

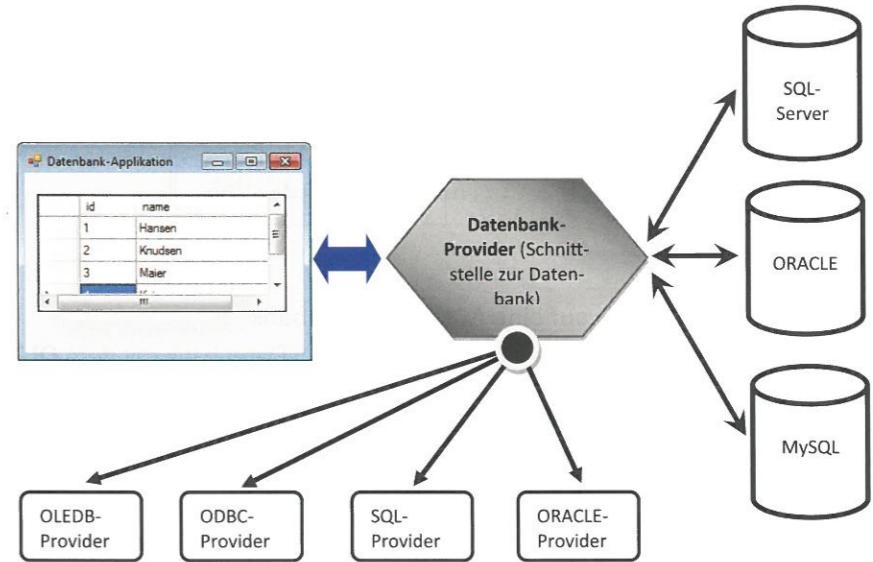
10 Datenbankzugriff mit .NET und C#

10.1 Datenbankzugriff mit .NET und C#

Die Programmiersprache C# bietet eine Vielzahl von Dateioperationen, um Daten dauerhaft zu speichern oder zu lesen. Bei vielen oder auch komplexen Daten ist es sinnvoll, die Speicherung in einer Datenbank in Betracht zu ziehen (wie bei der Programmiersprache Java im vorherigen Kapitel). Das .NET-Framework bietet dazu komfortable Möglichkeiten. Die Abfragesprache **SQL** spielt auch hier wieder eine wichtige Rolle.

10.1.1 Datenbankanbindung unter dem .NET-Framework

Das .NET-Framework bietet eine Vielzahl von Klassen, um die Anbindung an eine Datenbank zu realisieren. Diese Klassen sind unter dem Oberbegriff **ADO.NET** gesammelt. Dabei steht ADO für *ActiveX Data Objects* und ist eine Erweiterung der bereits vorhandenen Technik von Microsoft. Mit ADO.NET kann ein Zugriff auf Datenquellen wie **SQL-Server** oder auch auf **OLE DB**- und **ODBC**-Datenquellen erfolgen. Die folgende Abbildung zeigt das Grundprinzip von ADO.NET:

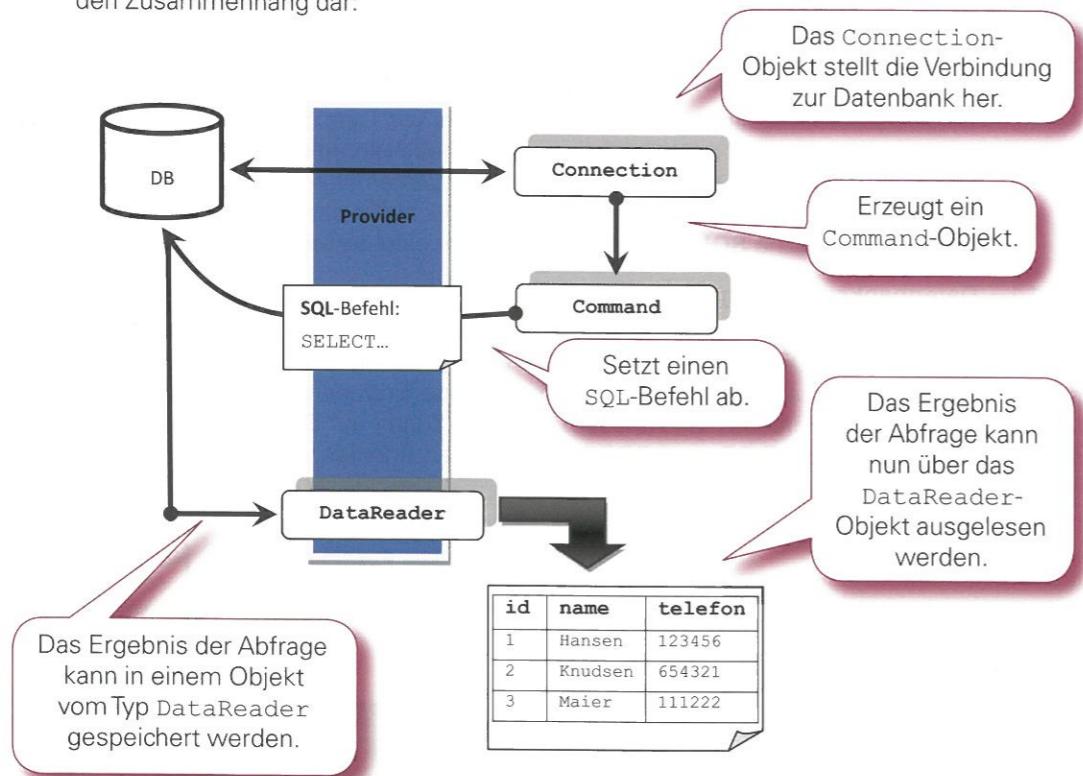


Die einzelnen Provider (Datenanbieter) stehen dabei für bestimmte Datenbankanbindungen:

- **OLE DB-Provider:** OLE DB steht für **Object Linking and Embedding Database** und ist eine Technik, die bei den Microsoft-Office-Anwendungen zum Einsatz kommt. Beispielsweise ist es möglich eine Excel-Tabelle in ein Word-Dokument so einzubinden, dass Änderungen an der Original-Tabelle auch immer in der Word-Tabelle sichtbar sind (und umgekehrt). Der OLE DB-Provider kann immer dann angewendet werden, wenn für eine Datenbank ein solcher Provider zu Verfügung steht (z. B. ACCESS).
- **ODBC-Provider:** ODBC steht für **Open Database Connectivity** und war eine der ersten Schnittstellen, die eine Vereinheitlichung des Datenbankzugriffs umsetzte. Jede Datenbank braucht nur eine ODBC-Schnittstelle mitzuliefern und ist damit für eine Windows-Anwendung einsetzbar.
- **SQL-Provider:** Dieser Provider stellt die Funktionalitäten für einen Zugriff auf den Microsoft SQL-Server zu Verfügung.
- **ORACLE-Provider:** Dieser Provider stellt die Funktionalitäten für einen Zugriff auf die ORACLE-Datenbank zu Verfügung.

10.1.2 Provider nutzen und eine Verbindung aufbauen

Um eine Verbindung zu einer Datenbank aufzubauen, muss der entsprechende Datenbankprovider vorliegen. Einige Treiber sind bereits in der Standard-Installation von **Visual C#** vorhanden (wie der **OLE DB-Provider**). Andere Provider müssen von den jeweiligen Herstellern bezogen und installiert werden. Die eigentliche Verbindung wird dann mit einem Objekt der Klasse **Connection** hergestellt. Je nach Datenbank sind Nutzernamen und Passwort anzugeben. Über ein **Command**-Objekt kann dann eine Abfrage gestartet und mit einem **DataReader**-Objekt ausgelesen werden. Die folgende Abbildung stellt den Zusammenhang dar:



10.1.3 Beispiel eines Zugriffs auf eine ACCESS-Datenbank

- Im Folgenden wird der Zugriff auf eine ACCESS-Datenbank mit dem **OLE DB-Provider** vorgestellt. Das Prinzip ist übertragbar auf andere relationale Datenbanksysteme wie beispielsweise den Microsoft-SQL-Server oder MySQL/MariaDB. Die Verbindung zur Datenbank wird mit einem **OleDbConnection**-Objekt aufgebaut, ein SQL-Befehl mit einem **OleDbCommand**-Objekt abgesetzt und das Ergebnis mit einem Objekt vom Typ **OleDbDataReader** ausgelesen.
- Als Entwicklungsumgebung wird in den folgenden Beispielen die kostenfreie Edition **Visual Studio Community** von Microsoft verwendet.

Der folgende Quellcode zeigt den Verbindungsaufbau zu einer ACCESS-Datenbank „*Kunden.accdb*“, die in einem Ordner (hier: *C:\temp*) zur Verfügung steht. Sie verfügt über eine Beispieldatenebene *Kunden* mit den Attributen *id* (Typ Zahl) und *name*, *strasse*, *ort* und *telefon* (jeweils Text):

Kunden				
<i>id</i>	<i>name</i>	<i>strasse</i>	<i>ort</i>	<i>telefon</i>
1	Hansen	Baumallee 1	Hamburg	123456
2	Knudsen	Sonnenstr.4	Berlin	654321
3	Albers	Paulistr. 8	Hamburg	111222

Datensatz: 1 von 3 ▶ Kein Filter Suchen

```

using System;
using System.Data;
using System.Data.OleDb;

namespace DB_Zugriff_CSharp
{
    class CDBZugriff
    {
        static void Main(string[] args)
        {
            string verbindungsstring =
                "Provider=Microsoft.ACE.OLEDB.12.0;
                Data Source=C:\\Temp\\Kunden.accdb";

            OleDbConnection dBVerbindung = null;
            OleDbCommand befehl = null;
            OleDbDataReader datenleser = null;
            bool offen = false;

            WICHTIG:
            Fehlerbehandlung
            try
            {
                dBVerbindung =
                    new OleDbConnection(verbindungsstring);
                dBVerbindung.Open();
                offen = true;
                Flag setzen
                Ein Verweis auf eine OLEDB-Verbindung.

                befehl =
                    dBVerbindung.CreateCommand();
                befehl.CommandText =
                    "SELECT * FROM Kunden";
                Eine Datenleser-Instanz auf der Grundlage
                des SQL-Befehls erstellen lassen.

                datenleser =
                    befehl.ExecuteReader();
                Sequenzielles Auslesen
                des Datenlesers
                while (datenleser.Read())
                {
                    Console.WriteLine("Name: "
                        + datenleser.GetString(1));
                }
            }
            Ein Verweis
            auf ein OLE
            DB-Kommando.

            Ein Verweis
            auf einen OLE DB-
            Datenleser.

            Eine Verbin-
            dungs-Instanz

            Datenbank
            öffnen

            Ein Befehlsobjekt
            erstellen lassen.

            SQL-Befehl (alles aus
            der Tabelle auswählen)
            zuweisen.

            Die Methode GetString() liefert
            den Wert der aktuellen Zeile und vom
            übergebenen Spaltenindex.
        }
    }
}
  
```

Der Quellcode zeigt den Verbindungsaufbau zu einer ACCESS-Datenbank „Kunden.accdb“. Er verwendet den Microsoft.ACE.OLEDB.12.0-Provider. Der Verbindungsstring definiert die Datenquelle als „C:\Temp\Kunden.accdb“. Der Klassencode besteht aus einer Klasse CDBZugriff mit einer statischen Methode Main. Diese Methode erstellt eine OleDbConnection-Instanz (dBVerbindung), öffnet sie und erstellt eine OleDbCommand-Instanz (befehl) mit dem SQL-Befehl „SELECT * FROM Kunden“. Danach wird ein OleDbDataReader-Instanz (datenleser) über den ExecuteReader-Methode des Commands erstellt. Schließlich wird ein Loop durchlaufen, der die Zeilen des Datenlesers mit Hilfe der GetString-Methode des Readers ausgibt. Die Kommentare im Code sind durch rote Pfeile mit den entsprechenden Begriffen im Diagramm verbunden.

```

    catch (Exception ausnahme)
    {
        Console.WriteLine("Datenbankfehler: "
            + ausnahme.Message);
    }
    finally
    {
        if (offen == true) dBVerbindung.Close();
    }
}
}

```

Falls die boolesche Variable offen den Wert true hat, wird die Verbindung geschlossen.

Nach dem Starten wird die Kundentabelle ausgelesen und mithilfe des Datenlesers werden Schritt für Schritt die Werte der 2. Spalte (Index 1) ausgegeben:

Hinweise:

Der Zugriff auf die Spaltenwerte einer Tabelle mit dem Datenleser erfolgt in Abhängigkeit vom jeweiligen Datentyp. Für jeden Datentyp steht eine geeignete Methode zur Verfügung:

- ▶ GetDateTime(Spaltenindex)
- ▶ GetString(Spaltenindex)
- ▶ GetInt32(Spaltenindex)
- ▶ ... weitere Typen

Beispielsweise kann die erste Spalte der Kunden-Tabelle mit der Methode GetInt32() ausgelesen werden, da es sich um einen ganzzahligen numerischen Typen (ACCESS-Typ Zahl) handelt:

```

while (datenleser.Read())
{
    Console.WriteLine("Erste Spalte: "
        + datenleser.GetInt32(0));
}

```

Die erste Spalte vom ACCESS-Feldtyp Zahl auslesen.

10.1.4 Nicht-Select-Befehle absetzen

Das Auslesen einer beliebigen Tabelle kann mithilfe der oben beschriebenen Anweisungen erfolgen. Möchte man hingegen nicht selektieren, sondern einfügen, ändern oder löschen, so kann ein **ExecuteNonQuery**-Befehl abgesetzt werden. Vorher wird der gewünschte SQL-Befehl in einer Zeichenkette erstellt. In dem folgenden Beispiel wird eine neue Zeile in die Kundentabelle eingefügt, eine bestehende Zeile geändert und eine Zeile gelöscht:

```

using System;
using System.Data;
using System.Data.OleDb;

```

```

namespace DB_Zugriff_CSharp
{

```

```

    class CDBZugriff
    {

```

```

        static void Main(string[] args)
        {

```

```

            string verbindungsstring =
                "Provider=Microsoft.ACE.OLEDB.12.0;
                 Data Source=C:\\Temp\\Kunden.accdb";
            OleDbConnection dBVerbindung = null;
            OleDbCommand befehl = null;
            bool offen = false;
            int anzahl=0;
            try
            {

```

Den Verbindungsstring mit der Provider-Angabe und der Datenquelle festlegen.

```

                dBVerbindung = new
                    OleDbConnection(verbindungsstring);
                dBVerbindung.Open();
                offen = true;
                befehl = dBVerbindung.CreateCommand();

```

Der SQL-Befehl, um eine Zeile einzufügen.

```

                befehl.CommandText = "INSERT INTO Kunden
                                     VALUES (4,'König','Seestr. 5',
                                             'Hamburg','45621');");

```

SQL-Befehl absetzen und die Anzahl der betroffenen Zeilen zurückhalten.

```

                anzahl = befehl.ExecuteNonQuery();
                Console.WriteLine("Anzahl der eingefügten Zeilen: "
                    + anzahl);

```

Ein UPDATE-Befehl

```

                befehl.CommandText = "UPDATE Kunden SET telefon =
                    '11111' WHERE name = 'Hansen';";

```

```

anzahl = befehl.ExecuteNonQuery();
Console.WriteLine("Anzahl der geänderten Zeilen: "
+ anzahl);

Ein DELETE-Befehl

befehl.CommandText = "DELETE FROM Kunden
WHERE name = 'Knudsen';";

anzahl = befehl.ExecuteNonQuery();
Console.WriteLine("Anzahl der gelöschten Zeilen: "
+ anzahl);
Console.WriteLine();

}

Schließen der Datenbank-Verbindung.

finally
{
    if (offen == true) dBVerbindung.Close();
}
}
}
}

Nach dem Starten werden die drei Nicht-Select-SQL-Befehle abgesetzt und die Anzahl der betroffenen Zeilen wird nach jedem Befehl ausgegeben:

```

```
C:\WINDOWS\system32\cmd.exe
Anzahl der eingefügten Zeilen: 1
Anzahl der geänderten Zeilen: 1
Anzahl der gelöschten Zeilen: 1

Drücken Sie eine beliebige Taste . . .
```

Zum Vergleich: Die Kundentabelle vor und nach den SQL-Befehlen:

Vorher:

	id	name	strasse	ort	telefon
	1	Hansen	Baumallee 1	Hamburg	123456
	2	Knudsen	Sonnenstr.4	Berlin	654321
	3	Albers	Paulistr. 8	Hamburg	111222

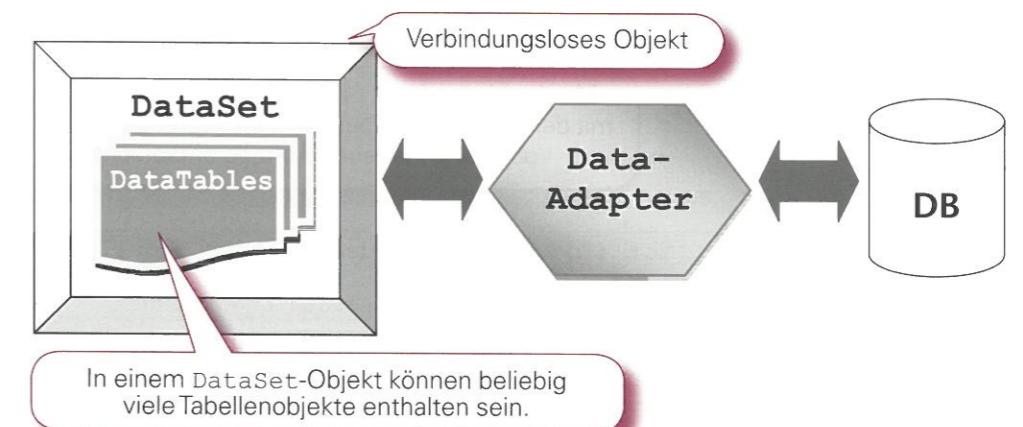
Nachher:

	id	name	strasse	ort	telefon
	1	Hansen	Baumallee 1	Hamburg	11111
	3	Albers	Paulistr. 8	Hamburg	111222
	4	König	Seestr. 5	Hamburg	45621

Der neue Datensatz erhält von ACCESS automatisch die ID 4. ACCESS achtet nicht auf lückenlose IDs nach dem Löschen und Einfügen von Datensätzen.

10.1.5 DataAdapter und DataSet

Bislang wurde die Datenbank geöffnet und sequenziell abgefragt oder über entsprechende SQL-Befehle modifiziert. Diese Vorgehensweise ist praktikabel, aber nicht sehr komfortabel. Aus diesem Grund gibt es die Möglichkeit, das komplette Ergebnis der Abfrage in einem speziellen Objekt der Klasse **DataSet** zu speichern. Dieses Objekt kann dann unabhängig von der Datenbank bearbeitet werden und erst im Anschluss findet eine Synchronisierung mit der Datenbank statt. Deshalb spricht man bei solchen Objekten auch davon, dass sie **verbindungslos** sind. Für den korrekten Austausch der Daten zwischen diesen Objekten und der Datenbank sorgen dann **Adapter-Objekte**. Die folgende Abbildung verdeutlicht diesen Zusammenhang:



Das folgende Beispiel zeigt die Verwendung von **DataSet** und **DataAdapter**:

```

using System;
using System.Data;
using System.Data.OleDb;

Einbinden der benötigten Namensräume.

```

```

namespace DB_Zugriff_CSharp
{
    class CDBZugriff
    {
        static void Main(string[] args)
        {
            string verbindungsstring =
                "Provider=Microsoft.ACE.OLEDB.12.0;
                Data Source=C:\\\\Temp\\\\Kunden.accdb";

```

Den Verbindungsstring mit der Provider-Angabe und der Datenquelle festlegen.

```

OleDbConnection dBVerbindung = null;
OleDbCommand befehl = null;
bool offen = false;

try
{
    dBVerbindung =
        new OleDbConnection(verbundungsstring);
    dBVerbindung.Open();
    offen = true;

    befehl = dBVerbindung.CreateCommand();
    befehl.CommandText = "SELECT * FROM Kunden";
}

Ein OleDbDataAdapter-Objekt instanzieren und
das OleDbCommand-Objekt übergeben.

```

```
OleDbDataAdapter da = new OleDbDataAdapter(befehl);
```

```
DataSet ds = new DataSet();
da.Fill(ds);
```

Ein DataSet-Objekt anlegen.

Mit der Methode Fill() wird das DataSet-Objekt mit den Datensätzen aus der Datenbank gefüllt, die der oben angegebenen Abfrage entsprechen.

```

for (int i = 0; i < ds.Tables[0].Rows.Count; i++)
{
    Console.WriteLine(
        ds.Tables[0].Rows[i]["name"].ToString());
}

```

Das Ergebnis der Abfrage kann über das Tables-Array des DataSet-Objektes angesprochen werden. Zusätzlich wird die gewünschte Zeile und Spalte angegeben:

```
C:\WINDOWS\system32\cmd.exe
Name: Hansen
Name: Knudsen
Name: Albers
Drücken Sie eine beliebige Taste . . .
```

```
ds.Tables[0].Rows[0]["name"] = "Laufer";
```

Eine Änderung der Daten erfolgt dann über eine einfache Zuweisung!

```

DataRow zeile = ds.Tables[0].NewRow();
zeile["id"] = 10;
zeile["name"] = "Kaiser";
ds.Tables[0].Rows.Add(zeile);

```

Das Hinzufügen eines Datensatzes erfolgt mit einem DataRow-Objekt und der Methode Add().

```
ds.Tables[0].Rows[1].Delete();
```

Das Löschen erfolgt über die Delete-Methode!

```

OleDbCommandBuilder cmb =
    new OleDbCommandBuilder(da);
da.Update(ds);

```

Die Synchronisierung mit der Datenbank erfordert ein CommandBuilder-Objekt, das die erforderlichen Update-Befehle zur Verfügung stellt.

```

}
catch (Exception ausnahme)
{
    Console.WriteLine("Datenbankfehler: "
        + ausnahme.Message);
}
finally
{
    if (offen == true) dBVerbindung.Close();
}
}
}
```

Nach der Synchronisierung sieht die Kunden-Tabelle so aus:

Kunden					
	id	name	strasse	ort	telefon
	1	Laufer	Baumallee 1	Hamburg	123456
	3	Albers	Paulistr. 8	Hamburg	111222
	10	Kaiser			

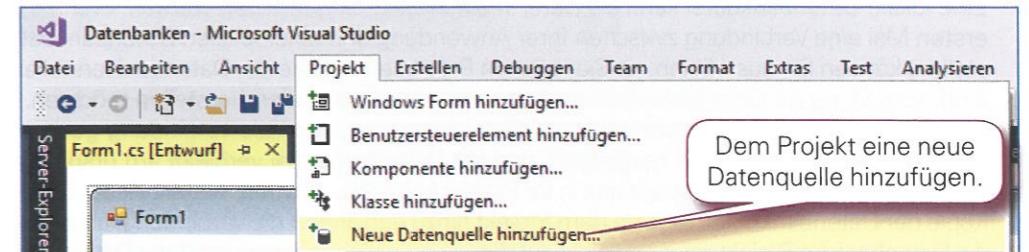
Datensatz: 1 von 3 Kein Filter Suchen

10.2 Den Datenbankassistenten von Visual C# nutzen

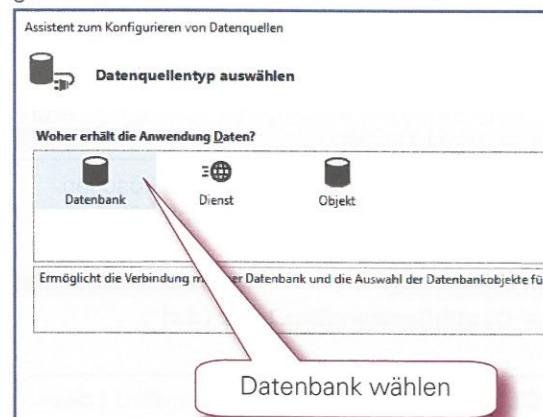
10.2.1 Eine Datenbank einbinden

Die Entwicklungsumgebung Visual C# bietet einen Assistenten an, mit dem Datenbanken automatisiert in ein Projekt eingebunden werden können. Damit verkürzt sich die Entwicklungszeit im Vergleich zu den oben beschriebenen Methoden beträchtlich. Allerdings hat der Entwickler damit auch weniger Freiheiten, da der Assistent viel Quellcode automatisch generiert.

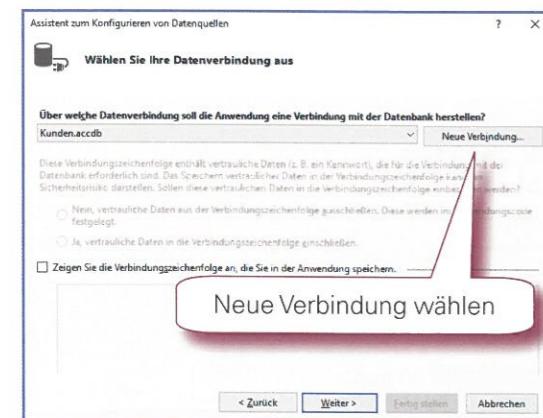
In einem ersten Schritt muss dem Projekt eine Datenquelle hinzugefügt werden. Das geschieht über den Menüpunkt „Projekt → Neue Datenquelle hinzufügen...“:



Im nächsten Schritt wird dann eine Datenbank gewählt:

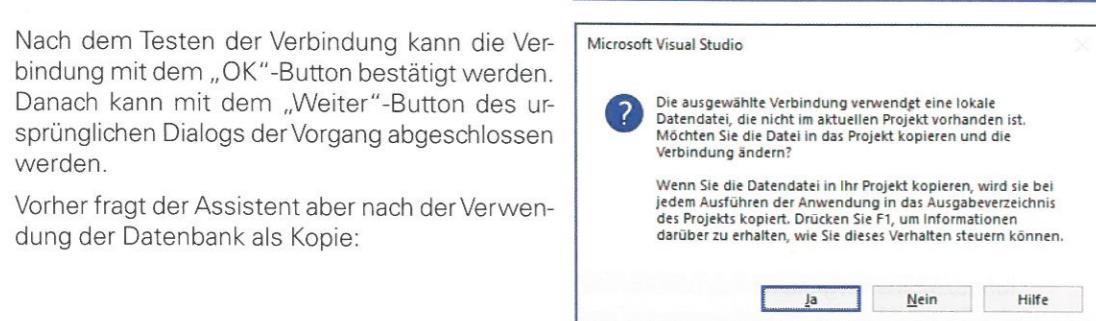


Danach eine neue Verbindung wählen:



Nach dem Testen der Verbindung kann die Verbindung mit dem „OK“-Button bestätigt werden. Danach kann mit dem „Weiter“-Button des ursprünglichen Dialogs der Vorgang abgeschlossen werden.

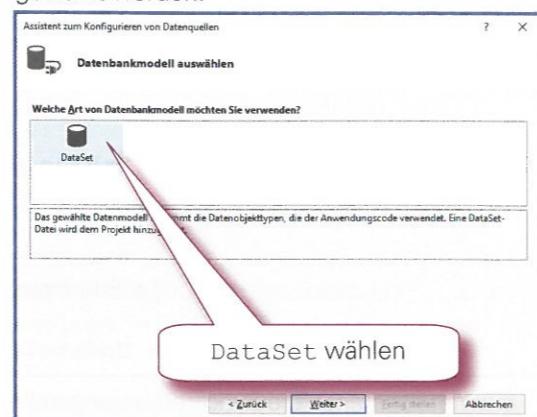
Vorher fragt der Assistent aber nach der Verwendung der Datenbank als Kopie:



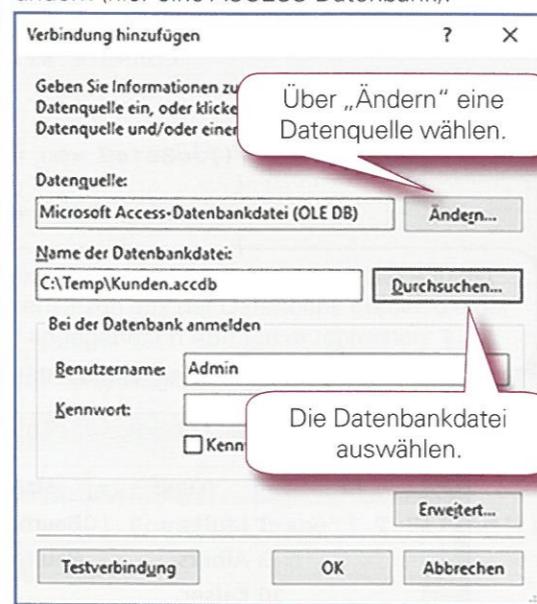
Die Microsoft-Online-Hilfe gibt Auskunft über diese Wahlmöglichkeit:

Eine lokale Datenbankdatei kann als Datei in ein Projekt eingebunden werden. Wenn Sie zum ersten Mal eine Verbindung zwischen Ihrer Anwendung und einer lokalen Datenbankdatei herstellen, können Sie auswählen, ob Sie in Ihrem Projekt eine Kopie der Datenbank erstellen oder eine Verbindung zur Datenbankdatei an deren aktuellen Speicherort herstellen möchten. Wenn Sie eine Verbindung zu der vorhandenen Datei herstellen, wird die Verbindung genauso wie zu jeder Remote-Datenbank hergestellt, und die Datenbankdatei verbleibt am ursprünglichen Speicherort. Wenn Sie die Datenbank in Ihr Projekt kopieren möchten, erstellt Visual-Studio eine Kopie der Datenbankdatei, fügt sie dem Projekt hinzu und ändert die Verbindung, sodass sie auf die Datenbank im Projekt zeigt und nicht auf den ursprünglichen Speicherort der Datenbankdatei.

Anschließend muss das Datenbankmodell gewählt werden:



Die Datenquelle auf die gewünschte Datenbank ändern (hier eine ACCESS-Datenbank):

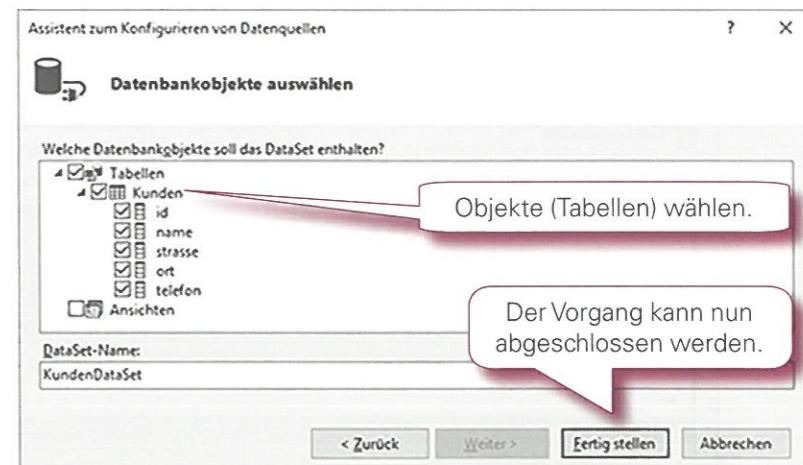


In diesem Beispiel wird mit „Nein“ geantwortet und es wird mit der Original-Datenbank (keine Kopie) gearbeitet.

Die Verbindung kann dann unter einem Namen gespeichert werden:



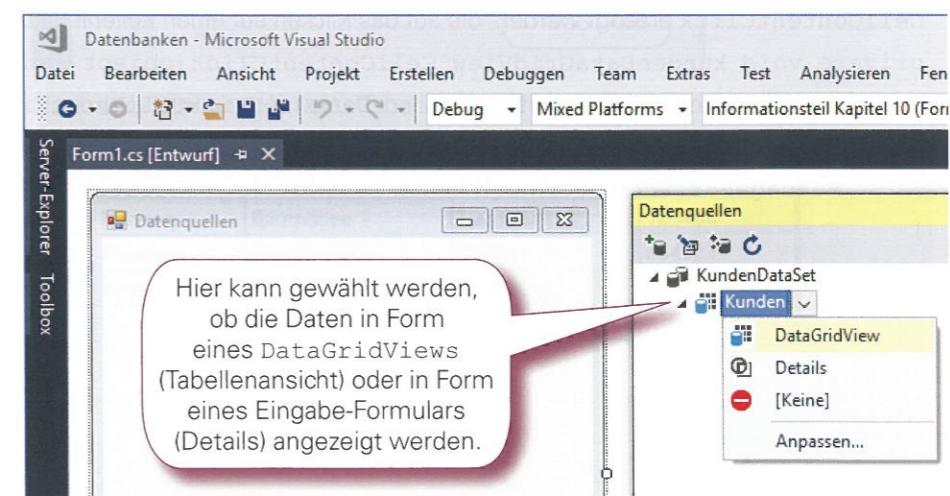
Nun können die Datenbankobjekte ausgewählt werden. In diesem Fall wird die komplette Kunden-tabelle ausgewählt:



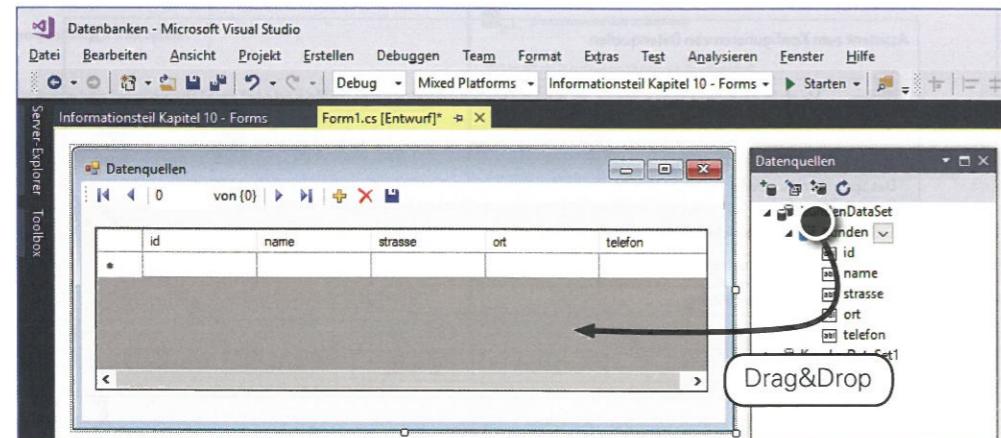
Der Vorgang kann nun abgeschlossen werden.

10.2.2 Windows-Forms-Steuerelemente automatisch anbinden

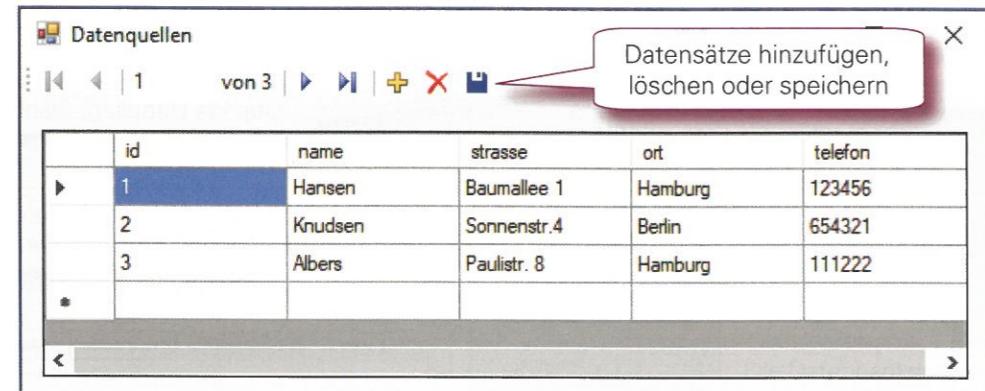
Mithilfe des Menüpunkts „Ansicht → Weitere Fenster → Datenquellen“ wird die Datenquelle angezeigt und kann anschließend für eine Forms-App verwendet werden.



Per *Drag&Drop* kann die Tabelle Kunden nun in der gewünschten Ansicht (hier DataGridView) auf die Form gezogen werden. Der Assistent legt automatisch das entsprechende Steuerelement und die Verbindung zur Datenbank und der Tabelle an:



Nach dem Starten steht eine funktionstüchtige Ansicht der Datenbanktabelle zur Verfügung.



Die Datensätze können bearbeitet sowie gelöscht werden. Ebenso können auch neue Datensätze hinzugefügt werden. Natürlich stehen dem Entwickler eine Vielzahl von Ereignissen und Eigenschaften zur Verfügung, mit denen das Element programmiert werden kann. Beispielsweise kann durch den Doppelklick auf eine Zelle die Ereignisbehandlungsmethode `CellContentClick` erzeugt werden, die auf das Klicken auf einen Zelleninhalt reagiert.

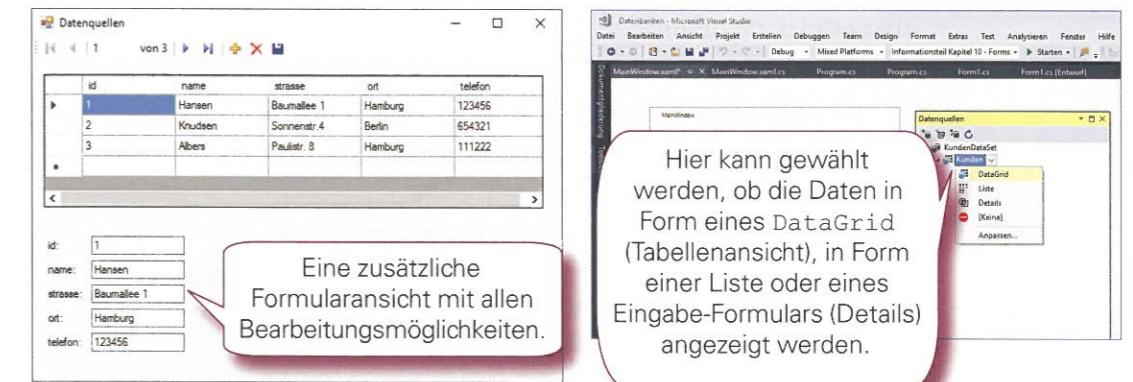
```
private void kundenDataGridView_CellContentClick(object sender,
                                                DataGridViewCellEventArgs e)
```

```
{
```

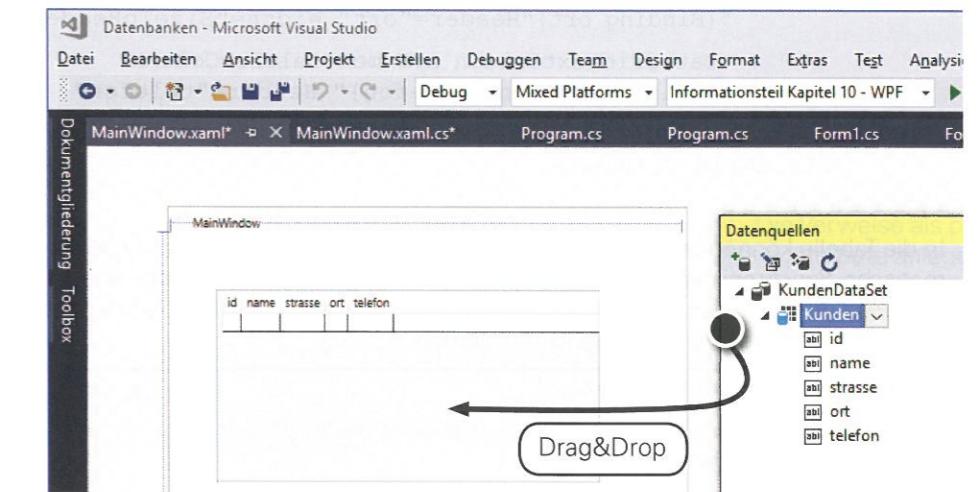
```
    MessageBox.Show("Auf einen Zelleninhalt geklickt!");
```

```
}
```

Zusätzlich zu der GridView-Ansicht kann durchaus auch die Detailansicht per *Drag&Drop* auf die Form gezogen werden. Damit stehen dem Benutzer nicht nur die Tabellenansicht, sondern auch eine Formularansicht zur Verfügung. Die Anzeige der Daten wird automatisch synchronisiert:



Per *Drag&Drop* kann die Tabelle Kunden nun in der gewünschten Ansicht (hier DataGridView) auf das Fenster gezogen werden. Der Assistent legt automatisch das entsprechende Steuerelement und die Verbindung zur Datenbank und der Tabelle an:



Nach dem Starten steht eine Tabellen-Ansicht der Datenbanktabelle zur Verfügung.

id	name	strasse	ort	telefon
1	Hansen	Baumallee 1	Hamburg	123456
2	Knudsen	Sonnenstr.4	Berlin	654321
3	Albers	Paulistr. 8	Hamburg	111222

Der zugehörige XAML-Code sieht so aus:

```
<DataGrid x:Name="kundenDataGrid" AutoGenerateColumns="False" EnableRowVirtualization="True" ItemsSource="{Binding}" Margin="45,59,51,64" RowDetailsVisibilityMode="VisibleWhenSelected">

    <DataGrid.Columns>
        <DataGridTextColumn x:Name="idColumn" Binding="{Binding id}" Header="id" Width="SizeToHeader"/>
        <DataGridTextColumn x:Name="nameColumn" Binding="{Binding name}" Header="name" Width="SizeToHeader"/>
        <DataGridTextColumn x:Name="strasseColumn" Binding="{Binding strasse}" Header="strasse" Width="SizeToHeader"/>
        <DataGridTextColumn x:Name="ortColumn" Binding="{Binding ort}" Header="ort" Width="SizeToHeader"/>
        <DataGridTextColumn x:Name="telefonColumn" Binding="{Binding telefon}" Header="telefon" Width="SizeToHeader"/>
    </DataGrid.Columns>
</DataGrid>
```

In die Tabelle können auch neue Werte eingetragen werden, allerdings findet keine automatische Synchronisierung statt. Dafür muss in der *Code-behind*-Datei eine bestimmte Methode der DatenAdapterklasse aufgerufen werden. Die folgende Ereignismethode zeigt die Vorgehensweise der Synchronisierung.

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    try
    {
        OleDbCommandBuilder cmbKunden = new OleDbCommandBuilder(kundenDataSetKundenTableAdapter);
        Adapter);
    }
}
```

Durch die Instanzierung eines CommandBuilder-Objektes erhält die Adapter-Instanz alle nötigen Informationen für das **Update!**

Die Spalten passen sich der Überschrift an.

Die WPF-Datenbindung!

```
kundenDataSetKundenTableAdapter.Update(kundenDataSet);
    MessageBox.Show("Update erfolgreich");
}

catch (System.Exception ex)
{
    MessageBox.Show("Update-Fehler:" + ex);
}
}
```

Update veranlassen!

Allerdings muss dazu in der *Code-behind*-Datei der Quellcode so geändert werden, dass die Verweise für die Datenbananbindung als Attribute angelegt werden. Ansonsten wäre ein Zugriff auf das Adapterobjekt wie in der obigen Methode nicht möglich.

Der angepasste Quellcode sieht so aus:

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
    }

    private KundenDataSet kundenDataSet;
    Die Verweise als private Attribute anlegen.

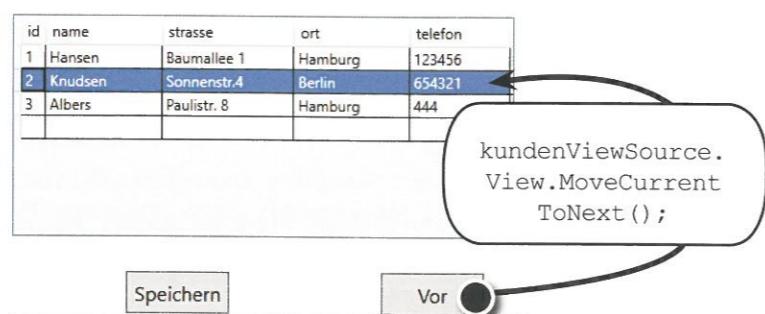
    private KundenDataSetTableAdapters.KundenTableAdapter kundenDataSetKundenTableAdapter;
    private System.Windows.Data.CollectionViewSource kundenViewSource;
    private void Window_Loaded(object sender, RoutedEventArgs e)
    {
        kundenDataSet =
            ((KundenDataSet)(this.FindResource("kundenDataSet")));
        kundenDataSetKundenTableAdapter = new
            KundenDataSetTableAdapters.KundenTableAdapter();
        kundenDataSetKundenTableAdapter.Fill(kundenDataSet.Kunden);
        kundenViewSource =
            ((System.Windows.Data.CollectionViewSource)
                (this.FindResource("kundenViewSource")));
        kundenViewSource.View.MoveCurrentToFirst();
    }
    :
    :
}
```

Die Verweise als private Attribute anlegen.

Navigationsmethoden

Neben der Update-Methode, um die Änderungen festzuschreiben, können mit den Move-Methoden der ViewSource-Klasse Bewegungen in den Datensätzen erzeugt werden:

```
kundenViewSource.View.  
MoveCurrentToNext();  
  
kundenViewSource.View.  
MoveCurrentToPrevious();  
  
kundenViewSource.View.  
MoveCurrentToFirst();  
  
kundenViewSource.View.  
MoveCurrentToLast();
```

**10.3 Aufgaben zu Kapitel 10****Aufgabe 1**

Für die Inventur einer Firma wird ein mobiler Scanner genutzt, der alle Artikel mithilfe eines Barcodes einscannen kann. Zusätzlich kann die Anzahl der Artikel über ein Barcode-Datenblatt eingescannt werden. Nach der Inventur liegen alle Daten in Form einer Textdatei auf einem Speicherchip des Scanners vor. Schreiben Sie eine einfache Konsolenanwendung, die eine solche Textdatei einliest und in einer Datenbanktabelle speichert. Die Datenbanktabelle wird vorher in einer geeigneten Datenbank (z. B. ACCESS) mit den entsprechenden SQL-Befehlen angelegt. Anschließend werden folgende statistische Kenndaten aus der Tabelle ausgelesen:

- Die drei Artikel, von denen die höchste Anzahl vorhanden ist.
- Die drei Artikel, von denen die geringste Anzahl vorhanden ist.
- Die durchschnittliche Anzahl der Artikel.

Nach einer Inventur könnte die Textdatei so aussehen:

```
Inventur.txt - Editor  
Datei Bearbeiten Format Ansicht ?  
DVD-Laufwerk  
14 RAM 4GB } Artikel und Anzahl  
34 USB-Festplatte 1TB  
4 Tastatur Standard  
16 Maus Standard  
19 Monitor 22''  
5
```

Nach dem Starten könnte die Bildschirmausgabe so aussehen:

```
C:\WINDOWS\system32\cmd.exe  
Die drei Artikel mit der höchsten Anzahl:  
RAM 4GB (34)  
Maus Standard (19)  
Tastatur Standard (16)  
  
Die drei Artikel mit der niedrigsten Anzahl:  
Monitor 24' (3)  
USB-Festplatte 1TB (4)  
Monitor 22' (5)  
  
Die durchschnittliche Artikelanzahl lautet: 12,2  
Drücken Sie eine beliebige Taste . . .
```

Hinweise:

- Nutzen Sie für das Einfügen der Datensätze entweder eine SQL-Anweisung oder ein DataSet-Objekt.
- Die Berechnung der Kenndaten geschieht entweder mit den entsprechenden SQL-Funktionen (wie AVG) oder direkt im C#-Programm.

Aufgabe 2**Ausgangssituation**

In einer Firma sind die Bestelldaten der Kunden in zwei Datenbanktabellen (beispielsweise mit ACCESS) abgelegt. Für die Mitarbeiter soll eine einfache GUI-Anwendung geschrieben werden, mit der die Bestelldaten eines Kunden übersichtlich dargestellt werden können. Die zugrunde liegenden Tabellen sehen so aus:

Kundentabelle:

Kunden	
ID	Name
1	Maier
2	Knudsen
3	Kaiser
4	Franzen
5	Knobloch

Beziehung der Tabellen:



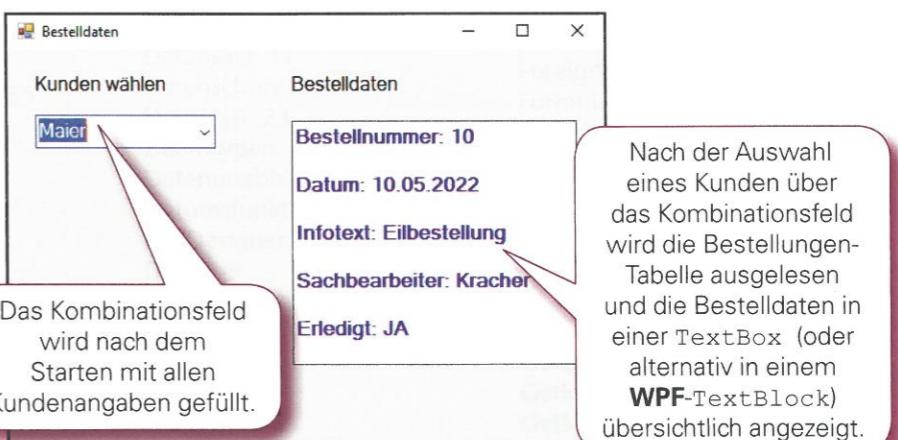
Bestellungen-Tabelle:

Bestellungen					
Kunden_ID	Bestellnummer	Datum	Infotext	Sachbearbeiter	Erledigt
1	10	10-05-2022	Eilbestellung	Kracher	Ja
3	11	22-05-2022	gefährliche Fracht	klauber	Ja
4	12	20-05-2022	guter Kunde	Hütter	Nein

Die Bestellungen-Tabelle hat einen Fremdschlüssel Kunden_ID, der die „1:n“ – Beziehung der beiden Tabellen umsetzt.

Aufgabenstellung:

Legen Sie die beiden Tabellen in einer geeigneten Datenbank an (beispielsweise ACCESS) und füllen Sie die Tabellen mit den entsprechenden Werten. Implementieren Sie dann eine Windows-Forms oder eine WPF-Anwendung, die auf die Datenbank zugreift und die Tabellen ausliest. Die Oberfläche der Anwendung sollte so aussehen:



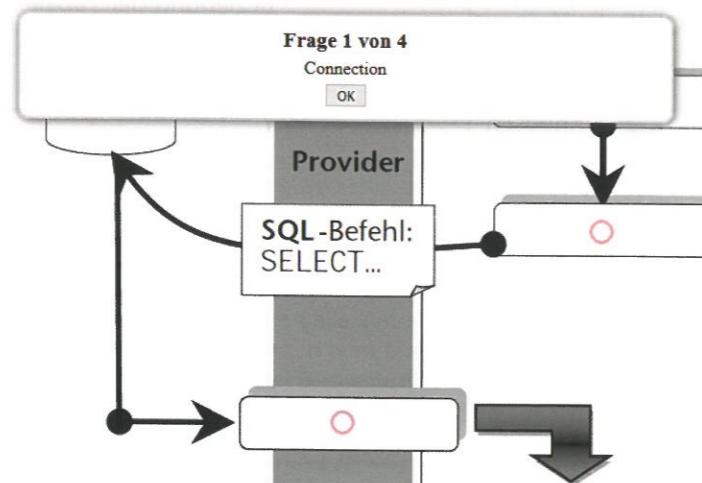
10.4 Digitale Inhalte zu Kapitel 10

Hinweis:

Um die Aufgaben online zu bearbeiten, bitte den QR-Code scannen oder den Link eingeben.

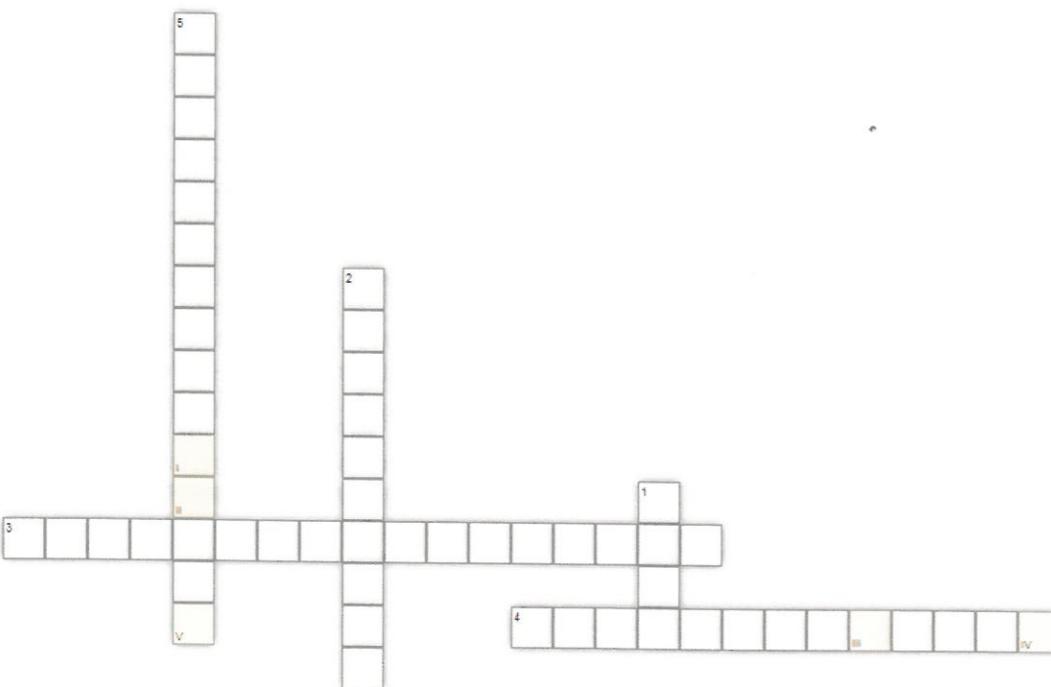
Aufgabe 1: Ablauf eines Datenbankzugriffs in C# erkennen

<https://vel.plus/qkPs>



Aufgabe 2: Kreuzworträtsel mit C#-Datenbankbegriffen

<https://vel.plus/hRHJ>



Index

Symbole

- \$_POST[] 167
- 1:1-Beziehung 25
- 1:m-Beziehung 24
- 1. Normalform 39
- 2. Normalform 40

A

- Abfragen 89
- Abfragen erstellen 145
- ABS(zahl) 110
- Access 77
- Adapter-Objekte 199
- ADO.NET 193
- Aggregatfunktionen 105
- Aktualisieren von Daten 119
- Aktualisierungsweitergabe 81
- Änderungsanomalien 120
- Anomalien 120
- Ansicht 72
- Ansicht-Shape 73
- Anwendungsprogramme 9
- Architektur 12
- Architektur eines DBMS 19
- Arrays 158
- ASC 109
- Assoziative Arrays 159
- Attribute 27
- Auswahlabfragen 98
- AVG 105

B

- Backend 13
- Basistabellen 20
- Bedingungen 98
- Befehlsschaltflächen 85
- Benutzersicht 11
- Berichte 87
- BETWEEN 102
- Beziehung 24, 27
- Beziehungen 80
- Beziehung erstellen 67
- Beziehungslinie 67
- Bottom-up 38

C

- CASCADE 56
- CDATE 109
- Childtabelle 26
- CINT(zahl) 109
- Client/Server-Datenbank 13
- DELETE 119
- Denormalisierung 42
- Detailtabelle 45
- DISTINCT 102
- DML 116, 118
- DQL 93

- Drei-Schichten-Architektur 19
- Dritte Normalform 41
- DriverManager
– getConnection 183

E

- EER-Modell 54
- Einbenutzerbetrieb 13
- Einfügeanomalien 120
- Eingabemaske 12
- Entität 24
- Entitäten-Beziehungs-
Modell 26
- Entity Relationship Model 26
- ERM 26
- ER-Modell 64
- ER-Modellierung 28
- ERM-Symbole 27
- ERP-System 9
- Erste Normalform 39
- ExecuteNonQuery 197
- ExecuteUpdate 185
- EXP(Wert) 110
- External View 20
- Externe Ebene 19

F

- Feldprüfungsbedingung 75
- Feldtyp 138
- fileperms(); 165
- filetype(); 165
- Fill-Methode 200
- Filterbedingungen 147
- fopen() 163
- foreach-Schleife 160
- Foreign Key 56
- Formular-Assistent 149
- Formulare 82, 149, 166
- Forward Engineering 61
- Fremdschlüssel 190, 209
- Frontend 13
- fwrite() 163

G

- GetDateTime-Methode 196
- GetInt32-Methode 196
- GetString-Methode 195, 196
- GRANT 171
- GROUP BY 110
- Gruppenfunktionen 105
- Gruppieren 110
- Gültigkeitsregeln 79