AUSGABE 05/2025

Access, SQL und Cloud AUTOMATION

MAGAZIN FÜR DIE PROGRAMMIERUNG VON MICROSOFT ACCESS, SQL SERVER UND CLOUD-AUTOMATIONEN MIT VBA UND CO.

IN DIESEM HEFT:

André Minhorst Verlag

Aus VB-Entwickler wird Access, SQL und Cloud Automation

Um der Entwicklung in der Welt rund um VBA,VB, Access und den übrigen Office-Anwendungen gerecht zu werden, haben wir den Titel dieses Magazins geändert – und damit wird sich auch der Inhalt auf neue Schwerpunkte konzentrieren.

In Access, SQL und Cloud Automation werden wir uns um alle Themen kümmern, die über das reine Programmieren von Access-Anwendungen hinausgehen. Dabei behandeln wir die folgenden Schwerpunkte:

- Access: Hier zeigen wir fortgeschrittene Techniken rund um die Entwicklung von Access-Anwendungen. VBA-Programmierung, Fehlerbehandlung, API-Programmierung – alles, was über das reine Erstellen von Tabellen, Abfragen, Formularen und Berichten hinausgeht.

- SQL Server: Wir zeigen, wie man zum SQL Server migriert, wie Access optimal mit SQL Server zusammenarbeitet und wie wir von Access aus performant auf SQL Server-Datenbanken zugreifen können.

- Cloud: Wer langfristig mit Access arbeiten möchte, muss Kompatibilität zu den vielen verschiedenen Cloud-Diensten und Saas-Lösungen herstellen und in der Lage sein, diese von Access aus zu steuern und Daten mit diesen auszutauschen. Genau darum kümmern wir uns, zum Beispiel durch Verwendung von VBA und Rest APIs.

Und natürlich automatisieren wir alles rund um Microsoft Access. Egal, ob mit reinem VBA innerhalb der Anwendung oder durch COM-DLLs, mit denen wir Funktionalitäten in Form von Bibliotheken bereitstellen oder durch COM-Add-Ins, mit denen wir die Funktionen der Benutzeroberfläche von Access oder dem VBA-Editor erweitern.

Alle aktuellen Artikel in unserer neuen Lernplattform

Außerdem findest Du ab jetzt alle Artikel in unserer neuen, modernen Lernplattform unter der folgenden Adresse:

`https://minhorst.learningsuite.io`

Dafür musst Du Dich dort allerdings erst einmal registrieren. Dein unschlagbarer Vorteil: Du brauchst Dich nur einmal zu registrieren und Deine Zugangsdaten gelten von da an für ein gesamtes Jahr.

Die Registrierung führst Du hier durch:

`https://andreminhorst.de/anmeldung-an-learningsuite`

Bitte gib dort neben Deinem Vornamen, Deinem Nachnamen und Deiner E-Mail-Adresse den Benutzernamen und das Kennwort an, dass Du auf Seite 2 dieses PDF-Dokuments findest. Danach bekommst Du eine Mail mit einem Link, über den Du die Registrierung abschließen kannst.

Du findest in der Lernplattform übrigens auch ein Forum, in dem Du Dich mit mir und anderen Lesern direkt austauschen kannst!

Achtung: vbentwickler.de bleibt erhalten!

Aus technischen Gründen findest Du die Artikel weiterhin auf der Webseite **https://www.vbentwickler.de**.

Viel Spaß beim Erkunden der neuen Lernplattform!

Dein André Minhorst

Code-Snippets im SQL Server Management Studio

Es gibt Code-Snippets, die man immer wieder verwendet – sei es, um Tabellen, Views oder gespeicherte Prozeduren zu erstellen, Tabellen oder Indizes zu definieren und vieles mehr. Vielleicht nutzt Du auch verschiedene System-Befehle, um verschiedene Aktionen im Abfragefenster auszulösen, um beispielsweise Informationen zur Datenbank zu ermitteln. Solche Snippets hast Du vielleicht in einer Textdatei gespeichert, um sie jederzeit in das Abfragefenster kopieren zu können. Es gibt jedoch eine viel mächtigere Möglichkeit, solche Code-Snippets zu nutzen: Das Abfragefenster bietet nämlich die Möglichkeit, Code-Snippets per Kontextmenü einzufügen. Das Beste daran ist: Wir können sogar eigene Code-Snippets definieren, die darüber ausgewählt und eingefügt werden können. In diesem Artikel schauen wir uns an, wie wir diese Funktion nutzen können und wie wir sogar eigene Code-Snippets in diesem Menü verfügbar machen können.

Eingebaute Code-Snippets nutzen

Als Erstes schauen wir uns an, wie die eingebauten Code-Snippets genutzt werden können.

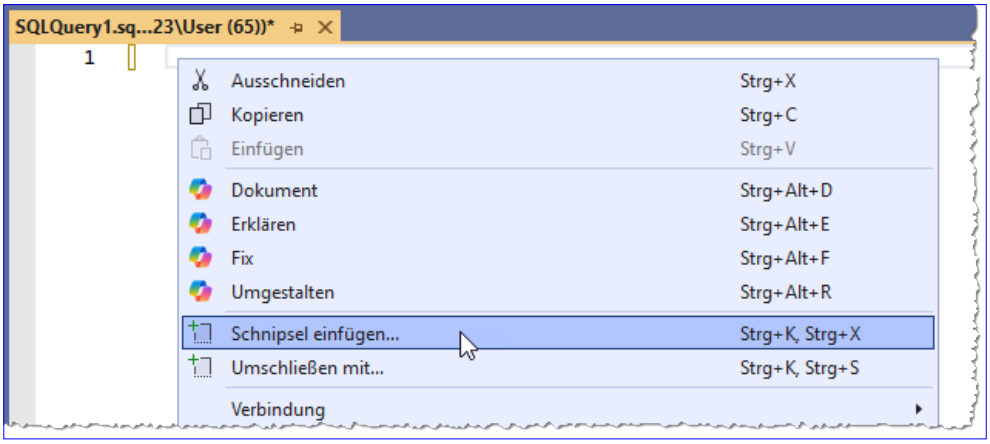

Bild 1: Schnipsel einfügen per Kontextmenü im Abfragefenster

Dazu öffnen wir ein beliebiges Abfragefenster, am besten im Kontext der Datenbank, in der wir den Code-Schnipsel nutzen wollen.

Dazu markieren wir die Datenbank und öffnen dann mit der Tastenkombination **Strg + N** ein neues Abfragefenster. Hier können wir das Kontextmenü öffnen und finden den Eintrag **Schnipsel einfügen...** vor. Alternativ können wir die Tastenkombination **Strg + K, Strg + X** nutzen (siehe Bild 1).

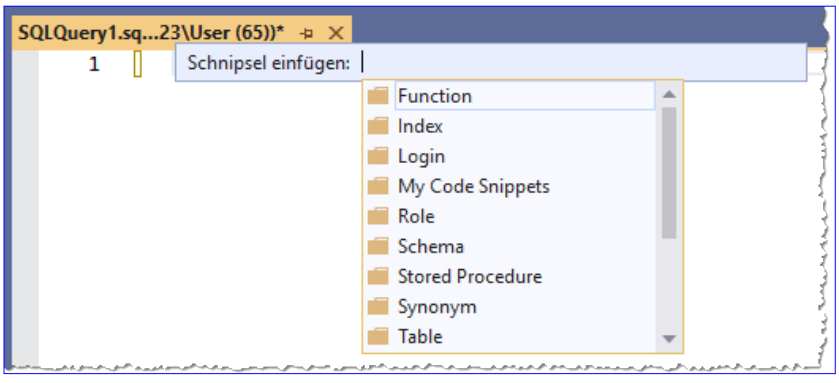

Bild 2: Auswahl der Kategorien für die Code-Schnipsel

Dies öffnet das Widget aus Bild 2. Hier sehen wir den Befehl **Schnipsel einfügen** und eine Auflistung der verschiedenen Kategorien.

Wählen wir eine aus, zum Beispiel **Stored Procedure**, sehen wir alle in dieser Kategorie verfügbaren Schnip-

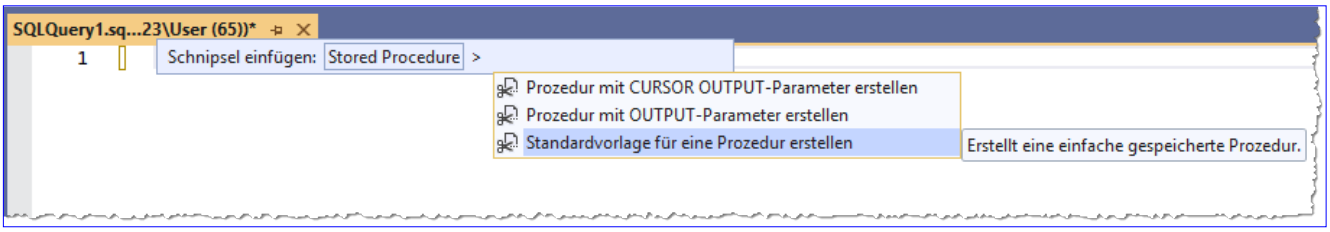

Bild 3: Auswahl der Schnipsel einer Kategorie

sel (siehe Bild 3). Außerdem zeigt dieser Bereich für den aktuell markierten Eintrag noch einen Hilfetext an.

Wir wählen den Eintrag **Standardvorlage für eine Prozedur erstellen** aus und sehen das Ergebnis aus Bild 4. Hier sind verschiedene Texte mit einem gelben Hintergrund markiert. Diese können wir nun mit der Tabulator-Taste durchlaufen. Der aktuelle Eintrag wird markiert, sodass wir diesen direkt durch den gewünschten Text ersetzen können. Wir würden also nun die einzelnen Texte durch die von uns gewünschten Texte einfach überschreiben und erhalten so die gewünschte gespeicherte Prozedur.

Eigene Schnipsel nutzen

Es wird noch besser: Wir können nämlich eigene Schnipsel definieren, die dann ebenfalls im Schnipsel-Menü angezeigt werden.

Erst einmal grundlegende Informationen:

- Es gibt zwei Bereiche, in denen die Schnipsel gespeichert werden. Der erste ist der Bereich mit den eingebauten Schnipseln. Der zweite ist der Bereich für die benutzerdefinierten Schnipsel.

- Wir können in beiden Bereichen eigene Schnipsel hinzufügen. Allerdings raten wir davon ab, Schnipsel zu den eingebauten Schnipseln hinzuzufügen, denn diese werden bei Updates von SQL Server Management Studio gegebenenfalls überschrieben.

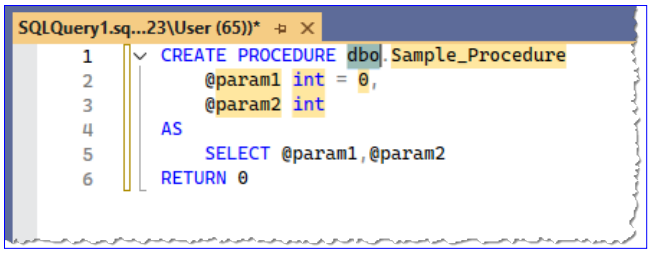

Bild 4: Ein frisch hinzugefügter Schnipsel

- Allerdings können wir die eingebauten Schnipsel als Beispielmaterial für eigene Schnipsel verwenden.

- Wenn wir einen eigenen Schnipsel in dem dafür vorgesehenen Bereich angelegt haben, wird dieser erst beim nächsten Öffnen des SQL Server Management Studios angezeigt.

Speicherort für benutzerdefinierte Schnipsel

Die benutzerdefinierten Schnipsel speichern wir in dem folgenden Ordner ab, hier zum Beispiel für die Version 21 von SQL Server Management Studio:

```
C:\Users\User\Documents\SQL Server Management Studio 21\
Code Snippets\SQL\My Code Snippets
```

Hier können wir direkt eigene Schnipsel speichern, aber wir können auch Unterordner erstellen, die dann entsprechend im Schnipsel-Widget angezeigt werden.

Einen ersten eigenen Schnipsel erstellen

Damit kommen wir zum spannenden Teil: Wir legen unseren ersten eigenen Schnipsel an.

Das können wir mit einem komplett neuen Schnipsel machen oder wir nutzen einen der Schnipsel als Vorlage.

Komplett neuen Schnipsel anlegen

Für einen komplett neuen Schnipsel erstellen wir eine neue Textdatei und fügen dieser den gewünschten Text hinzu. Das kann eine einfache Anweisung sein, die Du immer wieder benötigst und die Du nicht immer im Internet suchen und einfügen möchtest.

Das folgende Skript zeigt beispielsweise die zuletzt angelegten Backups an:

```
SELECT
    database_name,
    backup_start_date,
    backup_finish_date,
    backup_size / 1024 / 1024 AS BackupMB,
    physical_device_name
FROM msdb.dbo.backupset b
JOIN msdb.dbo.backupmediafamily m
  ON b.media_set_id = m.media_set_id
WHERE type = 'D'
ORDER BY backup_finish_date DESC;
```

Dieses Skript fügen wir nun in die Grundstruktur ein und speichern sie im oben angegebenen Ordner unter dem Namen **LetzteBackups.snippet**. Die Grundstruktur sieht wie folgt aus:

```
<CodeSnippets>
  <CodeSnippet Format="1.0.0">
    <Header>
      <Title>Letzte Backups anzeigen</Title>
      <Shortcut>lastbackups</Shortcut>
      <Description>Zeigt die zuletzt angefertigten Backups
an.</Description>
      <Author>André Minhorst</Author>
    </Header>
    <Snippet>
      <Code Language="SQL">
        <![CDATA[
SELECT
    database_name,
    backup_start_date,
    backup_finish_date,
    backup_size / 1024 / 1024 AS BackupMB,
    physical_device_name
FROM msdb.dbo.backupset b
JOIN msdb.dbo.backupmediafamily m
  ON b.media_set_id = m.media_set_id
WHERE type = 'D'
ORDER BY backup_finish_date DESC;
        ]]>
      </Code>
    </Snippet>
  </CodeSnippet>
</CodeSnippets>
```

Hier sehen wir einige Einstellungen wie den Titel, einen Shortcut, eine Beschreibung und den Autor des Schnipsels. Zwischen den Elementen **<![CDATA[** und]]> platzieren wir unseren T-SQL-Code.

Öffnen wir nun das SQL Server Management Studio erneut und wählen die Auswahl der Schnipsel aus, finden wir bereits unseren selbst angelegten Schnipsel vor (siehe Bild 5).

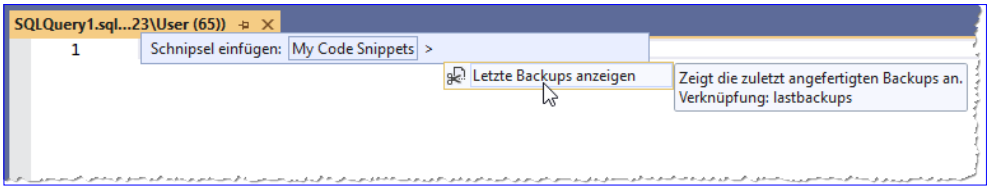

Bild 5: Unser neuer Schnipsel wird bereits zur Auswahl angeboten.

Wählen wir diesen aus, erscheint der von uns hinterlegte Code im Abfragefenster und kann direkt

Access-Datenbank zum SQL Server migrieren

Für die Migration der Tabellen einer Access-Datenbanken in eine SQL Server-Datenbank erledigt man am einfachsten mit einem von Microsoft bereitgestellten Tool namens SQL Server Migration Assistant. Diesem übergeben wir den Namen der zu migrierenden Datenbank, wählen die Tabellen und Abfragen aus, die zum SQL Server übertragen werden sollen und starten dann die Migration. Dies überträgt legt eine neue Datenbank im SQL Server an und überträgt die gewählten Tabellen und Abfragen von Access zum SQL Server. Mit dem SQL Server Migration Assistant können wir außerdem direkt Tabellenverknüpfungen zu den neu erstellten Tabellen in der Access-Datenbank anlegen, sodass wir grundsätzlich direkt mit der Access-Anwendung weiterarbeiten können – mit dem Unterschied, dass die Daten nun nicht mehr aus den Access-Tabellen kommen, sondern vom SQL Server. In diesem Artikel zeigen wir die grundlegende Verwendung des SQL Server Migration Assistants, wobei wir erst einmal eine Datenbank verwenden, deren Tabellen und Felder sich ohne größere Probleme zum SQL Server übertragen lassen.

SQL Server Migration Assistant herunterladen und installieren

Die jeweils aktuelle Version des SQL Server Migration Assistant finden wir, wenn wir im Internet nach genau diesen Schlüsselwörtern suchen.

Wir landen dann beispielsweise auf einer Seite wie der aus Bild 1. Hier klicken wir nicht etwa auf **SQL Server Migration Assistant for Access**, sondern scrollen weiter nach unten, wo wir unter **Downloads** den Eintrag **SSMA for Access** finden.

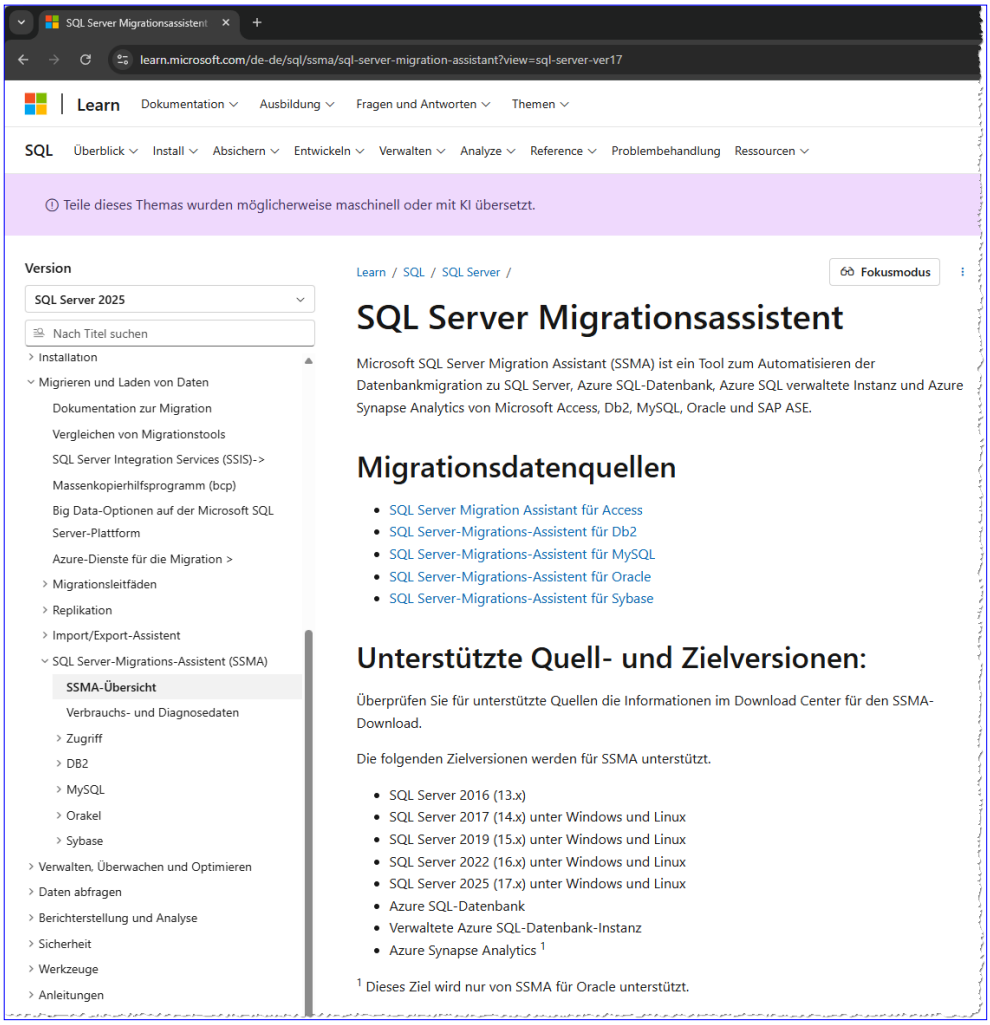

Bild 1: Download des SQL Server Migration Assistant

Auf der nun erscheinenden Seite klicken wir auf **Download**. Die Sprache können wir nicht ändern, da der SSMA nur in Englisch verfügbar ist.

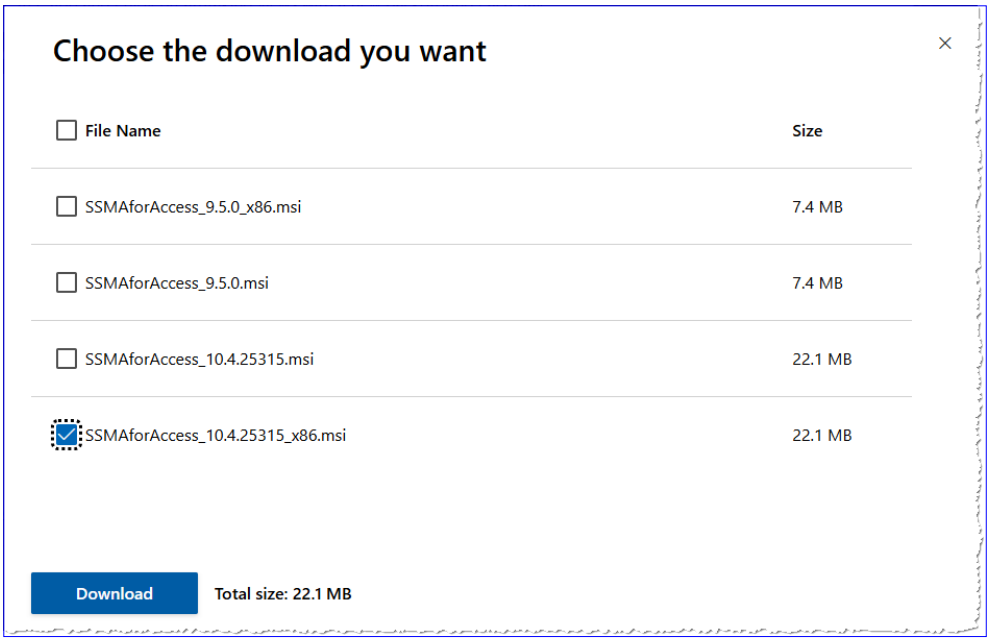

Bild 2: Auswahl der richtigen Version

Richtige Bitness auswählen

Es erscheint ein Popup, das die verschiedenen Versionen auflistet (siehe Bild 2). Hier gibt es zum Beispiel zwei verfügbare Versionen (**9.5** und **10.4**), die jeweils für 32-Bit und 64-Bit verfügbar sind.

Hier sollten wir die aktuellere Version wählen. Viel wichtiger ist jedoch, die richtige Bitness zu selektieren.

Ob wir die 32-Bit-Version (erkennbar am Zusatz **x86**) oder die 64-Bit-Version wählen (ohne Zusatz), hängt nicht etwa von der Bitness des installierten Betriebssystems zusammen, sondern von der Bitness der Access-Installation.

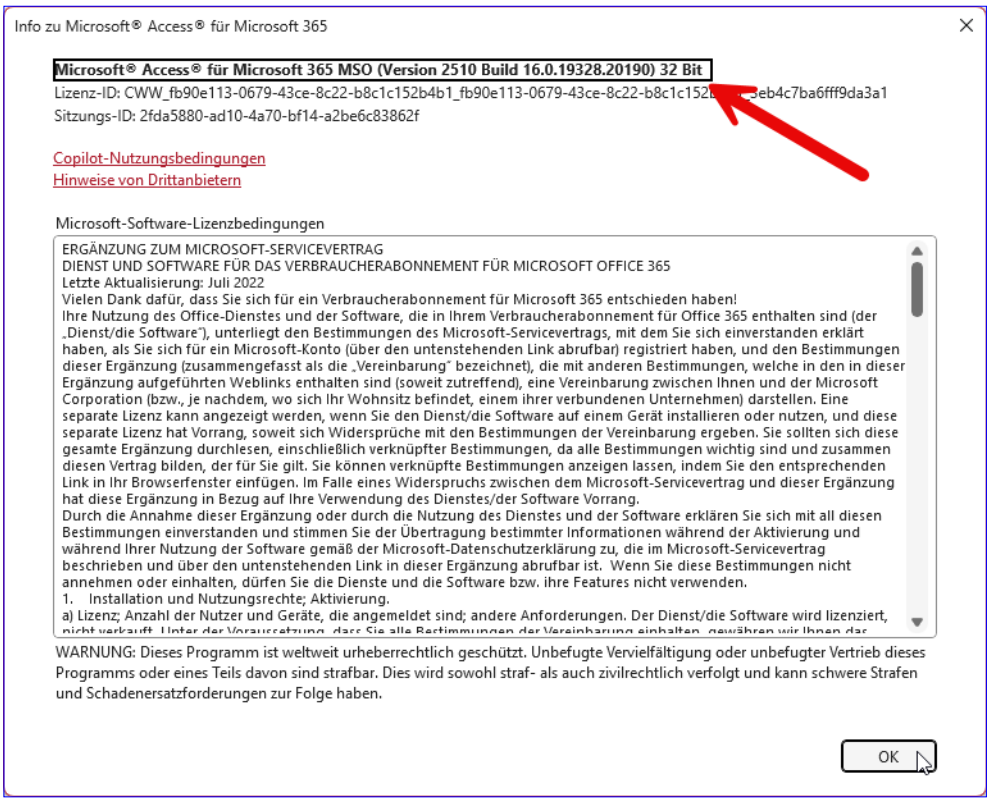

Bild 3: Ermitteln der Bitness der Access-Installation

Zur Sicherheit prüfen wir also, ob unsere Access-Version in der 32-Bit- oder in der 64-Bit-Version installiert ist. Das können wir beispielsweise über die Benutzeroberfläche von Access herausfinden. Dazu klicken wir in neueren Access-Versionen im Ribbon auf den Reiter **Datei** und im nun erscheinenden Bereich links auf **Konto**.

Rechts sehen wir nun eine Schaltfläche namens **Info**

zu Access. Damit öffnen wir den Dialog aus Bild 3. Oben sehen wir nun entweder den Text **32 Bit** oder **64 Bit**. In diesem Fall haben wir es mit einem 32-Bit-Access zu tun, also installieren wir den SQL Server Migration Assistant mit dem Zusatz **x86**.

Migration vorbereiten

Für die erste Migration könnte man eine ganze Reihe von Bedingungen direkt in den Tabellen der Access-Datenbank prüfen – ob die Beziehungen korrekt gesetzt sind, nur gültige Namen für Tabellen, Felder, Indizes und so weiter gesetzt sind und ob nur Datentypen verwendet werden, die der SQL Server auch unterstützt.

Wir können aber auch einfach eine Migration starten und abwarten, welche Fehler, Warnungen und Hinweise uns der SQL Server Migration Assistant liefert. Diese können wir dann in Access korrigieren und eine erneute Migration starten.

Wenn wir bei der Migration gleich Tabellenverknüpfungen zur Access-Datenbank hinzufügen wollen, sparen wir uns eine Menge nachträglicher Arbeit. Dies erledigt folgende Aufgaben:

- Die vorhandenen lokalen Tabellen werden nach einem bestimmten Schema umbenannt.
- Es werden Tabellenverknüpfungen zu den migrierten Tabellen angelegt.

In der Regel werden wir nach der initialen Migration allerdings Fehler, Warnungen und Hinweise auf Probleme erhalten, die wir gegebenenfalls erst in der originalen Access-Datenbank anpassen wollen, um anschließend in einer weiteren Migration die Tabellen mit weniger oder möglichst sogar ohne Fehlermeldungen zum SQL Server zu übertragen.

Dazu ein kleiner Vorgriff: Wir könnten dann auf die Idee kommen, die vom SQL Server Migration Assistant zur Access-Datenbank hinzugefügten Tabellenverknüpfungen einfach zu löschen und den umbenannten lokalen Tabellen einfach wieder den Originalnamen zuzuweisen.

Wenn wir diese Datenbank dann erneut mit dem SQL Server Migration Assistant migrieren wollen, wird dieser aber überraschenderweise keine Tabellen in der Access-Datenbank mehr finden, die sich migrieren lassen.

Der Grund dafür ist, dass der SSMA die Originaltabellen nicht nur umbenennt, sondern auch verschiedene Eigenschaften einstellt, mit denen der SSMA bei erneuter Migration erkennen kann, welche Tabellen bereits migriert wurden.

Wir könnten nun zwar per VBA-Code nicht nur die bereits migrierten Tabellen wieder mit den Originalnamen versehen, sondern auch die vom SSMA hinzugefügten Eigenschaften löschen, damit diese erneut migriert werden können.

Es ist aber wesentlich einfacher, vor der Migration eine Kopie der zu migrierenden Datenbank zu erstellen. Wenn wir dann nach der ersten Migration feststellen, dass wir noch Änderungen an den Tabellen der Access-Datenbank vornehmen wollen, damit diese im zweiten Durchlauf ohne Fehler, Warnungen und Hinweise migriert werden kann, können wir einfach die bereits migrierte Version verwerfen und mit der Kopie erneut starten.

Migrieren der Access-Datenbank

Damit starten wir die erste Migration. Die Beispieldatenbank haben wir so gestaltet, dass zumindest keine Fehler bei der Migration gemeldet werden. Welche Fehler, Warnungen und Hinweise bei der Migration auftreten können und wie wir diese beheben, werden wir uns aus Platzgründen ohnehin nicht in diesem Artikel ansehen.

Wenn wir den SQL Server Migration Assistant gestartet haben, zeigt dieser standardmäßig gleich den **Migration Wizard** an (siehe Bild 4), der uns die verschiedenen Schritte der Migration vorstellt. Hier können wir außerdem festlegen, ob der Wizard beim nächsten Start des SQL Server Migration Assistants erneut aufgerufen werden soll. Diese Einstellung behalten wir zunächst bei. Wir können sie aber auch deaktivieren und den Wizard bei späteren Starts des SSMA über den Menüpunkt **File|Migration Wizard** manuell aufrufen.

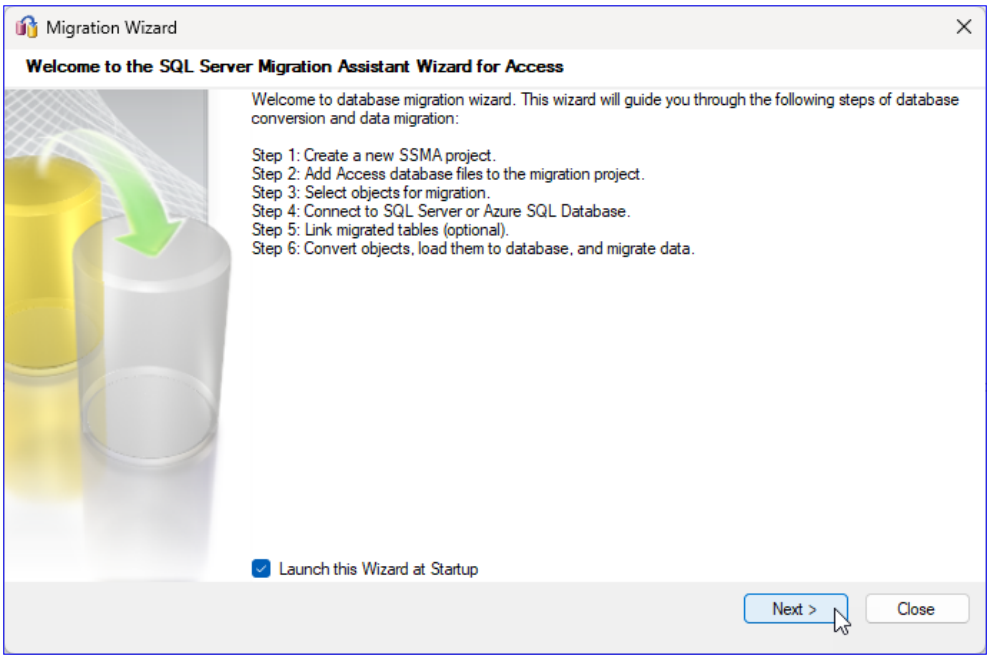

Bild 4: Start des SQL Server Migration Assistants

Im zweiten Schritt definieren wir die Daten des Migrationsprojekts. Hier geben wir einen Namen an und können den Pfad der zu speichernden Projektdatei festlegen (siehe Bild 5).

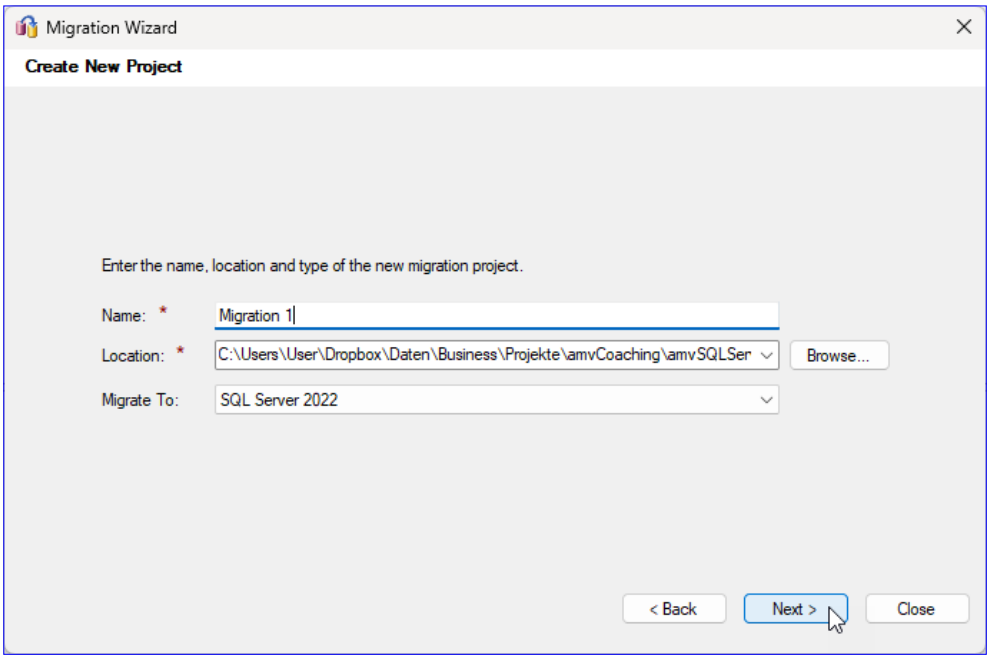

Bild 5: Angabe eines Projektnamens

Außerdem legen wir hier fest, zu welcher SQL Server-Version wir die Access-Datenbank migrieren wollen – hier **SQL Server 2022**.

SQL Server-Version ermitteln

Wenn Du nicht sicher bist, welche SQL Server-Version Du verwendest, kannst Du das im SQL Server Management Studio herausfinden. Dazu klickst Du im Objekt-Explorer mit der rechten Maustaste auf das oberste Element für die aktuelle Verbindung und wählst im Kontextmenü den Eintrag **Neue Abfrage** aus.

Hier gibst Du den folgenden Befehl ein und führst diesen mit der Taste **F5** aus:

```
SELECT @@Version
```

Dies liefert das Ergebnis aus Bild 6 – in diesem Fall wird also der SQL Server in der Version 2022 verwendet, die wir auch im SSMA auswählen sollten.

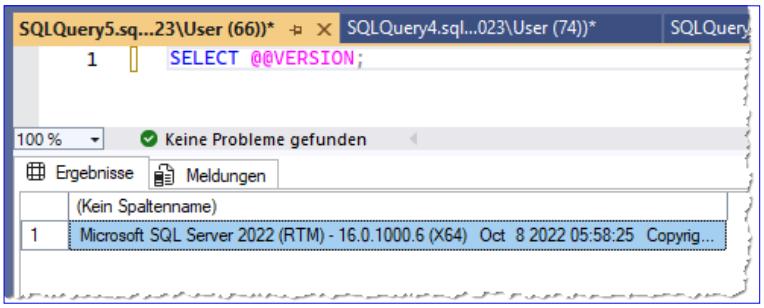

Bild 6: Ermitteln der SQL Server-Version

Im SSMA klicken wir nun auf **Next** und landen im nächsten Schritt, in dem wir über die Schaltfläche **Add Databases** die Access-Datenbank auswählen, deren Tabellen wir zum SQL Server migrieren wollen. Diese erscheint anschließend in der Liste der hinzuzufügenden Access-Datenbanken.

Hier ist zu erkennen, dass wir durchaus auch die Tabellen mehrerer Access-Datenbanken gleichzeitig in eine SQL Server-Datenbank migrieren können. Das ist zum Beispiel sinnvoll, wenn wir ein Frontend nutzen, das mit den Tabellen aus mehreren Access-Backends verknüpft ist, die alle in einer neuen SQL Server-Datenbank landen sollen. In diesem Fall wollen wir jedoch nur eine Access-Datenbank migrieren.

Lokale und verknüpfte Access-Tabellen

An dieser Stelle kommt oft die Frage auf, wie man mit dem Fall umgeht, dass eine Frontend-Datenbank verwendet wird, die ihre Daten über Tabellenverknüpfungen zu einer weiteren, als Backend verwendeten Access-Datenbank bezieht.

Hier hat man zwei Möglichkeiten:

- Man gibt die Frontend-Datenbank als Quelle für die Migration an, wodurch man sowohl lokale Tabellen als auch verknüpfte Tabellen migrieren kann.

- Oder man gibt die Backend-Datenbank als Quelle für die Migration an, wodurch nur die Tabellen aus der Backend-Datenbank zur Migration herangezogen werden können.

Welche Variante man wählt, hängt in erster Linie davon ab, ob der SSMA bei der Migration der Access-Datenbank direkt Tabellenverknüpfungen zu den neuen Tabellen aus der SQL Server-Datenbank anlegen soll.

Wenn wir die Variante wählen, bei der wir das Backend als Quelle für die Tabellen angeben, werden die Tabellenverknüpfungen auch automatisch in der Backenddatenbank angelegt. Das ist weniger sinnvoll, da wir diese ja im Frontend benötigen.

Wir würden also an dieser Stelle normalerweise die Frontenddatenbank mit dem Tabellenverknüpfungen zur Backenddatenbank auswählen.

Der SQL Server Migration Assistant erkennt dies automatisch und migriert dann die verknüpften Tabellen aus dem Backend. Wenn wir die Option zum automatischen Erstellen von Tabellenverknüpfungen zu den Tabellen in der zu erstellenden SQL Server-Datenbank wählen, benennt der SSMA die Tabellenverknüpfungen zu den Tabellen des Access-Backends um und ersetze diese durch die Verknüpfungen zu den Tabellen der SQL Server-Datenbank.

Tabellen für die Migration auswählen

Im SQL Server Migration Assistant gehen wir nun zum nächsten Schritt, in dem wir die zu migrierenden Objekte der Access-Datenbank auswählen können (siehe

SQL Server: Tabellen per VBA verknüpfen

Wenn Du die Tabellen einer Access-Datenbank mit dem SQL Server Migration Assistant zum SQL Server migriert hast, ist der Großteil einer SQL Server-Migration bereits geschafft. Allerdings stehen noch weitere Arbeiten wie das Anpassen des VBA-Codes, Abfragen, Formularen und Berichten bevor. Eine andere, wichtige Aufgabe ist das Sicherstellen der Funktion der Tabellenverknüpfungen. Diese wurden, wenn die richtige Option im SQL Server Migration Assistant markiert wurde, bereits initial angelegt, während die Original-Tabellen der Access-Datenbank umbenannt wurden. Wie aber stellen wir sicher, dass die Tabellenverknüpfungen auch nach dem Ändern des Tabellenentwurfs im SQL Server aktuell bleiben? Das erläutern wir in diesem Artikel.

Beispieldatenbank

Im Artikel **Access-Datenbank zum SQL Server migrieren** (**www.vbentwickler.de/484**) zeigen wir, wie Du eine Access-Datenbank zum SQL Server migrierst. Die dort erstellt SQL Server-Datenbank nutzen wir als Beispielmaterial für die folgenden Abschnitte. Wir gehen zum Start davon aus, dass wie eine leere Access-Datenbank haben, in der wir Tabellenverknüpfungen auf Basis dieser SQL Server-Datenbank erstellen wollen.

Warum Tabellen per VBA verknüpfen?

Bevor wir uns ans Werk machen, stellt sich die Frage: Warum sollte ich überhaupt meine Access-Datenbank überhaupt per VBA mit Tabellen aus dem SQL Server verknüpfen?

Wenn man eine Migration mit dem SQL Server Management Studio durchführe, werden ja bereits automatisch Tabellenverknüpfung hinzugefügt, und außerdem gibt es doch in Access ausreichend Möglichkeiten, Tabellen über die Benutzeroberfläche zu verknüpfen. Dazu gehört beispielsweise der Assistent, den wir über den Ribbon-Eintrag **Externe Daten|Aus Datenbank|Aus SQL Server** öffnen (siehe Bild 1).

Alternativ können wir auch den Befehl **Externe Daten|Neue Datenquelle|Aus anderen Quellen|ODBC-Datenbank** nutzen (siehe Bild 2).

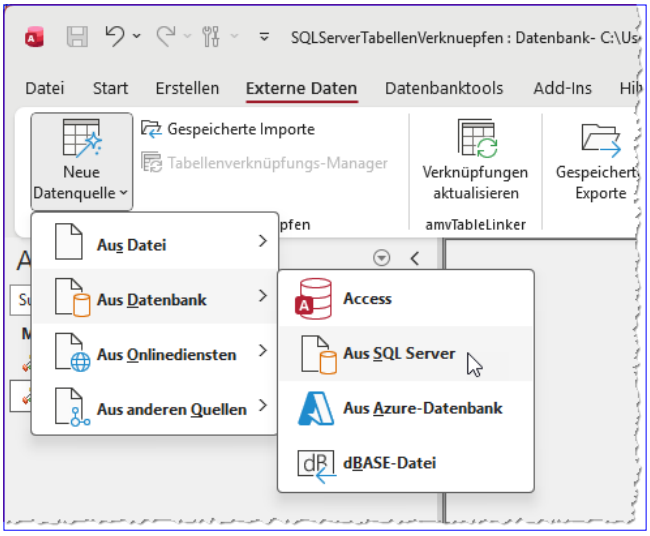

Bild 1: Öffnen des Assistenten zum Verknüpfen oder Importieren von SQL Server-Tabellen

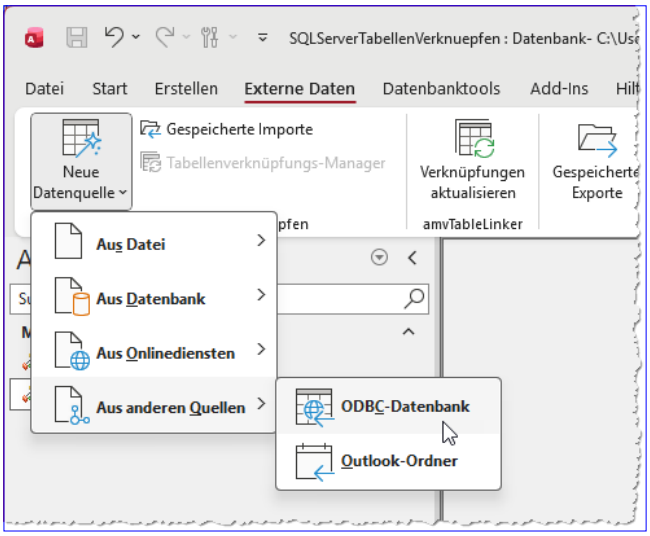

Bild 2: Öffnen des Assistenten für ODBC-Datenbanken

Beide führen dazu, dass wir eine Datenquelle erstellen, die auf einem Treiber besteht – der in beiden Fällen optimalerweise der aktuelle ODBC-Treiber für die Verbindung mit Microsoft SQL Server ist.

In beiden Fällen können wir Tabellenverknüpfungen erstellen, die funktionieren und die je nach den Einstellungen, die wir dort vornehmen, zum Beispiel wie folgt aussehen:

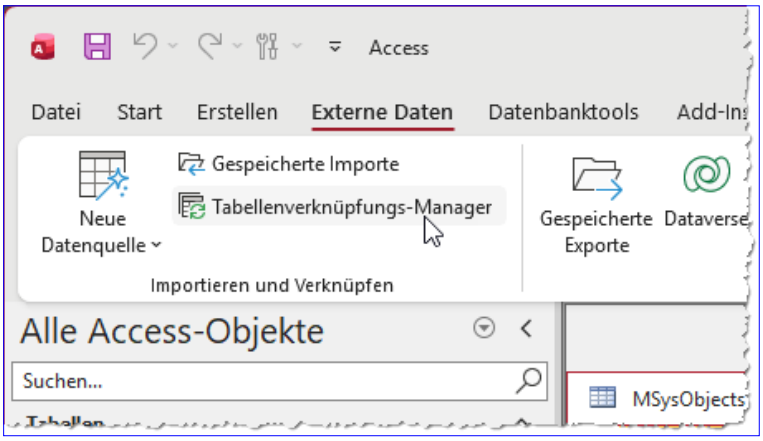

Bild 3: Öffnen des Tabellenverknüpfungs-Managers

```
Description=AccessSQLServer;DRIVER=ODBC Driver
18 for SQL Server;SERVER=amvDesktop2023;Trusted_Connec-
tion=Yes;APP=Microsoft Office;DATABASE=SQLServerTabellen-
verknuepfen;TrustServerCertificate=Yes;
```

Tabellenverknüpfungen verwalten

Wenn die Tabellen einmal verknüpft sind, können wir diese sogar über die Benutzeroberfläche verwalten.

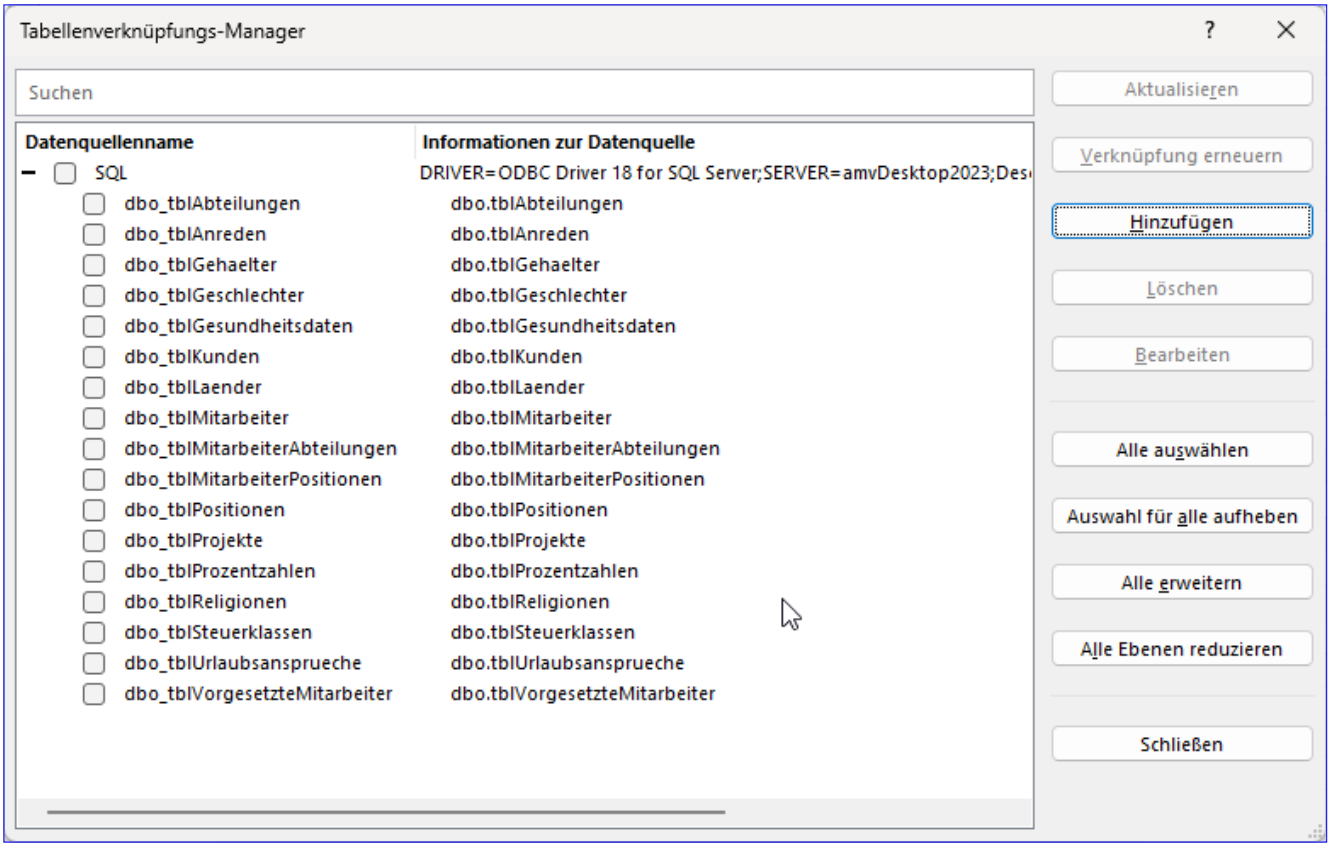

Bild 4: Der Tabellenverknüpfungs-Managers

Dazu bietet sich zunächst der **Tabellenverknüpfungs-Manager** an, den wir über den Ribbon-Eintrag aus Bild 3 öffnen.

Der Tabellenverknüpfungs-Manager erscheint anschließend wie in Bild 4.

Hier sehen wir für jede Datenquelle einen Haupteintrag, den wir öffnen können. Danach erscheinen alle Tabellen, die aus dieser Datenquelle stammen.

Wir können alle Tabellen markieren, indem wir einen Haken für den Haupteintrag setzen, oder auch einzelne Einträge markieren.

Dies aktiviert die beiden Schaltflächen **Aktualisieren** und **Verknüpfung erneuern**. Wenn wir auf **Aktualisieren** klicken, wird die Verknüpfung aktualisiert, wobei alle Änderungen, die wir zwischenzeitlich an der markierten Tabelle im SQL Server vorgenommen haben, an die Tabellenverknüpfung übertragen werden.

Wenn wir also beispielsweise ein neues Feld zur Tabelle **tblAbteilungen** hinzufügen und diesen Befehl betätigen, sehen wir beim nächsten Öffnen der Tabellenverknüpfung in Access das neue Feld.

Ähnlich funktioniert die Schaltfläche **Verknüpfung erneuern**. Wenn wir nur eine Tabelle neu verknüpfen wollen, erscheint der Dialog aus Bild 5, mit dem wir die bereits verknüpfte Tabelle übernehmen oder einen neuen Namen eingeben können.

Wenn wir jedoch die gesamte Datenquelle markieren, erscheint der Dialog aus Bild 6. Damit können wir also auch die Verbindungszeichenfolge bearbeiten. Danach wird der Dialog zum Verknüpfen einer Tabelle für alle

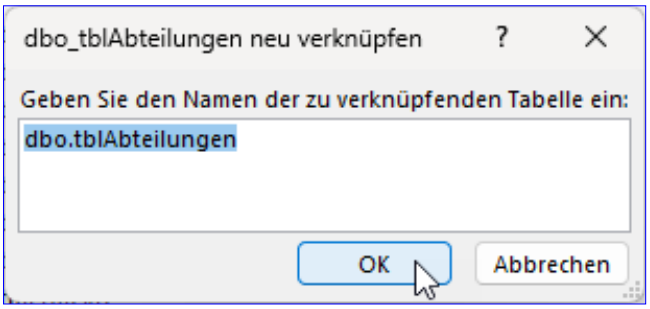

Bild 5: Neuverknüpfung einer Tabelle

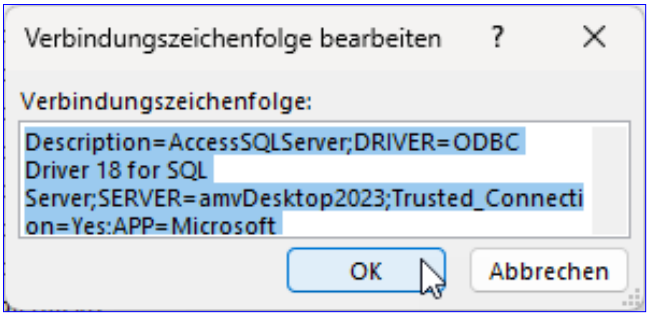

Bild 6: Neuverknüpfung einer Datenquelle

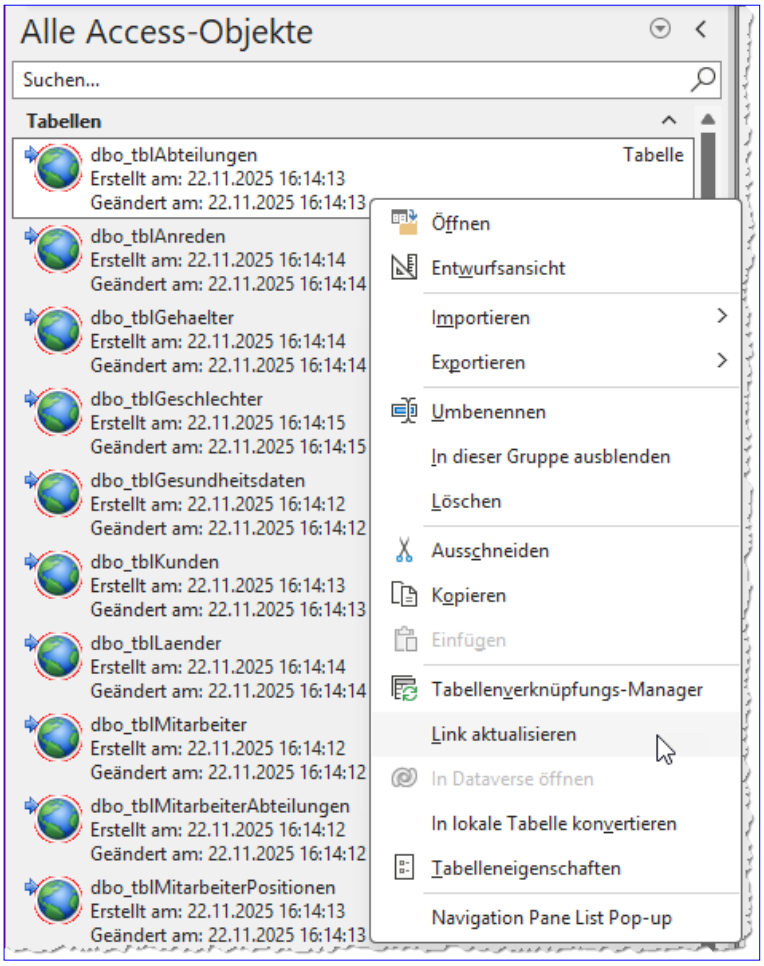

Bild 7: Aktualisieren einer Datenquelle

Tabellenverknüpfungen dieser Datenquelle jeweils einmal angezeigt.

Verknüpfungen über den Navigationsbereich aktualisieren

Die Funktion zum Aktualisieren einer Tabellenverknüpfung können wir auch schneller aufrufen. Dazu klicken wir mit der rechten Maustaste auf die Tabellenverknüpfung im Navigationsbereich und wählen dort den Befehl **Link aktualisieren** aus (siehe Bild 7).

Direkt darüber sehen wir noch eine weitere Möglichkeit, den Tabellenverknüpfungs-Manager zu öffnen.

Außerdem können wir eine Tabellenverknüpfung über das Kontextmenü in eine lokale Tabelle konvertieren.

Das ist hilfreich, wenn man zum Beispiel eine Datenbank an einen anderen Entwickler übergeben möchte, wenn dieser Arbeiten an der Datenbank durchführen soll, man aber nicht die SQL Server-Datenbank mitliefern möchte.

Wenn es solche Möglichkeiten gibt, warum sollten wir dann also das Aktualisieren oder Erneuern der Tabellenverknüpfungen per VBA programmieren?

Dazu gibt es verschiedene Gründe:

- Zum Aktualisieren oder Neuverknüpfen sind immer mehrere Klicks erforderlich. Das kostet Zeit und ist fehleranfällig, weil man schnell eine falsche Verbindungszeichenfolge oder einen falschen Tabellennamen eingegeben hat.

- In der Regel entwickeln wir auf dem eigenen Rechner. Gegebenenfalls wechseln wir dabei die SQL Server-Datenbank oder den SQL Server. Wenn wir das im Entwicklungsmodus noch über die Benutzeroberfläche machen, erhöht dies lediglich den oben beschriebenen Aufwand.

- Spätestens wenn wir die Anwendung in den Produktivbetrieb übernehmen, müssen wir sicherstellen, dass die Tabellenverknüpfungen mit der dort zu verwendenden Verbindungszeichenfolge erstellt werden. Auch das könnten wir noch vorbereitend durchführen, aber dann müssten wir immer bereits auf unserem lokalen Entwicklungsrechner die Tabellenverknüpfungen mit genau der Verbindungszeichenfolge ausstatten, die wir auch auf dem Zielsystem vorfinden.

- Ein weiterer Grund ist, dass wir beim Neuverknüpfen von Tabellen über die Benutzeroberfläche für die Tabellenverknüpfungen zum Beispiel der Tabelle **dbo.tblAbteilungen** einen Namen wie **dbo_tblAbteilungen** erhalten. Das heißt, dass wir diesen noch nachträglich anpassen müssen, wenn wir beispielsweise nur den Namen der Tabelle, also **tblAbteilungen**, ohne das vorangestellte Schema erhalten wollen.

Deshalb ist es sinnvoll, einige VBA-Routinen vorzubereiten, mit denen wir die Tabellenverknüpfungen jederzeit per Mausklick entweder aktualisieren oder erneuern können.

Wenn wir dann zum Beispiel auf dem Entwicklungssystem den Server wechseln, brauchen wir einfach nur eine andere Verbindungszeichenkette zu irgendwo in der Datenbank zu speichern, die dann auf dem Zielrechner zum Herstellen der Tabellenverknüpfungen verwendet wird.

Tabellenverknüpfungen aktualisieren bei Windows-Authentifizierung vs. SQL Server-Authentifizierung

Wenn wir die Tabellenverknüpfungen über die Benutzeroberfläche aktualisieren wollen, gibt es einen Unterschied bezüglich der beiden Authentifizierungsmethoden Windows-Authentifizierung und SQL Server-Authentifizierung.

Bei der Windows-Authentifizierung wird bekanntlich das Windows-Konto des aktuellen Benutzers im SQL Server überprüft.

Ist dieses bekannt und die entsprechende Anmeldung im SQL Server hat Berechtigungen für den Zugriff auf die entsprechenden Tabellen, erfolgt die Aktualisierung ohne weitere Interventionen des SQL Servers.

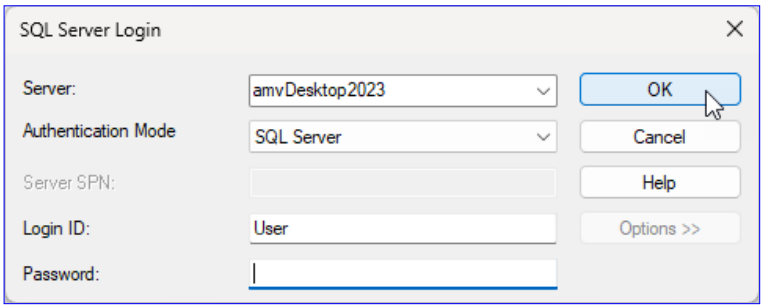

Bild 8: Abfrage der Verbindungsdaten von per SQL Server-Authentifizierung verknüpften Tabellen

Wenn wir jedoch die SQL Server-Authentifizierung verwendet wird, prüft der SQL Server, ob Benutzername und Kennwort der jeweiligen SQL Server-Anmeldungen vorliegen.

Das ist der Fall, wenn man in der laufenden Access-Session bereits einmal die Anmeldedaten für das Aktualisieren oder erneute Anlegen der Tabellenverknüpfungen verwendet hat. Diese werden dann intern gespeichert.

Sobald die Session jedoch geschlossen ist, also die laufende Access-Anwendung mit den zwischengespeicherten Anmeldedaten beendet wurde, liegen die Anmeldedaten nicht mehr im Speicher.

Öffnet man die Access-Anwendung mit den verknüpften Tabellen erneut und versucht, eine der Tabellen zu öffnen, erscheint die Meldung aus Bild 8.

Wenn wir die Daten einmal eingegeben haben, können wir alle Tabellen, die über die gleiche Verbindungszeichenfolge verknüpft sind, wieder öffnen.

Nachfolgend beschreiben wir, wie man die Verknüpfung zu einer einzelnen Tabelle per VBA aktualisiert und wie man alle Tabellenverknüpfungen erneuert.

Bei Verwendung der SQL Server-Authentifizierung erscheint daher auch hier die Meldung zur Eingabe der Verbindungsdaten. Eine automatische Aktualisierung der Tabellenverknüpfungen ist also nur möglich, wenn wir die Windows-Authentifizierung für den Zugriff auf die Tabellen des SQL Servers verwenden.

Wenn wir die SQL Server-Authentifizierung nutzen, können wir die Tabellen nicht einfach mit den nachfolgend beschriebenen VBA-Prozeduren aktualisieren, sondern müssen zumindest eine Tabellenverknüpfung löschen und erneut anlegen. Dadurch sind anschließend aber auch alle anderen Tabellen mit der gleichen Verbindung wieder zugreifbar.

Hinweis: Früher war es möglich, den Benutzernamen und das Kennwort in der Verbindungszeichenfolge für eine Tabelle dauerhaft zu speichern, aber diese Funktion hat Microsoft aus Sicherheitsgründen entfernt. Das ist sinnvoll, denn sonst könnten die Zugangsdaten über die Benutzeroberfläche ausgelesen werden, was potenzielle Sicherheitslücken mit sich bringt.

Tabellenverknüpfung aktualisieren per VBA

Als Erstes schauen wir uns an, wie wir die Tabellenverknüpfungen per VBA aktualisieren können – also so, als ob wir den Befehl **Link aktualisieren** im Kontextmenü einer Tabelle im Navigationsbereich aufrufen.

Wir gehen hier davon aus, dass uns eine frisch mit dem SQL Server Migration Assistant migrierte Datenbankanwendung vorliegt.

Um beispielsweise die Tabellenverknüpfung zur Tabelle **tblAbteilungen** zu aktualisieren, verwenden wir die Methode **RefreshLink** des jeweiligen **TableDef**-Objekts.

Das Aufwendigste daran ist das Referenzieren dieses Objekts. Dazu verwenden wir die folgende Prozedur:

```
Public Sub TabellenverknuepfungAktualisieren()
    Dim db As DAO.Database
    Dim tdf As DAO.TableDef
    Set db = CurrentDb
    Set tdf = db.TableDefs("tblAbteilungen")
    tdf.RefreshLink
    Application.RefreshDatabaseWindow
End Sub
```

Diese füllt die Variable **db** mit einem Verweis auf das aktuelle **Database**-Objekt. Dann referenzieren wir das **TableDef**-Objekt der Tabelle **tblAbteilungen** und rufen seine Methode **RefreshLink** auf. Schließlich aktualisieren wir noch den Navigationsbereich, damit die Änderung direkt sichtbar wird.

Sollten wir in der Zwischenzeit den Entwurf der SQL Server-Tabelle **tblAbteilungen** angepasst haben, werden diese Änderungen nun beim Öffnen der Tabelle in Access direkt sichtbar.

Alle Tabellenverknüpfungen aktualisieren

Eine einzige Tabellenverknüpfung wollen wir meist nur während der Entwicklung aktualisieren, was sich schneller durch das Betätigen des Befehls **Link aktualisieren** des Kontextmenüs des jeweiligen Eintrags im Navigationsbereich von Access realisieren.

Wir wollen vermutlich eher direkt alle Tabellen aktualisieren, zum Beispiel direkt dann, wenn der Benutzer die Anwendung öffnet. Dann können wir die folgende Prozedur aufrufen:

```
Public Sub AlleTabellenverknuepfungenAktualisieren()
    Dim db As DAO.Database
    Dim tdf As DAO.TableDef
    Set db = CurrentDb
    For Each tdf In db.TableDefs
        If Not Len(tdf.Connect) = 0 Then
            If Left(tdf.Connect, 5) = "ODBC;" Then
                Debug.Print tdf.Connect
                tdf.RefreshLink
            End If
        End If
    Next tdf
    Application.RefreshDatabaseWindow
End Sub
```

Die Prozedur durchläuft alle Elemente der **TableDefs**-Auflistung und prüft in zwei **If...Then**-Bedingung, ob die Tabelle erstens einen Wert für die Eigenschaft **Connect** enthält und zweitens, ob dieser mit **ODBC;** beginnt.

Für Tabellen, die per ODBC verknüpft sind, liefert die Eigenschaft **Connect** beispielsweise Werte wie den folgenden:

```
ODBC;Description=AccessSQLServer;DRIVER=ODBC Driver 18
for SQL Server;SERVER=amvDesktop2023;Trusted_Connecti-
on=Yes;APP=Microsoft Office;DATABASE=SQLServerTabellenver-
knuepfen;TrustServerCertificate=Yes;
```

Tabellenverknüpfung mit neuer Verbindungszeichenfolge aktualisieren

Wenn wir die Frontend-Datenbank vom Entwicklungsrechner zum Rechner des Benutzers übertragen, bei dem einer der folgenden Faktoren zutrifft, reicht das reine Aktualisieren mit **RefreshLink** nicht aus:

- der Servername lautet anders,
- der SQL Server-Datenbankname lautet anders,

- es wird nicht die gleiche Authentifizierungsart verwendet, also Windows-Authentifizierung statt SQL Server-Authentifizierung oder umgekehrt oder

- es wird SQL Server-Authentifizierung verwendet, aber mit anderen Benutzerdaten.

In diesem Fall müssen wir vor dem Aufruf der Methode **RefreshLink** noch die neue Verbindungszeichenfolge mit den geänderten Parametern für die Eigenschaft **Connect** festlegen.

Die Prozedur **TabellenverknuepfungAktualisieren** müssen wir dann wie in Listing 1 um die Definition der zu verwendenden Verbindungszeichenfolge erweitern und diese der Eigenschaft **Connect** zuweisen.

Wann Tabellenverknüpfungen aktualisieren und wann löschen und erneuern?

Damit stellt sich nun die Frage, in welchem Szenario wir mit dem Aktualisieren der Tabellenverknüpfungen auskommen und wann wie die Tabellenverknüpfungen löschen und und neu erstellen müssen.

Da wir sogar eine neue Verbindungszeichenfolge beim Aktualisieren der Tabellenverknüpfungen über die **Connect**-Eigenschaft des **TableDef**-Objekts übergeben können, sollte dies für sehr viele Fälle bereits ausreichen.

Es gibt jedoch auch Fälle, wo tatsächlich ein Löschen und Neuerstellen der Tabellenverknüpfungen erforderlich ist:

- wenn sich Tabellennamen geändert haben

- wenn neue Tabellen hinzugekommen sind

- wenn Tabellen gelöscht wurden und verwaiste Tabellenverknüpfungen zurücklassen würden

Für diese Fälle haben wir einen Satz von Prozeduren und Funktionen programmiert, die wir in den folgenden Abschnitten beschreiben.

Besser aktualisieren als löschen und neu erstellen

In Fällen, wo das Löschen und erneute Erstellen der Tabellenverknüpfungen nicht erforderlich ist, sollte man es immer mit dem Aktualisieren der Tabellenverknüpfungen versuchen.

Damit erhalten wir zumindest die folgenden beiden Vorteile:

```
Public Sub TabelleAktualisieren_NeueVerbindungszeichenfolge()
    Dim db As DAO.Database
    Dim tdf As DAO.TableDef
    Dim strODBCVerbindungszeichenfolge As String
    Set db = CurrentDb
    Set tdf = db.TableDefs("tblAbteilungen")
    strODBCVerbindungszeichenfolge = "ODBC;DRIVER=ODBC Driver 18 for SQL Server;SERVER=amvDesktop2023;" _
        & "DATABASE=SQLServerTabellenVerknuepfen;UID=sa;PWD=********;Encrypt=YES;TrustServerCertificate=YES;"
    tdf.Connect = strODBCVerbindungszeichenfolge
    tdf.RefreshLink
    Application.RefreshDatabaseWindow
End Sub
```

Listing 1: Aktualisieren mit neuer Verbindungszeichenfolge

SQL Server-Zugangsdaten sicher speichern

Im Artikel »SQL Server: Tabellen verknüpfen« (www.vbentwickler.de/485) haben wir gezeigt, wie wir die Tabellen eines SQL Servers per ODBC als Tabellenverknüpfung in Access-Datenbanken verfügbar machen können. Dabei gibt es zwei Varianten, um über entsprechende Verbindungszeichenfolgen auf die Tabellen des SQL Servers zuzugreifen: Windows-Authentifizierung und SQL Server-Authentifizierung. Bei der ersten werden die Zugangsdaten über den aktuellen Windows-Benutzer ermittelt. Dieses Kennwort kennt in der Regel nur der Benutzer und die Verknüpfung der Tabellen kann nach seiner Anmeldung am Rechner ohne weitere Interaktion erfolgen. Bei der SQL Server-Authentifizierung jedoch müssen zusätzlich die Zugangsdaten des Benutzers zum SQL Server angegeben werden. Damit der Benutzer diese nicht immer manuell eingeben muss, möchten wir diese sicher speichern – so, dass niemand diese auslesen kann, auch wenn er Zugriff zum Rechner des Benutzers hat. Wie das gelingt, zeigen wir in diesem Artikel.

Wenn ein Benutzer sich an seinen Rechner angemeldet hat und seine Access-Anwendung geöffnet hat, muss er, um auf per ODBC verknüpfte SQL Server-Tabellen zuzugreifen, auf irgendeine Weise seine Zugangsdaten für den SQL Server und die entsprechende Datenbank angeben, damit diese zum Aktualisieren oder Neuanlegen der Tabellenverknüpfungen genutzt werden können. Wie das Aktualisieren oder Neuverknüpfen gelingt, haben wir in aller Ausführlichkeit im Artikel **SQL Server: Tabellen verknüpfen** (**www.vbentwickler.de/485**) beschrieben. Dazu müssen der Benutzername und das Kennwort einmalig angegeben und in die Verbindungszeichenfolge zum Herstellen der Tabellenverknüpfungen eingetragen werden. Nach dem initialen Aktualisieren oder Neuanlegen der Tabellenverknüpfungen ist während der aktiven Access-Session keine weitere Eingabe dieser Daten erforderlich, da diese intern gespeichert werden. Erst wenn der Benutzer diese Access-Session durch Schließen der Anwendung beendet, werden die temporär gespeicherten Zugangsdaten aus dem Speicher gelöscht.

Da jede Eingabe Zeit kostet, wollen wir hier eine Lösung vorstellen, mit der wir die Zugangsdaten an irgendeiner Stelle im System speichern können, damit der Benutzer diese nicht bei jedem Start der Anwendung manuell eingeben muss. Wir gehen an dieser Stelle davon aus, dass der Benutzer sich nach Feierabend von seinem Rechner abmeldet und diesen auch in kurzen Pausen sperrt. Ansonsten könnten andere Benutzer auf den Rechner zugreifen und die Daten der verknüpften Tabellen mühelos auslesen und in eine andere Datenbank kopieren.

Dennoch würde er dazu eine gewisse Zeit benötigen. Sicherstellen wollen wir auf jeden Fall, dass ein Dritter sich keinen Zugang zu den von uns im System des Benutzers gespeicherten Zugangsdaten zum SQL Server zu verschaffen. Dies wäre kritisch, weil ein Dritter diese dann nutzen könnte, um von einem anderen Rechner in aller Ruhe auf die Tabellen der entsprechenden SQL Server-Datenbank zuzugreifen und diese zu kopieren, zu ändern oder zu löschen oder auch neue Daten hinzuzufügen.

Die vorgestellte Lösung soll also nicht nur dazu dienen, dem Benutzer mehr Komfort beim Starten der Access-Anwendung zu bieten, indem er nicht jedes Mal seine SQL Server-Zugangsdaten eingeben muss. Sie soll auch dafür sorgen, dass diese Zugangsdaten so

gespeichert werden, dass kein Dritter diese auslesen kann, auch wenn er uneingeschränkten Zugriff auf den Rechner hat.

Geeignete Speicherorte

Es gibt verschiedene geeignete Speicherorte für solche Zugangsdaten. Der naheliegendste ist eine Optionentabelle in der Access-Anwendung selbst. Der Benutzer müsste einmal seine SQL Server-Zugangsdaten eingeben. Diese würden dann direkt nach der Eingabe in der entsprechenden Tabelle gespeichert werden.

Eine weitere Option ist eine externe Datei – dabei kann es sich um eine simple Textdatei handeln oder auch eine XML- oder eine JSON-Datei, in die wir diese Informationen in strukturierter Form eintragen können.

Schließlich gibt es noch die Registry. Diese bietet im Pfad **HKEY_CURRENT_USER\Software\VB and VBA Program Settings** einen Bereich, der speziell per VBA oder VB leicht beschrieben oder ausgelesen werden kann, ohne dass wir Funktionen der Windows-API bemühen müssen.

All diese Speicherorte haben Vor- und Nachteile gemeinsam. Der Vorteil ist, dass man diese nur auslesen kann, wenn man unter dem entsprechenden Benutzerkonto angemeldet ist. Voraussetzung dafür ist, dass sich die Dateien im jeweiligen Benutzerordner befinden. Bei dem angesprochenen Registry-Bereich ist das automatisch sichergestellt, da sich dieser unter dem Element **HKEY_CURRENT_USER** befindet, der ohnehin nur im Kontext der entsprechenden Benutzeranmeldung auslesbar ist.

Aber wenn der Benutzer einmal seinen Arbeitsplatz verlässt, zum Beispiel um sich einen Kaffee zu holen, ohne sich abzumelden oder den Bildschirm zu sperren, könnte ein Dritter diese Daten ebenfalls mühelos auslesen und anschließend von einem anderen Rechner über diesen Zugang auf den SQL Server zugreifen.

Es gilt also, noch eine weitere Sicherheitsstufe zu integrieren.

Sicheres Speichern der Zugangsdaten

Dies geschieht wiederum in zwei Stufen. Die erste Stufe ist, dass wir die Zugangsdaten nach der Eingabe und vor dem Speichern durch einen entsprechenden Algorithmus verschlüsseln. Dieser Algorithmus wird als Funktion in der Datenbankanwendung hinterlegt.

Nun kann ein Dritter ohne weitere Maßnahmen immer noch das VBA-Projekt der Access-Anwendung öffnen und Einblick in diesen Algorithmus erhalten, um diesen dann auf einem anderen Rechner auszuführen. Hier folgt die zweite Stufe, indem wir sicherstellen, dass Benutzer keine ungesicherte **.accdb**-Version der Access-Anwendung nutzen, sondern nur eine **.accde**-Version, die keinen Einblick in den VBA-Code mehr gewährt. So ist sichergestellt, dass niemand den Verschlüsselungsalgorithmus ermitteln kann.

Schließlich sollten wir nicht nur einfach den Benutzernamen und das Kennwort verschlüsseln, zum Beispiel mit SHA1. Das Problem ist: Wenn jemand Zugriff auf die Registry oder die Datei hat, in der die Zugangsdaten gespeichert sind, kann er einfach alle gängigen Algorithmen zur Entschlüsselung einmal durchgehen und wird so früher oder später auf die richtige Methode stoßen und kann die Zugangsdaten entschlüsseln.

Das Zauberwort an dieser Stelle lautet Pepper. So nennt man eine Zeichenkette, die als zusätzlicher Faktor für die Verschlüsselung verwendet werden kann. Diesen Faktor speichern wir wiederum im VBA-Projekt, das wir durch Umwandeln der Datenbankdatei in eine **.accde** weitgehend unlesbar machen.

Weitgehend deshalb, weil es durchaus Unternehmen gibt, die **.accde**-Dateien entschlüsseln können, so dass auch unsere Pepper-Zeichenkette ausgelesen werden kann. Solche Unternehmen lassen sich aber, wenn sie

im legalen Bereich operieren, vom Auftraggeber bestätigen, dass sie berechtigt sind, das VBA-Projekt dieser Anwendung wieder lesbar zu machen.

Verwenden von DPAPI zum Ver- und Entschlüsseln

In diesem Artikel verwenden wir keine der üblichen Verschlüsselungsmethoden wie SHA1 et cetera, sondern DPAPI. DPAPI steht für Data Protection API und ist ein integrierter Windows-Mechanismus, mit dem Anwendungen Daten einfach und sicher verschlüsseln können, ohne selbst Kryptographie implementieren zu müssen.

Die Besonderheit von DPAPI ist, dass es einen Schlüssel verwendet, der an das Windows-Benutzerkonto oder an den Computer gebunden ist.

Es gibt zwei Modi:

- **USER-MODE**: Hier ist der Schlüssel ist an den aktuell angemeldeten Windows-Benutzer gebunden. Nur dieser Benutzer kann die Daten wieder entschlüsseln. Dazu kann er die API-Funktionen **CryptProtectData** und **CryptUnprotectData** nutzen.

- **MACHINE-MODE**: Hier ist der Schlüssel ist an den Rechner gebunden. Damit kann jedes Programm unter jedem Benutzer kann entschlüsseln, was zum Beispiel für Serverdienste verwendet werden kann.

Für Access-Frontend-Anwendungen ist USER-MODE perfekt. Wenn wir die Funktion **CryptProtectData** nutzen, wird die zu verschlüsselnde Zeichenkette in einen Datenblock gepackt. Windows verschlüsselt diesen Datenblock mit einem Schlüssel, der aus dem Windows-Benutzerpasswort und geheimen Systemschlüsseln abgeleitet wird. Dies liefert ein Byte-Array zurück, den wir dann zum Beispiel in der Registry speichern können.

Wenn wir später mit der Funktion **CrypUnprotectData** entschlüsseln, übergeben wir das verschlüsselte Byte-Array. Windows prüft dann, ob der aktuelle Benutzer der gleiche Benutzer ist, der verschlüsselt hat. Wenn ja, wird der verschlüsselte Text unverschlüsselt wieder ausgegeben. Anderenfalls schlägt die Verschlüsselung fehl.

Code zum Ver- und Entschlüsseln

Um eine Verschlüsselung mit einer Pepper-Zeichenkette zu nutzen, benötigen wir als Erstes eine Konstante, in der wir diese Zeichenkette speichern:

```
Private Const APP_PEPPER As String = "DEIN_GEHEIMER_PEPPER"
```

Hier musst Du Deine eigene Pepper-Zeichenfolge eintragen.

Außerdem benötigen wir die Deklaration zweier API-Funktionen namens **CryptProtectData** und **CryptUnprotectData** und zwei weitere API-Funktionen namens **RtlMoveMemory** und **LocalFree**. Diese finden wir, neben dem Typ **DATA_BLOB**, in Listing 1.

Die Funktion ProtectString

Die Funktion **ProtectString** übernimmt die Aufgabe, unsere Zeichenkette, also zum Beispiel den Benutzernamen oder das Kennwort sicher zu verschlüsseln, bevor sie beispielsweise in der Registry gespeichert wird. Dabei kombiniert die Funktion den in **APP_PEPPER** hinterlegten Pepper mit der Windows-internen DPAPI-Verschlüsselung (siehe Listing 2).

Zunächst prüft die Funktion, ob der übergebene String überhaupt einen Wert enthält. Ist das nicht der Fall, bricht sie sofort ab und liefert einen leeren Rückgabewert.

Danach wird der Pepper aus **APP_PEPPER** vor das eigentliche Klartextpasswort gesetzt. Dieser Pepper wird nicht mitgespeichert und dient als zusätzliche

Schutzschicht, da ein Angreifer ohne Kenntnis dieses Werts den entschlüsselten Text nicht korrekt validieren könnte.

Anschließend wird die zusammengesetzte Zeichenkette (**APP_PEPPER** plus Benutzername/Kennwort) durch die Funktion **StrConv** in ein ANSI-Byte-Array umgewandelt. DPAPI arbeitet intern mit Binärdaten, daher muss der String zunächst in ein Array von Bytes transformiert werden. Dieses Bytearray wird danach in eine **DATA_BLOB**-Struktur übergeben, die lediglich aus der Anzahl der Bytes und einem Zeiger auf das erste Byte besteht. Wichtig ist hierbei, dass die Struktur nicht selbst Daten enthält, sondern direkt auf das vorhandene Array zeigt.

Nun kommt der zentrale Schritt: Die Funktion ruft **CryptProtectData** auf, eine Windows-API-Funktion, die die Daten benutzergebunden verschlüsselt. Das bedeutet, dass nur derselbe Windows-Benutzer die Da-

```
Private Type DATA_BLOB
    cbData As Long
    pbData As LongPtr
End Type

#If VBA7 Then
    Private Declare PtrSafe Function CryptProtectData Lib "crypt32.dll" ( _
        ByRef pDataIn As DATA_BLOB, _
        ByVal szDataDescr As LongPtr, _
        ByVal pOptionalEntropy As LongPtr, _
        ByVal pvReserved As LongPtr, _
        ByVal pPromptStruct As LongPtr, _
        ByVal dwFlags As Long, _
        ByRef pDataOut As DATA_BLOB) As Long

    Private Declare PtrSafe Function CryptUnprotectData Lib "crypt32.dll" ( _
        ByRef pDataIn As DATA_BLOB, _
        ByVal ppszDataDescr As LongPtr, _
        ByVal pOptionalEntropy As LongPtr, _
        ByVal pvReserved As LongPtr, _
        ByVal pPromptStruct As LongPtr, _
        ByVal dwFlags As Long, _
        ByRef pDataOut As DATA_BLOB) As Long

    Private Declare PtrSafe Sub RtlMoveMemory Lib "kernel32" ( _
        ByRef Destination As Any, _
        ByVal Source As LongPtr, _
        ByVal Length As LongPtr)

    Private Declare PtrSafe Function LocalFree Lib "kernel32" ( _
        ByVal hMem As LongPtr) As LongPtr
#End If
```

Listing 1: API-Deklarationen für das Ver- und Entschlüsseln

Dateien im SQL Server speichern mit FileTables

In Access gibt es verschiedene Möglichkeiten, Dateien zu speichern. Diese kann man zum Beispiel in Anlagefeldern ablegen, oder man belässt die eigentlichen Dateien im Dateisystem und notiert in den Tabellen lediglich den Pfad zu den Dateien oder auch nur die Dateinamen. Welche Variante man wählt, hängt letztlich von der Menge und der Größe der Dateien ab, sprich: vom benötigten Speicherplatz. Der ist nämlich in Access-Datenbanken auf zwei Gigabytes begrenzt. In SQL Server-Datenbanken sieht dies ganz anders aus. Selbst bei Verwendung der kostenlosen Express-Edition dürfen die einzelnen Datenbank-Dateien bis zu zehn Gigabytes an Platz auf der Festplatte einnehmen. Auch hier gibt es verschiedene Varianten zum Speichern der Dateien. Die erste ist, die Dateien in varbinary(max)-Feldern zu hinterlegen. Oder man nutzt dazu sogenannte Filetables. Hier werden die Dateien in einem speziell dazu vorgesehenen Bereich der Festplatte abgelegt, aber unter die Kontrolle des SQL Servers gestellt – und gleichzeitig über einen speziellen Typ von Tabelle verwaltet. Wie das gelingt, zeigen wir in diesem Artikel.

Dateien in Access und der Speicherplatz

Eines der großen Probleme von Access-Datenbanken ist der vergleichsweise geringe Speicherplatz. Sicher, wenn wir nur reine Textinformationen oder Zahlen verwalten, kommen wir damit eine Weile aus.

Soll die Datenbank aber auch in Feldern etwa des Typs **OLE-Objekt** oder **Anlage** noch Elemente wie Bilder oder Dateien aufnehmen, geraten wir schnell ans Limit.

Unter Access gibt es dabei die Möglichkeit, nur den Pfad zur jeweiligen Datei zu speichern und diese im Dateisystem zu belassen, aber damit sind einige Einschränkungen verbunden – zum Beispiel müssen wir die Pfadangaben ändern, wenn die Datenbank samt referenzierter Dateien einmal verschoben werden soll und wir müssen uns selbst darum kümmern, dass die Daten immer an Ort und Stelle bleiben und gesichert werden.

Alternativen im SQL Server

Die Möglichkeiten in einer SQL Server-Datenbank sehen ähnlich aus. Das Pendant zum Anlagefeld oder dem OLE-Objekt-Feld sind Felder mit dem Datentyp **varbinary(max)**.

Dies ähnelt allerdings eher dem Datentyp **OLE-Object** in Access – die Dateien werden dort im Binärformat gespeichert, was bedeutet, dass wir diese, wenn wir beispielsweise Bilddateien in einem Bildsteuerelement anzeigen wollen, zuvor erst in das geeignete Format umwandeln müssen.

Praktischer für diesen Fall ist unter Access das Anlagefeld. Darin gespeicherte Bilder lassen sich durch Angabe des jeweiligen Feldes als Steuerelementinhalt direkt im Bildsteuerelement anzeigen.

FileTable: Kombination aus Dateisystem und Tabelle

Daher ist die Verwendung eines sogenannten FileTables unter SQL Server eine geeignete Alternative. Zwar lässt sich ein Bildsteuerelement nicht direkt an ein Feld dieser FileTable binden, aber in der FileTable werden die Pfadangaben zur jeweiligen Datei gespeichert, den wir ebenfalls als Datenquelle von Bildsteuerelementen angeben können.

Gleichzeitig können wir, entsprechende Berechtigungen vorausgesetzt, in einer FileTable gespeicherte Dateien in einem Bildsteuerelement von Access anzeigen.

Daher schauen wir uns in diesem Artikel einmal genau an, wie wir eine FileTable im SQL Server erstellen und nutzen können.

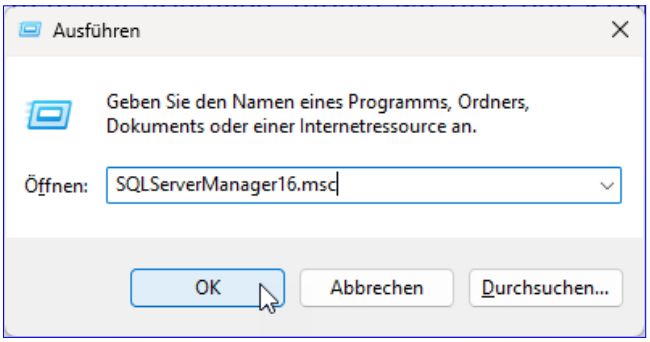

Bild 1: Aufrufen des SQL Server-Konfigurationsmanagers

In weiteren Artikeln betrachten wir, wie die in einer **FileTable**-Tabelle gespeicherte Bilddateien in einem Access-Bildsteuerelement angezeigt werden können.

Voraussetzungen für den Einsatz von FileTables

Die erste und wichtigste Voraussetzung ist die Aktivierung eines speziellen Features des SQL Servers, nämlich **FILESTREAM**.

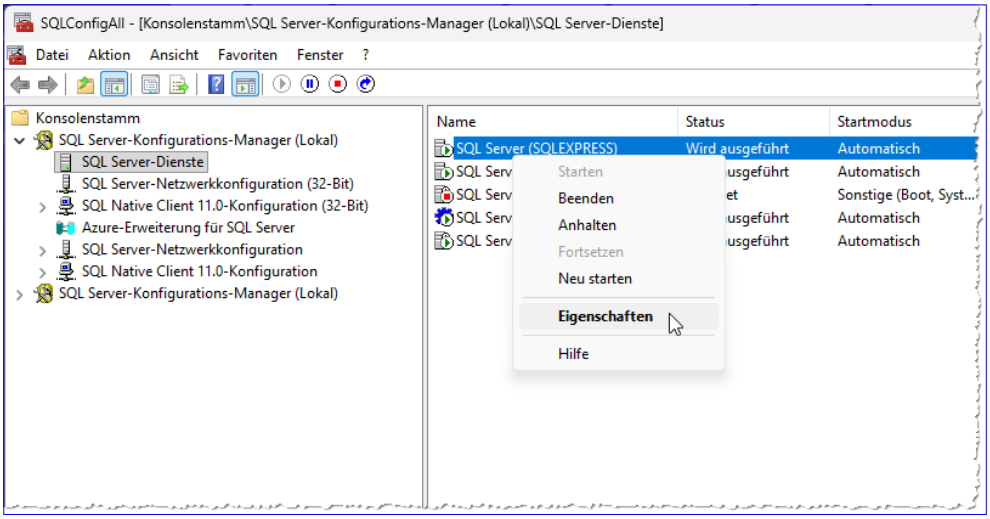

Bild 2: Anzeigen der Eigenschaften der SQL Server-Instanz

Dieses Feature wurde mit SQL Server 2008 eingeführt, **FileTable** gibt es seit 2012. Wenn Du also nicht mit einer vorsintflutlichen Version von SQL Server arbeitest, sollten die Voraussetzungen grundsätzlich erfüllt sein.

FILESTREAM ist standardmäßig nicht aktiviert, wir müssen diese Funktion also erst einmal an den Start bringen.

FILESTREAM aktivieren

Wir beschreiben das Aktivieren von **FILESTREAM** in den folgenden Abschnitten für eine frisch installierte Instanz der SQL Server Express Edition.

Dazu starten wir den SQL Server-Konfigurations-Manager, was zum Beispiel für SQL Server 2022 wie folgt gelingt:

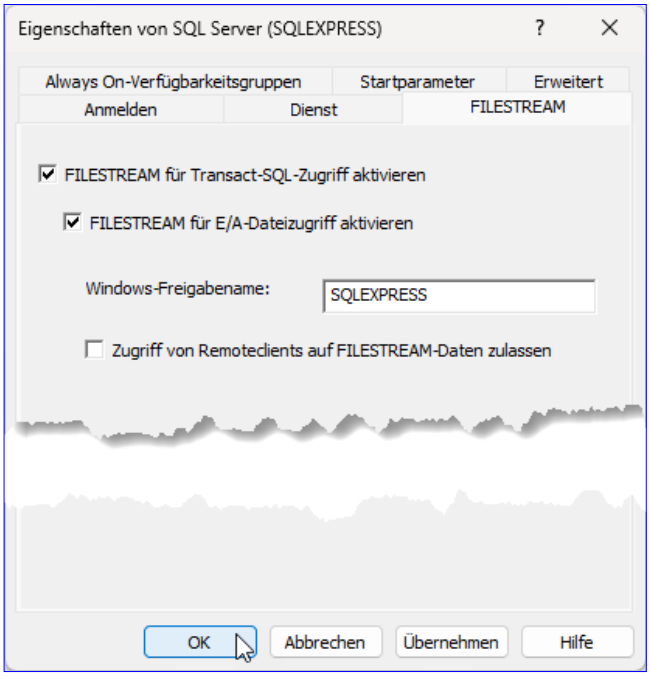

Bild 3: Eigenschaften mit dem **FILESTREAM**-Bereich

- Ausführen-Fenster mit **Windows + R** starten

- Dort geben wir wie in Bild 1 **SQLServerManager16.msc** ein (Versionsnummer entsprechend Deiner Version anpassen).

Danach klicken wir doppelt auf den Eintrag SQL Server-Dienste und dann mit der rechten Maustaste rechts auf die gewünschte Instanz, hier **SQL Server (SQLEXPRESS)** und wählen den Eintrag **Eigenschaften** aus – wobei der in Klammern angegebene Name der Instanzname ist und auch abweichen kann (siehe Bild 2).

Im nun erscheinenden Dialog wechseln wir zur Registerseite **FILESTREAM** (siehe Bild 3).

Dort finden wir folgende Optionen:

- **FILESTREAM für Transact-SQL-Zugriff aktivieren**: Aktiviert den **FILESTREAM** auf SQL Server-Basis.

- **FILESTREAM für E/A-Datenzugriff aktivieren**: Aktiviert **FI-**

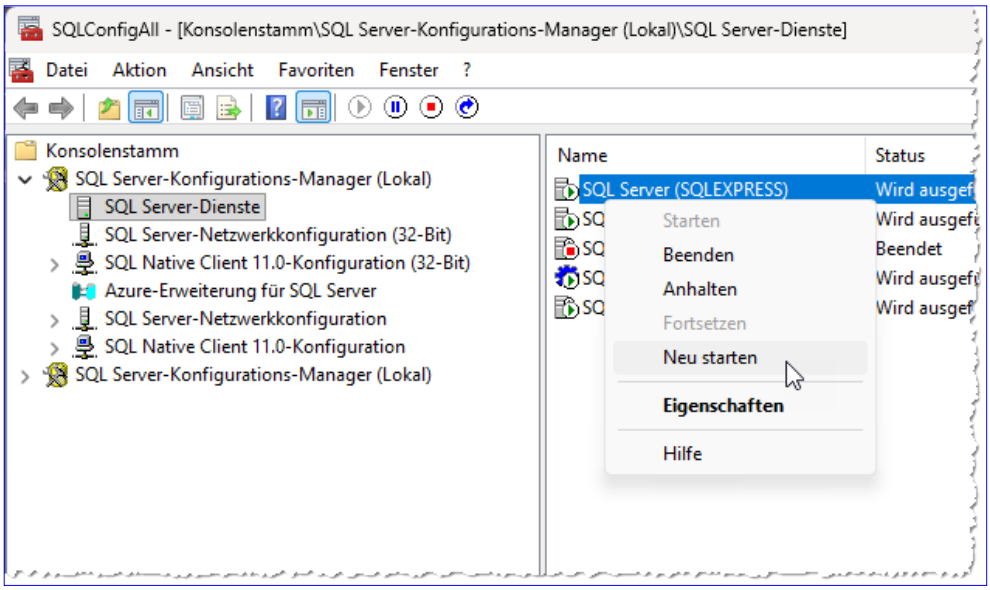

Bild 4: Neustart des SQL Servers

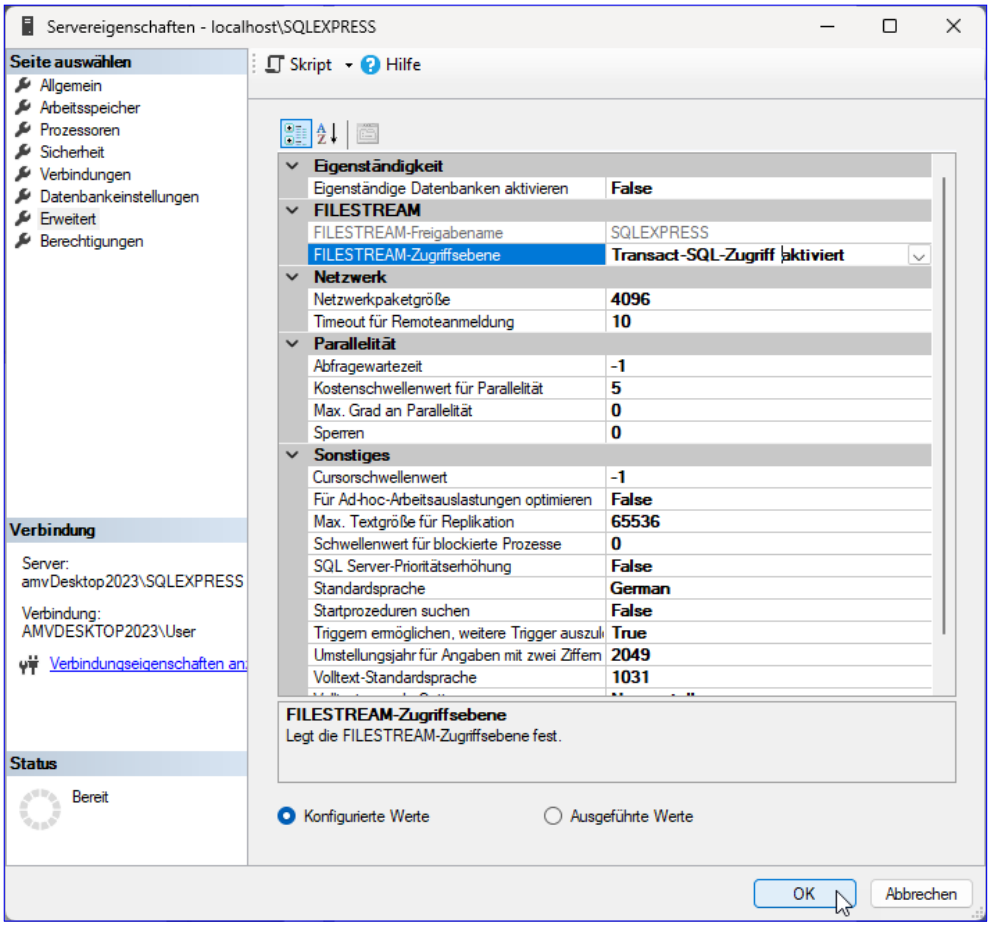

Bild 5: **FILESTREAM** in den SQL Server-Eigenschaften im SQL Server Management Studio

LESTREAM für den Zugriff über das Dateisystem.

- **Windows-Freigabename**: Gibt den Namen des Verzeichnisses an, in dem die Dateien gespeichert werden.
- **Zugriff von Remoteclients auf FILESTREAM-Daten zulassen**: Erlaubt den Zugriff von anderen Rechnern im Netzwerk.

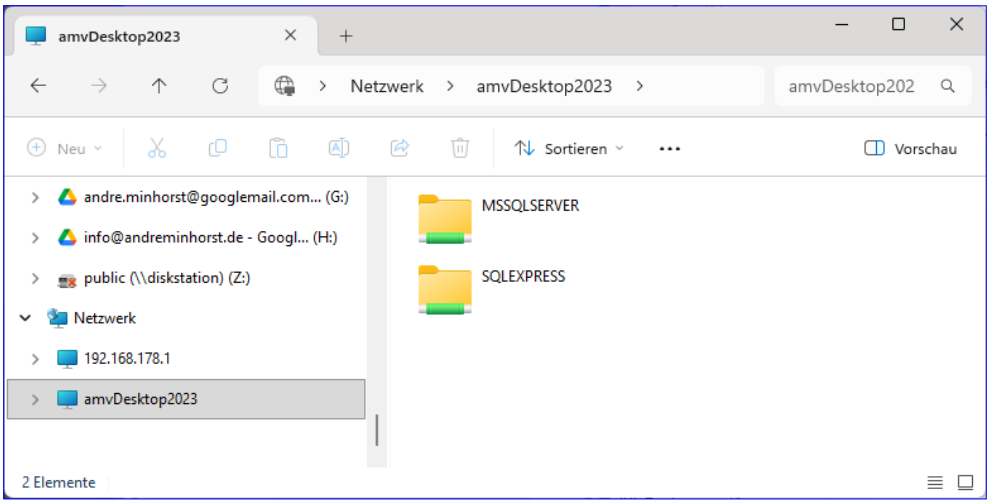

Bild 6: Freigabe für **SQLEXPRESS**

Hier aktivieren wir die ersten beiden Optionen und belassen den Windows-Freigabenamen bei **SQLEXPRESS**.

Danach muss der SQL Server neu gestartet werden, was wir wiederum im SQL Server Konfigurationsmanager erledigen, indem wir dort den Kontextmenü **Neu starten** aufrufen (siehe Bild 4).

Alternativ starten wir den SQL Server direkt im SQL Server Management Studio neu, indem wir mit der rechten Maustaste auf den Eintrag für den Server klicken und den Befehl **Neu starten** betätigen.

Die hier vorgenommenen Einstellungen können wir auch über das SQL Server Management Studio vornehmen. Dazu zeigen wir die Eigenschaften des Eintrags für den SQL Server an und wechseln dort zur Seite **Erweitert** (siehe Bild 5).

Wenn wir **FILESTREAM** für den Zugriff über das Dateisystem aktiviert haben, finden wir im Windows Explorer eine entsprechende Freigabe vor (siehe Bild 6). Wenn diese nicht direkt sichtbar ist, geben

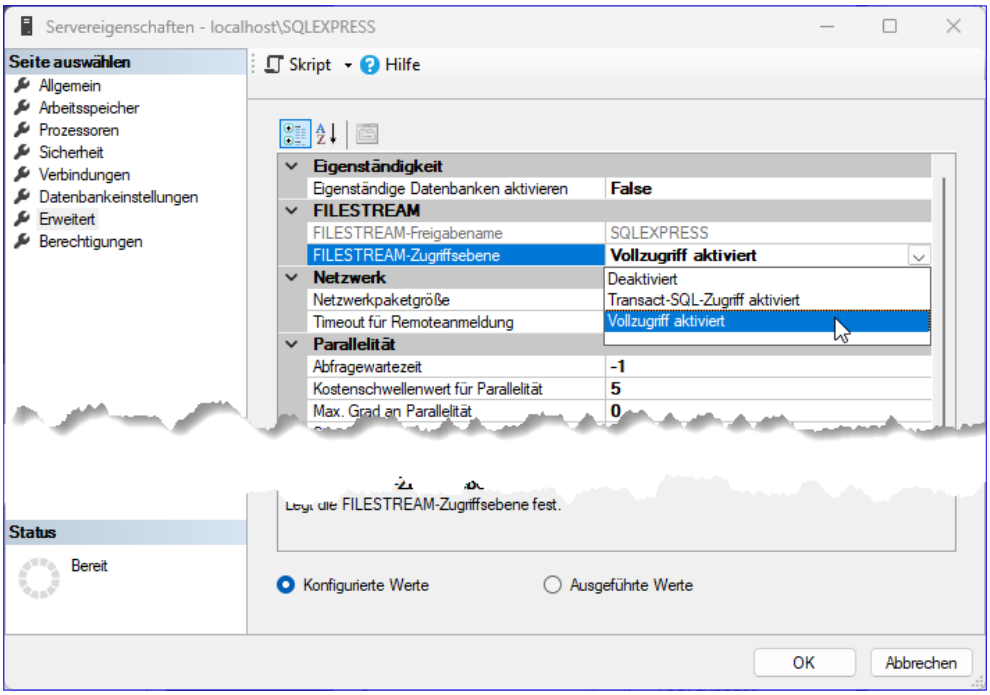

Bild 7: Erweitern auf Vollzugriff

```
CREATE DATABASE FileTableDB
ON
PRIMARY (NAME=FileTableDB,
FILENAME='C:\Program Files\Microsoft SQL Server\MSSQL17.SQLEXPRESS\MSSQL\DATA\FileTableDB.mdf'),
FILEGROUP FileTableDB_GROUP CONTAINS FILESTREAM(NAME=FileTable_DB,
FILENAME='C:\Program Files\Microsoft SQL Server\MSSQL17.SQLEXPRESS\MSSQL\DATA\FileTableDB')
LOG ON (NAME=FileTableDB_LOG,
FILENAME='C:\Program Files\Microsoft SQL Server\MSSQL17.SQLEXPRESS\MSSQL\DATA\FileTableDB.ldf')
```

Listing 1: Erstellen einer Datenbank mit FILESTREAM

wir in der Adressleiste des Windows Explorers folgendes ein:

```
\\Rechnername
```

Dann erscheinen zunächst die SQL Server-Instanzen. Hier klicken wir im Falle der Express-Edition doppelt auf **SQLEXPRESS**.

Auf diese können wir aber aktuell noch nicht zugreifen.

Dazu müssen wir im SQL Server Management Studio noch den vollen Zugriff aktivieren (siehe Bild 7).

Danach können wir schließlich auf die Freigabe **amvDesktop2023\SQLEXPRESS** (oder wie sie bei Dir heißt) zugreifen, finden dort aber bisher weder Ordner noch Dateien vor.

Wozu diese Freigabe benötigt wird, erläutern wir in den folgenden Abschnitten.

Die **FileTable**-Technik basiert auf **FILESTREAM**. Damit können wir sowohl Dateien über die Datenbank speichern und diese dann über das Dateisystem öffnen als auch Dateien in dem für die Datenbank vorgesehenen Verzeichnis speichern und diese damit unter den Zugriff der SQL Server-Datenbank stellen.

FILESTREAM-Beispieldatenbank erstellen

Zu Beispielzwecken erstellen wir zunächst eine neue Datenbank, die mit **FILESTREAM** ausgestattet ist.

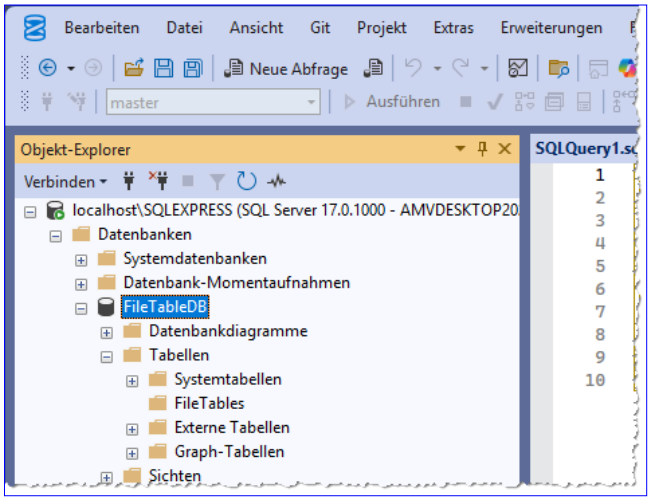

Bild 8: Die neue **FileTable**-Datenbank

Das erledigen wir im SQL Server Management Studio, indem wir in einer neuen Abfrage den Code aus Listing 1 ausführen.

Danach erscheint die neue Datenbank wie in Bild 8 im SQL Server Management Studio.

FILESTREAM-Verzeichnis festlegen

Weiter oben haben wir bei der Anpassung des SQL Servers für die Verwendung von **FILESTREAM** und **FileTable** bereits einen Windows-Freigabenamen festgelegt, unter dem die vom SQL Server verwalteten Dateien aufzufinden sein sollen.

Diese Freigabe wird von der kompletten SQL Server-Instanz genutzt, was bedeutet, dass nicht nur eine, sondern auch mehrere Datenbanken darauf zugreifen.

Deshalb müssen wir pro Datenbank noch ein Unterverzeichnis erstellen. Dieses legen wir in den Eigenschaften der Datenbank auf der Seite Optionen unter **FILESTREAM** fest.

Stelle also für die Eigenschaft **FILESTREAM**-Verzeichnisname einen entsprechenden Wert wie etwa **FileTableDB** ein und lege mit der Eigenschaft **Nicht transaktionsgebundener FILESTREAM-Zugriff** fest, wie weit der Zugriff von außerhalb des SQL Servers zugelassen werden soll – also beispielsweise über den Windows Explorer (siehe Bild 9). Das Verzeichnis können wir auch mit folgender Anweisung festlegen:

```
ALTER DATABASE FileTableDB SET
FILESTREAM(DIRECTORY_NAME='FileTa-
bleDB');
```

Und diese Anweisung stellt die Zugriffsart auf **FULL**, **READONLY** oder **OFF** ein:

```
ALTER DATABASE FileTable SET FILESTREAM(NON_TRANSACTED_AC-
CESS=FULL);
```

Wenn wir nun noch einmal den Windows Explorer bemühen, werden wir unter der bereits vorhandenen Freigabe einen neuen Ordner vorfinden, der den angegebenen Namen trägt (siehe Bild 10).

Zum aktuellen Zeitpunkt können wir allerdings noch keine Dateien über das Dateisystem in diesem Verzeichnis ablegen. Um dies zu ändern, müssen wir eine

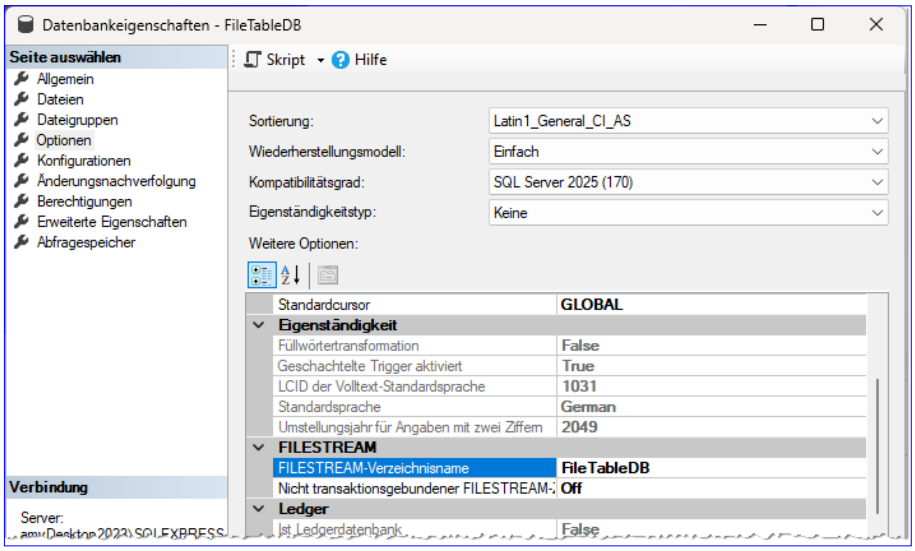

Bild 9: Anlegen eines Verzeichnisses für die **FileStream**-Datenbank

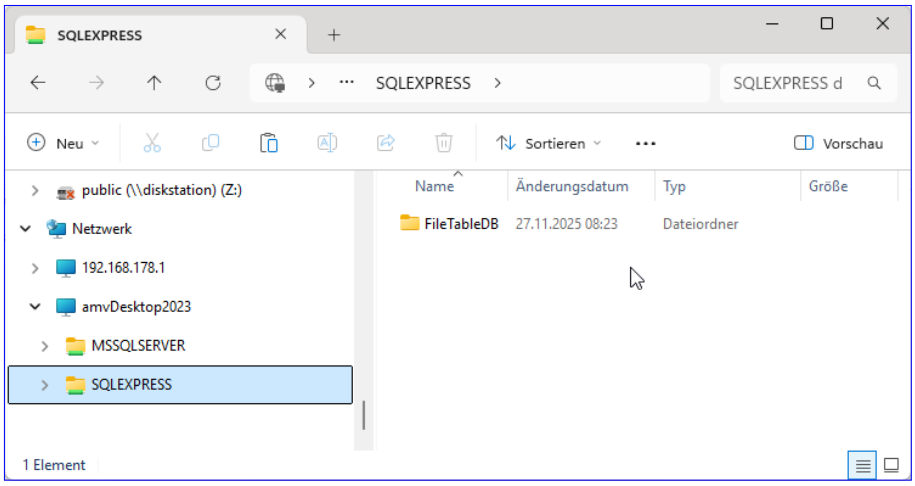

Bild 10: Das Verzeichnis für die **FileTable**-Datenbank

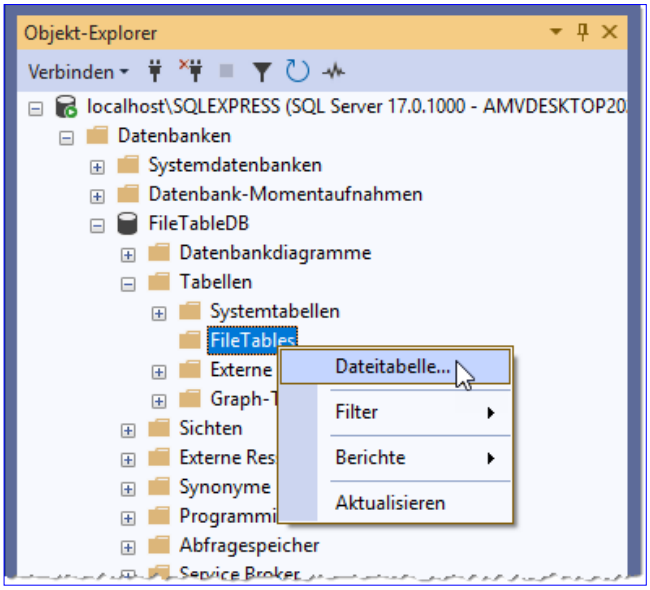

Bild 11: Erstellen einer neuen **FileTable**