ADO.NET

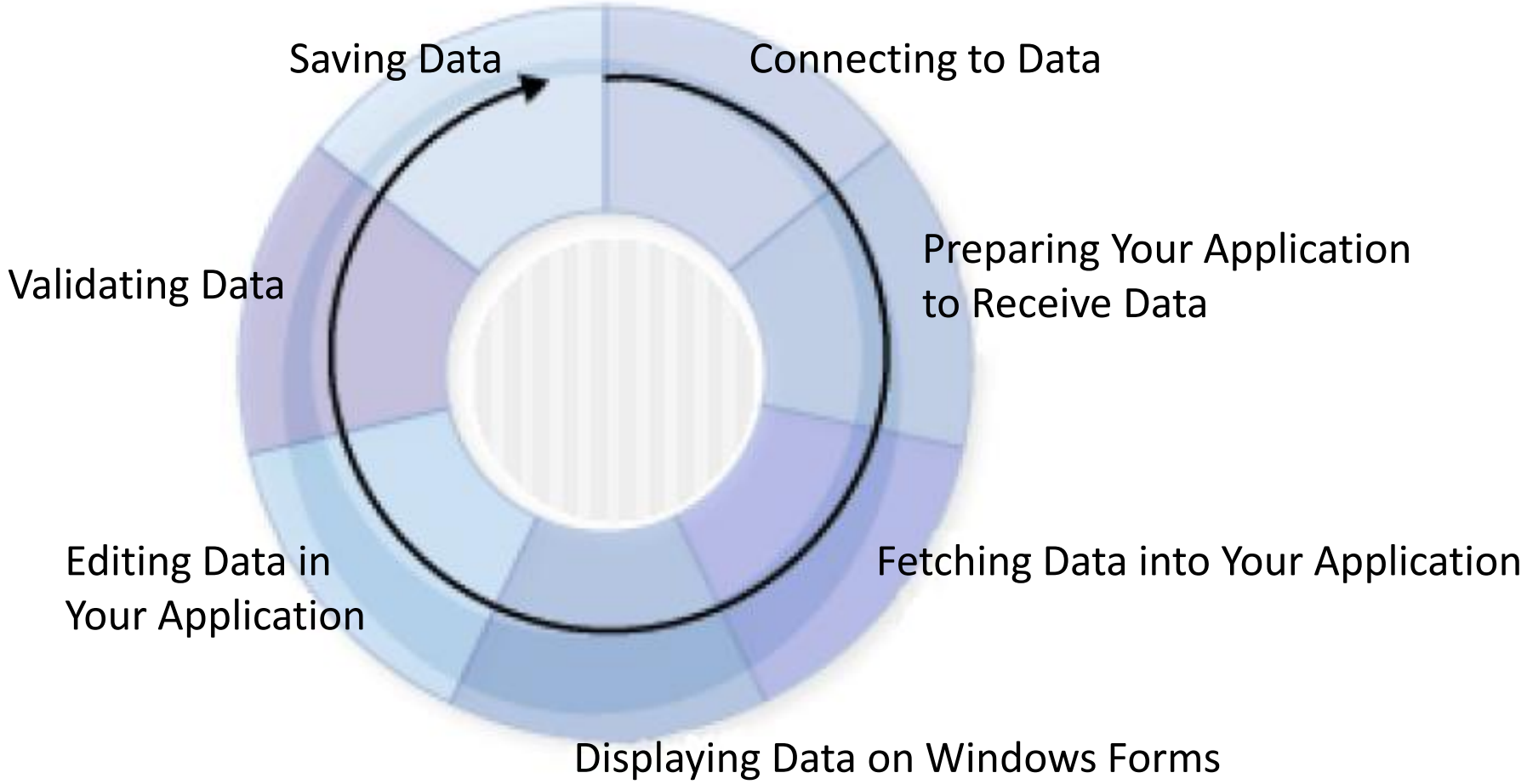
Verbindung zwischen der Anwendung und der Datenbank

- ADO.NET stellt konsistente Zugriff auf Datenquellen wie SQL-Server und XML, sowie auf Datenquellen, die durch OLE DB (Object Linking and Embedding Database) und ODBC (Open Database Connecting) verfügbar gemacht werden
- Man kann mit ADO.NET eine Verbindung mit den Datenquellen herstellen und die Daten abrufen, verarbeiten und aktualisieren
- ADO.NET trennt den Datenzugriff von der Datenbearbeitung in einzelne Komponente (die separat oder zusammen verwendet werden können)

Verbindung zwischen der Anwendung und der Datenbank

- ADO.NET enthält Objekte und Methoden für das Herstellen einer Verbindung mit einer Datenquelle und das Abrufen der darin enthaltenen Daten
- Die Daten werden entweder direkt verarbeitet oder in einem ADO.NET DataSet Objekt temporär gespeicher

Data Cycle


Saving Data
Connecting to Data
Validating Data
Preparing Your Application to Receive Data
Editing Data in Your Application
Fetching Data into Your Application
Displaying Data on Windows Forms

Connecting to Data

- Connecting to Data: ermöglicht eine bidirektionale Kommunikation zwischen der Anwendung und der Datenbank (*TableAdapter*, *ObjectContext*)
- Preparing App to Receive Data: man kann die Daten lokal speichern in bestimmten Objekten (*DataSets*, *Entities*, LINQ von SQL Objekte)
- Fetching Data : Ausführen von Abfragen und gespeicherten Prozeduren (*TableAdapters*, LINQ to Entities, direkte Verbindung zwischen Entities und gespeicherte Prozeduren)
- Displaying Data : data-bound Controls
- Editing and Validating Data : Tupeln einfügen, aktualisieren, löschen; überprüfen ob die Daten alle Integritätsregeln einhalten
- Saving Data : *TableAdapterManager, SaveChanges*

Datenmodellen

- Typisierte und nicht typisierte DataSets
- Konzeptuelles Modell basierend auf Entity Data Model
  - Beschreibt die Datenstruktur unabhängig von der gespeicherten Form
  - kann von Entity Framework oder von WCF Data Services benutzt werden
- LINQ (Language-Integrated Query) von SQL Objekte
  - ist eine Komponente von .NET Framework, die eine Infrastruktur für die Verwaltung relationaler Daten als Objekte bereitstellt
  - Benutzt DataContext um Zugriff zu den Daten zu haben
  - relationale Daten erscheinen als Auflistung zweidimensionaler Tabellen (*Beziehungen* oder *Flatfiles*), bei denen gemeinsame Spalten Tabellen verbinden

LINQ von SQL Objekte - Beispiel

```
// DataContext takes a connection string.
DataContext db = new DataContext(@"c:\Northwnd.mdf");
// Get a typed table to run queries.
Table<Customer> Customers = db.GetTable<Customer>();
// Query for customers from London.
var query = from cust in Customers where cust.City == "London" select cust;
foreach (var cust in query)
Console.WriteLine("id = {0}, City = {1}", cust.CustomerID, cust.City);
public partial class Northwind : DataContext {
  public Table<Customer> Customers;
  public Table<Order> Orders;
  public Northwind(string connection) : base(connection) { } }
```

DataSets

- Die Struktur eines **DataSets** kann mit der Struktur einer relationalen Datenbank verglichen werden → ein hierarchisches Objektmodell von Tabellen, Zeilen, Spalten, Einschränkungen und Beziehungen
- Ein DataSet enthält eine Auflistung von Datentabellen und eine Auflistung von DataRelation-Objekten
- In DataSets werden die Daten verwendet in der Anwendung temporär gespeichert → ein lokaler Datencache im Arbeitspeicher wird bereitgestellt
- Die Arbeit mit den Daten in einem DataSet kann auch dann fortgesetzt werden, wenn die Anwendung von der Datenbank getrennt wird
- Das DataSet verwaltet die Informationen zu Datenänderungen, damit die Aktualisierungen an die Datenbank zurückgesendet werden können, sobald die Anwendung erneut verbunden wird

Typisierte vs. Nicht Typisierte DataSets

- Typisierte DataSets:
  - Man kann auf Tabellen und Spalten anhand ihres Namens zugreifen, man muss keine auflistungsbasierten Methoden verwenden
  - Die Typkonvertierung wird in einer dafür aufgebauten Klasse realisiert; diese Klasse nutzt die im Schema definierten Spaltentypen
  - Siehe https://support.microsoft.com/de-de/kb/320714
- Nicht Typisierte DataSets:
  - Beim Zugriff auf nicht typisierte DataSets werden Feldinhalte nicht typisiert zurückgegeben, d.h. sie sind von Typ Objekt
  - Um deren Inhalt zu nutzen, muss eine Typkonvertierung ausgeführt werden
- Beispiel
  - Nicht Typisierte DataSets:

    ```
    string name = (string)ds.Tables["Adressen"].Rows[0]["Name"];
    ```
  - Typisierte DataSets:

    ```
    string name = dsAdressen.RowAdr[0].Name;
    ```

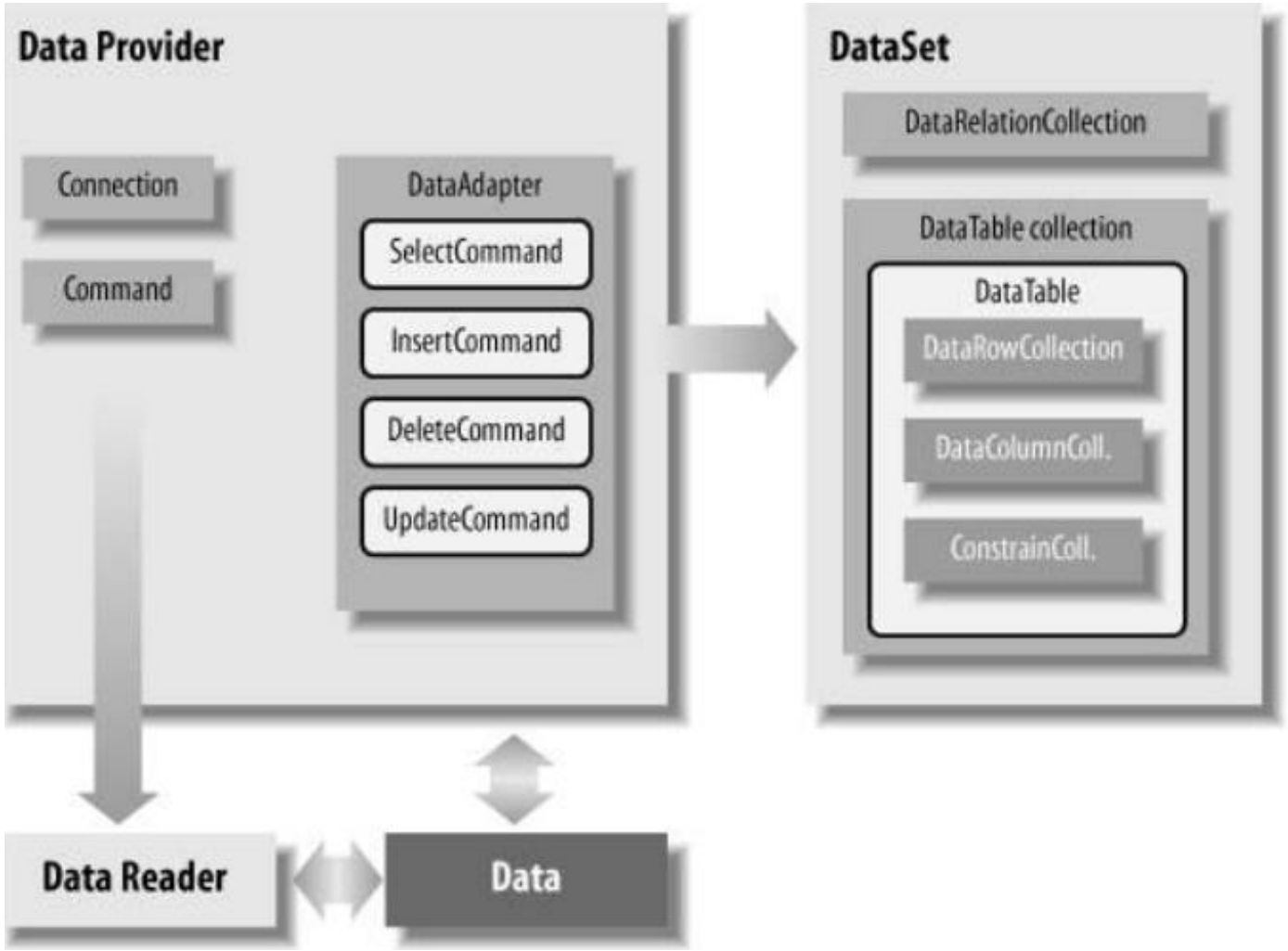
Hierarchische Aktualisierung - TableAdapterManager

- TableAdapterManager – eine Komponente, die Funktionen zum Speichern von Daten in verwandten Tabellen bereitstellt
- Komplexer als das Speichern von Daten einer einzelnen Tabelle → man muss die Konsistenzregeln (referenzielle Integrität) einhalten
- Fremdschlüsselbeziehungen werden verwendet, um so die richtige Reihenfolge beim Senden von Einfüge-, Aktualisierungs- und Löschvorgängen vom DataSet zur Datenbank festzulegen, ohne dabei die Fremdschlüsseleinschränkungen in der Datenbank zu verletzen (referenzielle Integrität)

Hierarchische Aktualisierung - TableAdapterManager

- Der gesamte Vorgang der Datenspeicherung in verknüpften Datentabellen mit einem **TableAdapterManager** wird als Hierarchische Aktualisierung bezeichnet.
- Wenn die Anwendung typisierte DataSets verwendet und Benutzern ermöglicht, die Daten in verknüpften Datentabellen (Datentabellen in einer 1:n-Beziehung wie Customers oder Orders) zu ändern, empfiehlt sich die Verwendung hierarchischer Aktualisierungen

ADO.NET


Data Provider
Connection
Command
DataAdapter
SelectCommand
InsertCommand
DeleteCommand
UpdateCommand
DataSet
DataRelationCollection
DataTable collection
DataTable
DataRowCollection
DataColumnColl.
ConstrainColl.
Data Reader
Data

Console

- Standardstreams für Eingabe, Ausgabe und Fehler bei Konsolenanwendungen
- Eigenschaften:
  - WindowLeft, WindowTop, WindowHeight, WindowWidth, BackgroundColor, Title, etc.
- Methoden:
  - Write(...), WriteLine(...)
  - Read(...), ReadLine(...), ReadKey(...)

SQLConnection

- Wird in Verbindung mit ***SqlDataAdapter*** und ***SqlCommand*** verwendet, um die Leistung beim Verbinden mit einer SQL Server Datenbank zu verbessern
- Wenn die SqlConnection den Gültigkeitsbereich verlässt(goes out of scope), wird diese nicht geschlossen → die Verbindung muss explizit geschlossen werden (*Close* aufrufen)
- Um sicherzustellen, dass die Verbindung immer geschlossen wird, kann man einen **using**-Block verwenden
- Eigenschaften:
  - ***ConnectionString*** – erforderliches Parameter für das Herstellen der Verbindung (enthält Namen der Datenabnk, u.a.); siehe https://www.connectionstrings.com/
  - ***ConnectionTimeout*** – die Zeit, die beim Verbindungsaufbau gewartet werden soll, bis der versuch beendet und ein Fehler generiert wird
- Methoden:
  - Open()
  - Close()
- Wenn ein ***SqlException*** generiert wird, bleibt das ***SqlConnection*** offen wenn der Severity Level ≤ 19 ist (es kann aber sein das manche Anfragen nicht mehr richtig funktionieren)

SQLCommand

- Stellt eine Transact-SQL-Anweisung oder eine gespeicherte Prozedur dar, die in einer SQL Server-Datenbank ausgeführt werden soll
- Diese Klasse kann nicht vererbt werden
- Eigenschaften:
  - ***CommandText***
  - ***CommandTimeout***
- Methoden:
  - ***ExecuteNonQuery*** – gibt die Anzahl der betroffenen Zeilen zurück
  - ***ExecuteScalar*** – führt die Abfrage aus und gibt die erste Spalte der ersten Zeile im Resultset zurück
  - ***ExecuteReader*** - erstellt einen ***SQLDataReader***
- ***SQLDataReader*** – zum Lesen eines Vorwärtsstreams (forward-only stream) von Zeilen aus einer SQL Server-Datenbank

SQLDataAdapter

- Dient als Brücke zwischen einer DataSet und SQL Server für das Abrufen und Speichern von Daten
- Eigenschaften:
  - ***UpdateCommand –*** eine Transact-SQL-Anweisung oder gespeicherte Prozedur zum Aktualisieren von Datensätzen in der Datenquelle
  - ***InsertCommand –*** eine Transact-SQL-Anweisung oder gespeicherte Prozedur zum Einfügen neuer Datensätze in der Datenquelle
  - ***DeleteCommand –*** eine Transact-SQL-Anweisung oder gespeicherte Prozedur zum Löschen von Datensätzen aus dem DataSet
- Methoden:
  - ***Fill(DataSet, String) –*** fügt hinzu oder aktualisiert Zeilen in der DataSet entsprechend den in der Datenquelle

Data Binding

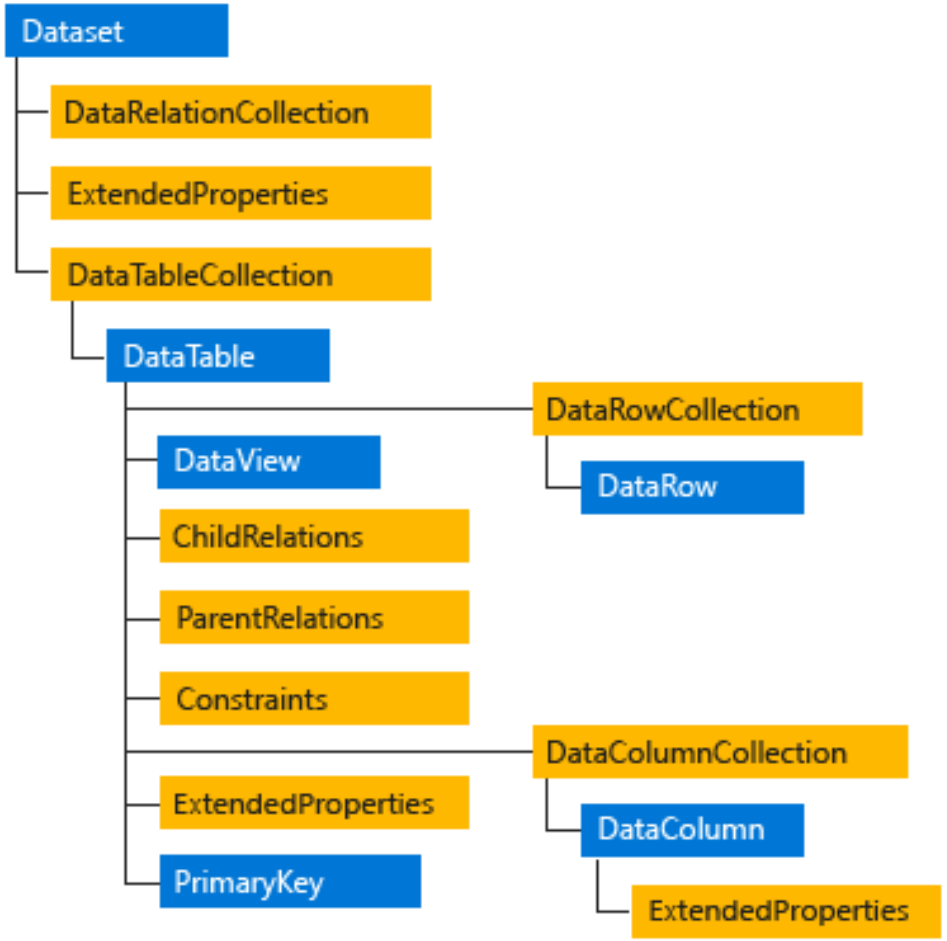
- ***DataBindings*** – jede Eigenschaft eines beliebigen Steuerelements (control) kann man an die Datenquelle binden
- Z.B. Man bindet die Text-Eigenschaft eines TextBox-Steuerelements and die Datenquelle
- ***Binding*** Klasse – stellt die Bindung zwischen der Eigenschaft des Steuerelements und der Eigenschaft des Objektes (Datenquelle) dar
- ***Binding(String propertyName, Object dataSource, String dataMember***)
  - propertyName – Name der Steuerelementeigenschaft
  - dataSource – Datenquelle
  - dataMember – Eigenschaft oder Liste, an die die Bindung erfolgen soll
- Siehe https://msdn.microsoft.com/de-de/library/4wkkxwcz(v=vs.110).aspx

DataSet

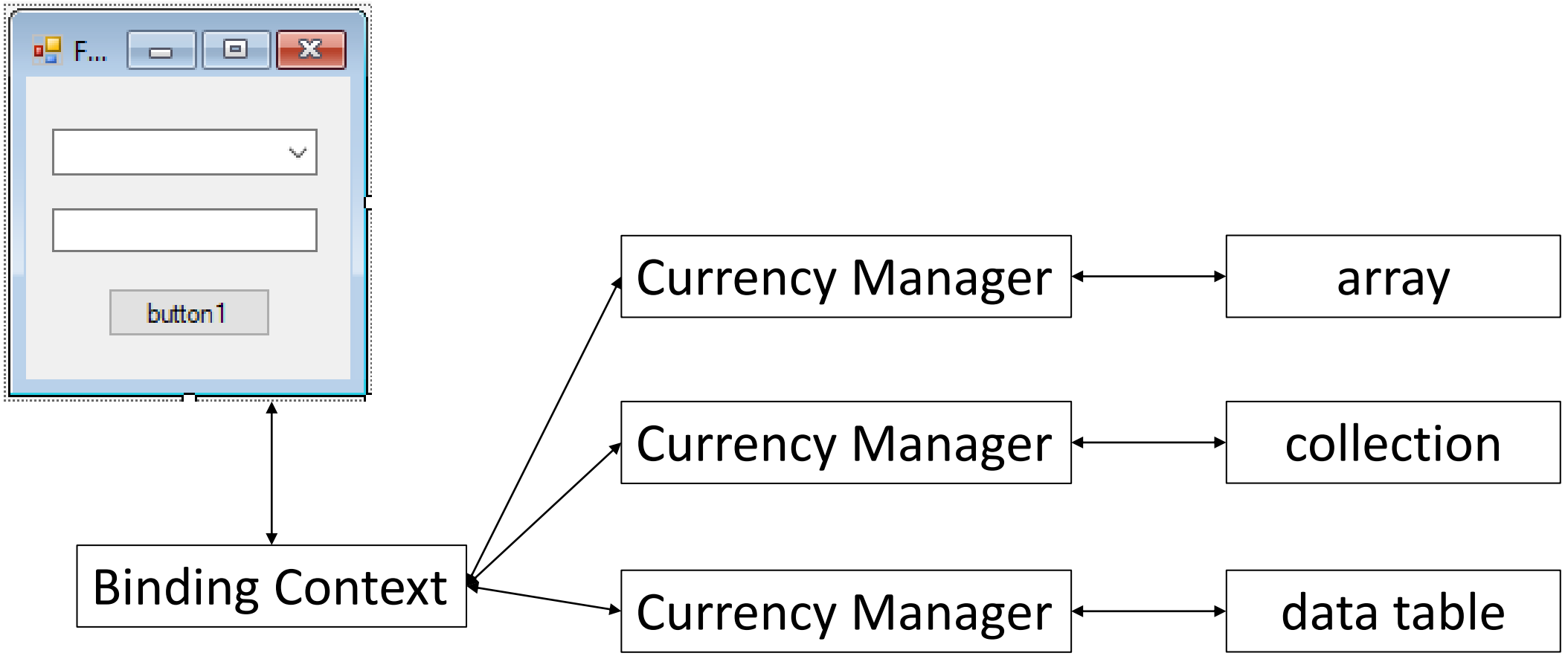
- Stellt einer speicherresidenten Datencache dar
- Eigenschaften:
  - ***Tables –*** Auflistung von Tabellen in der DataSet
  - ***Relations –*** Auflistung der Beziehungen, die Tabellen verknüpfen (übergeordnete/untergeordnete Tabellen)
- Methoden:
  - ***Clear() –*** löscht alle Zeilen aus allen Tabellen in dem DataSet
  - ***Haschanges() –*** gibt an, ob es neue, geänderte oder gelöschte Zeilen gibt

DataSets

- Die DataSet Klasse enthält:
  - ***DataTableCollection***
  - ***DataRelationCollection***
- Dia ***DataTable*** Klasse enthält:
  - ***DataRowCollection***
  - ***DataColumnCollection***
  - ***ChildRelations***
  - ***ParentRelations***
- Die ***DataRow*** Klasse enthält die Eigenschaft ***RowState*** (mögliche Werte: *Deleted, Modified, Added* und *Unchanged*)



Data binding – Windows Forms



Data providers in .NET

- Mit Windows Forms-Datenbindung, können Sie auf Daten aus Datenbanken sowie Daten in andere Datenstrukturen, z. B. Arrays und Sammlungen, zugreifen, solange diese bestimmte Mindestanforderungen erfüllen
- Ein **BindingSource** ist die am häufigsten verwendete Windows Forms-Datenquelle und stellt einen Proxy zwischen einer Datenquelle und einem Windows Forms-Steuerelement dar

BindingSource

- ADO.NET bietet eine Reihe von Datenstrukturen, die geeignet sind, für die Bindung an:
  - **DataColumn** (z. B. die Text Eigenschaft des Steuerelements TextBox mit einer Spalte in einer Datentabelle verbinden)
  - **DataTable** – bestehend aus Columns, Rows, Constraints (z. B. die Bindung der **DataGridView** Steuerelement an eine Datentabelle)
  - **DataView** - eine angepasste Ansicht einer einzelnen Datentabelle
  - **DataSet** – bestehend aus Tables, Relationships, Constraints
  - **DataViewManager** – eine angepasste Ansicht des gesamten DataSet

Data Consumer in .NET

- **CurrencyManager**
  - Die data-bound controls werden miteinander synchronisiert
  - Für jede Data Source verbunden mit dem Windows Form gibt es ein **CurrencyManager** Objekt
  - Die **Current** Eigenschaft gibt das aktuelle Element in der zugrunde liegenden Liste zurück.
  - Um das aktuelle Element zu ändern, muss man den Wert der **Position** Eigenschaft ändern
- **BindingContext**
  - Verwaltet die Kollektion der **CurrenyManger** Objekten für ein beliebiges Objekt, das von der Control-Klasse erbt
  - Siehe https://docs.microsoft.com/en-us/dotnet/api/system.windows.forms.bindingcontext?view=netframework-4.7.2

TableAdapter

- Ein **DataSet** enthält by default keine Daten
- Eine Komponente des **TableAdapter** füllt ein Dataset mit Daten aus der Datenbank, basierend auf eine oder mehrere Abfragen oder gespeicherte Prozeduren, die Sie angeben:

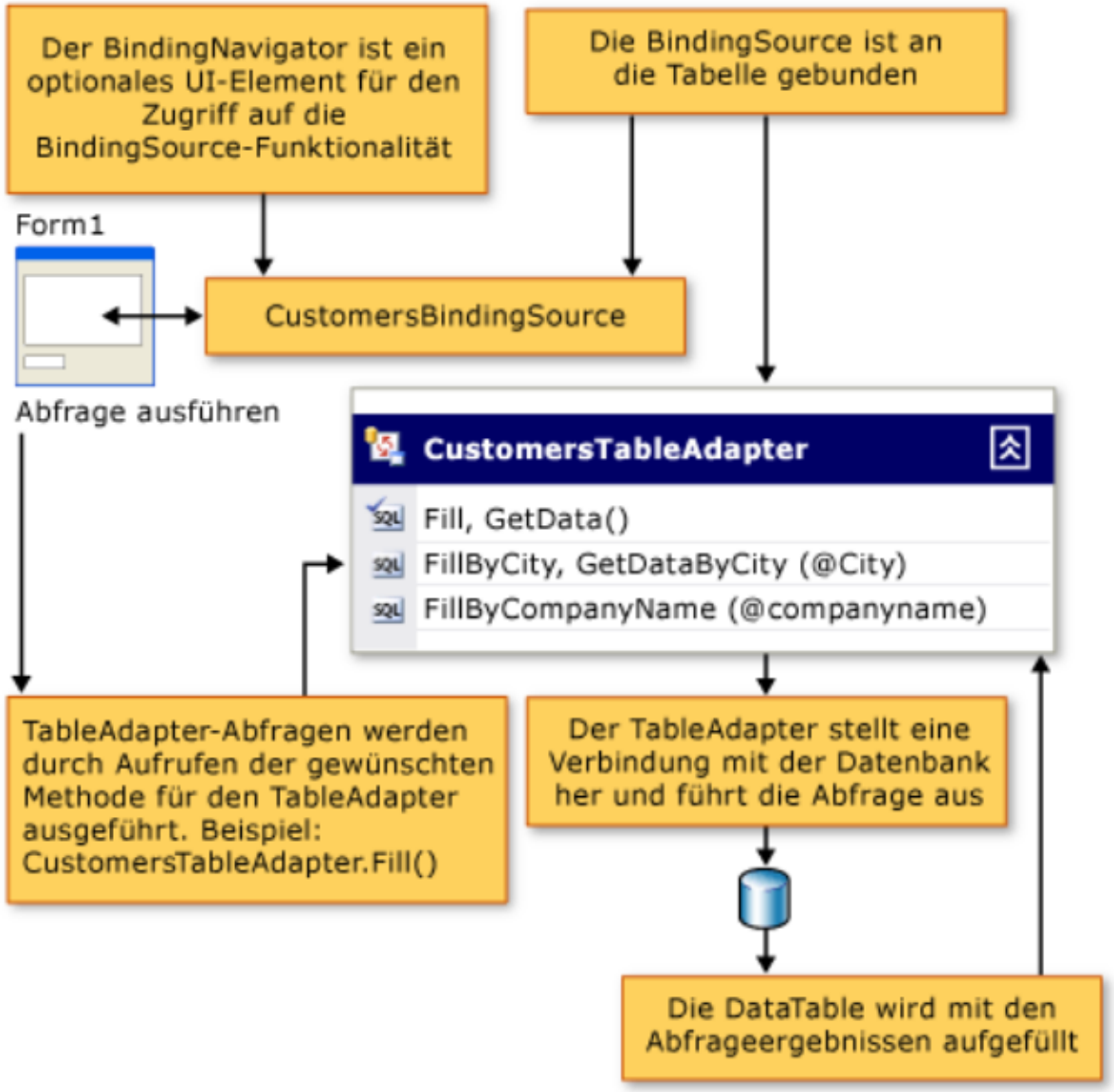
  ```
  aTableAdapter.Fill(aDataSet.TableName);
  ```

- TableAdapters senden außerdem aktualisierte Daten aus Ihrer Anwendung wieder in der Datenbank.

  ```
  aTableAdapter.Update(aDataSet.TableName)
  ```

- Die Update Methode aktualisiert und führt den richtigen Befehl (INSERT, UPDATE oder DELETE) basierend auf den **RowState** der einzelnen Datenzeilen in der Tabelle.

Interaktion zwischen TableAdapters und Datenbank



DataRelation

- Erstelle **DataRelation** Objekte, die die Beziehung zwischen den Tabellen in einem DataSet beschreibt
- Man kann ein **DataRelation** Objekt benutzten um Tupeln zu finden mithilfe der Methode **GetChildRows** für einen **DataRow** aus der Vatertabelle
- Umgekehrt kann man die Methode **GetParentRow** für einen **DataRow** aus der Kindtabelle benutzen
- Für Beispiele siehe:
  - https://docs.microsoft.com/en-us/dotnet/api/system.data.datarelation?view=netframework-4.7.2
  - https://docs.microsoft.com/en-us/dotnet/api/system.data.datatable.childrelations?view=netframework-4.7.2

Constraints

- Es gibt zwei Arten von Constraints:
  - **UniqueConstraint** – überprüft ob die Werte für eine Spalte eindeutig sind
  - **ForeignKeyConstraint** – wenn man ein DataRelation erstellt, dann wird automatisch ein FK Constraint erstellt
- Ein **DataTable** hat eine Kollektion von **Constraints**
- Ein DataSet hat einen Boolean Eigenschaft **EnforceConstraints**, die angibt ob der Constraint erzwungen wird oder nicht (by default ist es wahr)
- Siehe https://docs.microsoft.com/en-us/dotnet/api/system.data.constraint?view=netframework-4.7.2

Constraint - Beispiel

```
// Declare parent column and child column variables.
DataColumn parentColumn,childColumn;

ForeignKeyConstraint foreignKeyConstraint;

// Set parent and child column variables.

parentColumn = dataSet.Tables[table1].Columns[column1];
childColumn = dataSet.Tables[table2].Columns[column2];

foreignKeyConstraint = new ForeignKeyConstraint
("SupplierForeignKeyConstraint", parentColumn, childColumn);

// Set null values when a value is deleted.
foreignKeyConstraint.DeleteRule = Rule.SetNull;
foreignKeyConstraint.UpdateRule = Rule.Cascade;

// Add the constraint, and set EnforceConstraints to true.
dataSet.Tables[table1].Constraints.Add(foreignKeyConstraint);

dataSet.EnforceConstraints = true;
```