HAUFE.
Ausgabe 03/2024

ACCESS
IM UNTERNEHMEN

SQL SERVER IM WEB

Machen Sie Ihre Datenbank über das Internet verfügbar – mit Webserver, SQL Server und dem Management Studio (ab Seite 51).

In diesem Heft:

SQL Server im Web

Webserver werden immer günstiger. Datenbanken werden immer größer. Access allein genügt oft nicht mehr den Anforderungen. Mit dem SQL Server bietet sich eine passende Alternative zum Speichern der Daten an, in vielen Konstellationen sogar kostenlos. Und wenn wir schon mit SQL Server arbeiten, warum dann nicht direkt noch einen Schritt weitergehen und die Daten über das Internet verfügbar machen? Das gelingt zum Beispiel mit einem Windows-Webserver, den man heute für kleines Geld mieten kann. Wir zeigen, wie Sie die Daten Ihrer Datenbankanwendung überall verfügbar machen.

In gleich vier Beiträgen schauen wir uns dieses Thema an. Dabei gehen wir davon aus, dass wir die Daten einer Access-Datenbank nicht mehr nur im lokalen Netzwerk verfügbar machen wollen. Sondern wir wollen auch über das Internet auf die Daten zugreifen – und zwar so performant wie möglich. Das lässt sich heutzutage leicht realisieren, indem wir einen Internetserver mieten. Diesen bekommen wir bereits für wenig mehr als zehn Euro im Monat. Zusammen mit dem kostenlosen SQL Server 2022 Express haben wir so eine Lösung auch für den schmalen Geldbeutel verfügbar. Im ersten Beitrag zu diesem Thema namens **SQL Server im Web, Teil 1: Webserver** zeigen wir ab Seite 51, wie man einen solchen Webserver mietet und vom lokalen Windows-Rechner aus verfügbar macht.

Einen Schritt weiter gehen wir im zweiten Teil der Beitragsreihe mit dem Titel **SQL Server im Web, Teil 2: SQL Server 2022 Express installieren** ab Seite 58. Hier installieren wir auf dem gemieteten Webserver die kostenlose Datenbanksoftware SQL Server 2022 Express. Damit wir damit komfortabel arbeiten können, benötigen wir eine passende Benutzeroberfläche. Der SQL Server selbst kommt nicht mit einer kompletten Entwicklungsumgebung wie Access. Also installieren wir zusätzlich das SQL Server Management Studio auf dem Server. Wie das gelingt, zeigt der Beitrag **SQL Server im Web, Teil 3: SQL Server Management Studio installieren** ab Seite 65.

Doch damit nicht genug. Bisher können wir nur über die Remotedesktopverbindung auf den Webserver zugreifen und so die Datenbank auf dem SQL Server administrieren. Im Beitrag **SQL Server im Web, Teil 4: Fernzugriff per SSMS** zeigen wir ab Seite 68, wir vom lokalen Windowsrechner aus auf den SQL Server zugreifen und diesen verwalten können. Wie es damit weitergeht, schauen wir uns in der folgenden Ausgabe von **Access im Unternehmen** an.

Dazu bereiten wir uns im Beitrag **Datenmodell Mitarbeiterverwaltung** ab Seite 2 vor, wo wir ein Beispieldatenmodell für die Migration einer Access-Datenbank zum SQL Server vorbereiten. Beim Experimentieren mit dem SQL Server sind uns Probleme beim Umgang mit Datumsfeldern aufgefallen, die wir im Beitrag **Probleme mit dem Datentyp »Datum/Uhrzeit erweitert«** ab Seite 10 beschreiben.

Wer viel VBA-Code erzeugt, interessiert sich vielleicht für die beiden Beiträge **Access-Quellcodeverwaltung mit GitHub Desktop** und **Datenbanken vergleichen mit GitHub Desktop** (ab Seite 17).

Und schließlich zeigen wir ab Seite 40 noch, wie wir Daten aus einem Unterformular nach Excel exportieren können: **Access-Unterformulare: Filtern & gezielt nach Excel exportieren**.

Viel Spaß beim Lesen!

Ihr André Minhorst

ACCESS
IM UNTERNEHMEN

Hier geht es weiter ...

Wenn Sie bis hierhin gelesen haben, könnte ich mir vorstellen, dass Sie der Rest auch noch interessieren würde. Das Beste kommt nämlich meistens erst am Ende!

Für weniger als 50 Cent am Tag* bekommen Sie nicht nur diesen Beitrag, sondern auch noch mehr als tausend weitere Artikel über unser Onlinearchiv auf www.access-im-unternehmen.de. Plus alle zwei Monate 76 Seiten neues Know-how rund um die Programmierung mit Microsoft Access.

Alle Informationen hier – jetzt bestellen und direkt loslesen!

ZUM SHOP

Datenmodell Mitarbeiterverwaltung

Eigentlich wollten wir nur ein kleines Datenmodell erstellen, das einige Tabellen enthält, die alle Beziehungstypen und Felddatentypen abbildet. Dieses wollten wir als Beispiel für eine SQL Server-Migration verwenden. Allerdings ist das Datenmodell so umfangreich geworden, dass wir uns entschieden haben, dieses einmal in einem Beitrag vorzustellen. Es enthält alle wichtigen Datentypen, alle Beziehungstypen und auch verschiedene Eigenschaften wie eindeutige und nicht eindeutige Indizes, Fremdschlüsselfelder, Felder, die den Wert Null enthalten dürfen und solche, die es nicht dürfen und vieles mehr. Dies ist ein Entwurf für ein solches Datenmodell, das keinen Anspruch auf Vollständigkeit hat – und es werden auch nicht alle Aspekte behandelt, die man vielleicht noch in einer solchen Verwaltung erwartet. Die Verwaltung von Gehältern, Urlauben et cetera würden wir gegebenenfalls in weiteren Beiträgen vorstellen.

Verwalten von Mitarbeitern

Die Verwaltung von Mitarbeitern ist in Unternehmen, die eine relevante Anzahl Mitarbeiter haben, existenziell. In diesem Beitrag schauen wir uns an, welche Informationen eine Mitarbeiterverwaltung erfassen kann und wie diese auf die verschiedenen Tabellen aufgeteilt werden. Der Entwurf des Datenmodells erhebt keinesfalls den Anspruch auf Vollständigkeit. Wie schon im Einleitungstext erwähnt, gibt es nicht nur noch weitere Bereiche, die hier nicht erfasst werden, wie Lohnbuchhaltung oder Urlaubsverwaltung. Auch die hier vorgestellten Tabellen sind sicher nicht vollständig und decken nicht alle denkba-

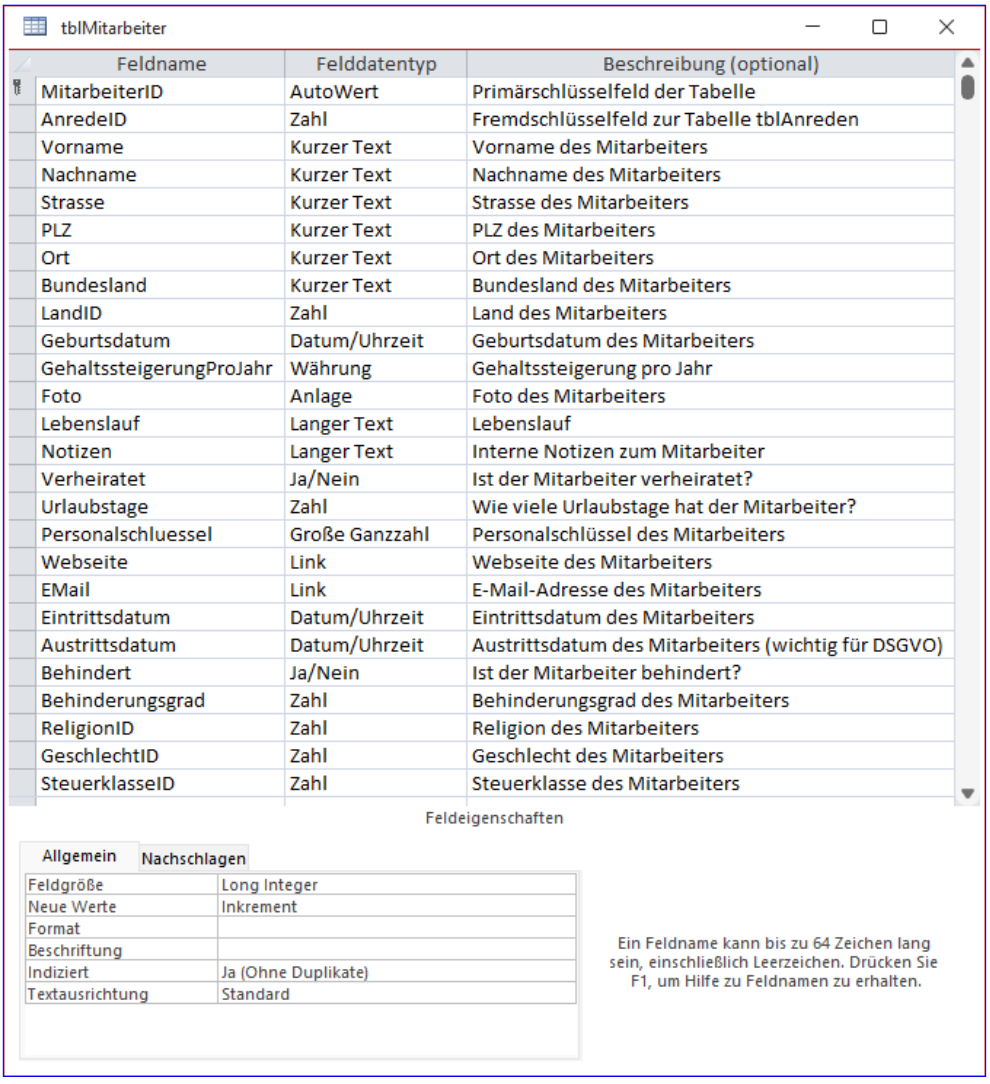

Bild 1: Entwurf der Mitarbeitertabelle

ren Anwendungsfälle ab. Jedoch können sie als Grundlage dienen, wenn Sie selbst einmal vor der Herausforderung stehen, eine Mitarbeiterverwaltung zu programmieren. Das Datenmodell enthält außerdem alle wichtigen Felddatentypen, Beziehungstypen, Beziehungseigenschaften, Indizes und Feldeigenschaften. Dies war die ursprüngliche Motivation: Wir wollten ein Datenmodell erschaffen, mit dem wir die Migration von Access zum SQL Server dokumentieren können. Es ist jedoch sinnvoll, dieses Datenmodell auch einmal zu erläutern, damit deutlich wird, wo die Besonderheiten stecken.

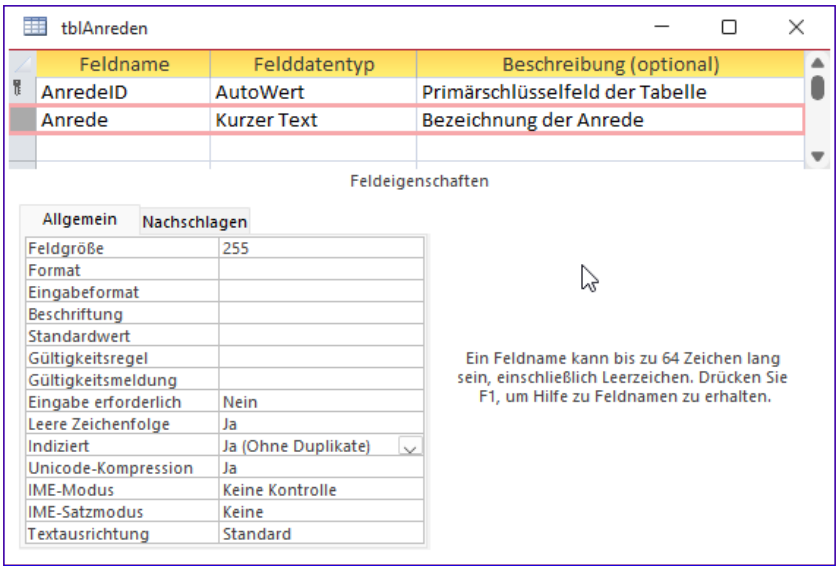

Bild 2: Entwurf der Tabelle **tblAnreden**

Die Tabelle zum Verwalten der Mitarbeiter

Die wichtigste Tabelle ist die Tabelle **tblMitarbeiter**. Ihr Entwurf sieht wie in Bild 1 aus. Neben dem obligatorischen Primärschüsselfeld namens **MitarbeiterID** finden wir einige Felder für Name und Adresse vor, womit wir schon einmal den Datentyp **Kurzer Text** abgedeckt haben.

Diese Felder haben wir im Gegensatz zu einigen anderen Feldern mit dem Wert **Ja** für die Eigenschaft **Eingabe erforderlich** ausgestattet, damit diese immer gefüllt werden müssen.

Die Anrede wählen wir über ein Nachschlagefeld aus, das wir für das Feld **AnredeID** hinterlegen. Die damit verknüpfte Tabelle heißt **tblAnreden** und enthält die beiden Felder **AnredeID** und **Anrede**. In dieser Tabelle haben wir für das Feld **Anrede** einen eindeutigen Index definiert (siehe Bild 2).

Damit stellen wir sicher, dass jede Anrede nur einmal eingegeben werden kann. Die Beziehungen zwischen der Haupttabelle und den Nachschlagetabellen wie **tblAnreden** werden über das Fremdschlüsselfeld der Tabelle **tblMitarbeiter** und dem Primärschlüsselfeld der jeweiligen Nachschlagetabelle wie in Bild 3 definiert.

Das Land geben wir ebenfalls nicht als Text ein, sondern wählen es über ein Nachschlagefeld aus, wobei die Daten in der Tabelle **tblLaender** stecken. Diese enthält, wie für die meisten Lookuptabellen typisch, lediglich zwei Felder – das Primärschlüsselfeld **LandID** und das Feld **Land** mit der Länderbezeichnung. Für dieses Feld haben wir einen eindeutigen Index definiert.

Mit dem Feld **Geburtsdatum** decken wir das Datumsfeld ab.

Das Feld **GehaltssteigerungProJahr** definieren wir mit dem Felddatentyp **Währung** und stellen für die Eigen-

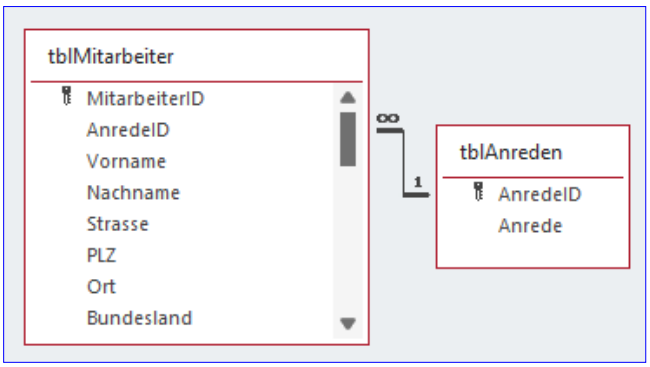

Bild 3: Verknüpfung zwischen der Haupttabelle **tblMitarbeiter** und einer Lookuptabelle, hier **tblAnreden**

schaft **Format** den Wert **Prozentzahl** ein. So erhalten wir für die Werte dieses Feldes die Genauigkeit eines Dezimalfeldes und die Darstellung als Prozentzahl.

Mit dem Feld **Foto** decken wir den Datentyp **Anlage** ab, der unter SQL Server so nicht verfügbar ist – so viel vorneweg.

Bild 4: Inhalt eines Memofeldes mit aktivierter Eigenschaft **Nur anfügen**.

Memofeld mit verschiedenen Eigenschaften

Wir wollen auch Memofelder, die heute unter dem Felddatentyp **Langer Text** zu finden sind, mit den verschiedenen speziellen Eigenschaften abbilden.

In der Tabelle **tblMitarbeiter** finden wir so die beiden Felder **Lebenslauf** und **Notizen**. **Lebenslauf** ist ein Feld mit dem Datentyp **Langer Text** und den Standardeinstellungen. Für das Feld **Notizen** haben wir den Wert der Eigenschaft **Nur anfügen** auf **Ja** eingestellt. Das bedeutet, dass wenn wir Änderungen an dem Inhalt vornehmen, intern die neueste Version gespeichert wird, die vorherigen Versionen aber auch noch vorhanden sind. Diese können wir beispielsweise per VBA mit folgendem Befehl abrufen, hier im Format für das Direktfenster:

```
? ColumnHistory("tblMitarbeiter", "Notizen", "MitarbeiterID = 1")
```

Oder wir klicken mit der rechten Maustaste auf das Tabellenfeld oder ein daran gebundenes Formularfeld und wählen aus dem Kontextmenü den Eintrag **Spaltenverlauf anzeigen** aus, der den Inhalt wie in Bild 4 anzeigt. Beides ist nicht besonders komfortabel, sodass man sich ohnehin eine eigene Funktion zur Anzeige ausdenken muss.

Und wenn man schon dabei ist, könnte man diese auch direkt so ersetzen, dass sie gar nicht verwendet werden muss. Wie das gelingen kann, zeigen wir im Beitrag **Textfeldhistorie in eigener Tabelle speichern** (**www.access-im-unternehmen.de/1500**).

Auf ein Memofeld mit dem Textformat **Rich-Text** kommen wir später in einer anderen Tabelle zurück.

Webseite und **EMail** sind beides Felder mit dem Datentyp **Link**.

Mit **Eintrittsdatum** und **Austrittsdatum** stellt die Tabelle weitere Felder des Datentyps **Datum/Uhrzeit** bereit.

Das Feld **Behindert**, mit dem wir den Datentyp **Ja/Nein** abbilden, gibt an, ob der Mitarbeiter eine Behinderung hat.

Behinderungsgrad ist ein Zahlenfeld, dem wir über das Format **Prozentzahl** die Ansicht als Prozentzahl hinzugefügt haben – diesmal allerdings mit der Feldgröße **Single**, was mit einer geringeren Genauigkeit bei den Nachkommastellen einhergeht als bei Datentyp **Währung**.

ReligionID, **GeschlechtID** und **SteuerklasseID** sind jeweils Fremdschlüsselfelder, die wir als Nachschlagefelder ausgelegt haben und die mit den Tabellen **tblReligionen**, **tblGeschlechter** und **tblSteuerklassen** verknüpft sind.

Weitere Tabellen zum Verwalten von Lookupdaten

Bevor wir uns den übrigen Tabellen zuwenden, schauen wir uns noch alle Lookup- oder Nachschlagetabellen an, die wir neben **tblAnreden** und **tblLaender** noch benöti-

ACCESS
IM UNTERNEHMEN

Hier geht es weiter ...

Wenn Sie bis hierhin gelesen haben, könnte ich mir vorstellen, dass Sie der Rest auch noch interessieren würde. Das Beste kommt nämlich meistens erst am Ende!

Für weniger als 50 Cent am Tag* bekommen Sie nicht nur diesen Beitrag, sondern auch noch mehr als tausend weitere Artikel über unser Onlinearchiv auf www.access-im-unternehmen.de. Plus alle zwei Monate 76 Seiten neues Know-how rund um die Programmierung mit Microsoft Access.

Alle Informationen hier – jetzt bestellen und direkt loslesen!

ZUM SHOP

Probleme mit dem Datentyp »Datum/Uhrzeit erweitert«

Der Felddatentyp »Datum/Uhrzeit erweitert« wurde mit Access 2021 beziehungsweise mit Access aus Office 365 eingeführt. Er speichert genau wie der Datentyp »Datum/Uhrzeit« Datums- und Zeitangaben. Allerdings hat er eine höhere Genauigkeit und Uhrzeiten können nun auch Bruchteile von Sekunden enthalten. Der Grund für die Einführung ist die Herstellung von Kompatibilität mit dem SQL Server, der den Datentyp »datetime2« enthält. Wenn wir Tabellenverknüpfungen zu den Tabellen einer SQL Server-Datenbank herstellen, werden Felder des Typs »datetime2« automatisch mit dem neuen Access-Datentyp »Datum/Uhrzeit erweitert« übersetzt. Allerdings bringt das diverse Probleme mit sich. Zum Beispiel können die in Access eingebauten Datumsfunktionen nicht richtig mit diesem Datentyp umgehen. Was das im Detail bedeutet und welche Lösungsmöglichkeiten es gibt, erläutern wir in diesem Beitrag.

Grundlagenwissen: Interne Speicherung von Datumswerten

Bevor wir richtig einsteigen und für alle, die das noch nicht wissen: Datum-/Zeitwerte werden in Access intern als **Double**-Zahlen gespeichert.

Der Teil vor dem Komma entspricht der Anzahl der Tage seit dem 30.12.1899, der Teil nach dem Komma gibt den Anteil an einem kompletten Tag an – 0,5 entspricht also 12:00 Uhr, 0,75 entspricht 18:00 Uhr und so weiter.

Der Datentyp »Datum/Uhrzeit erweitert«

Für den Einstieg schauen wir uns den neuen Datentyp **Datum/Uhrzeit erweitert** einmal im Detail an und vergleichen diesen mit dem bisher verwendeten Datentyp **Datum/Uhrzeit**.

Beispieltabelle

Um mit dem Datentyp **Datum/Uhrzeit erweitert** zu experimentieren und die Unterschiede zum Datentyp **Datum/Uhrzeit** aufzuzeigen, wollen wir eine einfache Tabelle mit je einem Feld dieser beiden Datentypen anlegen. Dabei stoßen wir gleich auf den ersten Hinweis, dass die Verwendung unter Umständen kompliziert werden könnte ...

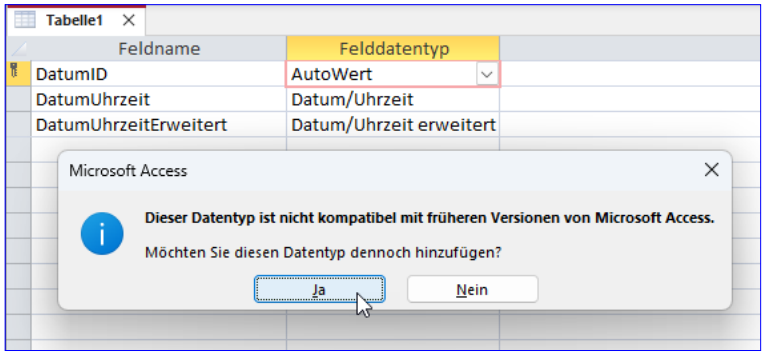

Bild 1: Meldung beim Hinzufügen eines **Datum/Uhrzeit erweitert**-Feldes

Meldung beim Anlegen von »Datum/Uhrzeit erweitert«-Feldern

Wenn wir nämlich einer Datenbank erstmalig ein Feld mit dem Datentyp **Datum/Uhrzeit erweitert** hinzufügen und versuchen, die Tabelle zu speichern, erhalten wir die Meldung aus Bild 1.

Der Datentyp ist also nicht mit älteren Versionen von Access kompatibel, sprich: Mit Version 2019 und älter.

Wir haben die Probe gemacht und die Beispieldatenbank unter Access 2016 geöffnet. Gleich beim Starten erhalten wir die Meldung aus Bild 2. Daher an dieser Stelle der wichtige Hinweis: Wenn Sie Ihre Datenbankanwendung

mit Benutzern teilen wollen, die Access 2019 oder älter verwenden, nutzen Sie nicht den Datentyp **Datum/Uhrzeit erweitert**. Das Gleiche gilt übrigens auch für den Datentyp **Große Ganzzahl**. In beiden Fällen kann die Datenbank nicht mehr mit älteren Access-Versionen geöffnet werden.

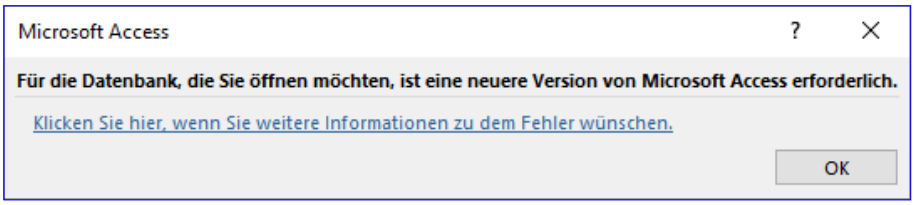

Bild 2: Versuch, eine Datenbank mit Datum/Uhrzeit erweitert-Feld in Access 2016 zu öffnen

Vergleich zwischen »Datum/Uhrzeit« und »Datum/Uhrzeit erweitert«

In den folgenden Abschnitten schauen wir uns die offensichtlichen und auch die nicht so offensichtlichen Unterschiede zwischen den beiden Datentypen an.

Offensichtliche Unterschiede in einer Tabelle

Wenn wir je ein Feld der beiden Datentypen **Datum/Uhrzeit** und **Datum/Uhrzeit erweitert** anlegen und die Eigenschaften vergleichen, sehen wir einen zusätzlichen Eintrag bei dem neu eingeführten Datentyp. Diese Eigenschaft heißt **Dezimalstellenanzeige** (siehe Bild 3). Wozu benötigen wir bei einem Datum eine Dezimalstellenanzeige?

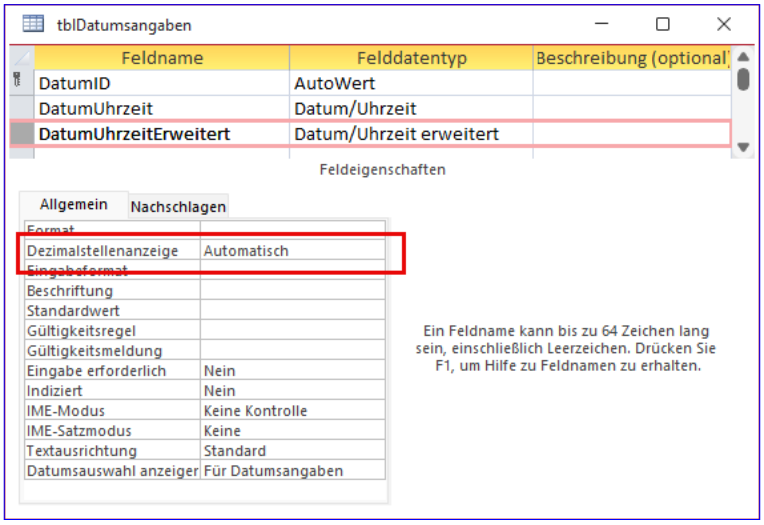

Bild 3: Zusätzliche Eigenschaft **Dezimalstellenanzeige**

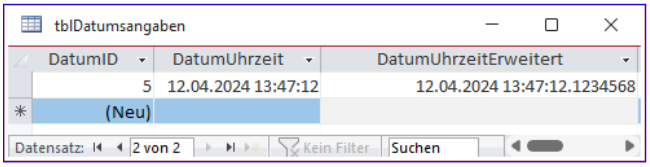

Bild 4: Standardanzeige der Dezimalstellen

Ganz einfach: Eingangs haben wir schon erwähnt, dass dieser Datentyp nicht nur Datum und Uhrzeit speichern kann, sondern auch Bruchteile von Sekunden, also eine erweiterte Uhrzeit verglichen mit dem **Datum/Uhrzeit**-Feld. Wie können also auch Werte wie den folgenden speichern:

```
12.04.2024 13:47:12.123456789
```

Die Eigenschaft **Dezimalstellenanzeige** gibt dabei schlicht an, wieviele Stellen der Bruchteile von Sekunden angezeigt werden sollen.

Wenn wir den Standardwert **Automatisch** übernehmen, erscheinen die Dezimalstellen bei Datum aus dem obigen Beispiel wie in Bild 4. Es wird also auf die siebte Nachkommastelle gerundet.

Wenn wir den Wert **0** für **Dezimalstellenanzeige** angeben, erscheinen Datum und Uhrzeit wie im herkömmlichen **Datum/Uhrzeit**-Feld.

Für die Werte von **1** bis **7** zeigt das Feld die jeweilige Anzahl an Nachkommastellen an.

Für Werte von **8** bis **15** (dem größten verfügbaren Wert) zeigt das Feld auch den auf die siebte Stelle gerundeten Wert an.

Kleinster Wert der Datums-Datentypen

Der kleinste Wert, den wir für Felder mit dem Felddatentyp **Datum/Uhrzeit** angeben können, ist der **1.1.100**. Geben wir ein Datum wie **1.1.99** ein, interpretiert Access dies als **1.1.1999**.

Geben wir für ein **Datum/Uhrzeit erweitert**-Feld den Wert **1.1.100** ein, wird dieses als **1.1.2024** interpretiert. Interessanterweise werden auch Eingaben wie **1.1.1984**, **1.1.2023** oder **1.1.2030** in **1.1.2024** umgewandelt. Wir werden uns also gleich noch einmal genauer ansehen, wie man Datumsangaben richtig in Felder dieses Datentyps eingibt.

Vorher schauen wir uns noch an, welcher tatsächlich der kleinstmögliche Wert für ein **Datum/Uhrzeit erweitert**-Feld ist. Dieser lautet: **1.1.1**.

Größter Wert der Datums-Datentypen

Der größtmögliche Wert für ein Feld mit den Datentyp **Datum/Uhrzeit** lautet **31.12.9999 23:59:59**.

Der Datentyp **Datum/Uhrzeit erweitert** liefert fast den gleichen größtmöglichen Wert, hängt jedoch noch ein paar Dezimalstellen dran, also lautet der angezeigte Wert **31.12.9999 23:59:59.9999999**.

Das ist aber auch nur der Wert, den die Dokumentation von Microsoft liefert. Wir haben einmal den Wert **31.12.9999 23:59:59.9999999** eingegeben, der auch übernommen wird. Wenn wir jedoch noch eine **9** anhängen, also **31.12.9999 23:59:59.99999999**, zeigt das Feld den Wert **31.12.9999 24:00:00.0000000** an. Was nach unserer Erwartung als **1.1.10000 00:00:00.0000000** angezeigt werden sollte.

In der Dokumentation wird als größter Wert für den Datentyp **Datum/Uhrzeit** übrigens **9999-12-31 23:59:59.999** angegeben – diesen Wert können wir weder in ein Feld dieses Datentyps eingeben noch lässt sich dieser mit VBA als Datum interpretieren (siehe Bild 5).

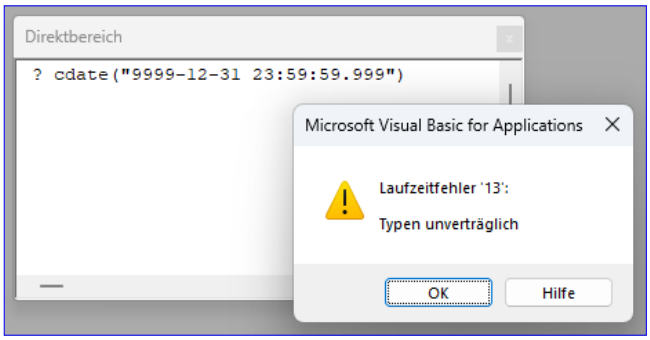

Bild 5: Fehler beim Auswerten von Dezimalstellen beim Datum

Merkwürdiges Eingabeverhalten bei »Datum/Uhrzeit erweitert«

Das Thema Eingabe haben wir weiter oben bereits angesprochen. Wir haben dort festgestellt, dass wir, wenn wir reine Datumsangaben außerhalb von 2024 eingeben, das Jahr immer auf 2024 geändert wird. Die Eingabe von **23.1.1971** wird in **23.01.2024** umgewandelt. Was genau läuft da schief? Wir haben uns den Inhalt einmal per VBA ausgeben lassen. Das wollten wir erst mit **DLookup** erledigen, was aber einen Fehler auslöste – dazu später mehr. Mit dem Zugriff über **OpenRecordset** hat es schließlich geklappt. Und hier ist das Ergebnis:

```
? CurrentDb.OpenRecordset("SELECT DatumUhrzeitErweitert
FROM tblDatumsangaben").Fields(0)
01/23/2024 00:00:00.1971000
```

Access interpretiert die Jahreszahl im gelieferten Datum also offensichtlich als Dezimalteil der Sekunden. Tag und Monat werden korrekt verarbeitet, das Jahr jedoch nicht. Wenn wir ein Datum aus dem aktuellen Jahr eingeben, also beispielsweise 1.1.2024, wird die Jahreszahl auch nur scheinbar korrekt interpretiert – tatsächlich landet diese auch im Dezimalteil der Sekunden und das Jahr 2024 wird hinzugefügt. Hier der für die Angabe von 1.1.2024 gespeicherte Wert:

```
01/01/2024 00:00:00.2024000
```

Wie auch immer dies zu erklären ist: Die Eingabe von Datumsangaben in Felder des Typs **Datum/Uhrzeit er-**

ACCESS
IM UNTERNEHMEN

Hier geht es weiter ...

Wenn Sie bis hierhin gelesen haben, könnte ich mir vorstellen, dass Sie der Rest auch noch interessieren würde. Das Beste kommt nämlich meistens erst am Ende!

Für weniger als 50 Cent am Tag* bekommen Sie nicht nur diesen Beitrag, sondern auch noch mehr als tausend weitere Artikel über unser Onlinearchiv auf www.access-im-unternehmen.de. Plus alle zwei Monate 76 Seiten neues Know-how rund um die Programmierung mit Microsoft Access.

Alle Informationen hier – jetzt bestellen und direkt loslesen!

ZUM SHOP

Access-Quellcodeverwaltung mit GitHub Desktop

Es gibt verschiedene Systeme, mit denen wir die Objekte aus Access extrahieren und unter eine Versionsverwaltung stellen können. In diesem Beitrag stellen wir eine Kombination aus dem kostenlosen Tool »Version Control Add-In« und »GitHub Desktop« vor. Dabei zeigen wir, wie die notwendige Komponente zu Access hinzugefügt wird, wie wir diese nutzen, um die Access-Elemente zu extrahieren und wie wir diese mit GitHub Desktop verwalten, um beispielsweise Unterschiede zwischen verschiedenen Versionen einer Datenbank ausfindig zu machen.

Vorbereitungen

Als Erstes treffen wir einige Vorbereitungen:

- Download des **Version Control Add-Ins** von **https://github.com/joyfullservice/msaccess-vcs-addin/releases/tag/v4.0.34**
- Installation des **Version Control Add-Ins**: Im Download finden wir eine Datei namens **Version Control.accda**. Diese fügen wir zu einem vertrauenswürdigen Ordner hinzu und öffnen diese dann. Hier führen wir die Installation mit einem Klick auf die Schaltfläche **Install Add-In** aus (siehe Bild 1).

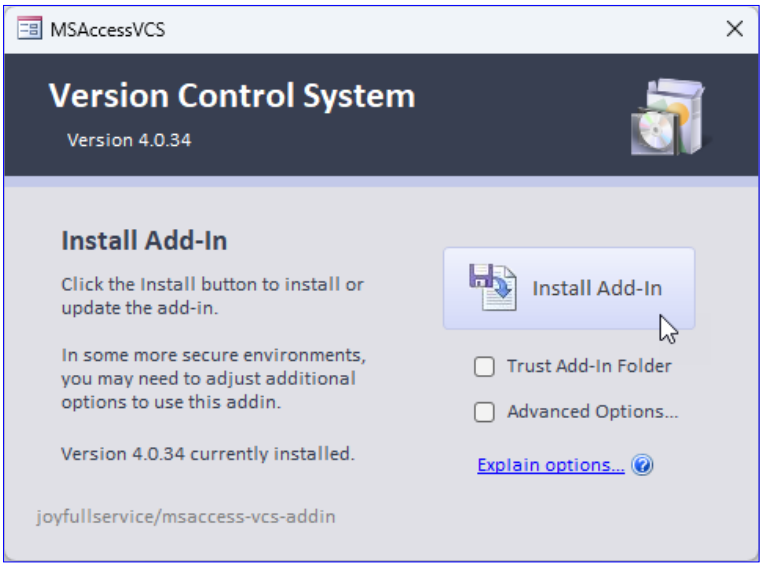

Bild 1: Installieren des Version Control Add-Ins

- Danach finden wir das Add-In bereits als eigenes Ribbon-Tab in Access vor (siehe Bild 2).
- Download von **GitHub Desktop** von **https://desktop.github.com/**
- Installation von **GitHub Desktop** durch Ausführen einer Datei wie **GitHubDesktopSetup-x64.exe**.
- Gegebenenfalls Anlegen eines Benutzerkontos bei **github.com**

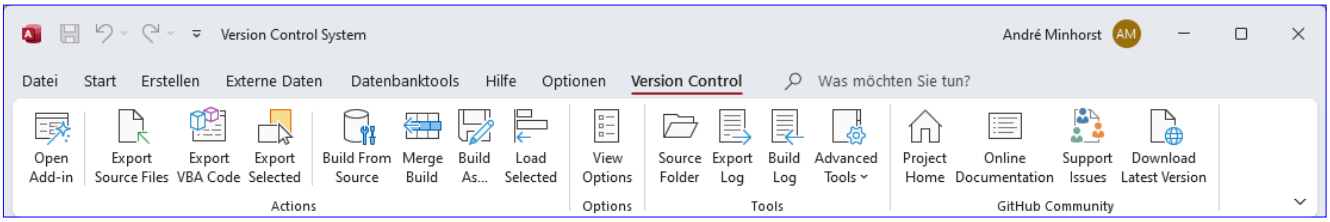

Bild 2: Die Benutzeroberfläche des Version Control Add-Ins

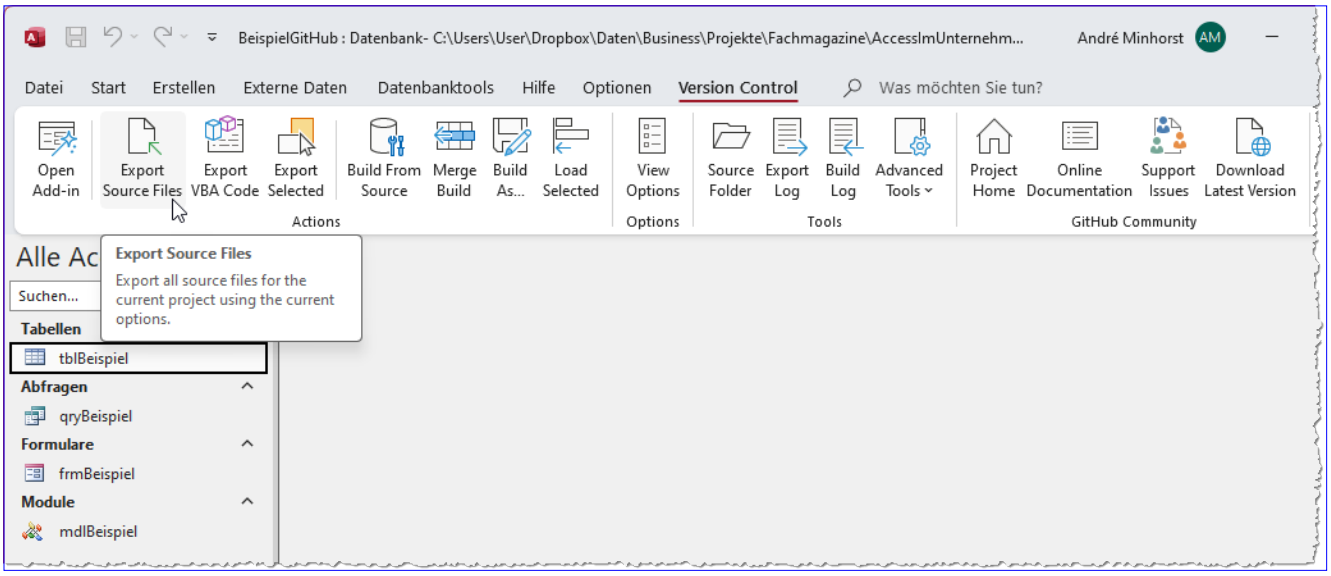

Bild 3: Erster Export der Objekte

Export der Datenbankelemente

Damit können wir den ersten Export der Datenbankelemente einer kleinen Beispieldatenbank durchführen. Wir klicken dazu einfach auf die Schaltfläche **Export Source Files** (siehe Bild 3).

Dieser Vorgang verläuft sehr schnell. Das Ergebnis dokumentiert das Tool in dem Dialog aus Bild 4.

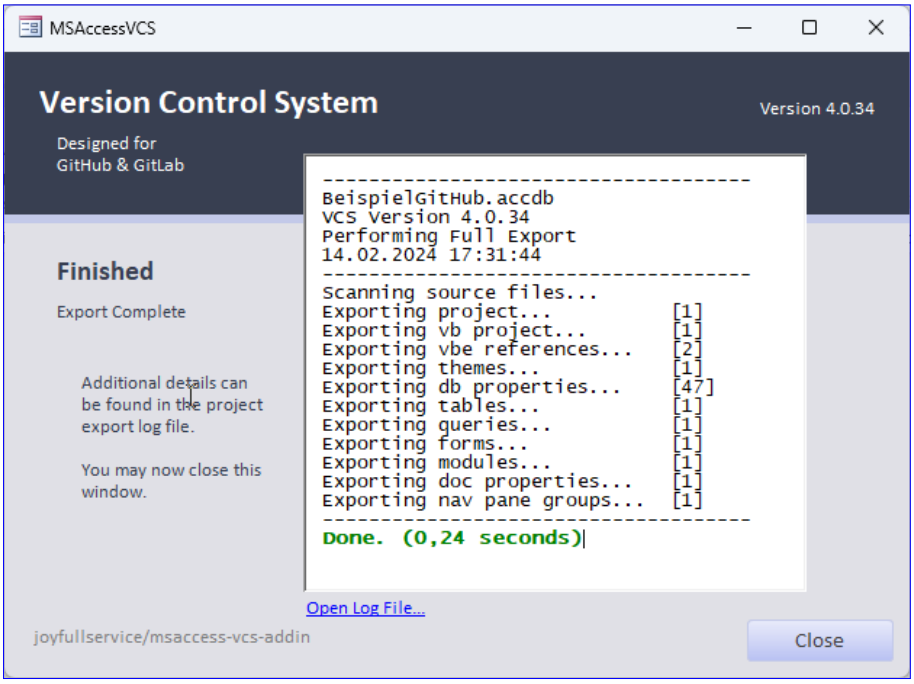

Bild 4: Dokumentation des Exports

Ergebnis des Exports

Untersuchen wir nun, was der Export für uns erledigt hat. Dazu schauen wir uns das Verzeichnis der Datenbank an, in dem wir einen neuen Ordner finden mit dem Namen der Datei und angehängtem **.src** (siehe Bild 5).

In diesem Verzeichnis sehen wir Ordner für Tabellen (**tbldefs**), Abfragen (**queries**), Formulare (**forms**) und Module (**modules**) sowie einige weitere Dateien, die Daten wie beispielsweise die Datenbankoptionen, Einstellungen des Navigationsbereichs, Verweise und mehr enthalten.

Datenbank aus Export neu erstellen

Bevor wir uns anschauen, wie wir mit dem Export in **GitHub Desktop** arbeiten können, werfen wir einen Blick auf

weitere Funktionen von **Version Control**. Da wäre zum Beispiel die Schaltfläche **Build From Source**. Wir rufen diese aus einer neuen, leeren Datenbank auf und erstellen die gesamte Anwendung neu auf Basis der gerade exportierten. Das heißt, dass alle Elemente importiert werden und dass alle weiteren Einstellungen wie Verweise und Access-Optionen wiederhergestellt werden.

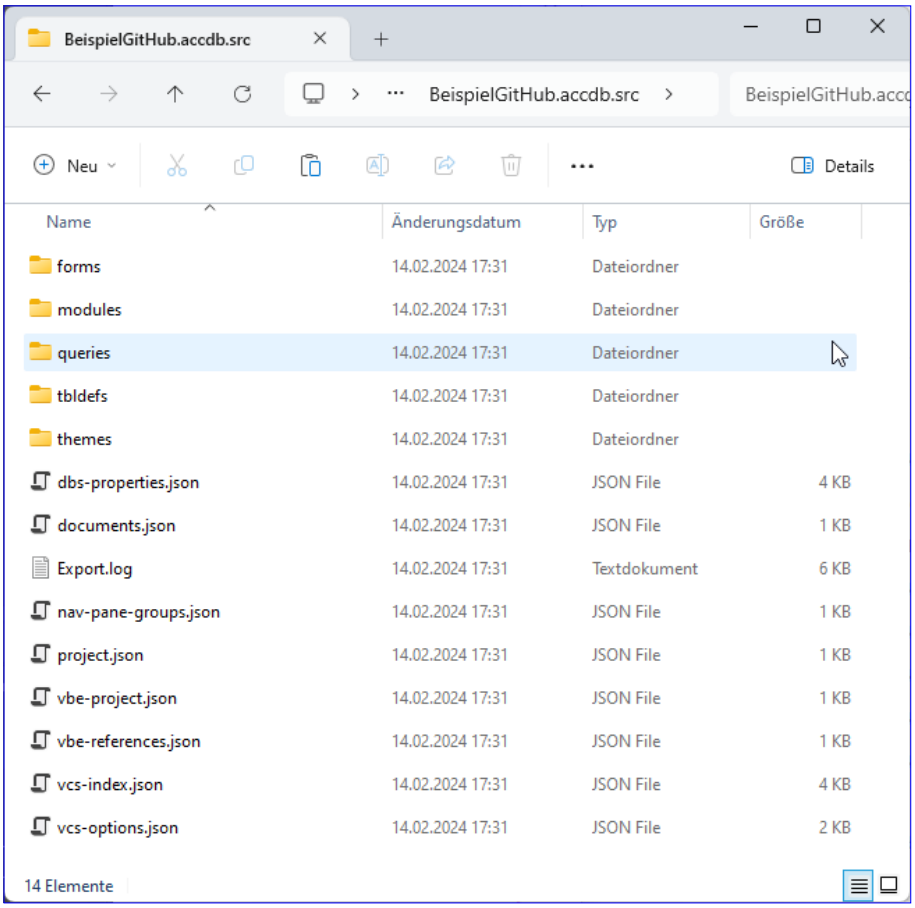

Bild 5: Exportierte Access-Elemente und Informationen

Nur VBA-Code exportieren

Mit der Schaltfläche **Export VBA Code** können wir ausschließlich die VBA-Module exportieren. Dies exportiert nicht nur die reinen Standard- und Klassenmodule, sondern auch noch die Klassenmodule von Formularen und Berichten.

Ausgewählte Elemente exportieren

Wir können auch ein oder mehrere Elemente im Navigationsbereich selektieren und diese exportieren. Dazu nutzen wir die Schaltfläche **Export Selected**.

Extern geänderte Elemente »mergen«

Wir können nun, und das ist eine Funktion für den Fall, dass mehrere Entwickler an einer Datenbank arbeiten, die Änderung eines anderen Entwicklers an einem Element in unsere Access-Datenbank integrieren.

Stellen wir uns den folgenden Ablauf vor: Wir exportieren die Elemente unserer Datenbank in das oben erwähnte Verzeichnis.

Ein anderer Entwickler liest diese Elemente in seine Version der Datenbank ein. Dann ändert er eine oder mehrere Dateien und schreibt diese in Form der geänderten Elemente in das Verzeichnis zurück. Wir können nun die geänderten Elemente wiederum in unsere Version der Datenbank einlesen. Dazu ist es nötig, dass wir einmal den Befehl **Build From Source** aufgerufen haben.

Danach können wir jederzeit alle Änderungen durch einen Klick auf die Schaltfläche **Merge Build** in unsere Version einlesen. Wir haben zu Beispielzwecken einfach eine Zeile in der Datei **mdlBeispiel** geändert. Rufen wir dann den Befehl **Merge Build** auf, erhalten wir den Dialog aus Bild 6.

Dieser Dialog meldet uns, welche Objekte seit dem letzten Export von außen geändert wurden – in diesem Fall genau die Datei **mdlBeispiel.bas**, die den Code des Moduls

mdlBeispiel enthält. Klicken wir auf **Continue**, wird die neue Version des Moduls in unsere Version der Datenbank eingelesen.

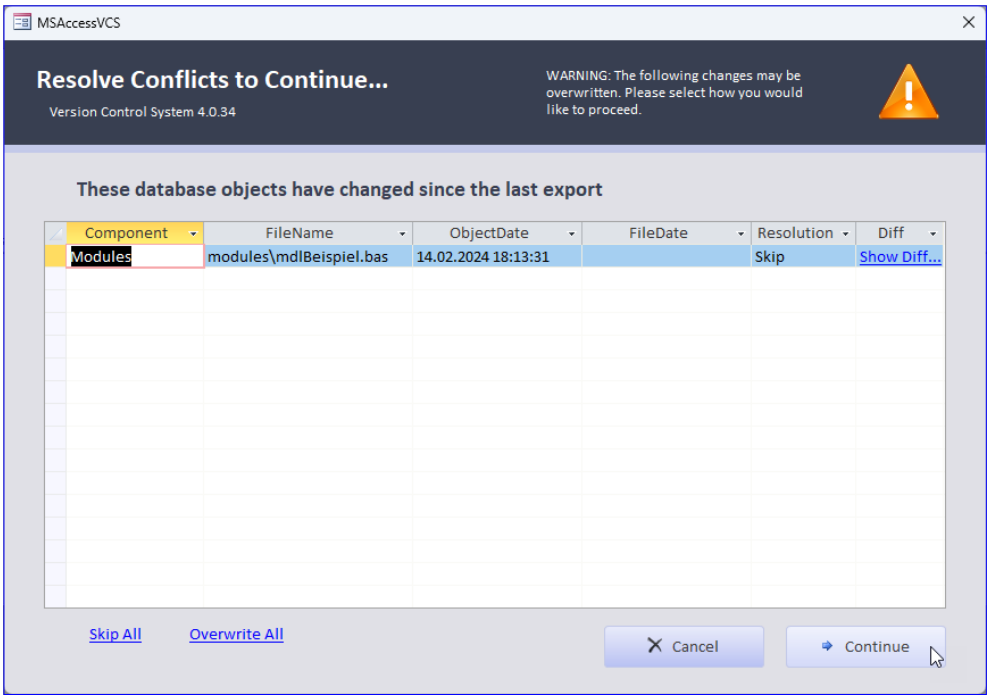

Bild 6: Import geänderter Elemente

Datenbank neu aus Quellen erstellen

Die nächste Schaltfläche mit dem Text **Build As...** ermöglicht es uns, aus dem aktuellen Inhalt des exportierten Verzeichnisses eine neue Version der Datenbank zu erstellen. Dazu geben wir einfach den Dateinamen der zu erstellenden Datei ein.

Einzelne Elemente aus dem Verzeichnis laden

Der nächste Befehl **Load Selected** setzt voraus, dass wir mindestens ein Element im Navigationsbereich markiert haben. Dieses wird dann aus dem Verzeichnis neu eingelesen. Wenn sich die beiden Versionen unterscheiden, erscheint wieder der Dialog zum Lösen von Konflikten.

Weitere Befehle

In der nächsten Gruppe des Ribbons finden wir weitere Befehle:

- **Source Folder**: Öffnet den Zielordner im Windows Explorer.
- **Export Log**: Zeigt die Logdatei zum Exportieren von Elementen an.
- **Build Log**: Zeigt die Logdatei zum Erstellen von Elementen an.
- **Advanced Tools**: Weitere Tools

Nutzen der Quellcodeverwaltung

Bisher haben wir einfach nur Elemente exportiert und importiert und gezeigt, wie wir diese nach Änderungen an den exportierten Elementen wieder importieren können.

Jetzt gehen wir einen Schritt weiter und stellen den exportierten Ordner unter Versionsverwaltung. Dazu öffnen wir **GitHub Desktop**.

Hier finden wir den Startbildschirm aus Bild 7 vor. Wir wählen den Befehl **Add an Existing Repository from your hard drive...** aus – wir haben die Dateien ja bereits angelegt. Nun wählen wir das Verzeichnis mit der Endung **.src** aus.

Nach dem Bestätigen fragt **GitHub Desktop** uns, ob wir ein Repository erstellen möchten, da das Verzeichnis kein Repository repräsentiert.

Dem stimmen wir zu, indem wir auf den Link **create a repository** klicken (siehe Bild 8).

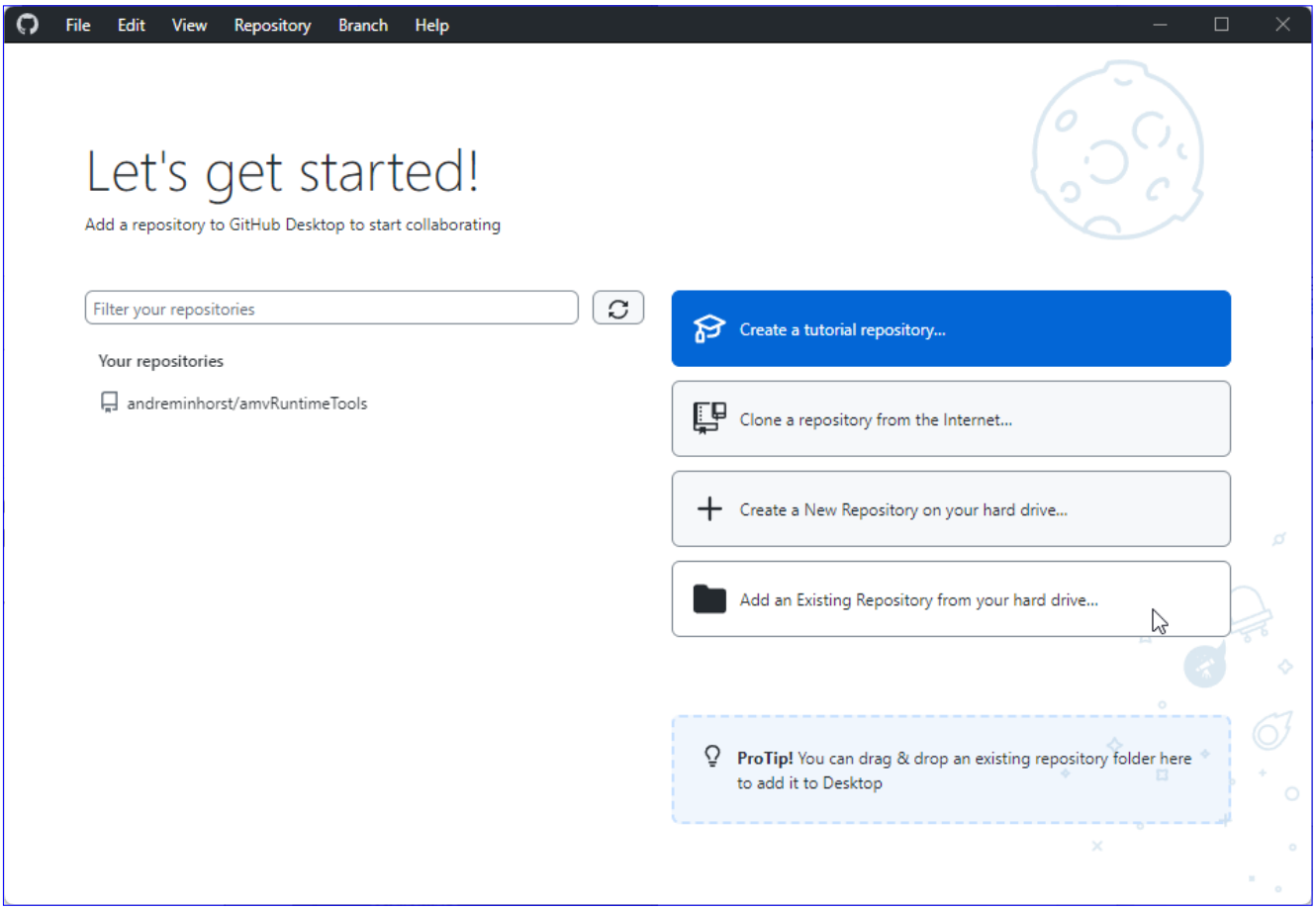

Bild 7: GitHub Desktop nach dem Start

Danach erscheint der Dialog **Create a new repository**, der bereits den Pfad enthält, den wir soeben ausgewählt haben. Die Einstellungen übernehmen wir und klicken auf **Create repository** (siehe Bild 9).

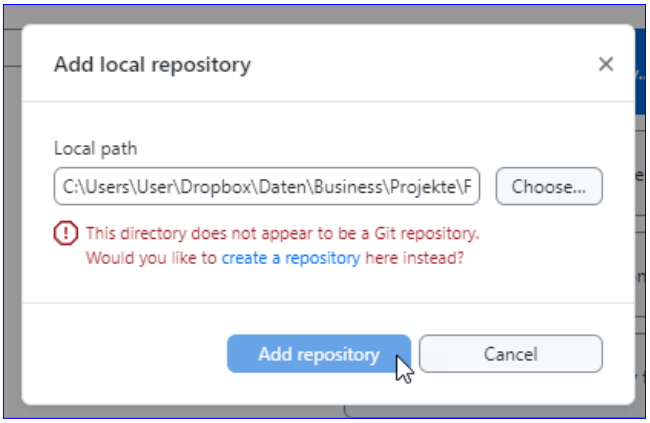

Bild 8: Frage, ob ein Repository erstellt werden soll

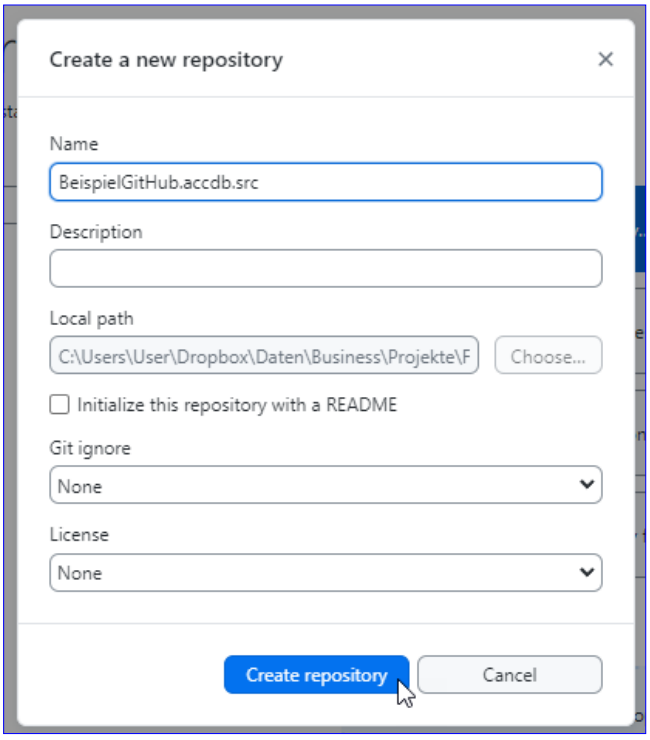

Bild 9: Eingeben der Eigenschaften des Repositorys

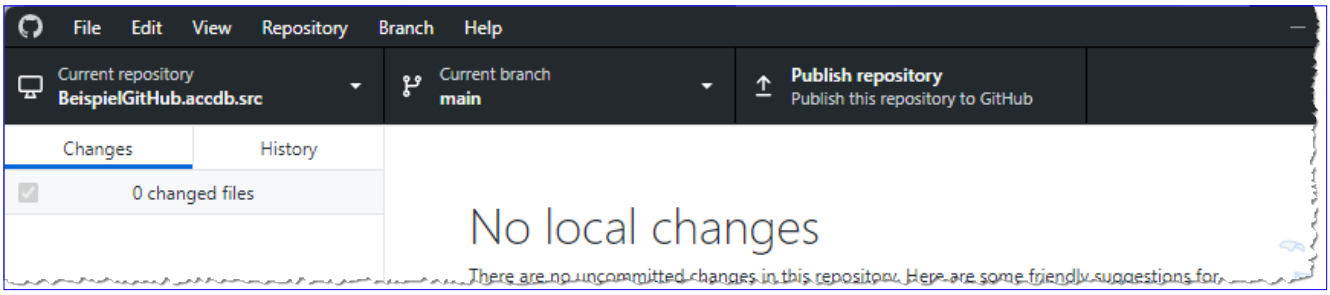

Bild 10: GitHub Desktop mit dem neuen Repository

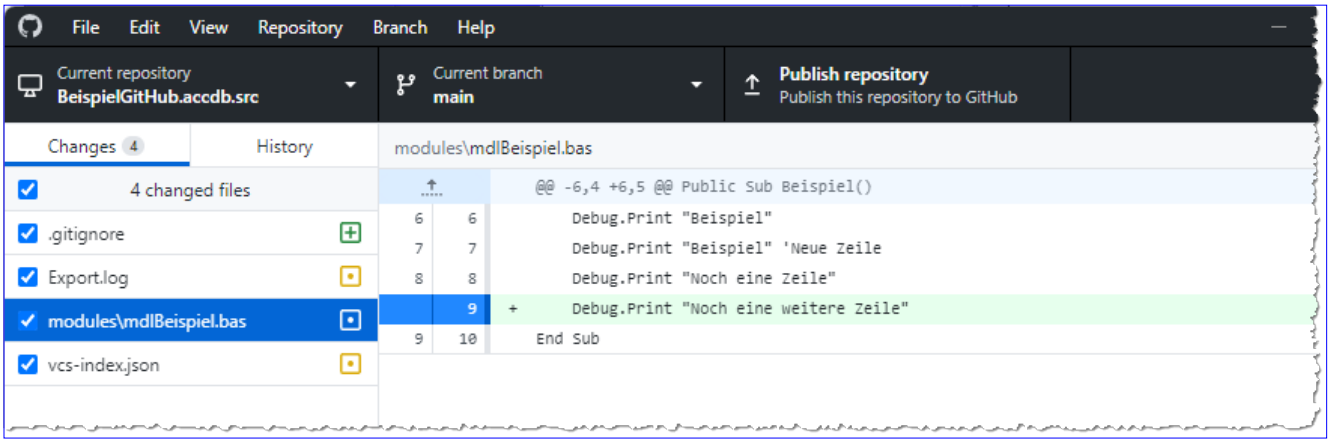

Bild 11: Anzeige der Änderungen des Moduls

GitHub Desktop zeigt nun das neue Repository an, allerdings nur durch die Anzeige unter **Current repository** (siehe Bild 10).

Ansonsten sehen wir keine Änderungen in der Historie, was auch kein Wunder ist, weil wir das Repository neu erstellt haben.

Dadurch wurde der aktuelle Stand bereits als erstes Commit gespeichert. Dieser Commit heißt **Initial Commit**.

Im Verzeichnis finden wir eine neue Datei namens **.gitattributes**, welche Informationen über das Git-Repository enthält.

Jetzt ändern wir eine Zeile im Modul **mdlBeispiel** und exportieren die Elemente erneut.

Schauen wir uns nun den aktuellen Zustand in **GitHub Desktop** an, sehen wir vier Dateien unter **Changes**. Interessant ist die Datei **modules/mdlBeispiel.bas**. Diese

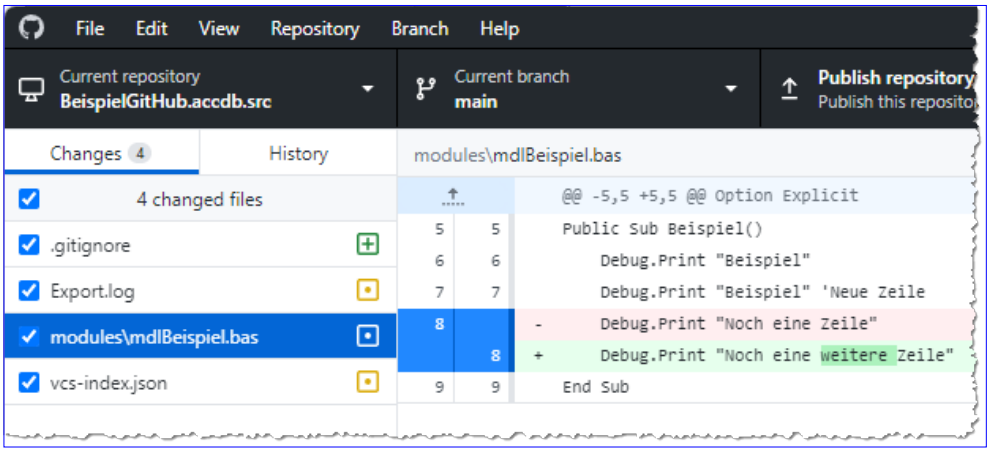

Bild 12: Anzeige weiterer Änderungen des Moduls

ACCESS
IM UNTERNEHMEN

Hier geht es weiter ...

Wenn Sie bis hierhin gelesen haben, könnte ich mir vorstellen, dass Sie der Rest auch noch interessieren würde. Das Beste kommt nämlich meistens erst am Ende!

Für weniger als 50 Cent am Tag* bekommen Sie nicht nur diesen Beitrag, sondern auch noch mehr als tausend weitere Artikel über unser Onlinearchiv auf www.access-im-unternehmen.de. Plus alle zwei Monate 76 Seiten neues Know-how rund um die Programmierung mit Microsoft Access.

Alle Informationen hier – jetzt bestellen und direkt loslesen!

ZUM SHOP

Datenbanken vergleichen mit GitHub Desktop

Im Beitrag »Access-Quellcodeverwaltung mit GitHub Desktop« (www.access-im-unternehmen.de/1495) haben wir gezeigt, wie man mit dem Tool »Version Control Add-In« und mit »GitHub Desktop« den Inhalt einer Access-Anwendung in Form verschiedener Textdateien in ein externes Verzeichnis extrahieren und davon verschiedene Versionen verwalten und wiederherstellen kann. Im vorliegenden Beitrag wollen wir uns nun einen speziellen Einsatzzweck der Kombination dieser beiden Tools ansehen. Angenommen, wir legen regelmäßig Backups unserer Anwendung an. Bei einer neuen Version treten plötzlich Fehler auf, die auf die Änderungen seit dem letzten Backup zurückzuführen sind. Um einzugrenzen, welche Änderungen das Problem auslösen, können wir nun die Elemente der älteren Version exportieren, in ein Repository aufnehmen und als Zwischenstand speichern und dann die aktuelle Version ebenfalls in das Repository schreiben. Dort können wir nun genau anzeigen lassen, welche Unterschiede zwischen den beiden Versionen existieren und dort nach der Ursache für das Problem suchen.

Anwendungsbeispiel: Unterschiede zwischen zwei Versionen einer Datenbank ermitteln

Im Beitrag **Access-Quellcodeverwaltung mit GitHub Desktop** (**www.access-im-unternehmen.de/1495**) haben wir die beiden Tools **Version Control Add-In** und **GitHub Desktop** vorgestellt. Damit können wir aus einer Access-Datenbank alle enthaltenen Elemente exportieren und wiederherstellen (**Version Control Add-In**) oder auch die jeweils exportierten Dateien mit einem Versionsverwaltungssystem verwalten (**GitHub Desktop**).

Um das Beispiel dieses Beitrags nachvollziehen zu können, müssen die beiden oben erwähnten Tools wie in dem genannten Beitrag installiert werden.

Auch wenn man dieses Tool nicht zum Versionieren nutzen möchte, gibt es doch zumindest einen sehr interessanten Anwendungszweck: Wir können damit die Versionsstände zweier Versionen der gleichen Datenbank vergleichen.

Stellen wir uns also vor, wir würden regelmäßig Sicherungskopien unserer Anwendung anfertigen. Das ist Voraussetzung dafür, dass der folgende Plan überhaupt funktioniert. Nachdem wir seit dem letzten gespeicherten Backup einige Änderungen in der Datenbankanwendung durchgeführt haben, funktioniert plötzlich etwas nicht mehr wie gewünscht. Dummerweise haben wir viele Änderungen an verschiedenen Stellen durchgeführt, sodass wir nicht mehr genau wissen, was wir genau geändert haben.

Hier kommen die in diesem Beitrag vorgestellten Techniken ins Spiel. Wir exportieren mit **Version Control System** alle Elemente der früheren Version, erfassen diese mit der Versionsverwaltung GitHub, exportieren dann alle Elemente der aktuellen Version und lassen uns von GitHub die Unterschiede der beiden Anwendungen anzeigen.

Danach ist es unsere Aufgabe, herauszufinden, welche Änderung seit dem letzten Update die Probleme verursacht haben. Das ist gegebenenfalls immer noch Arbeit, aber zumindest wissen wir, an welchen Stellen wir suchen müssen.

Zu Beispielzwecken kopieren wir unsere Beispieldatenbank dazu einmal, sodass wir anschließend eine Datenbank namens **BeispielGitHub.accdb** und eine namens

BeispielGitHub - Kopie.accdb vorfinden. Die Kopie öffnen wir und ändern einige Elemente:

- Hinzufügen eines Feldes namens **EinDrittesFeld** zur Tabelle **tblBeispiel**
- Wir fügen eine weitere Tabelle hinzu, indem wir die Tabelle **tblBeispiel** kopieren und den Namen in **tblBeispiel2** ändern.
- Wir ändern die Datensatzquelle des Formulars **frmBeispiel** in **tblBeispiel2** und fügen noch das neue Feld **EinDrittesFeld** zum Detailbereich hinzu.
- Außerdem fügen wir diesem Formular ein Bild namens **add.png** hinzu, um zu prüfen, ob auch Änderungen an der Tabelle **MSysResources** berücksichtigt werden.
- Wir kopieren im Modul **mdlBeispiel** die Prozedur **Beispiel** und fügen diese unter dem Namen **Beispiel2** erneut ein.

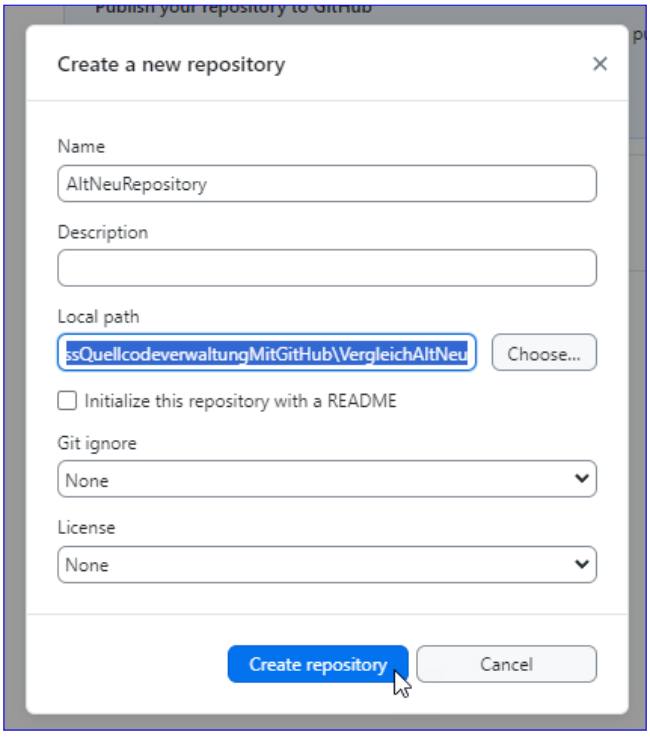

Bild 2: Auswahl des Verzeichnisses

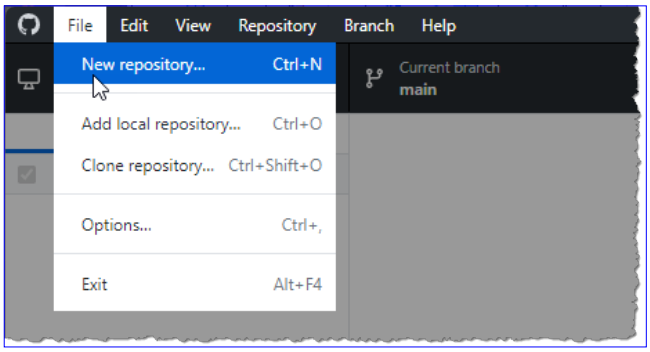

Bild 1: Anlegen eines neuen Repositorys

- Wir fügen einen Verweis auf die Bibliothek **Microsoft Office 16.0 Object Library** hinzu.
- Wir stellen in den Optionen unter **Aktuelle Datenbank** die Option **Formular anzeigen** auf **frmBeispiel** ein.

Das soll zum Testen reichen. Nun führen wir die folgenden Schritte durch:

- Wir erstellen einen neuen Ordner namens **VergleichAltNeu**.
- Wir kopieren die beiden Datenbankdateien **BeispielGitHub.accdb** und **BeispielGitHub - Kopie.accdb** in den neuen Ordner.
- Wir öffnen **GitHub Desktop**.
- Hier legen wir mit dem Menübefehl **File|New Repository...** ein neues Repository speziell für diesen Zweck an (siehe Bild 1).
- Im Dialog **Create a new repository** geben wir einen Namen wie **AltNeuRepository** ein und stellen **Local path** auf das soeben erstellte Verzeichnis **VergleichAltNeu** ein (siehe Bild 2).
- Das Verzeichnis mit den beiden Versionen der Datenbank enthält nun zusätzlich einen weiteren Ordner, den GitHub Desktop erstellt hat (siehe Bild 3).

- Wechseln wir in diesen Ordner, sehen wir dort einen versteckten Unterordner namens **.git** und eine Datei namens **.gitattributes** (siehe Bild 4).

Nun folgen Schritte, die wir später im Detail beschreiben:

- Exportieren der Elemente der alten Version der Datenbank in den Ordner **VergleichAltNeu**.
- Wechsel zu GitHub Desktop, wo wir die nun hinzugefügten Elemente als Unterschiede sehen und diese als neuen Commit anlegen.
- Exportieren der Elemente der neuen Version der Datenbank in den Ordner **VergleichAltNeu**.
- Wechsel zu GitHub Desktop, wo nun die Unterschiede angezeigt werden.

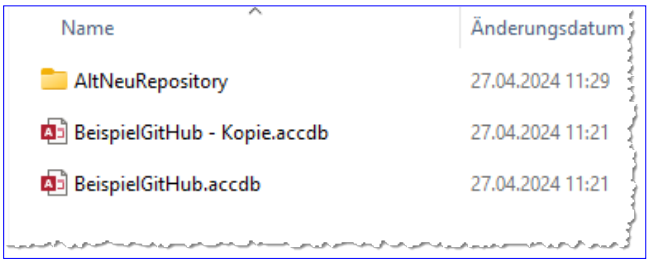

Bild 3: Aussehen des neuen Verzeichnisses

Bild 4: Dateien von GitHub Desktop

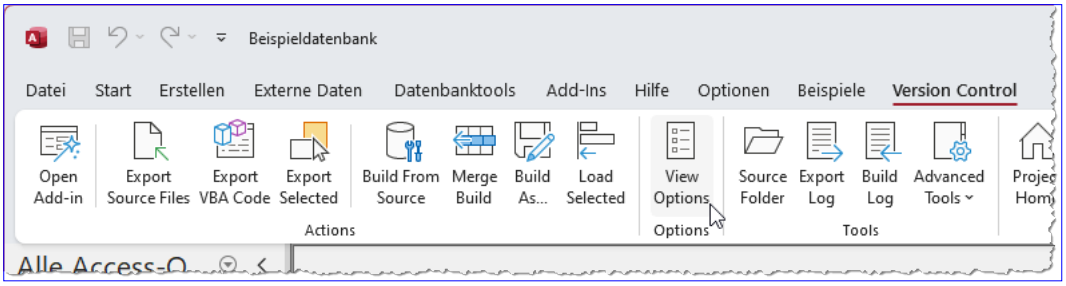

Bild 5: Öffnen der Optionen

Hier können wir nun die Unterschiede zwischen den Versionen im Detail ansehen.

Exportieren der Elemente der alten Version

Diese Schritte sehen im Detail wie folgt aus. Als Erstes öffnen wir die erste Datenbank **BeispielGitHub.accdb**. Hier stellen wir das Zielverzeichnis für den Export der Elemente ein. Dazu klicken wir im Ribbon auf den Befehl **View Options** (siehe Bild 5).

Im nun erscheinenden Dialog **Options** wechseln wir zur Registerseite **Export** und stellen dort unter **Export Folder** den

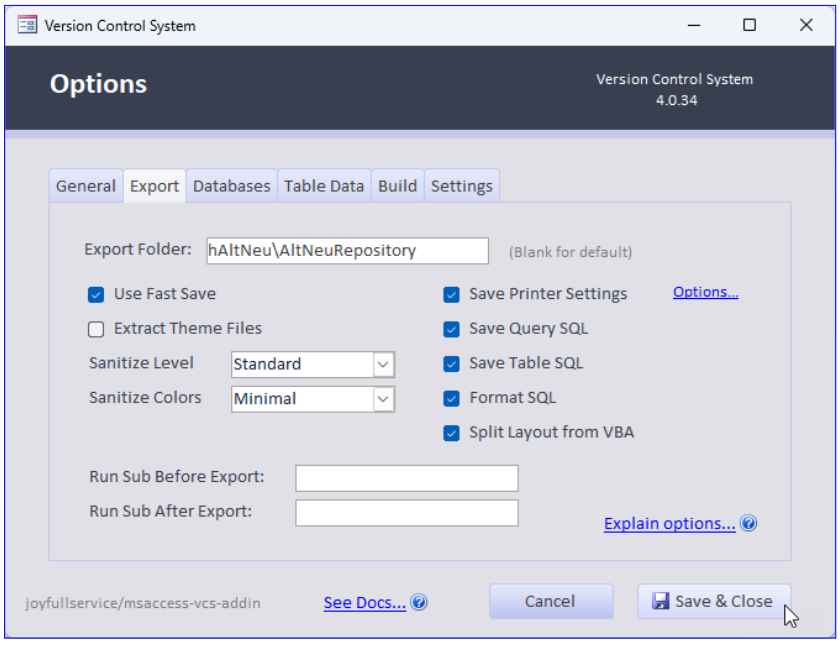

Bild 6: Einstellen des Zielordners

ACCESS
IM UNTERNEHMEN

Hier geht es weiter ...

Wenn Sie bis hierhin gelesen haben, könnte ich mir vorstellen, dass Sie der Rest auch noch interessieren würde. Das Beste kommt nämlich meistens erst am Ende!

Für weniger als 50 Cent am Tag* bekommen Sie nicht nur diesen Beitrag, sondern auch noch mehr als tausend weitere Artikel über unser Onlinearchiv auf www.access-im-unternehmen.de. Plus alle zwei Monate 76 Seiten neues Know-how rund um die Programmierung mit Microsoft Access.

Alle Informationen hier – jetzt bestellen und direkt loslesen!

ZUM SHOP

Access-Unterformulare: Filtern & gezielt nach Excel exportieren

In Unterformularen in der Datenblattansicht lassen sich Daten prima filtern oder sortieren. Mit der DoCmd-Methode TransferSpreadsheet lassen sich Daten einer Tabelle oder Abfrage einfach in eine Excel-Datei exportieren. Aber wie bekommen wir beides unter einen Hut? Wir möchten also in einem Unterformular die Daten filtern und sortieren können und diese in dieser Ansicht in eine Excel-Datei exportieren können. Dazu brauchen wir ein wenig VBA und Kenntnisse der Eigenschaften eines Formulars. In diesem Beitrag zeigen wir, wie wir die Daten der Datenherkunft des Unterformulars wie im Unterformular angegeben filtern und sortieren und exakt so in eine Excel-Datei schreiben.

Beispieldatenbank

Wir verwenden ein einfaches Beispiel, in dem eine Tabelle namens **tblBeispiele** mit den beiden Feldern **ID** und **Beispieltext** die Daten liefert (siehe Bild 1).

Bild 1: Beispieltabelle mit einfachen Texten

Dazu legen wir zwei Formulare an. Das Unterformular **sfmExportExcel** verwendet die Tabelle **tblBeispiele** als Datensatzquelle und zeigt die beiden Felder **ID** und **Beispieltext** im Detailbereich an. Seine Eigenschaft **Standardansicht** stellen wir auf **Datenblatt** ein.

Dieses Unterformular fügen wir in das Hauptformular **frmExportExcel** ein. Außerdem legen wir im Hauptformular eine Schaltfläche namens **cmdExport** an.

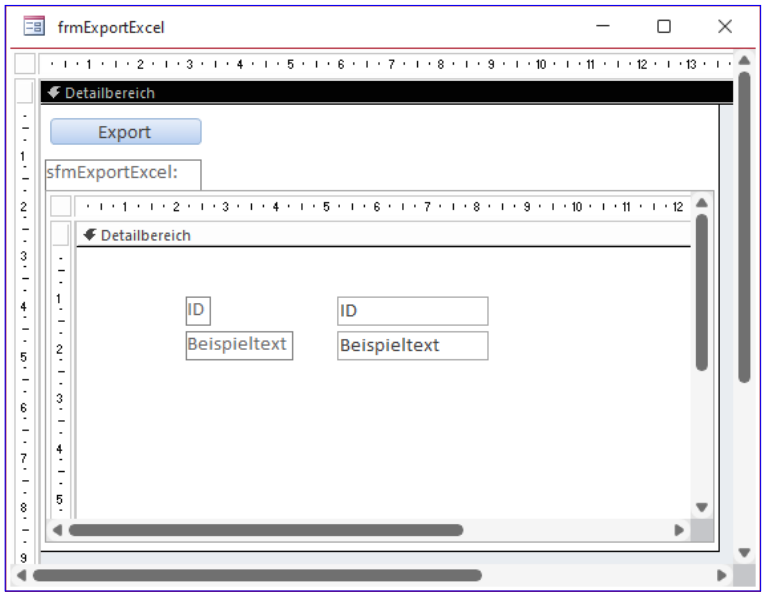

Bild 2: Haupt- und Unterformular mit den Beispieldaten

Diese soll den Export der aktuell im Unterformular angezeigten Datensätze in eine Excel-Datei starten (siehe Bild 2).

Vorgehensweise für das Ermitteln von Filter und Sortierung und den Export

Unsere Lösung basiert darauf, dass wir für ein Unterformular, das der Benutzer gefiltert und/oder sortiert hat, den Filterausdruck als auch den Sortierausdruck aus entsprechenden Eigenschaften des Unterformulars auslesen können.

Wir wissen auch, welche Datensatzquelle das Unterformular verwendet. Damit können wir folgendes tun:

- Filter auslesen
- Sortierung auslesen
- SQL-Ausdruck zusammenstellen, der per **SELECT** alle Datensätze der Tabelle liefert und für die Schlüsselwörter **WHERE** und **ORDER BY** die angewendeten Filter und Sortierungen einträgt
- Aus diesem SQL-Ausdruck eine Abfrage erstellen und speichern und diese Abfrage dann als Datenquelle für den Export mit **DoCmd.TransferSpreadsheet** verwenden
- Die Abfrage wieder löschen

Aktuellen Filter ermitteln

Die grobe, auf unser konkretes Beispiel ausgelegte Prozedur für diesen Zweck finden wir in Listing 1. Die Prozedur legt in der Variablen **strExcel** den Pfad der Zieldatei fest,

```
Private Sub cmdExport_Click()
    Dim db As DAO.Database
    Dim qdf As QueryDef
    Dim strQuery As String
    Dim strSQL As String
    Dim strFilter As String
    Dim strExcel As String
    Dim strOrderBy As String
    strExcel = CurrentProject.Path & "\Beispiele.xlsx"
    On Error Resume Next
    Kill strExcel
    On Error GoTo 0
    Set db = CurrentDb
    strQuery = "qryTemp"
    strSQL = "SELECT * FROM tblBeispiele"
    If Me!sfmExportExcel.Form.FilterOn = True Then
        strFilter = Me!sfmExportExcel.Form.Filter
    End If
    If Me!sfmExportExcel.Form.OrderByOn = True Then
        strOrderBy = Me!sfmExportExcel.Form.OrderBy#
    End If
    If Not Len(strFilter) = 0 Then
        strSQL = strSQL & " WHERE " & strFilter
    End If
    If Not Len(strOrderBy) = 0 Then
        strSQL = strSQL & " ORDER BY " & strOrderBy
    End If
    Set qdf = db.CreateQueryDef(strQuery, strSQL)
    DoCmd.TransferSpreadsheet acExport, acSpreadsheetTypeExcel12Xml, strQuery, strExcel, True
    db.QueryDefs.Delete strQuery
End Sub
```

Listing 1: Exportieren der aktuellen Daten des Unterformulars in eine Excel-Tabelle

die im aktuellen Verzeichnis unter dem Namen **Beispiele.xlsx** gespeichert werden soll.

Gegebenenfalls ist diese Datei bereits vorhanden. Für diesen Fall führen die **Kill**-Anweisung mit dem Pfad aus **strExcel** aus – dies bei deaktivierter Fehlerbehandlung, damit keine Fehlermeldung erscheint, wenn die Datei nicht vorhanden sein sollte und die **Kill**-Anweisung ins Leere läuft.

Dann referenziert die Prozedur das aktuelle Datenbankobjekt und legt in **strQuery** den Namen für die temporär zu erzeugende Abfrage fest, hier **qryTemp**.

Die grundlegende SQL-Anweisung legen wir als **SELECT**-Anweisung über alle Felder der Tabelle **tblBeispiele** fest. Dann prüft die Prozedur anhand der Eigenschaft **FilterOn** des Unterformulars, ob der Filter aktiviert ist. In diesem Fall trägt sie den aktuellen Filter in die Variable **strFilter** ein. Das Gleiche erledigen wir auch für die Sortierung, indem wir den Wert von **OrderByOn** prüfen und gegebenenfalls den Inhalt der Eigenschaft **OrderBy** in die Variable **strOrderBy** übertragen.

Hat **strFilter** einen Wert, hängen wir an die **SELECT**-Anweisung in **strSQL** eine entsprechende **WHERE**-Klausel an. Gleiches erledigen wir für die **ORDER BY**-Klausel, wenn **strOrderBy** keine leere Zeichenkette enthält.

Schließlich erstellen wir mit der **CreateQueryDef**-Methode des **Database**-Objekts eine neue Abfrage mit dem Namen aus **strQuery** und der SQL-Anweisung aus **strSQL**.

Damit können wir nun den eigentlichen Export starten.

Dazu nutzen wir die Methode **TransferSpreadsheet** der **DoCmd**-Klasse. Diese nutzt für die beiden ersten Parameter die Werte **acExport** und **acSpreadsheetTypeExcel12Xml**, damit ein Export in eine **.xlsx**-Datei angestoßen wird. Der dritte Parameter enthält den Namen der Datenquelle, in diesem Fall der mit **strQuery** angegebenen Abfrage. Mit dem vierten Parameter geben wir die zu erstellende Datei an und mit dem fünften die Information, dass Spaltenüberschriften mit exportiert werden sollen. Schließlich löschen wir die Abfrage aus **strQuery** wieder.

Anschließend führen wir eine absteigende Sortierung nach dem Feld **ID** und eine Filterung nach dem Feld **Beispieltext** aus, der nur noch Werte liefert, die ein **A** enthalten. Dann öffnen wir die erstellte Excel-Datei und finden genau die Daten vor, wie wir sie im Unterformular definiert haben (siehe Bild 3).

Variable Gestaltung der Prozedur für den Export nach Excel

Damit haben wir eine recht spezifische Implementierung geschaffen. Diese werden wir nun ein wenig flexibler gestalten, sodass sich diese leicht auf andere Formulare übertragen lässt. Warum aber sollten wir das überhaupt tun?

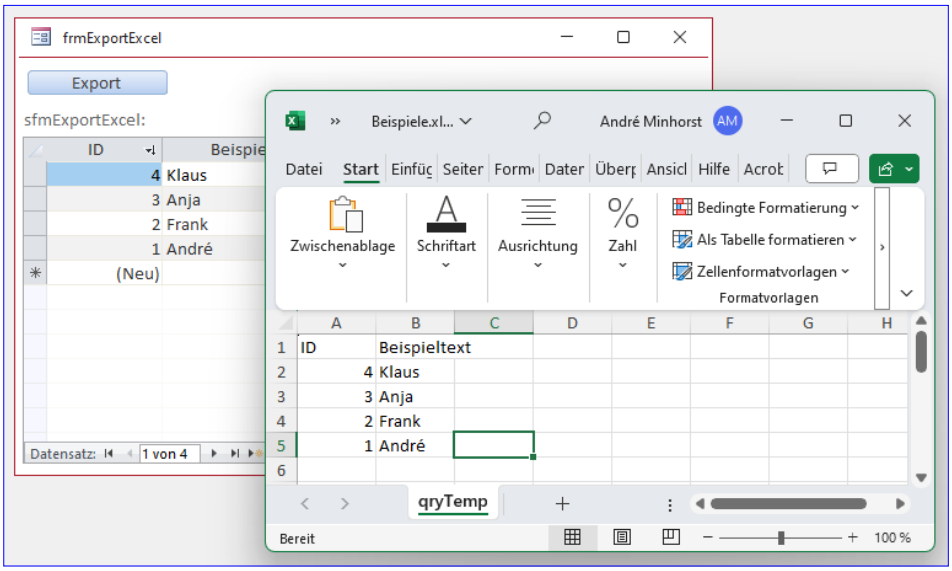

Bild 3: Der Export mit Filter und Sortierung

- Weil wir sonst immer die Referenz zum Unterformular anpassen müssten.

- Weil wir die Tabelle, auf der die Datensatzherkunft des Unterformulars und somit der Excel-Export basiert, bisher fix angegeben haben. Auch diese wollen wir dynamisch ermitteln.

- Weil wir noch nicht berücksichtigen, dass die Datensatzquelle des Unterformulars selbst auch noch Filter und Sortierungen enthalten könnten.

Auswahl der Zieldatei

Dazu fügen wir zunächst ein Textfeld namens **txtExportdatei** hinzu sowie eine Schaltfläche namens **cmdDateiauswahl**. Diese statten wir für das Ereignis **Beim Klicken** mit der folgenden Prozedur aus:

```
Private Sub cmdDateiauswahl_Click()
    Dim objFiledialog As Office.FileDialog
    Set objFiledialog = _
        Application.FileDialog(msoFileDialogSaveAs)
    With objFiledialog
        .Title = "Zieldatei festlegen"
        .InitialFileName = _
            CurrentProject.Path & "\Export.xlsx"
        If .Show = True Then
            Me!txtExportdatei = .SelectedItems.Item(1)
        End If
    End With
End Sub
```

Diese öffnet einen Dialog, mit dem wir die zu erstellende Datei festlegen können. Die gewählte Datei landet im Textfeld **txtExportdatei**.

Funktion zum Exportieren der Excel-Datei

Für maximale Flexibilität haben wir eine Funktion geschrieben, die Sie von einer Schaltfläche aus aufrufen können. Die Beispielschaltfläche **cmdExportFlexibel** nutzt dazu den folgenden Aufruf:

```
Private Sub cmdExportFlexibel_Click()
    ExportFlexibel Me!txtExportdatei, Me!sfmExportExcel.Form
End Sub
```

Die Funktion **ExportFlexibel** aus Listing 2 verwendet diese Daten. Alles weitere ist tatsächlich flexibel – Sie können diese Funktion für alle weiteren Formulare mit Unterformular so nutzen.

Auch diese Prozedur löscht zunächst eine eventuell bereits vorhandene Zieldatei. Wir verwenden weiterhin den Namen **qryTemp** für die temporäre Datei.

Der nächste Schritt unterscheidet sich bereits deutlich von der ersten Variante. Hier verwenden wir nämlich nicht einfach die Abfrage **SELECT * FROM tblBeispiele** als Datensatzherkunft für die zu erstellende Abfrage als Quelle für den Export, sondern untersuchen die Datensatzquelle des Unterformulars explizit.

Dazu prüfen wir, ob die ersten sechs Buchstaben der Datensatzquelle dem Ausdruck **SELECT** entsprechen. In diesem Fall enthält die Datensatzquelle gegebenenfalls selbst eine **WHERE**- oder **ORDER BY**-Klausel. Hier weisen wir den Inhalt von **sfm.RecordSource** direkt der Variablen **strSQL** zu:

```
    If Left(sfm.RecordSource, 6) = "SELECT" Then
        strSQL = sfm.RecordSource
```

Anderenfalls erstellen wir eine **SELECT**-Anweisung auf Basis der angegebenen Tabelle oder Abfrage:

```
    Else
        strSQL = "SELECT * FROM " & sfm.RecordSource
    End If
```

Dann tragen wir, sofern der Filter und die Sortierung aktiviert sind, die entsprechenden Ausdrücke aus den Eigenschaften **Filter** und **OrderBy** in die Variablen **strFilter** und **strOrderBy** ein.

```
Public Sub ExportFlexibel(strExcelpath As String, sfm As Form)
    Dim db As DAO.Database
    Dim qdf As QueryDef
    Dim strQuery As String
    Dim strSQL As String
    Dim strFilter As String
    Dim strOrderBy As String
    Dim strRecordsource As String
    On Error Resume Next
    Kill strExcelpath
    On Error GoTo 0
    Set db = CurrentDb
    strQuery = "qryTemp"
    If Left(sfm.RecordSource, 6) = "SELECT" Then
        strSQL = sfm.RecordSource
    Else
        strSQL = "SELECT * FROM " & sfm.RecordSource
    End If
    If sfm.FilterOn = True Then
        strFilter = sfm.Filter
    End If
    If sfm.OrderByOn = True Then
        strOrderBy = sfm.OrderBy
    End If
    strSQL = AddWhereAndOrderByToSQL(strSQL, strFilter, strOrderBy)
    Set qdf = db.CreateQueryDef(strQuery, strSQL)
    DoCmd.TransferSpreadsheet acExport, acSpreadsheetTypeExcel12Xml, strQuery, strExcelpath, True
    db.QueryDefs.Delete strQuery
End Sub
```

Listing 2: Exportieren der aktuellen Daten des Unterformulars in eine Excel-Tabelle, flexible Variante

SQL-Ausdruck zusammenstellen

Nun folgt eine größere Operation, nämlich die Untersuchung des Inhalts von **strSQL** auf eventuell bereits vorhandene **WHERE**- oder **ORDER BY**-Klauseln und das Einfügen der entsprechenden Klauseln aus dem Unterformular in diese Ausdrücke. Diesen Teil haben wir in die Funktion **AddWhereAndOrderByToSQL** ausgegliedert, der wir drei Informationen übergeben:

- den aktuellen SQL-Ausdruck, also die gegebenenfalls um **SELECT * FROM** erweiterte Datensatzquelle des Unterformulars,
- den Filterausdruck aus **strFilter** und
- den Sortierausdruck aus **strOrderBy**.

Das Ergebnis dieser Funktion, die wir weiter unten beschreiben, landet wieder in der Variablen **strSQL**. Daraus erstellen wir wieder das **QueryDef**-Objekt, das wir als Datenquelle für den Aufruf von **DoCmd.TransferSpreadsheet** nutzen. Danach löschen wir wie zuvor noch die für temporäre Zwecke erstellte Abfrage.

Anreichern eines SQL-Ausdrucks um weitere WHERE- und ORDER BY-Ausdrücke

Warum brauchen wir eine eigene Funktion, um die für das Unterformular angegebenen Filter und Sortierungen hinzuzufügen, statt diese einfach anzuhängen?

ACCESS
IM UNTERNEHMEN

Hier geht es weiter ...

Wenn Sie bis hierhin gelesen haben, könnte ich mir vorstellen, dass Sie der Rest auch noch interessieren würde. Das Beste kommt nämlich meistens erst am Ende!

Für weniger als 50 Cent am Tag* bekommen Sie nicht nur diesen Beitrag, sondern auch noch mehr als tausend weitere Artikel über unser Onlinearchiv auf www.access-im-unternehmen.de. Plus alle zwei Monate 76 Seiten neues Know-how rund um die Programmierung mit Microsoft Access.

Alle Informationen hier – jetzt bestellen und direkt loslesen!

ZUM SHOP

SQL Server im Web, Teil 1: Webserver

Uns erreichen immer mehr Anfragen, wie man seine Datenbank ins Internet bringt. Oder, um genauer zu sein: Wie man diese so verfügbar macht, dass man von mobilen Geräten über das Internet darauf zugreifen kann. Ob man sich nun von einem Notebook aus mit der Datenbank verbindet oder ob man eine alternative Technik wie eine Webseite dafür nutzt. Und gleichzeitig auch noch vom heimischen Rechner oder aus dem Büro über das Access-Frontend auf die Daten zugreifen kann. Also schauen wir uns einmal die Voraussetzung dazu an – einen über das Internet erreichbaren Rechner. Hier haben wir verschiedene Möglichkeiten. Wir können einen Cloud-Service wie den von Microsoft nutzen, aber wir können uns auch einen eigenen Server mieten, auf dem wir tun und lassen können, was wir wollen – SQL Server installieren, Webseiten anlegen et cetera. Wir schauen uns den letzten Punkt an, da er uns maximale Freiheit bietet, aber auch maximale Verantwortlichkeit einfordert: Das Mieten eines Webservers (hier virtuell) und die Verwendung von SQL Server 2022 Express auf diesem Webserver. In diesem ersten Teil schauen wir uns erst einmal an, wie wir einen Webserver mieten und diesen erstmalig mit der Remotedesktopverbindung öffnen und anzeigen können.

Cloud-Dienste, speziell Microsoft

Die erste Alternative, die einem in den Sinn kommt, wenn man eine SQL Server-Datenbank über das Internet verfügbar machen möchte, sind die Cloud-Dienste von Microsoft. Man kann dort beispielsweise eine Azure SQL-Datenbank mieten.

Die Verwaltung solcher Dienste war unserer Erfahrung nach in der Vergangenheit aber immer recht kompliziert, weil Microsoft solche Dienste nicht nur einzeln anbietet, sondern über die Verwaltungsoberfläche sehr viele verschiedene Dienste verwaltet werden können.. Es ist schwer, dort den Überblick zu behalten.

Daher schauen wir uns in diesem Beitrag mit einem SQL Server auf einem eigenen Webserver einmal eine Alternative an. Wir ersparen uns damit den Umgang mit der komplizierten Benutzeroberfläche und haben durch den festen monatlichen Preis für die Servermiete außerdem die Kosten immer genau im Blick.

Webserver: Verschiedene Möglichkeiten

Beim Betreiben eines Webservers gibt es verschiedene Möglichkeiten. Man kann einen Server lokal betreiben und diesen über ein Virtual Private Network (VPN) verfügbar machen und seine Dienste dort anbieten. Oder man mietet einen dafür vorkonfigurierten Server. Hier gibt es wiederum verschiedene Ansätze.

Der erste ist, einen dedizierten Server zu mieten, den man komplett selbst betreut oder den man managen lässt. Der zweite ist ein virtueller privater Server (VPS). Dabei handelt es sich um eine virtuelle Maschine, die gemeinsam mit anderen solchen Maschinen auf einem physischen Server gehostet wird.

Die Unterschiede zwischen den beiden Varianten liegen erstens in der verfügbaren Leistung und dem Speicherplatz, die beim dedizierten Server in der Regel höher sind. Der zweite Unterschied ist der Preis, der VPS in der Regel günstiger ist.

Mit oder ohne Support?

Eine weitere Entscheidung, die man treffen muss, ist der Umfang des Supports. Zwischen umfassender Betreuung und kompletter Eigenverantwortung gibt es verschiedene Angebote. Wer als Anfänger einen Linux-Server zum Hosten seiner Webseiten mietet, ist vermutlich mit einem höheren Grad an Unterstützung besser bedient. Wer eine virtuelle Windows-Maschine zum Experimentieren mieten möchte und ein wenig Eigeninitiative zeigt, kommt vermutlich auch ohne Support aus.

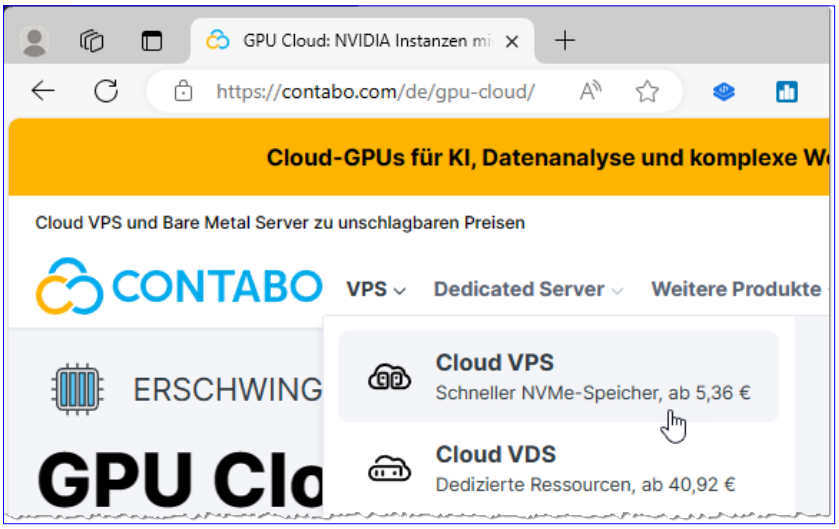

Bild 1: Angebot für einen Cloud VPS

Variante in diesem Beitrag: VPS

Für die Beschreibung der Vorgehensweise in diesem Beitrag mieten wir uns einen virtuellen privaten Server.

Als Anbieter wählen wir **Contabo**, ein Unternehmen, bei dem wir gute Erfahrungen gemacht haben, und das sehr günstige Server anbietet. Außerdem finden wir hier eine

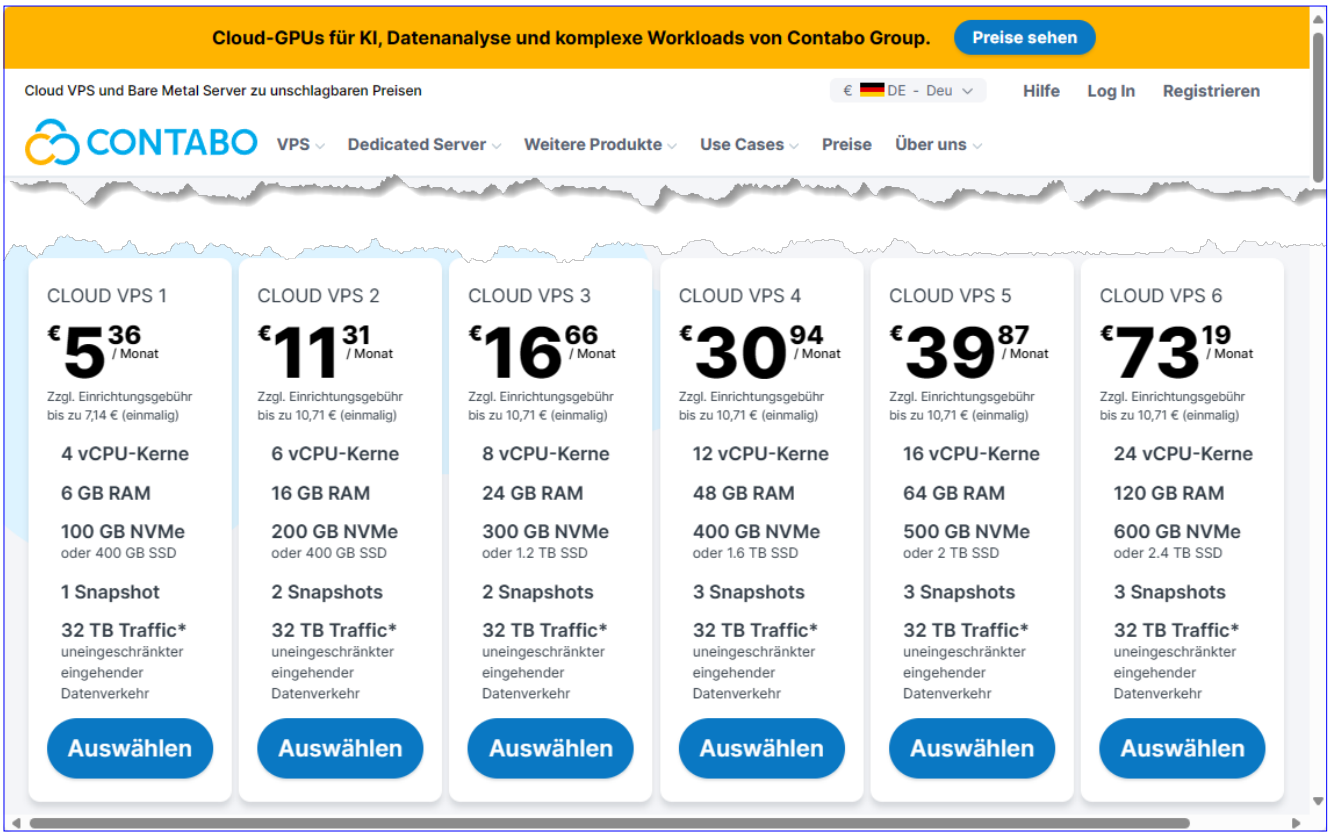

Bild 2: Verschiedene Preisstufen

breite Palette möglicher Betriebssysteme, unter anderem auch Windows.

Hier starten wir über das Basisangebot aus Bild 1. Damit landen wir auf einer Übersichtsseite, die verschiedene Preise für verschiedene Leistungen anbietet.

Hier reicht uns zunächst die kleinste Variante, womit wir bei einem Basispreis von 5,36 EUR pro Monat liegen (siehe Bild 2).

Zum Experimentieren ist das ein mehr als akzeptabler Preis, allerdings sind wir noch nicht beim Endpreis angekommen. Es gibt noch weitere Faktoren:

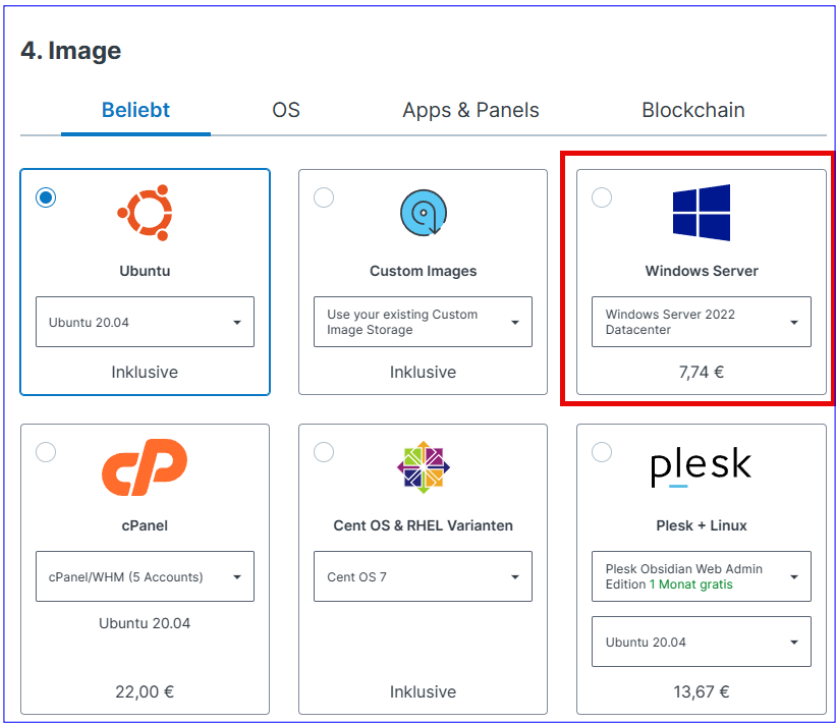

Bild 3: Auswahl des Betriebssystems

- Die initiale Vertragslaufzeit. Wenn wir monatsweise buchen, müssen wir zu Beginn die volle Einrichtungsgebühr von 7,14 EUR bezahlen. Bei der Auswahl von 3 oder 6 Monaten verringert sich dieser Betrag, bei 12 Monaten entfällt die Einrichtungsgebühr.

- Wir können noch die Region auswählen, die sich durch unterschiedliche Latenzzeiten, also Reaktionszeiten, unterscheiden. Wir wählen hier **Europäische Union**.

- Auch die Speicherart können wir auswählen. 400 GB SSD oder 100 GB NVMe sind im Preis enthalten. Wenn wir gleich mit 600 GB SSD oder 150 GB NVMe starten wollen, kostet das einen kleinen monatlichen Aufpreis. 400 GB SSD-Speicher beziehungsweise 100 GB NVMe sollten jedoch reichen. Bleibt also noch die Entscheidung zwischen SSD oder NVMe. Für NVMe sprechen eine höhere Bandbreite, was bei vielen gleichzeitigen Zugriffen wichtig ist, und die niedrigere Latenz. Wenn eine Anwendung auf die Datenbank zugreift, bei der Performance wichtig ist, ist NVMe-Speicher vermutlich sinnvoller.

- Schließlich müssen wir uns noch für ein Betriebssystem entscheiden (siehe Bild 3). Wir wählen hier Windows Server, da wir den Microsoft SQL Server nutzen wollen. Damit erhalten wir einen zusätzlichen Kostenfaktor von 7,74 EUR pro Monat. Hier können wir noch zwischen verschiedenen Windows-Versionen wählen, die jedoch alle die gleichen monatlichen Gebühren auslösen.

- Vor der Bestellung geben wir noch das Kennwort für den Benutzer namens **Administrator** ein. Dieses Kennwort sollten Sie sich gut merken, da es nicht zurückgesetzt werden kann – es ist ein Zurücksetzen des Rechners nötig, um dieses neu zu vergeben.

- Schließlich müssen wir noch angeben, ob wir Private Networking aktivieren möchten. Das erlaubt eine verschlüsselte Verbindung zu dem Server und kostet weitere 2,73 EUR pro Monat. Dies haben wir zunächst ausgelassen, da wir keine sensiblen Daten auf dem Server ablegen wollen.

ACCESS
IM UNTERNEHMEN

Hier geht es weiter ...

Wenn Sie bis hierhin gelesen haben, könnte ich mir vorstellen, dass Sie der Rest auch noch interessieren würde. Das Beste kommt nämlich meistens erst am Ende!

Für weniger als 50 Cent am Tag* bekommen Sie nicht nur diesen Beitrag, sondern auch noch mehr als tausend weitere Artikel über unser Onlinearchiv auf www.access-im-unternehmen.de. Plus alle zwei Monate 76 Seiten neues Know-how rund um die Programmierung mit Microsoft Access.

Alle Informationen hier – jetzt bestellen und direkt loslesen!

ZUM SHOP

SQL Server im Web, Teil 2: SQL Server 2022 Express installieren

Uns erreichen immer mehr Anfragen, wie man seine Datenbank ins Internet bringt. Im ersten Teil dieser Beitragsreihe mit dem Titel »SQL Server im Web, Teil 1: Webserver« haben wir uns angesehen, wie wir einen Webserver beziehungsweise einen Virtual Private Server mieten. Im zweiten Teil dieser Beitragsreihe gehen wir nun einen Schritt weiter und installieren den SQL Server 2022 Express auf unserem frisch gemieteten Webserver. Außerdem nehmen wir diesen in Betrieb und installieren für die ersten Schritte mit dem SQL Server 2022 Express noch das SQL Server Management Studio. Danach prüfen wir, ob wir auch vom lokalen Rechner aus auf den SQL Server 2022 Express auf dem Webserver zugreifen können und schaffen gegebenenfalls die notwendigen Voraussetzungen.

SQL Server 2022 Express herunterladen

Um den SQL Server 2022 Express herunterzuladen, öffnen wir in der Remotedesktopverbindung einen Webbrowser und suchen nach SQL Server 2022 Express. Hier finden wir schnell die Downloadseite und laden das gewünschte Produkt herunter (siehe Bild 1).

Installation von SQL Server 2022 Express starten

Nach dem Download starten wir die Datei **SQL2022-SSEI-Expr.exe**. Diese bietet uns mit dem Bildschirm aus Bild 2 verschiedene Möglichkeiten zum Installieren. Die zweite Option namens **Benutzerdefiniert** erlaubt das Installieren verschiedener Komponenten neben dem eigentlichen

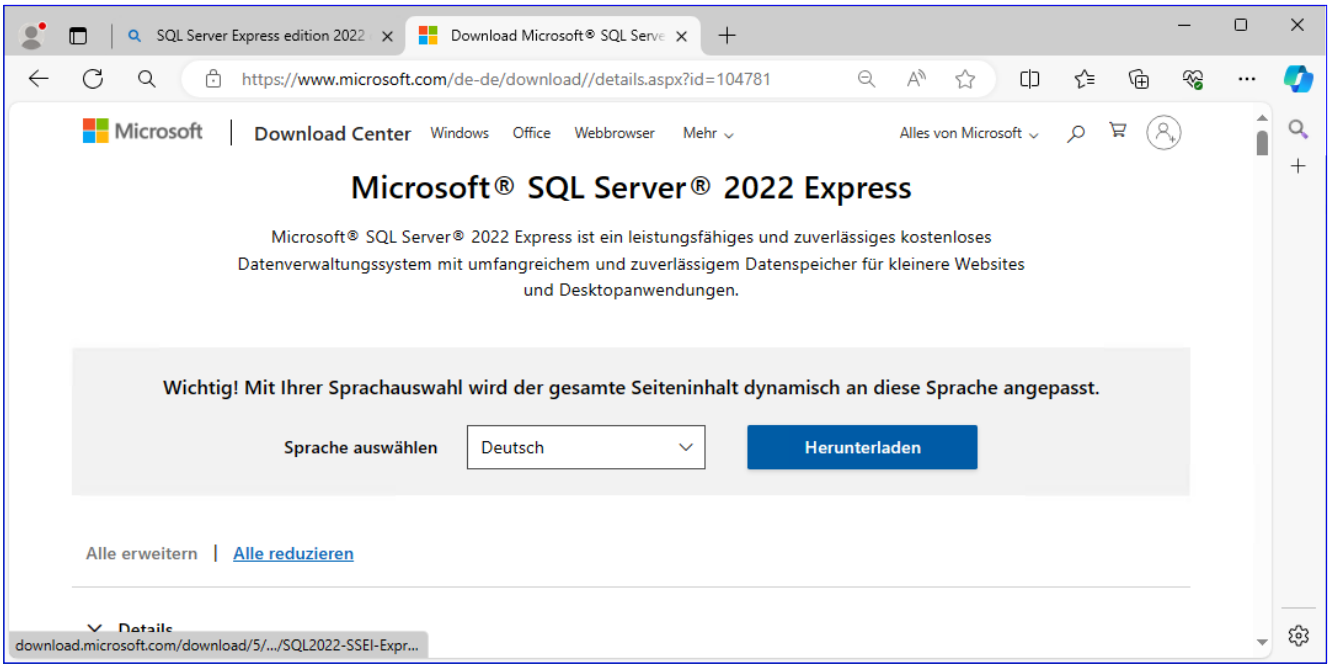

Bild 1: Installieren der SQL Server 2022 Express Edition

SQL Server, zum Beispiel zusätzliche Elemente wie **Reporting Services** oder das **SQL Server Management Studio** (**SSMS**).

Und auch wenn wir den SQL Server hauptsächlich von unserem lokalen Rechner aus administrieren wollen, kann es sinnvoll sein, auch noch das SSMS zu installieren, was wir aber im Anschluss erledigen.

Das SSMS erleichtert beispielsweise einige noch zu erledigende Aufgaben wie das Anlegen eines Benutzer et cetera.

Da wir genauer festlegen möchten, welche Optionen beim Installieren verwendet werden, wählen wir die zweite Option **Benutzerdefiniert** aus und landen im SQL Server-Installationscenter.

Bild 2: Auswahl der Installationsoptionen

Das SQL Server Installationscenter

Im nun erscheinenden Dialog wählen wir den ersten Eintrag aus, also **Neue eigenständige SQL Server-Installation oder Hinzufügen von Funktionen zu einer vorhandenen Installation** aus (siehe Bild 3).

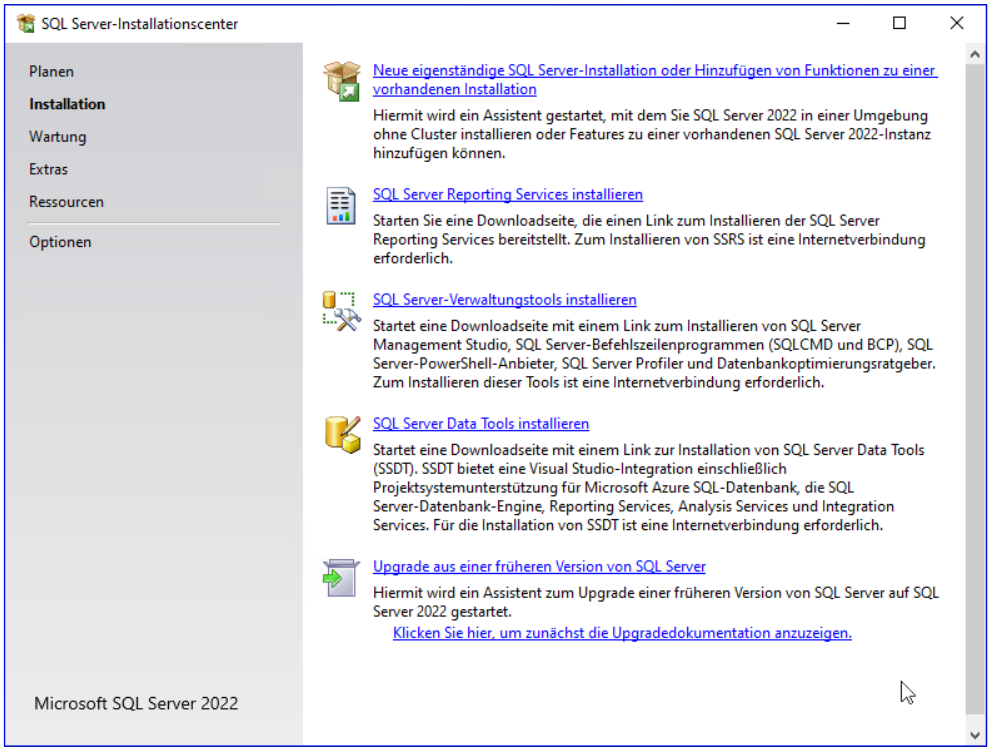

Bild 3: Das SQL Server-Installationscenter

Danach folgen die einzelnen Schritte der Installation:

Den Lizenzbedingungen zustimmen

In diesem ersten Schritt zeigt uns das Setup die Lizenzbedingungen, die wir mit einem Haken im entsprechenden Feld akzeptieren müssen. Mit einem Klick auf die Schaltfläche **Weiter** gelangen wir zum nächsten Schritt (siehe Bild 4).

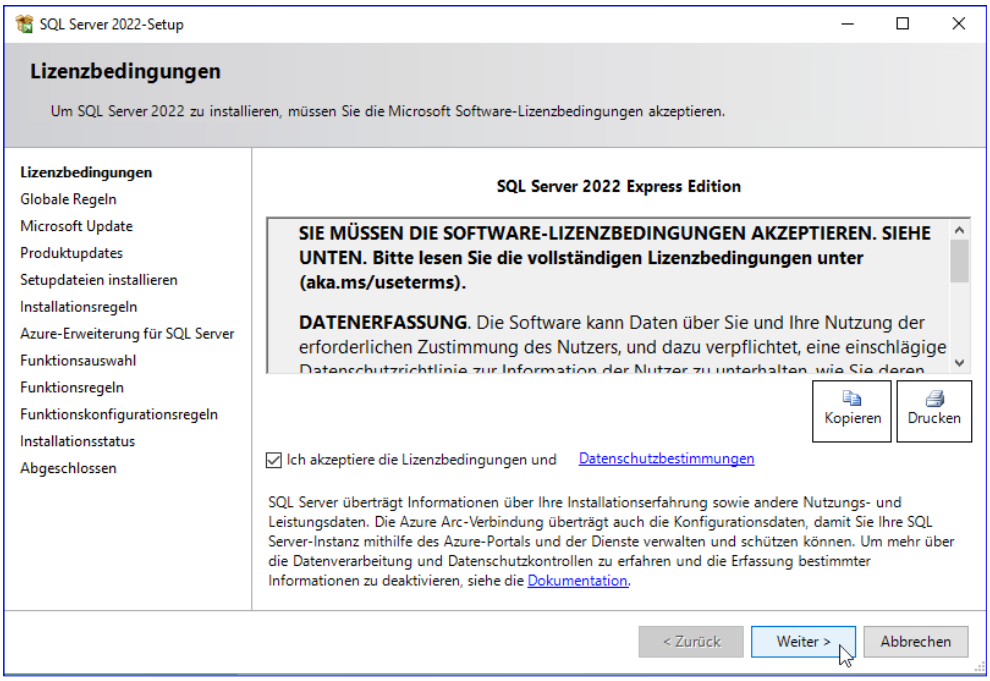

Bild 4: Den Lizenzbedingungen zustimmen

Nach Updates suchen

An dieser Stelle können wir mit einem Haken für die Option **Mit Microsoft Update nach Updates suchen (empfohlen)** dafür sorgen, dass beim Installieren geprüft wird, ob es Updates für den SQL Server 2022 Express gibt. Wir empfehlen, diese Option zu nutzen (siehe Bild 5).

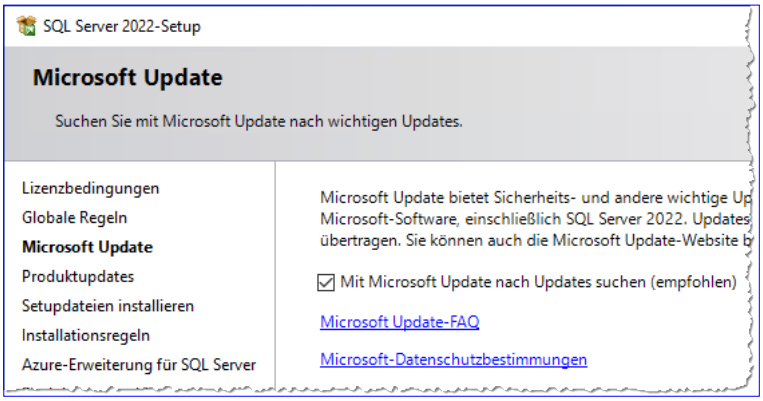

Bild 5: Optional nach Updates suchen

Installationsregeln prüfen

Der nächste Schritt enthält das Prüfen einiger Installationsregeln, also die Untersuchung, ob die Voraussetzungen für die Installation erfüllt sind.

In der Regel wird im entsprechenden Dialog aus Bild 6 immer ein Warndreieck angezeigt. Diese Warnung müssen wir nicht im Rahmen der Installation berücksichtigen. Es kann verschiedene Gründe für diese Warnung geben, die wir jedoch im Anschluss an die Installation untersuchen

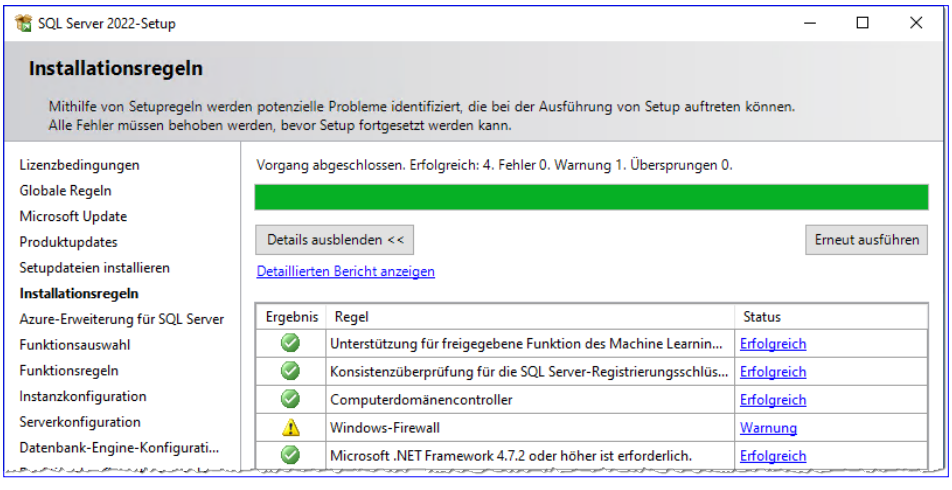

Bild 6: Prüfen der Installationsregeln

ACCESS
IM UNTERNEHMEN

Hier geht es weiter ...

Wenn Sie bis hierhin gelesen haben, könnte ich mir vorstellen, dass Sie der Rest auch noch interessieren würde. Das Beste kommt nämlich meistens erst am Ende!

Für weniger als 50 Cent am Tag* bekommen Sie nicht nur diesen Beitrag, sondern auch noch mehr als tausend weitere Artikel über unser Onlinearchiv auf www.access-im-unternehmen.de. Plus alle zwei Monate 76 Seiten neues Know-how rund um die Programmierung mit Microsoft Access.

Alle Informationen hier – jetzt bestellen und direkt loslesen!

ZUM SHOP

SQL Server im Web, Teil 3: SQL Server Management Studio installieren

Uns erreichen immer mehr Anfragen, wie man seine Datenbank ins Internet bringt. In den ersten beiden Teilen dieser Beitragsreihe mit dem Titel »SQL Server im Web, Teil 1: Webserver« und »SQL Server im Web, Teil 2: SQL Server 2022 Express installieren« haben wir uns angesehen, wie wir einen Webserver beziehungsweise einen Virtual Private Server mieten und darauf den SQL Server 2022 Express installieren. Im dritten Teil dieser Beitragsreihe installieren wir nun das SQL Server Management Studio auf unserem Webserver. Damit wollen wir die Administration des SQL Servers direkt von seinem Serverrechner aus ermöglichen, auf den wir zu diesem Zweck über die Remotedesktopverbindung zugreifen.

SQL Server Management Studio installieren

SQL Server Management Studio finden wir wie in Bild 1, wenn wir im Internet nach diesem Suchbegriff suchen – ergänzt um **Download** und **Deutsch** (wenn es die deutsche Version sein soll). Wir finden dann den Download einer Datei etwa namens **SSMS-Setup-DEU.exe** vor, die wir direkt starten. Das Setup fragt den gewünschten Speicherort für die Anwendung ab (siehe Bild 2) und beginnt nach einem Klick auf **Installieren** bereits die Installation.

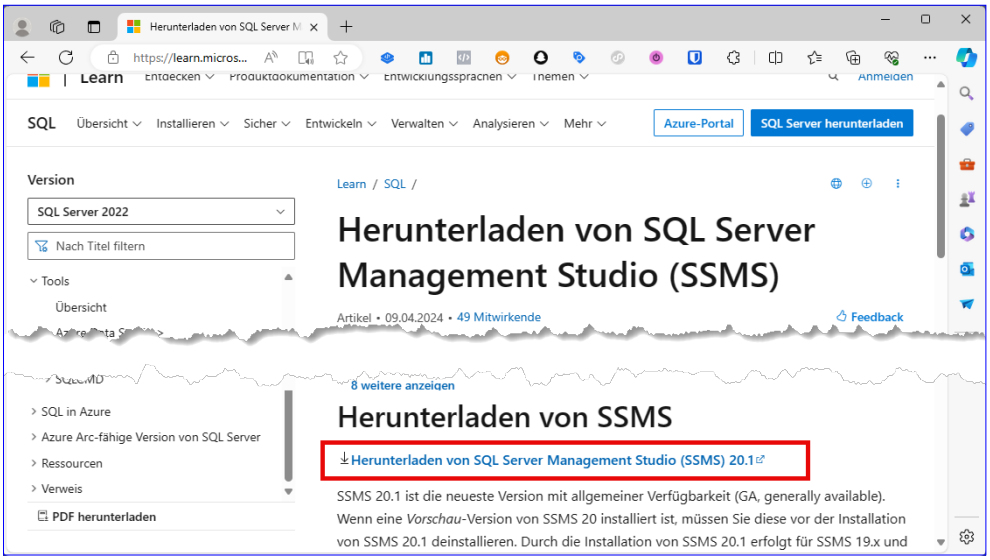

Bild 1: Installieren des SQL Server 2022 Express Edition

Einige Augenblicke später können wir die Installation bereits als erledigt betrachten. Woran erkennen wir, dass die Installation erfolgreich war? Dazu brauchen wir nur in der Windows-Suche den Begriff **SQL Server** einzugeben. Hier finden wir gleich einige Einträge. Uns interessiert der Eintrag aus Bild 3, dem wir auch gleich nochmal entnehmen können, dass wir es mit der Version 20 zu tun haben.

SQL Server Management Studio starten

Nach dem Start erscheint SQL Server Management Studio erst einmal ohne jegliche Elemente im Objekt-Explorer. Dem wollen wir Abhilfe schaffen, indem wir uns mit der lokalen SQL Server-Instanz verknüpfen. Dazu klicken wir auf die Schaltfläche **Verbinden**, was weitere Einträge öffnet. Hier wählen wir den Befehl **Datenbank-Engine...** aus (siehe Bild 4).

Anschließend erscheint der Dialog **Verbindung mit Server herstellen** aus Bild 5. Hier finden wir verschie-

ACCESS
IM UNTERNEHMEN

Hier geht es weiter ...

Wenn Sie bis hierhin gelesen haben, könnte ich mir vorstellen, dass Sie der Rest auch noch interessieren würde. Das Beste kommt nämlich meistens erst am Ende!

Für weniger als 50 Cent am Tag* bekommen Sie nicht nur diesen Beitrag, sondern auch noch mehr als tausend weitere Artikel über unser Onlinearchiv auf www.access-im-unternehmen.de. Plus alle zwei Monate 76 Seiten neues Know-how rund um die Programmierung mit Microsoft Access.

Alle Informationen hier – jetzt bestellen und direkt loslesen!

ZUM SHOP

SQL Server im Web, Teil 4: Fernzugriff per SSMS

In den ersten drei Teilen dieser Beitragsreihe haben wir einen Webserver aufgesetzt, den SQL Server installiert und das SQL Server Management Studio hinzugefügt. Damit können wir über die Remotedesktopverbindung auf den Server zugreifen und dort den SQL Server administrieren. Wir wollen aber nicht immer die Remotedesktopverbindung nutzen, sondern direkt von unserem lokalen Rechner aus auf den SQL Server auf dem Webserver zugreifen. Das wird schon etwas spannender als die Einrichtung und der Zugriff vom SQL Server Management Studio direkt auf dem Webserver. Warum? Weil wir unterschiedliche Ports verwenden müssen und diese je nach Anwendungsfall standardmäßig freigeschaltet sind oder auch nicht. Was es damit auf sich hat, erläutern wir in diesem Beitrag.

TCP/IP und die Ports

Netzwerkverbindungen werden seit gefühlten Ewigkeiten über das TCP/IP-Protokoll durchgeführt. Damit wir beispielsweise von unserem lokalen Rechner überhaupt auf den Webserver zugreifen können, egal ob wir über die Remotedesktopverbindung die Benutzeroberfläche sehen wollen oder vom SQL Server Management Studio auf dem lokalen Rechner Daten einer SQL Server-Datenbank lesen wollen, müssen verschiedene Dienste auf dem Webserver auf verschiedenen Kanälen auf Anfragen von unserem lokalen Rechner »lauschen«.

Damit man nicht beliebig von außen auf unser System zugreifen kann, gibt es die sogenannte Firewall, die den Zugriff reguliert. Ist die Firewall aktiviert und sind keine Kanäle geöffnet, können wir praktisch nicht von außen über das Netzwerk auf den Webserver zugreifen.

Die Firewall sollte immer aktiviert sein, aber es sollten nicht alle Kanäle geöffnet sein. Diese Kanäle werden Ports genannt, jeder mit einer bestimmten Nummer versehen, die wir für den Zugriff von außen öffnen können.

Einige dieser Ports sind standardmäßig geöffnet, andere müssen wir erst noch freigeben. Die Ports sind wiederum verschiedenen Diensten zugeordnet. Für Remoteverbindungen verwendet Windows beispielsweise den Port **3389**.

Während dieser Port für den Zugriff über das Programm **Remotedesktopverbindung** standardmäßig geöffnet ist, müssen wir den Port für den Zugriff auf den SQL Server erst noch freigeben. Dieser Port hat normalerweise die Nummer **1433**. Man kann aber auch einen anderen Port nutzen, beispielsweise um Zugriffe zu erschweren.

Die Nummer **1433** ist üblicherweise der Standardinstanz des SQL Servers vorbehalten. Weitere Instanzen erhalten je nach Konfiguration dynamisch Ports zugewiesen.

Das ist aber für uns erst einmal uninteressant, da wir den SQL Server 2022 Express wie im Beitrag **SQL Server im Web, Teil 2: SQL Server Express installieren** (**www.access-im-unternehmen.de/1504**) beschrieben als Standardinstanz installiert haben.

Verbindung zum SQL Server vom Client-Rechner aufbauen

Wir können uns zwar nun per Remotedesktopverbindung auf den VPS-Rechner aufschalten, aber eigentlich wollen wir dort eine SQL Server-Datenbank erstellen, auf die wir vom Client-Rechner oder auch von anderen Geräten aus

zugreifen können. Wir probieren das erst einmal ganz naiv aus, ohne irgendwelche Änderungen am System vorzunehmen.

Dazu starten wir SQL Server Management Studio und finden direkt den Anmeldedialog aus Bild 1 vor. Hier geben wir unter **Servername** die IP des Webservers ein. Bei der Authenitifizierungsmethode bleiben uns nicht viele Möglichkeiten: Da der Webserver nicht in der gleichen Domäne liegt wie unser Rechner, müssen wir die SQL Server-Authentifizierung wählen.

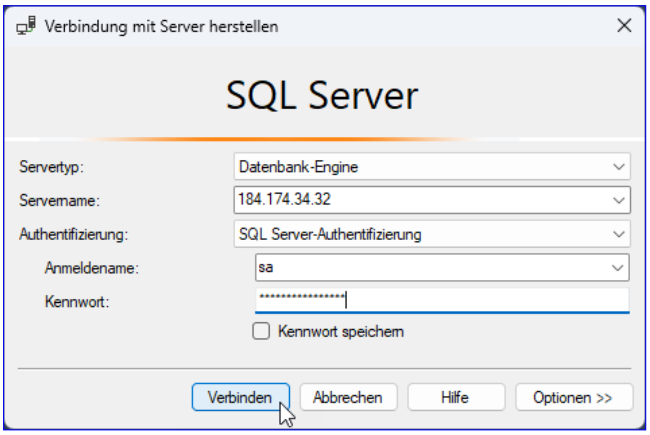

Bild 1: Der Verbindungsdialog

Bei der Installation wurde automatisch der Benutzer **sa** mit SysAdmin-Rechten angelegt, den wir hier für die Anmeldung benutzen wollen. Außerdem geben wir das passende Kennwort ein und betätigen dann die **Verbinden**-Schaltfläche.

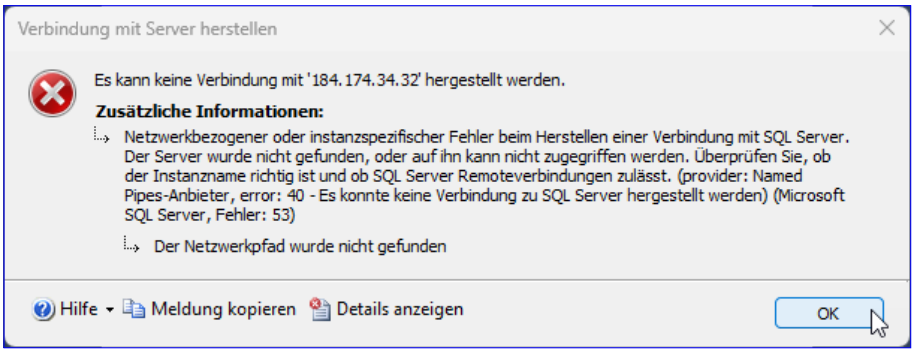

Bild 2: Fehlermeldung beim Versuch, eine Verbindung aufzubauen

Es kann sein, dass noch nicht alle Voraussetzungen für die Verbindung erfüllt sind. Dann erscheint beispielsweise eine Meldung wie die aus Bild 2. Hier sehen wir, dass der Server nicht gefunden wurde oder das nicht auf diesen zugegriffen werden kann.

Anforderungen an die Remote-Verbindung

Also schauen wir uns an, ob die Anforderungen an eine Remote-Verbindung zum SQL Server erfüllt sind. Dabei konzentrieren wir uns auf die folgenden beiden:

Das wir grundsätzlich auf den Webserver zugreifen können, wissen wir bereits – wir haben, wie im Beitrag **SQL Server im Web, Teil 1: Webserver** (**www.access-im-unternehmen.de/1503**) beschrieben, schon per Remotedesktopverbindung auf den Server zugegriffen.

Wir können das nochmal testen, indem wir die Eingabeaufforderung öffnen und dort den Befehl **Ping** mit der Angabe der IP des Webservers absetzen. Dies liefert das Ergebnis aus Bild 3.

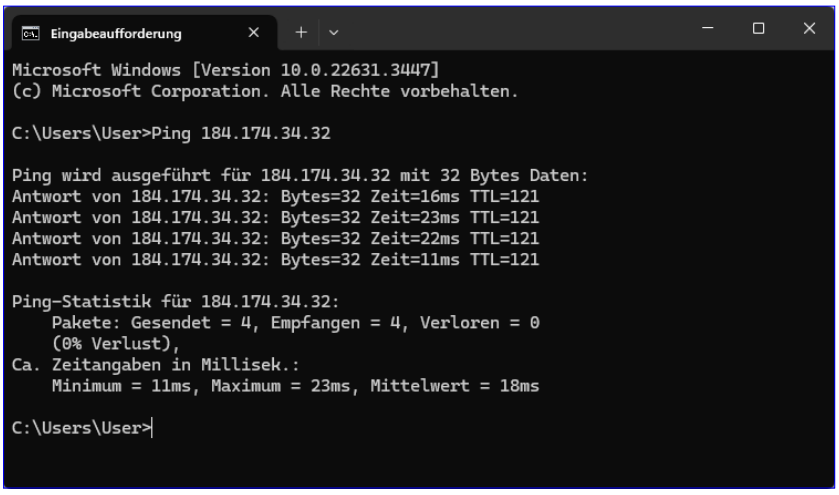

Bild 3: Die Verbindung zum Webserver funktioniert grundsätzlich.

- TCP/IP-Protokoll für den SQL Server aktivieren
- Port 1433 in der Firewall freischalten

Diese beiden Schritte führen wir direkt auf dem Webserver aus. Dazu müssen wir uns, wie im oben genannten Beitrag beschrieben, per Remotedesktopverbindung auf den Webserver schalten.

Es kann noch weitere Gründe geben, dass eine Verbindung nicht möglich ist – diese sind jedoch so individuell, dass wir sie nicht an dieser Stelle beschreiben.

TCP/IP-Protokoll für SQL Server aktivieren

Um das TCP/IP-Protokoll für den SQL Server zu aktivieren, starten wir auf dem Webserver den SQL Server 2022-Konfigurations-Manager. Diesen finden wir wie in Bild 4 durch Eingabe von **SQL Server** in der Windows-Suche.

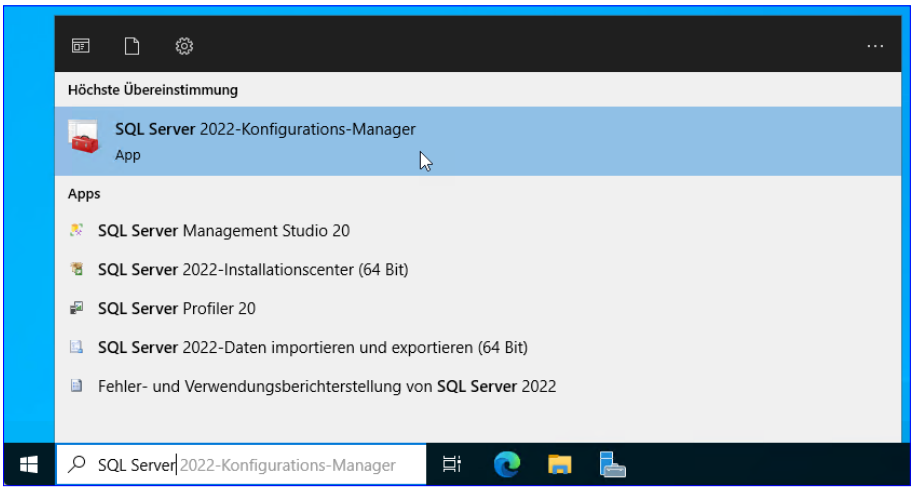

Bild 4: Starten des SQL Server 2022-Konfigurations-Managers

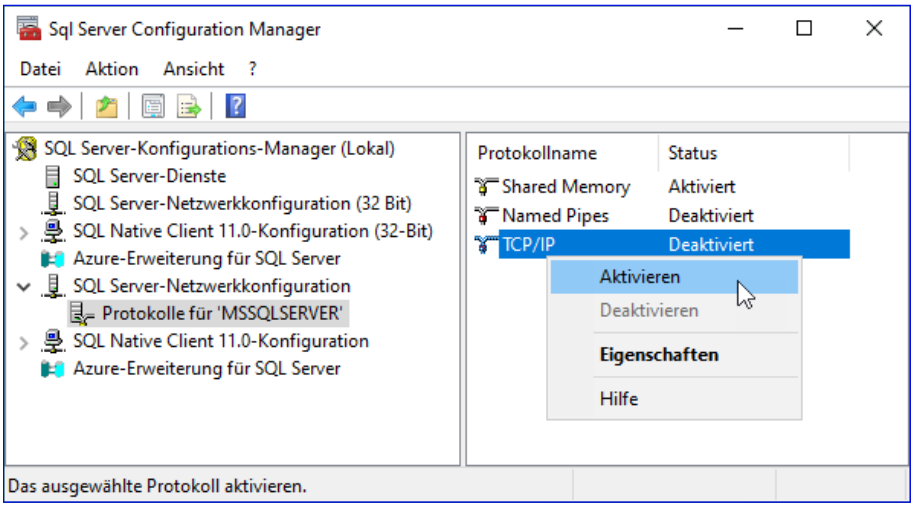

Bild 5: Aktivieren des TCP/IP-Protokolls

Nach dem Start navigieren wir im Fenster **SQL Server Configuration Manager** zum Eintrag **Protokolle für 'MSSQLSERVER'**. Hier finden wir den Eintrag **TCP/IP**, der gegebenenfalls noch deaktiviert ist. Diesen aktivieren wir über den Befehl **Aktivieren** des Kontextmenüs (siehe Bild 5).

Danach erscheint eine Meldung mit dem Hinweis, dass der Dienst neu gestartet werden muss (siehe Bild 6).

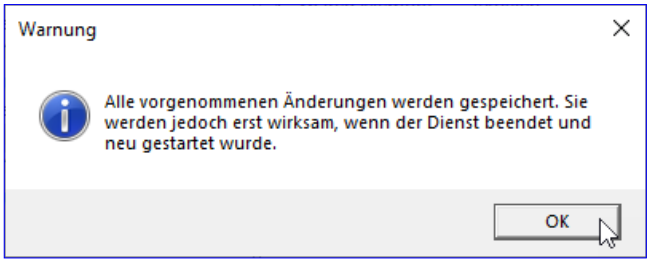

Bild 6: Ein Neustart ist nötig.

Diesen Neustart können wir ebenfalls im SQL Server 2022-Konfigurations-Manager erledigen. Dazu klicken wir oben auf **SQL Server-Dienste** und sehen dann in der rechten Liste den Eintrag **SQL Server (MSSQLSERVER)** (siehe Bild 7). Hier führen wir den Befehl **Neu starten** aus dem Kontextmenü aus.

ACCESS
IM UNTERNEHMEN

Hier geht es weiter ...

Wenn Sie bis hierhin gelesen haben, könnte ich mir vorstellen, dass Sie der Rest auch noch interessieren würde. Das Beste kommt nämlich meistens erst am Ende!

Für weniger als 50 Cent am Tag* bekommen Sie nicht nur diesen Beitrag, sondern auch noch mehr als tausend weitere Artikel über unser Onlinearchiv auf www.access-im-unternehmen.de. Plus alle zwei Monate 76 Seiten neues Know-how rund um die Programmierung mit Microsoft Access.

Alle Informationen hier – jetzt bestellen und direkt loslesen!

ZUM SHOP