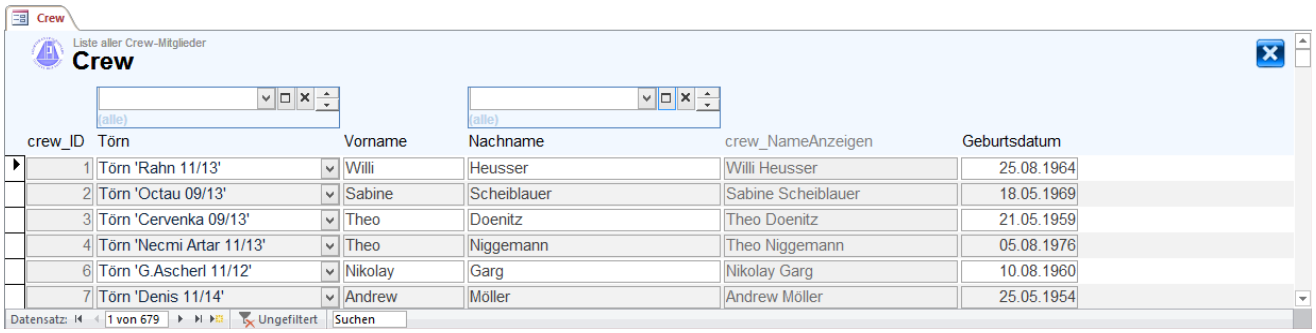
Die Formular- und Berichtsköpfe sind hier nicht mit dem Standardverfahren von Access über *Entwurf > Kopfzeile/Fusszeile > Logo* bzw. *Titel* eingefügt, wie das auch vom Assistenten gemacht wird. Das führt nämlich zu einer ziemlich statischen Gestaltung, bei der nachträglich weder die Position der Elemente angepasst noch weitere Objekte sinnvoll eingefügt werden können.

Dazu müsste dann ziemlich mühsam jedes Formular und jeder Bericht doch wieder einzeln im Entwurf geöffnet werden, weil die Designs nur die Schrift oder die Farben einheitlich anpassen können. Die bessere Lösung mit den programmierbaren Unterformularen und –berichten habe ich bereits 2009 auf der AEK 12 ("Kleine Helferlein für Access: Accessoires") vorgestellt und will das hier nicht wiederholen.

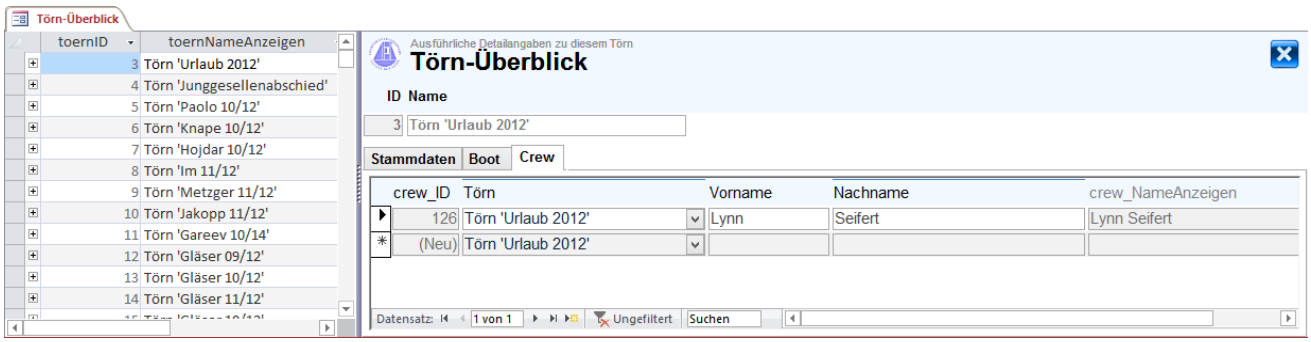
Hauptformulare als Unterformulare

Es kommt ziemlich häufig vor, dass die Information aus einem Hauptformular an anderer Stelle wegen einer 1:n-Verknüpfung ebenfalls als Unterformular benötigt wird. Da wäre es Arbeitsbeschaffung, davon eine Kopie anzufertigen, nur damit die überflüssigen Kopf-Elemente entfernt werden können.

Dieses Problem tritt in der Beispiel-Datenbank auf bei der Darstellung der Crew-Mitglieder. Diese werden in einem Listen-Formular wie in der folgenden Abbildung angezeigt:



Diese Listen-Darstellung wird sinnvollerweise aber auch im Crew-Unterformular (unten rechts, im Registersteuerelement) vom Törn-Überblick eingesetzt:



Natürlich ist es kein Problem, das Formular *frmListeCrew* einfach als Herkunftsobjekt für das Unterformular anzugeben. Dann würde es jedoch weiterhin seinen Formularkopf mit dem Logo und den beiden Titeln anzeigen, was viel Platz verbraucht und in Konkurrenz zum eigentlichen Elternformular-Kopf sehr irritieren würde.

Daher gibt es eine Prozedur *SubFormOhneKopf* mit dem Unterformular als Parameter, welches dafür sorgt, dass dort dessen eingebetteter Formularkopf und (falls vorhanden) auch dessen Filter/Sortier-Controls nicht angezeigt werden. Diese Prozedur wird im Elternformular nach dem Zuweisen dieses Unterformulars aufgerufen.

Leider reicht es nicht, deren `Visible`-Eigenschaft auf `False` zu stellen, weil solche Elemente weiterhin ihren Platz verbrauchen. Da auch die Namen und Menge der Labels für die Spaltenüberschriften variieren können, stellt der Code also einfach die Höhe für alle Nicht-Label-Controls auf `0`.

Anschließend wird die Höhe des Formularkopfs auf einen sicher viel zu kleinen Wert von 10 Twips gestellt. Access korrigiert diesen selbstständig und ohne Fehlermeldung auf die tatsächlich notwendige Höhe, so dass damit automatisch auch mehrzeilige Label-Anordnungen berücksichtigt werden.

Fußzeilen im Bericht

Eigentlich wäre es schlau, das Konzept der eingebetteten Kopfzeilen auch auf Fußzeilen anzuwenden, insbesondere im Bericht. Dort werden unten auf der Seite gerne die typischen Elemente wie Berichtsname, Druckdatum/-uhrzeit und Seitenzahl angegeben. Da das für alle Berichte ebenfalls einheitlich sein soll, gehört das also in einen Unterbericht.

Wie der Konjunktiv im vorherigen Absatz schon vermuten lässt: Leider funktioniert das nicht besonders gut, denn das `[Seite]` bzw. `[Seiten]`-Feld steht nun im Unterbericht und zeigt nur *dessen* Seitenzahl. Im Gegensatz zum Elternbericht besteht dieser aber nur aus einer einzigen Seite und zeigt das unveränderlich an.

Die einzige Chance, dem entgegenzuwirken, besteht darin, per VBA aus dem Elternbericht und dessen `OnFormat`-Ereignis die aktuelle Seitenzahl zu ermitteln und stattdessen in ein Label des Unterberichts hineinzuschreiben. Das funktioniert zwar technisch, hat sich in der Praxis aber als so unzuverlässig herausge-

stellt, dass ich es in neue Datenbanken gar nicht mehr integriere. Sobald nämlich Daten erst nach verschiedenen Versuchen (Gruppe zusammenhalten o.ä.!) endgültig auf einer Seite platziert werden, stimmt die Seitenzahl häufig nicht. Selbst die Berücksichtigung von `FormatCount` im *Detailbereich_Format* war mir nicht ausreichend zuverlässig. Daher enthält *AEKcess* dieses Feature ausdrücklich nicht.

Auch der Berichtsname lässt sich in einem Unterbericht übrigens nicht mal eben mit einer fertigen Funktion anzeigen. Immerhin kann eine eigene Funktion mit `Me.Parent.Name` diesen Wert ohne Schwierigkeiten ermitteln.

Ich beschränke mich daher für Berichte inzwischen darauf, nur die unproblematischen Informationen (z.B. Druckdatum/-uhrzeit, aktueller Anwender, Berichtsname, o.ä.) in einem Teilbereich der Fußzeile mit einem Unterbericht darzustellen. Die kritischen Daten (also die aktuelle und die Gesamt-Seitenzahl) hingegen stehen wie lokal im Bericht auf dem verbleibenden Rest der Fußzeile.

Optische Änderungen an der Seitenzahl führen dann also doch wieder zu umfangreichen Nacharbeiten an jedem Berichtsentwurf, aber wenigstens ein Teil der Fußzeile ist zentral.

Export

Das Wichtigste an Access ist für die meisten Anwender offenbar die Möglichkeit, diese Daten nach Excel zu kopieren und dort nach Belieben "nachzubessern". In typischen Datenbanken gibt es also dauernd Gelegenheiten, bei denen Daten exportiert werden müssen.

In vielen Fällen könnte ich schlicht den Access-eigenen Export-Assistenten aufrufen und hätte einen Teil der Anforderungen gelöst. Daher ist dieser im *AEKcess*-Ribbon auch enthalten. Hauptkritikpunkt daran ist aber die Notwendigkeit, immer wieder einen Pfad und Dateinamen für den Export angeben zu müssen. Das führt dazu, dass es in Firmen einen regelrechten Wildwuchs an exportierten Dateien in allen möglichen und unmöglichen Verzeichnissen gibt.

In *AEKcess* besteht der Export aus den Modulen *USys_modExport* mit allgemeinen und *modExport* mit spezifischen Prozeduren. Dazu kommen jeweils Klassenmodule *clsExport…*, welche die Details und eventuell abweichenden Methoden beschreiben. Ein Button, Menü oder eine sonstige Prozedur ruft dann einfach eine Funktion (wegen Menü-Aufruf notwendig) *ExportiereBooteXLSX* auf, die beispielsweise wie folgt deklariert ist:

```
Function ExportiereBooteXLSX() As Boolean
    Exportieren expBoote, typExcel, False
End Function
```

Die zentrale Prozedur *Exportieren* entscheidet dann anhand dieser beiden Enumerationswerte *expBoote* und *typExcel*, dass also die Klasse *clsExportBoote* benutzt und in das Excel-Format geschrieben werden soll. Alle Exportdateien werden hier grundsätzlich in einem *Export*-Unterverzeichnis des Datenbank-Pfads gespeichert. In "echten" Installationen kann das alternativ auch das *Eigene Dokumente*-Verzeichnis des jeweiligen Benutzers sein, um Schreibkonflikte zu vermeiden und die Dateien anschließend leicht wieder auffinden zu können.

Damit gleiche Exportinhalte nicht immer wieder neue Dateinamen erzeugen, habe ich hier festgelegt, dass der Dateiname nur von der Klasse abhängig ist. Diese Information liefert das Klassenmodul mit der Get-Property *clsExport_Dateiname* (der etwas eigenwillige Prozedurname ergibt sich aus der *Implements*-Anweisung), wie der Code der Klasse hier im Ausschnitt zeigt:

```
Property Get clsExport_Dateiname() As String
    clsExport_Dateiname = "Export_Boote"
End Property
```

Mit der Klasse *clsExportDiesesObjekt* hingegen kann das jeweils aktive Formular oder der aktive Bericht exportiert werden, daher ist dort der tatsächliche Dateiname vom Objektnamen abhängig und der Code entsprechend ausführlicher:

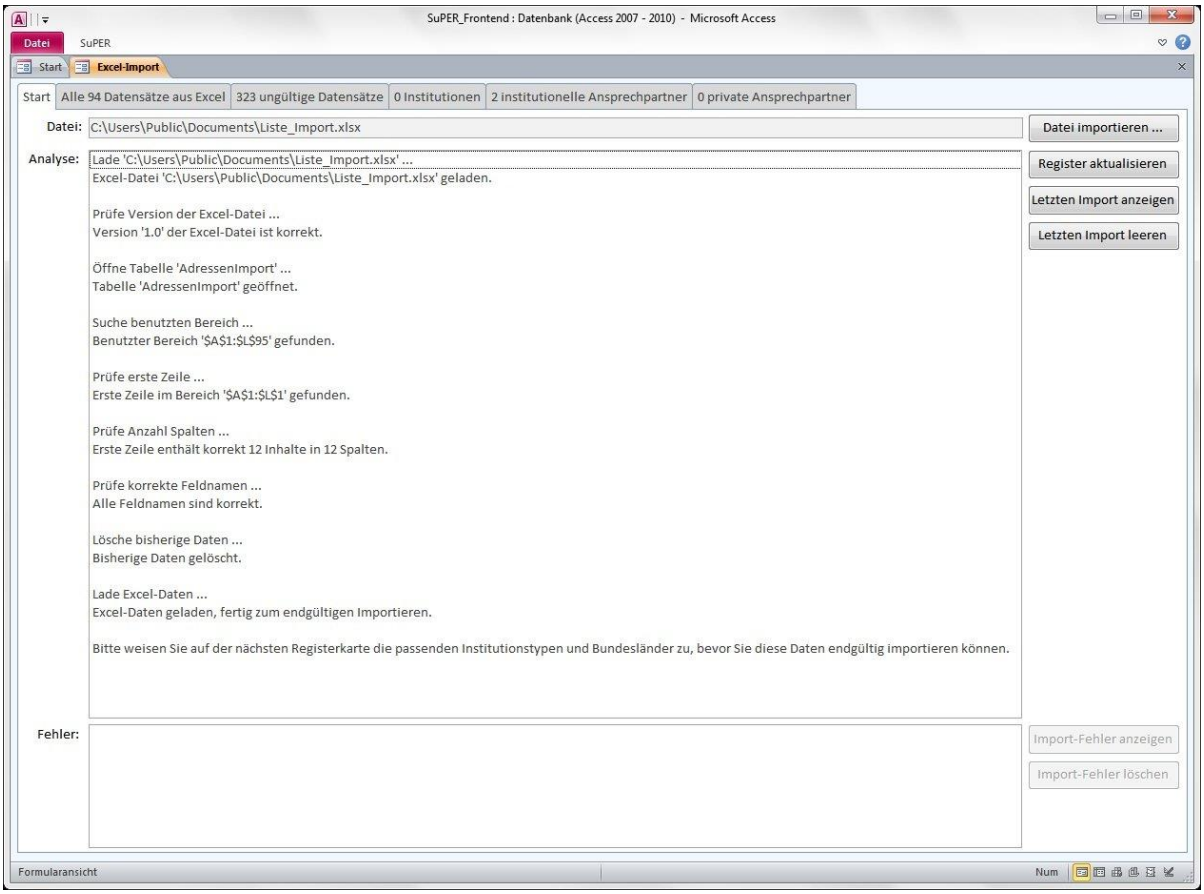
```
Property Get clsExport_Dateiname() As String
    Dim objScreen As Object 'Formular oder Bericht

    On Error Resume Next
    Set objScreen = Screen.ActiveForm
    Set objScreen = Screen.ActiveReport
    On Error GoTo 0
    clsExport_Dateiname = "Export_" & objScreen.Name
    'Sonderzeichen entfernen, die in Dateinamen nicht erlaubt sind
    '(gekürzt)
End Property
```

Mit dieser Konstruktion lassen sich beliebige Export-Möglichkeiten leicht durch jeweils neue Klassenmodule (und einen passenden Enumerationswert) ergänzen.

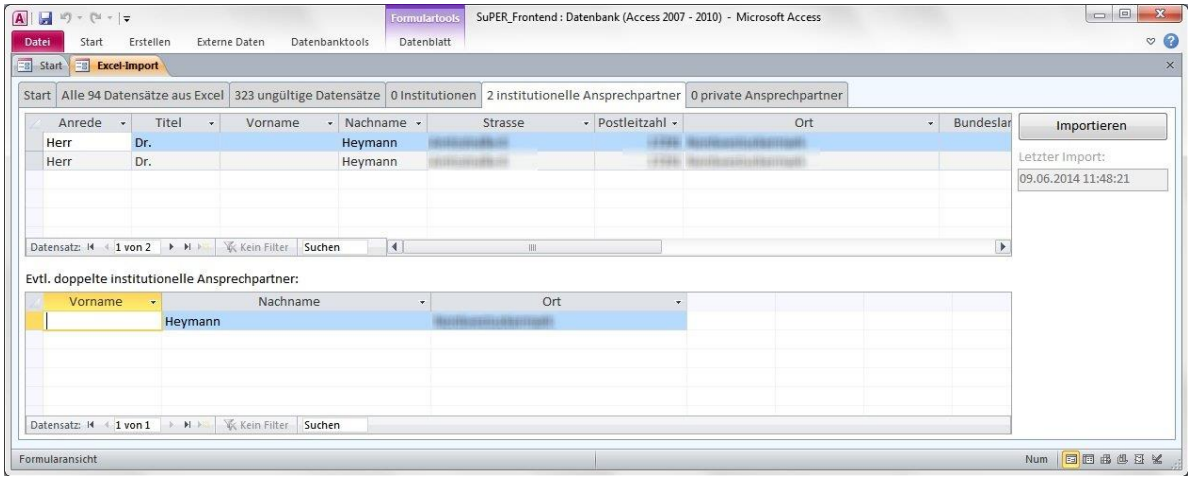
Einen Import habe ich hier ausdrücklich nicht integriert. Technisch funktioniert das zwar ebenso wie der Export und benötigt dann erst einmal nur ein entsprechendes Set an Modulen und Klassenmodulen. Inhaltlich sind aber so viel mehr Details zu prüfen, dass es sich nicht brauchbar standardisieren lässt. Es beginnt mit der Kontrolle, ob die zu importierende Excel-Datei überhaupt die erwartete Anzahl an Spalten mit den richtigen Feldnamen und Datentypen enthält.

Beispielhaft ist das hier an einem anderen Projekt zu sehen, wobei die übrigen Registerkarten dieses Formulars erst nach der Analyse sichtbar wurden:



Nachdem die Excel-Datei syntaktisch als korrekt eingestuft und die Daten in eine Zwischentabelle importiert wurden, müssen diese noch auf Zulässigkeit und Sinnhaftigkeit geprüft werden. Die Daten könnten ja bereits in der Datenbank enthalten sein, nicht zu vorhandenen Datensät-

zen passen, fehlende Referenzwerte nutzen oder mehrfach aufgelistet sein. Exemplarisch ist das zu sehen im Register "Institutionelle Ansprechpartner", wo der gleiche Name mehrfach in der Importdatei enthalten war:



Alle diese Prüfungen sind so speziell von der Datenbank abhängig, dass ich bisher keine allgemeine Struktur dafür finden konnte, obwohl natürlich viele dieser prüfenden Elemente von Datenbank zu Datenbank weiterkopiert und angepasst werden.

Entwickler-Modus

Der Entwickler der Datenbank muss für seine Arbeit meistens mehr sehen, dürfen und können als selbst der (Daten-)Administrator. Es ist aber ziemlich lästig, sich diese zusätzlichen Fähigkeiten immer wieder zu geben oder zu entziehen. Das betrifft vor allem das Ausblenden der Standard-Register im Ribbon, welches einen schreibenden Eingriff in die (meistens sogar ausgeblendete) Tabelle *USysRibbon* erfordert.

Zwei Menübefehle zeigen daher die einfache Umschaltung zwischen dem, was ein Entwickler möchte und dem, was ein Benutzer darf:

- Ribbon umschalten
- Allgemeine Parameter ändern

Passend zum Konzept der Menürechte sind diese Befehle sinnvollerweise selbstverständlich nur dem Entwickler zugänglich.

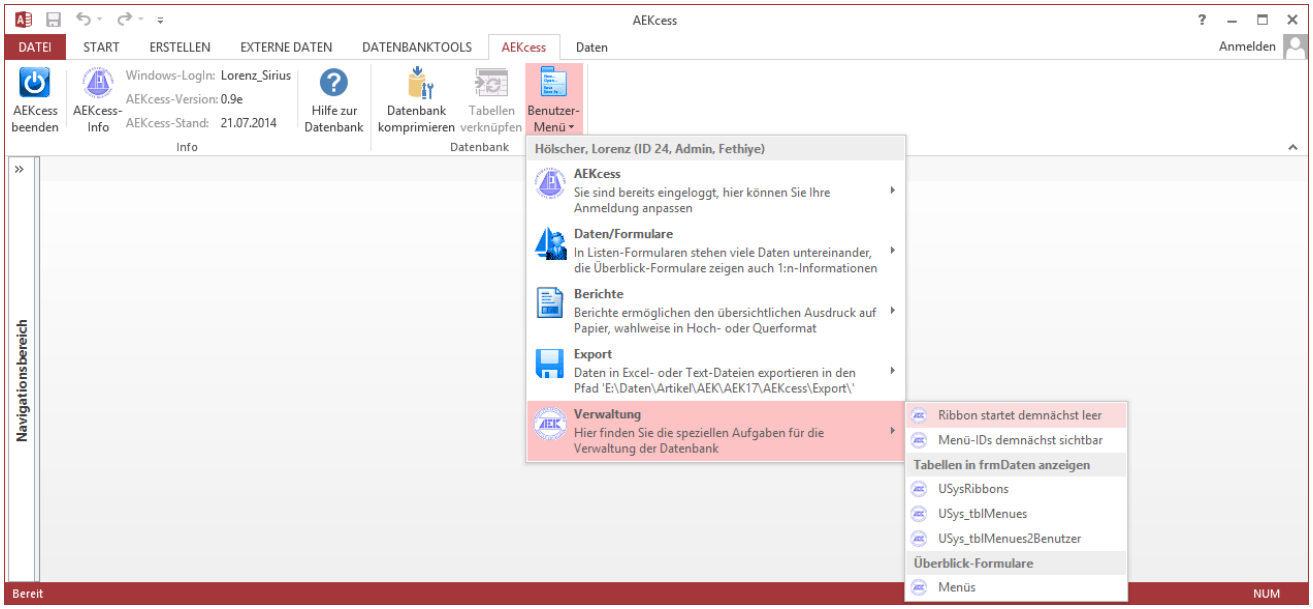
Um das Ribbon (speziell die Einstellung `<ribbon startFromScratch="true">`) umzuschalten, gibt es die Funktion *RibbonStartFromScratchWechseln* im Modul *USys_modFunktionen*. Sie ermittelt den aktuellen Inhalt von `startFromScratch` und ändert ihn auf `True` bzw. `False`. Anschließend muss die Datenbank erneut eingelesen werden, was am einfachsten mittels Komprimieren erfolgt.

Für die wechselnde Anzeige im Menü gibt es zusätzlich die Funktion *IstRibbonStartFromScratch*, welche den aktuellen Zustand ermittelt und im Menütitel mit

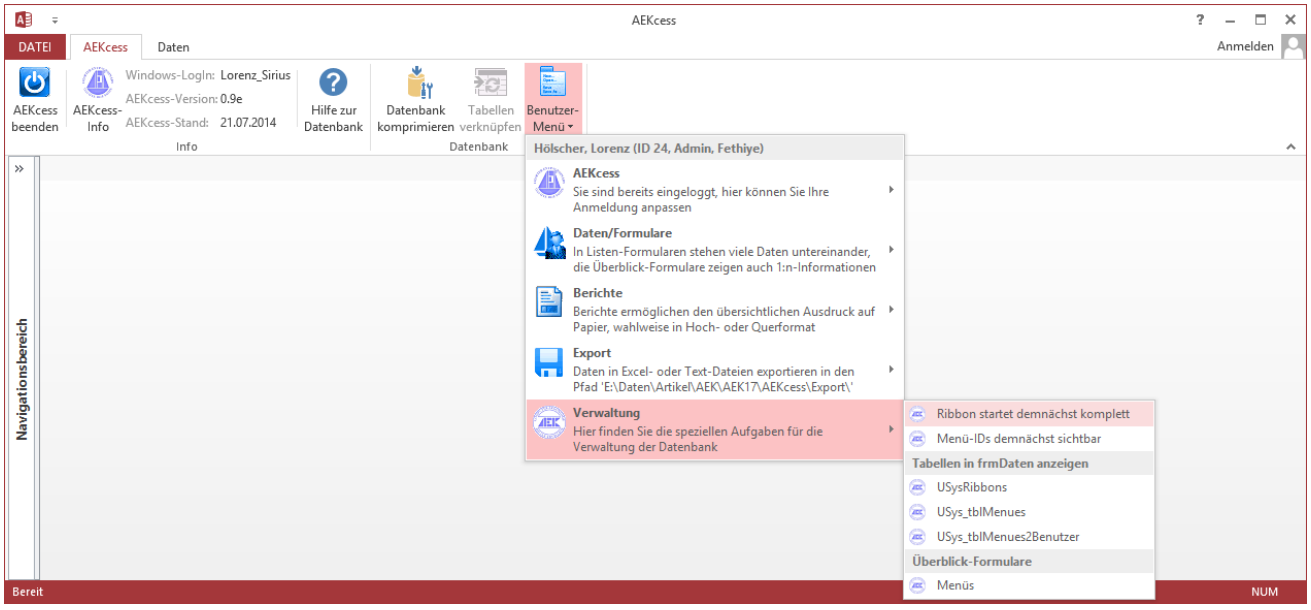
```
="Ribbon startet demnächst " & Iif(IstRibbonStartFromScratch(),"leer","komplett")
```

den passenden Text anzeigt.

Die folgende Abbildung zeigt Menüeintrag und Ribbon im Entwickler-Modus, also das Ribbon noch mit allen Standard-Registerkarten:



Nach Anklicken von "Ribbon startet demnächst leer" und erneutem Öffnen der Datenbank präsentiert sich diese korrekt für normale Anwender ohne die Standard-Registerkarten. Weil immer noch der Datenbank-Entwickler angemeldet ist, könnte er diese mit dem entsprechenden Menübefehl wieder zurückstellen.



Andere Anpassungen in der Datenbank können mit einer einfachen Variablen und `If`/`End If` im Code gesteuert werden. Um die Variablen dauerhaft festlegen zu können, bedarf es aber der benutzerdefinierten Datenbank-Eigenschaften. Dies ist hier exemplarisch für die Idee durchgeführt worden, vor den Menüeinträgen die *menueID* zu sehen, damit zufällig gleichnamige Menütitel besser unterschieden werden können.

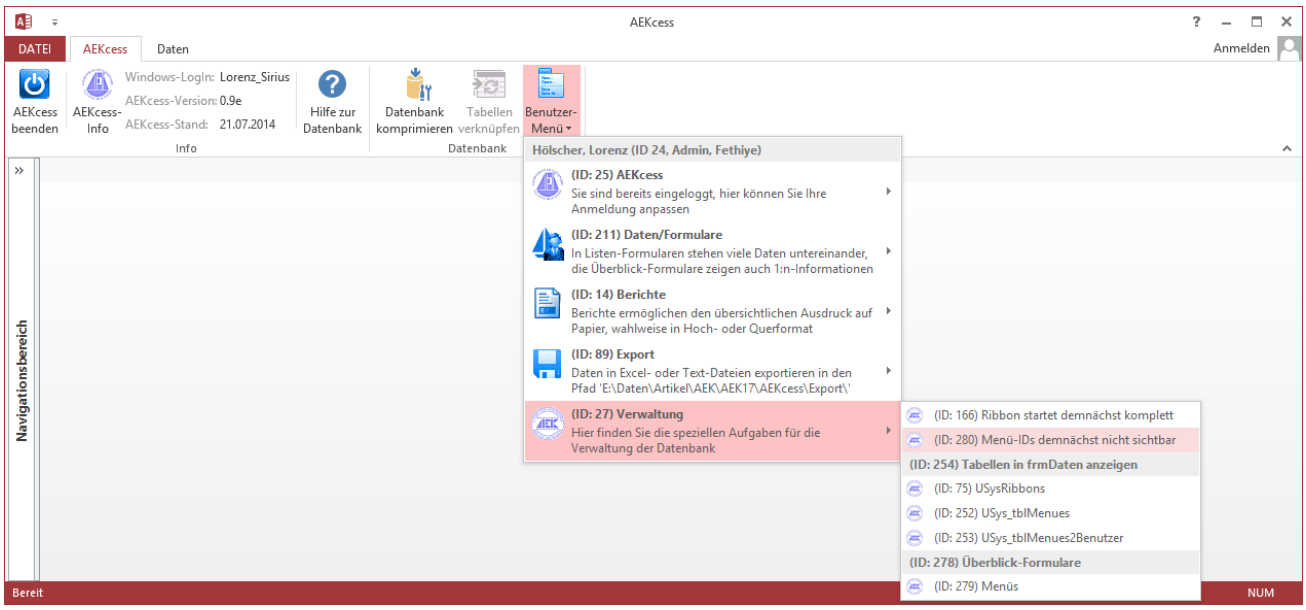
Anhand der Property *MenueIDsSichtbar* und der Formel

```
="Menü-IDs demnächst " & Iif(MenueIDsSichtbar(),"nicht ","") & "sichtbar"
```

in *tblMenues* wechselt der Menütext passend zur aktuellen Einstellung.

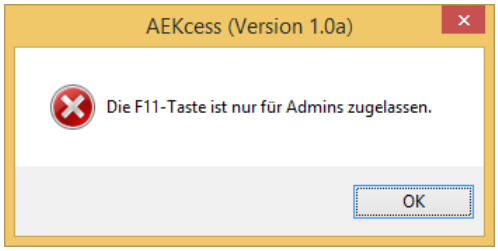
Der Klick auf "Menü-IDs demnächst sichtbar" ruft die Funktion *MenueIDSichtbarWechseln* auf. Diese nutzt die Property *MenueIDsSichtbar* zur Änderung des -1/0-Werts für die benutzerdefinierte Eigenschaft *Hat-*

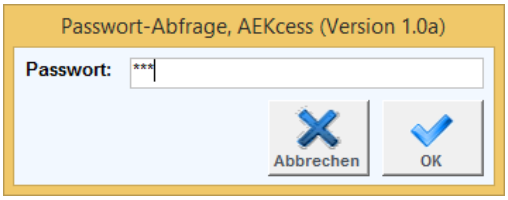
MenueID. Dann wird (sofort und ohne Datenbank-Komprimierung) beim nächsten Aufruf des Menüs vor jedem Text die zugehörige *menueID* eingeblendet:



Mit dieser Technik lassen sich beliebige Einstellungen für eine Datenbank dauerhaft für Entwickler und Benutzer festlegen.

Damit normale Anwender nicht einfach mit der F11-Taste in den Navigationsbereich wechseln, ist diese via *AutoKeys*-Makro abgefangen. Ohne *AEKcess*-Anmeldung, also mit Gast-Status, wird der Zugriff komplett blockiert (Bild links). Als angemeldeter Benutzer müssen Sie trotzdem die entsprechenden Admin-Rechte laut *USys_tblBenutzer*-Tabelle besitzen, damit Sie nach Password-Eingabe (Bild rechts, hier lautet es übrigens "AEK") den Navigationsbereich sehen können:





Ausblick

AEKcess enthält natürlich vor allem Konzepte und Entwicklungen, wie sie speziell für meine Kunden wichtig sind oder waren. Alles, was ich hier vorgestellt habe, ist für AEK-Teilnehmer frei zur Eigennutzung und Weiterentwicklung. Ich freue mich über Rückmeldungen, Lob oder Verbesserungsvorschläge.

Nun könntet Ihr diese *AEKcess*-Datenbank einfach als Steinbruch benutzen und alles herausklauben, was Euch geeignet erscheint. Oder wir entwickeln sie gemeinsam mit den guten Ideen aus Euren Datenbanken weiter und ergänzen das, was noch gut dazu passen würde.

Lorenz Hölscher