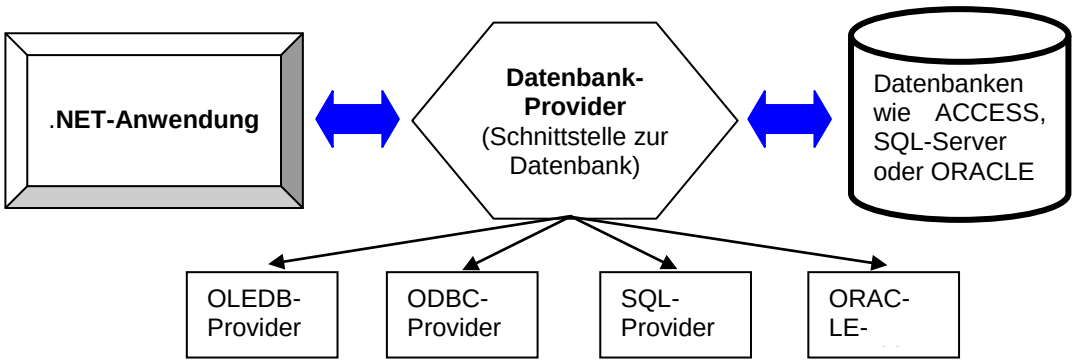
Datenbankanbindung

Datenbankzugriff

Das Speichern von Daten kann eine Anwendung natürlich mithilfe von Dateioperationen selbst durchführen. Für wenige Daten ist das wahrscheinlich auch die erste Wahl bei der Entwicklung einer Anwendung, weil sie damit relativ unabhängig ist. Wenn allerdings viele Daten (oder Datensätze) zu speichern sind und die Daten zusätzlich einen komplizierten Aufbau haben, dann ist die Speicherung in einer Datenbank in Betracht zu ziehen. Der große Vorteil bei einer Datenbankanbindung ist die Unabhängigkeit der Anwendung von der technischen Umsetzung der Datenspeicherung. Das erledigt die Datenbank im Hintergrund. Auch das Ändern oder Löschen von Daten ist bequem durch einige Datenbankbefehle (SQL-Befehle) zu realisieren. Die Abfragesprache **SQL** (Structured Query Language) spielt hierbei eine wichtige Rolle. Bei den folgenden Ausführungen werden deshalb auch grundlegende Kenntnisse in SQL vorausgesetzt.

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 **OLEDB**- und **ODBC**-Datenquellen erfolgen. Die folgende Abbildung zeigt das Grundprinzip von ADO.NET:

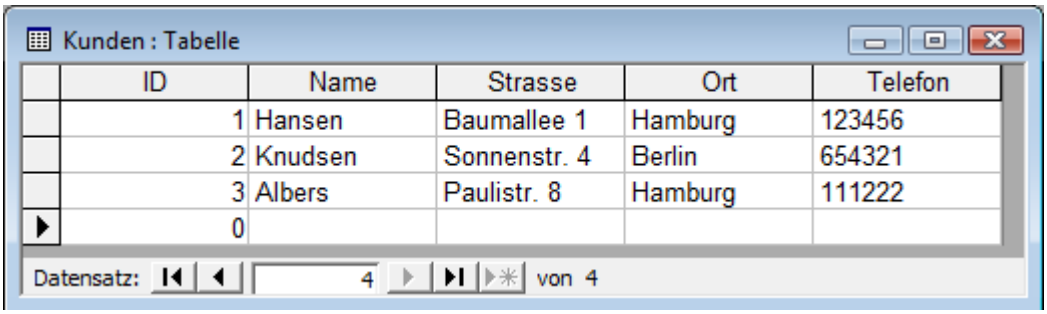


Die einzelnen Provider (Datenanbieter) stehen dabei für bestimmte Datenbankanbindungen:

- ✓ **OLE DB–Provider**: OLE DB steht für **Object Linking and Embedding Database** und ist eine Technik, die bereits bei den Microsoft-Office-Anwendungen zum Einsatz kam. 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 (beispielsweise ACCESS).
- ✓ **ODBC–Provider**: ODBC steht für **Open Database Connectivity** und war eine der ersten Schnittstellen, die eine Vereinheitlichung des Datenbankzugriffs umsetzte. Jede Datenbank brauchte nur eine ODBC-Schnittstelle mitzuliefern und war 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.

Im Folgenden wird der Zugriff auf eine ACCESS-Datenbank mit dem OLE DB-Provider vorgestellt. Das Prinzip ist aber übertragbar auf andere relationale Datenbanken wie beispielsweise den Microsoft-SQL-Server. Für eine einfache Datenbankanbindung sind die Klassen `OleDbConnection`, `OleDbCommand` und `OleDbDataReader` nötig. Damit kann eine Verbindung zur Datenbank aufgebaut werden (`OleDbConnection`), ein SQL-Befehl abgesetzt werden (`OleDbCommand`) und die Ergebnistabelle gelesen werden (`OleDbDataReader`). Das folgende

Beispiel zeigt den Verbindungsaufbau zu einer ACCESS-Datenbank "Kunden.mdb", die in einem Ordner (hier: `C:\temp`) zu Verfügung steht. Die Datenbank verfügt über eine Beispieltabelle `Kunden` mit den Attributen `ID` (Typ Zahl) und `Name`, `Strasse`, `Ort` und `Telefon` (jeweils Text):




```
string verbindungsstring =      "Provider=Microsoft.Jet.OLEDB.4.0;
                                Data Source=C:\\Temp\\Kunden.mdb";
```

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

Ein Verweis auf eine OLEDB-

```
System.Data.OleDb.OleDbConnection dBVerbindung = null;
System.Data.OleDb.OleDbCommand befehl = null;
System.Data.OleDb.OleDbDataReader datenleser = null;
bool offen = false;
```

Ein Verweis auf ein OLEDB-

Ein Verweis auf ein OLEDB-

WICHTIG: Fehler-behandlung

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

Eine Verbin-dungs-Instanz

Datenbank öffnen

Flag setzen

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

Ein Befehlsob-jekt erstellen lassen.

SQL-Befehl (alles aus der Tabelle selek-tieren)

Eine Datenleser-Instanz auf der Grundlage des SQL-Befehls erstellen lassen.

```
        datenleser = befehl.ExecuteReader();

        while (datenleser.Read())
        {
                MessageBox.Show(datenleser.GetString(1));
        }
}
```

Sequenzielles Auslesen des Datenlesers

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

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

Falls die bool'sche Variable `offen` den Wert `true` hat, dann 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) in einer `MessageBox` ausgegeben:

Hinweis:

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

- ✓ `GetDateTime( Spaltenindex )`
- ✓ `GetString( Spaltenindex )`
- ✓ `GetInt32( Spaltenindex )`
- ✓ … weitere Typen

Beispielsweise könnte 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())
{
      MessageBox.Show("Erste Spalte: " + datenleser.GetInt32(0));
}
```

Die erste Spalte vom ACCESS-Typ `Zahl` auslesen.

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 so genannter **`ExecuteNonQuery`**-Befehl abgesetzt werden. Vorher sollte der gewünschte SQL-Befehl in einer Zeichenkette erstellt werden. In dem folgenden Beispiel wird eine neue Zeile in die Kundentabelle eingefügt, eine bestehende Zeile geändert und eine Zeile gelöscht:

```
string verbindungsstring =    "Provider=Microsoft.Jet.OLEDB.4.0;Data
                              Source=C:\\Temp\\Kunden.mdb";

System.Data.OleDb.OleDbConnection dBVerbindung = null;
System.Data.OleDb.OleDbCommand befehl = null;
System.Data.OleDb.OleDbDataReader datenleser = null;
bool offen = false;
int anzahl=0;

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

      befehl = dBVerbindung.CreateCommand();

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

      anzahl = befehl.ExecuteNonQuery();

      MessageBox.Show("Anzahl der eingefügten Zeilen: " + anzahl);

      befehl.CommandText =    "UPDATE Kunden SET Telefon = '11111'
                              WHERE Name = 'Hansen';";

      anzahl = befehl.ExecuteNonQuery();
      MessageBox.Show("Anzahl der geänderten Zeilen: " + anzahl);

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

      anzahl = befehl.ExecuteNonQuery();
      MessageBox.Show("Anzahl der gelöschten Zeilen: " + anzahl);
}
catch (Exception ausnahme)
{
      MessageBox.Show("Datenbankfehler: " + ausnahme.Message);
}
finally
```

Der SQL-Befehl, um eine Zeile einzufügen.

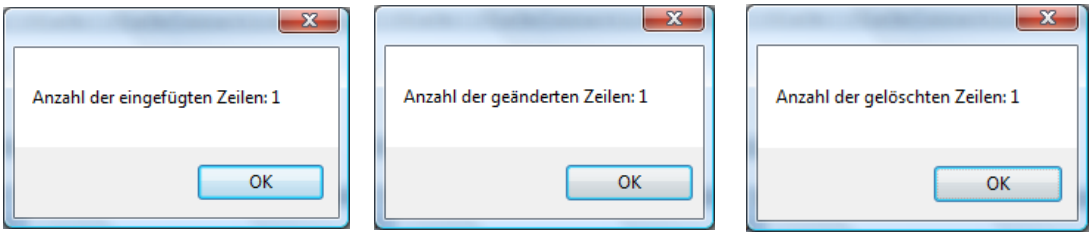
SQL-Befehl absetzen und die Anzahl der betroffenen Zeilen zurückerhalten.

Ein UPDATE-Befehl

Ein UPDATE-Befehl

```
{
        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 in einer `MessageBox` ausgegeben:



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

Vorher:

Kunden : Tabelle

ID	Name	Strasse	Ort	Telefon
1	Hansen	Baumallee 1	Hamburg	123456
2	Knudsen	Sonnenstr. 4	Berlin	654321
3	Albers	Paulistr. 8	Hamburg	111222
0				

Datensatz: 4 von 4

Nachher:

Kunden : Tabelle

ID	Name	Strasse	Ort	Telefon
1	Hansen	Baumallee 1	Hamburg	11111
3	Albers	Paulistr. 8	Hamburg	111222
4	König	Seestr. 5	Hamburg	45621
0				

Datensatz: 4 von 4

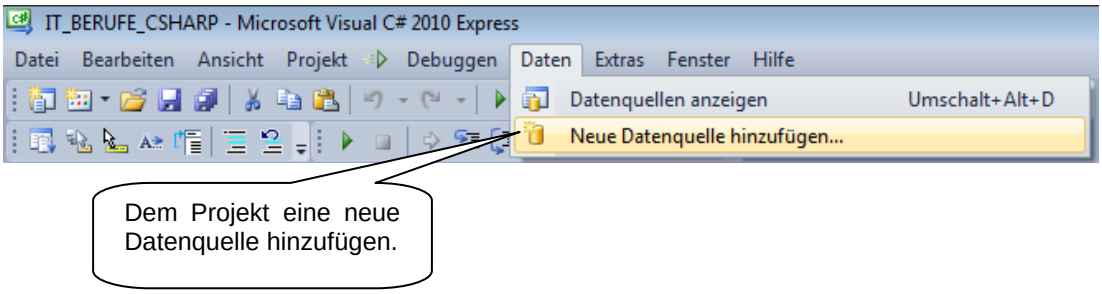
Der neue Datensatz erhält von ACCESS automatisch die ID 4. ACCESS achtet nicht auf lückenlose IDs nach dem Löschen und Einfügen von Datensätzen.

Den Datenbankassistenten nutzen

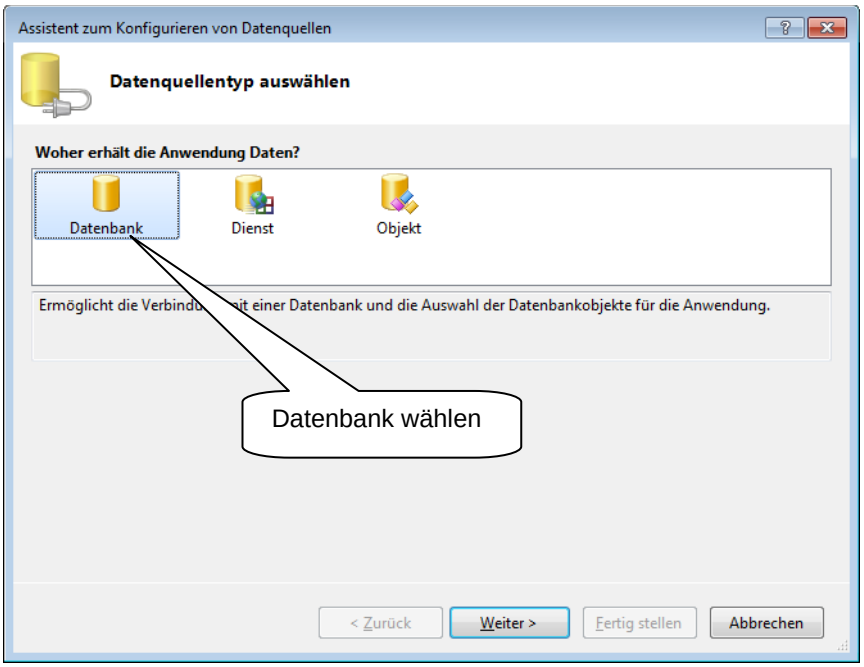
Eine Datenbank einbinden

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

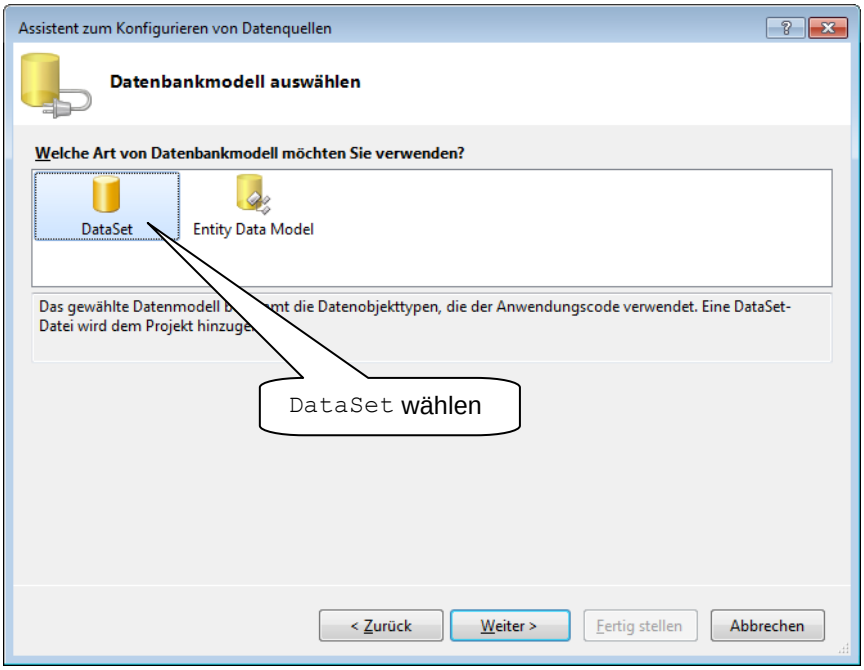
In einem ersten Schritt muss dem Projekt eine Datenquelle hinzugefügt werden. Das geschieht über den Menüpunkt "*Daten → Neue Datenquelle hinzufügen…*".



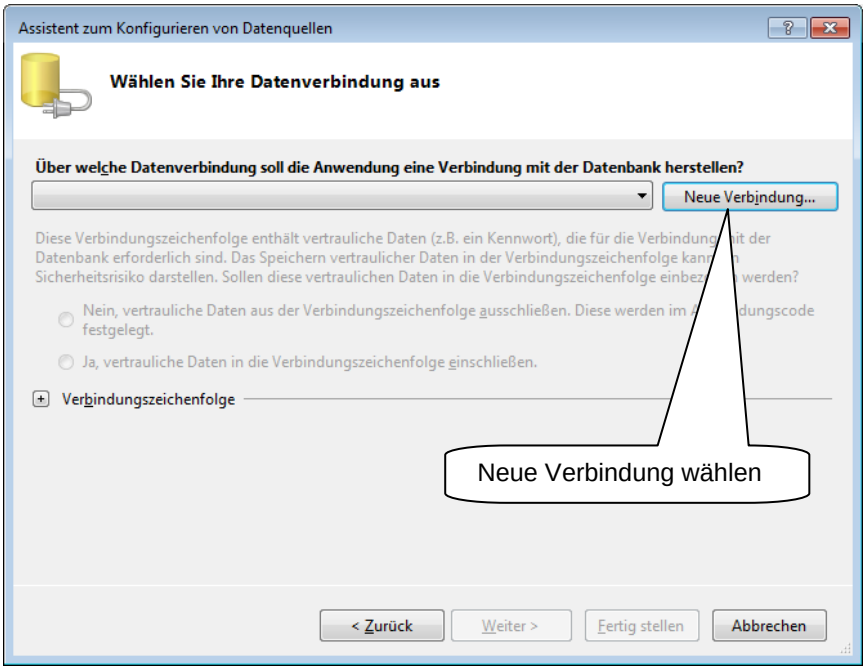
Im nächsten Schritt wird dann eine Datenbank gewählt:



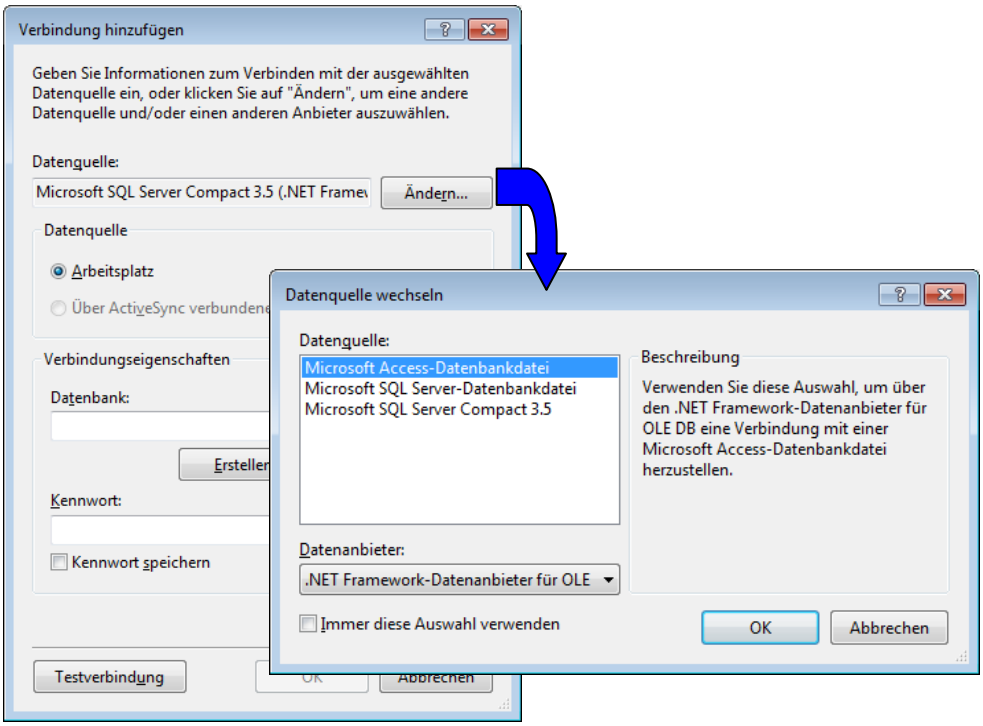
Anschließend muss das Datenbankmodell gewählt werden:



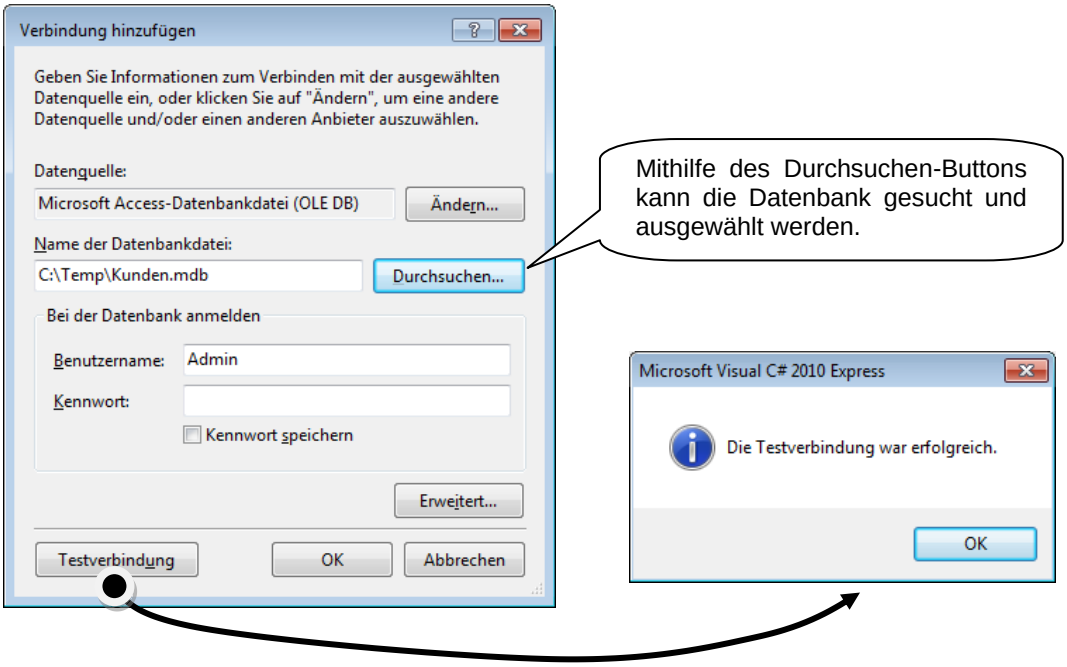
Danach eine neue Verbindung wählen:



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

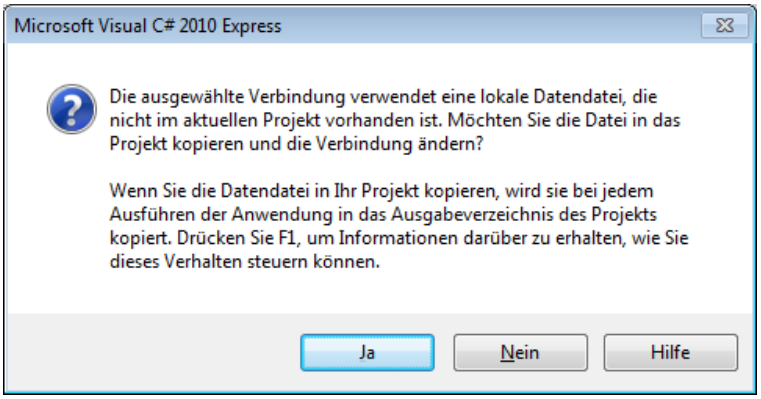


Anschliessend wird die ACCESS-Datenbank angegeben (hier die bereits bekannte Kunden.mdb):



Nach dem Testen der Verbindung kann die Verbindung mit dem "OK"-Button bestätigt werden.

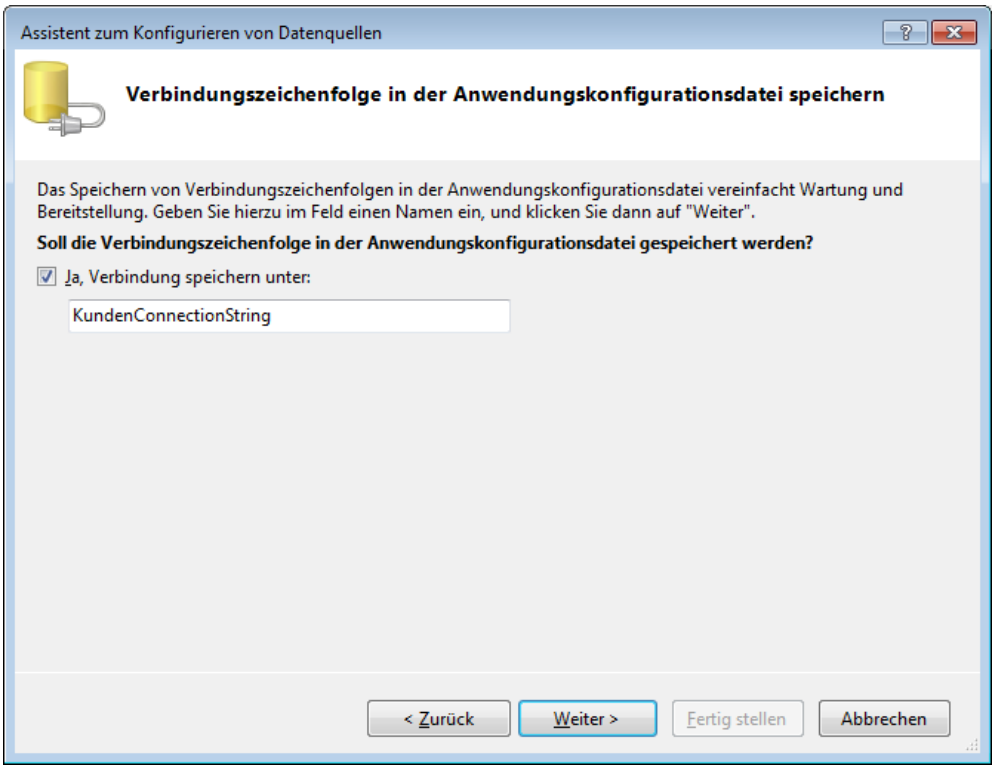
Danach kann mit dem "Weiter"-Button des ursprünglichen Dialogs der Vorgang abgeschlossen werden. Vorher fragt der Assistent aber nach der Verwendung der Datenbank als Kopie:



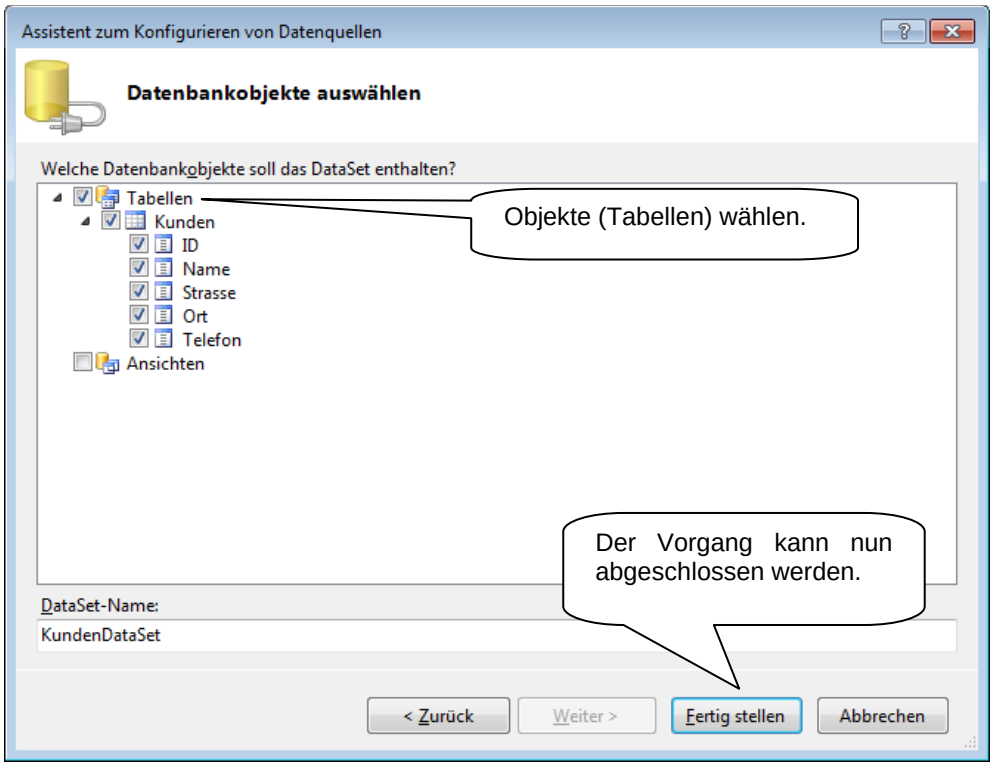
Die Microsoft-Online-Hilfe gibt Auskunft über diese Wahlmöglichkeit:

> 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.

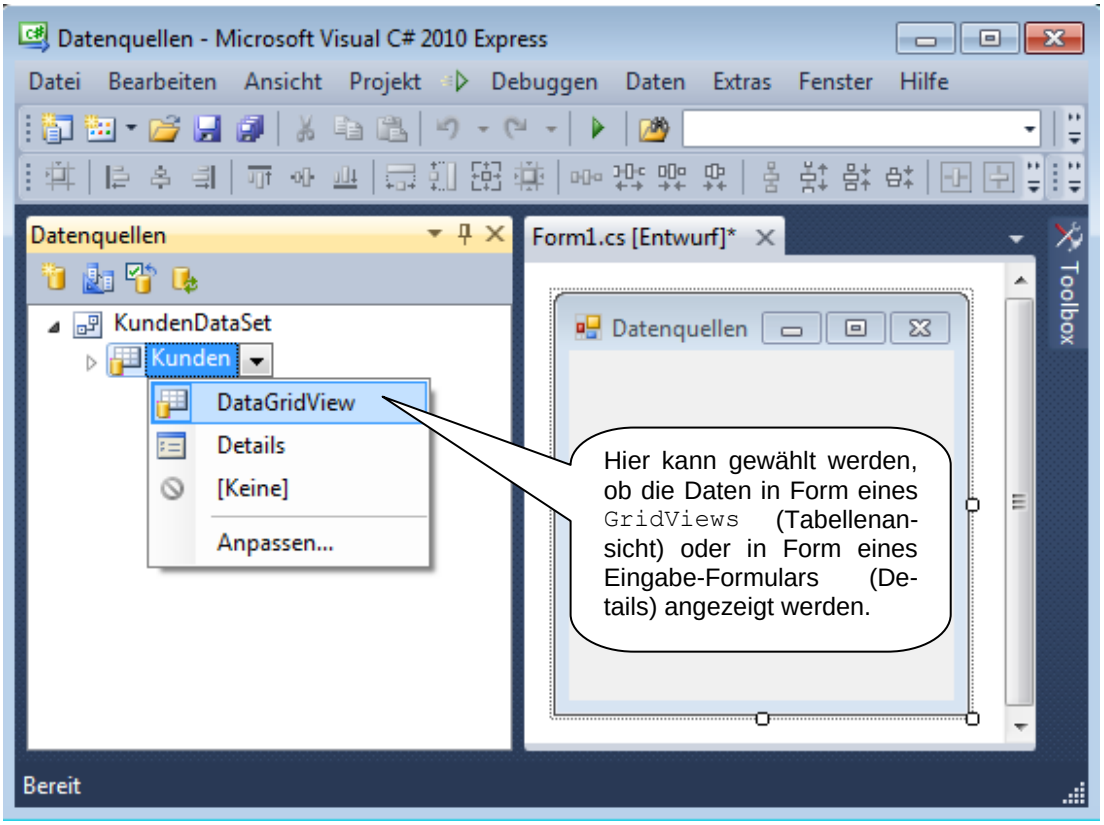
In diesem Beispiel wird mit "Nein" geantwortet und es wird mit der Original-Datenbank (keine Kopie) gearbeitet. Die Verbindung wird dann unter einem Namen gespeichert.



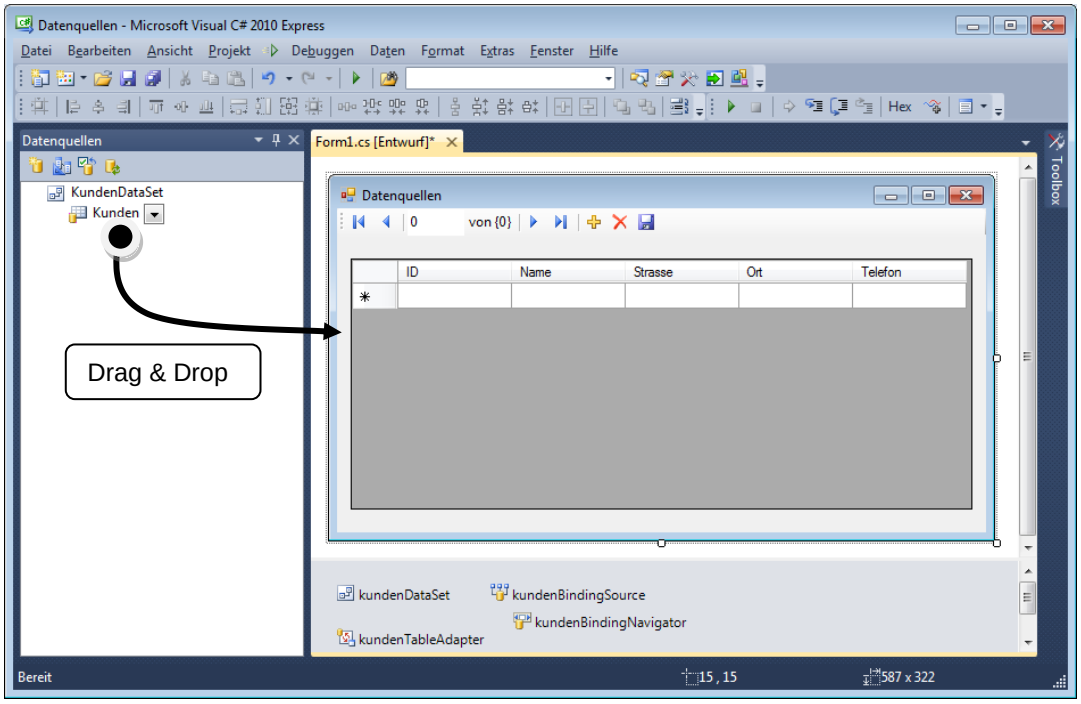
Nun können die Datenbankobjekte ausgewählt werden. In diesem Fall wird die komplette Kundentabelle ausgewählt:



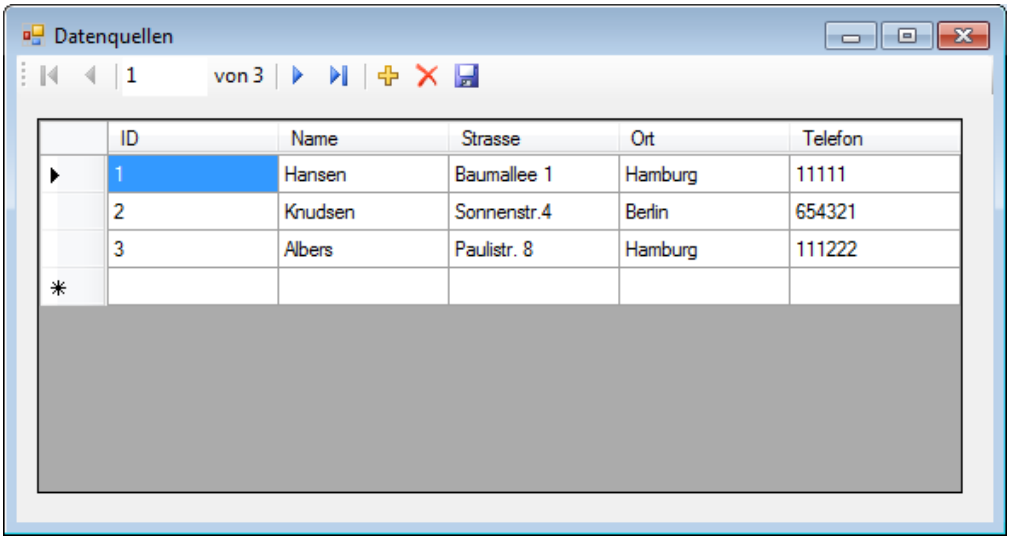
Windows-Forms-Steuerelemente automatisch anbinden
Mithilfe des Menüpunkts "*Daten → Datenquellen anzeigen*" wird die Datenquelle angezeigt und kann anschließend für die Forms-Anwendung verwendet werden.



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



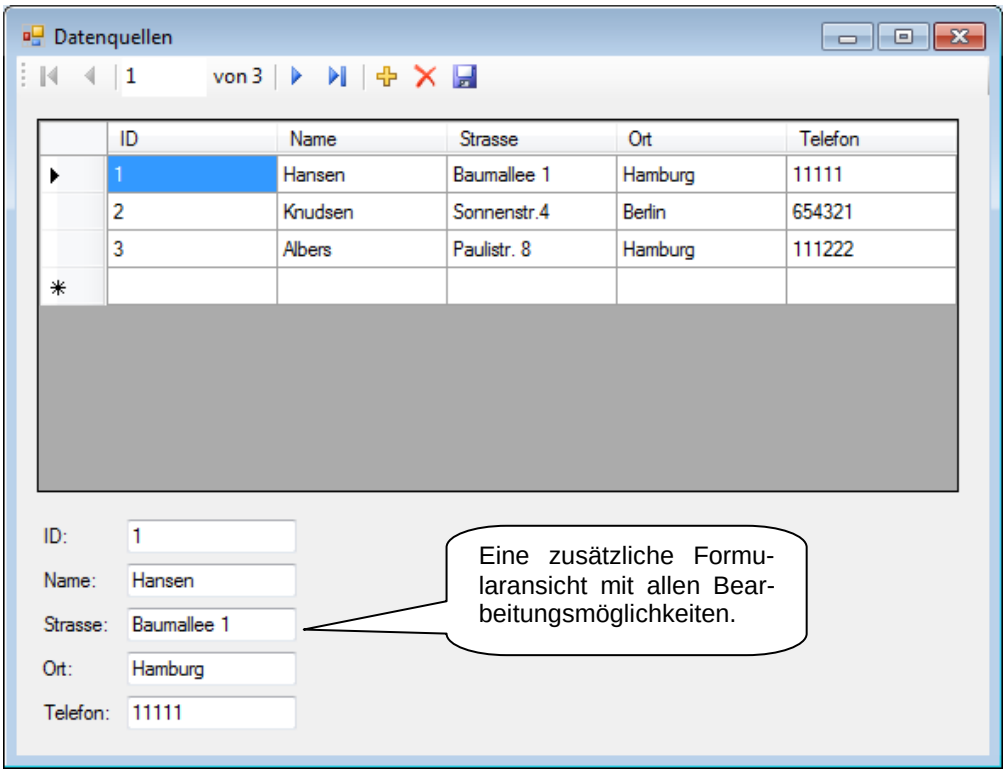
Nach dem Starten steht eine funktionstüchtige Ansicht der Datenbanktabelle zur Verfügung.



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` generiert werden, die auf das Klicken auf einen Zelleninhalt reagiert.

```
private void kundenDataGridView_CellContentClick(object sender,
                                                 DataGridViewCellEventArgs e)
{
      MessageBox.Show("Auf einen Zelleinhalt 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 ist automatisch synchronisiert:



ACHTUNG:

In der zugrunde liegenden Tabelle muss unbedingt ein Primärschlüssel vorhanden sein, sonst kann kein `Update`-Befehl ausgeführt werden und das Speichern von Änderungen führt zu der folgenden Fehlermeldung: