Prof. Dr.-Ing. Jörg Schlingheider
Studiengang Ingenieurinformatik
Wilhelminenhofstr. 75A
12459 Berlin
Tel.: 030-5019-4354
Email: joerg.schlingheider@htw-berlin.de

Hochschule für Technik
und Wirtschaft Berlin
University of Applied Sciences
www.htw-berlin.de

Übung

Für den Zugriff auf Datenbanken stehen verschiedene Möglichkeiten zur Verfügung. Grundlage jeden Datenbankzugriffs ist eine Datenbankverbindung, die mit unterschiedlichen Datenbankschnittstellen aus C# heraus realisiert werden kann. Beispiele derartiger Schnittstellen sind

- ADO
- DAO
- ODBC
- OLEDB

Der grundsätzliche Zugriff ähnelt sich bei den Schnittstellen sehr. Welche Schnittstelle verwendet wird, hängt vor allem davon ab, welche vom Datenbankanbieter angeboten werden. Für MYSQL beispielsweise wird eine ODBC-Schnittstelle angeboten, die auf dem Client installiert werden muß und dann angesprochen werden kann.

Der Aufbau einer OLEDB-Verbindung kann mit folgendem Quelltext hergestellt werden:

```
public class db
{
    OleDbConnection mm_con = null;

    public bool Init()
    {
        bool mm_ret = true;
        //Datenbankpfad holen... aber nicht wie hier hard coded, sondern aus settings oder konfig.
        string mm_dbfile = "C:\\xyz.accdb";
        string mm_oleconstr = "Provider=Microsoft.ACE.OLEDB.12.0; Data Source=" + mm_dbfile;
        try
        {
            //Connection-Objekt herstellen
            mm_con = new OleDbConnection(mm_oleconstr);
            //und öffnen
            mm_con.Open();
        }
        catch
        {
            mm_ret = false;
        }
        return mm_ret;
    }
```

Nachdem die Verbindung zur Datenbank fehlerfrei hergestellt wurde, können entsprechende Datenbank-Kommandos in SQL-Syntax auf der Datenbank ausgeführt werden.

Dazu wird der ein String mit sql-Syntax dem Konstruktor von oleDbCommand als Argument übergeben und ein entsprechendes oleDbCommand-Objekt erzeugt.

```
//sql-Syntax für Abfrage erstellen
string mm_sqlCommand = "SELECT * FROM Person WHERE Alter > 30";
//SQL-Kommando erzeugen
OleDbCommand mm_Command = new OleDbCommand(mm_sqlCommand, mm_con);
```

Das OleDbCommand-Objekt stellt nun verschiedene Möglichkeiten des Datenzugriffs bereit. Ein Objekt zum Lesen der selektierten Daten wird bspw. mit dem OleDbDataReader bereit gestellt:

```
OleDbDataReader mm_Reader = null;
mm_Reader = mm_Command.ExecuteReader();
```

Dieser Reader kann nun mit der Methode Read() ähnlich wie bei Console.Read() durchlaufen werden, um alle Datensätze zu holen. Jedes Read() positioniert den Reader auf den nächsten Datensatz. Mit den GetX-Methoden kann auf die entsprechende Spalte zugegriffen werden, wobei der Datentyp für den Zugriff stimmen muß.

Wenn Daten in die Datenbank geschrieben oder aktualisiert werden sollen, wird ein SQL-Statement mit INSERT oder UPDATE erstellt und direkt über das Command-Objekt ausgeführt:

```
string mm_sql = "INSERT INTO Person ([Name], [Vorname]) VALUES('" +
    Nachname + "', '" + Vorname + "')";
OleDbCommand mm_Command = new OleDbCommand(mm_sql, mm_con);
mm_Command.ExecuteNonQuery();
```

Alternativ zu dieser Technik, bei der die Statements individuell für Lesen und Schreiben aufbereitet werden und auf der Datenbank ausgeführt werden, kann eine weitere Art des Zusammenspiels der Daten verwendet werden. Dabei wird ein DataSet-Objekt gefüllt, das über einen Adapter mit der Datenbank in Beziehung steht:

```
string mm_SQL = "SELECT * FROM Person ORDER BY ID";
DataSet mm_myDS = new DataSet();
//Adapter erstellen
OleDbDataAdapter mm_adapter = new OleDbDataAdapter();
//SQL-Commando an Adapter binden
mm_adapter.SelectCommand = new OleDbCommand(mm_SQL, mm_con);
//Adapter vorbereiten für Aktualisierungscommands
OleDbCommandBuilder builder = new OleDbCommandBuilder(mm_adapter);
//und über den Adapter die Tabelle "personentabelle" (eigener Name) im
//DataSet füllen
mm_adapter.Fill(mm_myDS, "personentabelle");
//Weitere Adapter können weitere Tabellen im Dataset anlegen
```

Nun ist die Tabelle „personentabelle“ im DataSet mm_myDS mit den Datenbankeinträgen gemäß SQL-Statement gefüllt.

Über das DataSet-Objekt können nun die darin kopierten Datensätze manipuliert werden. Diese Manipulationen wirken sich zunächst nur auf das aktuell im Speicher befindliche Dataset-Objekt aus, lassen die Datenbank aber unverändert:

```
//Zugriff auf Daten des Dataset
int mm_count = mm_myDS.Tables["personentabelle"].Rows.Count;
//Name des ersten Eintrags holen
string mm_name = mm_myDS.Tables["personentabelle"].Rows[0]["Name"].ToString();

//Wert in Zeile 0 überschreiben
mm_myDS.Tables["personentabelle"].Rows[0]["Name"] = "Willi";
//oder über DataRow (siehe unten)
DataRow mm_DR = mm_myDS.Tables["personentabelle"].Rows[0];

//Neue Zeile einfügen
DataRow mm_dr = mm_myDS.Tables["personentabelle"].NewRow();
mm_myDS.Tables["personentabelle"].Rows.Add(mm_dr);
mm_dr["Name"] = "Herbert";

mm_adapter.Update(mm_myDS.Tables["personentabelle"]);
```

Erst mit der letzten Zeile erfolgt die Aktualisierung der Datenbank über den Adapter, der die Verbindung zwischen Dataset (mit der korrekten Tabelle) und der Datenbank (aus dem SQL-Statement beim Erzeugen des Adapters) herstellt.

Beim Update können aufgrund von Zugriffsbeschränkungen oder Konsistenzverletzungen Fehler auftreten, die in einem try-catch gefangen werden sollten.

Wenn der Zugriff anstelle von OLEDB über ODBC erfolgen soll, reicht es i.d.R. aus, in den Klassen oleDb durch odbc zu ersetzen.

Die DataSet-Technologie kann auch in der GUI verwendet werden, indem z.B. DataSets an Steuerelemente gebunden werden. Ein Steuerelement, um zwischen den Datensätzen einer Tabelle zu navigieren ist der BindingNavigator. Die Zuordnung zur Tabelle ist denkbar einfach:

```
bindingNavigator1.BindingSource = new BindingSource();
bindingNavigator1.BindingSource.DataSource = m_myDS.Tables["personentabelle"];
```

Jetzt müssen nur noch die Datensatz-Spalten an die Steuerelemente gebunden werden, dann können Datensätze einfach bearbeitet, erstellt und gelöscht werden.

Das Binden eines TextBox-Elementes erfolgt durch folgenden Quelltext:

```
//Binden der Eigenschaft "Text" der Textbox an die Spalte "Name" der Datenquelle
txtName.DataBindings.Add("Text", bindingNavigator1.BindingSource, "Name");
```

Nach diesen einleitenden Erklärungen soll schrittweise damit begonnen werden, eine kleine GUI für eine Adress-Datenbank zu erstellen. Gehen Sie dazu wie folgt vor:

- Erstellen einer Access-Datenbank mit einer Tabelle Person mit den Spalten ID (AutoWert), Name und Vorname als Text
- Legen Sie einige Datensätze an
- Erstellen einer Windows-Forms-Anwendung
- Erstellen einer Klasse Datenbank, in der alle Datenbank-spezifischen Operationen gekapselt werden sollen (Fassaden-Muster)
- Die Klasse soll als Singleton realisiert werden, so daß es nur ein Objekt der Datenbank-Klasse gibt
- Implementieren des privaten Konstruktors, so daß hierbei die Herstellung einer Connection zur Datenbank erfolgt

Testen Sie die Funktionsfähigkeit Ihres Programms bis zu diesem Schritt.

Wenn dieser Programmteil den Anforderungen entspricht, erweitern Sie das Programm um folgende Elemente:

- Erstellen eines BindingNavigators im Formular
- Hinzufügen einer DataSet-Membervariablen in der Klasse Datenbank
- Erstellen eines Adapters für die Tabelle Person und Füllen des DataSet-Objektes im Konstruktor
- Hinzufügen einer Eigenschaft vom Typ DataTable zur Klasse Datenbank. Diese Eigenschaft soll die Tabelle Person des privaten DataSet-Objektes als ReadOnly-Objekt nach außen verfügbar machen
- Zuweisen der Datatable-Eigenschaft des Singleton zu der BindingSource-Eigenschaften wie oben exemplarisch beschrieben

Testen, ob die Anzahl der Datensätze im Navigator stimmt. Sollte auch dieser Programmteil funktionieren, erstellen Sie TextBoxen für Name und Vorname und legen Sie bitte die TextBox-Datasource-Eigenschaften wie oben beschrieben auf die BindingSource-Spalten des Navigators.

Nun sollten Sie durch alle vorhandenen Datensätze Ihrer Anwendung scrollen können und in den Textboxen die eingetragenen Werte überprüfen können.

Ergänzen Sie im Folgenden Ihre Anwendung derart, daß

- in der Datenbank auch ein Geburtstag angegeben werden kann
- dieser in der Oberfläche ebenfalls dargestellt wird (z.B. mit dem DateTimePicker)
- das Aktualisieren von Datensätzen funktioniert, wobei der Benutzer gefragt wird, ob geänderte Daten gespeichert werden sollen und hierbei auch ein Abbruch möglich ist
- auch die Neuanlage von Datensätzen nach Abfrage möglich ist