HAUFE.
Ausgabe 02/2022

ACCESS
IM UNTERNEHMEN

BEISPIELDATEN GENERIEREN

Nutzen Sie .NET-Techniken, um per Mausklick Beispieldaten für Ihre Entwicklungsprojekte zu generieren (ab Seite 55).

In diesem Heft:

E-MAIL-ANLAGEN

Fügen Sie einer Mail unter Outlook flexibel eine oder mehrere Anlagen hinzu und versenden Sie diese.

SEITE 69

DATEIAUSWAHL PER FILEDIALOG

Wählen Sie Dateien zum Öffnen oder Speichern aus oder selektieren Sie Verzeichnisse mit der FileDialog-Klasse.

SEITE 39

SQL SERVER-SICHERHEIT, TEIL 7

Realisieren Sie Sicherheitsmechanismen über die Windows-Authentifizierung.

SEITE 2

Beispieldaten mit .NET

Wenn Sie schon einmal eine Datenbankanwendung von Grund auf erstellt haben, für die es auch noch keine Vorgängeranwendung gab, stehen Sie vor einem Problem: Woher nehmen Sie die Daten, um die Tabellen, Abfragen, Formulare, Berichte und den VBA-Code Ihrer Anwendung zu testen? Natürlich können Sie immer wieder von Hand neue Kundendatensätze eingeben, die Sie sich zuvor ausgedacht haben, aber das ist bestenfalls für das Testen eines dafür vorgesehenen Eingabeformulars sinnvoll.

Wer mehr testen möchte als nur die Eingabe von Daten in ein dafür vorgesehenes Formular, benötigt dazu gegebenenfalls eine ganze Menge von Daten – und zwar solche Daten, die immer wieder schnell rekonstruiert werden können.

Für Access beziehungsweise VBA gibt es keine fertige Lösung, um Tabellen schnell mit Daten zu füllen, die dem jeweiligen Datentyp entsprechen. Aber wozu gibt es .NET? Durch die gute Erweiterbarkeit von .NET-Anwendungen gibt es zahllose Erweiterungen, darunter auch solche, mit denen sich Beispieldaten generieren lassen. Eine davon heißt Bogus und diese stellen wir im Beitrag **Beispieldaten generieren mit .NET und Bogus** ab Seite 55 vor. Die Bibliothek stellt Klassen und Funktionen bereit, mit denen Sie verschiedenartige Daten generieren können, und zwar in beliebigen Mengen. So lassen sich etwas Adressdatensätze zusammenstellen und Bestellungen, und Sie finden auch noch Möglichkeiten, um die so generierten Bestellungen den Adressen zuzuordnen.

Das funktioniert leider nicht von Access aus, sondern wir benötigen ein Tool, mit dem wir auf einfache Weise .NET-Code programmieren und laufen lassen können. Dazu nutzen wir das Tool LINQPad. Es erlaubt das einfache Schreiben und Testen von Code – und das eben auch für die zuvor beschreibene Lösung zum Erstellen von Beispielcode. Alles über LINQPad und wie Sie damit auf die Tabellen von Access-Datenbanken zugreifen können, lernen Sie im Beitrag **Datenzugriff mit .NET, LINQPad und LINQ to DB** ab Seite 49. Hier nutzen wir zusätzlich die Bibliothek LINQ to DB für den Datenzugriff.

Auch in unserer Reihe zum Thema Sicherheit und SQL Server geht es weiter. Unter dem Titel **SQL Server-Security – Teil 7: Windows-Authentifizierung** zeigt unser Autor Bernd Jungbluth ab Seite 3, wie Sie die Windows-Authentifizierung für die Steuerung des Zugriffs auf SQL Server-Datenbanken nutzen können und was es dabei zu beachten gibt.

Bei der Entwicklung von Anwendungen benötigen Sie früher oder später Dateiauswahldialoge. Diese stellt Office als Bordmittel bereit. Dennoch kommen Sie um ein wenig VBA-Programmierung nicht herum. Damit alles wie gewünscht gelingt, wenn Sie Dateien zum Öffnen oder Speichern auswählen wollen oder wenn es darum geht, einen Ordner zu selektieren, hilft Ihnen der Beitrag **Dateien und Verzeichnisse auswählen mit FileDialog** ab Seite 39 weiter.

Vielleicht möchten Sie die dort gewählten Dateien ja als Anlage zu einer neuen E-Mail hinzufügen und diese so verschicken? Dann finden Sie im Beitrag **E-Mails mit Anlagen mit Outlook versenden** ab Seite 69 noch die richtigen Programmierungen. Fügen Sie ganz einfach eine oder mehrere Dateien aus einem oder mehreren verschiedenen Verzeichnissen zur Mail hinzu und senden Sie diese per Mail an den gewünschten Empfänger!

Nun viel Spaß beim Lesen!

Ihr André Minhorst

SQL Server-Security – Teil 7: Windows-Authentifizierung

Bernd Jungbluth, Horn (info@berndjungbluth.de)

Kennwörter. Sie sind ein mehr oder weniger notwendiges Übel, stellen Sie doch den Zugang zu Systemen sicher. Dennoch sind sie nicht selten eine Schwachstelle im Sicherheitskonzept, wie im aktuellen Szenario dieser Beitragsreihe. Für eine automatische Anmeldung am SQL Server wird der Anmeldename und das Kennwort in der Access-Applikation gespeichert. Dort aber lassen sich diese Informationen nicht ausreichend schützen. Dabei kann es so einfach sein. Mit der Windows-Authentifizierung sind die Anmeldedaten für den Zugang zum SQL Server nicht mehr erforderlich. In diesem Teil der Beitragsreihe lernen Sie die Vorteile der Windows-Authentifizierung kennen.

Warnung

Die beschriebenen Aktionen haben Auswirkungen auf Ihre SQL Server-Installation. Führen Sie die Aktionen nur in einer Testumgebung aus. Verwenden Sie unter keinen Umständen Ihren produktiven SQL Server!

Das aktuelle Berechtigungskonzept der Beispielumgebung basiert auf der SQL Server-Authentifizierung. Der Zugang zum SQL Server erfolgt über die Anmeldungen **WaWiMa** und **WaWiPersonal**. Ein Anwender braucht lediglich das Kennwort zu einer der beiden Anmeldungen und schon kann er mit den entsprechenden Rechten auf die Daten des SQL Servers zugreifen. Einen direkten Bezug zum tatsächlichen Anwender gibt es bei dieser Art der Authentifizierung nicht.

Das ist bei der Windows-Authentifizierung anders. Hier müssen Sie keinen Anmeldenamen mitsamt Kennwort anlegen, sondern sie verweisen lediglich auf das Windows-Benutzerkonto des Anwenders. Diese so erstellte Anmeldung ordnen Sie dann wie gewohnt der Datenbank zu. Dem dadurch in der Datenbank erstellten Benutzer erteilen Sie dort die Rechte über die Datenbankrollen. Es ändert sich also lediglich die Authentifizierung am SQL Server, die Autorisierung an der Datenbank und die darauf folgende Rechtevergabe bleiben gleich.

Nur haben Sie weniger Aufwand bei der Verwaltung der Anmeldungen und durch den Wegfall der Kennwörter erhalten Sie mehr Sicherheit. Mehr noch, denn durch den Verweis zum Windows-Benutzerkonto erfolgt der Datenzugriff immer mit direktem Bezug zum Anwender. Ein anonymer Zugriff ist nicht mehr möglich. Sie sehen, die Windows-Authentifizierung ist einfach und effektiv.

Sie kennen die Windows-Authentifizierung bereits von Ihrem Windows-Benutzerkonto. Das haben Sie bei der Installation des SQL Servers als Anmeldung zugeordnet. Sie finden es im SQL Server Management Studio unter **Sicherheit\Anmeldungen**. Diese Anmeldung ist Mitglied der Serverrolle **sysadmin** und bietet Ihnen somit alle Rechte innerhalb Ihres SQL Servers, weshalb sie sich nicht für einen adäquaten Test der Rechtevergabe eignet. Dazu benötigen Sie ein weiteres Windows-Benutzerkonto.

Ein Windows-Benutzerkonto

Möglicherweise arbeiten Sie mit Ihrem Rechner in einer Windows-Domäne und Ihr Windows-Benutzerkonto wird über ein Active Directory verwaltet. Vielleicht aber testen Sie die Beispiele dieser Beitragsreihe in einer Entwicklungsumgebung mit einem lokalen Windows-Benutzerkonto. Da das Anlegen eines Windows-Benutzerkontos im Active Directory an fehlenden Rechten oder mangelnder

Bereitschaft Ihres IT-Systemadministrators scheitern könnte, legen Sie für die weitere Vorgehensweise ein lokales Windows-Benutzerkonto an. Dazu klicken Sie mit der rechten Maustaste auf das Windowslogo in der Taskleiste und wählen den Eintrag **Computerverwaltung** (siehe Bild 1). In der Computerverwaltung erweitern Sie auf der linken Seite den Ordner **Lokale Benutzer und Gruppen**.

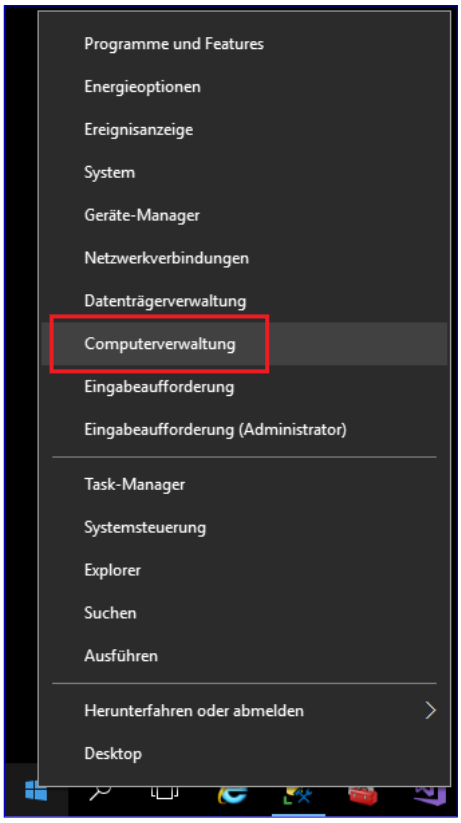

Bild 1: Start der Computerverwaltung

Nach einem Rechtsklick auf den Unterordner **Benutzer** öffnen Sie mit dem Eintrag **Neuer Benutzer** den gleichnamigen Dialog. Hier geben Sie als Benutzernamen **Bienlein** ein. Anschließend sichern Sie das Benutzerkonto mit einem Kennwort in den Eingabefeldern **Kennwort** und **Kennwort** bestätigen. Eine Neuvergabe des Kennworts bei der ersten Verwendung des Windows-Benutzerkontos ist nicht erforderlich. Es handelt sich schließlich um ein Testkonto, das Sie selbst verwenden. Daher können Sie die Option **Benutzer muss Kennwort bei der nächsten Anmeldung ändern** deaktivieren. Mit einem Klick auf **Erstellen** wird das neue Windows-Benutzerkonto angelegt (siehe Bild 2).

Glückwunsch. Nun haben Sie auf Ihrem Rechner einen virtuellen Anwender namens **Bienlein**. Beenden Sie den Dialog mit der Schaltfläche **Schließen**.

Die Anmeldung Bienlein

Im nächsten Schritt erstellen Sie für das Windows-Benutzerkonto **Bienlein** eine Anmeldung in Ihrem SQL Server. Starten Sie das SQL Server Management Studio und melden Sie sich an der SQL Server-Instanz mit Ihrem Windows-Benutzerkonto oder einer anderen Anmeldung mit administrativen Rechten an.

Im SQL Server Management Studio navigieren Sie im Objekt-Explorer zum Ordner **Sicherheit** und erweitern dort den Unterordner **Anmeldungen**. Wählen Sie in dessen Kontextmenü den Eintrag **Neue Anmeldung** (siehe Bild 3) und klicken Sie in dem darauffolgenden Dialog auf die Schaltfläche **Suchen**.

In dem Auswahlfenster **Benutzer oder Gruppe auswählen** legen Sie den Verweis zu dem Windows-Benutzerkonto **Bienlein** fest. Der schnellste Weg ist die direkte Eingabe des Windows-Benutzerkontos und einem anschließenden Klick auf **Namen überprüfen**. Ist die Eingabe korrekt, wird diese durch Ihren Rechnernamen ergänzt (siehe Bild 4). Mit einem Klick auf **OK** übernehmen Sie die Auswahl.

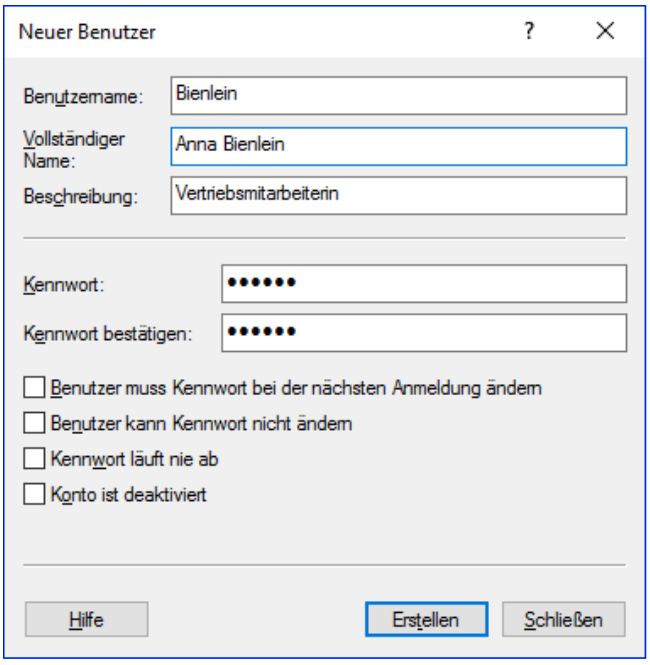

Bild 2: Windows-Benutzer Bienlein

Die Windows-Benutzerkonten Ihrer Domäne werden Sie über diesen Weg nicht finden. Hierfür müssen Sie in dem Auswahlfenster **Benutzer oder Gruppe auswählen** zunächst über die Schaltfläche **Pfade** den Suchpfad zu Ihrer Domäne festlegen. Anschließend können Sie wie oben beschrieben das Benutzerkonto aus Ihrem Active Directory auswählen.

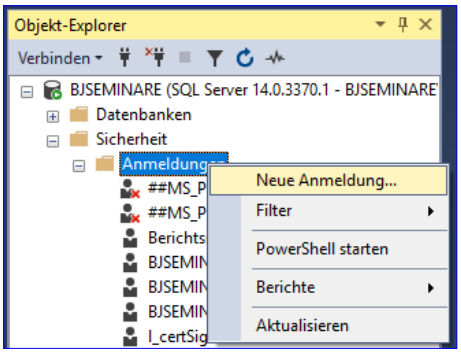

Bild 3: Neue Anmeldung im SQL Server

Sie haben es bestimmt bemerkt. Die Option **Windows-Authentifizierung** ist im Dialog standardmäßig aktiviert. Aus gutem Grund. Schließlich empfiehlt Microsoft diese Authentifizierungsmethode. Die SQL Server-Authentifizierung hingegen sollte nur in Ausnahmefällen verwendet werden. Zum Beispiel, wenn für den Betrieb der Client-Applikation keine Windows-Benutzerkonten verfügbar sind.

Die weiteren Schritte sind Ihnen von den beiden Anmeldungen **WaWiMa** und **WaWiPersonal** bekannt. Als Erstes weisen Sie der neuen Anmeldung in der Auswahlliste **Standarddatenbank** die Datenbank **WaWi_SQL** zu. Sollte mit einer Client-Applikation eine Verbindung zum SQL Server ohne die explizite Angabe einer Datenbank erfolgen, wird bei dieser Anmeldung immer die Datenbank **WaWi_SQL** verwendet.

In der Seite **Benutzerzuordnung** autorisieren Sie die Anmeldung für den Zugriff auf die Datenbanken. Hierzu wählen Sie im oberen Bereich die Datenbank **WaWi_SQL** aus. Durch diese Auswahl wird in der Datenbank ein Benutzer erstellt. Den Namen dieses Benutzers sehen Sie neben der gewählten Datenbank in der Spalte **Benutzer**.

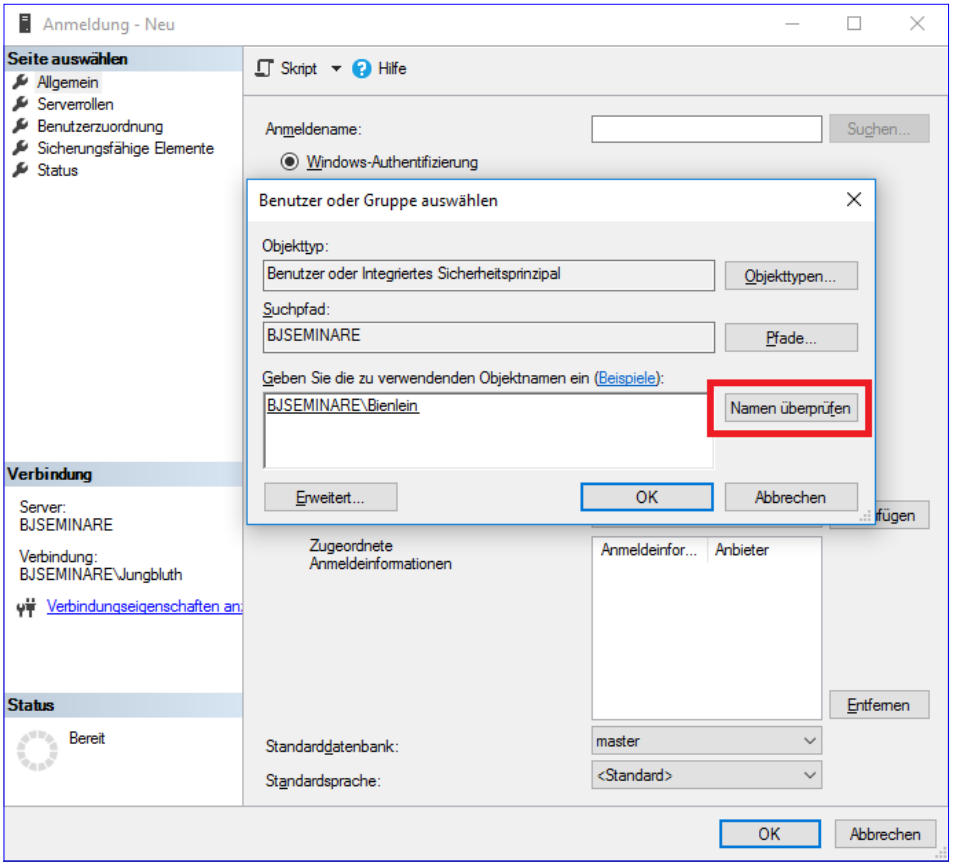

Bild 4: Anmeldung mit Windows-Authentifizierung

Zwar können Sie den Namen ändern, es ist jedoch nicht unbedingt empfehlenswert. Eines der obersten Gebote der Security ist die Einfachheit. Gestalten Sie die Vergabe und Pflege von Zugriffsberechtigungen so einfach wie möglich. Mit einem von der Anmeldung abweichenden Benutzernamen erhöhen Sie nur den Aufwand. Bei einer Änderung der Zugriffsrechte dieses Benutzers müssten Sie immer erst einmal nachschauen, zu welcher Windows-Authentifizierung er letztendlich gehört.

Ein Standardschema brauchen Sie nicht anzugeben. Ohne Angabe wird das Schema **dbo** verwendet. Die Bedeutung dieses Schemas sowie die Vorteile eigener Schemata ist das Thema einer der nächsten Beiträge.

Die Zugriffsrechte des neuen Benutzers legen Sie im unteren Bereich des Dialogs fest. Aktivieren Sie die Datenbankrollen **db_datareader**, **db_datawriter** und **edb_execute** (siehe Bild 5). Mit dieser Auswahl darf der Benutzer die Daten aller Tabellen lesen und ändern sowie alle gespeicherten Prozeduren ausführen. Die Zuordnung zur Datenbankrolle **public** können Sie nicht ändern. Jeder Benutzer ist hier standardmäßig Mitglied. Die Möglichkeiten und vor allem die Gefahren dieser Datenbankrolle lernen Sie im nächsten Teil dieser Beitragsreihe kennen.

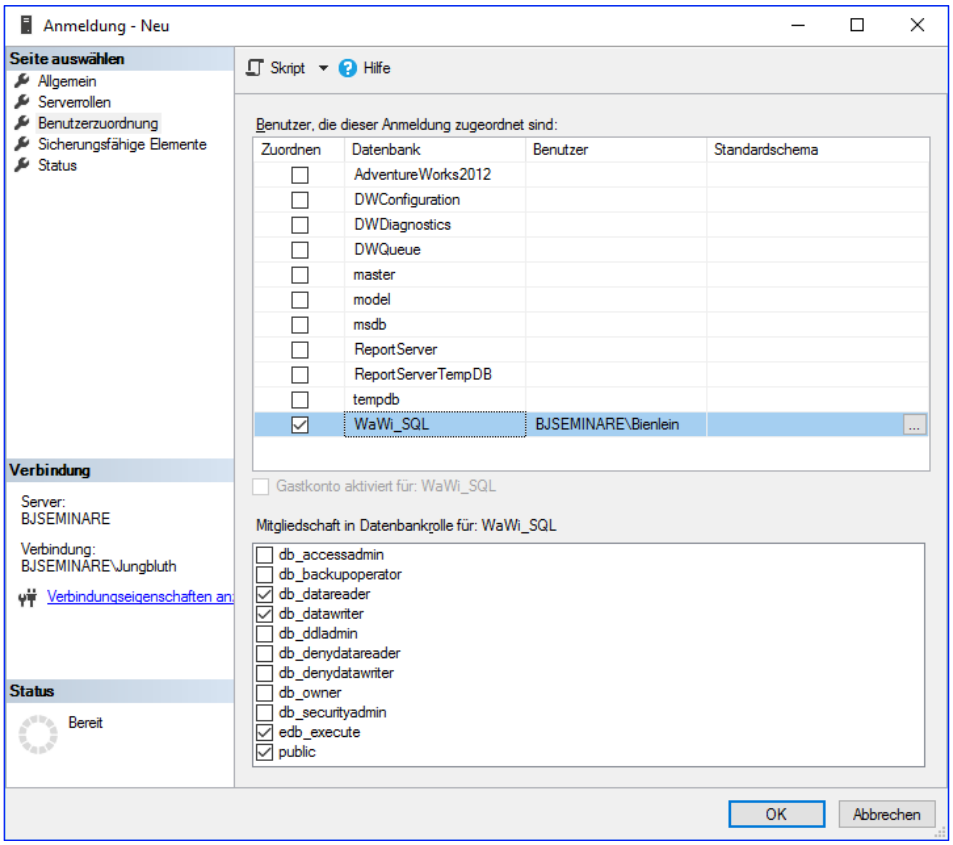

Bild 5: Benutzerzuordnung einer Anmeldung

Soweit so gut. Mit einem Klick auf **OK** legen Sie die neue Anmeldung an. Im Ordner **Anmeldungen** unter **Sicherheit** sehen Sie den neuen Eintrag **Bienlein**, wobei dem Namen **Bienlein** die Bezeichnung Ihres Rechners vorangestellt ist. Der Einfachheit halber wird im weiteren Text nur der Name des Windows-Benutzerkontos verwendet und auf den Hinweis des Rechnernamens verzichtet.

Mit der Anmeldung wurde der zugehörige Benutzer in der Datenbank **WaWi_SQL** erstellt. Wechseln Sie zur Datenbank **WaWi_SQL** und erweitern Sie dort den Ordner **Sicherheit**. Der Unterordner **Benutzer** listet nun neben den bekannten Einträgen **WaWiMa** und **WaWiPersonal** den neuen Benutzer **Bienlein** auf (siehe Bild 6).

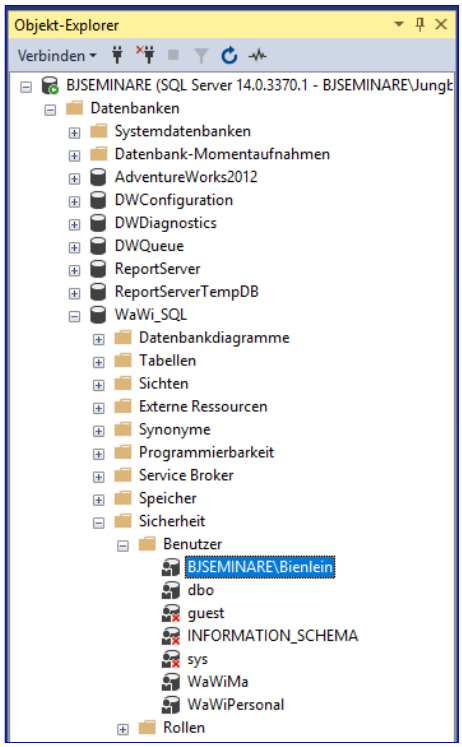

Bild 6: Datenbankbenutzer Bienlein

Frau Bienlein arbeitet im Vertrieb. Daher soll sie dieselben Rechte erhalten wie der Benutzer **WaWiMa**. Diesem wird aktuell der Zugriff auf die Tabellen und gespeicherten Prozeduren der Personalabteilung verweigert. Um diese Einschränkungen festzulegen, öffnen Sie mit einem Doppelklick auf den Eintrag **Bienlein** den Dialog **Datenbankbenut-**

zer. Auf der Seite **Sicherungsfähige Elemente** legen Sie die Rechte für einzelne Tabellen und gespeicherte Prozeduren fest. Zur Auswahl dieser Objekte gibt es mehrere Wege. Ein Weg erfolgt über das Schema. Klicken Sie auf **Suchen** und aktivieren Sie im Auswahlfenster **Objekte hinzufügen** die Option **Alle Objekte, die dem Schema angehören** (siehe Bild 7). Unter **Schemaname** wählen Sie das Schema **dbo** und bestätigen dies mit einem Klick auf **OK**.

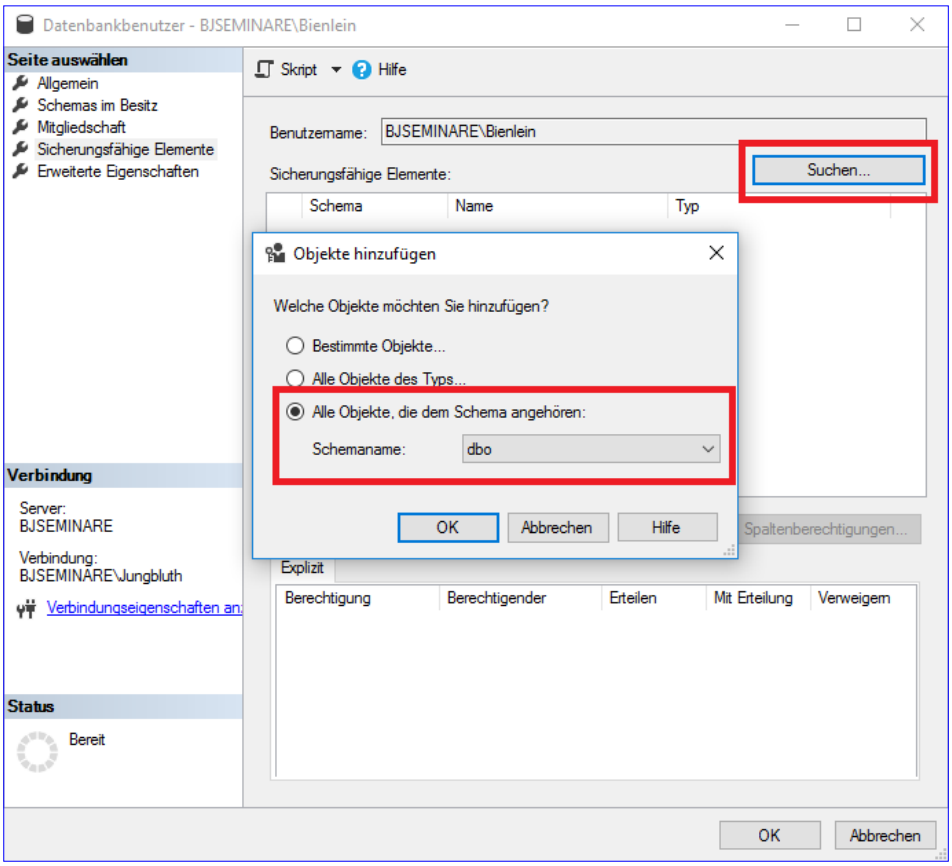

Bild 7: Auswahl der Objekte nach Schema

Der Dialog **Datenbankbenutzer** listet nun unter **Sicherungsfähige Elemente** alle Objekte der Datenbank vom Schema **dbo** auf. Dort markieren Sie die Tabelle **Bewerber** und aktivieren im unteren Bereich in der Zeile **Aktualisieren** die Option **Verweigern** (siehe Bild 8). Mit den Rechten zur gespeicherten Prozedur **pSelectGeburtstagsliste** verfahren Sie ähnlich. Sie wählen die gespeicherte Prozedur im oberen Bereich aus und aktivieren dann im unteren Bereich in der Zeile **Ausführen** die Option **Verweigern**.

Das war noch nicht alles. Sie müssen noch weitere Zugriffsrechte verweigern. Um Ihnen die umständlichen Klicks zu ersparen und gleichzeitig eine Dokumentation zu dieser Rechtevergabe zu erhalten, wählen Sie im oberen Teil des Dialogs unter **Skript** den Eintrag **Skript für Aktion in Fenster „Neue Abfrage“ schreiben**. Das SQL Server Management Studio zeigt Ihnen danach die T-SQL-Befehle zur Rechtevergabe in einer neuen Registerkarte. Genau hier werden Sie die Rechtevergabe vervollständigen. Doch vorher beenden Sie den Dialog **Datenbankbenutzer** mit einem Klick auf **Abbrechen**.

In der Registerkarte ergänzen Sie diese Zeile mit den Befehlen **SELECT**, **INSERT** und **DELETE**:

```
DENY UPDATE ON [dbo].[Bewerber] TO [Ihr Rechnername\Bienlein]
```

Danach sieht die Zeile wie folgt aus:

```
DENY SELECT, INSERT, UPDATE, DELETE ON [dbo].[Bewerber] TO [Ihr Rechnername\Bienlein]
```

Danach kopieren Sie die Zeile und fügen sie zweimal in das Skript ein. Ersetzen Sie in der ersten eingefügten Zeile den Namen der Tabelle mit **Mitarbeiter** und in der zweiten mit **Stellen**:

```
DENY SELECT, INSERT, UPDATE, DELETE ON [dbo].[Mitarbeiter] TO [Ihr Rechnername \Bienlein]
```

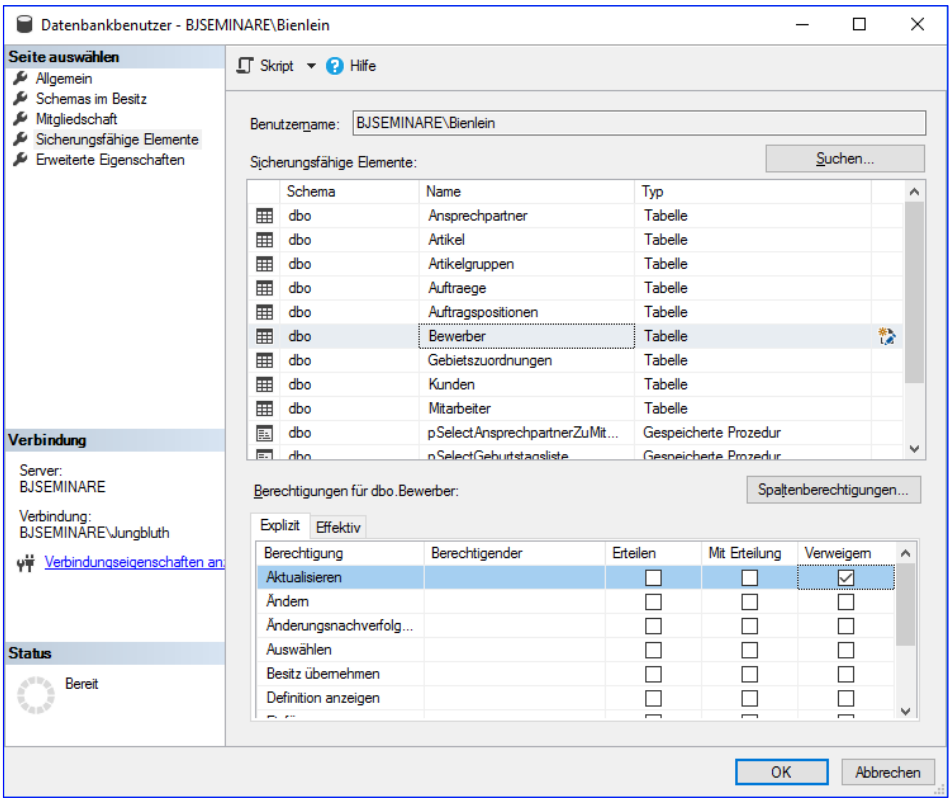

Bild 8: Rechtevergabe eines Benutzers

```
DENY SELECT, INSERT, UPDATE, DELETE ON [dbo].[Stellen] TO
[Ihr Rechnername \Bienlein]
```

Die nächsten drei Zeilen mit **GO**, **USE WaWi_SQL** und einem weiteren **GO** sind für das Skript nicht erforderlich. Entfernen Sie diese Zeilen und ergänzen Sie anschließend das Skript mit entsprechenden Hinweisen und Kommentaren. Heraus kommt ein T-SQL-Skript wie in Bild 9, das Sie in Ihrer Dokumentation zur Rechtevergabe speichern. Mit einem Klick auf die Schaltfläche **Ausführen** erweitern Sie die Zugriffsrechte des Benutzers **Bienlein**.

Wobei es sich beim Erweitern hier eher um ein Einschränken handelt. Grundsätzlich besitzt der Benutzer **Bienlein** durch die Zuordnung zu den Datenbankrollen **db_datareader**, **db_datawriter** und **edb_execute** Lese- und Schreibrechte an allen Tabellen der Datenbank sowie das Recht zum Ausführen aller gespeicherten Prozeduren. Der Zugriff auf die Tabellen **Bewerber**, **Mitarbeiter**, **Stellen** und der gespeicherten Prozedur **pSelectGeburtstagsliste** wird ihm jedoch durch das Recht **DENY** explizit verweigert – und dieses Recht steht über allen anderen.

Zur Kontrolle öffnen Sie erneut den Dialog **Datenbankbenutzer** mit einem Doppelklick auf den Eintrag **Bienlein**. Dieser zeigt Ihnen nun die erteilten Rechte in der Seite **Sicherungsfähige Elemente** (siehe Bild 10). Die zugeordneten Datenbankrollen sehen Sie in der Seite **Mitgliedschaft**.

Mit der Rechtevergabe des Benutzers ist das Erstellen der Anmeldung vollständig. Die neue Anmeldung basiert auf einer Windows-Authentifizierung, in diesem Fall auf dem Windows-Benutzerkonto **Bienlein**. Jetzt ist noch die Beispielapplikation an die neue Authentifizierungsmethode anzupassen.

```
-- Ergänzende Rechtevergabe in Datenbank WaWi_SQL
-- Erteilt am 20211008 von Datenbankadministrator

-- Verweigern des Zugriffs auf Daten der Personalverwaltung durch die Anmeldung BJSEMINARE\Bienlein

USE WaWi_SQL;
GO
-- Gespeicherte Prozeduren der Personalverwaltung
DENY EXECUTE ON dbo.pSelectGeburtstagsliste TO [BJSEMINARE\Bienlein];

-- Tabellen der Personalverwaltung
DENY SELECT, INSERT, UPDATE, DELETE ON dbo.Bewerber     TO [BJSEMINARE\Bienlein];
DENY SELECT, INSERT, UPDATE, DELETE ON dbo.Mitarbeiter  TO [BJSEMINARE\Bienlein];
DENY SELECT, INSERT, UPDATE, DELETE ON dbo.Stellen      TO [BJSEMINARE\Bienlein];
GO
```

Bild 9: Rechtevergabe per T-SQL

Access und die Windows-Authentifizierung

Die Authentifizierung am SQL Server findet in der Beispielapplikation mit der Funktion **fTabellenEinbinden** statt. Diese ermittelt über die Access-Abfrage **Anmeldedaten** die zum aktuellen Anwender passende SQL Server-Anmeldung. Für die Mitarbeiter im Vertrieb ist das die Anmeldung **WaWiMa**, für die Kollegen in der Personalabteilung die Anmeldung **WaWiPersonal**. Die Zuordnungen wie die Anmeldedaten sind in den Tabellen **Benutzer** und **Datenquellen** hinterlegt. Zum Schutz dieser Daten und insbesondere der dort gespeicherten Kennwörter sind beide Tabellen als Systemobjekte definiert und somit nicht ohne weiteres sichtbar.

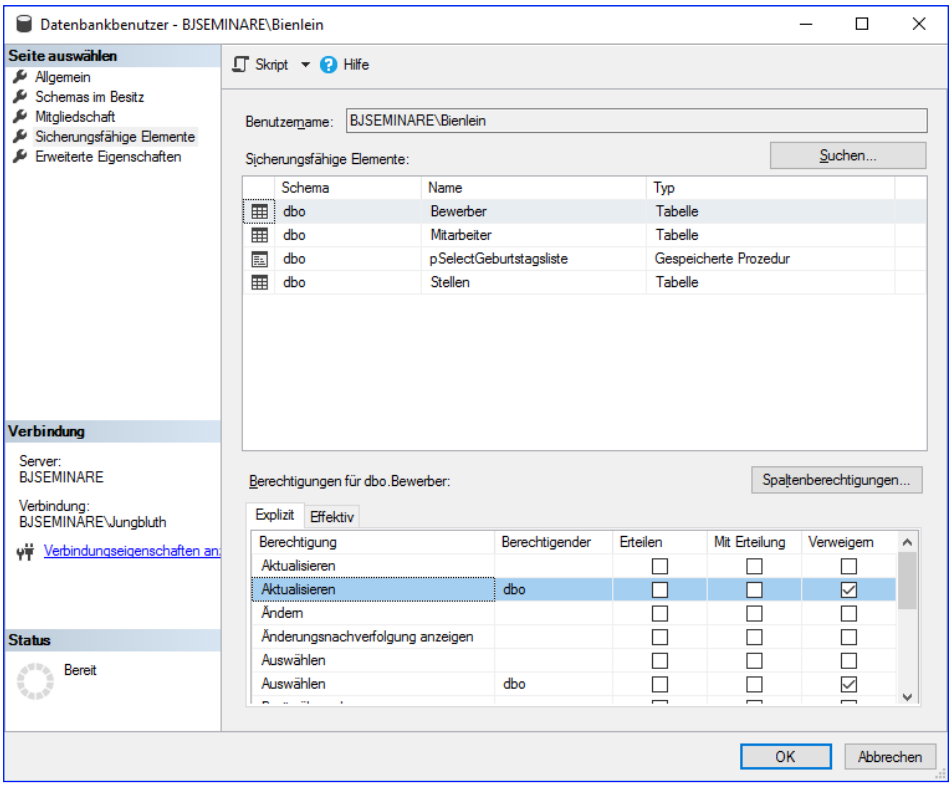

Bild 10: Verweigerter Zugriff für Bienlein

Der ganze Aufwand ist nur zum Schutz der Kennwörter zu den Anmeldungen **WaWiMa** und **WaWiPersonal** erforderlich. Diese dürfen lediglich den Entwicklern der Applikation beziehungsweise den Datenbankadministratoren bekannt sein. Den Anwendern und vor allem möglichen Angreifern soll das unerlaubte Ermitteln und Einsetzen dieser Kennwörter so schwer wie möglich gemacht werden.

Wobei letztere wohl eher mit einem Texteditor direkt in die ACCDB schauen. Was auf diesem Weg dort alles zu sehen ist, haben Sie im Teil 7 der Beitragsreihe erfahren.

Mit der Windows-Authentifizierung fällt der komplette Aufwand weg. Das Speichern von Kennwörtern ist nicht mehr erforderlich. Die Zugriffsrechte ergeben sich durch das aktuell verwendete Windows-Benutzerkonto des Anwenders. Frau Bienlein greift nun mit der zu ihrem Windows-Benutzerkonto eingerichteten Anmeldung auf die Datenbank **WaWi_SQL** zu. Damit diese Art der Anmeldung funktioniert, muss die Verbindungszeichenfolge in der Beispielapplikation angepasst werden.

Öffnen Sie die Beispielapplikation **WaWi v2** und wechseln Sie zur Funktion **fOdbcPerTabelle** im Modul **OBDC**. Hier wird die Verbindungszeichenfolge zusammengesetzt. Die nun anstehenden Änderungen werden die Funktion um einiges verkürzen. Es beginnt mit der Quelle der einzelnen Bestandteile einer Verbindungszeichenfolge, der Abfrage **Anmeldedaten**. Diese liefert die Bezeichnung des SQL Servers, den Namen der Datenbank sowie die zum aktuellen Anwender passende Anmeldung inklusive des Kennworts. Da die Anmeldedaten nicht mehr benötigt werden, ändern Sie in der folgenden Zeile die Quelle des Recordsets von Anmeldedaten in Datenquellen.

```
Set rsVerbindung = CurrentDb.OpenRecordset("Datenquellen",
dbOpenSnapshot)
```

Durch den Wegfall der Anmeldedaten sind die Variablen **strBenutzer** und **strKennwort** ebenfalls nicht mehr notwendig. Daher löschen Sie die Deklaration der Variablen sowie diese beiden Zeilen:

```
If IsNull(rsVerbindung("Benutzer")) Then boKomplett =
False
If IsNull(rsVerbindung("Kennwort")) Then boKomplett =
False
```

Aus dem gleichen Grund trennen Sie sich auch von diesen Zeilen:

```
";UID=" & rsVerbindung("Benutzer") & _
";PWD=" & fKennwortLesen(rsVerbindung("Kennwort"))
```

Jetzt ändern Sie den Eintrag **Trusted_Connection=No** in **Trusted_Connection=Yes**. Durch diesen Parameter wird bei der Verbindung zum SQL Server die aktuelle Windows-Anmeldung des Anwenders übergeben. Das Erstellen der Verbindungszeichenfolge sieht nun so aus:

```
'Datenquelle ermitteln
Set rsVerbindung = CurrentDb.OpenRecordset(_
                        "Datenquellen", dbOpenSnapshot)
If Not rsVerbindung.EOF Then
    'Inhalte prüfen
    If IsNull(rsVerbindung("Server")) Then _
                                    boKomplett = False
    If IsNull(rsVerbindung("Datenbank")) Then _
                                    boKomplett = False
    'Anmeldedaten vollständig?
    If boKomplett = True Then
        'Connection definieren
        fOdbcPerTabelle = _
            "ODBC;DRIVER=ODBC Driver 17 for SQL Server" _
            & ";Trusted_Connection=Yes" & _
            ";Server=" & rsVerbindung("Server") & _
            ";Database=" & rsVerbindung("Datenbank")
    End If
Else
    'Keine Anmeldedaten gefunden
    boKomplett = False
End If
```

Natürlich könnten der Name des SQL Servers und der Name der Datenbank direkt in der Funktion eingetragen werden. Dadurch wäre die lokale Tabelle **Datenquellen** ebenso hinfällig. Aber warum sollten Sie auf diesen Komfort verzichten wollen? Über die beiden Werte lässt sich die Access-Applikation recht einfach auf einen anderen SQL Server oder eine andere Datenbank umschalten. So sind Sie zum Beispiel in der Lage, schnell zwischen Entwicklungsumgebung und Produktivumgebung zu wechseln. Dabei kann die Datenbank zur Entwicklungsumgebung unter einem anderen Namen im gleichen SQL Server liegen oder unter dem Originalnamen in einer anderen SQL Server-Instanz.

Fehlt der Name des SQL Servers beziehungsweise der Datenbank, erhalten Sie durch die Messagebox am Ende der Funktion eine Meldung. Diese können Sie noch etwas anpassen, denn sie prüft nicht mehr die Anmeldedaten, sondern lediglich die Verbindungsdaten:

```
'Verbindungsdaten unvollständig
If boKomplett = False Then
    MsgBox "Die Verbindungsdaten sind nicht _
        vollständig erfasst." & vbCrLf & "Bitte ergänzen _
          Sie die fehlenden Informationen.", vbCritical, _
                                CurrentDb.Properties!AppTitle
End If
```

Soweit zum Datenzugriff per ODBC der für die eingebundenen Tabellen und die Pass Through-Abfragen relevant ist. Die Beispielapplikation nutzt zusätzlich die Datenzugriffsmethode ADO und diese wiederum verwendet OLE DB für den Zugriff auf die Daten. Es ist also eine weitere Anpassung erforderlich.

Dazu wechseln Sie zur Funktion **fAdoPerTabelle** im Modul **ADO**. Die nun anstehenden Änderungen sind ähnlich der

eben durchgeführten. Als erstes ändern Sie hier ebenfalls die Quelle des Recordsets in Datenquellen.

```
Set rsVerbindung = CurrentDb.OpenRecordset(_
                        "Datenquellen", dbOpenSnapshot)
```

Danach entfernen Sie die Deklarationen der Variablen **strBenutzer** und **strKennwort** sowie die folgenden Zeilen:

```
";User ID=" & rsVerbindung("Benutzer") & _
";Password=" & fKennwortLesen(rsVerbindung("Kennwort"))
```

Anschließend ergänzen Sie den Aufbau der Verbindungszeichenfolge mit dem Parameter **Integrated Security**. Mit diesem Parameter findet bei OLE DB die Übergabe der aktuellen Windows-Anmeldung statt. Der Parameter akzeptiert als Wert sowohl **SSPI** wie auch **Yes**. Sie sollten an dieser Stelle **SSPI** verwenden, denn der Wert **Yes** wird nicht von allen OLE DB-Treibern unterstützt. **SSPI** steht für **Security Support Provider Interface**. Diese Schnittstelle ermöglicht die Verwendung von Funktionen verfügbarer Sicherheitsmodelle, unter anderem das Prüfen einer Authentifizierung. Mit dem Eintrag **Integrated Security** haben die Zeilen zu den Parametern **User ID** und **Password** keine Bedeutung mehr. Weshalb Sie die beiden Zeilen nun löschen. Danach sollte der Aufbau der Verbindungszeichenfolge so aussehen:

```
fAdoPerTabelle = "Provider=MSOLEDBSQL;Integrated _
       Security=SSPI;Server=" & rsVerbindung("Server") & _
                  ";Database=" & rsVerbindung("Datenbank")
```

Zum Abschluss passen Sie noch die Meldung am Ende der Funktion an. Am besten nutzen Sie denselben Text wie in der Funktion **fOdbcPerTabelle**.

Nach diesen Änderungen wird weder der Anmeldename noch das Kennwort zum Erstellen der Verbindungszeichenfolgen verwendet. Daher werden Sie die Quelle dieser Informationen nun ebenfalls löschen. Beide sind in der Tabelle **Datenquellen** hinterlegt. Bevor Sie die Tabelle ändern können, müssen Sie ihr zunächst den Status der Systemtabelle entziehen. Führen Sie dazu im VBA-Direktbereich die folgende Anweisung aus:

```
fTabelleAlsSystemObjekt "Datenquellen", False
```

Wiederholen Sie die Anweisung für die Tabelle **Benutzer**. Hier gibt es gleich eine weitere Änderung. Danach wechseln Sie zu Access in den Navigationsbereich und öffnen die Tabelle **Datenquellen** im Entwurfsmodus. Entfernen Sie die nicht mehr benötigten Spalten **Benutzer** und **Kennwort** und speichern Sie die neue Struktur der Tabelle. Anschließend öffnen Sie die Datenblattansicht. Ohne die Information der Anmeldedaten sehen Sie jetzt zwei Datensätze mit gleichem Inhalt. Beide zeigen die Bezeichnung zu Ihrem SQL Server und zur Datenbank **WaWi_SQL**. Löschen Sie einen der Einträge. Und wenn Sie schon beim Löschen sind, entfernen Sie gleich noch die Tabelle **Benutzer** und die Access-Abfrage **Anmeldedaten**. Jegliche Information dieser Tabelle sowie der Abfrage ist für die Windows-Authentifizierung irrelevant.

Zugegeben, das waren jetzt viele Änderungen. Im Umkehrschluss zeigt dies aber sehr deutlich, von wieviel Ballast Sie sich bei einer Windows-Authentifizierung trennen. Zum Test des neuen Anmeldeverfahrens ist ein Neustart der Beispielapplikation erforderlich, denn es besteht bereits eine Verbindung zum SQL Server. Sind Sie der Installationsanleitung gefolgt, handelt es sich hierbei um die Anmeldung **WaWiPersonal** und somit um eine Anmeldung basierend auf einer SQL Server-Authentifizierung. Wie Sie im letzten Beitrag erfahren haben, werden Sie solche Verbindungen nur durch einen Neustart der Access-Applikation wieder los.

Ein Neustart öffnet das unsichtbare Formular **Verbindung**, das die Funktion **fTabellenEinbinden** und dort wiederum die eben angepasste Funktion **fOdbcPerTabelle** ausführt. Nachdem die Funktion **fTabellenEinbinden** alle Tabellen neu verknüpft und die Verbindungszeichenfolgen der Pass Through-Abfragen angepasst hat, wird das Formular

Steuerelemente per VBA erstellen

Im Beitrag »Formulare per VBA erstellen« (www.access-im-unternehmen.de/1332) haben wir gezeigt, wie Sie per VBA ein neues, leeres Formular erstellen und seine Eigenschaften einstellen. Darauf wollen wir in diesem Beitrag aufbauen und zeigen, wie Sie dem Formular per VBA die gewünschten Steuerelemente hinzufügen können. Und auch Steuerelemente haben eine Menge Eigenschaften, die wir nach dem Anlegen festlegen müssen – Position, Aussehen und auch wieder Ereigniseigenschaften. Nach der Lektüre des vorliegenden Beitrags haben Sie alle Werkzeuge, die Sie brauchen, um beispielsweise Access-Add-Ins zu nutzen, um einer Anwendung neue Formulare und Steuerelemente hinzuzufügen.

Vorbereitung: Formular anlegen

Als Vorbereitung wollen wir mit den Techniken, die wir im oben genannten Beitrag zum Erstellen von Formularen gelernt haben, zunächst ein neues Formular erzeugen. Dazu verwenden wir eine Funktion, die weitere Funktionen aus dem oben genannten Beitrag nutzt.

Die folgende Funktion erwartet den Namen des zu erstellenden Formulars und übergibt diesen an eine weitere Funktion namens **CreateNewForm**. Liefert diese den Wert **True** zurück, wurde das Formular erfolgreich erstellt.

Dann öffnen wir dieses Formular in der Formularansicht und stellen noch die Anzeige von Kopf- und Fußbereich ein – diese Bereiche werden wir weiter unten nämlich auch mit Steuerelementen versehen. Schließlich gibt die Funktion einen Verweis auf das noch in der Entwurfsansicht geöffnete **Form**-Objekt zurück:

```
Public Function GetNewForm(strForm As String) As Form
    Dim frm As Form
    If CreateNewForm(strForm) = True Then
        DoCmd.OpenForm strForm, acDesign
        Set frm = Forms(strForm)
        With frm
            RunCommand acCmdFormHdrFtr
            .Section(acHeader).Visible = True
            .Section(acFooter).Visible = True
        End With
    End If
    Set GetNewForm = frm
End Function
```

Steuerelemente anlegen per VBA

Diese Funktion rufen wir von der folgenden Prozedur aus auf und referenzieren das erstellte und geöffnete Formular mit der Variablen **frm**:

```
Public Sub SteuerelementeAnlegen()
    Dim frm As Form
    Set frm = GetNewForm("frmMitSteuerelementen")
    'Steuerelemente anlegen
End Sub
```

Danach können wir bereits loslegen, indem wir die gewünschten Anweisungen anstelle des Kommentars **'Steuerelemente anlegen** einfügen. Die minimale Version der Methode **CreateControl** hat nur zwei Parameter – den Namen des Formulars und den Typ des Steuerelements, hier **acTextBox** für ein Textfeld:

```
CreateControl frm.Name, acTextBox
```

Dieser Befehl legt bereits ein einfaches Steuerelement an, wobei die Position links oben im Detailbereich des Formulars ist.

Wichtig ist, dass Sie genau wie über die Benutzeroberfläche nur Steuerelemente anlegen können, wenn das Formular in der Entwurfsansicht geöffnet ist.

Die Methode CreateControl

Die Methode **CreateControl**, eigentlich ein Teil des **Application**-Objekts, aber auch ohne dessen Angabe aufrufbar, hat noch weitere Parameter. Nur die ersten beiden Parameter **FormName** und **ControlType** sind Pflichtparameter:

- **FormName**: Name des Formulars, in dem das Steuerelement angelegt werden soll.
- **ControlType**: Typ des Steuerelements, der durch eine Konstante angegeben wird – zum Beispiel **acTextBox**, **acLabel** oder **acCommandButton**.
- **Section**: Bereich, in dem das Steuerelement angelegt werden soll. Sinnvolle Werte für das Anlegen von Steuerelementen in Formularen sind **acDetail**, **acHeader** und **acFooter**. Die übrigen Möglichkeiten, die per IntelliSense angezeigt werden, sind für Berichte gedacht.
- **Parent**: Verweis auf das übergeordnete Steuerelement. Hiermit können Sie beispielsweise festlegen, zu welchem Textfeld ein Bezeichnungsfeld gehört.
- **ColumnName**: Angabe eines Feldes der Datensatzquelle des Formulars, an welches das Steuerelement gebunden werden soll. Füllt die Eigenschaft **Steuerelementinhalt** und mehr – siehe weiter unten.
- **Left**: Position vom linken Rand des Formulars
- **Top**: Position vom oberen Rand des Formulars
- **Width**: Breite des Steuerelements
- **Height**: Höhe des Steuerelements

Textfeld mit Bezeichnungsfeld anlegen

Um ein Textfeld mit einem Bezeichnungsfeld zu erstellen, benötigen Sie zwei Aufrufe der **CreateControl**-Methode. Mit dem ersten erstellen Sie das Textfeld, mit dem zweiten das Bezeichnungsfeld.

Für das Bezeichnungsfeld stellen wir den Parameter **Parent** auf das Textfeld ein. Auf diese Weise werden die beiden Steuerelemente so verknüpft, wie es auch beim Anlegen von Textfeldern über den Entwurf geschieht oder wenn Sie Felder aus der Feldliste in den Entwurf ziehen.

Für ein einfaches Textfeld mit einem Bezeichnungsfeld benötigen Sie die folgenden Anweisungen. Die Variablen **txt** und **lbl** dienen zum Referenzieren der neu erstellten Steuerelemente. Das Textfeld erstellen wir im Formular mit dem Namen, den wir mit **frm.Name** für das mit **frm** wie oben referenzierte Formular abfragen, als **acTextBox** im Bereich **acDetail**. Außerdem legen wir die Position mit 1400 Twips vom linken und 100 Twips vom oberen Rand und die Größe mit einer Breite von 2000 Twips und einer Höhe von 300 Twips fest:

```
Dim txt As TextBox
Dim lbl As Label
Set txt = CreateControl(frm.Name, acTextBox, _
                        acDetail, , , 1400, 100, 2000, 300)
```

Danach erstellen wir das Bezeichnungsfeld ebenfalls in **frm.Name** als **acLabel** im Bereich **acDetail**. Es soll dem zuvor erstellten Textfeld untergeordnet werden, also geben wir für den Parameter **Parent** den Namen des Textfeldes an (mit **txt.Name**). **txt.Name** funktioniert immer, auch wenn wir den Namen des Textfeldes nicht explizit festgelegt haben und liefert für das erste Textfeld im Formular beispielsweise den Wert **Text0**.

Die Position und die Abmessungen definieren wir so, dass das Bezeichnungsfeld links vom Textfeld erscheint. Anschließend legen wir noch die Beschriftung fest, indem wir die Eigenschaft **Caption** für das zuvor definierte und mit

der Variablen **lbl** referenzierte Bezeichnungsfeld auf den Wert **Textfeld:** festlegen:

```
Set lbl = CreateControl(frm.Name, acLabel, _
                acDetail, txt.Name, , 100, 100, 1200, 300)
lbl.Caption = "Textfeld:"
```

Indem Sie das Bezeichnungsfeld mit dem **Parent**-Parameter an das Textfeld binden, sorgen Sie gleichzeitig dafür, dass die Eigenschaft **Bezeichnungsname** des Textfeldes auf den Namen des Bezeichnungsfeldes eingestellt wird.

Das Ergebnis finden Sie in Bild 1 (siehe Prozedur **FormularMitTextfeldUndBezeichnungsfeld**).

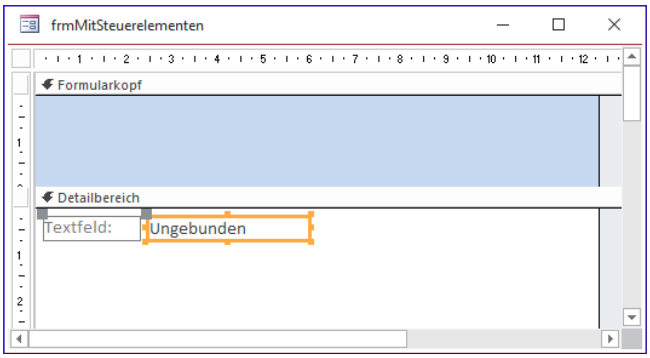

Bild 1: Textfeld mit Bezeichnungsfeld

Gebundene Textfelder anlegen

Nun gehen wir einen Schritt weiter und wollen zwei gebundenen Textfelder anlegen. Dazu stellen wir als Erstes die Eigenschaft **Recordsource** (**Datensatzquelle**) des Formulars auf die Tabelle **tblArtikel** ein:

```
frm.RecordSource = "tblArtikel"
```

Danach erstellen wir nacheinander zwei Textfelder mit Bezeichnungsfeldern, wobei wir im Gegensatz zum vorherigen Beispiel nun den Parameter **ColumnName** nutzen, um das Feld anzugeben, an welches das jeweilige Textfeld gebunden werden soll:

```
Dim txt As TextBox
Dim lbl As Label
Set txt = CreateControl(frm.Name, acTextBox, _
            acDetail, , "ArtikelID", 1500, 100, 2000, 300)
Set lbl = CreateControl(frm.Name, acLabel, _
                acDetail, txt.Name, , 100, 100, 1300, 300)
lbl.Caption = "ArtikelID:"
Set txt = CreateControl(frm.Name, acTextBox, _
          acDetail, , "Artikelname", 1500, 500, 2000, 300)
Set lbl = CreateControl(frm.Name, acLabel, _
                acDetail, txt.Name, , 100, 500, 1300, 300)
lbl.Caption = "Artikelname:"
```

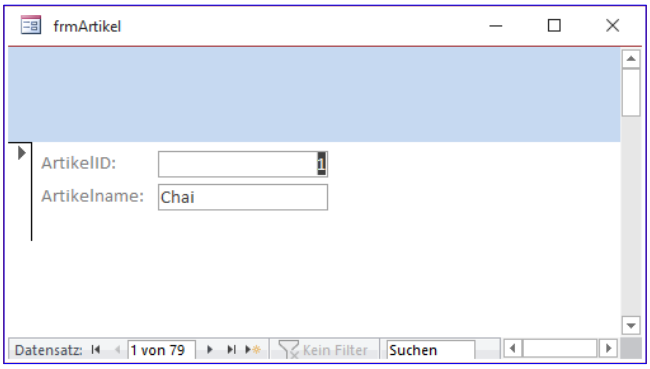

Bild 2: Formular mit gebundenen Textfeldern

Das so erstellte Formular sieht in der Formularansicht wie in Bild 2 aus (siehe Prozedur **GebundeneTextfelder**).

Gebundene Kombinationsfelder anlegen

Wenn wir die Tabelle **tblArtikel** betrachten, finden wir als Nächstes zwei Nachschlagefelder.

Diese wollen wir nun in Form von Kombinationsfeldern zum Formular hinzufügen. Für das erste Nachschlagefeld namens **KategorieID** sieht der dazu nötige Code wie folgt aus:

```
Dim cbo As ComboBox
...
Set cbo = CreateControl(frm.Name, acComboBox, _
          acDetail, , "KategorieID", 1500, 900, 2000, 300)
Set lbl = CreateControl(frm.Name, acLabel, _
                acDetail, cbo.Name, , 100, 900, 1300, 300)
lbl.Caption = "KategorieID:"
```

Wie wir in Bild 3 sehen, brauchen wir nur das zugrunde liegende Nachschlagefeld der Tabelle mit dem Parameter **ColumnName** anzugeben, damit ein neues Kombinationsfeld mit allen in der Tabelle für dieses Feld festgelegten Eigenschaften erstellt wird (siehe Prozedur **GebundeneKombinationsfelder**).

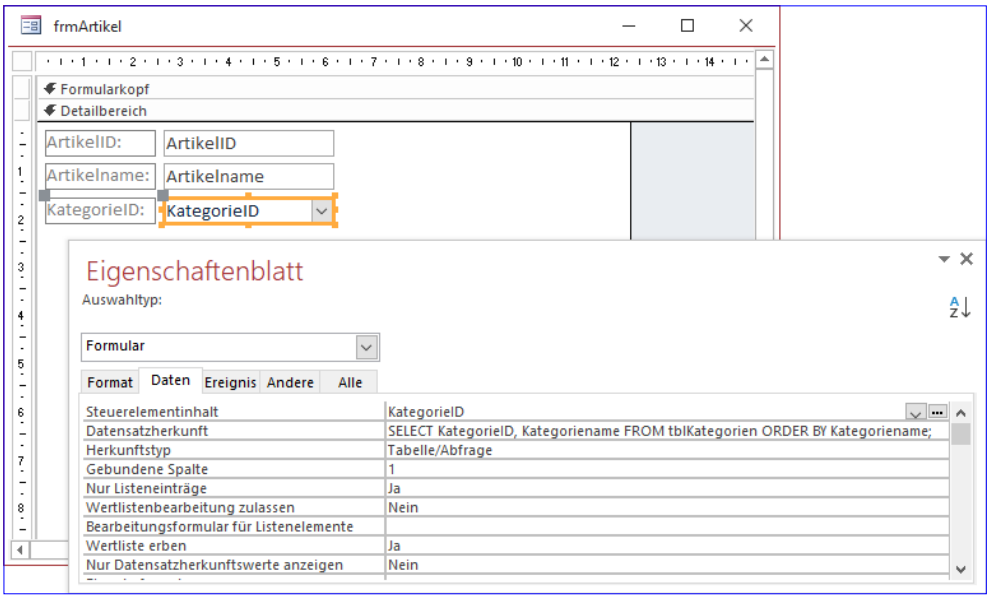

Bild 3: Formular mit gebundenem Kombinationsfeld

Übernahme von Eigenschaften bei gebundenen Steuerelementen

Wenn Sie mit dem Parameter **ColumnName** ein Feld der zugrunde liegenden Datensatzquelle angeben, stellt dies nicht nur die Eigenschaft **Steuerelementinhalt** auf den Namen des zu bindenden Feldes ein. Dazu gehören Eigenschaften wie **Textformat**, **Eingabeformat**, **Standardwert**, **Gültigkeitsregel** oder **Gültigkeitsmeldung**. Bei Kombinationsfeldern auf Basis von Nachschlagefeldern werden, wir oben gezeigt, sogar die Eigenschaften des Nachschlagefeldes übernommen.

Detailformular erstellen

Ein oft erledigter Schritt ist das Erstellen eines Detailformulars, also ein Formular, das die Daten einer Tabelle oder Abfrage anzeigen soll. Dazu sind normalerweise einige Schritte nötig wie das Erstellen des Formulars, das Hinzufügen der Datensatzquelle, das Hineinziehen der Felder in den Formularentwurf und gegebenenfalls noch das Anordnen der Felder, weil Access beim Hineinziehen von Feldern aus der Feldliste beispielsweise Kontrollkästchen immer links vom Bezeichnungsfeld anordnet und die übrigen Steuerelemente immer rechts vom Bezeichnungsfeld.

Deshalb zeigen wir im Folgenden eine praktische Anwendung der Techniken, die Sie weiter oben gelernt haben.

Die Prozedur **DetailformularErstellen** erwartet drei Parameter:

- **strForm**: Name des zu erstellenden Formulars
- **strRecordsource**: Name der Datensatzquelle
- **lngCaptionWidth**: Breite der Bezeichnungsfelder mit einem Standardwert von 1500 Twips

Die Prozedur, deren Code Sie in Listing 1 finden, schließt zunächst eine gegebenenfalls noch geöffnete Instanz des Formulars mit dem in **strForm** übergebenen Namen. Dann erstellt sie dieses mit der Funktion **GetNewForm** unter dem angegebenen Namen neu und referenziert das neu erstellte und in der Entwurfsansicht geöffnete Formular mit der Variablen **frm**.

Die einzige Formulareigenschaft, welche die Prozedur einstellt, ist **RecordSource**. Sie erhält den Wert aus **strRecordsource**.

Danach referenziert die Prozedur mit **db** das aktuelle **Database**-Objekt und mit **rst** ein Recordset auf Basis der mit

```
Public Sub DetailformularErstellen(strForm As String, strRecordsource As String, Optional lngCaptionWidth As Long = 1500)
    Dim frm As Form
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Dim fld As DAO.Field
    Dim strDisplayControl As String
    Dim strCaption As String
    Dim txt As TextBox
    Dim lbl As Label
    Dim cbo As ComboBox
    Dim chk As CheckBox
    Dim lngTop As Long
    DoCmd.Close acForm, strForm
    Set frm = GetNewForm(strForm)
    frm.RecordSource = strRecordsource
    Set db = CurrentDb
    Set rst = db.OpenRecordset(strRecordsource)
    For Each fld In rst.Fields
        strDisplayControl = 0
        strCaption = ""
        On Error Resume Next
        intDisplayControl = fld.Properties("DisplayControl")
        strCaption = fld.Properties("Caption")
        On Error GoTo 0
        If Len(strCaption) = 0 Then
            strCaption = fld.Name
        End If
        Select Case intDisplayControl
            Case acCheckBox
                Set chk = CreateControl(frm.Name, acCheckBox, acDetail, , fld.Name, lngCaptionWidth + 200, lngTop + 100)
                chk.Name = "chk" & fld.Name
                Set lbl = CreateControl(frm.Name, acLabel, acDetail, chk.Name, , 100, lngTop + 100, lngCaptionWidth, 300)
            Case acComboBox
                Set cbo = CreateControl(frm.Name, acComboBox, acDetail, , fld.Name, lngCaptionWidth + 200, _
                    lngTop + 100, 2000, 300)
                cbo.Name = "cbo" & fld.Name
                Set lbl = CreateControl(frm.Name, acLabel, acDetail, cbo.Name, , 100, lngTop + 100, lngCaptionWidth, 300)
            Case Else
                Set txt = CreateControl(frm.Name, acTextBox, acDetail, , fld.Name, lngCaptionWidth + 200, _
                    lngTop + 100, 2000, 300)
                txt.Name = "txt" & fld.Name
                Set lbl = CreateControl(frm.Name, acLabel, acDetail, txt.Name, , 100, lngTop + 100, lngCaptionWidth, 300)
        End Select
        lbl.Caption = strCaption
        lngTop = lngTop + 400
    Next fld
End Sub
```

Listing 1: Prozedur zum Erstellen eines Detailformulars

strRecordsource übergebenen Datensatzquelle, bei der es sich um eine Tabelle, Abfrage oder auch einen SQL-Ausdruck handeln kann. Anschließend durchläuft sie in einer **For Each**-Schleife alle Felder der **Fields**-Auflistung des Recordsets.

In dieser Schleife stellt die Prozedur zunächst die beiden Variablen **intDisplayControl** und **strCaption** auf die Werte **0** beziehungsweise eine leere Zeichenkette ein. **intDisplayControl** soll später den Steuerelementtyp aufnehmen, der für die Anzeige des Feldes in der Datenblattansicht definiert wurde. Dieser Steuerelementtyp wird beim Entwerfen einer Tabelle automatisch von Access vorgegeben – bei Feldern des Datentyps **Kurzer Text** zum Beispiel Textfeld, bei Nachschlagefeldern Kombinationsfeld oder bei **Ja/Nein**-Feldern Kontrollkästchen.

Bei reinen Zahlenfeldern wird kein Steuerelement voreingestellt. Bild 4 zeigt, wo Sie die Eigenschaft **DisplayControl** beziehungsweise **Steuerelement anzeigen** im Entwurf der Tabelle einsehen können. **DisplayControl** liefert Werte einer Enumeration wie **acTextBox**, **acComboBox** oder **acCheckBox**.

Die Variable **strCaption** soll den Wert der Eigenschaft **Beschriftung** aufnehmen oder, wenn keine Beschriftung angelegt wurde, den Feldnamen. Die Eigenschaft **Beschriftung** heißt unter VBA **Caption**. Sie können diese auf der Registerseite **Allgemein** des oben referenzierten Screenshots festlegen, diese Beschriftung wird dann als Spaltenüberschrift oder auch als Label-Beschriftung beim Ziehen von Feldern aus der Feldliste verwendet.

Warum setzen wir die beiden Variablen **intDisplayControl** und **strCaption** immer wieder auf **0** und **""**? Weil die Properties **DisplayControl** und **Caption**, mit denen wir diese füllen wollen, nicht immer vorhanden sind, und wir versuchen, diese unter Deaktivierung der Fehlerbehandlung aus den Properties abzufragen. Da wir dies innerhalb einer Schleife machen, würden die Variablen, wenn sie nicht aus den Properties gefüllt werden können, gegebenenfalls

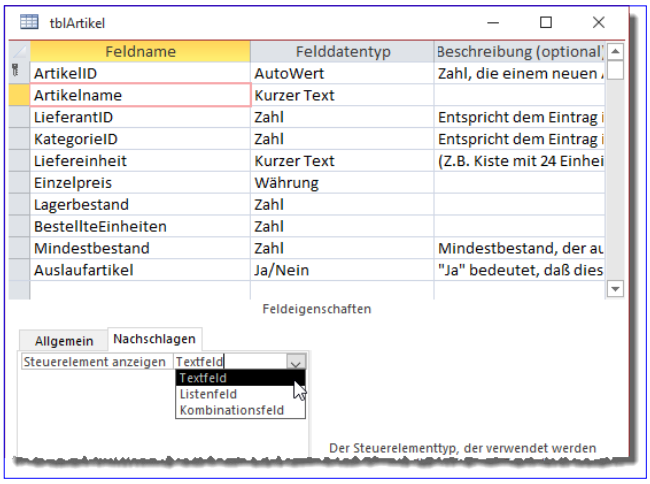

Bild 4: Wert der Eigenschaft **DisplayControl**

noch den Wert aus dem vorherigen Durchgang enthalten, wenn wir sie nicht leeren.

Die Werte der Eigenschaften **DisplayControl** und **Caption** fragen wir also bei deaktivierter Fehlerbehandlung über die Auflistung **Properties** ab und speichern diese in **intDisplayControl** und **strCaption**. War die Property **Caption** nicht vorhanden, ist **strCaption** danach leer und wir füllen es ersatzweise mit dem Namen des Feldes.

intDisplayControl nimmt einen der Werte **acCheckbox**, **acComboBox**, **acTextBox** oder **0** an. Diesen prüfen wir in der folgenden **Select Case**-Bedingung. Im Falle von **acCheckBox** erstellt die Prozedur mit der **CreateControl**-Methode ein neues Steuerelement des Typs **acCheckbox** im Detailbereich des Formulars. Das Feld wird an den mit **fld.Name** gelieferten Feldnamen gebunden.

Die Breite ermitteln wir aus dem Parameter **lngCaptionWidth** plus **200**, damit wir 200 Twips Abstand zwischen Label und Steuerelement erhalten. Den Abstand von oben stellen wir über den Wert der Variablen **lngTop** plus **100** ein. **lngTop** erhöhen wir nach jedem Steuerelement um **400**, damit das nächste Steuerelement um 400 Twips weiter unten angeordnet wird. Im Falle von **acCheckBox** brauchen wir Höhe und Breite nicht anzugeben, da die **CheckBox** immer gleich groß ist.

Anschließend weisen wir dem mit **chk** referenzierten neuen **CheckBox**-Steuerelement über die Eigenschaft **Name** den Namen des Feldes plus Präfix **chk** zu. Das folgende **Label**-Steuerelement legen wir links vom **CheckBox**-Steuerelement an. Weiter unten, außerhalb der **Select Case**-Bedingung, stellen wir noch seine Eigenschaft **Caption** auf den Wert aus **strCaption** ein.

Auf ähnliche Weise verfahren wir mit den Feldern, die für die Eigenschaft **DisplayControl** die Werte **acComboBox** oder **acTextBox** hinterlegt haben. Der Unterschied ist, dass wir diese Variablen des jeweiligen Typs zuweisen (**ComboBox** und **TextBox**) und dass wir für diese noch die Breite und die Höhe auf die Werte **2000** und **300** einstellen.

Schließlich weisen wir auch diesen Steuerelementen als Name ein Präfix wie **cbo** oder **txt** plus dem Feldnamen zu und erstellen die entsprechenden **Label**-Steuerelemente links neben dem Kombinations- oder Textfeld.

Die letzte Bedingung prüft übrigens nicht auf den Wert **acTextBox**, sondern verarbeitet alle übrigen Werte. Der Hintergrund ist, dass wir es ja nicht unbedingt mit **acTextBox** zu tun haben, sondern dass bei dem Steuerelement die Eigenschaft **DisplayControl** nicht definiert ist – dann soll dieses auch als **TextBox**-Steuerelement realisiert werden.

Nach dem Verlassen der **Select Case**-Bedingung stellen wir noch die Eigenschaft **Caption** des **Label**-Steuerelements aus **lbl** ein und erhöhen den Wert von **lngTop** um **400**, damit das folgende Steuerelement unter dem aktuellen Steuerelement angelegt wird.

Den Aufruf gestalten wir beispielsweise wie folgt:

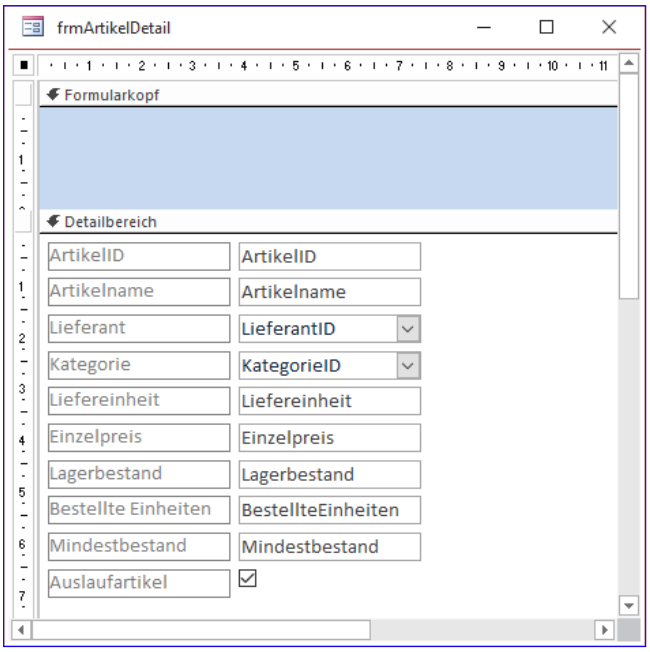

Bild 5: Entwurf des per Code erstellten Formulars zur Anzeige von Artikeln in der Detailansicht

```
DetailformularErstellen "frmArtikelDetail", _
                        "tblArtikel", 2000
```

Das Ergebnis finden Sie in Bild 5.

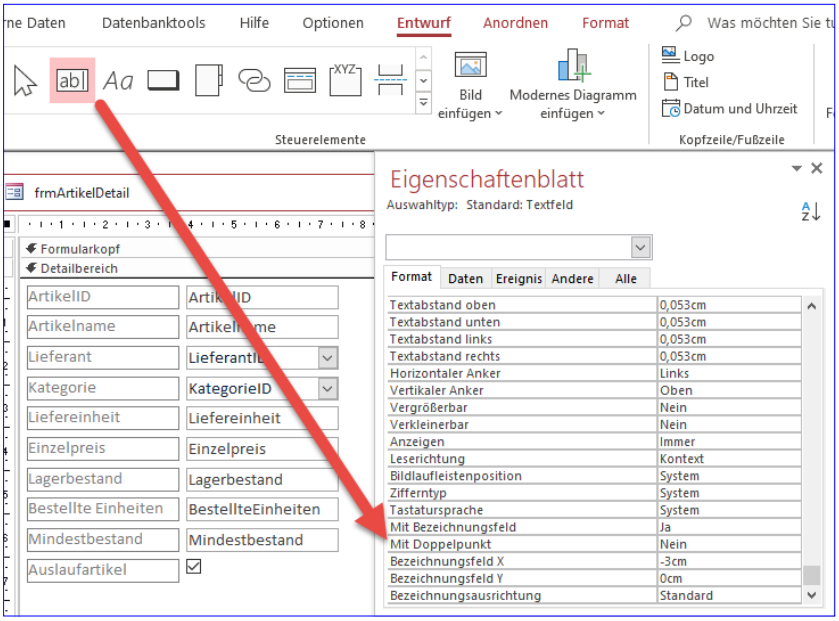

Bild 6: Einstellen der Eigenschaften für das Standardsteuerelement, hier für ein Textfeld.

Doppelpunkte sicherstellen

Hier sehen wir noch einen kleinen Makel, denn die Beschriftungen werden nicht mit einem Doppelpunkt abgeschlossen. Wenn Sie das wünschen, können Sie es auf zwei Arten erledigen:

- Ergänzen der **Label**-Beschriftung um den Doppelpunkt
- Einstellen beziehungsweise sicherstellen, dass der Doppelpunkt automatisch hinzugefügt wird

Die letztere Variante wäre sinnvoller, denn es kann auch sein, dass bei Ihnen die Doppelpunkte automatisch hinzugefügt werden. Dies hängt von der Einstellung der Eigenschaft **Mit Doppelpunkt** der jeweiligen Standardsteuerelemente ab. Diese Eigenschaft sehen Sie bei bereits hinzugefügten Steuerelementen nicht, sondern nur, wenn Sie im Ribbon auf das jeweilige Steuerelement klicken – beispielsweise auf das **Textfeld**-Steuerelement –, ohne das Steuerelement tatsächlich zum Entwurf des Formulars hinzuzufügen und dann im Eigenschaftenblatt auf der Registerseite **Format** unten die Eigenschaft **Mit Doppelpunkt** betrachten (siehe Bild 6).

Aber selbst wenn wir diese Eigenschaft wie folgt per VBA zu Beginn der Prozedur beispielsweise für die Textfelder dieses Formulars einstellen und den Formularentwurf anschließend speichern, führt dies nicht dazu, dass die Label anschließend automatisch mit abschließenden Doppelpunkten versehen werden. Das Standardsteuerelement für die Textbox erhalten Sie beispielsweise mit **frm.DefaultControl(acTextBox)** und weisen wie folgt den Wert **True** der Eigenschaft **AddColon** beziehungsweise **Mit Doppelpunkt** hinzu:

```
Set txt = frm.DefaultControl(acTextBox)
txt.AddColon = True
DoCmd.Save acForm, frm.Name
```

Es scheint also unabhängig davon zu sein, ob die Eigenschaft **AddColon** aktiviert ist oder nicht – wenn wir die **Label**-Steuerelemente separat hinzufügen, was per VBA nicht anders möglich ist, können wir die Doppelpunkte per VBA hinzufügen. Offensichtlich werden Doppelpunkte auch bei aktivierter Eigenschaft **Mit Doppelpunkt** für das Standardsteuerelement nur angelegt, wenn Sie das Textfeld über die Benutzeroberfläche hinzufügen oder durch das Ziehen von Feldern aus der Feldliste.

Also passen wir die Prozedur schlicht so an, dass die Doppelpunkte immer hinzugefügt werden:

```
lbl.Caption = strCaption & ":"
```

Danach erhalten wir die Beschriftungen mit Doppelpunkten.

Formular in der Datenblattansicht erstellen

Auf ähnliche Weise können Sie ein Formular erstellen, das standardmäßig in der Datenblattansicht angezeigt wird. Dazu müssen Sie zu der zuvor beschriebenen Prozedur lediglich die Zuweisung einer Eigenschaft hinzufügen, mit der wir die Standardansicht auf **Datenblatt** einstellen:

```
frm.DefaultView = acDefViewDatasheet
```

Sie erhalten dann genau das gleiche Ergebnis, allerdings in der Datenblattansicht. Die einzigen Elemente, die wir wegnehmen könnten, sind die Angaben der Position der Steuerelemente – alle anderen werden auch im Formular in der Datenblattansicht benötigt.

Allerdings wollen Sie vielleicht später manuelle Änderungen am Entwurf vornehmen, was schwierig wird, wenn alle Elemente übereinander angelegt wurden.

Die Prozedur für diesen Zweck heißt **Datenblattformular-Erstellen** und kann beispielsweise wie folgt aufgerufen werden:

```
DatenblattformularErstellen "sfmArtikelUebersicht", _
                            "tblArtikel", 2000
```

Dateien und Verzeichnisse auswählen mit FileDialog

Bereits seit einiger Zeit bietet die Office-Bibliothek das FileDialog-Objekt an. In früheren Versionen gab es dort einige Einschränkungen, weshalb Programmierer gern auf Alternativen zurückgegriffen haben wie etwa die entsprechenden Funktionen der Windows-API oder auch die der nicht dokumentierten WizHook-Klasse. Irgendwann hat Microsoft jedoch auch für Access alle Funktionen der FileDialog-Klasse freigeschaltet, unter anderem auch das Auswählen von zu speichernden Dateien. Daher schauen wir uns in diesem Beitrag einmal an, welche Möglichkeiten die FileDialog-Klasse nun bietet und ob wir diese für unsere Zwecke nutzen können.

Die FileDialog-Klasse verfügbar machen

Um auf die **FileDialog**-Klasse und ihre Methoden und Eigenschaften zugreifen zu können, benötigen Sie entweder einen Verweis auf die Bibliothek **Microsoft Office x.0 Object Library** oder Sie referenzieren diese per Late Binding. Da wir die Vorzüge von IntelliSense zu schätzen wissen, nutzen wir hier die Bibliothek, die Sie im VBA-Editor einbinden können. Dazu wählen Sie den Menüeintrag **Extras|Verweise** aus und selektieren im nun erscheinenden Dialog den entsprechenden Eintrag (siehe Bild 1).

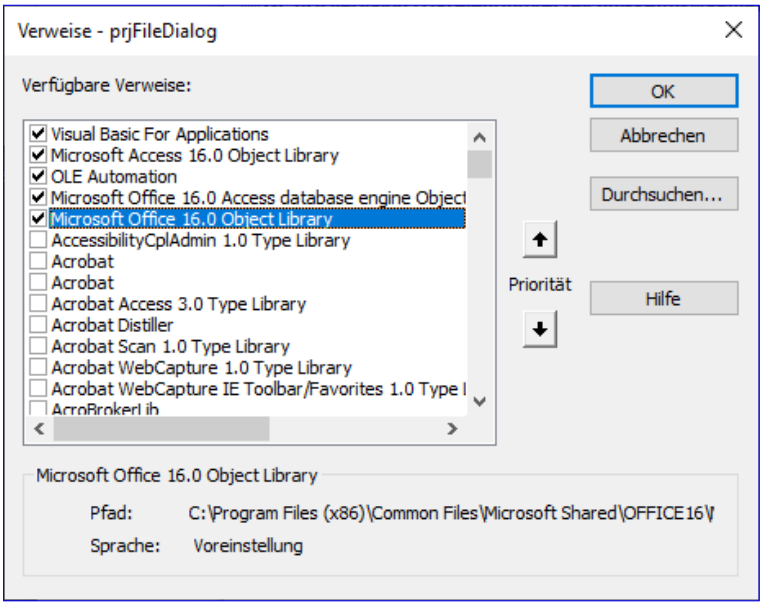

Bild 1: Verweis auf die Office-Bibliothek

Einen FileDialog anzeigen

Grundsätzlich zeigen Sie einen **FileDialog** wie folgt an:

```
Dim objFiledialog As Office.FileDialog
Set objFiledialog = FileDialog(msoFileDialogFilePicker)
objFiledialog.Show
```

Sie benötigen also eine Objektvariable, die den Typ **Office.FileDialog** aufweist und füllen diese mit einem Verweis auf die **FileDialog**-Klasse unter Angabe des Typs, den Sie wünschen – in diesem Fall **msoFileDialogPicker**.

Dann zeigen Sie den Dialog mit der Methode **Show** an. Das Ergebnis dieses einfachen Aufrufs finden Sie in Bild 2.

Auswahl im FileDialog auswerten

Nun können Sie nach der Auswahl wie im oben angegebenen Beispiel mit dem Ergebnis noch nichts anfangen. Das erledigen wir, indem wir das Ergebnis des Aufrufs prüfen und dann die selektierten Daten auswerten.

Die ersten beiden Zeilen bleiben gleich:

```
Dim objFiledialog As Office.FileDialog
Set objFiledialog = FileDialog(msoFileDialogFilePicker)
```

Danach rufen wir die **Show**-Methode jedoch als Teil einer **If...Then**-Bedingung auf und prüfen in dieser das Ergebnis.

Dieses kann **True** oder **False** lauten. Es lautet **True**, wenn der Benutzer die Eingabe mit der **Öffnen**-Schaltfläche abgeschlossen hat, und **False**, wenn er die **Abbrechen**-Schaltfläche wählt.

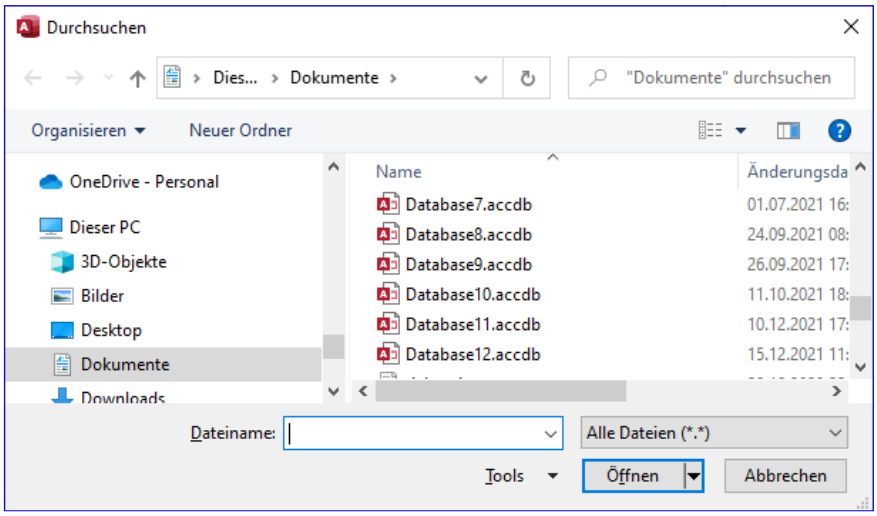

Bild 2: Ein einfacher **FileDialog**

Im ersten Fall greifen wir mit **objFileDialog.SelectedItems(1)** auf den ersten ausgewählten Eintrag zu und geben diesen im Direktbereich des VBA-Editors aus, im zweiten zeigen wir den Text **Keine Datei ausgewählt** an:

```
If objFiledialog.Show = True Then
    Debug.Print objFiledialog.SelectedItems(1)
Else
    Debug.Print "Keine Datei ausgewählt"
End If
```

Verschiedene Dialogtypen

Sie können verschiedene Dialogtypen nutzen. Dazu geben Sie unterschiedliche Werte beim Zuweisen des **FileDialog**-Objekts an.

Diese lauten:

- **msoFileDialogFilePicker**: Zeigt einen Dialog zum Auswählen einer oder mehrerer Dateien an.
- **msoFileDialogFolderPicker**: Zeigt einen Dialog zum Auswählen eines Verzeichnisses an (siehe Bild 3).
- **msoFileDialogOpen**: Zeigt den gleichen Dialog an wie **msoFileDialogFilePicker**, jedoch mit dem Titel **Öffnen**. Ist für andere Office-Anwendungen wie Word oder Excel geeignet, wo die Dokumente so direkt geöffnet werden können.
- **msoFileDialogSaveAs**: Zeigt einen Dialog zum Angeben einer zu speichernden Datei an. Es kann eine vorhandene Datei ausgewählt werden oder auch ein neuer Dateiname angegeben werden (siehe Bild 4). Die Schaltfläche zum Bestätigen der Eingabe ist hier mit **Speichern** beschriftet.

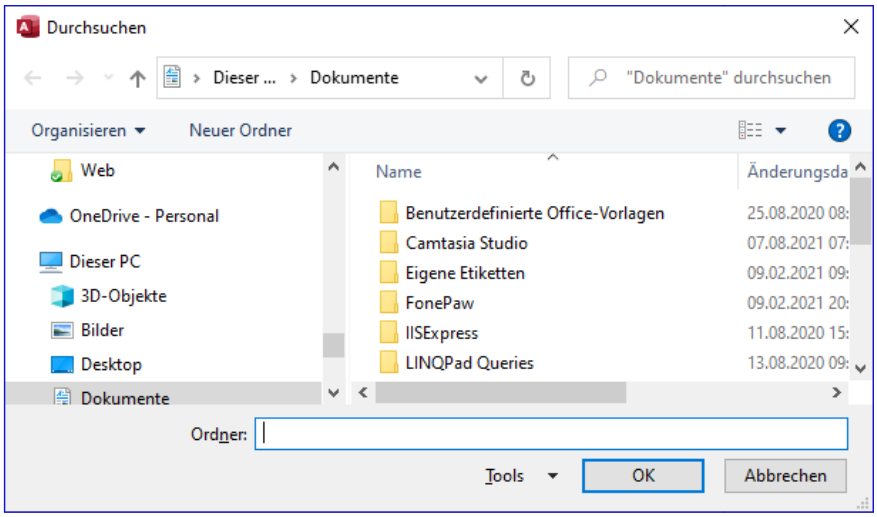

Bild 3: Der **FileDialog** mit **msoFileDialogFolderPicker** zeigt nur Ordner an, keine Dateien.

Eigenschaften werden beibehalten

Wenn Sie die nachfolgend beschrieben Eigenschaften nutzen, werden die Einstellungen beim erneuten Öffnen eines **FileDialog**-Fensters aus der gleichen Access-Session heraus beibehalten, wenn Sie nicht explizit neue Werte für die Eigenschaften angeben.

Mehrfachauswahl erlauben

Mit der Eigenschaft **AllowMultiSelect** können Sie festlegen, ob der Benutzer nur einen Eintrag (**False**) oder mehrere Einträge auswählen kann (**True**).

Diese Eigenschaft stellen Sie nach dem Zuweisen der **FileDialog**-Klasse an die Variable **objFileDialog** ein:

```
objFiledialog.AllowMultiSelect = True
```

Wenn Sie so einen **msoFileDialogFilePicker**-Dialog öffnen, können Sie beispielsweise bei gedrückter **Strg**-Taste mehrere Dateien wie in Bild 5 auswählen.

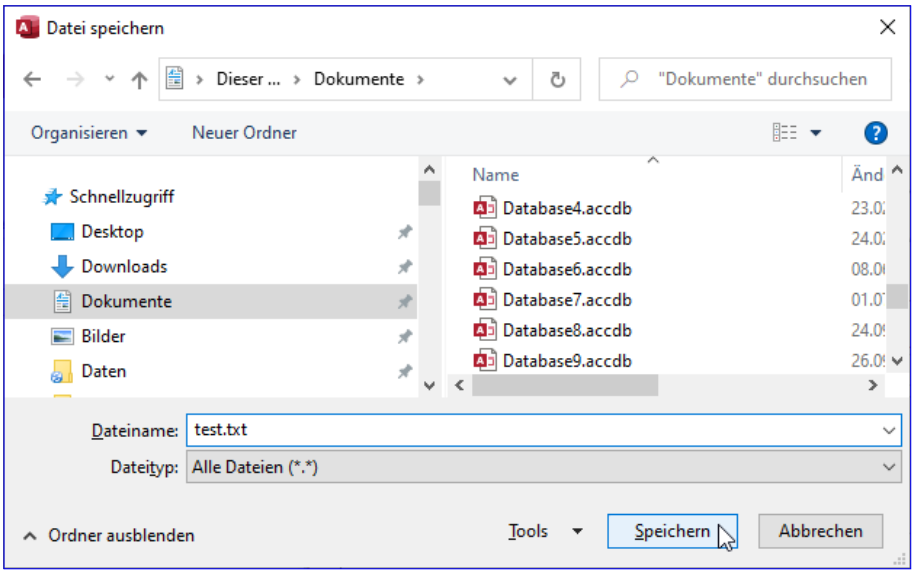

Bild 4: Der **FileDialog** mit **msoFileDialogSaveAs** erlaubt auch die Eingabe noch nicht vorhandener Dateinamen.

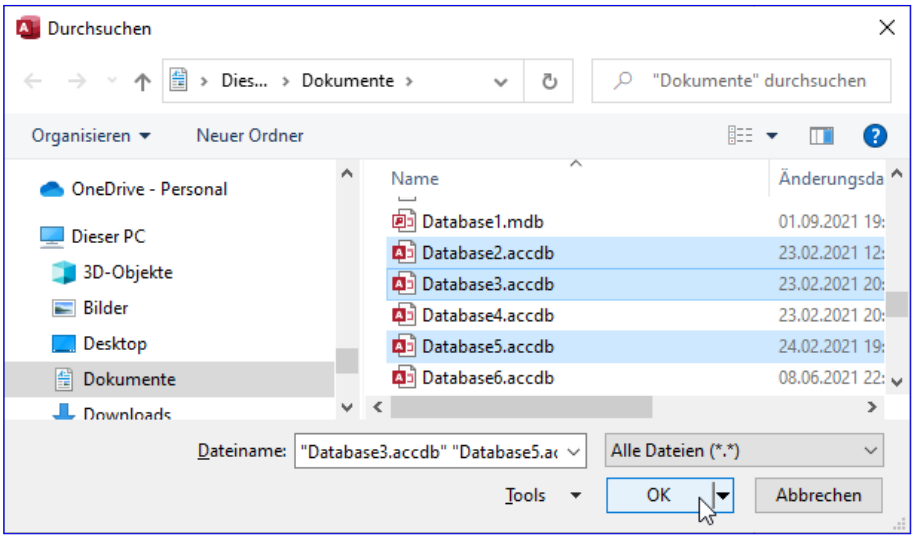

Bild 5: Mehrfachauswahl mit dem **FileDialog**-Objekt

Danach ist allerdings eine andere Auswertung erforderlich als bei der einfachen Auswahl, denn wir erhalten ja nicht nur einen, sondern gegebenenfalls auch mehrere Einträge zurück. Im Beispiel aus Listing 1 ermitteln wir zunächst die Anzahl der selektierten Dateien und geben diese im Direktbereich aus. Anschließend durchlaufen wir in einer Schleife alle Einträge der Auflistung **SelectedItems** und geben diese ebenfalls im Direktbereich aus.

Beschriftung der Schaltfläche zum Auswählen ändern

Mit der Eigenschaft **ButtonName** können Sie die sichtbare Bezeichnung der Schaltfläche zum Auswählen der gewählten Dateien oder Verzeichnisse ändern. Das gelingt allerdings nicht bei allen **FileDialog**-Typen auf Anhieb, sondern bei manchen erst nach der Auswahl eines Eintrags. Bei den Dialogen der Typen **msoFileDialogFolderPicker** und **msoFileDialogSaveAs** erscheint der

```
Private Sub FileDialog_FilePicker_Mehrfach()
    Dim objFiledialog As Office.FileDialog
    Dim l As Long
    Set objFiledialog = FileDialog(msoFileDialogFilePicker)
    objFiledialog.AllowMultiSelect = True
    If objFiledialog.Show = True Then
        Debug.Print "Es wurden " & objFiledialog.SelectedItems.Count & " Dateien ausgewählt:"
        For l = 1 To objFiledialog.SelectedItems.Count
            Debug.Print objFiledialog.SelectedItems(l)
        Next l
    Else
        Debug.Print "Keine Datei ausgewählt"
    End If
End Sub
```

Listing 1: Ausgabe einer Mehrfachauswahl im Direktbereich

gewünschte Text sofort. Das sieht beispielsweise wie im folgenden Codeausschnitt aus:

```
Dim objFiledialog As Office.FileDialog
Set objFiledialog = FileDialog(msoFileDialogFolderPicker)
objFiledialog.ButtonName = "FolderPicker"
If objFiledialog.Show = True Then
    ...
```

Das Ergebnis sehen Sie in Bild 6.

Bei den anderen Einstellungen wird der gewünschte Text erst angezeigt, wenn der Benutzer mindestens eine Datei ausgewählt hat (siehe Bild 7).

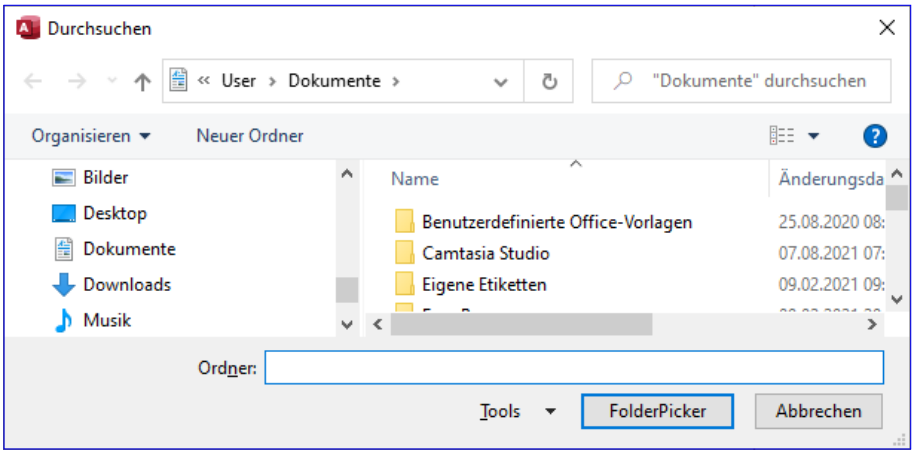

Bild 6: Ändern der Beschriftung der linken Schaltfläche

Überschrift des Dialogs einstellen

Mit der Eigenschaft **Title** können Sie den Text in der Titelzeile festlegen:

```
objFiledialog.Title = "Datei auswählen"
```

Beim Öffnen anzuzeigenden Ordner einstellen

Wenn Sie direkt beim Öffnen des Dialogs einen bestimmten Ordner anzeigen wollen, übergeben Sie diesen für die Eigenschaft **InitialFileName**.

Den Ordnernamen müssen Sie immer mit einem Backslash (\) abschließen. Wenn Sie beispielsweise das Verzeichnis **c:** anzeigen wollen, geben Sie **c:** an:

```
objFiledialog.InitialFileName =
"c:\"
```

Wollen Sie beispielsweise im Verzeichnis der aktuellen Datenbank starten, verwenden Sie die Funktion **CurrentProject.Path** und schließen dies mit dem Backslash-Zeichen ab:

```
objFiledialog.InitialFileName =
CurrentProject.Path & "\"
```

Beim Öffnen auszuwählende Datei vorbelegen

Gegebenenfalls wissen Sie auch schon, wie die auszuwählende Datei heißen soll. Das ist beispielsweise bei Konfigurationsdateien oft der Fall oder auch bei der Auswahl von Backenddatenbanken.

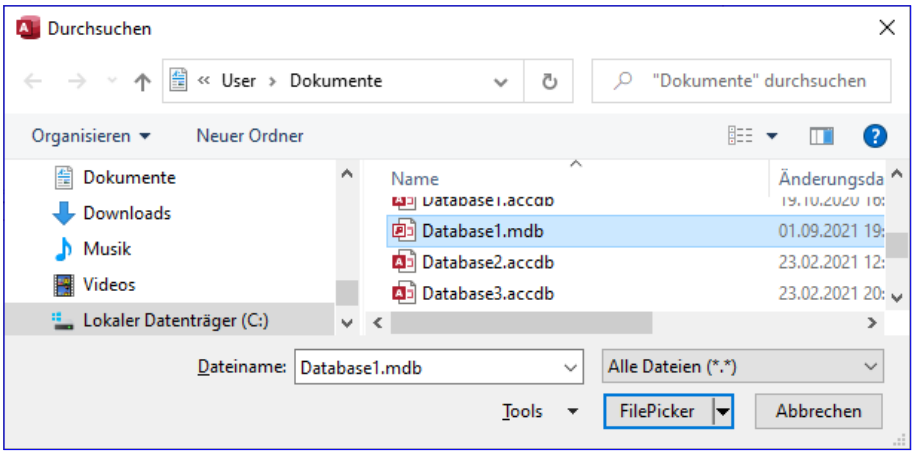

Bild 7: Ändern der Beschriftung der linken Schaltfläche erst nach erfolgter Auswahl

Dann geben Sie für die Eigenschaft **InitialFileName** einfach den vollständigen Pfad zu der auszuwählenden Datei an – hier für eine Backenddatenbank:

```
objFiledialog.InitialFileName = CurrentProject.Path & "\
Backend.accdb"
```

Ansicht beim Öffnen

Mit der Eigenschaft **InitialView** können Sie festlegen, welche Ansicht beim Öffnen des **FileDialog**-Objekts verwendet werden soll. Hier gibt es die folgenden Werte:

- **msoFileDialogViewDetails**
- **msoFileDialogViewLargeIcons**
- **msoFileDialogViewList**
- **msoFileDialogViewPreview**
- **msoFileDialogViewProperties**
- **msoFileDialogViewSmallIcons**
- **msoFileDialogViewThumbnail**
- **msoFileDialogViewTiles**
- **msoFileDialogViewWebView**

Auf dem Testsystem mit Windows 10 und Office 365 hat diese Einstellung keine Auswirkung.

Filtern der angezeigten Dateien

Sie können festlegen, nach welchen Dateien beziehungsweise Dateiendungen das **FileDialog**-Fenster filtern soll. Dazu verwenden Sie die Eigenschaft **FilterIndex**, mit der Sie festlegen, welcher der aufgeführten Filter verwendet werden soll und die Auflistung **Filters**, der Sie eigene Filter hinzufügen können.

Die Auflistung **Filters** enthielt auf dem Testsystem den Filter für alle Dateien, also für ***.***, und den Beschreibungstext **Alle Dateien**. Die **Filters**-Auflistung listet Elemente des Typs **Filter** auf, der wiederum verschiedene Eigenschaften liefert. Als Erstes können Sie beispielsweise im Direktbereich die Anzahl der definierten Filter ausgeben lassen:

```
? FileDialog(msoFileDialogFilePicker).Filters.Count
```

Um die Eigenschaften der einzelnen **Filter**-Elemente der **Filters**-Auflistung auszulesen, verwenden wir eine **For Each**-Schleife wie die Folgende:

```
Dim objFiledialog As Office.FileDialog
Dim objFilter As FileDialogFilter
Set objFiledialog = FileDialog(msoFileDialogFilePicker)
```

Datenzugriff mit .NET, LINQPad und LINQ to DB

.NET bietet eine ganze Menge Möglichkeiten, die uns unter Access/VBA nicht zur Verfügung stehen. Und diese Möglichkeiten wachsen ständig weiter, denn Entwickler stellen ihre eigenen Erweiterungen auf der NuGet-Plattform zur Verfügung. Was haben wir als Access-Entwickler nun davon? Mittlerweile gibt es Bibliotheken wie LINQ to DB, mit denen Sie leicht von einer .NET-Anwendung auf Access-Datenbanken zugreifen können. Und es gibt mit LINQPad eine Benutzeroberfläche, mit der Sie einfache Prozeduren mit Visual Basic programmieren können, ohne Visual Studio zu benötigen. Wir wollen diese beiden Tools als Vorbereitung zu einem weiteren Beitrag vorstellen. Dort werden wir diese nutzen, um die Tabellen einer Access-Datenbank mit zufälligen Beispieldaten zu füllen.

Wir werden in diesem Beitrag gleich mehrere Tools einführen, die Sie als Access-Entwickler noch nicht kennen – aber keine Sorge: Wir beschreiben alles ganz genau, sodass Sie die Techniken ohne Probleme für eigene Projekte anpassen können. Es handelt sich um die folgenden Tools:

- **LINQPad**: Dies ist ein Editor, mit dem Sie C#- und VB-Code eingeben und ausführen können, ohne dass Sie Visual Studio benötigen.

- **Linq To DB**: Dies ist eine Schnittstelle, mit der Sie beispielsweise von LINQPad direkt auf die Daten einer Access-Datenbank zugreifen können.

Damit haben wir ein perfektes Team: **LINQPad** ist unsere Entwicklungsumgebung zum Programmieren von Code, der unserer Datenbank Beispieldaten hinzufügt. **LINQ To DB** schafft die Verbindung zur Access-Datenbank.

Und in einem weiteren Beitrag namens **Beispieldaten generieren mit .NET und Bogus** (**www.access-im-unternehmen.de/1359**) zeigen wir, wie Sie mit einer weiteren Bibliothek namens **Bogus** die Tabellen einer Access-Datenbank mit zufällig generierten Beispieldaten füllen.

LINQPad herunterladen und installieren

Die »Spielwiese für .NET-Programmierer« können Sie unter dem folgenden Link herunterladen:

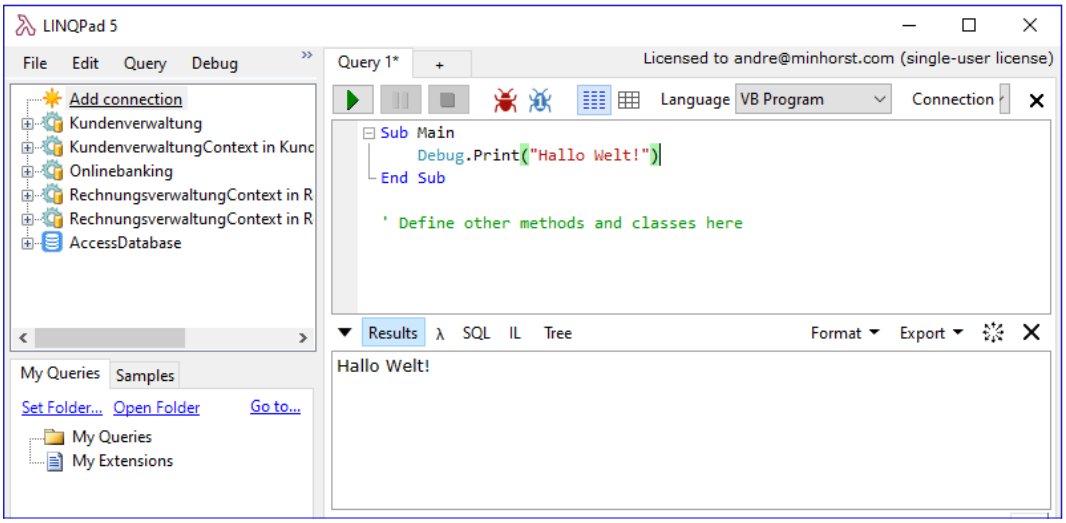

Bild 1: Ein erstes Beispiel in LINQPad

https://www.linqpad.net/

Danach installieren Sie das Tool und starten es.

Programme mit LINQPad schreiben

Da wir den Umgang mit VBA gewohnt sind, liegt es nahe, dass wir unter LINQPad **Visual Basic** nutzen und nicht C#. Um lauffähige Programme mit Prozeduren zu entwickeln, nutzen wir dort die Vorlage **VB Program**, die Sie für den einzigen bisher angezeigten Registerreiter unter **Language** auswählen.

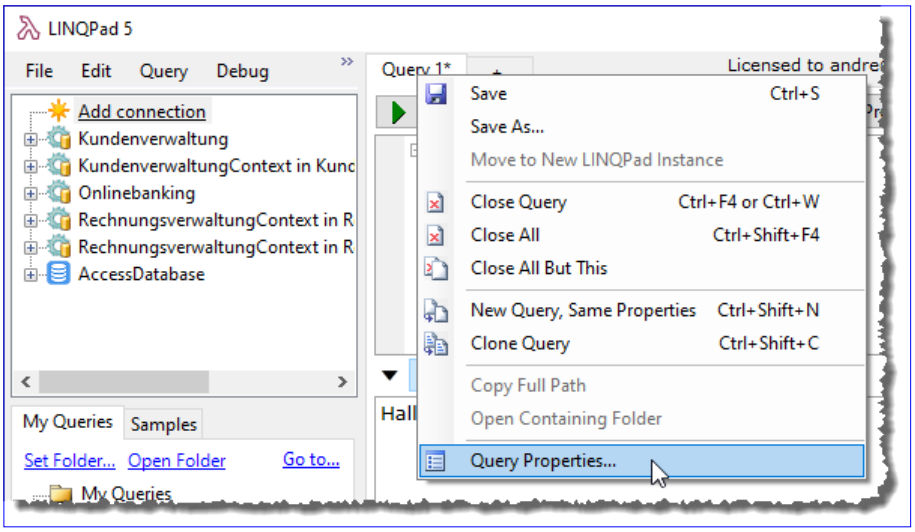

Bild 2: Anzeigen der Query Properties

Hier ergänzen Sie nun die bereits vorliegende Prozedur Main um eine **Debug.Print**-Anweisung und führen die Prozedur durch Betätigen der Taste **F5** aus (siehe Bild 1).

F5 löst immer die **Main**-Prozedur aus. Dieser können Sie nun den kompletten benötigten Code zuweisen, Sie können aber auch **Function**- oder **Sub**-Prozeduren definieren und aufrufen oder auch Klassen deklarieren und initialisieren.

An dieser Stelle direkt der Hinweis, dass Sie Parameter unter VB immer in Klammern einfassen müssen. Außerdem erfolgen Zuweisungen auch an Objektvariablen ohne das **Set**-Schlüsselwort.

Linq To DB installieren

Damit wir auf unsere Beispieldatenbank zugreifen können, benötigen wir eine Erweiterung namens **Linq To DB**. Um diese hinzuzufügen, öffnen Sie mit dem Kontextmenü-Eintrag **Query Properties...** für den aktuellen Registerreiter die Einstellung für das aktuelle Fenster (siehe Bild 2).

Dies öffnet den Dialog **Query Properties**, der unten die Schaltfläche **Add NuGet...** bereithält (siehe Bild 3). Über **NuGet**-Pakete können Sie verschiedene Pakete mit Tools zu einem Projekt hinzufügen.

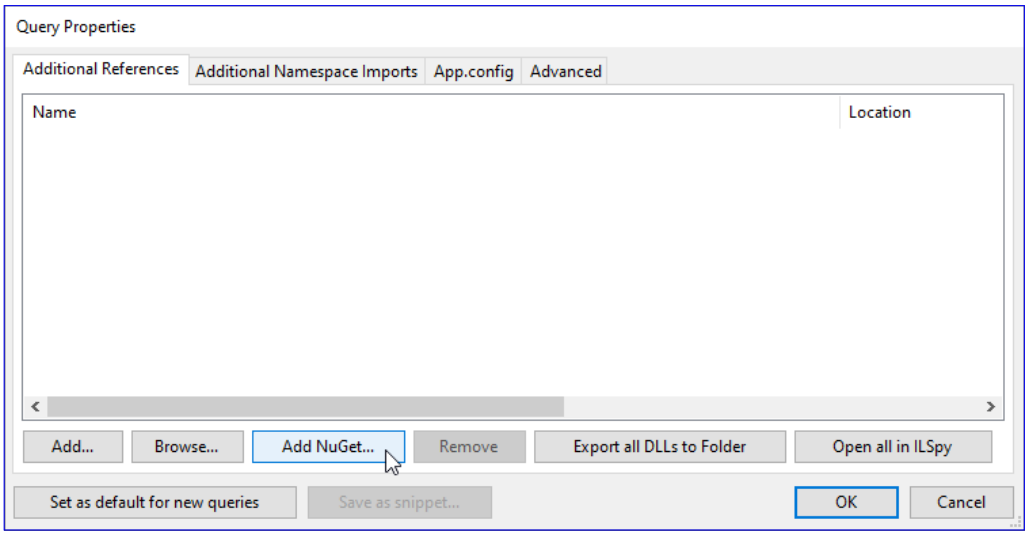

Bild 3: Hinzufügen von NuGet-Paketen

Klicken Sie diese an, erscheint der Dialog **LINQPad NuGetManager**.

Hier geben Sie unter **Search online** den Suchbegriff **Linq To DB** ein und finden schnell einen passenden Eintrag, den Sie mit einem Klick auf die Schaltfläche **Add to Query** zum aktuellen Query-Fenster hinzufügen (siehe Bild 4).

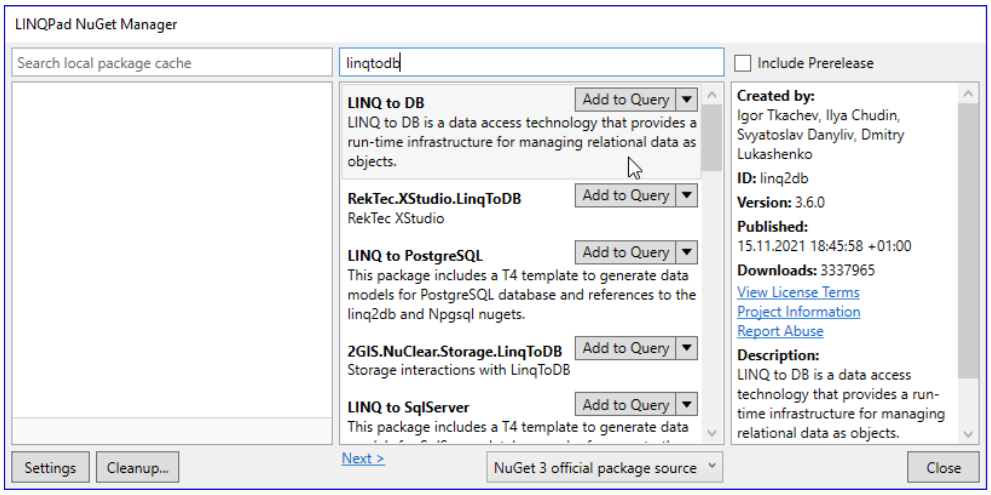

Bild 4: **LINQ to DB** hinzufügen

Tabellen der Datenbank verfügbar machen

Danach können Sie bereits eine Verbindung zur gewünschten Access-Datenbank hinzufügen. Dazu klicken Sie im linken Bereich des Fensters auf den Eintrag **Add connection**.

Es erscheint der Dialog **Choose Data Context**. Hier selektieren Sie im oberen Bereich den Eintrag **LINQ to DB** und klicken anschließend auf die Schaltfläche **Next** (siehe Bild 5).

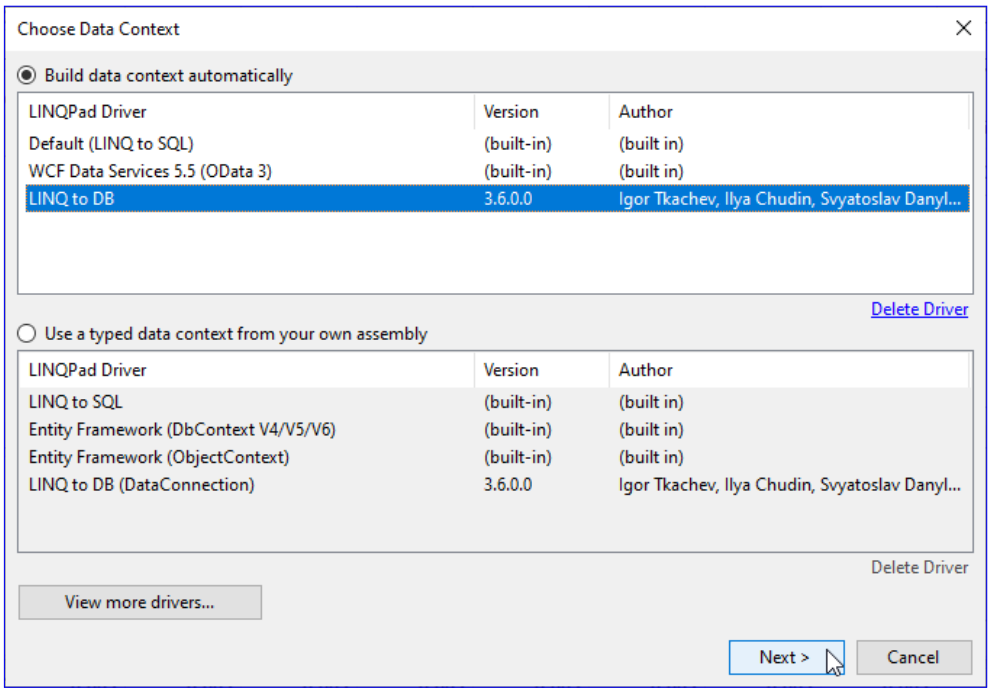

Bild 5: Herstellen einer Verbindung zur Datenbank

Der folgende Schritt zeigt den Dialog **LINQ to DB Connection** an. Hier wählen Sie oben zunächst den **Data Provider** aus, in diesem Fall **Microsoft Access (OleDb)**. Dann geben Sie die Verbindungszeichenfolge ein.

Bei der folgenden Verbindungszeichenfolge brauchen Sie nur den Pfad zur Access-Datenbank anzupassen:

```
Provider=Microsoft.ACE.OLEDB.12.0;Data Source=C:\
Users\...\LINQPad.accdb;Persist Security Info=False
```

Damit testen Sie schließlich noch die Verbindung (siehe Bild 6).

War dies erfolgreich, können Sie den Dialog mit **OK** beenden.

Beispieldaten generieren mit .NET und Bogus

Das Produzieren von Beispieldaten ist immer wieder eine mühselige Aufgabe. Beispieldaten benötigen Sie, um beim Entwickeln neuer Anwendungen die Funktionen zu testen, die mit der Anzeige, dem Bearbeiten oder Löschen von Daten zusammenhängen. Und auch zum Testen des Hinzufügens von Daten benötigen Sie gegebenenfalls schon Daten in verknüpften Tabellen zur Auswahl. Unter .NET gibt es verschiedene Bibliotheken, die das Generieren von Beispieldaten erleichtern. Leider sind diese nicht so ohne Weiteres unter Access verfügbar. Zum Glück gibt es Tools, mit denen Sie diese Bibliotheken dennoch für Ihre Zwecke einsetzen können. In diesem Beitrag nutzen wir den Editor LINQPad, um Beispieldaten mit der Bogus-Bibliothek zu erzeugen und diese dann mit der Bibliothek LINQ to DB den Tabellen einer Beispieldatenbank hinzuzufügen.

Vorbereitungen

Für die beschriebenen Techniken benötigen Sie die Entwicklungsumgebung LINQPad sowie die Datenzugriffsbibliothek LINQ to DB. Wie Sie diese beiden herunterladen, installieren und verwenden, beschreiben wir im Beitrag **Datenzugriff mit .NET, LINQPad und LINQ to DB** (**www.access-im-unternehmen.de/1358**).

Wie Sie diese beiden nutzen, um mit einer weiteren Bibliothek namens Bogus zufällige Beispieldaten für eine Datenbank zu erzeugen, zeigen wir im vorliegenden Beitrag.

Bogus zum Projekt hinzufügen

Zum Ermitteln der Beispieldaten, die wir zu den Tabellen der Anwendung hinzufügen wollen, nutzen wir die bereits eingangs erwähnte Bibliothek **Bogus**.

Diese müssen wir genau wie **LINQ to DB** zunächst verfügbar machen. Dazu klicken Sie unter **LINQPad** mit der rechten Maustaste auf den Registerreiter des **Query**-Bereichs und wählen den Eintrag **Query Properties...** aus (siehe Bild 1).

Im nun erscheinenden Dialog **Query Properties** klicken Sie auf die Schaltfläche **Add NuGet...** und geben im folgenden Dialog unter **Search online** den Suchbegriff **Bogus** ein. Den nun angezeigten Eintrag aus Bild 2 fügen Sie dann mit einem Klick auf die Schaltfläche **Add to Query** hinzu.

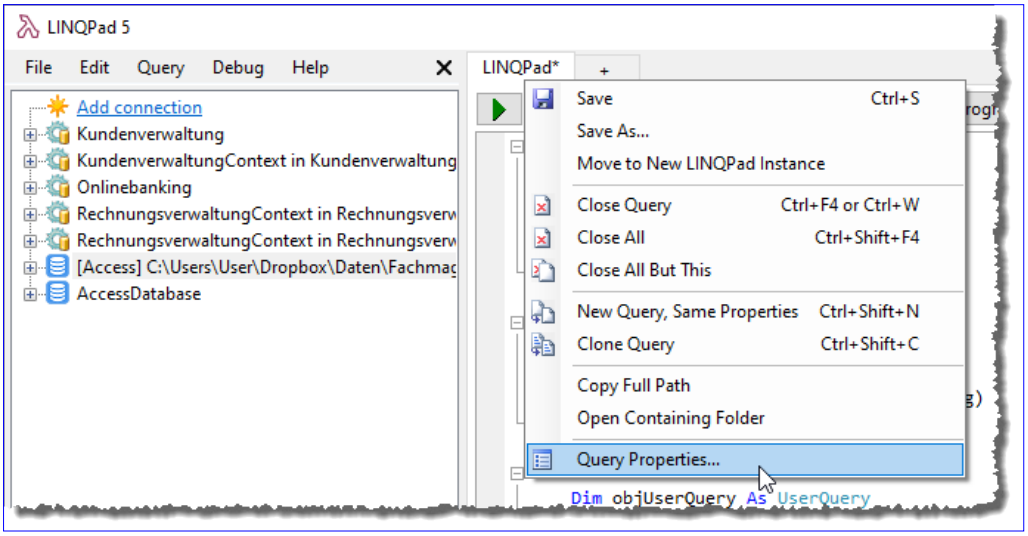

Bild 1: Aufrufen des Dialogs **Query Properties** per Kontextmenü

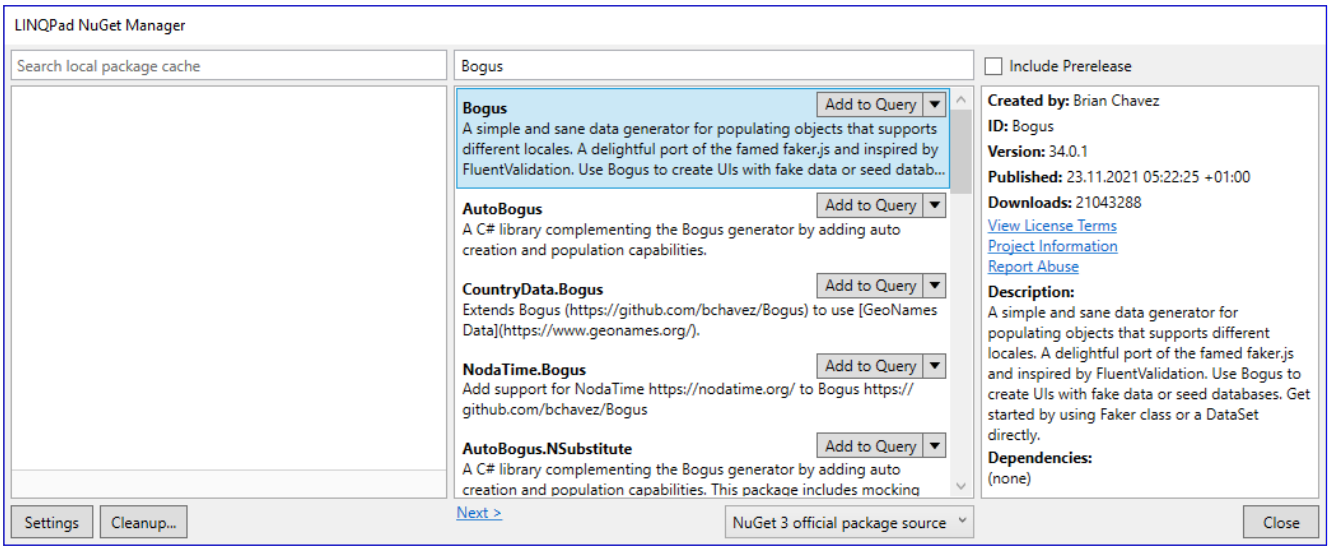

Bild 2: Hinzufügen des **Bogus**-Pakets

Es erscheint nun ein neuer Eintrag im linken Bereich des Dialogs **LINQPad NuGet Manager**. Hier klicken Sie noch auf **Add namespaces** (siehe Bild 3).

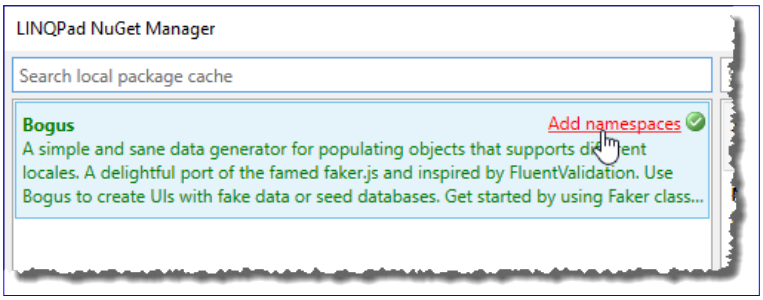

Bild 3: Hinzufügen des **Bogus**-Namespaces

Dies öffnet einen weiteren Dialog namens **Add Namespaces From NuGet Assemblies**. Hier wählen Sie den Eintrag **Bogus** aus und klicken auf **OK** (siehe Bild 4).

Nachdem Sie diesen Dialog und auch den Dialog **LINQPad NuGet Manager** geschlossen haben, finden Sie im Dialog **Query Properties** unter **Additional References** den Eintrag **Bogus** vor (siehe Bild 5).

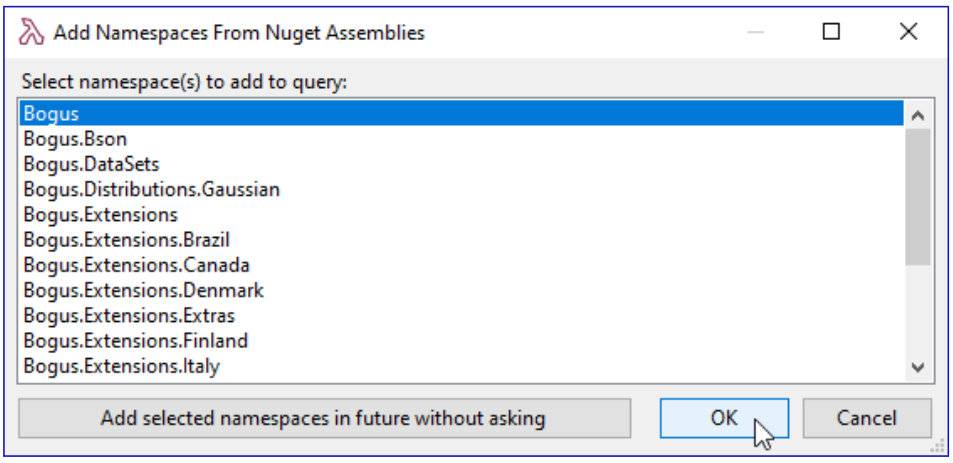

Bild 4: Hinzufügen eines Verweises auf den **Bogus**-Namespace

Damit sind die Vorbereitungen abgeschlossen und wir können uns dem Einsatz von Bogus zum Generieren von Beispieldaten zuwenden.

Vorweg: Anreden anlegen

Einen Schritt erledigen wir allerdings noch vorneweg – das Anlegen der Datensätze in der Tabelle **tblAnreden**. Für diese benötigen wir keinen Zufallsgenerator und somit auch nicht

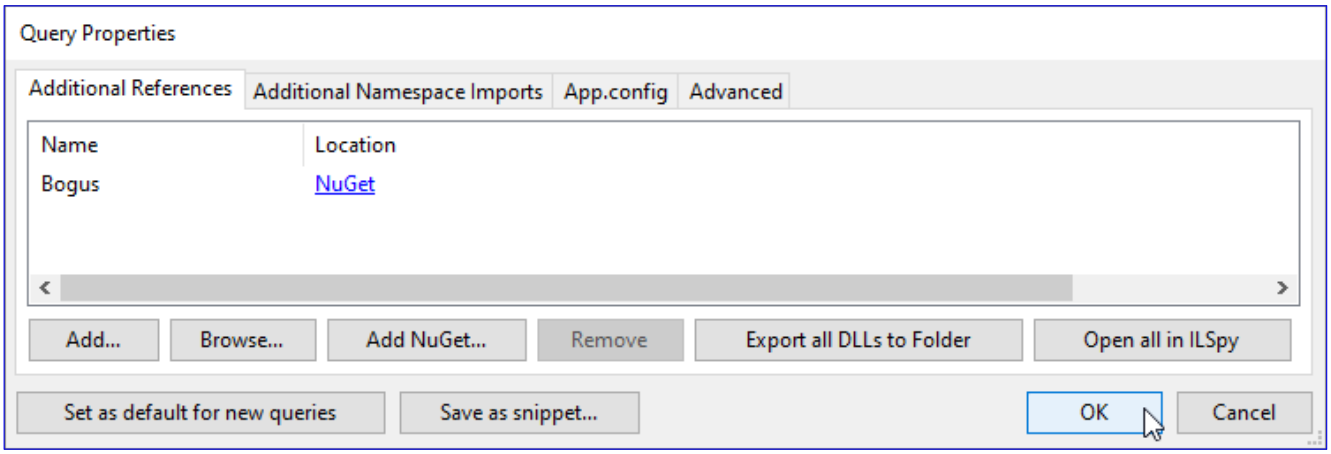

Bild 5: Der Verweis auf den **Bogus**-Namespace

die Bibliothek **Bogus**. Wir fügen die Anreden einfach mit der folgenden Prozedur hinzu:

```
Public Sub AnredenAnlegen
    Dim objUserQuery As UserQuery
    Dim objAnrede As tblAnreden
    objUserQuery = tblAnreden.DataContext
    objAnrede = New tblAnreden
    With objAnrede
        .ID = 1
        .Anredebezeichnung = "Herr"
    End With
    objUserQuery.Insert(objAnrede)
    objAnrede = New tblAnreden()
    With objAnrede
        .ID = 2
        .Anredebezeichnung = "Frau"
    End With
    objUserQuery.Insert(objAnrede)
End Sub
```

Den Code können wir mit Visual Basic übrigens auch wie folgt schreiben und sparen dabei nicht nur eine Variable, sondern auch einige Zeilen Code:

```
Public Sub AnredenAnlegen
    Dim objUserQuery As UserQuery
    objUserQuery = tblAnreden.DataContext
    objUserQuery.Insert(New tblAnreden With {.ID = 1, _
                        .Anredebezeichnung = "Herr"})
    objUserQuery.Insert(New tblAnreden With {.ID = 2, _
                        .Anredebezeichnung = "Frau"})
End Sub
```

Damit wir immer mit einer frisch aufgesetzten Tabelle **tblAnreden** starten, können wir zuvor die vorhandenen Anreden löschen:

```
Public Sub AnredenLoeschen
    Dim objUserQuery As UserQuery
    objUserQuery = tblAnreden.DataContext
    objUserQuery.Execute("DELETE FROM tblAnreden")
End Sub
```

Und schließlich geben wir die frisch erzeugten Anreden nachher einmal im Direktbereich von LINQPad aus:

```
Public Sub AnredenAusgeben
    Dim objAnrede As tblAnreden
    For Each objAnrede In tblAnreden
        Debug.Print(objAnrede.ID & " " _
                    & objAnrede.Anredebezeichnung)
    Next objAnrede
End Sub
```

Zufallsdaten mit Bogus ermitteln

Bogus bietet verschiedene Klassen, die Funktionen zum Erstellen von Zufallsdaten thematisch zusammenfassen. Diese werden unter folgendem Link ausführlicher beschrieben:

```
https://github.com/bchavez/Bogus#bogus-api-support
```

Es gibt die folgenden Klassen:

- **Address**: Liefert verschiedene Eigenschaften von Adressen wie Straße, PLZ, Ort und Land, aber auch beispielsweise Höhen- und Breitengrade oder Angaben wie Himmelsrichtungen.
- **Commerce**: Interessant für alles, was mit dem Handel zu tun hat. Bietet Daten wie Produktnamen, Farben, Adjektive oder Materialien, Preise, Werte für Barcodes oder Kategorien und Abteilungen.
- **Company**: Liefert Firmennamen, Gesellschaftsformen und Phrasen zu Unternehmen
- **Database**: Liefert Feldnamen, Datentypen und weitere Datenbank-relevante Informationen
- **Date**: Liefert Datumsangaben in verschiedenen Bereichen sowie Monate oder Wochentage
- **Finance**: Gibt beispielsweise IBANs aus, wobei Sie festlegen können, aus welchem Land diese stammen und ob diese formatiert sein sollen. Außerdem liefert diese Klasse Geldbeträge, Transaktionstypen, Währungen, Kreditkartennummern, Prüfziffern und vieles mehr.
- **Hacker**: Liefert zufällige Abkürzungen, Adjektive, Nomen, Verben oder komplette Phrasen, allerdings alles nur auf Englisch.
- **Images**: Hiermit können Sie beispielsweise URLs zum Download von Bildern abfragen, welche Bilder in den angegebenen Dimensionen liefern.
- **Internet**: Erstellt E-Mail-Adressen, Benutzernamen, Domainnamen, Domainendungen, Portnummern, IP-Adressen, Kennwörter und vieles mehr.
- **Lorem**: Bietet einige Zufallsdaten rund um die Erstellung von Texten. Hierbei können Sie einzelne Wörter, Absätze, Buchstaben, Sätze et cetera generieren lassen – allerdings nur auf Lateinisch!
- **Name**: Liefert Vornamen, Nachnamen, komplette Namen, aber auch Berufsbezeichnungen
- **Phone**: Liefert Telefonnummern
- **Rant**: Liefert Reviews, allerdings nur auf Englisch
- **System**: Liefert Dateinamen, Verzeichnisse (nur im Unix-Format), Dateiendungen, Versionsnummern, Texte von Ausnahmen, Fahrzeugnummern
- **Vehicle**: Liefert Fahrzeughersteller, -modelle und -typen, VINs (Fahrzeugnummern) und Kraftstoffarten
- **Random**: Liefert Zufallswerte für die verschiedenen Datentypen und in den angegebenen Bereichen

Zufallsdaten mit der Faker-Klasse

Mit den in den Klassen enthaltenen Funktionen können Sie die verschiedensten Daten ermitteln. Um dies auszuprobieren, erstellen Sie einfach eine neues Objekt auf Basis der Klasse **Bogus.Faker** und greifen dann direkt auf die Klassen aus der obigen Auflistung und die darin enthaltenen Funktionen zu.

Im folgenden Beispiel schreiben wir den Code direkt in die **Main**-Klasse der LINQPad-Datei:

```
Sub Main
    Dim objFaker As New Bogus.Faker
    Debug.Print(objFaker.Name.FirstName)
End Sub
```

Dies gibt einfach einen zufällig gewählten Nachnamen im Direktbereich aus. Auf die gleiche Weise probieren Sie auch die anderen Funktionen dieser und anderer Klassen aus.

Adressdaten für Deutschland

Beim Ausprobieren von Funktionen wie **Address.StreetAddress**, **Address.ZipCode** oder **Address.City** werden Sie feststellen, dass die Daten sich auf englische Adressen beziehen.

Sie können allerdings leicht auf deutsche Anschriften umschwenken, indem Sie beim Erstellen der **Faker**-Klasse den Wert **"de"** als Parameter übergeben. Die folgenden Zeilen liefern beispielsweise direkt deutsche Namen und Adressdaten:

```
Dim objFaker As New Bogus.Faker("de")
Debug.Print(objFaker.Name.FirstName)
Debug.Print(objFaker.Address.StreetAddress)
Debug.Print(objFaker.Address.ZipCode)
Debug.Print(objFaker.Address.City)
```

Kunden anlegen

Nachdem Sie aus dem Beitrag **Datenzugriff mit .NET, LINQPad und LINQ to DB** (**www.access-im-unternehmen.de/1358**) wissen, wie Sie neue Datensätze mit den gewünschten Feldwerten zu einer Tabelle wie **tblKunden** hinzufügen, haben Sie nun prinzipiell alle Techniken zum Erstellen von auf Zufallsdaten basierenden Datenbankinhalten zur Hand.

Sie brauchen die Anweisungen zum Füllen der Felder nur noch mit den oben vorgestellten Funktionen der verschiedenen Klassen des **Bogus.Faker**-Objekts zu füllen.

Dann müssten wir allerdings für jeden Kunden ein neues **Bogus.Faker**-Objekt erstellen. Das brauchen wir jedoch nur ein einziges Mal zu tun, denn die **Bogus.Faker**-Klasse bietet eine Methode namens **Generate** an. Mit der können Sie, wenn Sie für den Parameter einen Zahlenwert angeben, eigentlich sogar gleich mehrere Objekte gleichzeitig erzeugen, aber diese können wir im aktuellen Kontext nicht verarbeiten. Also nutzen wir die **Generate**-Methode nur zum Erstellen eines Kunden gleichzeitig.

Bevor wir mit dem eigentlichen Code zum Erstellen einsteigen, deklarieren wir noch eine **Enum**-Auflistung für das Geschlecht – wozu wir diese benötigen, erläutern wir anschließend:

```
Public Enum Gender
    Male = 0
    Female = 1
End Enum
```

Die Prozedur **KundenAnlegen** erwartet die Anzahl der anzulegenden Kunden als Parameter. Dann deklariert sie die benötigten Variablen:

```
Public Sub KundenAnlegen(intMenge As Int32)
    Dim Beispielkunde As New tblKunden
    Dim objUserQuery As UserQuery
    Dim i As Int32
    Dim intGender As int32
```

Sie erstellt dann ein **UserQuery**-Objekt auf Basis der Tabelle **tblKunden** und ein neues **Bogus.Faker**-Objekt, dem Sie die Klasse **tblKunden** als Typ zuweisen.

Außerdem fügen wir noch den Wert **"de"** als Parameter an, damit Bogus deutsche Daten liefert:

```
    objUserQuery = tblKunden.DataContext
    Dim objFaker As New Bogus.Faker(Of tblKunden)("de")
```

Dann folgt ein Teil, von dem nur die inneren Zeilen interessant sind – und mit dem wir die Regeln festlegen, nach denen wir neue Objekte des Typs **tblKunden** füllen.

Wichtig ist nur, dass Sie wissen, dass der Buchstabe **k** für das **tblKunden**-Objekt steht und **f** für das **Bogus.Faker**-

Objekt. In der ersten Anweisung nutzen wir die Funktion **PickRandom**, um zufällig einen der beiden Werte der Enumeration **Gender** zu ermitteln (also **0** für männlich und **1** für weiblich) und schreiben das Ergebnis in die Variable **intGender**:

```
objFaker.Rules(Function(f, k)
    intGender = f.PickRandom(Of Gender)
```

Diesen Wert benötigen wir anschließend gleich zwei Mal. Als Erstes addieren wir den Wert **1** hinzu und tragen das Ergebnis für die Eigenschaft **AnredeID** ein, von der wir wissen, dass der Wert **1** der Anrede **Herr** und der Wert **2** der Anrede **Frau** entspricht:

```
        k.AnredeID = intGender + 1
```

Dann übergeben wir **intGender** als Parameter der Funktion **FirstName** zum Ermitteln eines zufälligen Vornamens. Durch diesen Parameter legen wir fest, ob **FirstName** einen männlichen oder einen weiblichen Nachnamen liefert:

```
        k.Vorname = f.Name.FirstName(intGender)
```

Die übrigen Zeilen legen fest, aus welchen Funktionen und Klassen von **Bogus.Faker** die Daten für die übrigen Felder bezogen werden sollen:

```
        k.Nachname = f.Name.LastName()
        k.Strasse = f.Address.StreetAddress
        k.PLZ = f.Address.ZipCode
        k.Ort = f.Address.City
        k.Land = f.Address.Country
        k.EMail = f.Internet.Email
        k.Telefon = f.Phone.PhoneNumber
        k.Telefax = f.Phone.PhoneNumber
    End Function)
```

Damit haben wir die Vorlage für das Erstellen von Elementen des Typs **tblKunden** definiert.

Damit können wir nun in eine **For...Next**-Schleife einsteigen, welche die Werte von **1** bis zur Anzahl der zu erstellenden Elemente aus dem Parameter **intMerge** durchläuft. Innerhalb der Schleife erstellen wir mit der **Generate**-Methode auf Basis des **objFaker**-Objekts ein neues Element des Typs **tblKunden** und referenzieren es mit der Variablen **Beispielkunde**.

Diesem weisen wir dann noch den Wert aus der Laufvariablen **i** für das Feld **ID** hinzu und speichern es mit der **Insert**-Methode in dem mit **objUserQuery** referenzierten **UserQuery**-Objekt auf Basis der Tabelle **tblKunden**:

```
    For i = 1 To intMenge
        Beispielkunde = objFaker.Generate
        Beispielkunde.ID = i
        objUserQuery.Insert(Beispielkunde)
    Next
End Sub
```

Die so erzeugten Kunden können Sie sich in der Beispieldatenbank ansehen oder Sie lassen sich diese mit einer einfachen Prozedur in LINQPad ausgeben:

```
Public Sub KundenAusgeben
    Dim objKunde As tblKunden
    For Each objKunde In tblKunden
        Debug.Print(objKunde.ID & " " & objKunde.AnredeID 7
                            & " " & objKunde.Vorname & " " 7
                                 & objKunde.Nachname & " " 7
                                  & objKunde.Strasse & " " 7
                        & objKunde.PLZ & " " & objKunde.Ort)
    Next objKunde
End Sub
```

Damit wir immer frisch loslegen können, erstellen wir noch eine Prozedur zum Löschen der bereits vorhandenen Kunden:

```
Public Sub KundenLoeschen
    Dim objUserQuery As UserQuery
```

```
    objUserQuery = tblKunden.DataContext
    objUserQuery.Execute("DELETE FROM tblKunden")
End Sub
```

Wenn Sie nun noch die folgenden Prozeduraufrufe in der Hauptmethode **Main** platzieren, erstellt diese die Daten in den Tabellen **tblAnreden** und **tblKunden** nach dem Löschen jeweils neu und gibt diese im Direktbereich von LINQPad aus:

```
Sub Main
    KundenLoeschen
    AnredenLoeschen
    AnredenAnlegen
    AnredenAusgeben
    KundenAnlegen(10)
    KundenAusgeben
End Sub
```

Beispieldaten für umfangreicheres Datenmodell

Nachdem wir uns die Grundlagen angesehen haben, wollen wir nun noch ein etwas umfangreicheres Datenmodell mit Beispieldaten füllen.

Dabei wollen wir die gängigsten Beziehungstypen abdecken – also 1:n- und m:n-Beziehungen. Dazu schauen wir uns das Datenmodell aus Bild 6 an. Die Reihenfolge beim Befüllen der Tabellen sieht wie folgt aus:

- **tblAnreden** (enthält keine Fremdschlüsselfelder)
- **tblProdukte** (enthält keine Fremdschlüsselfelder)
- **tblKunden** (verweist nur auf Tabelle **tblAnreden**)
- **tblBestellungen** (verweist nur auf Tabelle **tblKunden**)
- **tblBestellpositionen** (verweist auf die Tabellen **tblBestellungen** und **tblProdukte**)

Dies ist die Reihenfolge zum Füllen mit Daten. Wenn wir die Tabellen vor dem Füllen mit Beispieldaten leeren wollen, müssen wir das in umgekehrter Reihenfolge erledigen – zumindest, wenn referenzielle Integrität ohne Löschweitergabe aktiviert ist und beispielsweise das Löschen von Daten aus der Tabelle **tblAnreden** nicht möglich ist, wenn es noch Datensätze in der Tabelle **tblKunden** gibt, die noch mit den zu löschenden Daten verknüpft sind.

Hinzufügen von Produkten

Die Produkte-Tabelle haben wir mit einem speziellen Feature ausgestattet. Es enthält ein Anlage-Feld zum Spei-

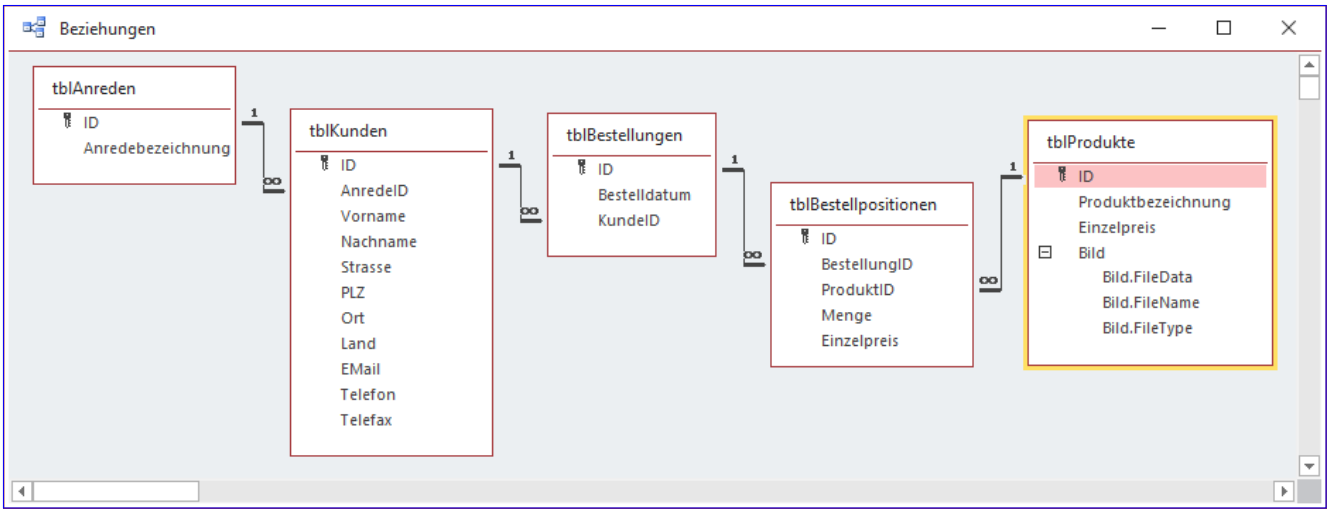

Bild 6: Datenmodell zum Füllen mit Beispieldaten

E-Mails mit Anlagen mit Outlook versenden

Das Versenden von E-Mails haben wir bereits ausführlich in Access im Unternehmen beschrieben. Dort kam auch gelegentlich das Thema auf, wie Sie Dateien an solche E-Mails anhängen. Allerdings gibt es bei genauerem Hinsehen Anforderungen, die wir noch nicht behandelt haben – zum Beispiel das Anhängen vieler Dateien auf einen Rutsch oder auch das Hinzufügen von Dateien, die nicht aus einem Verzeichnis stammen. Hierzu sind dann mehrere Aufrufe des jeweiligen Dialogs zum Auswählen der Dateien erforderlich. Auf diese Spezialfälle gehen wir im vorliegenden Beitrag ein.

Anforderungen an die Beispiellösung

Dieser Beitrag soll eine Beispiellösung mit folgenden Funktionen beschreiben:

- Versenden einer E-Mail unter Angabe von Empfänger, Betreff und Inhalt
- Hinzufügen von Anlagen über einen Dateiauswahldialog. Dabei sollen mehrere Anlagen aus einem Verzeichnis gleichzeitig selektiert werden können und beim Hinzufügen weiterer Anlagen aus einem weiteren Verzeichnis sollen die bereits hinzugefügten Anlagen erhalten bleiben.
- Bereits hinzugefügte Anlagen sollen markiert und aus der E-Mail gelöscht werden können.

Vorbereitungen

Wir benötigen Verweise auf zwei VBA-Bibliotheken. Diese fügen Sie dem VBA-Projekt über den **Verweise**-Dialog hinzu, den Sie mit dem Menübefehl **Extras|Verweise** des VBA-Editors öffnen:

- **Microsoft Office x.0 Object LIbrary**: Liefert das **FileDialog**-Objekt, mit dem wir den Dialog zum Auswählen der Anlagen anzeigen.
- **Microsoft Outlook x.0 Object Library**: Stellt das **MailItem**-Objekt zum Versenden der E-Mail bereit.

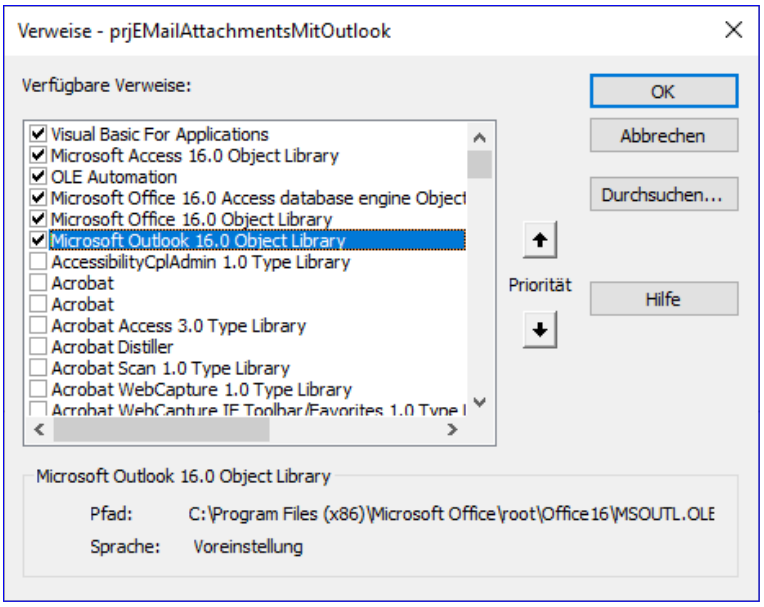

Bild 1: Verweise des VBA-Projekts

Mit den beiden hinzugefügten Verweisen sieht der Verweise-Dialog wie in Bild 1 aus.

Formular zum Erstellen der E-Mail mit Anlage

Das Formular, in dem wir das Versenden von E-Mails mit Anlagen steuern wollen, heißt **frmMailsMitAttachment** und sieht in der Entwurfsansicht wie in Bild 2 aus.

Das Formular verwendet die folgenden Steuerelemente:

- **txtAn**: Dient der Erfassung des Empfängers der E-Mail.
- **txtBetreff**: Nimmt den Betreff der E-Mail auf.

- **txtInhalt**: Hier können Sie den Inhalt der E-Mail eingeben.

- **lstAnlagen**: Listenfeld zur Anzeige der bereits hinzugefügten Anlagen. Das Listenfeld soll die Mehrfachauswahl erlauben, damit auch mehrere Anlagen gleichzeitig aus der Liste entfernt werden können. Dazu legen wir für die Eigenschaft **Mehrfachauswahl** den Wert **Erweitert** fest. Außerdem stellen wir für dieses Listenfeld die Eigenschaften **Horizontaler Anker** und **Vertikaler Anker** auf **Beide** ein, damit das Listenfeld mit dem Formular vergrößert werden kann. Damit das dazugehörige Bezeichnungsfeld an Ort und Stelle bleibt, erhalten **Horizontaler Anker** und **Vertikaler Anker** hier die Werte **Links** und **Oben**. Damit wir die ausgewählten Dateien einzeln hinzufügen können, stellen wir außerdem die Eigenschaft **Herkunftstyp** auf **Wertliste** ein.

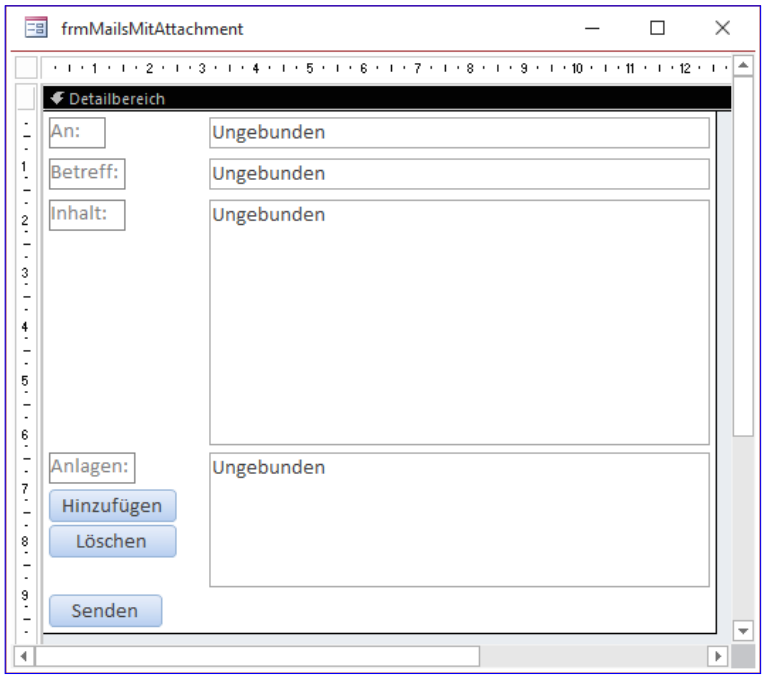

Bild 2: Formular in der Entwurfsansicht

- **cmdHinzufuegen**: Öffnet einen Dateiauswahldialog zum Auswählen der hinzuzufügenden Anlagen und fügt diese nach der Auswahl zum Listenfeld **lstAnlagen** hinzu.

- **cmdLoeschen**: Löscht die aktuell markierten Einträge des Listenfeldes. Die Schaltfläche soll nur markiert sein, wenn im Listenfeld **lstAnlagen** mindestens ein Eintrag markiert ist.

- **cmdSenden**: Sendet die E-Mail mit den Angaben der Felder **txtAn**, **txtBetreff** und **txtInhalt** und mit den im Listenfeld **lstAnlagen** angegebenen Dateien. Für diese Schaltfläche stellen wir die Eigenschaft **Vertikaler Anker** auf **Unten** ein, damit es beim Vergrößern des Formulars unten verankert ist.

Die Eigenschaften **Datensatzmarkierer**, **Navigationsschaltflächen**, **Bildlaufleisten** und **Trennlinien** sollen den Wert **Nein** und die Eigenschaft **Automatisch zentrieren** den Wert **Ja** haben.

Anlagen zur Liste hinzufügen

Die Schaltfläche **cmdHinzufuegen** soll einen Dateiauswahldialog öffnen, mit dem der Benutzer eine oder mehrere Dateien aus einem Verzeichnis auswählen und zur Liste hinzufügen kann. Die Liste zeigt die Dateien dann mit dem vollständigen Pfad an. Dabei wollen wir das mehrfache Hinzufügen von Dateien unterbinden. Dies wäre möglich, da wir erlauben wollen, dass der Benutzer durch mehrfaches Anklicken der **Hinzufügen**-Schaltfläche auch mehrfach Dateien aus dem gleichen Verzeichnis hinzufügen könnte.

Also prüfen wir vor dem Hinzufügen einer Datei zum Listenfeld **lstAnlagen**, ob diese bereits in der Liste enthalten ist.

Die durch die Schaltfläche **cmdHinzufuegen** ausgelöste Prozedur finden Sie in Listing 1. Hier deklarieren wir zwei Laufvariablen sowie das **FileDialog**-Objekt und

eine **Boolean**-Variable. Die Variable **objFileDialog** füllen wir mit einem Objekt der Klasse **FileDialog** mit dem Typ **msoFileDialogPicker** – mehr Informationen über diese Klasse finden Sie im Beitrag **Dateien und Verzeichnisse auswählen mit FileDialog** (**www.access-im-unternehmen.de/1345**).

Wir stellen mit der Eigenschaft **AllowMultiSelect** ein, dass der Benutzer mehrere Dateien gleichzeitig auswählen kann. Mit **Title** legen wir den Titel fest, dann leeren wir mit **Filters.Clear** die vorhandenen Filter und fügen mit **Add** den Filter für alle Dateitypen hinzu. Die Bezeichnung der Schaltfläche zum Übernehmen der Auswahl stellen wir auf **ButtonName** ein.

Danach folgt der Aufruf des Dialogs mit der **Show**-Methode, was wie in Bild 3 aussieht. Nach der Auswahl mit der Schaltfläche **Hinzufügen** wird der **If**-Teil der **If...Then**-Ab-

```
Private Sub cmdHinzufuegen_Click()
    Dim objFileDialog As FileDialog
    Dim l As Long
    Dim m As Long
    Dim bolVorhanden As Boolean
    Set objFileDialog = FileDialog(msoFileDialogFilePicker)
    With objFileDialog
        .AllowMultiSelect = True
        .Title = "Anlagen auswählen"
        .Filters.Clear
        .Filters.Add "Alle Dateien", "*.*"
        .ButtonName = "Hinzufügen"
        If .Show = True Then
            For l = 1 To .SelectedItems.Count
                If Not Len(Me!lstAnlagen.RowSource + .SelectedItems(l)) > 32750 Then
                    bolVorhanden = False
                    For m = 1 To Me!lstAnlagen.ListCount
                        If Me!lstAnlagen.ItemData(m) = .SelectedItems(l) Then
                            bolVorhanden = True
                            Exit For
                        End If
                    Next m
                    If Not bolVorhanden Then
                        Me!lstAnlagen.AddItem .SelectedItems(l)
                    End If
                Else
                    MsgBox "Es können keine weiteren Dateien zur Liste hinzugefügt werden."
                    Exit Sub
                End If

            Next l
        End If
    End With
End Sub
```

Listing 1: Prozedur zum Auswählen von Dateien