C# Programming Techniques

Peter Demeester

Academiejaar 2017-2018

ALVORENS TE BEGINNEN

C# Programming Techniques

OPO C# Programming Techniques (7 stp)

- OLA C# Programming Techniques (Theorie) (1 stp)
  - Theorie-examen: gesloten boek
- OLA C# Programming Techniques (Lab) (3 stp)
  - Labo's op GIT (+/- zoals in C# OO Programming) (40%)
  - Labo-examen (60%)
- OLA Advanced Applied Programming (3 stp)
  - 2 opdrachten: 1ste opdracht is 1 stp, 2de opdracht is 2 stp waard!
  - Ter herinnering: 1 stp komt overeen met 25 tot 30 uren werk

C# Programming Techniques

Totaal OPO-cijfer:

- (1/7 * theorie) + (3/7 * lab) + (3/7 * GP)
- Uitzondering: extreem tekort op 1 van de OLA's

AAP, C# PT => woensdag 3de lestijd

Labo C# PT + uitleg/feedback over AAP
=> donderdag 3de lestijd

individuele feedback van labo's +
oplossing demonsteren!

Deze week: nog geen labo => hou Toledo in de gaten!

Om de 3 weken oplossing demonstreren

Introductie

ADO.NET?

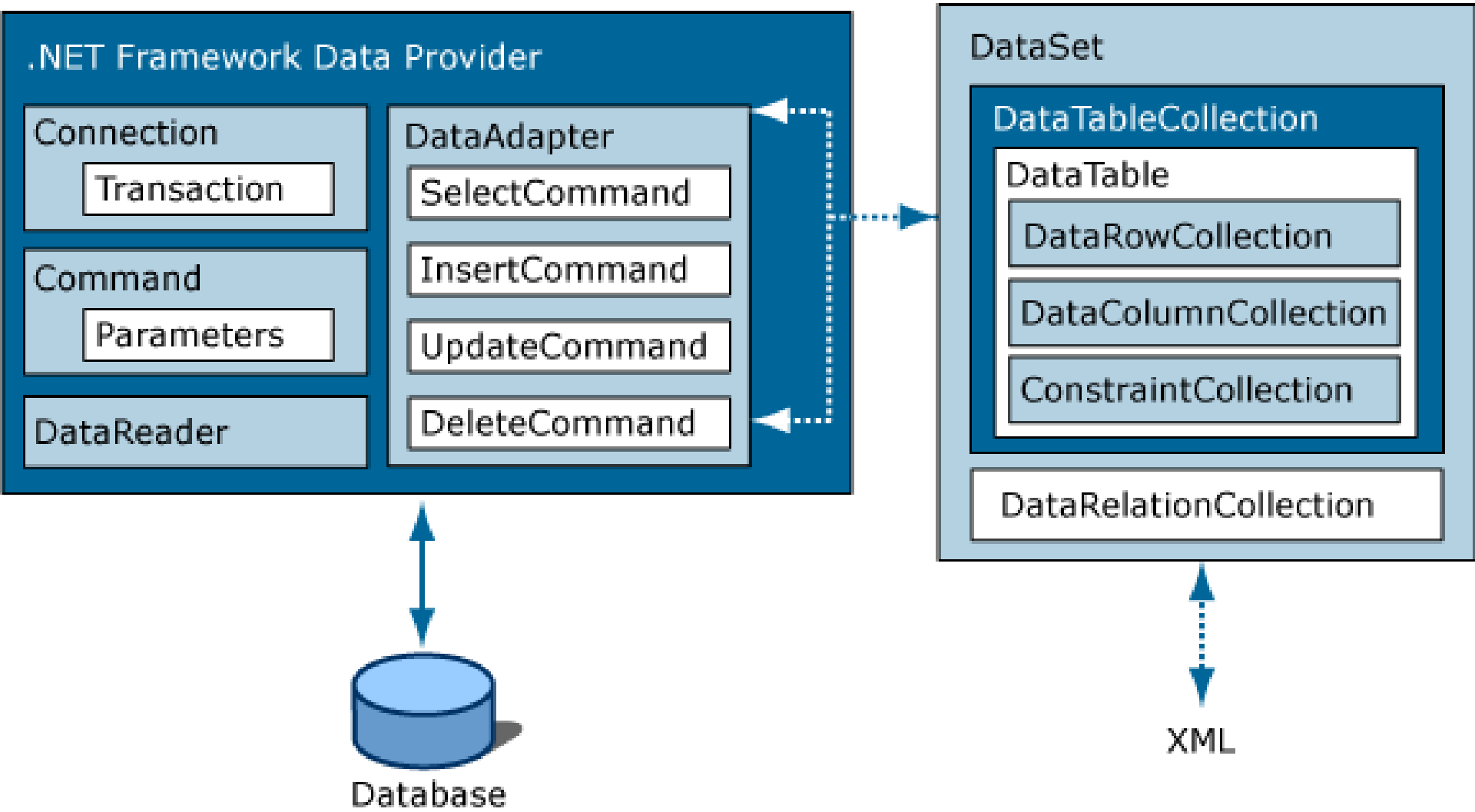
ADO.NET: objectgeöriënteerde verzameling van bibliotheken die toelaat om met verschillende "databronnen" te interageren

Databron?

- Databank
- Tekstbestand
- Excel
- XML file

We zullen ons vooral concentreren op databanken

ADO.NET architectuur



DATA PROVIDERS

Data Providers

- **ADO.NET voorziet een gemeenschappelijke manier om met verschillende databronnen te interageren**
- **Toch zijn er uiteraard verschillende bibliotheken, afhankelijk van soort databron**
- **Deze bibliotheken worden DataProviders genoemd**
  - Afhankelijk van databron of protocol

Voorbeelden van Data Providers

Provider Name	API prefix	Data Source Description
ODBC Data Provider	Odbc	Data Sources with an ODBC interface. Normally older data bases.
OleDb Data Provider	OleDb	Data Sources that expose an OleDb interface, i.e. Access or Excel.
Oracle Data Provider	Oracle	For Oracle Databases.
SQL Data Provider	Sql	For interacting with Microsoft SQL Server.
Borland Data Provider	Bdp	Generic access to many databases such as Interbase, SQL Server, IBM DB2, and Oracle.

Voorbeeld

Stel dat je een connectie wilt maken met een Excel werkblad

- **OleDb**DataProvider gebruiken om te connecteren met databron (in dit geval Excel) die **OleDb** interface implementeert
- Daartoe gebruiken we **OleDb**Connection

Indien we zouden willen connecteren met een SQL Server

- **Sql**Connection

Voor MySql wordt dit dan

- **MySql**Connection

.NET Framework Data Providers Objects

Verschillende objecten die kunnen gebruikt worden om met data te werken

Connection: connectie met databron

- Vb: SqlConnection: identificeert databank server, databanknaam, gebruikersnaam, paswoord + andere parameters die nodig zijn om te connecteren met databank
- Dit connectie-object wordt gebruikt door het command-object, zodat databank gekend is waarop commando's moeten worden uitgevoerd

Command:

- databank commando's om data terug te geven, te wijzigen, om stored procedures te runnen, sturen en ontvangen van parameter informatie
- Vb: MySqlCommand

DataReader:

- voorziet een high-performance stream van data van de databron
- "Fast, forward-only, read-only access to data". Enkel mogelijk om data uit de stream te halen op een sequentiële manier. *Goed voor snelheid, maar als je data wilt manipuleren: beter met DataSet*
- Vb: SqlDataReader

.NET Framework Data Providers Objects

DataAdapter: brug tussen DataSet en datasource

- Gebruikt command objecten om SQL commando's uit te voeren: SELECT, INSERT, UPDATE, DELETE
- Veranderingen aan DataSet doorsturen naar databron (databank)
- Vb: SqlDataAdapter

Buitenbeentje: DataSet:

- in memory representatie van data
- disconnected
- Kunnen rijen en kolommen bevatten, net zoals databanktabellen + ook relaties tussen tabellen (FK – PK)
- *Wordt gebruikt door alle DataProviders: heeft dus geen specifieke prefix!!!*

DBCONNECTION

DbConnection object

Connection object:

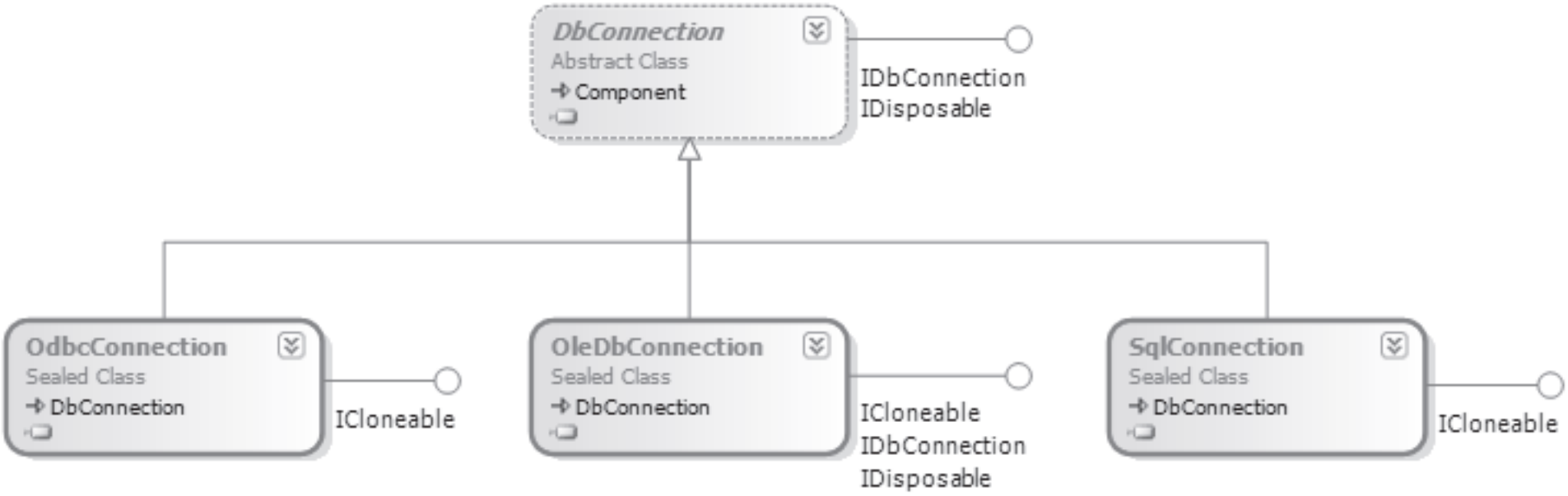
- Connectie maken met databank
- Beheert alle low-level logica geassocieerd met de specifieke databankprotocollen

Connectie object instantiëren, connectie openen en daarna sluiten

Connecties kunnen belangrijk zijn:

- Denk maar aan een website die met databank gekoppeld is
- Als er duizenden mensen tegelijk databank raadplegen, kan tot problemen leiden indien niet goed geïmplementeerd

DbConnection



SqlConnection object

Declareren van object gebeurt als volgt:

```
SqlConnection conn = new SqlConnection("Data Source=(local);
  Initial Catalog=Northwind; Integrated Security=SSPI");
```

Parameters van de constructor:

Connection String Parameter Name	Description
Data Source	Identifies the server. Could be local machine, machine domain name, or IP Address.
Initial Catalog	Database name.
Integrated Security	Set to SSPI to make connection with user's Windows login
User ID	Name of user configured in SQL Server.
Password	Password matching SQL Server User ID.

Voorbeeld:
```
SqlConnection conn = new SqlConnection("Data Source =
DatabaseServer; Initial Catalog=Northwind; User ID=YourUserID;
Password=YourPassword");
```

Gebruik van SqlConnection

Andere ADO.NET objecten zoals SqlCommand of SqlDataAdapter gebruiken een connection object als parameter

De volgorde van de operaties tijdens het bestaan van een SqlConnection

1. Instantiëren van SqlConnection
2. Connectie openen
3. Geef het connectie object door aan andere ADO.NET objecten
4. Voer andere database operaties uit met ADO.NET objecten
5. Sluit de connectie

Gebruik van SqlConnection: codevoorbeeld

```
using System;
using System.Data;
using System.Data.SqlClient;
/// <summary>
/// Demonstrates how to work with SqlConnection objects
/// </summary>
class SqlConnectionDemo {
    static void Main() {
        // 1. Instantiate the connection
        SqlConnection conn = new SqlConnection(
        "Data Source=(local);Initial Catalog=Northwind;
        Integrated Security=SSPI");
        SqlDataReader rdr = null;
```

Gebruik van SqlConnection: codevoorbeeld

```
try {
        // 2. Open the connection
        conn.Open();
        // 3. Pass the connection to a command object
        SqlCommand cmd = new SqlCommand("select * from
                                        Customers", conn);
        // 4. Use the connection, get query results
        rdr = cmd.ExecuteReader();
        // print the CustomerID of each record
        while (rdr.Read()) {
            Console.WriteLine(rdr[0]);
        }
    }
```

Gebruik van SqlConnection: codevoorbeeld

```
        finally {
            // close the reader
            if (rdr != null) {
                rdr.Close();
            }

            // 5. Close the connection
            if (conn != null) {
                conn.Close();
            }
        }
    }
}
```

Uitleg codevoorbeeld SqlConnection

- Connectie openen door `Open()` methode van `SqlConnection` object `conn` te gebruiken
- Alvorens een `SqlCommand` te gebruiken, moet je ADO.NET laten weten welke connectie het nodig heeft. In dit geval `conn`
- Elke operatie gebaseerd op `SqlCommand` zal gebruik maken van deze connectie
- `SqlCommand` voert een query uit op de Customers tabel
- Result set die teruggeven wordt is een `SqlDataReader`
- While loop leest de eerst kolom (`rdr[0]` of `rdr["CustomerID"]`)

Uitleg codevoorbeeld SqlConnection

- **Op het einde: sluiten van connection object**
- **Wanneer dat niet gebeurt:**
  - Serieuze consequenties op gebied van performantie en schaalbaarheid van de applicatie
- **`Close()` methode wordt opgeroepen in finally blok**
- **Finally blok zorgt ervoor dat alles binnen het blok uitgevoerd wordt, onafhankelijk of ervoor een exceptie of niet wordt opgeworpen**
- **Daarom:**
  - Zorg er steeds voor dat connectie steeds gesloten worden
- **Zorg er ook voor dat connectie nooit null is**
- ***Bij een `SqlDataAdapter` is het open en sluiten van een connectie niet nodig (zie later)***
- **Bekijk ook eens: http://www.connectionstrings.com**

MySqlConnection

Syntax is een beetje anders dan voor SqlConnection

```
private MySqlConnection connection;
private void Initialize()  {
        server = "localhost";
        database = “myDatabase";
        uid = “peter";
        password = “Azerty123";
        string connectionString;
        connectionString = "SERVER=" + server + ";" +
        "DATABASE=" + database + ";" + "UID=" + uid + ";" +
        "PASSWORD=" + password + ";";
        connection = new MySqlConnection(connectionString);
    }
```

DBCOMMAND

DbCommand-SqlCommand object

DbCommand

- Beschrijft welk type interactie je met de databank wilt
- Bijv:
  - Select
  - Modify
  - Insert
  - Delete
- Commando's op rijen in een tabel

SqlCommand object creëren

```
SqlCommand cmd = new SqlCommand("select CategoryName
from Categories", conn);
```

Constructor bestaat uit een query, die zegt wat er moet gebeuren, en een referentie naar een SqlConnection object

Belangrijkste methodes van SqlCommand

- `ExecuteReader` => voert query uit en retourneert resultaat als een `SqlDataReader` object
- `ExecuteNonQuery` => voert query uit en geeft # rijen die beïnvloed zijn terug
- `ExecuteScalar` => voert query uit en retourneert 1ste kolom van de 1ste rij

Querying data

```
// 1. Instantiate a new command with a
query and connection
SqlCommand cmd = new SqlCommand("select
CategoryName from Categories", conn);
```

```
// 2. Call Execute reader to get query results
SqlDataReader rdr = cmd.ExecuteReader();
```

Om select commando uit te voeren

- ExecuteReader() methode uitvoeren
- Resultaat is een SqlDataReader (zie later)

Inserting Data

```
// prepare command string
string insertString = "insert into Categories
  (CategoryName, Description) values ('Miscellaneous',
'Whatever doesn''t fit elsewhere')";

 // 1. Instantiate a new command with a query and connection
SqlCommand cmd = new SqlCommand(insertString, conn);

 // 2. Call ExecuteNonQuery to send command
cmd.ExecuteNonQuery();
```

Inserting Data

Ipv string als eerste parameter

- Variabele insertString
- Bemerk " in doesn"t
  - Escape karakter

`ExecuteNonQuery` method op SqlCommand object

- Geeft geen resultaat terug
- Voert enkel insert commando uit

Updating Data

```
// prepare command string
 string updateString = "update Categories
                        set CategoryName = 'Other'
                        where CategoryName = 'Miscellaneous‘ ";

 // 1. Instantiate a new command with command text only
 SqlCommand cmd = new SqlCommand(updateString);

 // 2. Set the Connection property
 cmd.Connection = conn;

 // 3. Call ExecuteNonQuery to send command
 cmd.ExecuteNonQuery();
```

Deleting Data

```
// prepare command string
string deleteString = “delete from Categories
                      where CategoryName = 'Other‘ ";
// 1. Instantiate a new command
SqlCommand cmd = new SqlCommand();
// 2. Set the CommandText property
cmd.CommandText = deleteString;
// 3. Set the Connection property
cmd.Connection = conn;
// 4. Call ExecuteNonQuery to send command
cmd.ExecuteNonQuery();
```

Eén enkel waarde terugkrijgen

```
// 1. Instantiate a new command
SqlCommand cmd = new SqlCommand("select count(*)
from Categories", conn);

// 2. Call ExecuteScalar to send command
int count = (int)cmd.ExecuteScalar();
```

Return type van methode `ExecuteScalar()` is een object

- Vandaar casten!

Codevoorbeeld: alles te samen

```
using System;
 using System.Data;
 using System.Data.SqlClient;

 /// <summary>
 /// Demonstrates how to work with SqlCommand objects
 /// </summary>
 class SqlCommandDemo {
     SqlConnection conn;
     public SqlCommandDemo(){
         // Instantiate the connection
         conn = new SqlConnection("Data Source=(local);Initial Catalog
                =Northwind; Integrated Security=SSPI");
     }
```

Codevoorbeeld: alles te samen

```
// call methods that demo SqlCommand capabilities
static void Main() {
    SqlCommandDemo scd = new SqlCommandDemo();
    Console.WriteLine();
    Console.WriteLine("Categories Before Insert");
    Console.WriteLine("-----------------------");
    // use ExecuteReader method
    scd.ReadData();
    // use ExecuteNonQuery method for Insert
    scd.Insertdata();
    Console.WriteLine();
    Console.WriteLine("Categories After Insert");
    Console.WriteLine("---------------------------");
    scd.ReadData();
```

Codevoorbeeld: alles te samen

```
// use ExecuteNonQuery method for Update
scd.UpdateData();
Console.WriteLine();
Console.WriteLine("Categories After Update");
Console.WriteLine("------------------------------");
scd.ReadData();
// use ExecuteNonQuery method for Delete
scd.DeleteData();
Console.WriteLine();
Console.WriteLine("Categories After Delete");
Console.WriteLine("------------------------------");
scd.ReadData();
```

Codevoorbeeld: alles te samen

```
    // use ExecuteScalar method
    int numberOfRecords = scd.GetNumberOfRecords();
    Console.WriteLine();
    Console.WriteLine("Number of Records: {0}", numberOfRecords);
}
/// <summary>
/// use ExecuteReader method
/// </summary>
public void ReadData() {
   SqlDataReader rdr = null;
    try {
        // Open the connection
        conn.Open();
```

Codevoorbeeld: alles te samen

```
    // 1. Instantiate a new command with a query and connection
    SqlCommand cmd = new SqlCommand("select CategoryName from
                                     Categories", conn);
  // 2. Call Execute reader to get query results
  rdr = cmd.ExecuteReader();
  // print the CategoryName of each record
  while (rdr.Read()) {
       Console.WriteLine(rdr[0]);
  }
}
finally {
     // close the reader
     if (rdr != null) {
         rdr.Close();
     }
```

Codevoorbeeld: alles te samen

```
        // Close the connection
        if (conn != null) {
            conn.Close();
        }
    }
}
/// <summary>
/// use ExecuteNonQuery method for Insert
/// </summary>
public void Insertdata() {
    try {
        // Open the connection
        conn.Open();
```

Codevoorbeeld: alles te samen

```
    // prepare command string
    string insertString = "insert into Categories
          (CategoryName, Description) values ('Miscellaneous',
        'Whatever doesn''t fit elsewhere')";
    // 1. Instantiate a new command with a query and connection
    SqlCommand cmd = new SqlCommand(insertString, conn);
    // 2. Call ExecuteNonQuery to send command
    cmd.ExecuteNonQuery();
}
finally {
     // Close the connection
     if (conn != null) {
          conn.Close();
     }
}
}
```

Codevoorbeeld: alles te samen

```
/// <summary>
/// use ExecuteNonQuery method for Update
/// </summary>
public void UpdateData() {
    try{
        // Open the connection
        conn.Open();
        // prepare command string
        string updateString = "update Categories
                              set CategoryName = 'Other'
                              where CategoryName = 'Miscellaneous'";
        // 1. Instantiate a new command with command text only
        SqlCommand cmd = new SqlCommand(updateString);
```

Codevoorbeeld: alles te samen

```
        // 2. Set the Connection property
        cmd.Connection = conn;

        // 3. Call ExecuteNonQuery to send command
        cmd.ExecuteNonQuery();
    }
     finally {
        // Close the connection
        if (conn != null) {
            conn.Close();
        }
     }
}
```

Codevoorbeeld: alles te samen

```
/// <summary>
/// use ExecuteNonQuery method for Delete
/// </summary>
public void DeleteData(){
    try{
        // Open the connection
        conn.Open();
        // prepare command string
        string deleteString = "delete from Categories
                              where CategoryName = 'Other'";
        // 1. Instantiate a new command
        SqlCommand cmd = new SqlCommand();
```

Codevoorbeeld: alles te samen

```
        // 2. Set the CommandText property
        cmd.CommandText = deleteString;
        // 3. Set the Connection property
        cmd.Connection = conn;
        // 4. Call ExecuteNonQuery to send command
        cmd.ExecuteNonQuery();
    }
    finally {
        // Close the connection
        if (conn != null) {
            conn.Close();
        }
    }
}
```

Codevoorbeeld: alles te samen

```
/// <summary>
/// use ExecuteScalar method
/// </summary>
/// <returns>number of records</returns>
public int GetNumberOfRecords() {
    int count = -1;
    try{
        // Open the connection
        conn.Open();
        // 1. Instantiate a new command
        SqlCommand cmd = new SqlCommand("select count(*) from
                                        Categories", conn);
        // 2. Call ExecuteScalar to send command
        count = (int)cmd.ExecuteScalar();
    }
```

Codevoorbeeld: alles te samen

```
        finally {
             // Close the connection
              if (conn != null) {
                  conn.Close();
              }
           }
           return count;
       }
}
```

DBDATAREADER

Data lezen met SqlDataReader

SqlDataReader wordt gebruikt om data te lezen op de meest efficiënte manier

Het kan NIET gebruikt worden om data weg te schrijven

Wordt meestal omschreven als

- "Fast-forward firehose-like stream of data"

Data kan enkel gelezen worden op een forward-only sequential manier

- Eens data gelezen is, moet je het ergens opslaan want je kan **niet** terug keren en het opnieuw lezen
- Doordat enkel forward-only mogelijk is:
  - Heel snel!
  - Geen overhead!

Dus: als je enkel een groep data éénmaal moet lezen en je wilt dat het vlug gaat, dan is SqlDataReader de beste keuze!

Ook als de data zodanig groot is dat het niet allemaal in RAM geheugen kan => gebruik SqlDataReader!

SqlDataReader object creëren

```
SqlDataReader rdr = cmd.ExecuteReader();
```

Om een `SqlDataReader` aan te maken moet je `ExecuteReader` methode van `SqlCommand` aanroepen, die een `SqlDataReader` object retourneert

Data lezen

Om data te lezen

- Rij per rij uit tabel halen
- Eens rij gelezen is, is ze niet meer beschikbaar
- Om de rij opnieuw te kunnen lezen, moet je een nieuwe instantie aanmaken, en de datastream opnieuw lezen

Typisch wordt het lezen gedaan a.d.h.v. een while loop

Data lezen: codevoorbeeld

```
while (rdr.Read()) {
      // get the results of each column
      string contact = (string)rdr["ContactName"];
      string company = (string)rdr["CompanyName"];
      string city    = (string)rdr["City"];

      // print out the results
      Console.Write(contact);
      Console.Write(city);
      Console.Write(company);
      Console.WriteLine();
}
```

Data lezen: codevoorbeeld

Read methode van `SqlDataReader` retourneert een boolean

- True als er nog rijen te lezen zijn
- Wanneer laatste record gelezen is => false

Merk op dat je zowel via de kolomindex als via de kolomnaam data van een kolom kan halen

Vergeet niet te casten

Vergeet niet om je `SqlDataReader` te sluiten

Codevoorbeeld

```
using System;
using System.Data;
using System.Data.SqlClient;
namespace SqlDataReaderExample{
    class ReaderDemo {
        static void Main() {
            ReaderDemo rd = new ReaderDemo();
            rd.SimpleRead();
        }
        public void SimpleRead() {
            // declare the SqlDataReader, which is used in
            // both the try block and the finally block
            SqlDataReader rdr = null;
```

Codevoorbeeld

```
// create a connection object
SqlConnection conn = new SqlConnection(
    "Data Source=(local);Initial Catalog=Northwind;
    Integrated Security=SSPI");
// create a command object
SqlCommand cmd  = new SqlCommand("select * from Customers", conn);
try {
    // open the connection
    conn.Open();
    // 1. get an instance of the SqlDataReader
    rdr = cmd.ExecuteReader();
```

Codevoorbeeld

```
    // print a set of column headers
    Console.WriteLine("Contact Name           City              Company Name");
    Console.WriteLine("------------           ------------      ------------");
    // 2. print necessary columns of each record
    while (rdr.Read()) {
        // get the results of each column
        string contact = (string)rdr["ContactName"];
        string company = (string)rdr["CompanyName"];
        string city    = (string)rdr["City"];
        // print out the results
        Console.Write(contact);
        Console.Write(city);
        Console.Write(company);
        Console.WriteLine();
    }
}
```

Codevoorbeeld

```
        finally{
            // 3. close the reader
            if (rdr != null){
                rdr.Close();
            }
            // close the connection
            if (conn != null){
                conn.Close();
            }
        }
      }
    }
}
```

DataReader

Heel geschikt om heel snel data te lezen

Ideaal om bijv.

- ListBox of DropDownList te populeren

Indien je data wilt wijzigen en terug sturen naar databank

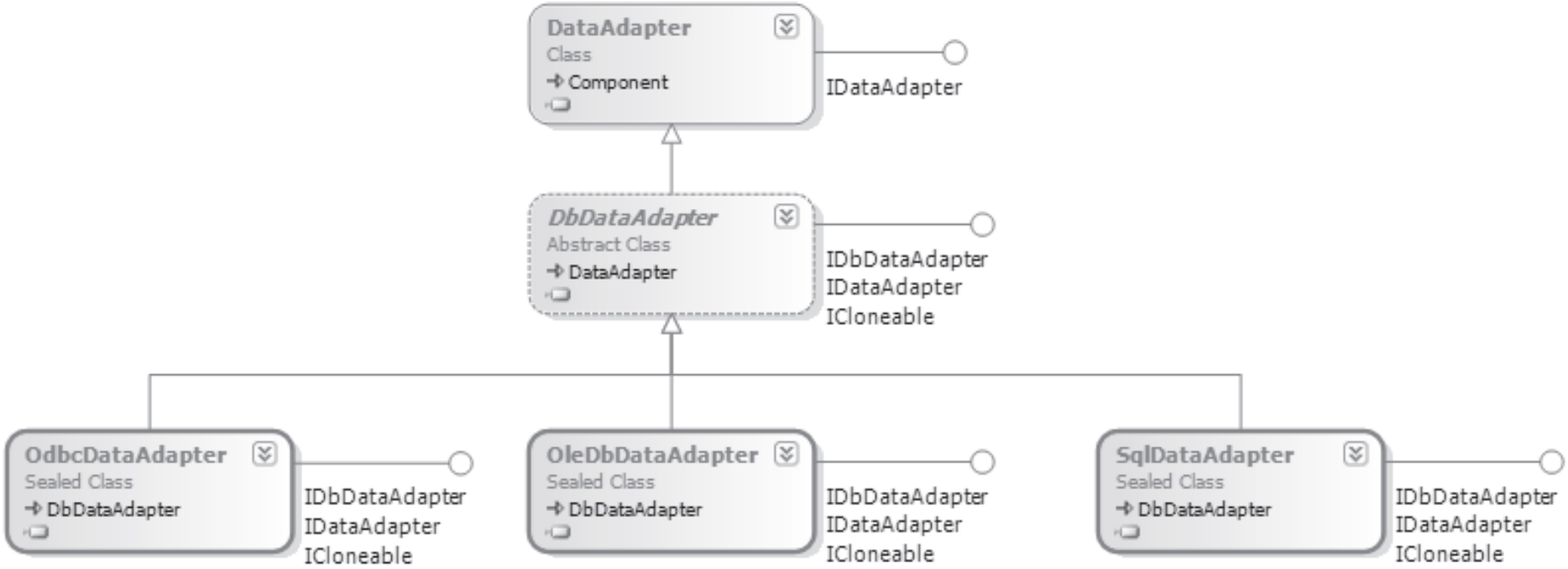
- Gebruik `DataAdapter` (zie straks)

DataReader

```
SqlConnection connection = new SqlConnection();
SqlCommand cmd = connection.CreateCommand();
cmd.CommandType = CommandType.Text;
cmd.CommandText = "SELECT ProductID, ProductName FROM Products";
connection.Open();
SqlDataReader rdr = cmd.ExecuteReader();
DataTable products = new DataTable();
products.Load(rdr, LoadOption.PreserveChanges);
connection.Close();
cmbProducts.DataSource = products;
cmbProducts.DisplayMember = "ProductName";
cmbProducts.ValueMember = "ProductID";
```

DBDATAADAPTER

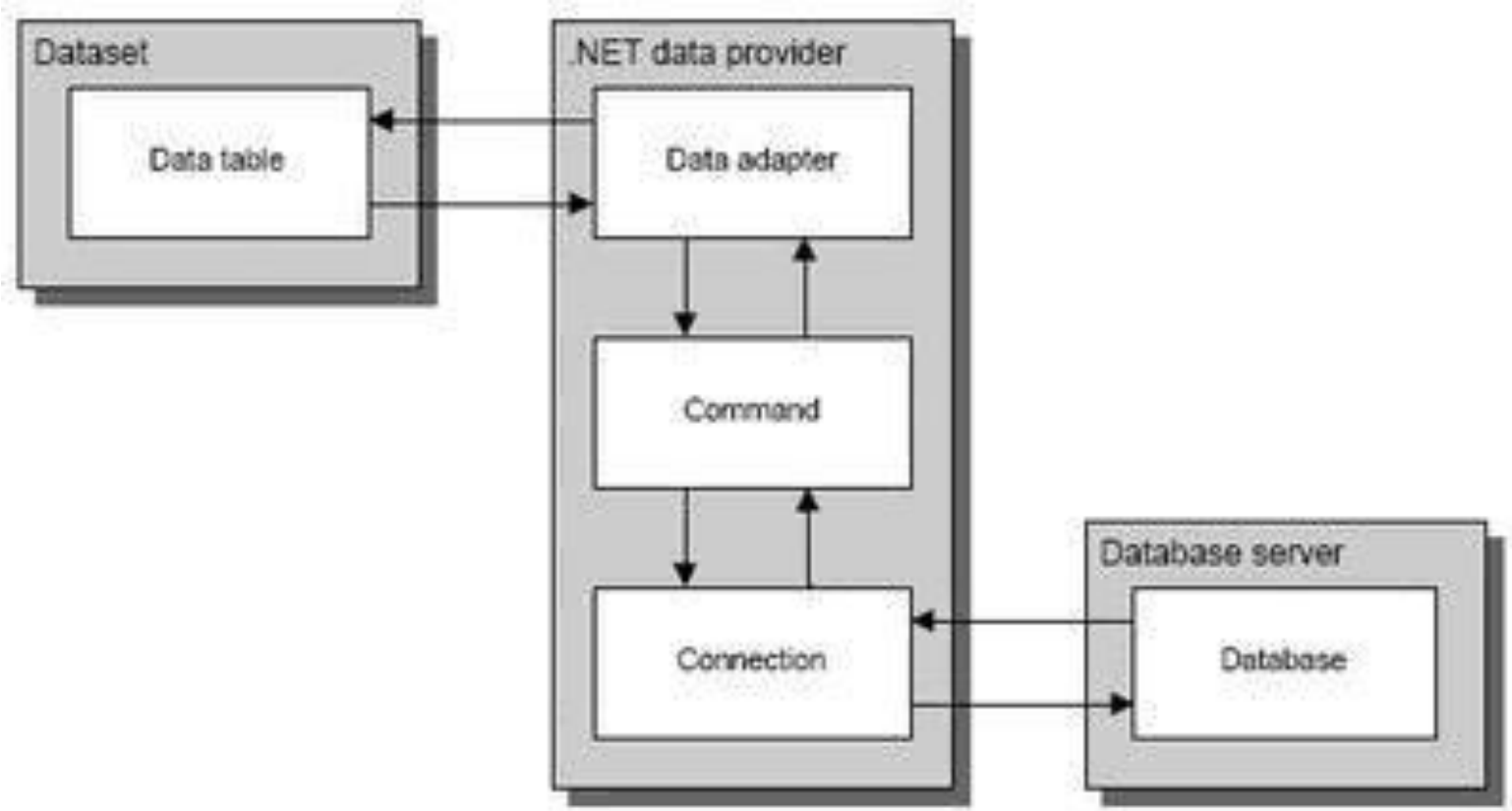
DbDataAdapter-SqlDataAdapter



DbDataAdapter

`DbDataAdapter` wordt gebruikt om data tussen data source (databank, Excel file,…) en datatable uit te wisselen (opvragen en updaten)

DataSet vs DataAdapter



DataSet vs SqlDataAdapter

`SqlDataAdapter` voert de volgende taken uit

- om een `DataSet` op te vullen met data
  - Connectie openen
  - Data ontvangen en in `DataSet` stoppen
  - Connectie sluiten
- Om een databron te updaten met de veranderingen aangebracht in `DataSet`
  - Connectie openen
  - Veranderingen schrijven van `DataSet` naar databron
  - Connectie sluiten

Werken met gedisconnecteerde data: DataSet en SqlDataAdapter

DataSet

- In-memory data opslag die verschillende tabellen kan bewaren
- Kan enkel data bewaren en niet met de databron interageren

SqlDataAdapter

- Beheert de connectie met de databron
- Opent connectie enkel wanneer nodig, en sluit het van zodra werk uitgevoerd is

Een DataSet en SqlDataAdapter object creëren

```
DataSet dsCustomers = new DataSet();
SqlDataAdapter daCustomers = new
SqlDataAdapter("select CustomerID,
CompanyName from Customers", conn);
```

SQL select statement zegt welke data naar DataSet zal gestuurd worden

conn moet reeds bestaan, maar nog niet geopend zijn: dit is de verantwoordelijkheid van de `SqlDataAdapter`

`SqlDataAdapter` bevat alle commando's om te interageren met de databron

Creëren van een SqlDataAdapter: insert, update en delete

Manier om insert, update of delete commando's automatisch toe te voegen

```
SqlCommandBuilder cmdBldr = new SqlCommandBuilder(daCustomers);
```

Hiervoor heb je wel een bestaande `SqlDataAdapter` (=`daCustomers`) nodig

Hierdoor weet `SqlCommandBuilder` aan welke `SqlDataAdapter` hij commando's moet toevoegen (zie ook volgende les!)

Meer informatie over SqlCommandBuilder: zie http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlcommandbuilder.aspx

DATASET

DataSet opvullen

```
daCustomers.Fill(dsCustomers, "Customers");
```

Aan de hand van bovenstaande commando (`Fill` methode van SqlDataAdapter) kan de DataSet opgevuld worden

Fill methode gebruikt 2 parameters:

- DataSet: moet reeds geïnstantieerd zijn
- Tabelnaam:
  - naam van de tabel die zal gecreëerd worden in DataSet
  - Naam mag vrij gekozen worden
  - Wordt enkel gebruikt om er later naar te kunnen verwijzen

Fill methode heeft ook versie (remember overloading) met slechts 1 parameter

- Enkel DataSet
- Tabelnaam krijgt dan een defaultnaam: bijv. table1

DataSet gebruiken

Code om DataSet te binden aan DataGrid in een Form

```
dgCustomers.DataSource = dsCustomers;
dgCustomers.DataMember = "Customers";
```

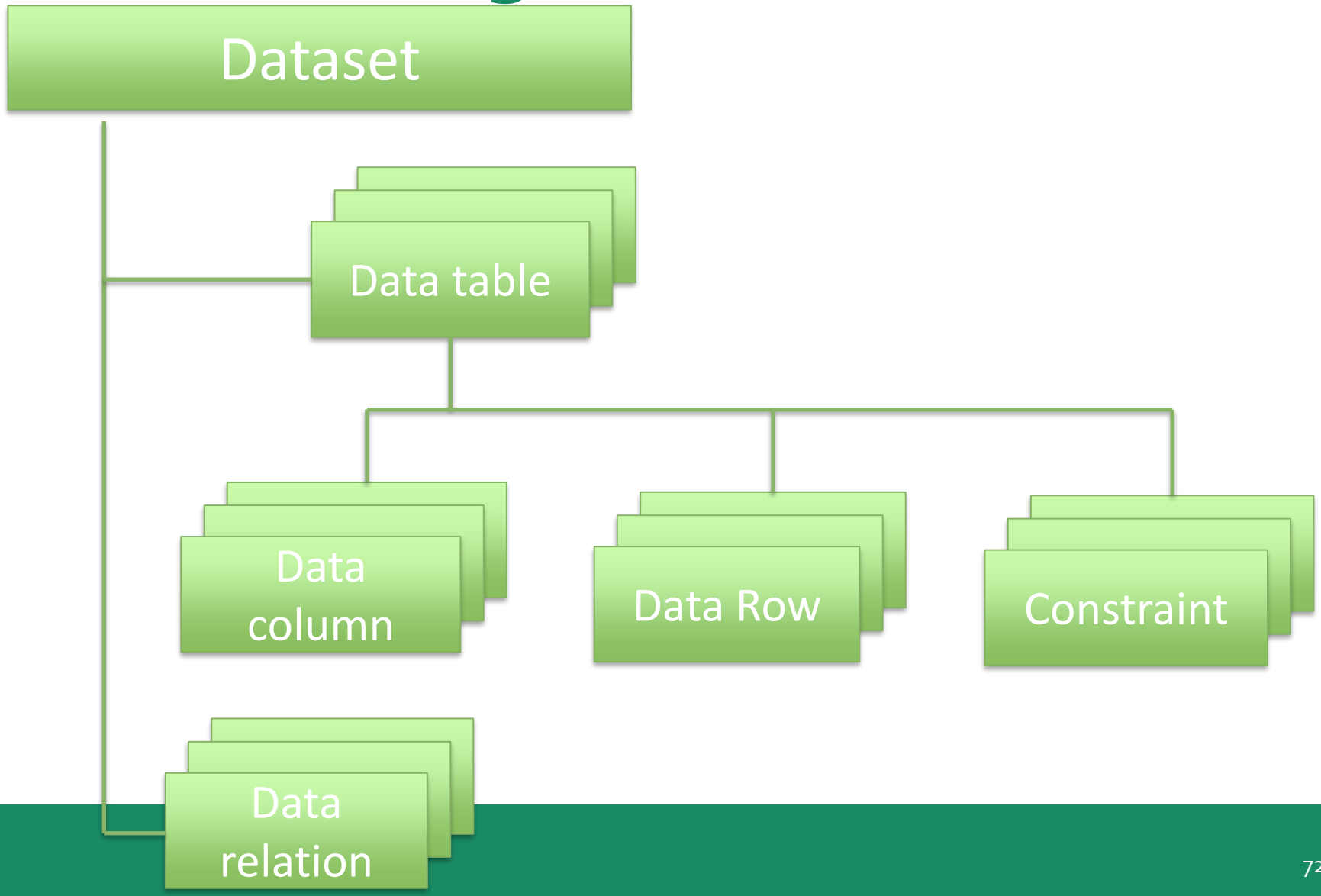
Veranderingen updaten

Na veranderingen aan data (in DataSet) aangebracht

- Wijzigingen terugschrijven naar databank:

```
daCustomers.Update(dsCustomers,
"Customers");
```

Dataset organisatie



Dataset organisatie

Dataset:

- Georganiseerd zoals relationele databank
- Kan meerdere tabellen bevatten
- Elke tabel kan meerdere kolommen en rijen bevatten
- Elke tabel kan 1 of meer constraints bevatten
  - Definiëren unieke sleutel, foreign key

Onthou dat:

- Elke tabel correspondeert met resultset die geretourneerd wordt door SELECT statement
- Alle objecten worden opgeslaan in collections

Dataset klassen

Properties van Dataset klasse

- DataSetName
- Tables
  - aDataSet.Tables.Count()
  - aDataSet.Tables[“Vendors”]
- Relations

Properties en methodes van DataTable klasse

- TableName (P)
- Columns (P)
- Rows (P)
- Constraints (P)
- NewRow (M)

Dataset klassen

Properties van DataColumn

- ColumnName
- AllowDBNull
- AutoIncrement

Properties en methodes van DataRow

- Indexer (P) => geeft toegang tot specifieke kolom van de rij
- Delete (M)
- IsNull (M)

Codevoorbeeld

```
string message="";
foreach(DataRow dr in
      vendorsDataSet.Tables["Vendors"].Rows){
         message += dr["Name"] + "\n";
}
MessageBox.Show(message);
```

Alles samenvoegen

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Windows.Forms;

class DisconnectedDataform : Form {
    private SqlConnection conn;
    private SqlDataAdapter daCustomers;
    private DataSet dsCustomers;
    private DataGrid dgCustomers;
    private const string tableName = "Customers";
```

Alles samenvoegen

```
// initialize form with DataGrid and Button
public DisconnectedDataform() {
    // fill dataset
    Initdata();
    // set up datagrid
    dgCustomers = new DataGrid();
    dgCustomers.Location = new Point(5, 5);
    dgCustomers.Size = new Size(
                this.ClientRectangle.Size.Width - 10,
                this.ClientRectangle.Height - 50);
    dgCustomers.DataSource = dsCustomers;
    dgCustomers.DataMember = tableName;
```

Alles samenvoegen

```
    // create update button
    Button btnUpdate = new Button();
    btnUpdate.Text = "Update";
    btnUpdate.Location = new Point(
        this.ClientRectangle.Width/2 - btnUpdate.Width/2,
        this.ClientRectangle.Height - (btnUpdate.Height + 10));
    btnUpdate.Click += new EventHandler(btnUpdateClicked);

    // make sure controls appear on form
    Controls.AddRange(new Control[] { dgCustomers, btnUpdate });
}
```

Alles samenvoegen

```
// set up ADO.NET objects
public void Initdata() {
    // instantiate the connection
    conn = new SqlConnection( "Server=(local); DataBase=Northwind;
                               Integrated Security=SSPI");
    // 1. instantiate a new DataSet
    dsCustomers = new DataSet();
    // 2. init SqlDataAdapter with select command and connection
    daCustomers = new SqlDataAdapter(
        "select CustomerID, CompanyName from Customers", conn);
    // 3. fill in insert, update, and delete commands
    SqlCommandBuilder cmdBldr = new SqlCommandBuilder(daCustomers);
```

Alles samenvoegen

```
        // 4. fill the dataset
        daCustomers.Fill(dsCustomers, tableName);
    }
    // Update button was clicked
    public void btnUpdateClicked(object sender, EventArgs e)
    {
        // write changes back to DataBase
        daCustomers.Update(dsCustomers, tableName);
    }
    // start the Windows form
    static void Main()
    {
        Application.Run(new DisconnectedDataForm());
    }
}
```

PARAMETERS

Parameters toevoegen aan commando's

Stel

- Je wilt alle klanten weten uit een bepaalde stad
- Mogelijke (slechte!!!) oplossing:

```
// don't ever do this
   SqlCommand cmd = new SqlCommand(
   "select * from Customers where city = ‘ " + inputCity + " ’ ");
```

- Waarom is dit een slechte oplossing?

Beter manier is om gebruik te maken van parameters!

Alles geplaatst in een parameter wordt behandeld als field data (geen deel van de SQL statement) zodat je applicatie veel veiliger wordt

Meer weten over SQL Injectie? Zie bijv.
http://www.unixwiz.net/techtips/sql-injection.html

http://bobby-tables.com/csharp.html

Geparameteriseerde queries

3 stappen proces

- Maak de `SqlCommand` commando string met parameters
- Maak een `SqlParameter` object, wijs er een waarde aan toe
- Wijs `SqlParameter` object toe aan `SqlCommand` object Parameters eigenschap

SqlCommand object klaarmaken voor parameters

```
// 1. declare command object with parameter
SqlCommand cmd = new SqlCommand("select * from
Customers where city = @City", conn);
```

SqlParameter object aanmaken

```
// 2. define parameters used in command object
    SqlParameter param  = new SqlParameter();
    param.ParameterName = "@City";
    param.Value         = inputCity;
```

Associeer SqlParameter met SqlCommand object

```
// 3. add new parameter to command object
   cmd.Parameters.Add(param);
```

Alles samen

```
using System;
using System.Data;
using System.Data.SqlClient;

class ParamDemo {
    static void Main() {
        // conn and reader declared outside try block for
        // visibility in finally block
        SqlConnection conn   = null;
        SqlDataReader reader = null;
        string inputCity = "London";

        try {
```

Alles samen

```
// instantiate and open connection
conn =  new SqlConnection("Server=(local);DataBase=Northwind;
                          Integrated Security=SSPI");
conn.Open();
// don't ever do this!!!
// SqlCommand cmd = new SqlCommand(
// "select * from Customers where city = '" + inputCity + "'";
// 1. declare command object with parameter
SqlCommand cmd = new SqlCommand(
       "select * from Customers where city = @City", conn);
// 2. define parameters used in command object
SqlParameter param  = new SqlParameter();
param.ParameterName = "@City";
param.Value         = inputCity;
```

Alles samen

```
    // 3. add new parameter to command object
    cmd.Parameters.Add(param);
    // get data stream
    reader = cmd.ExecuteReader();
    // write each record
    while(reader.Read()) {
        Console.WriteLine("{0}, {1}", reader["CompanyName"],
                          reader["ContactName"]);
    }
}
```

Alles samen

```
    finally {
        // close reader
        if (reader != null) {
            reader.Close();
        }

        // close connection
        if (conn != null){
            conn.Close();
        }
        }
    }
}
```

Stored procedures gebruiken

SqlCommand kan ook gebruikt worden om stored procedures uit te voeren

Daartoe dienen 2 zaken te gebeuren

- SqlCommand object laten weten welke stored procedure er moet uitgevoerd worden
- SqlCommand object vertellen dat het een stored procedure moet uitvoeren

```
// 1. create a command object identifying the stored procedure
SqlCommand cmd  = new SqlCommand("Ten Most Expensive Products", conn);

// 2. set the command object so it knows to execute a stored procedure
cmd.CommandType = CommandType.StoredProcedure;
```

Parameters en stored procedures

```
// 1. create a command object identifying the stored procedure
SqlCommand cmd  = new SqlCommand("CustOrderHist", conn);

// 2. set the command object so it knows to execute a stored procedure
cmd.CommandType = CommandType.StoredProcedure;

// 3. add parameter to command, which will be passed to the stored procedure
cmd.Parameters.Add(new SqlParameter("@CustomerID", custId));
```

Codevoorbeeld

```
using System;
using System.Data;
using System.Data.SqlClient;

class StoredProcDemo {
    static void Main() {
        StoredProcDemo spd = new StoredProcDemo();
        // run a simple stored procedure
        spd.RunStoredProc();
        // run a stored procedure that takes a parameter
        spd.RunStoredProcParams();
    }
```

Codevoorbeeld

```
// run a simple stored procedure
public void RunStoredProc() {
    SqlConnection conn = null;
    SqlDataReader rdr  = null;
    Console.WriteLine("\nTop 10 Most Expensive Products:\n");
    try {
        // create and open a connection object
        conn = new SqlConnection("Server=(local);DataBase= Northwind;
                                  Integrated Security=SSPI");
        conn.Open();
        // 1. create a command object identifying the stored procedure
        SqlCommand cmd  = new SqlCommand("Ten Most Expensive Products",
                                         conn);
```

Codevoorbeeld

```
        // 2. set the command object so it knows to execute a stored
procedure
        cmd.CommandType = CommandType.StoredProcedure;
        // execute the command
        rdr = cmd.ExecuteReader();
        // iterate through results, printing each to console
        while (rdr.Read()) {
            Console.WriteLine("Product: {0,-25} Price: ${1,6:####.00}",
            rdr["TenMostExpensiveProducts"], rdr["UnitPrice"]);
        }
    }
```

Codevoorbeeld

```
    finally {
        if (conn != null) {
            conn.Close();
        }
        if (rdr != null){
            rdr.Close();
        }
    }
}
```

Codevoorbeeld

```
// run a stored procedure that takes a parameter
public void RunStoredProcParams()   {
    SqlConnection conn = null;
    SqlDataReader rdr  = null;
    // typically obtained from user input, but we take a short cut
    string custId = "FURIB";
    Console.WriteLine("\nCustomer Order History:\n");
    try {
        // create and open a connection object
        conn = new
        SqlConnection("Server=(local);DataBase=Northwind;Integrated
                       Security=SSPI");
        conn.Open();
```

Codevoorbeeld

```
    // 1. create a command object identifying the stored procedure
    SqlCommand cmd  = new SqlCommand("CustOrderHist", conn);
    // 2. set the command object so it knows to execute a stored procedure
    cmd.CommandType = CommandType.StoredProcedure;
    // 3. add parameter to command, which will be passed to the stored procedure
    cmd.Parameters.Add(new SqlParameter("@CustomerID", custId));
    // execute the command
    rdr = cmd.ExecuteReader();
    // iterate through results, printing each to console
    while (rdr.Read()){
        Console.WriteLine("Product: {0,-35} Total: {1,2}",
rdr["ProductName"],rdr["Total"]);
    }
}
```

Codevoorbeeld

```
        finally{
            if (conn != null) {
                conn.Close();
            }
            if (rdr != null) {
                rdr.Close();
            }
        }
    }
}
```

Bronnen

http://www.csharp-station.com/Tutorial/AdoDotNet/Lesson01

http://www.csharp-station.com/Tutorial/AdoDotNet/Lesson02

http://www.csharp-station.com/Tutorial/AdoDotNet/Lesson03

http://www.csharp-station.com/Tutorial/AdoDotNet/Lesson04

http://www.csharp-station.com/Tutorial/AdoDotNet/Lesson05

http://www.csharp-station.com/Tutorial/AdoDotNet/Lesson06

http://www.csharp-station.com/Tutorial/AdoDotNet/Lesson07

Microsoft ADO.NET 4 Step by Step

Murach's ADO.NET 4 Database Programming with C# 2010

MCTS Self-Paced Training Kit (Exam 70-516): Accessing Data with Microsoft .NET Framework 4