



**Vlaamse Dienst voor Arbeidsbemiddeling en
Beroepsopleiding**

**C#.NET
2015**

ADO.NET

Inhoudsopgave

1	INLEIDING	1
1.1	Doelstelling.....	1
1.2	Vereiste voorkennis.....	1
1.3	Nodige software	1
1.4	Situering ADO.NET	1
1.5	SqlServer en SqlExpress	2
1.6	SQL Server Management Studio	2
1.7	De voorbeelddatabase Bieren	3
1.7.1	Structuur van de database Bieren	3
1.8	De voorbeelddatabase Bank.....	6
1.8.1	Structuur van de database Bank	6
2	HET ADO.NET OBJECTMODEL	8
2.1	Inleiding	8
2.2	De .NET Data Providers.....	8
2.2.1	SqlServer.NET Data Provider	8
2.2.2	OleDb.NET Data Provider	9
2.2.3	ODBC (Open Database Connectivity) .NET Data Provider	9
2.2.4	Microsoft.NET Data Provider for Oracle	9
2.2.5	Andere Data Providers.....	9
2.2.6	Opbouw .NET Data Provider	10
2.2.7	Connection	10
2.2.8	Command	10
2.2.9	DataReader.....	11
2.3	Namespaces	11
3	JOUW SOLUTION.....	13
4	DE CONNECTION CLASS	14
4.1	Algemeen	14
4.2	Een SqlServer-database openen.....	15
4.2.1	Een SqlServer-database openen met Windows Authentication	15
4.2.2	Een SqlServer-database openen met SqlServer Authentication.....	17
4.3	DeConnectionString uit een configuratiebestand halen	19
4.3.1	App.xaml	21
4.4	De programmacode neutraal maken t.o.v. het type database	21
4.5	De databaseconnectie in een gemeenschappelijke DLL.	24
4.5.1	Het project voor de DLL zelf	24
4.5.2	De DLL aanspreken	26

4.5.3	De oefendatabase Bank	27
4.6	Connection Pooling	28
5	DE COMMAND CLASS	30
5.1	Algemeen	30
5.2	Een Command uitvoeren met ExecuteNonQuery()	32
5.3	Een SQL-statement met parameters (veranderlijke waarden).	34
5.4	Een Stored procedure oproepen	38
5.4.1	De Stored Procedure aanmaken	38
5.4.2	Parametertypes.....	40
5.4.3	De Stored Procedure aanroepen	41
5.4.4	Name based en positional based parameters.	42
6	TRANSACTIES.....	43
6.1	Algemeen	43
6.2	Transaction Isolation Levels	44
6.3	De class DbTransaction	46
6.4	De class TransactionScope	51
7	ÉÉN WAARDE UIT EEN DATABASE LEZEN.....	58
7.1	Algemeen	58
7.2	De Stored Procedure	58
7.3	De functie SaldoRekeningRaadplegen	59
8	ENKELE WAARDES LEZEN UIT EEN STORED PROCEDURE VAN SQLSERVER.....	62
8.1	Algemeen	62
8.2	Voorbeeld	62
8.2.1	De Stored Procedure maken in SqlServer.....	62
8.2.2	De functie RekeningInfoRaadplegen	64
9	AUTONUMBER-VELDEN	68
9.1	Algemeen	68
9.2	Voorbeeld	68
9.2.1	De Stored Procedure:	68
9.2.2	De functie NieuweKlant.....	69
10	DE CLASS DATAREADER.....	72
10.1	Algemeen	72

10.2	Voorbeeld	75
11	DATABINDING	79
11.1	Algemeen	79
11.2	Een lijst objecten tonen.....	80
11.3	Het Data Sources Window	81
11.4	Een Data Source gebruiken	82
11.4.1	Lay-out	84
11.4.2	Alle brouwers	85
11.4.3	Brouwers zoeken	86
11.4.4	De CollectionViewSource control	88
11.4.5	Aantal rijen	90
11.5	De DataGridView control	91
11.6	Simple data binding.....	93
11.7	Een control manueel verbinden met de BindingSource.....	95
11.8	Validation	96
11.8.1	Validation bij Simple Data Binding	97
11.8.2	Validation van Cells van Datagrid	99
11.8.3	RowValidation in Datagrid	100
11.8.4	Een ValidationRule in code wijzigen.....	101
11.8.5	Reageren op een foute validatie	103
11.9	Conversie van gegevens.....	105
11.9.1	Conversie van control naar property in object	105
11.9.2	Tonen van een decimale waarde als een valutawaarde (currency)	106
11.9.3	Opmaak en validatie	107
11.10	Gegevens sorteren	109
11.11	Gegevens filteren	109
11.12	Verwijderen en toevoegen (OnCollectionChanged).....	111
11.12.1	Objecten verwijderen	111
11.12.2	Objecten toevoegen.....	114
11.13	Veranderen van gegevens	118
12	MULTI-USER PROGRAMMA'S	122
12.1	Algemeen	122
12.2	De Database strips.....	122
12.3	Conflicten opvangen.	123
12.3.1	WPF-Window StripFiguren	123
12.3.2	Een tweede WPF-applicatie	127
12.3.3	Exceptions	128
12.3.4	Conflicten	129
13	COLOFON	130

1 INLEIDING

1.1 Doelstelling

Deze cursus behandelt het objectmodel van ADO.NET, het databasetoegangsmiddel binnen .NET. Je leert hoe je dit model kan gebruiken voor het ontwerpen van WPF-projecten die gebonden zijn aan databasegegevens. Het is de bedoeling dat je de ADO.NET-objecten leert kennen, dat je weet wat ze betekenen en hoe ze met elkaar kunnen samenwerken.

1.2 Vereiste voorkennis

- De cursus C#.NET Basiscursus
- De cursus C#.NET WPF
- Basiskennis van databases
- SQL (Structured Query Language)

1.3 Nodige software

- Visual Studio 2013

1.4 Situering ADO.NET

ADO.NET is het onderdeel van het .NET framework van Microsoft dat databasetoegang verschaft.

‘.NET moet het voor de wereld mogelijk maken om eenvoudige applicaties te ontwikkelen, die vanaf iedere plaats, op ieder moment te benaderen zijn en zonder problemen met elkaar kunnen communiceren’.

Bijna alle applicaties gebruiken op één of andere wijze gegevens uit externe databronnen. Het ontwerpen en onderhouden van databases is een belangrijke taak in alle grote ondernemingen, overheidsinstanties, organisaties, maar ook in de meeste kleinere bedrijven. Omvangrijke gegevensbronnen zoals klantgegevens, productievoorraden, saldorekeningen, werknemergegevens, donorlijsten, besteloverzichten, ... zijn essentieel in bedrijven en organisaties.

De gegevens kunnen van verschillende oorsprong zijn, tekstbestanden, binaire bestanden, e-mail, XML (een bestandsindeling bedoeld voor de uitwisseling van gestructureerde gegevens), spreadsheets, ... maar meestal zijn ze opgeslagen in een database. Je kunt krachtige databases ontwerpen met uiteenlopende databaseproducten zoals Microsoft Access, SqlServer, Oracle, ...

Ook moet de communicatie met gegevens over verschillende platformen kunnen gebeuren. Zo kan het dat de gegevens zich op een mainframe bevinden die onder Unix draait, en dat je deze gegevens benadert met een workstation met Windows als besturingssysteem. Bovendien speelt de communicatie met gegevens op het internet een belangrijke rol (het ultieme gedistribueerde model).

Microsoft heeft in het verleden verschillende technologieën op de markt gebracht om gegevens op te slaan en te gebruiken, en het is vaak moeilijk om hierin een keuze te maken: DAO, RDO, ODBC, OLE DB en ADO. Factoren waarmee je rekening moet houden zijn: kwaliteit, betrouwbaarheid, performantie, de frequentie waarmee je het systeem moet aanpassen om problemen op te lossen en de performantie te verbeteren, de levensduur van het systeem...

ADO.NET is ontworpen om de kwaliteiten van de vroegere technologieën te combineren met noodzakelijke aanpassingen aan nieuwe behoeften ten gevolge van de evolutie in de applicatieontwikkeling en de ontwikkeling van het .NET framework.

1.5 SqlServer en SqlExpress

In deze cursus werk je voorbeelden uit met SqlServer.

Als je een SqlServer op je computer (of op een computer van je netwerk) hebt, kan je die gebruiken om de cursus te volgen.

Als je geen SqlServer hebt, kan je SqlServer Express (SqlExpress) gebruiken. Dit is een gratis versie van SqlServer van Microsoft. Je kunt ook SqlServer én SqlExpress op dezelfde computer draaien.

(<http://www.microsoft.com/en-us/download/details.aspx?id=29062>)

SqlExpress biedt dezelfde functionaliteit als SqlServer, met enkele beperkingen die voor deze cursus niet van belang zijn.

Dit zijn de belangrijkste beperkingen van SqlExpress:

- Één database is op SqlExpress maximaal 4 gigabytes groot.
- SqlExpress gebruikt maximaal 1 GB Ram (ook als er meer RAM is).
- SqlExpress gebruikt maximaal 1 processor (ook als er meerdere processors zijn).

Vanaf Visual Studio 2012 is er een nieuwe versie van SqlExpress, nl LocalDb (aanspreeknaam: "(localdb)\V11.0"), die automatisch meegeïnstalleerd wordt met Visual Studio.

Dit is een versie voor ontwikkelaars, waarmee je SqlServer-databases kunt openen en gebruiken. In de cursus gebruiken we SqlServer, SqlExpress of LocalDb.

1.6 SQL Server Management Studio

Om gemakkelijk te werken met de SQL Server is het praktisch om de SQL Server Management Studio te installeren. Die is geheel gratis te vinden op www.microsoft.com/downloads

(Let op: iedere versie heeft zijn eigen Management Studio, ook SQL Express)

1.7 De voorbeelddatabase Bieren

In deze cursus gebruik je de database Bieren, die gegevens over bieren bevat.

De database Bieren bevat 3 tables:

- Brouwers Gegevens over de brouwers.
- Soorten Gegevens over de biersoorten, (alcoholvrij, alcoholarm, ...)
- Bieren Gegevens over de bieren zelf.

1.7.1 Structuur van de database Bieren

De table Brouwers bevat volgende velden:

Veldnaam	Type	Opmerkingen
BrouwerNr	Int	Primary Key én AutoNumber
BrNaam	Nvarchar	Geïndexeerd met toelating van dubbels
Adres	Nvarchar	
PostCode	SmallInt	
Gemeente	Nvarchar	
Omzet	Int	Betekenis: bedrag per jaar

Enkele voorbeeldrecords van deze table:

BrouwerNr	BrNaam	Adres	PostCode	Gemeente	Omzet
1	Achouffe	Route du Village 32	6666	Achouffe-Wibrin	10000
2	Alken	Stationstraat 2	3570	Alken	950000
3	Ambly	Rue Principale 45	6953	Ambly-Nassogne	500
4	Anker	Guido Gezellelaan 49	2800	Mechelen	3000
6	Artois	Vaartstraat 94	3000	Leuven	4000000

De table Soorten bevat volgende velden

Veldnaam	Type	Opmerkingen
SoortNr	Int	Primary Key én AutoNumber
Soort	Nvarchar	Geïndexeerd met toelating van dubbels

Enkele voorbeeldrecords van deze table:

SoortNr	Soort
2	Alcoholarm
3	Alcoholvrij
4	Ale
5	Alt
6	Amber

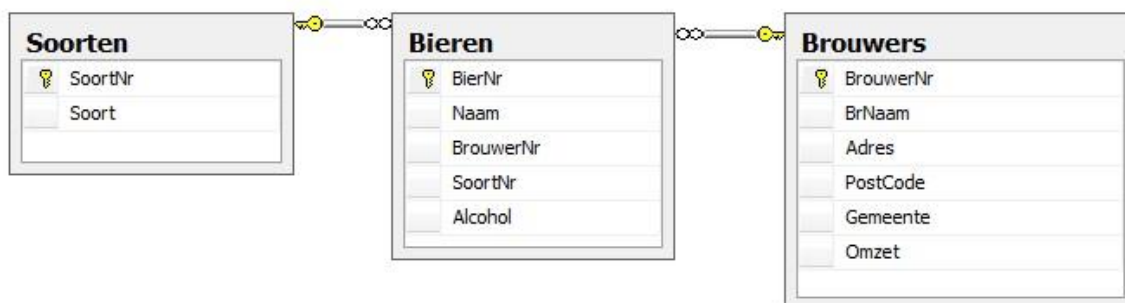
De table Bieren bevat volgende velden:

Veldnaam	Type	Opmerkingen
BierNr	Int	Primary Key én AutoNumber
Naam	Nvarchar	Geïndexeerd met toelating van dubbels
BrouwerNr	Int	
SoortNr	Int	
Alcohol	Real	

Enkele voorbeeldrecords van deze table:

BierNr	Naam	BrouwerNr	SoortNr	Alcohol
4	A.C.O.	104	18	7,00
5	Aalbeeks St. Corneliusbier (=Kapittel pater (Het))	113	18	6,50
7	Aardbeien witbier	56	53	2,50

Het relatiediagram:




Het oefenmateriaal bevat een script (*createBieren.sql*) dat deze database aanmaakt.

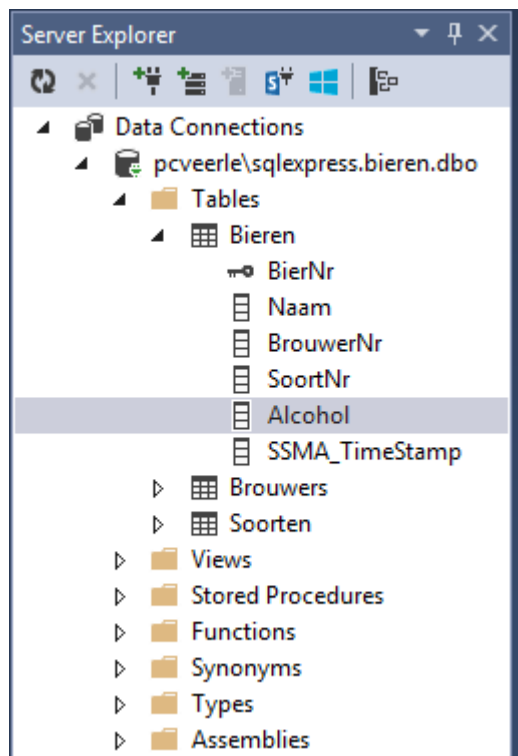
Je voert dit script uit via de tool SQL Server Management Studio (Express):

- Je start deze tool.
- Je tikt in het venster *Connect to Server .\sqlexpress*, als je SQL Express gebruikt, of je kiest je volwaardige SQL Server uit de lijst.
- Je laat de keuze bij *Authentication* op *Windows Authentication*.
- Je kiest *Connect*.
- Je kiest in het menu *File* de opdracht *Open* en de vervolgoopdracht *File*.
- Je selecteert *CreateBieren.sql* en je kiest *Open*.
- Je kiest in de knoppenbalk de opdracht *Execute*.
- Je klikt in het linkerdeel met de rechtermuisknop op de map *Databases* en je kiest de opdracht *Refresh*.
- Je klapt de map *Databases* open en je ziet de database *Bieren*.

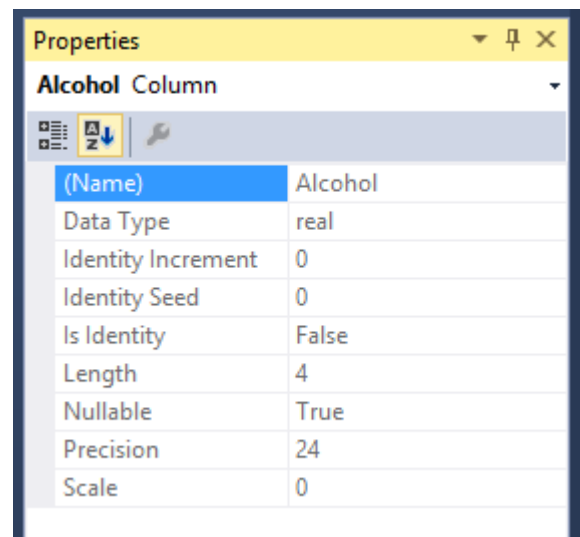
De database kan je inzien met Visual Studio met de Server Explorer:

- Klik op de knop *Connect to Database* () in de Server Explorer.
- Kies bij *Data source* naar welk type database je een verbinding maakt. Voor SqlServer of SqlExpress moet daar *Microsoft SQL Server (SqlClient)* staan. Als dit niet zo is kan je met de knop *Change...* voor *Microsoft SQL Server* kiezen.
- Kies bij *Data provider* *.NET Framework Data Provider for SQL Server*.
- Kies bij *Server Name* voor *(local)* als je de SqlServer bedoelt die op jouw eigen computer draait. Of tik *.\sqlexpress* als je de SqlExpress bedoelt die op jouw eigen computer draait.
- Als je kiest voor *Use Windows Authentication* log je op de database in met dezelfde gebruiker als waarop je momenteel op de computer bent ingelogd.
- Als je kiest voor *Use SQL Authentication*, tik je bij *User name* een gebruikersnaam die op SqlServer (SqlExpress) zelf is gedefinieerd. Bij *Password* tik je het bijbehorende paswoord.
- Kies *Use Windows Authentication*.
Opmerking: SqlServer en SqlExpress ondersteunen Windows Authentication en SQL Authentication. Standaard is bij SqlExpress enkel Windows Authentication actief.
Deze manier van inloggen kan mislukken als je op je computer niet genoeg rechten hebt. Eenvoudigste oplossing is dan dat je jezelf lid maakt van de groep Administrators via het Control Panel (onderdeel User Accounts).
- Kies bij *Select or enter a database name* de database die je wilt inzien: Klik op *OK*.

Visual Studio voegt aan de Server Explorer bij de Data Connections een vertakking toe, die de database Bieren voorstelt:



Als je een kolom van een table aanklikt (bijv. de kolom Alcohol van de table Bieren), zie je in het Properties venster meer informatie over deze kolom:



1.8 De voorbeelddatabase Bank

In deze cursus gebruik je ook de database Bank.

De database Bank bevat 2 tables:

- Klanten Gegevens over de klanten van de bank.
- Rekeningen Gegevens over de rekeningen van de klanten.

1.8.1 Structuur van de database Bank

De table Klanten bevat volgende velden:

Veldnaam	Type	Opmerkingen
KlantNr	Int	Primary Key én AutoNumber
Naam	Nvarchar	Geïndexeerd met toelating van dubbels

De records van deze table:

KlantNr	Naam
1	Asterix
2	Obelix

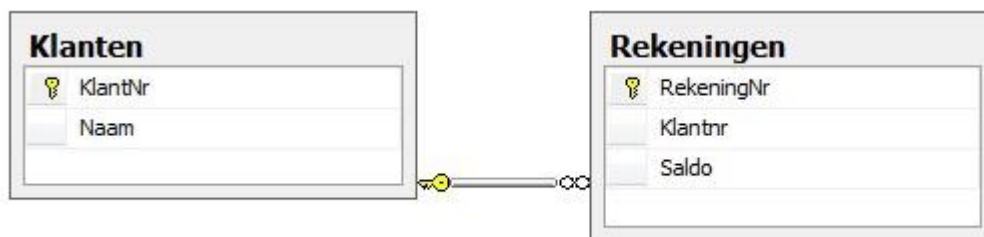
De table Rekeningen bevat volgende velden:

Veldnaam	Type	Opmerkingen
RekeningNr	Nvarchar	Primary Key
KlantNr	Int	
Saldo	Money	

De records van deze table:

RekeningNr	Klantnr	Saldo
111-1111111-70	1	1000
222-2222222-43	1	2000
333-3333333-16	2	3000
444-4444444-86	2	4000

Het relatiediagram:



Voor het creëren van deze database in Sql Server, vind je een stored procedure createBank.sql in de oefenmap.



Maak uit de oefenmap opgave 1

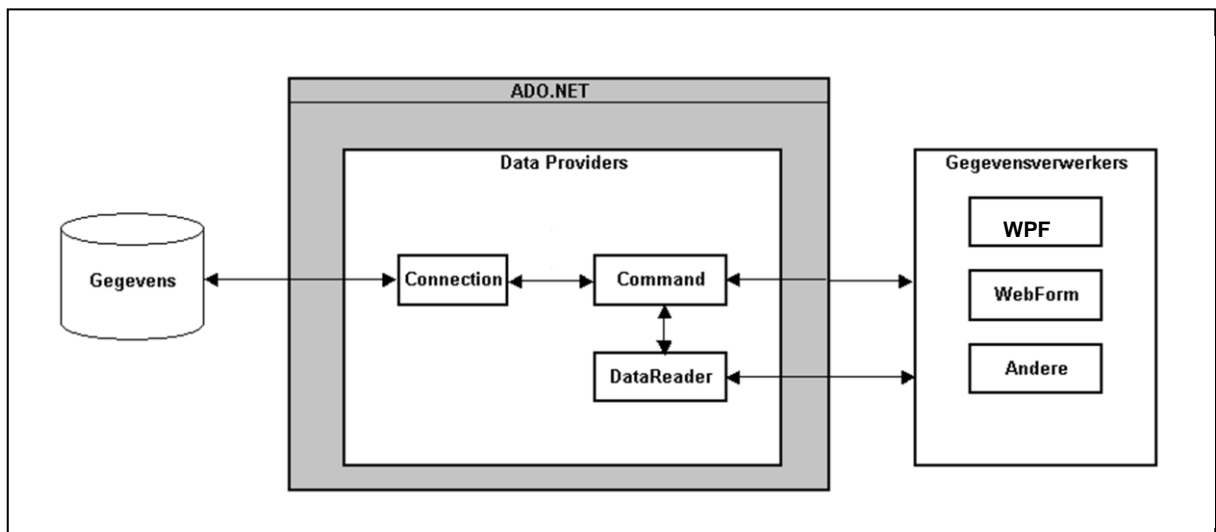
2 HET ADO.NET OBJECTMODEL

2.1 Inleiding

Hier volgt een overzicht met de voornaamste klassen van het ADO.NET-objectmodel. Verder in de cursus gaan we dieper in op de verschillende klassen.

Het ADO.NET-objectmodel bevat .NET Data Providers

Deze objecten communiceren rechtstreeks met de database: ze beheren de databaseverbinding en de transacties, ze selecteren de nodige gegevens uit de database en voeren wijzigingen door naar de database.



2.2 De .NET Data Providers

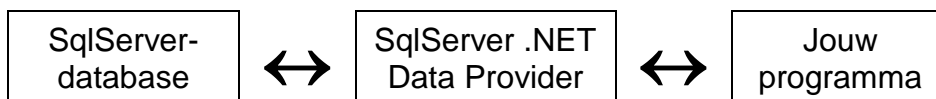
Een .NET Data Provider is een verzameling classes, die je toelaat te communiceren met een type database, meer bepaald om de verbinding met de database te maken, om gegevens uit de database te halen en om bewerkingen op de gegevens uit te voeren.

Er zijn meerdere .NET Data Providers beschikbaar.

2.2.1 SqlServer.NET Data Provider

Je gebruikt deze data provider om SqlServer (7.0 of hoger) te benaderen .

Deze provider biedt een hoge performantie, daar hij de SqlServer direct benadert zonder een omweg te maken via bijvoorbeeld OLE DB software. Vandaar dat deze provider de voorkeur verdient boven een OleDb Provider (zie verder).



De classes van de SqlServer .NET Data Provider bevinden zich in de namespace System.Data.SqlClient.

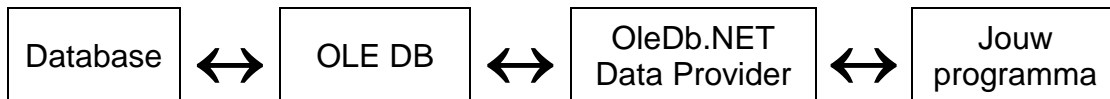
Microsoft levert deze provider mee met het .NET framework.

2.2.2 OleDb.NET Data Provider

Je gebruikt deze data provider om databases te openen waarvoor een OLE DB driver bestaat: Microsoft Access, SqlServer 6.5, DB2, ...

Je zou deze provider ook kunnen gebruiken voor SqlServer 7.0 of hoger, maar je verliest dan het voordeel van de hoge performantie, die de SqlServer.NET Data Provider aanbiedt.

De OleDb.NET Data Provider gebruikt intern oudere OLE DB software om met de database te communiceren:

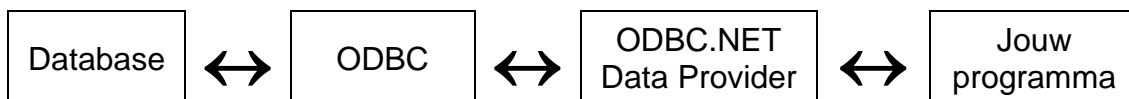


De classes van de OleDb.NET Data Provider bevinden zich in de namespace System.Data.OleDb. Microsoft levert deze provider mee met het .NET framework.

2.2.3 ODBC (Open Database Connectivity) .NET Data Provider

Je gebruikt deze provider voor databases waarvoor geen OleDb driver bestaat (oudere databases), dus voor databases die enkel via een ODBC software te benaderen zijn.

De ODBC.NET Data Provider gebruikt intern oudere ODBC software om met de database te communiceren:



De classes van de ODBC.NET Data Provider bevinden zich in de namespace System.Data.Odbc. Microsoft levert deze provider mee met het .NET framework.

2.2.4 Microsoft.NET Data Provider for Oracle

Deze provider voorziet rechtstreekse toegang tot een Oracle database, zonder gebruik te maken van een OleDb provider, wat een winst aan performantie en een aantal bijkomende voordelen oplevert. Daarom verdient deze provider de voorkeur boven de OleDb provider voor Oracle.



De classes van de Microsoft .NET Data Provider for Oracle bevinden zich in de namespace System.Data.OracleClient. Microsoft levert deze provider mee met het .NET framework.

2.2.5 Andere Data Providers

Ook andere firma's ontwikkelen Data Providers. Voor de open source database MySql bestaan Data Providers die specifiek voor MySql geschreven zijn zodat je op performante manier vanuit .NET met MySql kan samenwerken. IBM heeft een Data Provider voor zijn DB2 databasefamilie, ...

2.2.6 Opbouw .NET Data Provider

Elke .NET Data Provider bevat dezelfde klassen (Connection, DataReader, ...), die per provider een andere naam en sommige andere eigenschappen en methoden, hebben. Daar ze echter van een gemeenschappelijke base class erven, bieden de overeenkomstige klassen van de verschillende providers dezelfde functionaliteiten. Dit heeft als gevolg dat de code die we schrijven vaak weinig zal verschillen, of we nu de ene of de andere data provider gebruiken.

In deze cursus gebruiken we de SqlServer.NET Data Provider (voor de database Bieren van SqlServer of SqlExpress).

Elke .NET Data Provider bevat drie belangrijke classes:

- Connection
- Command
- DataReader

2.2.7 Connection

Een Connection-object stelt de verbinding met de database voor. Via de properties kan je o.a. de naam en de plaats van de database, gebruikersnaam en paswoord aangeven. Je gebruikt het Connection-object om de verbinding met de database te openen en te sluiten. Command-objecten communiceren via het Connection-object met de database.

2.2.8 Command

Je gebruikt Command-objecten om:

- SQL-statements als Strings van je programma naar de database te sturen en daar uit te voeren.
Met een Command-object kan je verschillende soorten SQL-statements uitvoeren: select, update, insert, delete, create table, ...
- Stored procedures van de database uit te voeren.
Stored procedures zijn SQL-statements die in de database zijn opgeslagen (in Access-termen spreekt men over queries).

In Access kan een Stored Procedure slechts één SQL-statement bevatten (select, insert, delete, update, ...). In SqlServer (en andere grote databasesystemen) kan een Stored Procedure meerdere SQL-statements, maar ook variabelen en programmeerstructuren (if then, ...) bevatten.

Daarna kan je deze Stored Procedure vanuit je applicatie oproepen. Daar de database Stored Procedures compileert bij het opslaan van de Stored Procedure, zijn ze sneller dan SQL-statements die je vanuit je programma naar de database stuurt. Je kunt een Stored Procedure beschouwen als een procedure (die ook parameters kan bevatten).

2.2.9 DataReader

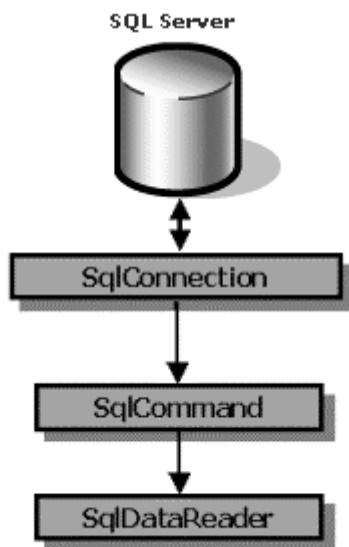
Met een DataReader-object lees je gegevens uit de database.

Met de DataReader lees je de gegevens forward-only en read-only:

- Forward-only
Je kunt enkel sequentieel record per record vooruitlezen.
- Read-only
Je kunt de gegevens die je leest niet wijzigen.

Een DataReader biedt snelle toegang tot gegevens die je maar één keer moet doorlopen om ze bijvoorbeeld in een webapplicatie door te sturen naar een browser, of om ze naar de printer te sturen.

Gezien er slechts één record tegelijk in het interne geheugen aanwezig is, gebruikt een DataReader dit geheugen economisch.



2.3 Namespaces

De basisclasses van ADO.NET vind je terug in de namespace System.Data.

De .NET Data Providers hebben elk hun eigen namespace.

De Data Providers hebben dus elk een eigen verzameling van classes om met een specifiek databasetype te communiceren.

Er is bijvoorbeeld geen algemene Connection class. Bij de SqlServer Data Provider gebruik je de SqlConnection class, bij de OleDb Data Provider gebruik je de OleDbConnection class. Alle Connection classes erven echter van dezelfde base class: DbConnection. Zij beschikken dus voor een groot deel over dezelfde functionaliteit, maar kunnen elk op zich uitgebreid zijn met providerspecifieke functionaliteiten. Dit geldt ook voor de andere classes van de DataProviders: de Command class, de DataReader class, ...

De volgende tabel geeft een overzicht van ADO.NET namespaces:

Namespace	Beschrijving
System.Data	Basisklassen van ADO.NET architectuur, die instaan voor het bijhouden van de gegevens: DataTable, DataColumn, DataRow, DataRelation, ...
System.Data.Common	Klassen die alle Data Providers delen.
System.Data.SqlClient	Specifieke klassen van de SqlServer Data Provider: SqlConnection, SqlCommand, ...
System.Data.OleDb	Specifieke klassen van de OleDb Data Provider: OleDbConnection, OleDbCommand, ...
System.Data.Odbc	Specifieke klassen van de Odbc Data Provider: OdbcConnection, OdbcCommand, ...
System.Data.OracleClient	Specifieke klassen van de Oracle Data Provider: OracleConnection, OracleCommand, ...

3 JOUW SOLUTION

In deze cursus zal je een solution maken met daarin een WPF-applicatie. Je zal werken met een SqlServer-database.

Maak een WPF Application-project in een Solution:

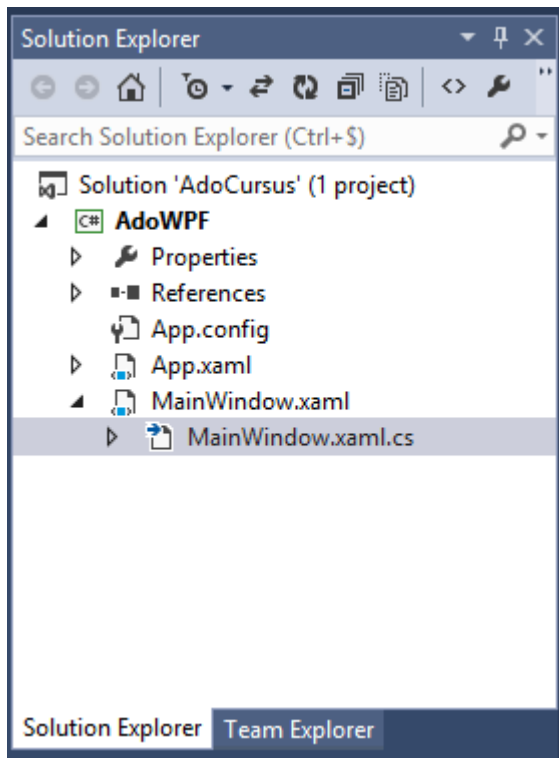
- Kies in het menu *File* de opdracht *New* en daarna de opdracht *Project*.
- Kies bij *Templates* voor *Visual C#*, *WPF Application*.
- Tik bij *Name* de naam van het project: *AdoWPF*
- Tik bij *Solution Name* de naam van de solution: *AdoCursus*
- Kies bij *Location* de directory waar Visual Studio de solution aanmaakt.
- Kies *OK*.

Importeer in het Window van dit project de namespace voor de SqlServer Data Provider:

- In het code venster van *MainWindow.xaml.cs* voeg je bovenaan na de laatste regel die begint met *using*

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

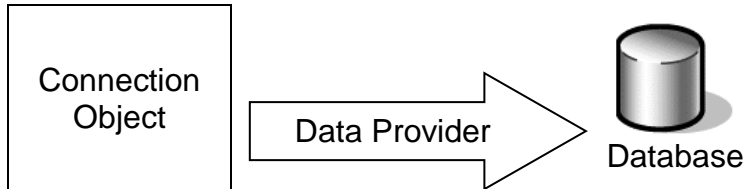
In de *Solution Explorer* zie je het overzicht van de Solution:



4 DE CONNECTION CLASS

4.1 Algemeen

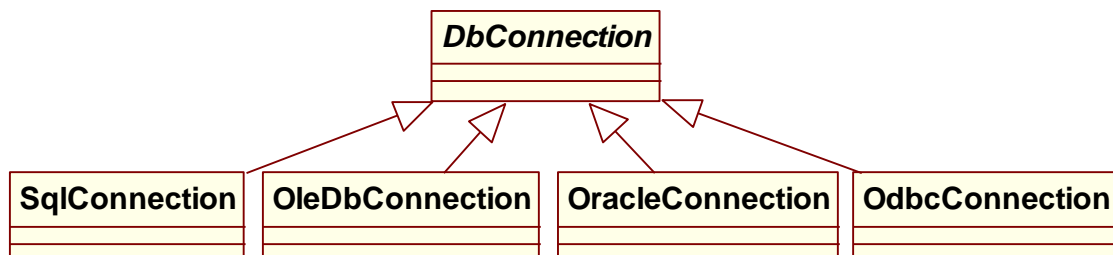
Je gebruikt de class Connection om een verbinding van je programma naar een database op te zetten. Zonder deze class kan je niets met een database doen.



Iedere .NET Data Provider heeft zijn eigen Connection class:

DataProvider	Class
SqlServer	SqlConnection
OleDb	OleDbConnection
Oracle	OracleConnection
Odbc	OdbcConnection

Al deze classes zijn gelijkaardig, want ze erven alle van een gemeenschappelijke class DbConnection:



Belangrijkste properties:

- **ConnectionString**
Deze string bevat informatie over de te openen database. Bij SqlServer vermeld je bijv. de naam van de pc waarop de SqlServer draait, de naam van de database, eventueel gebruikersnaam en paswoord.
Voorbeelden van ConnectionStrings vind je op www.connectionstrings.com.
- **State**
In deze enum-property kan je zien in welke status de connectie zich bevindt (bijv. Open of Close).

Belangrijkste methods

- **Open()**
Hiermee open je de databaseverbinding. Daarna kan je via deze verbinding handelingen op de database uitvoeren.
- **Close()**
Hiermee sluit je de databaseverbinding. Daarna kan je met deze verbinding geen handelingen meer op de database uitvoeren (tenzij je de verbinding terug

opent met de method Open). Het is belangrijk de verbinding expliciet te sluiten als je ze niet meer nodig hebt.

4.2 Een SqlServer-database openen.

Je kunt een SqlServer-database op twee manieren openen.

- met Windows Authentication.
Dan geef je in de ConnectionString-property van je SqlConnection-object geen gebruikersnaam en paswoord mee, wat dus zeer veilig is.
- met SqlServer Authentication.
Dan geef je in de ConnectionString-property van je SqlConnection-object een gebruikersnaam (en meestal paswoord) mee.
Het is met die gebruikersnaam dat je inlogt op SqlServer.

4.2.1 Een SqlServer-database openen met Windows Authentication

Als je de SqlServer opent met Windows Authentication, geef je in de ConnectionString-property geen gebruikersnaam mee om op de SqlServer in te loggen.

In een Windows-applicatie log je automatisch in met de gebruiker waarmee je op de computer aangelogd bent. Als dit niet lukt, heb je misschien niet genoeg rechten op je computer. Eenvoudigste oplossing is dan jezelf lid te maken van de groep Administrators via het Control Panel.

Plaats op de Form een Button met volgende properties:

- Name buttonBieren
De naam van een control op een WPF-Window vul je in bij de properties of in de Xaml code.
- Content Bieren openen

Plaats daaronder een Label met volgende properties:

- Name labelStatus
- Content leeg maken

Sleep de controls op het WPF-Window, waardoor ze in een Grid geplaatst worden. Dit levert de onderstaand XAML code op. (voor andere layouts kan je de cursus van WPF raadplegen)

```
<Window x:Class="AdoWPF.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:AdoWPF"
        mc:Ignorable="d"
        Title="MainWindow" Height="350" Width="525">
  <Grid>
    <Button x:Name="buttonBieren" Content="Bieren openen"
            HorizontalAlignment="Left" VerticalAlignment="Top"
            Width="100" Margin="214,59,0,0" Click="buttonBieren_Click"/>
    <Label x:Name="labelStatus" Content=""
           HorizontalAlignment="Left" VerticalAlignment="Top"
           Margin="230,154,0,0"/>
  </Grid>
</Window>
```

Dubbeltklik de Button naast de tekst. Visual Studio maakt een routine en koppelt die aan het Click-event van deze Button. Schrijf volgende code in deze routine:

```
try
{
    using (var conBieren = new SqlConnection())           (1)
    {
        conBieren.ConnectionString =
            @"server=.\sqlexpress;database=Bieren;integrated security=true"; (2)
        conBieren.Open();                               (3)
        labelStatus.Content = "Bieren geopend";          (4)
    }
}
catch (Exception ex)                                   (5)
{
    labelStatus.Content = ex.Message;                   (6)
}
```

(1) De SqlConnection-object staat in een *Using* blok. Op het eind van dit *Using* blok sluit .NET automatisch de bijhorende database-connectie

De variabele conBieren wordt geassocieerd met dit object.

(2) De SqlConnection-ConnectionString bevat volgende onderdelen:

a. server=NaamPc of server=.\sqlexpress

Als je de verbinding maakt naar een echte SqlServer gebruik je de eerste versie (server=NaamPc) en vervang je NaamPc door de naam van de computer waarop de SqlServer draait.

Als je de verbinding maakt naar SqlExpress op je computer, gebruik je de tweede versie (server=.\sqlexpress). In de voorbeeldcode maak je een verbinding naar een SqlExpress op de eigen computer.

Opmerking 1: als je de eerste versie gebruikt (verbinding met een echte SqlServer), en de SqlServer draait op dezelfde computer als jouw applicatie, mag je ook schrijven: server=(local)

Opmerking 2: in de plaats van het woord server, mag je ook de woorden data source gebruiken: data source=NaamPc of data source=.\sqlexpress

b. database=bieren

Na database= vermeld je de naam van de database waarop je wilt inloggen (bieren).

Opmerking: in de plaats van het woord database mag je ook de woorden initial catalog gebruiken: initial catalog=bieren

c. integrated security=true

Hiermee bekom je Windows Authentication.

(3) Je opent de database.

(4) Je toont in het Label dat de database nu open is.

(5) Als .NET de database niet kan openen (bijv. een tikfout in de ConnectionString-property), krijg je een fout toegeworpen.

- (6) Je toont de technische omschrijving van de fout.
Opmerking: in een echte applicatie toon je aan de eindgebruiker geen technische omschrijving van de fout, maar een gebruiksvriendelijke foutomschrijving, zoals *Probleem met database, contacteer helpdesk*

Je kunt nu het programma starten en proberen de database te openen.

Opmerking: je kunt ConnectionString-property ook meegeven als een parameter van de SqlConnection constructor:

```
try {
    using (var conBieren = new SqlConnection(
        @"server=.\sqlexpress;database=Bieren;integrated security=true"))
    {
        conBieren.Open();
        labelStatus.Content = "Bieren geopend";
    }
}
catch (Exception ex) {
    labelStatus.Content = ex.Message;
}
```

Test de code uit.

4.2.2 Een SqlServer-database openen met SqlServer Authentication

Bij de default installatie van SqlExpress is SqlServer Authentication niet actief.

Als je in de cursus SqlExpress gebruikt, zal je dit eerst wijzigen.

- Eerst moet je een grafische tool voor SqlExpress downloaden.
De tool heet *SQL Server Express Management Studio*.
Een hyperlink naar de download vind je via de pagina
<http://www.microsoft.com/sql/editions/express>
- Na de download installeer je deze grafische tool.
- Daarna start je deze grafische tool *Microsoft SQL Server 2014*.
- Eerst zie je een aanlogvenster (*Connect to Server*).
- Tik bij *Server name* de verwijzing naar je SqlExpress: *.\sqlexpress*
- Kies bij *Authentication* voor *Windows Authentication*.
- Kies de knop *Connect*.
- Links onder *Object Explorer* zie je *.\sqlexpress*
- Klik hierop met de rechtermuisknop en kies *Properties*.
- Kies links onder *Select a page* voor *Security*.
- Kies onder *Server authentication* voor
SQL Server and Windows Authentication mode.
- Kies de knop *OK*.
- Nu moet je SqlExpress herstarten.
- Klik onder *Object Explorer* met de rechtermuisknop op *.\sqlexpress* en kies de opdracht *Restart*.
- Klik Yes bij de vraag *Are you sure you want to restart ...*

De gebruiker sa is nog disabled, dus zijn status moet gewijzigd worden in enabled.

- Kies links het onderdeel *Security* van *.\sqlexpress* en daarbinnen *Logins*
- Dubbelklik de gebruiker *sa*.

- Kies in het venster dat naar voor komt links voor *Status*.
- Kies rechts bij *Login* voor *Enabled*.
- Kies de knop *OK*.

Geef de gebruiker sa een paswoord:

- Kies links het onderdeel *Security* van *.\sqlexpress* en daarbinnen *Logins*
- Dubbelklik de gebruiker *sa*.
- Kies in het venster dat naar voor komt links voor *General*.
- Tik bij *Password* het paswoord: *Vdab123*
Opmerking: een paswoord dat enkel letters bevat is niet sterk genoeg.
- Herhaal dit paswoord bij *Confirm password*.
- Kies de knop *OK*.

Als je een *SqlServer* opent met *SqlServer Authentication*, geef je in de *ConnectionString*-property een *UserID* (gebruikersnaam) en paswoord mee om op de *SqlServer* in te loggen.

Wijzig de code van ButtonBieren:

```
try
{
    using (var conBieren =
        new SqlConnection(@"server=.\sqlexpress;database=Bieren;
            user id=sa;password=Vdab123"))
        (1)
    {
        conBieren.Open();
        labelStatus.Text = "Bieren geopend";
    }
}
catch (Exception ex)
{
    labelStatus.Text = ex.Message;
}
```

- (1) De onderdelen `server` en `database` zijn dezelfde als het vorige voorbeeld. Na `user id=` schrijf je de naam van een gebruiker die op `SqlServer` gekend is en waarmee je inlogt. In het voorbeeld log je in met de gebruikersnaam `sa`, die standaard bestaat in `SqlServer` en alle rechten heeft (hij is de administrator). Het paswoord van de gebruiker vermeld je na `password=`

Test de code uit.

4.3 De `ConnectionString` uit een configuratiebestand halen

Je kunt de connection string ook opnemen als een string in een configuratiebestand met XML-indeling. Vanuit je programma haal je tijdens de uitvoering deze string op uit het configuratiebestand en je maakt er de `ConnectionString`-property mee. Ook op deze manier moet je het programma niet wijzigen als er iets verandert met de connectie. Je hoeft enkel het XML-bestand te wijzigen, wat je kunt doen met elke teksteditor, ook met het programma `NotePad`.

Een WPF-applicatie bevat al een XML-configuratiebestand: *app.config*.

Je ziet een XML-bestand met de basisstructuur voor een applicatieconfiguratiebestand:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
</configuration>
```

Als er iets staat tussen `<configuration>` en `</configuration>` mag je dat verwijderen.

Wijzig dit bestand als volgt (pas op, XML is hoofdlettergevoelig!):

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <connectionStrings>
        <add name="Bieren" connectionString=
            "server=.\sqlexpress;database=bieren;integrated security=true"/>
    </connectionStrings>
</configuration>
```

Je tikt je `connectionStrings` tussen `<connectionStrings>` en `</connectionStrings>`.

Er wordt één connectionString toegevoegd met `<add ... />`

Met `name="Bieren"`, geef je de connectionString de naam *Bieren*. In de code zal je de connectionString ophalen door te verwijzen naar die naam.

De connectionString zelf tik je in het attribuut *connectionString*.

Nu pas je de code aan van `MainWindow.xaml.cs` in het project *AdoWPF*.

Het .NET framework bevat een class *ConfigurationManager* die je helpt een connectionString uit `app.config` te halen.

Deze class bevindt zich in de DLL *System.Configuration*. Je applicatie bevat nog geen verwijzing naar deze DLL en kan daarom de DLL niet gebruiken.

Voeg een verwijzing toe naar deze DLL:

- Klik met de rechtermuisknop op *AdoWPF* in de *Solution Explorer*.
- Kies *Add Reference*.
- Kies het onderdeel *Framework* onder *Assemblies*.
- Vink *System.Configuration* aan in het middelste gedeelte onder *Name*.
- Plaats in de form-code onder alle `using` regels bovenaan de code een extra regel met
`using System.Configuration;`
om de namespace *System.Configuration* te importeren.

Je haalt de *ConnectionString* op uit de instelling met sleutelwaarde *Bieren* uit het configuratiebestand:

```
using (var conBieren = new SqlConnection())
{
    ConnectionStringSettings conString =
        ConfigurationManager.ConnectionStrings["Bieren"];           (1)
    conBieren.ConnectionString = conString.ConnectionString;          (2)
    conBieren.Open();
    labelStatus.Content = "Bieren geopend";
}
```

- (1) De shared property *ConnectionStrings* van de class *ConfigurationManager* bevat alle connectionString's uit `app.config`.
Je spreekt één van deze connectionString's aan via het *name* attribuut dat je de connectionString gaf in `app.config`. Je krijgt deze ene connectionString als een object van het type *ConnectionStringSetting*.
- (2) De property *ConnectionString* van een *ConnectionStringSetting*-object geeft je de inhoud van het attribuut *connectionString*.

4.3.1 App.xaml

Er bestaat nog een andere methode.

Een WPF Application bevat al een App.xaml en een bijhorend App.xaml.cs-bestand.

Open de bijhorende code en vervolledig die tot:

```
namespace AdoWPF
{
    /// <summary>
    /// Interaction logic for App.xaml
    /// </summary>
    public partial class App : Application
    {
        public App()
        {
            string connectionString =
                @"server=.\sqlexpress;database=Bieren;integrated security=true";
            Application.Current.Properties["Bieren2"] = connectionString;
        }
    }
}
```

In MainWindow halen we die property terug op:

```
using (var conBieren = new SqlConnection())
{
    conBieren.ConnectionString =
        Application.Current.Properties["Bieren2"].ToString();
    conBieren.Open();
    labelStatus.Content = "Bieren geopend";
}
```

4.4 De programmacode neutraal maken t.o.v. het type database

Zoals de code nu werkt, is ze expliciet gebonden aan een SqlServer-database, want dan programmeer je op een SqlConnection-object. Als je opdrachtgever beslist een ander databasetype te gebruiken, moet je het programma wijzigen en bij overgang van SqlServer naar (bijvoorbeeld) Access het type van je variabelen wijzigen van SqlConnection naar OleDbConnection.

Het is mogelijk code te schrijven die je niet moet wijzigen als je opdrachtgever het soort database wijzigt.

De classes OleDbConnection en SqlConnection (en ook bijv. OracleConnection) erven van dezelfde base class: DbConnection.

Als je voor je Connection-reference-variabele het type DbConnection gebruikt in plaats van OleDbConnection of SqlConnection, is de rest van de handelingen op deze variabele neutraal ten opzichte van de verschillende merken databases.

Je zal dus een variabele conBieren maken van het type DbConnection.

Uiteindelijk moet je deze reference-variabele verbinden met een OleDbConnection-object (om naar Access te gaan), of een SqlConnection-object (om naar SqlServer te gaan).

Gelukkig moet jij geen code schrijven om deze keuze te maken, maar krijg je hiervoor ondersteuning vanuit het .NET framework.

Als eerste stap pas je het XML configuratiebestand aan:

- Je geeft alle connectionStrings een extra attribuut *providerName*. Bij een connectionString die naar een Access database verwijst, ziet dit attribuut er als volgt uit: `providerName="System.Data.OleDb"`. Bij een connectionString die naar een SqlServer-bestand verwijst, ziet dit attribuut er als volgt uit: `providerName="System.Data.SqlClient"`.
- Je geeft alle connectionStrings hetzelfde *name* attribuut.
- Je plaatst alle connectionStrings in commentaar, behalve de connectionString naar de database die je op dat moment wil gebruiken. Je begint XML commentaar met `<!--` en sluit commentaar met `-->`.

In het voorbeeld ga je de verbinding naar de SqlServer database activeren, dus plaats je de verbinding naar de Access Database in commentaar.

De verbinding met de Access Database bestaat in ons project niet, maar dient enkel om een voorbeeld te geven.

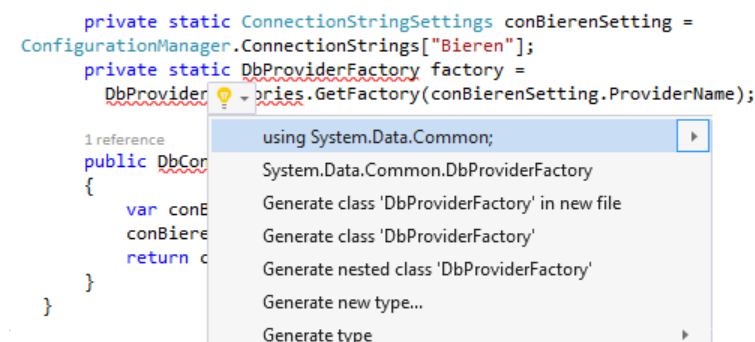
Voor *AdoWPF* doe je deze aanpassingen in *app.config*:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <connectionStrings>
    <add name="Bieren" connectionString=
      "server=.\\sqlexpress;database=bieren;integrated security=true"
      providerName="System.Data.SqlClient"/>
    <!--<add name="Bieren" connectionString=
      "provider=Microsoft.ACE.OLEDB.12.0; data source=c:\\oefen\\bieren.mdb"
      providerName="System.Data.OleDb"/>-->
  </connectionStrings>
</configuration>
```

Je importeert in het *AdoWPF*-project de namespace *System.Data.Common* (die de class *DbConnection* bevat) door bovenaan de code onder de andere *using-regels* volgende regel toe te voegen:

```
using System.Data.Common;
```

Met Visual Studio 2015 kan je eventueel je code ingeven en wachten op een foutmelding. Bij die foutmelding kan je dan klikken op het pijltje en kan je daar de reference importeren. (als die reference in je toegevoegde references zit in de Solution Explorer)



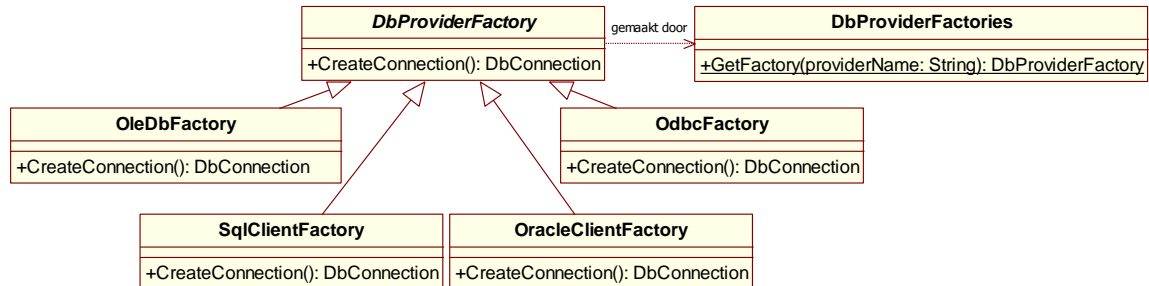
```
private static ConnectionStringSettings conBierenSetting =
  ConfigurationManager.ConnectionStrings["Bieren"];
private static DbProviderFactory factory =
  DbProviderFactories.GetFactory(conBierenSetting.ProviderName);

1 reference
public DbConnection
{
  var conBieren =
  conBierenSetting
  return conBieren
}

using System.Data.Common;
System.Data.Common.DbProviderFactory
Generate class 'DbProviderFactory' in new file
Generate class 'DbProviderFactory'
Generate nested class 'DbProviderFactory'
Generate new type...
Generate type
```

Een andere snelle manier om de using toe te voegen, is de volgende: op de nietgekende naam gaan staan en de toetsencombinatie Ctrl ; indrukken.

In de code zal je nieuwe classes gebruiken:



Eerst roep je de shared functie `GetFactory()` van de class `DbProviderFactories` op. Naargelang de string die je als parameter meegeeft aan deze functie maakt ze een ander type object. Ze geeft je dit object onder de gedaante van de base class van het teruggegeven object: de class `DbProviderFactory`.

Parameterwaarde van <code>GetFactory()</code>	Type object dat <code>GetFactory()</code> aanmaakt en teruggeeft
<code>System.Data.OleDb</code>	<code>OleDbFactory</code>
<code>System.Data.SqlClient</code>	<code>SqlClientFactory</code>
<code>System.Data.OracleClient</code>	<code>OracleClientFactory</code>
<code>System.Data.Odbc</code>	<code>OdbcFactory</code>

Geef aan deze functie de inhoud van het nieuwe attribuut `providerName` mee dat je vermeldt bij de `ConnectionString` in het XML-configuratiebestand.

Het XML-configuratiebestand bevat één `ConnectionString` die niet in commentaar staat. Het attribuut `providerName` van deze `ConnectionString` bevat de string `System.Data.SqlClient`.

Jij voedert deze string aan de functie `GetFactory()` en zal dus een `SqlClientFactory`-object terugkrijgen onder de gedaante van zijn base class: `DbProviderFactory`.

Op de returnwaarde van `GetFactory` roep je de method `CreateConnection` op. Naargelang het type object dat `GetFactory` je aangeboden heeft, krijg je een ander type `Connection` terug, maar altijd onder de gedaante van zijn base class `DbConnection`:

Returnwaarde van <code>GetFactory()</code>	Type connection dat je terugkrijgt van <code>CreateConnection</code>
<code>OleDbFactory</code>	<code>OleDbConnection</code>
<code>SqlClientFactory</code>	<code>SqlConnection</code>
<code>OracleClientFactory</code>	<code>OracleConnection</code>
<code>OdbcFactory</code>	<code>OdbcConnection</code>

Nu kan je deze connectie gebruiken, zonder je nog zorgen te maken naar welk type database ze verwijst.

De code van het Click-event van de button in *MainWindow.xaml.cs*:

```
try
{
    var conString = ConfigurationManager.ConnectionStrings["Bieren"];    (1)
    var factory = DbProviderFactories.GetFactory(conString.ProviderName);
    (2)
    using (var conBieren = factory.CreateConnection())                    (3)
    {
        conBieren.ConnectionString = conString.ConnectionString;
        conBieren.Open();
        labelStatus.Content = "Bieren geopend";
    }
}
catch (Exception ex)
{
    labelStatus.Content = ex.Message;
}
```

- (1) De shared property ConnectionStrings van de class ConfigurationManager bevat alle connectionStrings uit het XML-configuratiebestand. Je spreekt de connectionString aan die als *name* attribuut *Bieren* heeft. Je krijgt deze connectionString als een object van het type ConnectionStringSetting.
- (2) Je vraagt aan het ConnectionStringSetting-object de inhoud van het *providerName* attribuut (momenteel is dit `System.Data.SqlClient`). Je gebruikt deze string bij de aanroep van de method `GetFactory` van de class `DbProviderFactories`. Deze method zal momenteel een `SqlClientFactory`-object teruggeven onder de gedaante van `DbProviderFactory`.
- (3) Je vraagt aan de gekregen factory een Connection met de method `CreateConnection`. Momenteel is de ware aard van de factory `SqlClientFactory`, dus krijg je een `SqlConnection` terug, onder de gedaante van `DbConnection`.

Test de code uit.

Als je ooit wilt veranderen van database, hoef je het programma niet te wijzigen, enkel het XML-configuratiebestand. Je zet alle niet gebruikte connectionstrings in commentaar en laat enkel de nodige connectionString uit de commentaar.

4.5 De databaseconnectie in een gemeenschappelijke DLL.

4.5.1 Het project voor de DLL zelf

De code die de databaseconnectie maakt, bevindt zich nu in een WPF-Window.

Als je in een andere WPF-Window of WebForm eenzelfde databaseconnectie nodig hebt, is het af te raden de code naar deze andere form te kopiëren. Als de code achteraf wijzigt, moet je deze wijziging meerdere keren doen.

Als je de code in een class plaatst, kan je deze class vanuit verschillende forms oproepen.

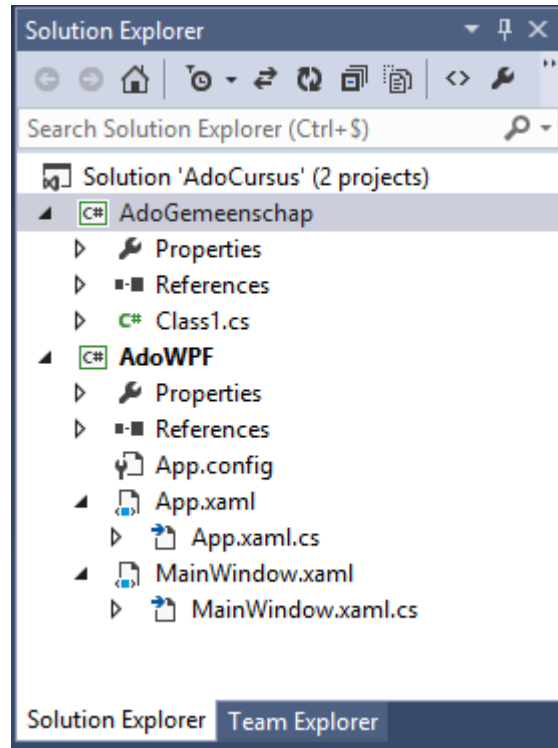
Als je de class in een project van het type Class Library maakt, kan je deze class vanuit alle types projecten oproepen (WPF, WebForm, Console, ...).

Visual Studio compileert een project van het type Class Library tot een DLL

Je voegt een project voor de DLL toe aan de solution:

- Kies in het menu *File* de opdracht *Add* en daarna de opdracht *New Project*.
- Kies bij *Templates* voor *Class Library*.
- Tik bij *Name* de naam van het project: *AdoGemeenschap* en kies *OK*.

In de *Solution Explorer* zie je dat de solution nu 2 projecten bevat:



Leg in het project *AdoGemeenschap* een referentie naar *System.Configuration.dll*:

- Klik met de rechtermuisknop op *AdoGemeenschap* in de *Solution Explorer*.
- Kies links *Add Reference* . . .
- Kies links voor Framework onder Assemblies.
- Vink onder *Name* het onderdeel *System.Configuration* aan
- Klik op de knop *OK*.

Het project bevat een class: *Class1*. Je geeft deze class een betere naam:

- Klik met de rechtermuisknop op *Class1.cs* in het project *AdoGemeenschap* in de *Solution Explorer*.
- Kies de opdracht *Rename*.
- Tik *BierenDbManager.cs* en druk Enter.
- Antwoord "Yes" op de vraag "You are renaming a file. Would you also like to perform a rename in this project of all references to the code element 'Class1'?"

Importeer in dit project enkele namespaces:

- Voeg bovenaan de code volgende regels toe
- ```
using System.Configuration;
using System.Data.Common;
using System.Data.SqlClient;
```

Wijzig deze class als volgt:

```
public class BierenDbManager
{
 private staticConnectionStringSettings conBierenSetting =
 ConfigurationManager.ConnectionStrings["Bieren"];
 private static DbProviderFactory factory =
```

(1)

```
DbProviderFactories.GetFactory(conBierenSetting.ProviderName); (2)
```

```
public DbConnection GetConnection() (3)
{
 var conBieren = factory.CreateConnection();
 conBieren.ConnectionString = conBierenSetting.ConnectionString;
 return conBieren;
}
```

- (1) De connectionString met als *name*-property *Bieren* heb je keer op keer nodig.  
In plaats van deze connectionString iedere keer op te halen, haal je hem maar één keer op (goed voor de performantie). Je doet dit door de ConnectionStringSetting-variabele als een static variabele bij te houden. Een static variabele wordt één keer geïnitieerd en behoudt vanaf dan zijn inhoud.
- (2) De factory die je de juiste connectie kan aanbieden (SqlServer) zal je keer op keer nodig hebben.  
In plaats van de factory iedere keer op te halen, haal je hem maar één keer op (goed voor de performantie). Je doet dit door de DbProviderFactory-variabele als een static variabele bij te houden. Een static variabele wordt één keer geïnitieerd en behoudt vanaf dan zijn inhoud.
- (3) Deze functie zal je oproepen in *AdoWPF*, telkens je een verbinding naar de Bieren-database nodig hebt. De functie vraagt een connectie aan de shared factory, initialiseert de connectie met de shared connectionString en geeft deze connectie terug aan de oproeper.

#### 4.5.2 De DLL aanspreken

Vooraleer je vanuit het WPF-project de class *BierenDbManager* kan aanspreken, moet je vanuit dit project een referentie leggen naar het project *AdoGemeenschap*:

- Klik in de *Solution Explorer* met de rechtermuisknop op *AdoWPF*.
- Kies de opdracht *Add Reference...*
- Kies links *Projects* onder *Solution*.
- Kies *AdoGemeenschap* in de lijst en vink dat aan.

De classes van de DLL bevinden zich in hun eigen namespace met de naam *AdoGemeenschap*. Om gebruik te maken van deze namespace moet je deze importeren in het project *AdoWPF*. Hiervoor tik je in *MainWindow.xaml.cs* bovenaan onder de rest van de *using* statements volgende regel bij:

```
using AdoGemeenschap;
```

Wijzig in *MainWindow.xaml.cs* de code van de routine *buttonBieren\_Click*:

```
try
{
 var manager = new BierenDbManager(); (1)
 using (var conBieren = manager.GetConnection()) (2)
 {
 conBieren.Open(); (3)
 labelStatus.Content = "Bieren geopend";
 }
}
catch (Exception ex)
```

```
{
 labelStatus.Content = ex.Message;
}
```

- (1) Je maakt een object van het type *BierenDbManager* uit *AdoGemeenschap*.
- (2) Je vraagt deze manager een connectie te maken naar de Access of SqlServer-database en je wijst met een referentievariabele naar dit *DbConnection*-object.
- (3) Je opent de verbinding naar de database.

Je kunt de code uitproberen.

### 4.5.3 De oefendatabase Bank

Maak voor de oefendatabase Bank instellingen bij in *app.config*

```
<connectionStrings>
 <add name="Bieren" connectionString=
 "server=.\sqlexpress;database=bieren;integrated security=true"
 providerName="System.Data.SqlClient"/>
 <add name="Bank" connectionString=
 "server=.\sqlexpress;database=bank;integrated security=true"
 providerName="System.Data.SqlClient"/>
</connectionStrings>
```

Voeg aan het project *AdoGemeenschap* een class *BankDbManager* met functies toe die connecties maken naar de Bank database. Deze class lijkt sterk op de class *BierenDbManager*.

```
...
using System.Configuration;
using System.Data.Common;

namespace AdoGemeenschap
{
 public class BankDbManager
 {
 private static ConnectionStringSettings conBankSetting =
 ConfigurationManager.ConnectionStrings["Bank"];
 private static DbProviderFactory factory =
 DbProviderFactories.GetFactory(conBankSetting.ProviderName);

 public DbConnection GetConnection()
 {
 var conBank = factory.CreateConnection();
 conBank.ConnectionString = conBankSetting.ConnectionString;
 return conBank;
 }
 }
}
```

Verder in de cursus zal je deze uitbreidingen gebruiken als je met de Bank database werkt.

## 4.6 Connection Pooling

Een connectie naar een database aanmaken en openen kan wat tijd vragen.

Vroeger werd dit probleem opgelost door bij het starten van de applicatie de connectie te openen, de connectie te gebruiken en de connectie pas te sluiten bij het afsluiten van de applicatie.

Dit is negatief voor de database, want hij moet connecties lange tijd bijhouden, ook op momenten dat de gebruiker handelingen in de applicatie doet die de database niet aanspreken.

Bij systemen met meerdere (veel) gebruikers groeit dit probleem, want je komt in situaties waar per gebruiker een connectie lang blijft openstaan. Dan heeft de database veel werk om deze connecties bij te houden, en dus minder tijd om zo snel mogelijk data aan te bieden aan de applicaties.

In .NET zal je een connectie pas openen op het moment dat de gebruiker een handeling op de database doet. Na deze handeling sluit jij in de code deze connectie.

Op het eerste zicht lijkt het dat iedere handeling nu veel tijd zal vragen, want het openen van een connectie vraagt tijd...

Achter de schermen gebeurt in .NET echter connection pooling. Dit houdt in dat als jij in de code een connectie sluit, de .NET runtime deze connectie niet echt sluit, maar in geopende toestand bijhoudt (in een 'connection pool'). Als je applicatie een tijdje later opnieuw eenzelfde connectie opent, neemt .NET de connectie uit de pool (waar ze nog open stond) en je code krijgt deze connectie.

Het is dus belangrijk de connecties expliciet terug te sluiten. Pas dan gaan ze onmiddellijk terug naar de pool, waar ze klaar zijn om opnieuw gebruikt te worden.

Een pool kan meerdere connecties bevatten. Dit is interessant in een webapplicatie, waar meerdere gebruikers tegelijk handelingen doen.

Als je in de code een connectie opent, controleert .NET of de pool nog een connectie bevat die niet actief is. Als dit niet het geval is, maakt .NET een nieuwe connectie naar de database. Als je in de code deze nieuwe connectie sluit, komt ze in de pool naast de vroegere connecties.

In de connectionstring kan je vermelden hoeveel connecties de pool maximaal kan bevatten. Bij een SqlServer (SqlExpress) connectionstring kan je met het onderdeel *Max Pool Size=99* bijvoorbeeld instellen dat de pool maximaal 99 connecties met die connectionstring kan bevatten.

Als een connectie in de pool een hele tijd niet gebruikt wordt, zal de pool de connectie uiteindelijk sluiten naar de database toe. Je kunt echter ook vermelden hoeveel connecties met dezelfde connectionstring de pool minimaal moet bevatten. Bij een SqlServer (SqlExpress) connectionstring kan je met het onderdeel *Min Pool Size=5* bijvoorbeeld instellen dat de pool minimaal 5 connecties met die connectionstring kan bevatten. De pool zal dan minstens 5 connecties met die connectionstring openhouden.



Nog enkele opmerkingen:

- De pool bevindt zich in de applicatie, niet in de database.
- Iedere .NET Data Provider heeft zijn eigen pool, die wordt aangemaakt als de eerste connectie voor die .NET Data Provider geopend wordt.
- Een connectie kan enkel uit de pool gerecupereerd worden (bij het openen van een connectie in je code) als de connectionstring van de connectie uit de pool identiek is aan de connectionstring van de connectie die je in je applicatie opent.
- De SqlServer .NET Data Provider gebruikt standaard connection pooling. De pool bevat standaard maximaal 100 en minimaal 0 connecties. In het volgende connectionstring voorbeeld worden deze waarden aangepast:

```
"server=.\sqlexpress;database=bieren;integrated security=true;
max pool size=50;min pool size=5"
```

- Of de OleDb .NET Data Provider pooling doet, hangt af van het type database waarmee deze provider verbonden is. Voor Access gebeurt standaard geen connection pooling. Je kunt connection pooling voor Access activeren door de connectionstring uit te breiden met een onderdeel *ole db services=-1*:

```
"provider=Microsoft.ACE.OLEDB.12.0; data source=c:\oefen\bieren.mdb;
ole db services=-1"
```



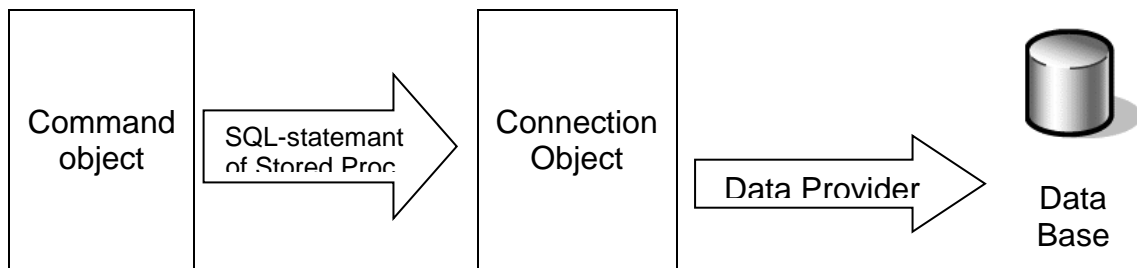
**Maak uit de oefenmap opgave 2**

## 5 DE COMMAND CLASS

### 5.1 Algemeen

Tot nu kan je enkel (maar dit is wel essentieel!) een database openen en sluiten. Eens je de database geopend hebt, is de volgende stap SQL-statements naar de database te sturen of Stored Procedures van de database op te roepen.

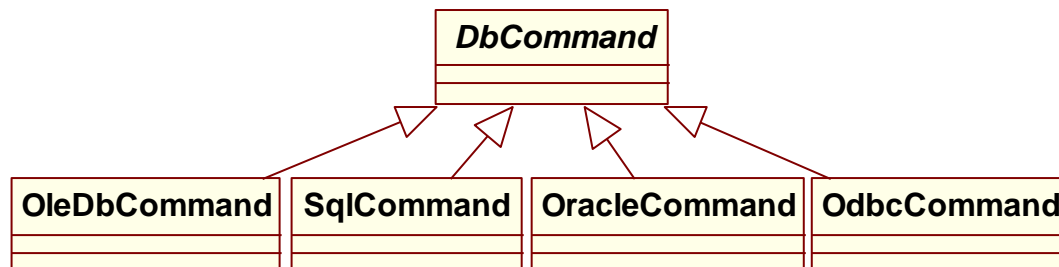
Hiervoor gebruik je objecten van de Command class.



Iedere .NET Data Provider heeft zijn eigen Command class:

DataProvider	Class
SqlServer	SqlCommand
OleDb	OleDbCommand
Oracle	OracleCommand
Odbc	OdbcCommand

Al deze classes zijn gelijkaardig: ze erven allen van de base class DbCommand:



Als je programmeert met een variabele van het type SqlCommand, kan je enkel opdrachten uitvoeren naar een SqlServer (SqlExpress) database. Als je echter programmeert met een DbCommand-variabele, kan je code schrijven die werkt op elk type database.

De DbConnection-variabele die je gekregen hebt van de DbProviderFactory, bevat een method CreateCommand. Als het object verbonden met de DbConnection-variabele een OleDbConnection is, geeft CreateCommand een OleDbCommand terug onder de gedaante van DbCommand. Als het object verbonden met de DbConnection-variabele een SqlConnection is, geeft CreateCommand een SqlCommand terug onder de gedaante van DbCommand. De method CreateCommand helpt je dus om code te schrijven die databasetype neutraal is.

## Belangrijkste properties van DbCommand:

- **Connection**  
In deze property verwijst je naar het Connection-object waarmee je de database geopend hebt. Als je deze property niet invult, weet de Command niet op welke database hij een opdracht moet uitvoeren. Als je een DbCommand-object laat aanmaken door de CreateCommand-method van een DbConnection, is de Connection-property van dat DbConnection automatisch ingevuld met het DbConnection-object waarop je CreateCommand uitvoerde.
- **CommandType**  
In deze enum vul je één van drie mogelijkheden in:
  - **CommandType.Text**  
Als je het SQL-statement als een string van je code naar de database stuurt.
  - **CommandType.StoredProcedure**  
Als je vanuit je programma een Stored Procedure oproept. Een Stored Procedure bevat één of meerdere SQL-statements en is opgeslagen in de database zelf.
  - **CommandType.TableDirect**  
Als je de data van alle rijen én alle kolommen van één table uit de database wil lezen. Deze keuze wordt niet veel gebruikt.
- **CommandText**  
In deze string vul je
  - Een SQL-statement in als de CommandType-property *CommandType.Text* bevat.
  - De naam van een Stored Procedure in als de CommandType-property *CommandType.StoredProcedure* bevat.
  - De naam van een table in als de CommandType-property *CommandType.TableDirect* bevat.

## Belangrijkste methods:

- **ExecuteNonQuery()**  
Hiermee voer je de command (het is te zeggen het SQL-statement of de Stored Procedure die je in de CommandText-property beschreven hebt) uit als dit SQL-statement of die Stored Procedure geen data (gegevens uit table(s)) teruggeeft. Voorbeelden van dat soort SQL-statements: insert, update, delete, create table.
- **ExecuteScalar()**  
Hiermee voer je de command (het is te zeggen het SQL-statement of de Stored Procedure die je in de CommandText-property beschreven hebt) uit als dit SQL-statement of die Stored Procedure data teruggeeft die bestaat uit één rij én één kolom. Voorbeelden van dat soort SQL-statements in de database Bank:  

```
select count(*) from Klanten
```

 geeft 2.  

```
select Saldo from Rekeningen where RekeningNr='111-1111111-70'
```

 geeft 1000.
- **ExecuteReader()**  
Hiermee voer je de command (het is te zeggen het SQL-statement of de Stored Procedure die je in de CommandText-property beschreven hebt) uit als dit SQL-statement of die Stored Procedure data bestaande uit meerdere rijen én/of kolommen teruggeeft.  
Voorbeeld van dat soort SQL-statements in de database Bank:

```
select * from klanten
geeft
```

1	Asterix
2	Obelix

- **Prepare()**  
 Hiermee vraag je aan de database dit command te compileren. Dit is interessant als je de command meerdere keren na elkaar uitvoert. De tweede en daaropvolgende keer zal de database de command sneller uitvoeren, omdat hij de gecompileerde versie van de command gebruikt.

## 5.2 Een Command uitvoeren met ExecuteNonQuery()

Voeg een class *RekeningenManager* toe aan het project *AdoGemeenschap*. Deze class zal alle databasebewerkingen bevatten van het concept rekeningen van een bank.

Voeg een functie *SaldoBonus* toe aan deze class *RekeningenManager*.

Deze functie verhoogt het saldo van alle rekeningen in de database bank met 10%. De returnwaarde van de functie is het aantal bijgewerkte rekeningen.

```
using System.Data;

...

public class RekeningenManager
{
 public Int32 SaldoBonus() (1)
 {
 var dbManager = new BankDbManager();
 using (var conBank = dbManager.GetConnection()) (2)
 {
 using (var comBonus = conBank.CreateCommand()) (3)
 {
 comBonus.CommandType = CommandType.Text; (4)
 comBonus.CommandText = "update Rekeningen set Saldo=Saldo*1.1"; (5)
 conBank.Open(); (6)
 return comBonus.ExecuteNonQuery(); (7)
 }
 }
 }
}
```

- (1) De functie geeft een getal terug dat aangeeft hoeveel rekeningen er in de database zijn bijgewerkt.
- (2) Je vraagt aan een *BankDbManager*-object via de functie *GetConnection* een connectie naar de database. Je gebruikt deze connectie in een using statement. Het using statement zal automatisch de connectie sluiten bij het verlaten van de functie.
- (3) Je vraagt aan de connectie een *DbCommand*-object. Gezien de class *DbCommand* de interface *IDisposable* implementeert is dit een object dat je na gebruik moet opkuisen. Door het command-object in een using statement te wikkelen, kuist .NET het object automatisch op bij het verlaten van de functie.

- (4) Je zult het SQL-statement dat de bonus toepast in het programma zelf als een string vermelden en niet als een Stored Procedure.
- (5) Je vermeldt het SQL-statement dat het saldo van alle rekeningen met 10% verhoogt.
- (6) Je opent de verbinding met de database.
- (7) Je voert het SQL-statement uit.  
De method `ExecuteNonQuery()` geeft je een getal terug:
  - Bij een Update statement: aantal gewijzigde records.
  - Bij een Insert statement: aantal toegevoegde records.
  - Bij een Delete statement: aantal verwijderde records.
 Jij gebruikt dit getal als returnwaarde van de functie.

Nu kan je deze method oproepen vanuit *AdoWPF*.

Voeg aan `MainWindow.xaml` een Button toe met volgende properties:

- Name `buttonBonus`
- Content `Bonus op alle rekeningen`

Dubbelklik de Button. Visual Studio maakt een routine en koppelt de routine aan het Click-event van de Button. Schrijf volgende code in de routine:

```
try
{
 var manager = new RekeningenManager(); (1)
 labelStatus.Content = manager.SaldoBonus() + " rekeningen aangepast"; (2)
}
catch (Exception ex)
{
 labelStatus.Content = ex.Message; (3)
}
```

- (1) Je maakt een object van het type `RekeningenManager` (dat de method `SaldoBonus` bevat).
- (2) Je roept de method `SaldoBonus` op, en toont het aantal bijgewerkte records (= aantal bijgewerkte rekeningen).
- (3) Als zich een fout voordeed (database niet gevonden, syntaxfout in het SQL-statement, toon je een foutmelding).

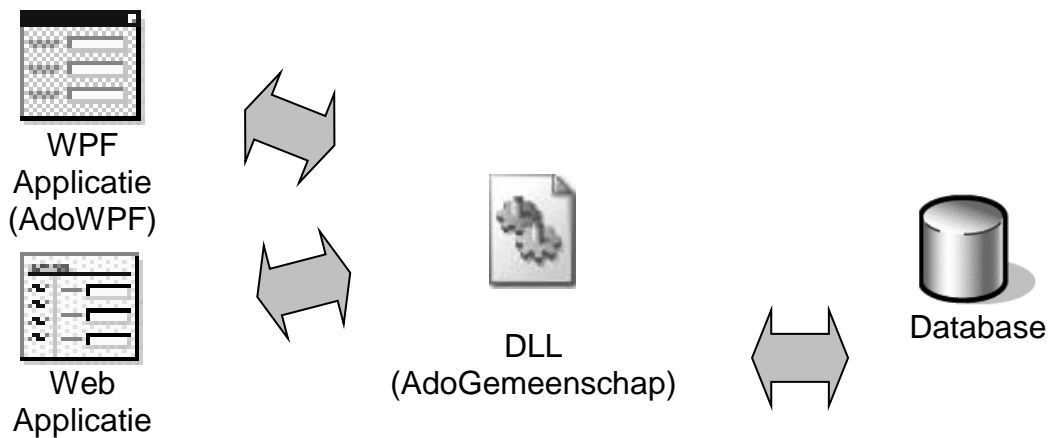
Je kunt deze code uittesten.

In de Server Explorer kan je de gewijzigde saldo's van de rekeningen zien. Kies in de Server Explorer de database Bank, klik met de rechtermuisknop op de tabel Rekeningen en kies voor Show Table Data.

Bij iedere uitvoering van de code verhogen de saldo's met 10%. Om de laatste toestand van de data te zien is het wel belangrijk dat in het venster waarmee je de data van een table inkijkt in de Server Explorer na iedere uitvoering van de code met de rechtermuisknop klikt en de opdracht *Refresh* kiest om het beeld te vernieuwen.

De DLL zit tussen de WPF-applicatie of eventueel een webapplicatie enerzijds en de database anderzijds.

Dit leidt tot een moderne meerlagige (multilayered) softwarearchitectuur:



### 5.3 Een SQL-statement met parameters (veranderlijke waarden).

Een SQL-statement bevat soms veranderlijke waarden.

Het SQL-statement waarmee je het saldo van één bepaalde rekening invult, heeft twee veranderlijke waarden:

- het rekeningnummer van de te wijzigen rekening
- het saldobedrag voor die rekening.

In het SQL-statement om 100 € te storten op rekening 111-1111111-70 zie je deze waarden onderstreept:

```
update Rekeningen
set Saldo=Saldo+100
where RekeningNr='111-1111111-70'
```

Deze waarden zijn anders als je 400 € stort op rekening 444-4444444-86:

```
update Rekeningen
set Saldo=Saldo+400
where RekeningNr='444-4444444-86'
```

Je geeft veranderlijke waarden in een SQL-statement aan met een parameternaam. Bij SqlServer moet de parameternaam beginnen met het teken @:

```
update Rekeningen
set Saldo=Saldo+@teStorten
where RekeningNr=@rekeningNr
```

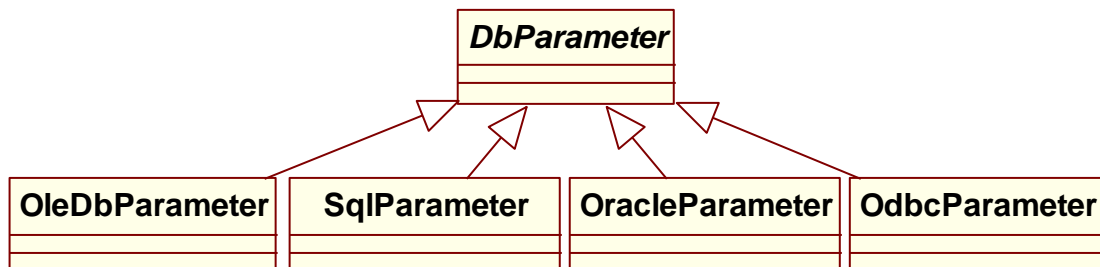
Vanuit C# code geef je via de Parameters-property de parameters van het Command hun echte waarde (zie verder in de code).

Iedere parameter uit de Parameters-property is terug een object.

Het type van dit object hangt terug af van de .NET Data Provider:

DataProvider	Class die een SQL parameter voorstelt
SqlServer	SqlParameter
OleDb	OleDbParameter
Oracle	OracleParameter
Odbc	OdbcParameter

Al deze classes zijn gelijkaardig: ze erven allen van de base class DbParameter:



Als je programmeert met een variabele van het type SqlParameter, kan je parameters in SQL-statements sturen naar een SqlServer (SqlExpress) database. Als je echter programmeert met een DbParameter-variabele, kan je code schrijven die werkt op elk type database.

Op een variabele van het type DbCommand kan je een functie CreateParameter uitvoeren. Deze functie geeft je een object onder de gedaante van het type DbParameter terug. Als het object verbonden met je DbCommand-variabele een OleDbCommand-object is, geeft deze functie je een OleDbParameter-object terug onder de gedaante van DbParameter. Als het object verbonden met je DbCommand-variabele een SqlCommand-object is, geeft deze functie je een SqlParameter-object terug onder de gedaante van DbParameter.

In de voorbeelden zal je deze CreateParameter functie gebruiken, om code te schrijven die met alle databasetypes kan werken.

Belangrijkste properties van DbParameter:

- **ParameterName**  
Hier vermeld je de naam van de parameter in het SQL-statement.
- **Value**  
Waarde die je naar de parameter wil sturen bij de uitvoering van het SQL-statement.

Als voorbeeld zal je bovenvermelde SQL-statement aanspreken met een Command-object vanuit een nieuwe functie *Storten* in de class *RekeningenManager* van het DLL-project *AdoGemeenschap*.

Daarna roep je deze functie op vanuit de WPF-applicatie.

Voeg de functie *Storten* toe aan de class *RekeningenManager*:

```
using System.Data.Common;
...

public Boolean Storten(Decimal teStorten, String rekeningNr) (1)
{
 var dbManager = new BankDbManager();
 using (var conBank = dbManager.GetConnection())
 {
 using (var comStorten = conBank.CreateCommand())
 {
 comStorten.CommandType = CommandType.Text;
 comStorten.CommandText = "update Rekeningen set
 Saldo=Saldo+@teStorten where RekeningNr=@RekeningNr"; (2)

 DbParameter parTeStorten = comStorten.CreateParameter(); (3)
 parTeStorten.ParameterName = "@teStorten"; (4)
 parTeStorten.Value = teStorten; (5)
 comStorten.Parameters.Add(parTeStorten); (6)

 DbParameter parRekeningNr = comStorten.CreateParameter();
 parRekeningNr.ParameterName = "@RekeningNr";
 parRekeningNr.Value = rekeningNr;
 comStorten.Parameters.Add(parRekeningNr);

 conBank.Open();
 return comStorten.ExecuteNonQuery() != 0; (7)
 }
 }
}
```

- (1) De functie krijgt het te storten bedrag als parameter binnen uit de Windows-applicatie. De functie krijgt ook het rekeningnr. van de aan te passen rekening binnen. De functie geeft een Boolean waarde terug met volgende betekenis:
  - true: de rekening is aangepast.
  - false: de rekening bestaat niet en is dus niet aangepast.
- (2) Je geeft aan het DbCommand-object een SQL-statement met twee parameters: @teStorten en @rekeningNr.
- (3) Je laat je command-object een parameter-object aanmaken.
- (4) Je vermeldt de naam van de parameter in het SQL-statement die je wilt aanspreken.
- (5) Als waarde voor de parameter vul je het te storten bedrag in die de functie zelf als parameter binnenkrijgt vanuit de Windows-applicatie.
- (6) Nadat het parameter-object is ingevuld, moet je het toevoegen aan de verzameling parameters van het command-object met de Add-method.
- (7) Je voert het SQL-statement uit. Als het aantal bijgewerkte records niet nul is, is er een rekening aangepast en geef je true terug. Anders bestaat de rekening niet en geef je false terug.



Nu kan je deze functie oproepen vanuit *MainWindow.xaml*.

Eerst maak je de lay-out:

Voeg een Label toe met volgende property:

- Content RekeningNr.:

Voeg daarnaast een TextBox toe met volgende properties:

- Name textBoxRekeningNr

Voeg nog een Label toe

- Content Te storten

Voeg daarnaast een TextBox toe met volgende properties:

- Name textBoxTeStorten

Voeg daar onder een Button toe met volgende properties:

- Name buttonStorten
- Content Storten

Dubbelklik deze Button. Visual Studio maakt een routine en koppelt deze aan het Click-event van deze Button. Schrijf volgende code in deze routine:

```

Decimal teStorten;
if (decimal.TryParse(textBoxTeStorten.Text, out teStorten)) (1)
{
 try
 {
 var manager = new RekeningenManager();
 if (manager.Storten(teStorten, textBoxRekeningNr.Text)) (2)
 { labelStatus.Content= "OK"; } (3)
 else
 { labelStatus.Content = "Rekening niet gevonden"; } (4)
 }
 catch (Exception ex)
 { labelStatus.Content = ex.Message; } (5)
}
else
{ labelStatus.Content = "Tik een getal bij het storten"; } (6)

```

- (1) Je probeert de inhoud van textBoxSaldo om te zetten naar een Decimal en bij te houden in de variabele teStorten.
- (2) Je roept de functie *Storten* van een *RekeningenManager-object* op. Als eerste parameter geef je de variabele teStorten mee, die de getalvoorstelling is van textBoxSaldo. Als tweede parameter geef je de inhoud van textBoxRekeningNr mee.
- (3) Als de functie true teruggeeft, toon je een positief bericht aan de gebruiker.
- (4) Als de functie false teruggeeft, toon je een foutmelding aan de gebruiker.
- (5) Als de functie *Storten* fout loopt, vang je hier de geworpen exception op.
- (6) De gebruiker tikte geen getal bij textBoxSaldo.

Je kunt de code uittesten en het gewijzigde saldo in de database nazien in de *Server Explorer*.

## 5.4 Een Stored procedure oproepen

Bij het gebruik van een Stored Procedure, plaats je het SQL-statement niet in je programmacode, maar sla je het SQL-statement op in de database zelf.

Dit heeft voordelen:

- Als je een SQL-statement als een string van je programmacode naar de database stuurt, moet de database bij iedere uitvoering van het SQL-statement meerdere stappen doen:
  - (1) Het SQL-statement controleren op syntaxfouten.
  - (2) Controleren of alle tables en velden die in het SQL-statement vermeld zijn daadwerkelijk bestaan.
  - (3) Onderzoeken welke kolomindex(en) de database kan gebruiken om het SQL-statement zo snel mogelijk uit te voeren.
 Bij het SQL-statement
 

```
update rekeningen set saldo=100 where rekeningnr=111-1111111-70
```

 kan de database dit record snel vinden door te zoeken op de primary key index (die gebaseerd is op de kolom rekeningnr).
  - (4) Het SQL-statement uitvoeren.

Als het SQL-statement in een Stored Procedure van de database zit, hoeven de stappen (1) tot (3) maar één keer te gebeuren: bij het opslaan van de Stored Procedure. Bij het uitvoeren van de Stored Procedure kan de database onmiddellijk naar stap (4) springen en dus tijd winnen.

- Als programmeur kan je een Stored Procedure uittesten in de database vooraleer hem te gebruiken in je programma. Bij deze test zie je al of het SQL-statement van de Stored Procedure geen syntaxfouten bevat en of het SQL-statement doet wat je verwacht.
- Als je het SQL-statement achteraf moet wijzigen vergt deze wijziging minder werk in een Stored Procedure dan in je programma.

### 5.4.1 De Stored Procedure aanmaken

Bij SqlServer kan je de Stored Procedure aanmaken in de *Server Explorer* van Visual Studio:

- Klik met de rechtermuisknop op *Stored Procedures* van de SqlServer-database *Bank* in de *Server Explorer* en kies *Add New Stored Procedure*. Je ziet een venster waarin je de code voor de Stored Procedure intikt.
- Wijzig deze code als volgt:

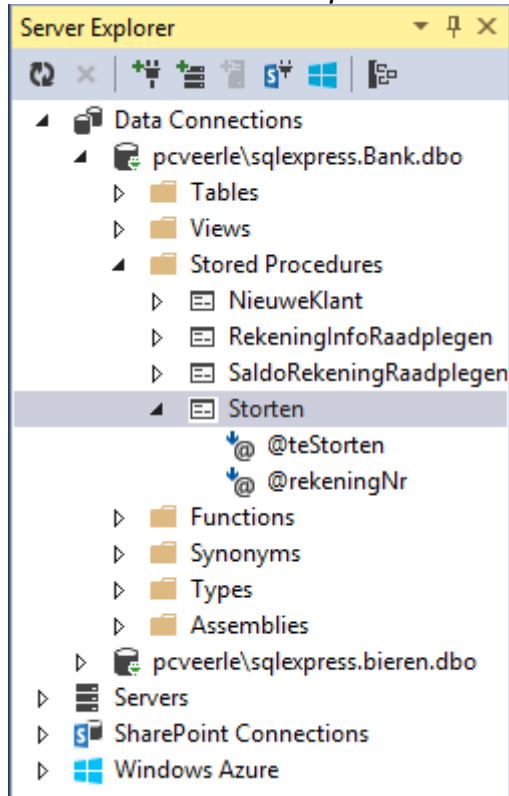
```
CREATE PROCEDURE Storten
 (@teStorten money, @rekeningNr nvarchar(14))
AS
 update Rekeningen
 set saldo=saldo+@teStorten
 where rekeningnr=@rekeningNr
```

De parameter @saldo krijgt het type money.

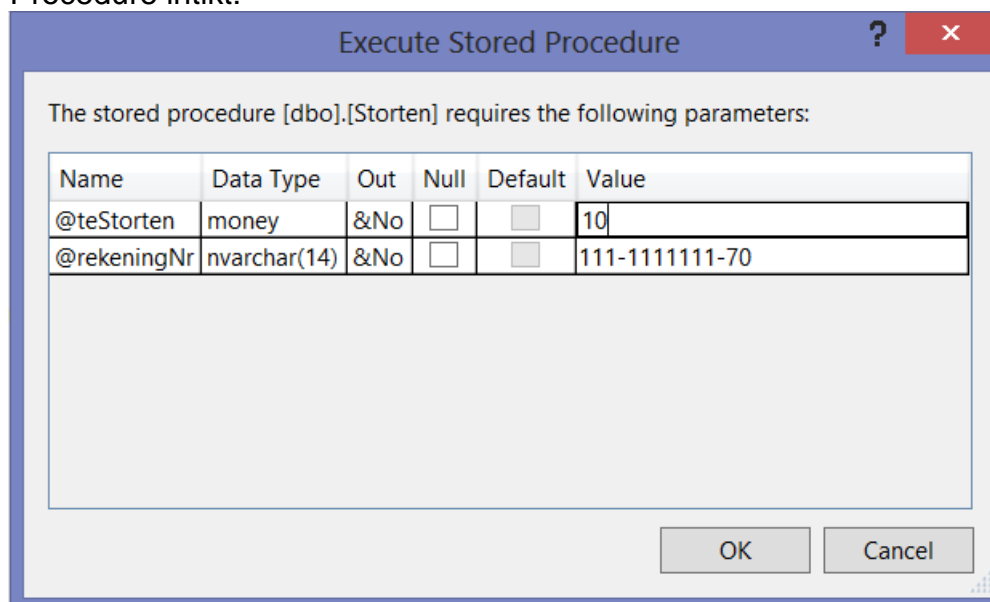
De parameter @rekeningNr krijgt het type nvarchar(14), wat wil zeggen: Unicode tekst met maximaal 14 tekens.

- De code kan worden uitgevoerd via een rechtermuisklik en execute.
- Sluit het ontwerpscherm van de Stored Procedure.

- Je ziet in de *Server Explorer* de Stored Procedure met de parameters:



- Je kunt de Stored Procedure uittesten.
- Klik met de rechtermuisknop op *Storten* en kies de opdracht *Execute*.
- Je ziet een venster waarin je een waarde voor de parameters van de Stored Procedure intikt:



- SqlServer voert de Stored Procedure uit.  
Je kunt het gewijzigde saldo zien in de tabel Rekeningen.

### 5.4.2 Parametertypes

Je zult DbParameter-objecten gebruiken om vanuit je code waarden te sturen naar de parameters beschreven in de Stored Procedure.

Naast een ParameterName en een Value-property heeft een DbParameter-object ook een DbType-property. Deze property bevat het type van de parameter in de stored procedure zelf.

Als je deze property niet invult, wordt hij bij de uitvoering van je code bepaald op basis van het type van de waarde die je naar de Value-property van het DbParameter-object stuurt:

Type van de waarde die je naar de Value-property stuurt:	DbType dat afgeleid wordt	Bijbehorend type in de Stored Procedure
Boolean	Boolean	Bit
Byte	Byte	tinyint
Char	String	Char
Date	DateTime	datetime
Decimal	Decimal	decimal
Double	Double	double
Integer	Int32	long integer
Short	Int16	integer
Single	Single	single
String	String	text, char, nchar, varchar, nvarchar

Als het type van de waarde die je naar de Value-property stuurt niet overeenstemt met het bijbehorende type in de stored procedure, vul je expliciet de DbType-property in, anders kan de waarde van je code verkeerd geïnterpreteerd worden door de stored procedure.

### 5.4.3 De Stored Procedure aanroepen

Wijzig de code van de functie *Storten* van de class *RekeningenManager* in het DLL-project *AdoGemeenschap*, zodat ze de Stored Procedure aanroept:

```
public Boolean Storten(Decimal teStorten, String rekeningNr)
{
 BankDbManager dbManager = new BankDbManager();
 using (var conBank = dbManager.GetConnection())
 {
 using (var comStorten = conBank.CreateCommand())
 {
 comStorten.CommandType = CommandType.StoredProcedure; (1)
 comStorten.CommandText = "Storten"; (2)

 DbParameter parTeStorten = comStorten.CreateParameter();
 parTeStorten.ParameterName = "@teStorten";
 parTeStorten.Value = teStorten;
 parTeStorten.DbType = DbType.Currency; (3)
 comStorten.Parameters.Add(parTeStorten);

 . . .
 }
 }
}
```

- (1) Als je een Stored Procedure oproept, plaats je de CommandType-property op CommandType.StoredProcedure.
- (2) In de CommandText-property vermeld je de naam van de op te roepen Stored Procedure.
- (3) Het type van de waarde die je naar de Value-property stuurt is decimal. Het parametertype is in de stored procedure zelf niet beschreven als decimal, maar als money (SqlServer synoniem voor currency). Je vult de DbType-property in met het type dat de parameter kreeg in de stored procedure. Voor de tweede parameter RekeningNr is het type van de waarde die je naar de Value-property stuurt een String. Het parametertype is in de stored procedure zelf nvarchar. Gezien er geen typeverschil is, hoef je de DbType-property niet in te vullen.

Je kunt de code uittesten en het gewijzigde saldo in de database nazien in de *Server Explorer*.

#### 5.4.4 Name based en positional based parameters.

Bij sommige databaseproducten (bijv. SqlServer) zijn de parameters van stored procedures *name based*.

De database associeert de parameters van de stored procedure enerzijds met de parameter-objecten (DbParameter-objecten) anderzijds op basis van de naam. Als een stored procedure de parameters @van en @tot bevat, moet je een DbParameter-object hebben met als ParameterName-property @van en een DbParameter-object met als ParameterName-property @tot.

De volgorde waarmee je de DbParameter-objecten toevoegt aan je DbCommand-object met de Add-method mag verschillen ten opzichte van de volgorde waarmee de parameters beschreven zijn in de stored procedure.

Bij andere databaseproducten (bijv. Access) zijn de parameters van stored procedure *positional based*.

De database associeert de parameters van de stored procedure enerzijds met de parameter-objecten (DbParameter-objecten) anderzijds op basis van de volgorde. Als een stored procedure (in volgorde) de parameters @van en @tot bevat, zal de eerste DbParameter die je aan je DbCommand-object toevoegt met de Add-method geassocieerd zijn met de eerst gedeclareerde parameter van de stored procedure (@van). De database houdt geen rekening met de ParameterName-property van dit DbParameter-object.

.NET associeert de tweede DbParameter die je aan je DbCommand-object toevoegt met de Add-method met de tweede gedeclareerde parameter van de stored procedure (@tot).

Je kunt code schrijven die werkt op name based parameter én op positional based parameters op volgende manier:

- Geef de ParameterName van je DbParameter-objecten exact dezelfde naam als de bijbehorende parameters gedeclareerd in de stored procedure. Zo zijn de name based parameter databaseproducten tevreden.
- Voeg de DbParameter-objecten aan je DbCommand-object toe in exact dezelfde volgorde als de bijbehorende parameters gedeclareerd in de stored procedure. Zo zijn de positional based parameter databaseproducten tevreden.



**Maak uit de oefenmap opgave 3**

## 6 TRANSACTIES

### 6.1 Algemeen

Nu je volop bezig bent Command-objecten te gebruiken om gegevens aan te passen is dit het geschikte moment te leren hoe je werkt met transacties.

Het doel van een transactie is meerdere SQL-statements als één geheel aan te zien. Dit geheel noemt men een transactie. Het is de verantwoordelijkheid van de database (Access, SqlServer, Oracle, ...) er voor te zorgen dat

- Ofwel de volledige transactie lukt, wat wil zeggen dat alle SQL-statements van de transactie uitgevoerd zijn. Dit noemt men een commit van de transactie.
- Ofwel de volledige transactie niet uitgevoerd werd bij problemen (fout in de database, fout in jouw applicatie, stroomuitval). Bij een probleem mag dus geen enkel van de SQL-statements van de transactie uitgevoerd zijn. Dit noemt men een rollback van de transactie.

Een voorbeeld van een transactie in de database Bank is het overschrijven van geld van een rekening naar een andere rekening.

Hiervoor zijn twee update statements nodig:

- Een update statement die het te transfereren geld aftrekt van het saldo van de van-rekening.
- Een update statement die het te transfereren geld bijtelt bij het saldo van de naar-rekening.

Voorbeeld: om 10 € over te schrijven van rekening 111-1111111-70 naar rekening 222-2222222-43 heb je twee update statements nodig:

Eerste statement:

```
update Rekeningen
set saldo=saldo - 10
where RekeningNr = '111-1111111-70'
```

Tweede opdracht:

```
update Rekeningen
set saldo=saldo + 10
where RekeningNr = '222-2222222-43'
```

Als één van deze twee statements mislukt, mag het andere statement ook niet uitgevoerd zijn.

Één van de statements kan mislukken door:

- Een fout in de databasesoftware
- Een fout in jouw applicatie
- Stroomuitval
- Het rekeningnummer van één van de rekeningen bestaat niet.

Door de SQL-statements te verzamelen in een transactie, zorgt de database er voor dat ofwel beide SQL-statements uitgevoerd worden, ofwel geen van beide.

Transacties hebben vier kenmerken (gekend als de ACID kenmerken):

- **Atomicity**  
De SQL-statements die tot de transactie behoren vormen één geheel. Een database voert een transactie helemaal, of niet uit. Als halverwege de transactie een fout gebeurt, brengt de database de bijgewerkte records terug in hun toestand juist voor de transactie begon.
- **Concistency**  
De transactie breekt geen databaseregels. Als bijvoorbeeld gedefinieerd is dat een kolom geen duplicaten kan bevatten, zal de database de transactie afbreken (rollback) op het moment dat die situatie dreigt voor te komen (ook al zou het eindresultaat de integriteit van de database niet schaden).
- **Isolation**  
Gedurende een transactie zijn de bewerkingen van de transactie niet zichtbaar voor andere lopende transacties. Dit gebeurt door het vergrendelen (locken) van de bijgewerkte records gedurende de transactie.
- **Durability**  
Een voltooide transactie is definitief vastgelegd in de database, zelfs al valt de computer uit juist na het voltooien van de transactie.

## 6.2 Transaction Isolation Levels

In een database die door meerdere gebruikers aangesproken wordt, kunnen zich volgende problemen voordoen:

- **Dirty Read**  
Een transactie leest recordwijzigingen die bij een andere transactie nog geen commit gekregen heeft. Dit isolation level gaat in tegen het Concistency kenmerk van een transactie en wordt daarom zelden gebruikt.
- **Nonrepeatable Read**  
Een transactie leest dezelfde records meer dan één keer, en krijgt bij de leesopdrachten na de eerste leesopdracht andere data uit deze records (omdat de data tussendoor door een andere transactie bijgewerkt of verwijderd werd).
- **Phantom Read**  
Een transactie leest dezelfde records meer dan één keer, en krijgt bij de leesopdrachten na de eerste leesopdracht meer records te zien dan bij de eerste leesopdracht. Dit kan als een andere transactie record(s) toevoegt die voldoen aan de where criteria van de leesopdracht van de oorspronkelijke transactie.

Om deze problemen op te lossen kan je op elke individuele transactie een transaction isolation level instellen. Hiermee bepaal je welke records van andere transacties zichtbaar zijn voor handelingen binnen de transactie waarop je het isolation level instelt.



De mogelijke transaction isolation levels:

- Read Uncommitted  
De transactie kan wijzigingen zien die tegelijk door andere transacties gebeuren en door die transactie nog geen commit gekregen hebben.
- SnapShot  
De eerste keer dat de transactie data leest, krijgt ze geen wijzigingen te zien die door andere transacties gebeuren. Van de gelezen data wordt een kopie gemaakt, speciaal voor deze transactie. Als de transactie dezelfde data nog eens leest, gebeurt de leesoperatie vanuit die kopie. Als de oorspronkelijke data ondertussen door andere transacties gewijzigd werd, krijgt de huidige transactie deze wijzigingen dus niet te zien.
- Read committed  
De transactie kan geen wijzigingen zien die tegelijk door andere transacties gebeuren (tot deze wijzigingen een commit krijgen).
- Repeatable read  
Data die door de transactie gelezen wordt, kan door geen andere transactie gewijzigd worden tot de huidige transactie tot een einde komt.
- Serializable  
De database houdt andere transacties in het oog, zodat ze geen data kunnen wijzigen, verwijderen of toevoegen die voldoen aan de criteria van de leesopdracht van de huidige transactie.

Problemen die kunnen optreden bij een bepaald isolation level:

Isolation level	Dirty read	Nonrepeatable read	Phantom read
Read uncommitted	Ja	Ja	Ja
Snapshot	Nee	Nee	Nee
Read committed	Nee	Ja	Ja
Repeatable	Nee	Nee	Ja
Serializable	Nee	Nee	Nee

De transaction isolation levels zijn vermeld in oplopende volgorde van 'isolatie' van een transactie ten opzichte van een andere transactie. Hierbij geldt echter dat hoe hoger het isolation level, hoe meer de database inspanningen moet doen om dit isolation level in stand te houden (performantie) en hoe meer er multi-user problemen kunnen optreden omdat meer records gelocked worden.

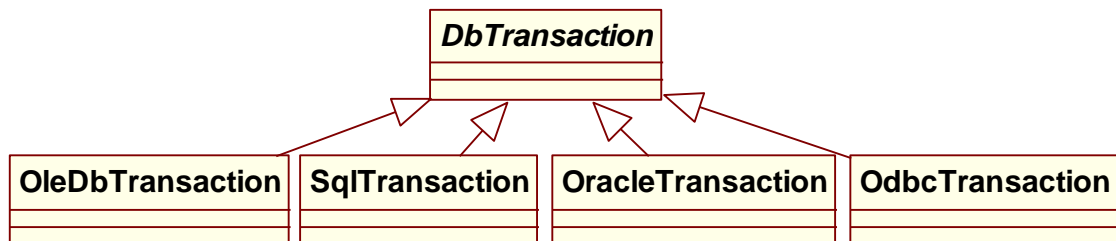
Ook is het zo dat niet alle databases al deze isolation levels ondersteunen.

## 6.3 De class DbTransaction

Iedere .NET Data Provider heeft zijn eigen class die een transactie voorstelt:

DataProvider	Class
SqlServer	SQLTransaction
OleDb	OleDbTransaction
Oracle	OracleTransaction
Odbc	OdbcTransaction

Al deze classes zijn gelijkaardig, want ze erven alle van de class DbTransaction:



Je kunt niet zelf een nieuw Transaction-object aanmaken:

```
var traOverschrijven = new DbTransaction();
```

geeft een compilerfout.

Je moet het Transaction-object laten aanmaken door je Connection-object (dat op dat moment al moet open staan) met de method BeginTransaction() en naar dit Transaction-object verwijzen met een referentie-variabele:

```
var traOverschrijven = conBank.BeginTransaction();
```

Als je conBank referentievariabele verwijst naar een OleDbConnection, krijg je een OleDbTransaction-object onder de gedaante van DbTransaction. Als je conBank referentievariabele verwijst naar een SqlConnection, krijg je een SqlTransaction-object onder de gedaante van DbTransaction.

Er bestaat ook een tweede versie van de method BeginTransaction, waarbij je het isolation level van de transactie kan instellen.

```
var traOverschrijven =
conBank.BeginTransaction(IsolationLevel.ReadCommitted);
```

Ieder Command-object dat je wilt opnemen in de transactie verbind je met de transactie via de Transaction-property van dat Command-object:

```
comStorten.Transaction = traOverschrijven;
```

Om een commit te doen op de transactie (alle statements van de transactie definitief wegschrijven in de database) gebruik je de Commit()-method:

```
traOverschrijven.Commit();
```

Om een rollback te doen op de transactie (alle statements van de transactie definitief ongedaan maken in de database) gebruik je de Rollback()-method:

```
traOverschrijven.Rollback();
```

Een rollback gebeurt automatisch bij elektriciteitsuitval, fouten in de database of jouw applicatie of als je de connectie sluit zonder een commit te doen.

Je maakt een method *Overschrijven* in de class *RekeningenManager* van het DLL - project *AdoGemeenschap*. Deze method kan je daarna oproepen vanuit de WPF-applicatie:

```
public void Overschrijven(Decimal bedrag, String vanRekening, String
naarRekening) (1)
{
 var dbManager = new BankDbManager();
 using (var conBank = dbManager.GetConnection())
 {
 conBank.Open();
 using (var traOverschrijven =
 conBank.BeginTransaction(IsolationLevel.ReadCommitted)) (2)
 {
 using (var comAftrekken = conBank.CreateCommand())
 {
 comAftrekken.Transaction = traOverschrijven; (3)
 comAftrekken.CommandType = CommandType.Text;
 comAftrekken.CommandText = "update Rekeningen set Saldo=Saldo-
 @bedrag where RekeningNr=@reknr";

 var parBedrag = comAftrekken.CreateParameter();
 parBedrag.ParameterName = "@bedrag";
 parBedrag.Value = bedrag;
 comAftrekken.Parameters.Add(parBedrag);

 var parRekNr = comAftrekken.CreateParameter();
 parRekNr.ParameterName = "@reknr";
 parRekNr.Value = vanRekening;
 comAftrekken.Parameters.Add(parRekNr);

 if (comAftrekken.ExecuteNonQuery() == 0)
 {
 traOverschrijven.Rollback();
 throw new Exception("Van rekening bestaat niet"); (4)
 }
 } // using comAftrekken
 using (var comBijstellen = conBank.CreateCommand())
 {
 comBijstellen.Transaction = traOverschrijven; (5)
 comBijstellen.CommandType = CommandType.Text;
 comBijstellen.CommandText = "update Rekeningen set
 Saldo=Saldo+@bedrag where RekeningNr=@reknr";

 var parBedrag = comBijstellen.CreateParameter();
 parBedrag.ParameterName = "@bedrag";
 parBedrag.Value = bedrag;
 comBijstellen.Parameters.Add(parBedrag);

 var parRekNr = comBijstellen.CreateParameter();
 parRekNr.ParameterName = "@reknr";
 parRekNr.Value = naarRekening;
 comBijstellen.Parameters.Add(parRekNr);

 if (comBijstellen.ExecuteNonQuery() == 0)
 {
 traOverschrijven.Rollback();
 throw new Exception("Naar rekening bestaat niet"); (6)
 }
 } // using comBijstellen
 }
 }
}
```

```
traOverschrijven.Commit();
} // using traOverschrijven
} // using conBank
}
```

(7)

- (1) De method krijgt het over te schrijven bedrag, het nummer van de van-rekening en het nummer van de naar-rekening als parameters binnen van de WPF-applicatie.
- (2) Je vraagt aan de connectie een transactie-object te maken.  
Gezien de class DbTransaction de interface IDisposable implementeert, wikkel je het object in een using opdracht. Zo zal .NET het DbTransaction-object automatisch opkuisen bij het verlaten van de sub.  
Je vraagt het isolation level committed. Zo zal de transactie geen records zien die nog door andere transacties bijgewerkt worden.
- (3) Je hebt een Command-object waarmee je het bedrag zal aftrekken van de van-rekening. Je maakt dit Command-object lid van de transactie.
- (4) Als de van-rekening niet bestaat, heeft het ook geen zin de naar-rekening aan te passen. Je doet een rollback op de transactie en werpt een fout. Door het werpen van de fout verlaat je ook de sub.
- (5) Je hebt een Command-object waarmee je het bedrag zal bijtellen bij de naar-rekening. Je maakt dit Command-object lid van de transactie.
- (6) Als de naar rekening niet bestaat moet de database ook de update van de van-rekening teniet doen.  
Je doet dit door op de transactie een rollback te doen. Je meldt aan de WPF-applicatie of de het probleem door een fout te werpen. Hierdoor verlaat je de sub.
- (7) Beide statements zijn vlekkeloos verlopen. De database mag de nieuwe saldo's definitief wegschrijven.  
Je bekomt dit door op de transactie een commit te doen.

Opmerking 1: beide Command-objecten gebruiken een SQL-statement in het programma zelf. Je kunt ook Stored Procedures gebruiken.

Opmerking 2: zolang een transactie niet afgewerkt is met commit() of rollback(), vergrendelt de database de records die je in de transactie bijwerkt. Andere gebruikers kunnen ze niet lezen of wijzigen. Houd je transactie daarom zo kort mogelijk in de tijd!

Nu kan je deze functie oproepen vanuit de WPF-applicatie.

Voeg een Window met de naam Overschrijven toe aan *AdoWPF* en stel dit Window in als opstartWindow van dit project.

Daarvoor moet je in App.xaml de StartupUri wijzigen.

```
<Application x:Class="AdoWPF.App"
 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 StartupUri="Overschrijven.xaml">
 <Application.Resources>
 </Application.Resources>
```

Voeg volgende controls toe:

The image shows a Windows Forms dialog box with the title 'Overschrijven'. Inside the dialog, there are three labels and three text boxes. The first label is 'Van Rekening Nr.' followed by a text box. The second label is 'Naar Rekening Nr.' followed by a text box. The third label is 'Bedrag:' followed by a text box. Below these text boxes is a button labeled 'Overschrijven'.

Een Label met volgende properties:

- Content Van Rekening Nr.:

Daarnaast een TextBox met volgende properties:

- Name textBoxVanRekNr  
(Als je de textboxes via de designer maakt, moet je de textboxes leegmaken.)

Onder het eerste Label een Label met volgende properties:

- Content Naar Rekening Nr.:

Daarnaast een TextBox met volgende properties:

- Name textBoxNaarRekNr

Onder het tweede Label een Label met volgende properties:

- Text Bedrag:

Daarnaast een TextBox met volgende properties:

- Name textBoxBedrag

Daaronder een Button met volgende properties

- Name buttonOverschrijven
- Content Overschrijven

Daaronder een Label met volgende properties:

- Name labelStatus
- Content Leeg maken

Dubbelklik de Button. Visual Studio maakt een routine en koppelt de routine aan het Click-event van de Button. Schrijf volgende code in de routine:

```
Decimal bedrag;
if (Decimal.TryParse(textBoxBedrag.Text, out bedrag)) (1)
{
 try
 {
 var manager = new RekeningenManager();
 manager.Overschrijven(bedrag, textBoxVanRekNr.Text, (2)
 textBoxNaarRekNr.Text); (3)
 labelStatus.Content = "OK"; (3)
 }
 catch (Exception ex) (4)
 {
 labelStatus.Content = ex.Message; (4)
 }
}
else
{
 labelStatus.Content = "bedrag bevat geen getal"; (5)
}
```

- (1) Je probeert de inhoud van textBoxBedrag om te zetten naar een Decimal.
- (2) Je voert de procedure Overschrijven uit.  
Als parameter bedrag voeder je de inhoud van textBoxBedrag.  
Als parameter vanRekening voeder je de inhoud van textBoxVanRekNr.  
Als parameter naarRekening voeder je de inhoud van textBoxNaarRekNr.
- (3) De functie is uitgevoerd zonder problemen. Je toont een positieve boodschap aan de gebruiker.
- (4) De functie heeft problemen gehad en heeft die gemeld door exceptions te werpen. Je vangt deze exceptions op en toont een foutboodschap aan de gebruiker.
- (5) De inhoud van textBoxBedrag is geen getal.

Je kunt de code uittesten en de gewijzigde saldo's in de *Server Explorer* raadplegen. Als je bij de naar-rekening een niet bestaand rekeningnummer intikt, zorgt de transactie er voor dat ook de van-rekening niet gewijzigd is.

## 6.4 De class TransactionScope

Er bestaat nog een andere class in het .NET framework om met transacties te werken: de class TransactionScope in de namespace System.Transactions.

Deze class heeft nadelen en voordelen vergeleken met de class DbTransaction die al in dit hoofdstuk behandeld werd:

- nadeel: de class TransactionScope werkt niet met alle types databases. De class werkt bijv. wel met SqlServer (SqlExpress), niet met Access.
- voordeel: de class TransactionScope is eenvoudiger te programmeren dan de class DbTransaction (zie voorbeeld verder).
- voordeel: de class TransactionScope laat local én distributed transacties toe. Een local transactie is een transactie die loopt over één database. De vorige class DbTransaction laat enkel local transacties toe. Een distributed transactie is een transactie die loopt over meerdere databases. Bij een distributed transactie doe je bewerkingen in meerdere databases. De distributed transactie zorgt er voor dat ofwel al deze bewerkingen vastgelegd worden, of (bij problemen) al deze bewerkingen teniet gedaan worden.

Als je met de class TransactionScope een distributed transactie doet (dus bewerkingen op meerdere databases), dan gebruikt deze class een Windows onderdeel met de naam DTC (Distributed Transaction Coordinator).

Dit onderdeel van Windows draait als een service (achtergrondproces). Het is belangrijk dat deze service draait ter ondersteuning van de distributed transactie. Je kunt via *Control Panel, Administrative Tools, Services* (🔧) nazien of deze service draait, en zo nodig starten.

De werking van de class TransactionScope is anders dan die van DbTransaction.

Je maakt een TransactionScope-object binnen een using structuur.

```
using (var eenTransactie
 = new System.Transactions.TransactionScope())
{. . .}
```

Alle connecties en commands die je binnen deze using structuur aanmaakt, behoren automatisch tot één en dezelfde transactie.

Als alle bewerkingen goed aflopen, voer je op je TransactionScope-object de method Complete uit: `eenTransactie.Complete();`

Bij het uitvoeren van deze method gebeurt een commit van alle bewerkingen van alle databaseconnecties die je uitgevoerd hebt binnen de using structuur.

Als je de using structuur verlaat zonder de method Complete() uit te voeren (bijv. een exception treedt op), gebeurt automatisch een rollback van alle bewerkingen van alle databaseconnecties die je opende binnen de using structuur.

TransactionScope-objecten kunnen in elkaar genest zijn:

```
using (var eenTransactie
 = new System.Transactions.TransactionScope)
{
 using (var eenGenesteTransactie
 = new System.Transactions.TransactionScope)
 { . . . }
}
```

Er bestaan TransactionScope constructors waarbij je een parameter voedert van het type TransactionScopeOption. Met deze parameter beslis je hoe het TransactionScope-object zich gedraagt ten opzichte van andere TransactionScope-objecten waarbinnen het genest is. De parameter kan volgende waarden bevatten:

- Required  
Er wordt voor het TransactionScope-object een transactie gemaakt in de database als de TransactionScope niet binnen een andere TransactionScope genest is. Als er wel genest is, gebruikt het TransactionScope-object de transactie die al door het bovenliggend TransactionScope-object gestart werd.
- RequiresNew  
Er wordt voor het TransactionScope-object een nieuwe transactie gemaakt, zelfs als het TransactionScope-object genest is binnen een ander TransactionScope-object.
- Suppress  
Alle databasebewerkingen binnen dit TransactionScope-object behoren niet tot een transactie, zelfs als het TransactionScope-object genest is binnen een ander TransactionScope-object.

Om het isolation level van de transactie in te stellen bestaan er TransactionScope constructors die een parameter van het type TransactionOption aanvaarden. Een TransactionOption-object heeft een IsolationLevel-property om de isolation level in te stellen.

De class TransactionScope bevindt zich in de DLL System.Transactions.

Leg in het project *AdoGemeenschap* een referentie naar *System.Transactions.dll*:

- Klik met de rechtermuisknop op het project *AdoGemeenschap* in de Solution Explorer.
- Kies *Add Reference*.
- Kies voor Assemblies en daaronder Framework.
- Zet bij System.Transactions onder Name een vinkje.
- Klik op de knop *OK*.



In het voorbeeld (method Overschrijven) zal je een distributed transactie gebruiken:

Voor een distributed transactie hebben we twee verschillende databases nodig.

Open de SQL Server Management Studio

- Je tikt in het venster *Connect to Server* *.\sqlexpress*, als je SQL Express gebruikt, of je kiest je volwaardige SQL Server uit de lijst.
- Je laat de keuze bij *Authentication* op *Windows Authentication*.
- Je kiest *Connect*.
- Je kiest in het menu *File* de opdracht *Open* en de vervolgoopdracht *File*.
- Je selecteert *createBank2.sql* uit de oefenmap en je kiest *Open*.
- Je kiest in de knoppenbalk de opdracht *Execute*.
- Je klikt in het linkerdeel met de rechtermuisknop op de map *Databases* en je kiest de opdracht *Refresh*.
- Je klappt de map *Databases* open en je ziet de database Bank2.

Voeg deze Bank2 database ook toe aan de Data Connections in je Server Explorer

Vergeet niet een extra connectionstring toe te voegen in de App.config.

```
<add name="Bank2" connectionString=
 "server=.\sqlexpress;database=bank2;integrated security=true"
 providerName="System.Data.SqlClient"/>
```

Voeg aan AdoGemeenschap een class Bank2DbManager toe met volgende code:

```
public class Bank2DbManager
{
 private static ConnectionStringSettings conBankSetting =
 ConfigurationManager.ConnectionStrings["Bank2"];
 private static DbProviderFactory factory =
 DbProviderFactories.GetFactory(conBankSetting.ProviderName);

 public DbConnection GetConnection()
 {
 var conBank = factory.CreateConnection();
 conBank.ConnectionString = conBankSetting.ConnectionString;
 return conBank;
 }
}
```

Voeg indien nodig de reference *System.Transactions* toe aan het project.

Bovenaan moet je ook nog de nodige references toevoegen:

```
using System.Transactions;
using System.Configuration;
using System.Data.Common;
```

Wijzig de functie Overschrijven in RekeningenManager volgt:

```
using System.Transactions;
```

```
...
```

```
public void Overschrijven(Decimal bedrag, String vanRekening, String
naarRekening)
{
 var dbManager = new BankDbManager();
 var dbManager2 = new Bank2DbManager();

 var opties = new TransactionOptions();
 opties.IsolationLevel =
 System.Transactions.IsolationLevel.ReadCommitted;
 using (var traOverschrijven
 = new TransactionScope(TransactionScopeOption.Required, opties)) (1)
 {
 using (var conBank = dbManager.GetConnection()) (2)
 {
 using (var comAftrekken = conBank.CreateCommand()) (3)
 {
 comAftrekken.CommandType = CommandType.Text;
 comAftrekken.CommandText = "update Rekeningen
 set Saldo=Saldo-@bedrag where RekeningNr=@reknr";

 var parBedrag = comAftrekken.CreateParameter();
 parBedrag.ParameterName = "@bedrag";
 parBedrag.Value = bedrag;
 comAftrekken.Parameters.Add(parBedrag);

 var parRekNr = comAftrekken.CreateParameter();
 parRekNr.ParameterName = "@reknr";
 parRekNr.Value = vanRekening;
 comAftrekken.Parameters.Add(parRekNr);

 conBank.Open();
 if (comAftrekken.ExecuteNonQuery() == 0)
 {
 throw new Exception("Van rekening bestaat niet");
 }
 } // using comAftrekken
 } // using conBank
 using (var conBank = dbManager2.GetConnection()) (4)
 {
 using (var comBijstellen = conBank.CreateCommand()) (5)
 {
 comBijstellen.CommandType = CommandType.Text;
 comBijstellen.CommandText = "update Rekeningen set
 Saldo=Saldo+@bedrag where RekeningNr=@reknr";

 var parBedrag = comBijstellen.CreateParameter();
 parBedrag.ParameterName = "@bedrag";
 parBedrag.Value = bedrag;
 comBijstellen.Parameters.Add(parBedrag);

 var parRekNr = comBijstellen.CreateParameter();
 parRekNr.ParameterName = "@reknr";
 parRekNr.Value = naarRekening;
 comBijstellen.Parameters.Add(parRekNr);

 conBank.Open();
```

```

 if (comBijtellen.ExecuteNonQuery() == 0)
 {
 throw new Exception("Naar rekening bestaat niet"); (6)
 }
 traOverschrijven.Complete(); (7)
} // using comBijtellen
} // using conBank
} // using traOverschrijven
}

```

- (1) Je maakt een TransactionScope-object in een using statement. Alle connecties en commands binnen dit using blok behoren automatisch tot één en dezelfde transactie. Het TransactionScope-object vraagt een nieuwe transactie of kan meespelen in een bestaande transactie die gestart is door een hoger liggend TransactionScope-object. Je vraagt als isolation level committed.
- (2) Deze connectie behoort automatisch tot een transactie: ze is aangemaakt binnen de using van het TransactionScope-object traOverschrijven.
- (3) Deze command behoort automatisch tot een transactie: ze is aangemaakt binnen de using van het TransactionScope-object traOverschrijven.
- (4) Ook deze connectie behoort automatisch tot dezelfde transactie: ze is aangemaakt binnen de using van het TransactionScope-object traOverschrijven. (let er wel op dat we hier DbManager2 gebruiken want we willen een connectie met de tweede databank Bank2)
- (5) Ook deze command behoort automatisch tot dezelfde transactie: ze is aangemaakt binnen de using van het TransactionScope-object traOverschrijven.
- (6) Je werpt een fout. Op de transactie van het TransactionScope-object gebeurt automatisch een rollback. De bewerkingen die je uitvoerde binnen de transactie worden teniet gedaan.
- (7) Als alles gelukt is, voer je de method Complete uit op je TransactionScope-object. Nu gebeurt een commit op de transactie van het TransactionScope-object. De bewerkingen die je uitvoerde binnen de transactie worden definitief vastgelegd.

Eerst controleer je of de Distributed Transaction Coordinator als service draait:

- Ga naar het *Control Panel* van Windows.
- Kies *Administrative Tools*.
- Kies *Services*.
- Kies de service met als *Name* eigenschap *Distributed Transaction Coordinator*.
- Als de service niet draait, zie je links *Start*, kies dan deze opdracht.

Interessant is ook dat je de werking van de Distributed Transaction Coordinator kan volgen, via zijn beheervenster:

- Ga naar het *Control Panel* van Windows.
- Kies *Administrative Tools* onder System and Security
- Kies *Component Services*.
- Klik links (onder *Console Root*) op *Component Services*
- Navigeer via *Component Services/Computers/My Computer/Distributed Transaction Coordinator/Local DTC* naar *Transaction Statistics*

Je ziet dat de Distributed Transaction Coordinator nog geen enkele transactie uitgevoerd heeft: bij *Total* staat 0.

Current		
Active	0	<input type="text"/>
Max. Active	0	<input type="text"/>
In Doubt	0	<input type="text"/>

Aggregate		
Committed	0	<input type="text"/>
Aborted	0	<input type="text"/>
Forced Commit	0	<input type="text"/>
Forced Abort	0	<input type="text"/>
Unknown	0	<input type="text"/>
Total	0	<input type="text"/>

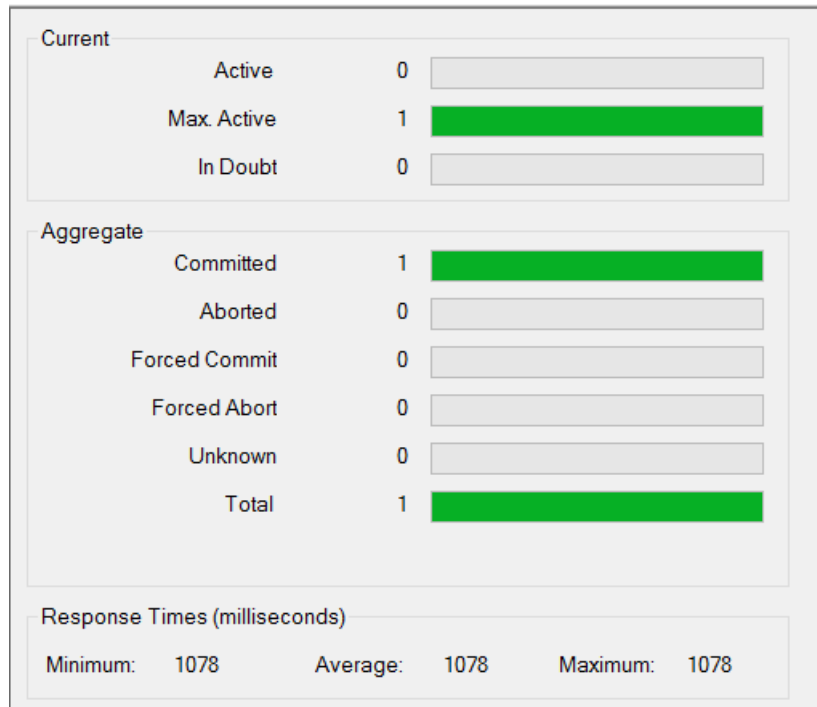
  

Response Times (milliseconds)			
Minimum:	0	Average:	0
		Maximum:	0

Zorg er voor dat in je XML-configuratiebestand de connectionstrings Bank en Bank2 voor SqlServer actief zijn, en de rest in commentaar staat.

Voer het programma uit en tik een correcte van rekening, naar rekening en bedrag. In de databases Bank en Bank2 zijn beide rekeningen aangepast.

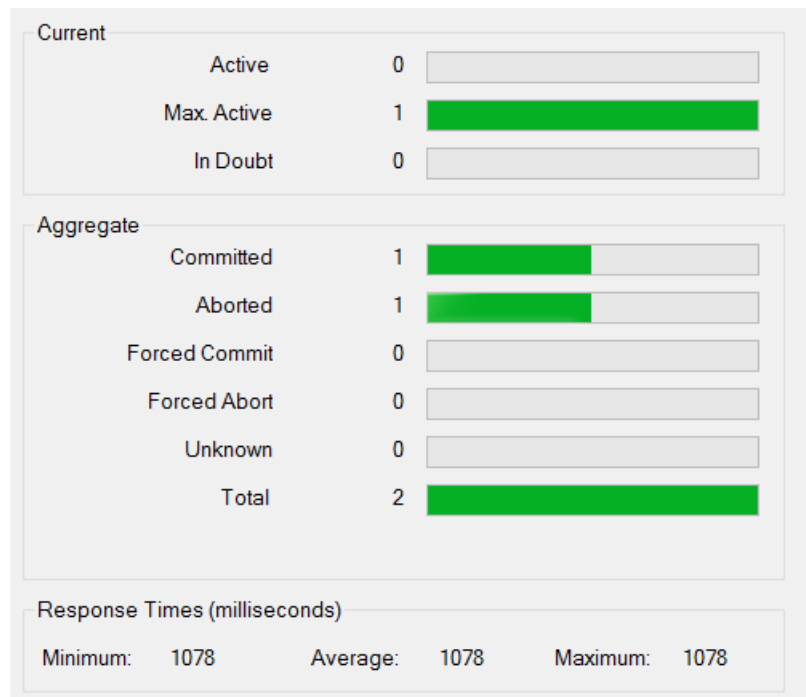
Als je terug gaat kijken in het beheerscherm van de Distributed Transaction Coordinator, zie je dat één transactie uitgevoerd werd en een commit kreeg: bij *Committed* staat 1



Voer het programma uit en tik een correcte van rekening, en bedrag, maar een verkeerde naar rekening.

In de database is geen enkele rekening aangepast.

Als je terug gaat kijken in het beheerscherm van de Distributed Transaction Coordinator, zie je dat er ondertussen twee transacties zijn uitgevoerd, waarvan één een commit kreeg (de eerste keer je het programma uitvoerde) en een twee een abort kreeg (de tweede keer je het programma uitvoerde): bij *Committed* staat 1, bij *Aborted* staat 1



Maak uit de oefenmap opgave 4

## 7 ÉÉN WAARDE UIT EEN DATABASE LEZEN

### 7.1 Algemeen

Met sommige SQL-statements of Stored Procedures lees je slechts één waarde uit de database. Voorbeelden:

```
select count(*) from Klanten
select Saldo from Rekeningen where RekeningNr='111-1111111-70'
```

Deze ene waarde kan je in je programma opvragen met de method `ExecuteScalar()` van het `Command`-object.

`ExecuteScalar()` gedraagt zich als een functie die de éne waarde van het SQL-statement teruggeeft als een waarde van het type-object.

### 7.2 De Stored Procedure

Bij `SqlServer` kan je de Stored Procedure aanmaken in de *Server Explorer* van Visual Studio:

- Klik met de rechtermuisknop op *Stored Procedures* van de `SqlServer`-database *Bank* in de *Server Explorer* en kies *Add New Stored Procedure*. Je ziet een venster waarin je de code voor de Stored Procedure kan intikken.
- Wijzig deze code als volgt:

```
CREATE PROCEDURE SaldoRekeningRaadplegen
(
 @rekeningNr nvarchar(14)
)
AS
select Saldo
from Rekeningen
where RekeningNr=@rekeningNr
```

- Voer de procedure uit met een rechtermuisklik/execute.
- Sluit het ontwerpscherm van de Stored Procedure.
- Je ziet in de *Server Explorer* de Stored Procedure met de parameters.
- Je kunt de Stored Procedure uittesten:  
Klik met de rechtermuisknop op *SaldoRekeningRaadplegen* en kies de opdracht *Execute*.
- Je ziet een venster waarin je een waarde voor de parameter van de Stored Procedure intikt. Tik één van de rekeningnummers.
- Visual Studio voert de Stored Procedure uit.  
Je kunt het saldo zien in een venster onder je code.

T-SQL		Results	Message
Saldo			
1	2000,00		
Return Value			
1	0		

### 7.3 De functie SaldoRekeningRaadplegen

Je maakt een functie die via de ExecuteScalar-method van het DbCommand-object het saldo van één rekening opvraagt, via de stored procedure.

Je maakt de functie *SaldoRekeningOpvragen* in de class *RekeningenManager* van het DLL-project *AdoGemeenschap*. Deze functie zal je daarna kunnen oproepen vanuit de WPF-applicatie:

```
public Decimal SaldoRekeningRaadplegen(String rekeningNr) (1)
{
 var dbManager = new BankDbManager();
 using (var conBank = dbManager.GetConnection())
 {
 using (var comSaldo = conBank.CreateCommand())
 {
 comSaldo.CommandType = CommandType.StoredProcedure;
 comSaldo.CommandText = "SaldoRekeningRaadplegen";
 var parRekNr = comSaldo.CreateParameter();
 parRekNr.ParameterName = "@rekeningNr";
 parRekNr.Value = rekeningNr;
 comSaldo.Parameters.Add(parRekNr);
 conBank.Open();
 Object resultaat = comSaldo.ExecuteScalar(); (2)
 if (resultaat == null) (3)
 {
 throw new Exception("Rekening bestaat niet");
 }
 else
 {
 return (Decimal)resultaat; (4)
 }
 }
 }
}
```

- (1) De functie krijgt het rekeningnummer waarvan je het saldo moet opzoeken binnen als een parameter van de WPF-applicatie die de functie oproept. De functie zelf geeft een Decimal-waarde terug: het saldo van de gevraagde rekening.
- (2) Je voert de Command uit met de method ExecuteScalar(). Het resultaat van de method is het saldo dat de Stored Procedure teruggeeft. Het type van het resultaat van ExecuteScalar() is altijd Object.
- (3) Als het gevraagde rekeningnummer niet bestaat, heeft de Stored Procedure geen record gevonden. Dan is het resultaat van ExecuteScalar() gelijk aan *null*. Je meldt dit als een exception naar de WPF-applicatie. Opmerking: als het SQL-statement een statistische functie (sum, avg, ...) teruggeeft die geen records gevonden heeft, is het resultaat van ExecuteScalar() niet *null*, maar de constante DBNull.Value.
- (4) Je zet het resultaat van ExecuteScalar() om naar een Decimal type en geeft deze waarde naar de WPF-applicatie terug.

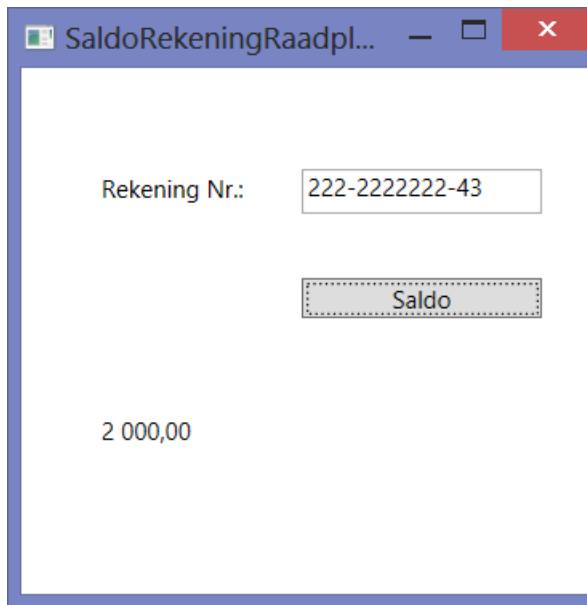
Nu kan je deze functie oproepen vanuit de WPF-applicatie.

Voeg een Window met de naam *SaldoRekeningRaadplegen* toe aan *AdoWPF* en stel dit Window in als opstartWindow van dit project.

Daarvoor moet je in App.xaml de StartupUri wijzigen.

```
<Application x:Class="AdoWPF.App"
 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 StartupUri="SaldoRekeningRaadplegen.xaml">
 <Application.Resources>
 </Application.Resources>
```

Voeg volgende controls toe:



Een Label met volgende properties:

- Content      Rekening Nr.:

Daarnaast een TextBox met volgende properties:

- Name          textBoxRekeningNr

Daar onder een Button met volgende properties:

- Name          buttonSaldo
- Content      Saldo

Daaronder een Label met volgende properties:

- Name          labelStatus
- Content      Leeg maken

Opmerking: maak het Label breed genoeg.

Dubbelklik buttonSaldo. Visual Studio maakt een routine en koppelt de routine aan het Click-event van de Button.

Voeg bovenaan de code `using AdoGemeenschap;` toe

Schrijf volgende code in de routine:

```
var manager = new RekeningenManager();
try
{
 labelStatus.Content =
manager.SaldoRekeningRaadplegen(textBoxRekeningNr.Text).ToString("N"); (1)
}
catch (Exception ex)
```



```
{
 labelStatus.Content = ex.Message; (2)
}
```

- (1) Je roept de functie SaldoRekeningRaadplegen op en voedert als parameter de inhoud van textBoxRekeningNr. Je krijgt een Decimal-waarde terug van de functie (het saldo van die rekening) en toont dit resultaat in labelStatus. Door "N" mee te geven als parameter van de ToString-method, zet .NET de decimal-waarde naar tekst om met puntjes tussen 1000-tallen.
- (2) Als de functie een probleem heeft (bijv. rekening niet gevonden), werpt de functie een Exception. Je vangt hier deze exception op en je toont een foutboodschap aan de gebruiker.

Je kunt de code uittesten en een saldo raadplegen.



**Maak uit de oefenmap opgave 5**

## 8 ENKELE WAARDES LEZEN UIT EEN STORED PROCEDURE VAN SQLSERVER

### 8.1 Algemeen

Als je slechts enkele waarden wil lezen (praktisch tot ongeveer maximaal 4 waarden) uit een database kan je bij SqlServer (en andere grotere databases zoals Oracle en DB2) in een Stored Procedure output-parameters gebruiken. (Access ondersteunt geen output-parameters.)

Met output-parameters kan de Stored Procedure gegevens in de database opzoeken (of berekeningen maken) en de opgezochte waarden via deze output-parameters naar je programma sturen.

Opmerkingen:

- Een Stored Procedure kan zowel input-parameters (waarden van je programma naar de Stored Procedure) als output-parameters bevatten (waarden van de Stored Procedure naar je programma).
- Er bestaan ook input-output-parameters in Stored Procedures. Dit zijn tweerichtingsparameters: ze krijgen een waarde binnen van het programma, de Stored Procedure wijzigt de waarde en stuurt ze terug naar het programma.
- Een Stored Procedure met output-parameters roep je ook op met een Command-object. Je gebruikt de method `ExecuteNonQuery()` om de Stored Procedure te starten.
- Als je slechts één waarde van een Stored Procedure nodig hebt, zal het gebruik van een output-parameter in SqlServer sneller werken dan het gebruik van de method `ExecuteScalar()` van je `SqlCommand`.
- Naast output-parameters kent een SqlServer Stored Procedure ook een return-parameter. De bedoeling van een return-parameter is dezelfde als die van een output-parameter: een waarde van de Stored Procedure naar het programma sturen op het einde van de Stored Procedure. De return-parameter heeft echter volgende beperkingen ten opzichte van output-parameters:
  - Een Stored Procedure kan maar één return-parameter hebben, terwijl een Stored Procedure meerdere output-parameters kan hebben.
  - In een return-parameter kan je enkel een waarde die een geheel getal is verwerken. Output-parameters kunnen elk type waarde (int, nvarchar, money, ...) verwerken.

### 8.2 Voorbeeld

Je zult het saldo én de naam van de klant opzoeken van de rekening met een bepaald rekeningnummer.

#### 8.2.1 De Stored Procedure maken in SqlServer

- Klik met de rechtermuisknop op *Stored Procedures* van de SqlServer-database *Bank* in de *Server Explorer* en kies *Add New Stored Procedure*. Je ziet een venster waarin je de code voor de Stored Procedure kan intikken.

- Wijzig deze code als volgt:

```
CREATE PROCEDURE RekeningInfoRaadplegen
(
 @RekeningNr nvarchar(14),
 @Saldo money OUTPUT,
 @KlantNaam nvarchar(50) OUTPUT
)
AS
select @Saldo=Saldo, @KlantNaam=Naam
from Rekeningen inner join Klanten
on Rekeningen.KlantNr = Klanten.KlantNr
where RekeningNr=@RekeningNr
```

- Het te zoeken rekeningnummer is de input-parameter @RekeningNr.
- Output-parameters (@Saldo en @KlantNaam) geef je het sleutelwoord OUTPUT mee.
- Je haalt waarden uit de kolom Saldo en de kolom Naam op uit de join gegevens tussen de table Rekeningen en de table Klanten voor de rekening waarvan het rekeningnummer gelijk is aan de parameter @RekeningNr. Je stuurt de waarde van de kolom Saldo naar de output-parameter @Saldo (@Saldo=Saldo) en je stuurt de waarde van de kolom Naam naar de output-parameter @KlantNaam (@KlantNaam=Naam)
- Klik op Update
- Sluit het ontwerpscherm van de Stored Procedure.
- Je ziet in de *Server Explorer* de Stored Procedure met de parameters.
- Je kunt de Stored Procedure uittesten:  
Klik met de rechtermuisknop op *RekeningInfoRaadplegen* en kies de opdracht *Execute*.
- Je ziet een venster waarin je een waarde voor de parameters van de Stored Procedure intikt. Tik bij @RekeningNr één van de rekeningnummers. Kies bij @Saldo en @KlantNaam voor NULL: dit zijn geen input-parameters, dus vul je ze met niets (NULL) in bij het starten van de stored procedure. (of laat de velden gewoon leeg)
- Klik op OK.
- Onderaan zie je het resultaat van de stored procedure:

T-SQL		Results	Message
	@Saldo	@KlantNaam	
1	1200,00	Asterix	
Return Value			
1	0		

### 8.2.2 De functie *RekeningInfoRaadplegen*

Je maakt een functie *RekeningInfoRaadplegen* in de class *RekeningenManager* van het DLL-project *AdoGemeenschap*. Deze functie zal je daarna kunnen oproepen vanuit de WPF-applicatie:

De functie krijgt een parameter binnen van de WPF-applicatie: het rekeningnr. van de rekening waarvan je de informatie ophaalt.

De functie zal het saldo van de rekening en de naam van de klant van de rekening teruggeven. Op zich kan een functie echter maar één waarde teruggeven. Je kunt dit oplossen door de twee waarden (saldo en klantnaam) te beschrijven als properties van een nieuwe class (*RekeningInfo*). Dan kan de functie *RekeningInfoRaadplegen* een object van deze class (*RekeningInfo*) teruggeven.

Je maakt eerst de class *RekeningInfo* in het DLL-project *AdoGemeenschap*:

```
public class RekeningInfo
{
 private Decimal saldoValue;
 private String klantNaamValue;

 public Decimal Saldo
 {get{return saldoValue;}}

 public String Klantnaam
 {get{return klantNaamValue;}}

 public RekeningInfo(Decimal saldo, String klantNaam)
 {
 saldoValue = saldo;
 klantNaamValue = klantNaam;
 }
}
```

Nu maak je de functie *RekeningInfoRaadplegen* in de class *RekeningenManager*:

```
public RekeningInfo RekeningInfoRaadplegen(String rekeningNr) (1)
{
 var dbManager = new BankDbManager();
 using (var conBank = dbManager.GetConnection())
 {
 using (var comSaldo = conBank.CreateCommand())
 {
 comSaldo.CommandType = CommandType.StoredProcedure;
 comSaldo.CommandText = "RekeningInfoRaadplegen";

 var parRekNr = comSaldo.CreateParameter();
 parRekNr.ParameterName = "@rekeningNr";
 parRekNr.Value = rekeningNr;
 comSaldo.Parameters.Add(parRekNr); (2)

 var parSaldo = comSaldo.CreateParameter();
 parSaldo.ParameterName = "@Saldo"; (3)
 parSaldo.DbType = DbType.Currency; (4)
 parSaldo.Direction = ParameterDirection.Output; (5)
 comSaldo.Parameters.Add(parSaldo);

 var parKlantNaam = comSaldo.CreateParameter();
 parKlantNaam.ParameterName = "@KlantNaam";
 parKlantNaam.DbType = DbType.String;
 parKlantNaam.Size = 50;
 parKlantNaam.Direction = ParameterDirection.Output; (6)
 comSaldo.Parameters.Add(parKlantNaam);
 }
 }
}
```

```

conBank.Open();
comSaldo.ExecuteNonQuery();
if (parSaldo.Value.Equals(DBNull.Value))
{
 throw new Exception("Rekening bestaat niet");
}
else
{
 return new
RekeningInfo((Decimal)parSaldo.Value, (String)parKlantNaam.Value);
}
}
}

```

- (1) De functie krijgt het rekeningnummer van de op te zoeken rekening binnen van de WPF-applicatie.  
De functie geeft een RekeningInfo-object terug.
- (2) Je voert de input-parameter @RekeningNr de waarde van de parameter rekeningNr die in de functie binnengekomen is.
- (3) Je vermeldt de naam van de output-parameter in de Stored Procedure.
- (4) Je vermeldt het type van de output-parameter in de Stored Procedure.  
Bij output-parameters moet je *altijd* het type vermelden.
- (5) Je vermeldt expliciet dat het om een output-parameter gaat. Als je dit vergeet te doen, aanziet .NET de parameter als input-parameter!
- (6) Bij een output-parameter die tekst zal bevatten (zoals een nvarchar parameter in de Stored Procedure) moet je in de Size-property het maximum aantal tekens vermelden.
- (7) Je voert het SqlCommand-object (en dus de Stored Procedure) uit met de method executeNonQuery(). De Stored Procedure zoekt de rekeninginformatie op en geeft het saldo en de klantnaam via de output-parameters aan je programma.
- (8) Als de gevraagde rekening niet gevonden werd, bevatten de output-variabelen NULL-waarden. Je kunt dit controleren door een output-variabele te vergelijken met de speciale waarde DBNull.Value.
- (9) Je maakt een RekeningInfo-object. Aan de constructor van dit object voeder je de waarde van de output-parameter @Saldo en de waarde van de output-parameter @KlantNaam. De waarde van een parameter krijg je als iets van het type Object. Daarom doe je typeconversie.

Nu kan je deze functie oproepen vanuit de WPF-applicatie.

Voeg een Window met de naam *RekeningInfoRaadplegen* toe aan *AdoWPF* en stel dit Window in als opstartWindow van dit project.

Daarvoor moet je in App.xaml de StartupUri wijzigen.

```

<Application x:Class="AdoWPF.App"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
StartupUri="RekeningInfoRaadplegen.xaml">

```

Voeg volgende controls toe:

Een Label met volgende properties:

- Content Rekening Nr :

Daarnaast een TextBox met volgende properties:

- Name textBoxRekeningNr
- Content Leeg maken

Daaronder een Button met volgende properties

- Name buttonInfo
- Content Info

Daaronder een Label met volgende properties:

- Name labelSaldo
- Content Leeg maken

Daaronder een Label met volgende properties:

- Name labelKlantNaam
- Content Leeg maken

Daaronder een Label met volgende properties:

- Name labelStatus
- Content Leeg maken

Dubbeltklik buttonInfo. Visual Studio maakt een routine en koppelt de routine aan het Click-event van de Button. Schrijf volgende code in de routine:

Voeg bovenaan de code `using AdoGemeenschap;` toe

```
try
{
 var manager = new RekeningenManager();
 var info =
 manager.RekeningInfoRaadplegen(textBoxRekeningNr.Text); (1)
 labelSaldo.Content= info.Saldo.ToString("N"); (2)
 labelKlantNaam.Content = info.Klantnaam; (3)
 labelStatus.Content= String.Empty;
```

```
}
catch (Exception ex) (4)
{
 labelSaldo.Content= String.Empty;
 labelKlantNaam.Content= String.Empty;
 labelStatus.Content= ex.Message;
}
```

- (1) Je roept de functie `RekeningInfoRaadplegen` op en voedert als parameter de inhoud van `textBoxRekeningNr`. Je krijgt een `RekeningInfo`-object terug waarnaar je verwijst met een reference-variabele resultaat van het type `RekeningInfo`.
- (2) Je haalt het saldo uit het `RekeningInfo`-object en je toont dit saldo in `labelSaldo`.
- (3) Je haalt de klantnaam uit het `RekeningInfo`-object en je toont die klantnaam in `labelKlantNaam`.
- (4) Als een fout optreedt in de functie `RekeningInfoRaadplegen` (bijv. de rekening bestaat niet), maak je `labelSaldo` en `labelKlantNaam` leeg en je toont de foutboodschap in `labelStatus`.

Zorg er voor dat in je XML-configuratiebestand de `connectionString` voor `SqlServer` actief staat. Daarna kan je de code uitproberen.



**Maak uit de oefenmap opgave 6**

## 9 AUTONUMBER-VELDEN

### 9.1 Algemeen

In Access, SqlServer en andere databasetypes kan je een kolom definiëren met autonummering (bijv. de kolom KlantNr in de tabel Klanten van de database Bank).

Als je een record toevoegt, vult de database zelf in deze kolom een volgend nummer in.

Het kan gebeuren dat je in je applicatie een record toevoegt en onmiddellijk de waarde van de autonumber kolom wilt weten. Hierbij moet je er rekening mee houden dat meerdere gebruikers tegelijk records kunnen toevoegen en je de waarde van de autonumber kolom wilt kennen van jouw toegevoegd record, en niet de waarde van de autonumber kolom van een record toegevoegd door een andere gebruiker.

In een SqlServer-database, vind je de inhoud van jouw autonumber kolom in de systeemvariabele @@identity. Als je onmiddellijk na je insert statement het statement `select @@identity` uitvoert, geeft dit statement je de inhoud van de autonumber kolom, op voorwaarde dat je tussendoor de connectie niet sluit.

Als voorbeeld zal je een record toevoegen aan de tabel Klanten van de database Bank, en zal je de gebruiker tonen welk klantnummer deze nieuwe klant heeft.

### 9.2 Voorbeeld

In SqlServer kan een Stored Procedure (in tegenstelling tot Access) meerdere SQL-statements bevatten. Je kunt dus zowel het insert statement als het statement `select @@identity` in één Stored Procedure schrijven.

Je scheidt de opdrachten met een puntkomma.

#### 9.2.1 De Stored Procedure:

- Klik met de rechtermuisknop op *Stored Procedures* van de SqlServer-database *Bank* in de *Server Explorer* en kies *Add New Stored Procedure*. Je kunt in een venster de code voor de Stored Procedure intikken.
- Wijzig deze code als volgt:

```
CREATE PROCEDURE NieuweKlant
(
 @Naam nvarchar(50)
)
AS
 insert into klanten(naam) values (@Naam);
 select @@identity
```

- Klik op Update.
- Sluit het ontwerpscherm van de Stored Procedure.



## 9.2.2 De functie NieuweKlant

Je maakt een class *KlantenManager* in het project *AdoGemeenschap*. Deze class zal alle databasebewerkingen bevatten voor het concept Klant van de Bank.

Denk eraan om de nodige namespace te importeren bovenaan de class:

```
using System.Data;
```

Je maakt een functie *NieuweKlant* in de class *KlantenManager* van het project *AdoGemeenschap*. Deze functie zal je daarna kunnen oproepen vanuit de WPF-applicatie:

```
public Int64 NieuweKlant(String naam)
{
 var manager = new BankDbManager();
 using (var conBank = manager.GetConnection())
 {
 using (var comToevoegen = conBank.CreateCommand())
 {
 comToevoegen.CommandType = CommandType.StoredProcedure;
 comToevoegen.CommandText = "NieuweKlant";
 var parNaam = comToevoegen.CreateParameter();
 parNaam.ParameterName = "@Naam";
 parNaam.Value = naam;
 comToevoegen.Parameters.Add(parNaam);

 conBank.Open();
 Int64 klantNr = Convert.ToInt64(comToevoegen.ExecuteScalar()); (1)
 return klantNr;
 } // using comToevoegen
 } // using conBank
}
```

- (1) Je voert de Stored Procedure *NieuweKlant* uit en vraagt de ene waarde die de laatste SQL-statement in deze Stored Procedure ophaalt (het nieuwe klantnummer).

Nu kan je deze functie oproepen vanuit de WPF-applicatie.

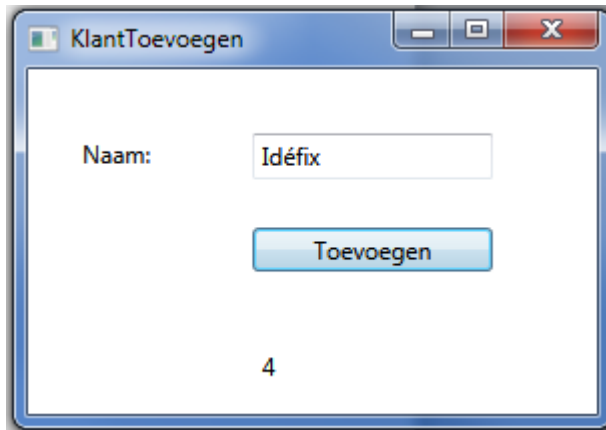
Voeg een Window met de naam *KlantToevoegen* toe aan *AdoWPF* en stel dit Window in als opstartWindow van dit project.

Daarvoor moet je in App.xaml de StartupUri wijzigen.

```
<Application x:Class="AdoWPF.App"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
StartupUri="KlantToevoegen.xaml">
```

Voeg volgende controls toe:



Een Label met volgende properties:

- Content                      Naam

Daarnaast een TextBox met volgende properties:

- Name                        textBoxNaam
- Text                        leegmaken

Daaronder een Button met volgende properties

- Name                        buttonToevoegen
- Content                    Toevoegen

Daaronder een Label met volgende properties:

- Name                        labelStatus
- Content                    Leeg maken

Dubbelklik buttonToevoegen. Visual Studio maakt een routine en koppelt de routine aan het Click-event van de Button. Schrijf volgende code in de routine:

```
...
using AdoGemeenschap
...
try
{
 var manager = new KlantenManager();
 labelStatus.Content = manager.NieuweKlant(textBoxNaam.Text).ToString();
}
catch (Exception ex)
{
 labelStatus.Content = ex.Message;
}
```

- (1) Je voert de functie `NieuweKlant` uit. Voor de parameter naam geef je de inhoud van `textBoxNaam` mee. Je krijgt het klantnummer terug en toont dit in `labelStatus`.
- (2) Als in `KlantToevoegen` een fout optreedt (bijv. naam `Stored Procedure` verkeerd geschreven), vang je deze fout op en je toont de foutboodschap in `labelStatus`.

Je kunt de code uitproberen en een klant toevoegen aan de `SqlServer`-database.



**Maak uit de oefenmap opgave 7**

## 10 DE CLASS DATAREADER

### 10.1 Algemeen

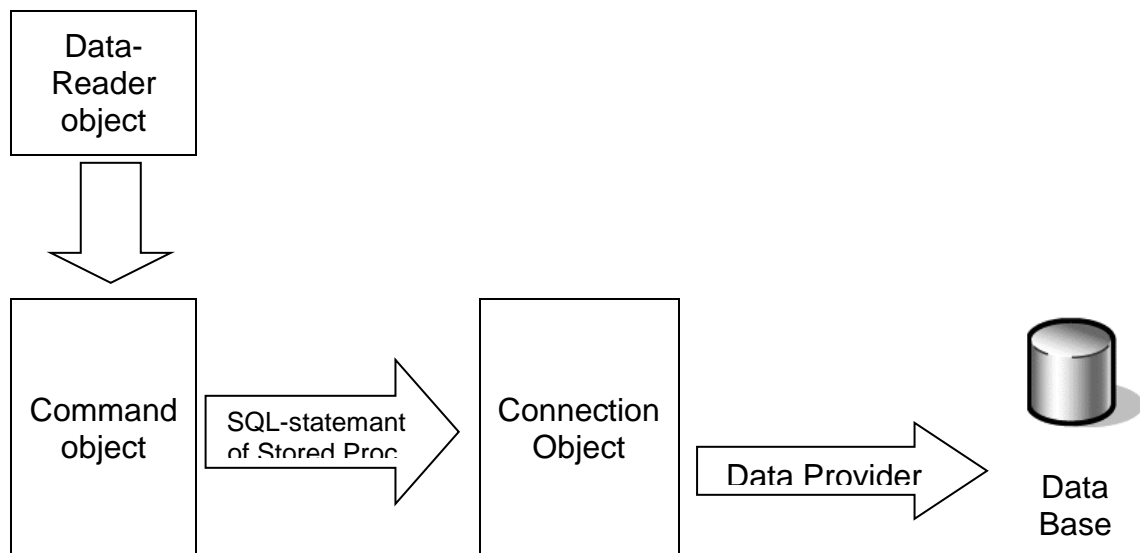
Om uit een SQL-select-statement of een stored procedure data te lezen die meerdere rijen en/of kolommen bevat (bijv. `select brnaam, brouwernr from brouwers order by brnaam` in de database Bieren) gebruik je op het Command-object de method `ExecuteReader()`.

Deze method geeft je een `DataReader`-object.

Met een `DataReader`-object kan je de waarden uit de rijen en kolommen lezen.

Een `DataReader` heeft volgende eigenschappen:

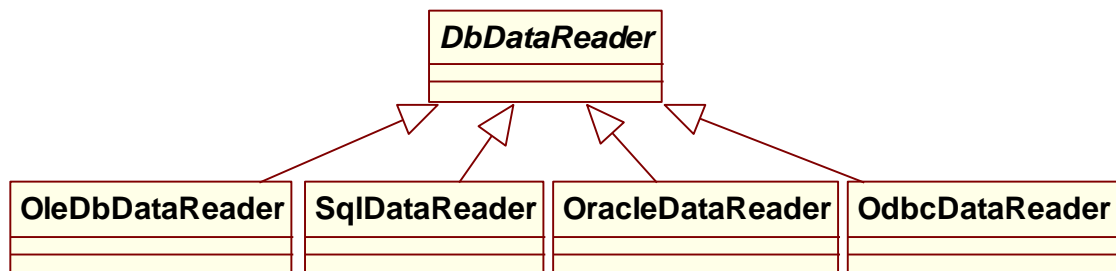
- Hij is forward-only. Je kunt de rijen enkel van voor naar achter lezen. Je kunt niet terugkeren naar vorige rijen.
- Hij is read-only. Je kunt de rijen enkel lezen, maar niet wijzigen.
- De `DataReader` houdt slechts één rij (de rij die je nu leest) in het geheugen. Hij is dus spaarzaam op geheugengebruik.
- De `DataReader` is een performante manier om veel gegevens uit een database te lezen.



Iedere .NET Data Provider heeft zijn eigen class die een `DataReader` voorstelt:

DataProvider	Class
SqlServer	SqlDataReader
OleDb	OleDbDataReader
Oracle	OracleDataReader
Odbc	OdbcDataReader

Al deze classes zijn gelijkaardig: ze erven allen van de class DbDataReader:



Je kunt niet zelf een nieuw DataReader-object aanmaken:

```
var reader = new DbDataReader();
```

 geeft een compilerfout.

Je moet het DataReader-object laten aanmaken door je Command-object met de method ExecuteReader() en naar dit DataReader-object verwijzen met een referentie-variabele:

```
var reader = eenCommand.ExecuteReader();
```

Als de referentievariabele eenCommand verwijst naar een OleDbCommand, krijg je een OleDbDataReader-object onder de gedaante van DbDataReader.

Als de referentievariabele eenCommand verwijst naar een SqlCommand, krijg je een SqlDataReader-object onder de gedaante van DbDataReader.

Je kunt dus terug code schrijven die niet gebonden is aan één type database.

Belangrijkste properties van DbDataReader:

- **HasRows**  
Bevat True als je DataReader minstens één rij bevat, anders False.
- **FieldCount**  
Het aantal kolommen in het SQL-statement.  
Bij het SQL-statement `select brnaam,brouwernr from brouwers order by brnaam` zal de FieldCount-property de waarde 2 bevatten.
- Als je de inhoud van een kolom uit de huidige rij wil opvragen, doe je dat met ofwel `Datareadernaam["kolomnaam"]` ofwel `Datareadernaam[kolomnummer]`. Je krijgt dit als een waarde van het type Object.

## Belangrijkste methods van DbDataReader:

- Read()

Hiermee lees je een volgende rij uit de DataReader. Als er nog een rij te lezen was, krijgt je true terug. Als je Read() uitvoert na de laatste rij, krijg je false terug. Als de DataReader start heb je nog geen rij in het geheugen. Je moet dus ook een Read() uitvoeren om de eerste rij binnen te nemen.

- Close()

Hiermee sluit je de DataReader. Dit is noodzakelijk, want een DataReader heeft het Connection-object exclusief (voor zichzelf alleen) vast tot je de DataReader expliciet sluit met Close(). Als je dit vergeet te doen, kan je op hetzelfde Connection-object geen andere handelingen meer uitvoeren!

Een uitzondering hierop is SqlServer (SqlExpress), versie 2005 of hoger. Deze database bevat een technologie die MARS noemt (Multiple Active Result Sets). Met Mars kan je meerdere DataReaders tegelijk openen en er uit lezen. Om MARS te gebruiken moet je volgende instelling toevoegen aan je connectionString: `MultipleActiveResultSets=true`

- GetString(kolomNr), GetInt32(kolomNr), GetDecimal(kolomNr), ...

Hiermee haal je uit de huidige rij die je leest de inhoud van de kolom op met het volgnummer kolomNr. De kolomnummering begint vanaf nul. In het SQL-statement `select brnaam, brouwnr from brouwers order by brnaam` heeft de kolom `brnaam` als kolomNr nul en de kolom `brouwnr` als kolomNr één. Er bestaat per type gegeven (String, Decimal, ...) een bijbehorende Get...() functie. Om de kolom `brnaam` op te halen gebruik je `GetString(0)`. Om de kolom `brouwnr` op te halen gebruik je `GetInt32(1)`. Deze Get...() functies zijn performanter dan de Item-property (zie hierboven), waarmee je ook de inhoud van een kolom kan opvragen.

- IsDBNull(kolomNr)

Deze functie geeft true terug als de inhoud van de kolom met het volgnummer kolomNr van de huidige rij leeg is (een NULL-waarde). Anders geeft ze false terug. Het is belangrijk deze functie te gebruiken bij kolommen die kunnen leeg zijn, want het uitvoeren van één van de Get...() functies op een lege kolom veroorzaakt een exception.

- **GetOrdinal(kolomNaam)**

Deze functie geeft het kolomNr van de kolom met de naam kolomNaam in het SQL-statement. Bij het SQL-statement `select brnaam,brouwernr from brouwers order by brnaam` geeft `GetOrdinal("brnaam")` de waarde 0 en geeft `GetOrdinal("brouwernr")` de waarde 1.

*Tip: Wanneer de stored procedure of het sql-statement een join is van diverse tabellen en daardoor verschillende kolommen aflevert die dezelfde naam hebben (bijvoorbeeld planten.naam en leveranciers.naam) dan zorg je er best voor dat je deze kolommen een alias-naam meegeeft (vb. select planten.naam as plantennaam,...) zodat de kolomnaam uniek blijft.*

- **GetName(kolomNr)**

Deze functie geeft de kolomnaam van de kolom met het volgnummer kolomNr. Bij het SQL-statement `select brnaam,brouwernr from brouwers order by brnaam` geeft `GetName(0)` de waarde brnaam en geeft `GetName(1)` de waarde brouwernr.

## 10.2 Voorbeeld

In het voorbeeld zal je de records van de table Brouwers in de database bieren inlezen. Om één brouwer op objectgeoriënteerde manier voor te stellen voeg je een class Brouwer toe aan het project AdoGemeenschap:

```
public class Brouwer
{
 private Int32 brouwersNrValue;
 private String brNaamValue;
 private String adresValue;
 private Int16 postcodeValue;
 private String gemeenteValue;
 private Int32? omzetValue;

 public Int32 BrouwerNr
 { get{return brouwersNrValue;}} (1)

 public String BrNaam
 {
 get { return brNaamValue; }
 set { brNaamValue = value; } }

 public String Adres
 {
 get { return adresValue; }
 set { adresValue = value; }
 }

 public Int16 Postcode
 {
 get { return postcodeValue; }
 set
 {
 if (value < 1000 || value > 9999) (2)
 { throw new Exception("Postcode moet tussen 1000 en 9999 liggen"); }
 else
 { }
 }
 }
}
```

```

 { postcodeValue = value; }
 }

 public String Gemeente
 {
 get { return gemeenteValue; }
 set { gemeenteValue = value; }
 }

 public Int32? Omzet (3)
 {
 get { return omzetValue; }
 set {
 if (value.HasValue && Convert.ToInt32(value) < 0)
 { throw new Exception("Omzet moet positief zijn"); } (4)
 else { omzetValue = value; }
 }
 }

 public Brouwer(Int32 brNr, String brNaam, String adres, Int16 postcode,
 String gemeente, Int32? omzet)
 {
 brouwersNrValue = brNr;
 this.BrNaam = brNaam;
 this.Adres = adres;
 this.Postcode = postcode;
 this.Gemeente = gemeente;
 this.Omzet = omzet;
 }
}

```

- (1) De property BrouwerNr hoort bij het autonumber-veld BrouwerNr in de relationele database. Je vult de variabele die bij deze property hoort in bij de constructor. Achteraf is het zinloos dat je dit nummer voor een brouwer-object nog zou kunnen wijzigen, gezien je een autonumber-veld eenmaal toegekend ook niet kan wijzigen. Daarom is deze property readonly.
- (2) Als een ongeldige waarde gestuurd wordt naar de property Postcode, werp je een fout. Deze fout zal in de Windows-applicatie omgezet worden in een foutmelding aan de eindgebruiker.
- (3) De property omzet heeft als type Integer?. De reden is dat de kolom omzet in de table Brouwers van de database niet overal is ingevuld. Een kolom die bij een record niet ingevuld is, bevat de waarde null. In objecten van de class Brouwer kan je hetzelfde voorstellen door een nullable datatype te gebruiken. Als een record in de database een null-waarde bevat in de kolom omzet, zal het overeenkomstige Brouwer-object in het interne geheugen de waarde *null* bevatten.
- (4) Als een ongeldige waarde gestuurd wordt naar de property Omzet, werp je een fout. Deze fout zal in de WPF-applicatie omgezet worden in een foutmelding aan de eindgebruiker.



Als volgende stap voeg je een class `BrouwerManager` toe aan het project `AdoGemeenschap`. Deze class zal alle databasebewerkingen bevatten tussen de table `Brouwers` in de database enerzijds en de class `Brouwer` anderzijds.

Denk eraan om de nodige namespace te importeren bovenaan de class:

```
using System.Data;
```

In deze class `BrouwerManager` maak je een functie die de records leest in de table `Brouwers` van de database `Bieren` waarvan de naam begin met een zoektekst en met deze records een verzameling `Brouwer`-objecten aanmaakt:

Als je de `String` `beginNaam` leeg laat, krijg je alle brouwers.

```
public class BrouwerManager
{
 public List<Brouwer> GetBrouwersBeginNaam(String beginNaam) (1)
 {
 List<Brouwer> brouwers = new List<Brouwer>(); (2)
 var manager = new BierenDbManager();
 using (var conBieren = manager.GetConnection())
 {
 using (var comBrouwers = conBieren.CreateCommand())
 {
 comBrouwers.CommandType = CommandType.Text;
 if (beginNaam != string.Empty)
 {
 comBrouwers.CommandText =
 "select * from Brouwers where
 BrNaam like @zoals order by BrNaam"; (3)
 var parZoals = comBrouwers.CreateParameter();
 parZoals.ParameterName = "@zoals";
 parZoals.Value = beginNaam + "%";
 comBrouwers.Parameters.Add(parZoals);
 }
 else comBrouwers.CommandText = "select * from Brouwers"; (4)
 }
 conBieren.Open();
 using (var rdrBrouwers = comBrouwers.ExecuteReader()) (5)
 {
 Int32 brouwerNrPos = rdrBrouwers.GetOrdinal("BrouwerNr"); (6)
 Int32 brNaamPos = rdrBrouwers.GetOrdinal("BrNaam");
 Int32 adresPos = rdrBrouwers.GetOrdinal("Adres");
 Int32 postcodePos = rdrBrouwers.GetOrdinal("Postcode");
 Int32 gemeentePos = rdrBrouwers.GetOrdinal("Gemeente");
 Int32 omzetPos = rdrBrouwers.GetOrdinal("Omzet");

 Int32? omzet;
 while (rdrBrouwers.Read()) (7)
 {
 if (rdrBrouwers.IsDBNull(omzetPos)) (8)
 { omzet = null; }
 else
 { omzet = rdrBrouwers.GetInt32(omzetPos); }
 brouwers.Add(new Brouwer(rdrBrouwers.GetInt32(brouwerNrPos),
 rdrBrouwers.GetString(brNaamPos),
 rdrBrouwers.GetString(adresPos),
 rdrBrouwers.GetInt16(postcodePos),
 rdrBrouwers.GetString(gemeentePos), omzet));
 (9)
 } // do while
 } // using rdrBrouwers
 } // using comBrouwers
 } // using conBieren
 return brouwers;
}
```

- (1) De functie geeft een variabel aantal Brouwer-objecten terug. De class List is een ideale class om een variabel aantal objecten te bevatten. Je beschrijft dat de functie een List van Brouwer-objecten zal teruggeven aan de oproeper van de functie. Deze oproeper is de WPF-applicatie. De parameter beginNaam geeft het begin aan van de naam van de brouwers die je in de database zoekt.
- (2) Je maakt een lege List van Brouwer-objecten.
- (3) De parameter @zoals bestaat uit de waarde die de functie binnenkrijgt in haar parameter beginNaam met daaraan een % toegevoegd. Als de parameter beginNaam de waarde *de* bevat, zal het SQL-statement zijn:

```
select ... from Brouwers where BrNaam like 'de%' order by BrNaam
```

- (4) Als de string beginNaam leeg is, verkrijg je alle brouwers.
- (5) Je voert het SQL-statement uit met de ExecuteReader()-method van het Command-object. Deze method maakt voor jou een DbDataReader-object. Je wijst met de reference-variabele rdrBrouwers naar dit object. Door dit object aan te maken in een using opdracht, wordt dit DbDataReader automatisch gesloten bij het einde van de using opdracht.
- (6) Je vraagt aan de DataReader het volgnummer van de kolom die in het SQL-statement de naam `BrouwerNr` heeft.
- (7) Je leest record per record uit de DataReader met de Read()-method, tot er geen records meer te lezen zijn. Dan geeft de Read()-method false terug.
- (8) Als de kolom `Omzet` in de database een null-waarde bevat, vul je de nullable variabele `omzet` met *null*, anders vul je deze variabele met de inhoud van de kolom `Omzet`.
- (9) Je maakt een nieuw Brouwer-object.  
Voor de parameter `brouwerNr` van de Brouwer-constructor haal je uit de huidige rij van de DataReader de inhoud van de kolom `BrouwerNr` op met de method `GetInt32()`. De kolom bevat in de database een geheel getal. Zo haal je ook de andere kolominhouden op en gebruikt ze als parameters van de Brouwer constructor.  
Het Brouwer-object dat je aanmaakt voeg je onmiddellijk toe aan de lijst van brouwers met de Add-method.

Kies in het menu *Build* de opdracht *Build AdoGemeenschap* om te controleren of de nieuw toegevoegde code geen tikfouten bevat.

Tot nu heb je met deze code een verzameling Brouwer-objecten opgebouwd aan de hand van data uit een relationele database. Deze verzameling Brouwer-objecten bevindt zich in het interne geheugen. In de volgende hoofdstukken zal je deze verzameling Brouwer-objecten afbeelden aan de eindgebruiker en door hem laten bewerken in WPF-applicaties.

## 11 DATABINDING

### 11.1 Algemeen

Data-binding bestaat uit het verbinden van data in het interne geheugen (bijv. onze verzameling Brouwer-objecten) aan controls op Windows zodat de gebruiker

- de data kan zien in de controls
- de data kan wijzigen door de controls op het Window te wijzigen.

Je kunt iedere property van een control verbinden aan data in het interne geheugen. Een voor de hand liggend voorbeeld is het verbinden van de Text-property van een TextBox met de omzet van een brouwer, zodat de gebruiker deze omzet als een getal in de TextBox ziet. Een minder voor de hand liggend voorbeeld is het verbinden van de Width-property van een TextBox met de omzet van een brouwer, zodat de gebruiker de omzet op een grafische manier kan inschatten.

Er bestaan twee soorten data-binding:

- Simple data binding.  
Een control van een Window toont één enkel datagegeven uit het interne geheugen. Voorbeelden daarvan zijn de TextBox en Label control. In deze controls kan je de omzet van één brouwer tonen, niet de omzet van meerdere brouwers.
- Complex data binding  
Een control van een Window toont meerdere datagegevens uit het interne geheugen. Voorbeelden daarvan zijn de ListBox en de ComboBox. In deze controls kan je de namen van meerdere brouwers tonen.

## 11.2 Een lijst objecten tonen

Je zult de lijst van Brouwer-objecten die de functie GetBrouwersBeginNaam teruggeeft, afbeelden in een Window zoals het voorbeeld hieronder:

BrouwerNr	Naam	Adres	Postcode	Gemeente	Omzet	
1	Achouffe	Route du Village 32	6666	Achouffe-Wibrin	€ 10.000,00	^
2	Alken	Stationstraat 2	3570	Alken	€ 950.000,00	
3	Ambly	Rue Principale 45	6953	Ambly-Nassogne	€ 500,00	
4	Anker	Guido Gezellelaan 49	2800	Mechelen	€ 3.000,00	
6	Artois	Vaartstraat 94	3000	Leuven	€ 4.000.000,00	
8	Bavik	Rijksweg 33	8531	Bavikhove	€ 110.000,00	
9	Belle Vue - Molenbeek	Henegouwenkaai 33	1080	Sint-Jans-Molenbeek		v

Brouwer Nr: 1      Br Naam: Achouffe

Adres: Route du Village 32      Postcode: 6666

Gemeente: Achouffe-Wibrin      Omzet: € 10.000,00

Achouffe  
Alken  
Ambly  
Anker  
Artois

- Het Label met het brouwernummer op de onderste helft van het Window en de TextBoxen zijn voorbeelden van simple data binding.  
Je ziet in deze controls data van één brouwer.
- De ListBox onder de TextBoxen en de DataGrid in het Window zijn voorbeelden van Complex data binding.  
Je ziet in deze controls data van meerdere brouwers.


## 11.3 Het Data Sources Window

De eerste stap is dat je de class Brouwer kenbaar maakt aan het WPF-project als een Data Source (een bron van data uit een relationele database). Visual Studio zal later, aan de hand van de properties beschreven in deze class Brouwer, helpen de controls op het Window op te bouwen.





Je moet je project opnieuw *Build*en om alle objecten zichtbaar te maken, en zorg ervoor dat je zeker op het project *AdoWPF* staat.

Kies in het menu *View* voor *Other Windows* en daarin *Data Sources* om het venster met Data Sources te zien.

Initieel bevat een WPF-project geen Data Sources.

Je voegt een Data Source toe met de knop  in de knoppenbalk van het Data Sources venster of via een klik op de tekst in het venster. Een WPF-project kan één of meerdere Data Sources bevatten.

In het volgend venster beslis je onder welke vorm je WPF-project data binnenkrijgt:

 Database	<p>Je WPF-project krijgt gegevens binnen uit een dataset. Een DataSet is een relationeel databasemodel in het interne geheugen. Meestal vul je een DataSet met gegevens uit een relationele database op de harde schijf (Access, SqlServer, ...)</p> <p>DataSets komen in deze cursus niet aan bod.</p>
 Service	<p>Je WPF-project krijgt gegevens binnen van een Service. Bij een Service roept je WPF-project een functie op van een ander programma (dat al of niet in .NET geschreven is) en krijgt van dat programma data terug in XML formaat.</p> <p>Services komen in deze cursus niet aan bod.</p>
 Object	<p>Je WPF-project krijgt gegevens binnen onder de vorm van één of meerdere objecten van een eigen geschreven class.</p>
 SharePoint	<p>Geeft je een verbinding met een Sharepoint site en laat je Sharepoint-objecten kiezen voor je applicatie.</p> <p>Dit komt verder niet aan bod in de cursus.</p>

Gezien de functies die we gaan gebruiken een verzameling objecten van de eigen geschreven class Brouwer aanbieden, kies je *Object*.

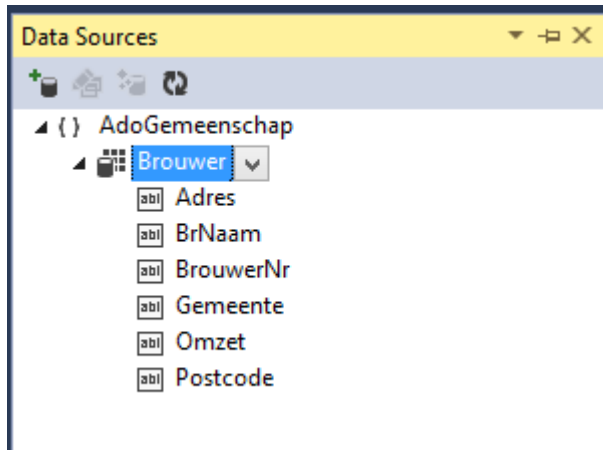
Kies daarna *Next*.

In het volgende venster krijg je de vraag *What objects do you want to bind to?*







Kies het project *AdoGemeenschap*, daarbinnen de namespace *AdoGemeenschap* en daarbinnen de class *Brouwer* (het soort object waarvan de functies *GetAllBrouwers* en *GetBouwersBeginNaam* een verzameling teruggeven).

Kies daarna *Finish*.

Je ziet de class Brouwer als een Data Source:



Mogelijke iconen bij de properties:

-  Tekst
-  Geheel getal
-  Getal met decimalen
-  Boolean (true/false)
-  Datum/tijd
-  Ander type

## 11.4 Een Data Source gebruiken

Als volgende stap kan je de Data Source gebruiken om vlot de lay-out van een Window te maken die met de Data Source ( brouwer-objecten) samenwerkt.

Voeg een Window met de naam *OverzichtBrouwers* toe aan *AdoWPF* en stel dit Window in als opstartWindow van dit project.

Daarvoor moet je in App.xaml de StartupUri wijzigen.

```
<Application x:Class="AdoWPF.App"
 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 StartupUri="OverzichtBrouwers.xaml">
```

Sleep vanuit het *Data Sources* venster de Data Source *Brouwer* op dit nieuwe Window.

Het volgende gebeurt in het Window:

Default worden de *CollectionViewSource* *brouwerViewSource* en de *DataGrid* *brouwerDataGrid* aan het Window toegevoegd

De *brouwerViewSource* dient als een binding source voor het buitenste Grid element. De binding wordt ingesteld door de parent Grid-property *DataContext* als `{StaticResource brouwerViewSource}` in te vullen. De binnenste Grid elementen erven die *DataContext*-waarde van de parent Grid. (de *brouwerDataGrid* *ItemSource*-property is ingesteld als `{Binding}`)

Het slepen van de Data Source op het WPF-Window levert de volgende xaml code op. (je hoeft dit niet zelf in te tikken, alles gebeurt automatisch tijdens het slepen van de DataSource op het Window).

```
<Window
 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
 xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
 xmlns:local="clr-namespace:AdoWPF"
 xmlns:AdoGemeenschap="clr-namespace:AdoGemeenschap;assembly=AdoGemeenschap"
 mc:Ignorable="d" x:Class="AdoWPF.OverzichtBrouwers"
 Title="OverzichtBrouwers" Height="300" Width="300" Loaded="Window_Loaded">
 <Window.Resources>
 <CollectionViewSource x:Key="brouwerViewSource"
 d:DesignSource="{d:DesignInstance {x:Type AdoGemeenschap:Brouwer},
 CreateList=True}" />
 </Window.Resources>

 <Grid DataContext="{StaticResource brouwerViewSource}">
 <DataGrid x:Name="brouwerDataGrid"
 RowDetailsVisibilityMode="VisibleWhenSelected" Margin="95,107,-203,-37"
 ItemsSource="{Binding}" EnableRowVirtualization="True"
 AutoGenerateColumns="False">
 <DataGrid.Columns>
 <DataGridTextColumn x:Name="adresColumn" Width="SizeToHeader"
 Header="Adres" Binding="{Binding Adres}" />
 <DataGridTextColumn x:Name="brNaamColumn" Width="SizeToHeader"
 Header="Br Naam" Binding="{Binding BrNaam}" />
 <DataGridTextColumn x:Name="brouwerNrColumn"
 Width="SizeToHeader"
 IsReadOnly="True" Header="Brouwer Nr"
 Binding="{Binding BrouwerNr}" />
 <DataGridTextColumn x:Name="gemeenteColumn" Width="SizeToHeader"
 Header="Gemeente" Binding="{Binding Gemeente}" />
 <DataGridTextColumn x:Name="omzetColumn" Width="SizeToHeader"
 Header="Omzet" Binding="{Binding Omzet}" />
 <DataGridTextColumn x:Name="postcodeColumn" Width="SizeToHeader"
 Header="Postcode" Binding="{Binding Postcode}" />
 </DataGrid.Columns>
 </DataGrid>
 </Grid>
</Window>
```

### 11.4.1 Lay-out

Laten we eerst de lay-out wat aanpassen, zodat we onze datagrid helemaal kunnen bekijken.

We werken met een DockPanel (zie ook in de cursus WPF)  
Pas OverzichtBrouwers.xaml aan zoals hieronder:

```
<Window
 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
 xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
 xmlns:AdoGemeenschap="clr-namespace:AdoGemeenschap;assembly=AdoGemeenschap"
 mc:Ignorable="d" x:Class="AdoWPF.OverzichtBrouwers"
 Title="OverzichtBrouwers" SizeToContent="Width" Loaded="Window_Loaded"> (1)
 <Window.Resources>
 <CollectionViewSource x:Key="brouwerViewSource"
d:DesignSource="{d:DesignInstance {x:Type AdoGemeenschap:Brouwer},
CreateList=True}"/>
 </Window.Resources>
 <DockPanel LastChildFill="True"> (2)
 <Border Height="30" Background="SkyBlue" BorderBrush="Black"
BorderThickness="1" DockPanel.Dock="Top"> (3)
 <StackPanel HorizontalAlignment="Center" Margin="0" Width="410"
Orientation="Horizontal">
 <TextBox HorizontalAlignment="Left" Name="textBoxZoeken"
Width="120" AcceptsReturn="False" />
 <Button Content="Zoeken" HorizontalAlignment="Left"
Name="buttonZoeken" Width="Auto"
Background="BlanchedAlmond" FontWeight="Bold" />
 </StackPanel>
 </Border>
 <Border DataContext="{StaticResource brouwerViewSource}" Background="SkyBlue"
BorderBrush="Black" BorderThickness="1">
 <DataGrid x:Name="brouwerDataGrid" (4)
RowDetailsVisibilityMode="VisibleWhenSelected" ItemsSource="{Binding}"
EnableRowVirtualization="True" AutoGenerateColumns="False">
 <DataGrid.Columns>
 <DataGridTextColumn x:Name="brouwerNrColumn" Header="BrouwerNr"
IsReadOnly="True" Binding="{Binding BrouwerNr}"/>
 <DataGridTextColumn x:Name="brNaamColumn" Header="Naam"
Binding="{Binding BrNaam}"/>
 <DataGridTextColumn x:Name="adresColumn" Header="Adres"
Binding="{Binding Adres}"/>
 <DataGridTextColumn x:Name="postcodeColumn" Header="Postcode"
Binding="{Binding Postcode}"/>
 <DataGridTextColumn x:Name="gemeenteColumn" Header="Gemeente"
Binding="{Binding Gemeente}"/>
 <DataGridTextColumn x:Name="omzetColumn" Header="Omzet"
Binding="{Binding Omzet}"/>
 </DataGrid.Columns>
 </DataGrid>
 </Border>
 </DockPanel>
</Window>
```

- (1) `SizeToContent="Width"`. De grootte van het Window past zich aan aan de breedste inhoud. In dit geval is dat de DataGrid.



- (2) `LastChildFill="True"` Bij een DockPanel wordt in dit geval het laatste onderdeel gebruikt om de rest van het panel op te vullen. Als alle panels boven, onderaan, links en rechts hun ruimte gebruiken, wordt de overgebleven ruimte gebruikt door het laatste child van het DockPanel. We gebruiken hiervoor de DataGrid.
- (3) Laat bovenaan wat ruimte voor controls die we later gaan toevoegen. Deze ruimte staat bovenaan doordat we `DockPanel.Dock="Top"` meegeven. In die ruimte plaatsen we een StackPanel (zie WPF cursus) met een TextBox en een Button. (wordt straks gebruikt)  
Door dit StackPanel bovenaan te plaatsen, vult de DataGrid nu de resterende ruimte op als Last Child.
- (4) Overtollige attributen zijn gewist, en de kolommen op hun plaats gezet

### 11.4.2 Alle brouwers

Als je nu de applicatie start, blijft de DataGrid initieel leeg. We moeten de collectie brouwers, die we terugkrijgen van onze functie `GetBrouwersBeginNaam`, koppelen aan de `CollectionViewSource`.

In `OverzichtBrouwers.xaml.cs` voegen we het volgende toe:

Bovenaan in de code komt `using AdoGemeenschap;`

In het `Window_Loaded`-event staat reeds automatisch:

```
System.Windows.Data.CollectionViewSource brouwerViewSource =
((System.Windows.Data.CollectionViewSource) (this.FindResource("brouwerViewSource"))
);
```

Vervolledig en wijzig de code in:

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
 CollectionViewSource brouwerViewSource =
 ((CollectionViewSource) (this.FindResource("brouwerViewSource"))); (1)
 var manager = new BrouwerManager(); (2)
 List<Brouwer> brouwersOb = new List<Brouwer>();
 brouwersOb = manager.GetBrouwersBeginNaam(textBoxZoeken.Text); (3)
 brouwerViewSource.Source = brouwersOb; (4)
}
```

- (1) `Using System.Windows.Data` staat reeds bovenaan in de code, dus `System.Windows.Data` hoef je niet te herhalen
- (2) We maken een nieuwe `BrouwerManager`-instantie
- (3) We steken de lijst die we terugkrijgen van `GetBrouwersBeginNaam` in `brouwersOb`
- (4) We definiëren de lijst `brouwersOb` als source. De textbox `textBoxZoeken` is nog leeg, dus krijgen we alle brouwers terug.

Voer de applicatie eens uit. De brouwers worden nu ingevuld bij het opstarten.

### 11.4.3 Brouwers zoeken

We kunnen ook enkel de brouwers opvragen die beginnen met één of meerdere letters die we intikken in een textbox.

De TextBox (name: textBoxZoeken) en de Button (name: buttonZoeken, Content: Zoeken) hebben we reeds toegevoegd aan het WPF-Window. (zie 11.4.1)

Je zou de code van de *Loaded* kunnen herhalen voor de *Button-Click*, maar herhaalde code kan je beter afzonderen in een aparte Method, dus:

```
public List<Brouwer> brouwersOb = new List<Brouwer>(); (1)

private void VulDeGrid()
{
 CollectionViewSource brouwerViewSource =
 (CollectionViewSource) (this.FindResource("brouwerViewSource"));
 var manager = new BrouwerManager();
 brouwersOb = manager.GetBrouwersBeginNaam(textBoxZoeken.Text);
 brouwerViewSource.Source = brouwersOb;
}
```

(1) We maken de *List* algemeen zodat ze ook voor de andere code beschikbaar wordt.

Als textBoxZoeken leeg is krijgen we alle brouwers. Als er een string instaat krijgen we alle brouwers die beginnen met die string.

Natuurlijk worden Window\_loaded en buttonZoeken\_Click aangepast:

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
 VulDeGrid();
 textBoxZoeken.Focus(); (1)
}

private void buttonZoeken_Click(object sender, RoutedEventArgs e)
{
 VulDeGrid();
}
```

Probeer de applicatie nog eens uit. Tik een letter in de textbox en klik op Zoeken.

Een alternatief voor het klikken op de Zoek button is het drukken op de Enter toets als je in de *TextBoxZoeken* staat.

```
private void textBoxZoeken_KeyUp(object sender, KeyEventArgs e)
{
 if (e.Key == Key.Enter) (1)
 {
 VulDeGrid(); (2)
 }
}
```

(1) Als de gebruiker een toets indrukt in de TextBox en weer loslaat, vuurt deze TextBox het KeyUp-event af. We controleren of de gebruiker op Enter heeft gedrukt. **(vergeet niet de evenhandler in te vullen bij de events!!)**

- (2) In dat geval roep je de method VulDeGrid op, die je verder in de code vindt.
- (3) Als de gebruiker geen beginletters intikt van de naam van de te zoeken brouwers, haalt de method GetBrouwersBeginNaam van de class BrouwerManager alle brouwers uit de relationele database op. Als er veel brouwers zijn en veel gelijktijdige gebruikers van de applicatie is dit nadelig voor de performantie. Je kunt dit eventueel vermijden door te controleren of textBoxZoeken.text leeg is, zodat je in dat geval een foutmelding kunt geven.

Je kunt het programma uitproberen.



**Maak uit de oefenmap opgave 8**

### 11.4.4 De CollectionViewSource control

Als je met een database werkt, is het interessant om te kunnen navigeren naar het volgende, vorige, eerste en laatste record.

De CollectionViewSource is dus een control die bijhoudt welke data je in het Window toont.

De View van een CollectionViewSource bevat volgende methods waarmee je als programmeur de data van de Data Source kan manipuleren.

Methods:

- MoveCurrentToFirst  
Het eerste object uit de verzameling selecteren.  
In ons geval: de eerste brouwer selecteren.
- MoveCurrentToLast  
Het laatste object uit de verzameling selecteren.  
In ons geval: de laatste brouwer selecteren.
- MoveCurrentToNext  
Het volgende object uit de verzameling selecteren.  
In ons geval: de volgende brouwer selecteren.
- MoveCurrentToPrevious  
Het vorige object uit de verzameling selecteren.  
In ons geval: de vorige brouwer selecteren.

We voegen eerst enkele controls toe aan het Window en plaatsen die in hetzelfde stackpanel als de textbox waarmee we data zoeken.

```
<Border Height="30" Background="SkyBlue" BorderBrush="Black"
 BorderThickness="1" DockPanel.Dock="Top">
 <StackPanel HorizontalAlignment="Center" Margin="0,0,0,0"
 Name="stackPanel1" Width="550" Orientation="Horizontal">
 <Button Content="|<<" Name="goToFirstButton"
 Width="Auto" IsEnabled="False"
 FontWeight="Bold" Background="BlanchedAlmond"
 Click="goToFirstButton_Click"></Button>
 <Button Content="<" Name="goToPreviousButton"
 Width="Auto" IsEnabled="False"
 FontWeight="Bold" Background="BlanchedAlmond"
 Click="goToPreviousButton_Click"></Button>
 <Button Content=">" Name="goToNextButton" Width="Auto"
 FontWeight="Bold" Background="BlanchedAlmond"
 Click="goToNextButton_Click"></Button>
 <Button Content=">>" Name="goToLastButton"
 Width="Auto"
 FontWeight="Bold" Background="BlanchedAlmond"
 Click="goToLastButton_Click"></Button>
 <TextBox HorizontalAlignment="Left" Name="textBoxZoeken"
 Width="120"
 AcceptsReturn="False" KeyUp="textBoxZoeken_KeyUp" />
 <Button Content="Zoeken" HorizontalAlignment="Left"
 Name="buttonZoeken" Width="Auto"
 Background="BlanchedAlmond"
 FontWeight="Bold" Click="buttonZoeken_Click" />
 </StackPanel>
</Border>
```

Bij alle buttons vullen we het Click-event in.

```
public partial class OverzichtBrouwers : Window
{
 private CollectionViewSource brouwerViewSource;
 public OverzichtBrouwers()
 {
 InitializeComponent();
 }
 ...
 private void goToFirstButton_Click(object sender, RoutedEventArgs e)
 {
 brouwerViewSource.View.MoveCurrentToFirst();
 goUpdate();
 }
 private void goToPreviousButton_Click(object sender, RoutedEventArgs e)
 {
 brouwerViewSource.View.MoveCurrentToPrevious();
 goUpdate();
 }
 private void goToNextButton_Click(object sender, RoutedEventArgs e)
 {
 brouwerViewSource.View.MoveCurrentToNext();
 goUpdate();
 }
 private void goToLastButton_Click(object sender, RoutedEventArgs e)
 {
 brouwerViewSource.View.MoveCurrentToLast();
 goUpdate();
 }

 private void goUpdate()
 {
 goToPreviousButton.IsEnabled = !(brouwerViewSource.View.CurrentPosition == 0);

 goToFirstButton.IsEnabled = !(brouwerViewSource.View.CurrentPosition == 0);

 goToNextButton.IsEnabled =
 !(brouwerViewSource.View.CurrentPosition == brouwerDataGrid.Items.Count - 1);

 goToLastButton.IsEnabled =
 !(brouwerViewSource.View.CurrentPosition == brouwerDataGrid.Items.Count - 1);

 if (brouwerDataGrid.Items.Count != 0)
 {
 if (brouwerDataGrid.SelectedItem != null)
 brouwerDataGrid.ScrollIntoView(brouwerDataGrid.SelectedItem);
 }
 }
}
```

- (1) Je moet eerst de declaratie van je `brouwerViewSource` bovenaan in je klasse `OverzichtBrouwers` plaatsen, anders wordt die niet overal herkend.  
**De declaratie moet dan verwijderd worden in `VulDeGrid`.**
- (2) In `goUpdate()` kijken we of we het laatste of eerste record bereikt hebben tijdens het navigeren, zodat de nodige buttons ingeschakeld zijn en de niet nodige buttons uitgeschakeld worden.
- (3) Met `brouwerDataGrid.ScrollIntoView(brouwerDataGrid.SelectedItem)` zorg je ervoor dat het geselecteerde record altijd zichtbaar is op het Window. Maar we controleren wel even of er een brouwer geselecteerd is. Want bij het inladen van de brouwers is er niet onmiddellijk een brouwer geselecteerd. Er wordt ook nog gecontroleerd of er wel degelijk brouwers zijn.

**Voeg `goUpdate()` ook toe aan `VulDeGrid()`** zodat de juiste buttons ingeschakeld zijn bij elke mogelijke oproep van `VulDeGrid()`

#### 11.4.5 Aantal rijen

Bovenaan het WPF-Window komt er een label waarin het totaal aantal rijen getoond wordt.

Voeg een label `totalRowCount` toe:

```
...
<Button Content=">>|" Name="goToLastButton" Width="Auto"
 FontWeight="Bold" Background="BlanchedAlmond"
 Click="goToLastButton_Click"></Button>
<Label Height="28" Name="labelTotalRowCount" Width="38" />
<TextBox HorizontalAlignment="Left" Name="textBoxZoeken" Width="120"
 AcceptsReturn="False" KeyUp="textBoxZoeken_KeyUp" />
```

Daarvoor is onderstaande wijziging nodig onderaan in `VulDeGrid()`:

```
private void VulDeGrid()
{
 . . .
 labelTotalRowCount.Content = brouwerDataGrid.Items.Count;
}
```

We voegen nu ook nog een Textbox (`textBoxGo`) toe waarin we een getal `x` kunnen ingeven om te navigeren naar het `xde` record. In dat veld zien we ook altijd de positie van het huidig geselecteerde record. Er is ook nog een button `go` om te navigeren naar dat `xde` record als we dat hebben ingetikt. Het label met Content “van” komt tussen `textBoxGo` en het `labelTotalRowCount`. (voorbeeld 4 van 45)

```
<Button Content=">>|" Name="goToLastButton" Width="Auto"
 FontWeight="Bold" Background="BlanchedAlmond"
 Click="goToLastButton_Click"></Button>
<TextBox Name="textBoxGo" Width="37"
 HorizontalContentAlignment="Center"
 VerticalContentAlignment="Center" />
```

```
<Label Content="van " Height="28" Name="label1" Width="30" />
<Label Height="28" Name="labelTotalRowCount" Width="38" />
<Button Content="Go!" Name="goButton" Width="Auto"
 Background="BlanchedAlmond" FontWeight="Bold"
 Click="goButton_Click"></Button>
<TextBox HorizontalAlignment="Left" Name="textBoxZoeken" Width="120"
 AcceptsReturn="False" KeyUp="textBoxZoeken_KeyUp" />
```

En de volgende code wordt toegevoegd na dubbelklikken op goButton:

```
private void goButton_Click(object sender, RoutedEventArgs e)
{
 int position;
 int.TryParse(textBoxGo.Text, out position);
 if (position > 0 && position <= brouwerDataGrid.Items.Count) (1)
 {
 brouwerViewSource.View.MoveCurrentToPosition(position - 1);
 }
 else
 {
 MessageBox.Show("The input index is not valid.");
 }
 goUpdate();
}
```

- (1) Als alle brouwers getoond worden, kunnen we hier werken met TotalRowCount, maar soms tonen we slechts een gedeelte van de brouwers en dan moeten we rekening houden met de getoonde brouwers in de Datagrid.

textBoxGo moet nog worden opgevuld met de huidige positie  
Onderaan in goUpdate() voeg je daarom het volgende toe:

```
textBoxGo.Text = (brouwerViewSource.View.CurrentPosition + 1).ToString();
```

Die huidige positie moet ook telkens wijzigen als het geselecteerde record in de datagrid wijzigt.


In het SelectionChanged-event van de DataGrid roepen we ook goUpdate() op. Klik daarom dubbel op de DataGrid in het xaml-bestand en zet daarin het volgende:

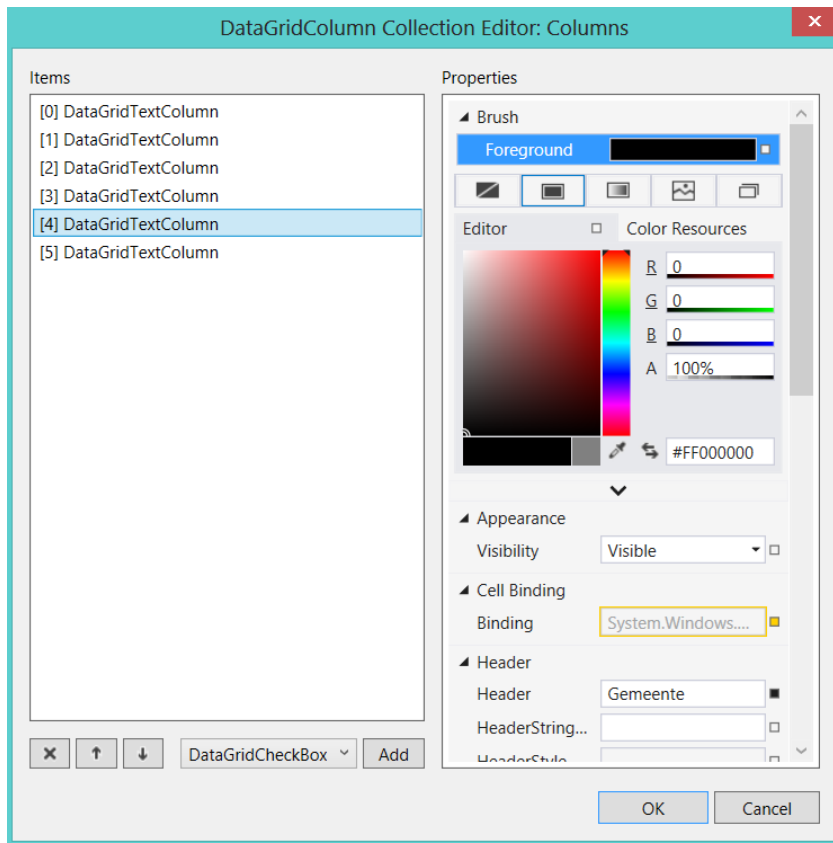
```
private void brouwerDataGrid_SelectionChanged(object sender,
 SelectionChangedEventArgs e)
{
 goUpdate();
}
```

## 11.5 De DataGrid control

De DataGrid control bevat rijen en kolommen waarin de gebruiker data ziet. Met de volgende properties kan je deze control personaliseren:

- CanUserAddRows  
Kan de gebruiker nieuwe rijen (brouwers) toevoegen?
- CanUserDeleteRows  
Kan de gebruiker rijen (brouwers) verwijderen?
- CanUserSortColumns  
Kan de gebruiker de kolommen verplaatsen?

- **CanUserResizeColumns**  
Kan de gebruiker kolombreedtes aanpassen?
- **CanUserResizeRows**  
Kan de gebruiker rijhoogten aanpassen?
- **AlternatingRowBackground**  
Welke kleur krijgen de even rijen?  
De oneven rijen krijgen de kleur van de RowBackground
- De kolombreedte kan je instellen in de property ColumnWidth.  
SizeToHeader maakt je kolommen zo breed als de titels.  
SizeToCells geeft de kolommen de breedte van de inhoud.  
Auto combineert de twee eerste.  
Pixel laat je toe om een aantal pixels op te geven.
- **Columns**  
Hier kan je via de knop  de kolommen personaliseren.



In de dropdown onderaan, kies je het kolomtype en klik je op Add. Rechts krijg je dan alle bijhorende properties.



## 11.6 Simple data binding

Op de onderste helft van het venster zal je nu simple data binding toepassen. Je toont de gegevens van één brouwer, via Labels en TextBoxen.

Opmerking: op één WPF-Window kan je complex data binding gebruiken, simple data binding gebruiken of complex en simple data binding combineren.

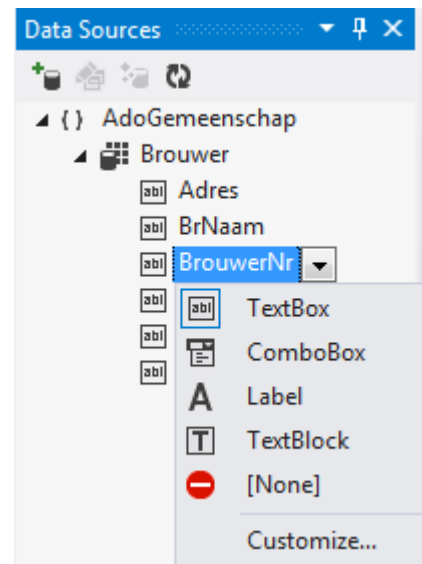
Eerst gaan we de ruimte voorzien op het WPF-Window. In OverzichtBrouwers.xaml voeg je de volgende code toe:

(Plaats deze code voor de border van de datagrid, want in het DockPanel staat LastChildFill="true". Dan vult de datagrid de overblijvende ruimte op.)

```
<Border Height="120" Background="SkyBlue" BorderBrush="Black"
 BorderThickness="1"
 DockPanel.Dock="Bottom" >
 <StackPanel Orientation="Horizontal" HorizontalAlignment="Center"
 Margin="4"
 Height="120">
 </StackPanel>
</Border>
```

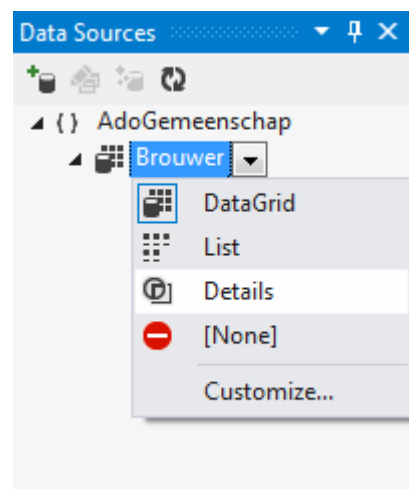
In het Data Sources venster kan je per property van het Brouwer-object bepalen welke control zal gebruikt worden om die property via simple data binding te tonen op het Window (zie rechts).

Standaard stelt Visual Studio voor om TextBoxen te gebruiken, behalve bij ReadOnly properties, waar Visual Studio voorstelt om Labels te gebruiken.



Bij de Brouwer zelf bepaal je of je een DataGridView zal gebruiken om brouwers af te beelden, of simple data binding (Details).

- Kies Details.
- Sleep Brouwer van het venster Data Sources naar de ruimte die we hiervoor voorzien hebben.



Je ziet per property van de Brouwer een Label en een Control.

We gaan die wat herschikken zodat de controls in de ruimte passen.

Wijzig de code zoals hieronder. We geven de grid 4 kolommen en 3 rijen en geven de controls de juiste plaats in die grid.

```
<Grid DataContext="{StaticResource brouwerViewSource}"
 HorizontalAlignment="Left" Name="gridDetail" VerticalAlignment="Top">
 <Grid.ColumnDefinitions>
 <ColumnDefinition Width="Auto" />
 <ColumnDefinition Width="Auto" />
 <ColumnDefinition Width="Auto" />
 <ColumnDefinition Width="Auto" />
 </Grid.ColumnDefinitions>
 <Grid.RowDefinitions>
 <RowDefinition Height="Auto" />
 <RowDefinition Height="Auto" />
 <RowDefinition Height="Auto" />
 </Grid.RowDefinitions>
 <Label Content="Brouwer Nr:" Grid.Column="0"
 Grid.Row="0" HorizontalAlignment="Left" Margin="3"
 VerticalAlignment="Center" />
 <Label Content="{Binding Path=BrouwerNr}" Grid.Column="1"
 Grid.Row="0" Height="28" HorizontalAlignment="Left"
 Margin="3" Name="brouwerNrLabel" VerticalAlignment="Center" />
 <Label Content="Br Naam:" Grid.Column="2" Grid.Row="0"
 HorizontalAlignment="Left" Margin="3"
 VerticalAlignment="Center" />
 <TextBox Grid.Column="3" Grid.Row="0" Height="23"
 HorizontalAlignment="Left" Margin="3" Name="brNaamTextBox"
 Text="{Binding Path=BrNaam, Mode=TwoWay,
 ValidatesOnExceptions=true, NotifyOnValidationError=true}"
 VerticalAlignment="Center" Width="120" />
 <Label Content="Adres:" Grid.Column="0" Grid.Row="1"
 HorizontalAlignment="Left" Margin="3"
 VerticalAlignment="Center" />
 <TextBox Grid.Column="1" Grid.Row="1" Height="23"
 HorizontalAlignment="Left" Margin="3" Name="adresTextBox"
 Text="{Binding Path=Adres, Mode=TwoWay,
 ValidatesOnExceptions=true, NotifyOnValidationError=true}"
 VerticalAlignment="Center" Width="120" />
 <Label Content="Postcode:" Grid.Column="2" Grid.Row="1"
 HorizontalAlignment="Left" Margin="3"
 VerticalAlignment="Center" />
 <TextBox Grid.Column="3" Grid.Row="1" Height="23"
 HorizontalAlignment="Left" Margin="3" Name="postcodeTextBox"
 Text="{Binding Path=Postcode, Mode=TwoWay,
 ValidatesOnExceptions=true, NotifyOnValidationError=true}"
 VerticalAlignment="Center" Width="120" />
 <Label Content="Gemeente:" Grid.Column="0" Grid.Row="2"
 HorizontalAlignment="Left" Margin="3"
 VerticalAlignment="Center" />
 <TextBox Grid.Column="1" Grid.Row="2" Height="23"
 HorizontalAlignment="Left" Margin="3" Name="gemeenteTextBox"
 Text="{Binding Path=Gemeente, Mode=TwoWay,
 ValidatesOnExceptions=true, NotifyOnValidationError=true}"
 VerticalAlignment="Center" Width="120" />
 <Label Content="Omzet:" Grid.Column="2" Grid.Row="2"
 HorizontalAlignment="Left" Margin="3"
 VerticalAlignment="Center" />
 <TextBox Grid.Column="3" Grid.Row="2" Height="23"
 HorizontalAlignment="Left" Margin="3" Name="omzetTextBox"
 Text="{Binding Path=Omzet, Mode=TwoWay,
 ValidatesOnExceptions=true, NotifyOnValidationError=true}"
 VerticalAlignment="Center" Width="120" />
```

```
</Grid>
```

Bekijk het resultaat.

Bij alle textboxes staat er in de automatisch gegenereerde code `Mode=TwoWay`. We voegen in de TextBoxen nog het attribuut `UpdateSourceTrigger=PropertyChanged` aan toe, zodanig dat als je iets wijzigt in de TextBox, dit ook wijzigt in de DataGridView.

Als je ook wilt dat bij het wijzigen in de DataGridView de inhoud van de TextBoxen automatisch mee wijzigt, dan moet je in de Binding hetzelfde attribuut bijvoegen bv.

```
<DataGridViewTextColumn x:Name="brNaamColumn" Header="Naam" Binding="{Binding BrNaam, UpdateSourceTrigger=PropertyChanged}"/>
```

Probeer maar uit.

## 11.7 Een control manueel verbinden met de BindingSource.

Naast de TextBoxen, zal je de brouwers als voorbeeld nog eens tonen in een ListBox.

Eerst voegen we bovenaan in het dockpanel onderstaande tags toe. De datagrid blijft als laatste staan in de code, want die moet de overblijvende ruimte opvullen. Het eerste panel met `DockPanel.Dock="Bottom"` in de code komt onderaan te staan. Het volgende panel met `DockPanel.Dock="Bottom"` komt erboven. Dus we plaatsen ons nieuwe Panel boven de Border van de Grid met de labels en textboxes.

```
<Border Height="120" Background="SkyBlue" BorderBrush="Black"
 BorderThickness="1" DockPanel.Dock="Bottom"
 DataContext="{StaticResource brouwerViewSource}">
 <StackPanel Orientation="Horizontal" HorizontalAlignment="Center"
 Margin="4" Height="120">
 <ListBox Height="100" HorizontalAlignment="Left" Margin="0,0,0,0"
 Name="listBoxBrouwers" VerticalAlignment="Top" Width="120"
 ItemsSource="{Binding}" DisplayMemberPath="BrNaam"/>
 </StackPanel>
</Border>
```

```
DataContext="{StaticResource brouwerViewSource}"
```

maakt dat ons DockPanel verbonden is met de brouwerViewSource

Nu volstaat het om in de Listbox als `ItemSource {Binding}` mee te geven. Daar we geen specifieke Binding opgeven, wordt de Binding van de parent overgenomen en dat is dus het DockPanel of m.a.w. de brouwerViewSource.

Nu moeten we enkel nog een bepaald veld opgeven, want een listBox kan maar één veld tonen en dat is hier dus de BrNaam.

```
DisplayMemberPath="BrNaam"
```

Als je nu het programma uitvoert, zie je dat de ListBox synchroon navigeert door dezelfde brouwers als de DataGrid.

We willen natuurlijk het geselecteerde item zien in de listBox, daar gaan we analoog met de DataGrid werken. Voeg volgende code toe aan goUpdate():

```
private void goUpdate()
{
 ...
 if (brouwerDataGrid.Items.Count != 0)
 {
 if (brouwerDataGrid.SelectedItem != null)
 {
 brouwerDataGrid.ScrollIntoView(brouwerDataGrid.SelectedItem);
 listBoxBrouwers.ScrollIntoView(brouwerDataGrid.SelectedItem);
 }
 }
 ...
}
```

## 11.8 Validation

Het data-binding-model laat je toe om ValidationRules te associëren met je binding-objecten.

Je kunt gebruik maken van de ingebouwde ExceptionValidationRule of je kunt een custom rule maken door die af te leiden van de ValidationRule class en de Validate-methode te implementeren.

De binding engine checkt elke ValidationRule die verbonden is met de binding telkens een inputwaarde naar de binding source wordt gestuurd. Indien dit fout gaat, wordt deze bijvoorbeeld bij een *TextBox* rood omkaderd.

We gaan verkeerde postcode-ingaves tonen als voorbeeld. Verwijder daarom de controle van de waarde uit de set van de Postcode-property in de Brouwer-klasse.

Die property wordt nu dus:

```
public Int16 Postcode
{
 get { return postcodeValue; }
 set { postcodeValue = value; }
}
```

Maak in je AdoWPF-project een class PostcodeRangeRule.cs

Voeg eerst bovenaan de code het volgende toe:

```
using System.Windows.Controls; (1)
using System.Globalization; (2)
```

- (1) Is nodig voor de ValidationRule class die we implementeren in onze class
- (2) Is nodig voor de CultureInfo in het ValidationResult

De PostcodeRangeRule class ziet er als volgt uit:

```
public class PostcodeRangeRule : ValidationRule
{
 public override ValidationResult Validate(object value,
 CultureInfo cultureInfo)
 {
 int postcode = 0;
 if (!int.TryParse(value.ToString(), out postcode)) (1)
 {
 return new ValidationResult(false,
 "Gelieve een getal in te geven"); (2)
 }
 if ((postcode < 1000) || (postcode > 9999))
 {
 return new ValidationResult(false,
 "De postcode moet tussen 1000 en 9999 liggen"); (3)
 }
 else
 {
 return ValidationResult.ValidResult; (4)
 }
 }
}
```

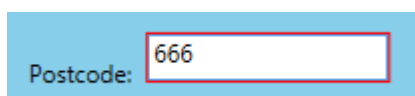
- (1) We proberen de binnengekomen waarde te casten naar een int.
- (2) Als dit mislukt, kunnen we van de waarde geen getal maken, en melden we dit ook
- (3) Als de waarde buiten de grenzen valt, geven we dat door.
- (4) Als alles klopt, dan geven we een "Valid" waarde terug.

*Build* het project om deze nieuwe klasse te herkennen in de XAML-code.

### 11.8.1 Validation bij Simple Data Binding

Het toevoegen van de *ValidationRule* voor de *TextBox* gebeurt door de *Text*-property af te zonderen van de andere properties zodat je extra attributen kan definiëren zoals de *Binding* en binnen de *Binding* de *ValidationRules* van deze *Binding*:

```
<TextBox Grid.Column="3" Grid.Row="1" Height="23"
 HorizontalAlignment="Left" Margin="0,0,3,15" x:Name="postcodeTextBox"
 VerticalAlignment="Center" Width="120">
 <TextBox.Text>
 <Binding Path="Postcode" Mode="TwoWay" UpdateSourceTrigger="PropertyChanged"
 ValidatesOnExceptions="true" NotifyOnValidationError="true">
 <Binding.ValidationRules>
 <local:PostcodeRangeRule/>
 </Binding.ValidationRules>
 </Binding>
 </TextBox.Text>
</TextBox>
```



Als je niet expliciet zegt hoe je een fout wil tonen, dan wordt een *TextBox* rood omkaderd. Je kan echter de visualisatie van de fout op twee manieren beïnvloeden:

- je kan visueel de *TextBox* veranderen door bijvoorbeeld een rode achtergrond te laten zien (*Style*)
- je kan een melding geven met de fout (*Control Template* dat wordt opgeroepen bij de *Validation.ErrorTemplate*).

Deze mogelijkheden staan los van elkaar.

Indien je de *Style* (met zijn *Trigger*) gaat definiëren als een *Window Resource*, dan kan je deze *Style* meerdere keren in het *Window* gebruiken (zie later):

Voeg onderaan binnen de *Window.Resources*-tag de *Style* toe

```
<Style x:Key="textBoxInError" TargetType="{x:Type TextBox}">
 <Style.Triggers>
 <Trigger Property="Validation.HasError" Value="True">
 <Setter Property="Background" Value="Red"></Setter>
 </Trigger>
 </Style.Triggers>
</Style>
```

Voeg daarna de property *Style="{StaticResource textBoxInError}"* toe in de *TextBox*-tag van *postcodeTextBox*.

Dit geeft de rode achtergrondkleur weer bij een foutieve validatie.

```
<TextBox Grid.Column="3" Grid.Row="1" Height="23"
 HorizontalAlignment="Left" Margin="0,0,3,15" x:Name="postcodeTextBox"
 VerticalAlignment="Center" Width="120"
 Style="{StaticResource textBoxInError}">
 <TextBox.Text>
 <Binding Path="Postcode" Mode="TwoWay" UpdateSourceTrigger="PropertyChanged"
 ValidatesOnExceptions="true" NotifyOnValidationError="true">
 <Binding.ValidationRules>
 <local:PostcodeRangeRule2/>
 </Binding.ValidationRules>
 </Binding>
 </TextBox.Text>
</TextBox>
```

Nu nog de tekst van de foutmelding onder de *TextBox* weergeven.

Voeg hiervoor een *ControlTemplate* toe in de *Window.Resource*-tag zodat eventueel meerdere *Controls* hier gebruik van kunnen maken:

```
<ControlTemplate x:Key="validationTemplate">
 <StackPanel Orientation="Vertical">
 <AdornedElementPlaceholder/>
 <TextBlock Text="{Binding [0].ErrorContent}" Foreground="Red"/>
 </StackPanel>
</ControlTemplate>
```

(1) Je definieert een verticaal *StackPanel* (om de gegevens onder elkaar te krijgen)

(2) De *AdornedElementPlaceholder*-tag zegt waar de *Control* zelf zich bevindt

- (3) De *TextBlock* komt eronder met als *Tekst* de eerste foutboodschap van de *Errors* van de huidige binding die je hieronder gaat definiëren

In de *TextBox*-tag van de *postcodeTextBox* moet de link nog gelegd worden tussen deze *Template* en de validatiefout:

voeg de property `Validation.ErrorTemplate="{StaticResource validationTemplate}"` toe in de *TextBox*-tag waardoor de binding van de errors actief komt.

```
<TextBox Grid.Column="3" Grid.Row="1" Height="23"
 HorizontalAlignment="Left" Margin="0,0,3,15" x:Name="postcodeTextBox"
 VerticalAlignment="Center" Width="120"
 Validation.ErrorTemplate="{StaticResource validationTemplate}"
 Style="{StaticResource textBoxInError}">
 <TextBox.Text>
 <Binding Path="Postcode" Mode="TwoWay" UpdateSourceTrigger="PropertyChanged"
 ValidatesOnExceptions="true" NotifyOnValidationError="true">
 <Binding.ValidationRules>
 <local:PostcodeRangeRule2/>
 </Binding.ValidationRules>
 </Binding>
 </TextBox.Text>
</TextBox>
```

Om deze foutboodschap op een deftige manier te tonen, ga je de *Margin* moeten aanpassen van *TextBox* en eventueel de *Height* van het detailgedeelte.

In sommige omstandigheden is het niet duidelijk bij welk element opmaak voor de *ValidationRule* hoort omdat deze werkt met de *VisualTree* van de XAML-code. Daar kan je duidelijkheid over scheppen door dit element te omringen door een *AdornerDecorator*-tag. Toegepast op het voorgaande voorbeeld:

```
<AdornerDecorator Grid.Column="3" Grid.Row="1">
 <TextBox Grid.Column="3" Grid.Row="1" Height="23"

 . . .

 </TextBox>
</AdornerDecorator>
```

Door deze werkwijze te gebruiken, kan er geen misverstand meer zijn waar de opmaak (en de visualisatie) van de validatie komt.

Probeer het resultaat maar uit !

### 11.8.2 Validation van Cells van Datagrid

Als je nu bijvoorbeeld ook de validatie in de *DataGrid* voor het postnummer wil toepassen, dan kan je de bestaande *Style* en de *ValidationRule* ook hiervoor gebruiken. De *Binding* van de *DataGridTextColumn* moet hiervoor worden afgezonderd zodat je de *ValidationRule* kan toevoegen.

De fout kan alleen voorkomen als je de *TextBox* editeert, dus is het de *EditingElementStyle* die je gaat aanpassen. De *Trigger* zorgt ervoor dat bij een foutieve validatie deze specifieke stijl geactiveerd wordt.

We zetten de property `UpdateSourceTrigger="PropertyChanged"` zodat er bij elke verandering gevalideerd wordt.

```
<DataGridTextColumn x:Name="postcodeColumn" Header="Postcode"
 EditingElementStyle="{StaticResource textBoxInError}">
```

```
<DataGridTextColumn.Binding>
 <Binding Path="Postcode" ValidatesOnExceptions="True"
ValidatesOnDataErrors="True" ValidatesOnNotifyDataErrors="True"
UpdateSourceTrigger="PropertyChanged" Mode="TwoWay">
 <Binding.ValidationRules>
 <local:PostcodeRangeRule />
 </Binding.ValidationRules>
 </Binding>
</DataGridTextColumn.Binding>
</DataGridTextColumn>
```

Dit geeft als resultaat :

Brouwernr	Naam	Adres	Postcode	Gemeente	Omzet	
! 1	Achouffe	Route du Village 32	666	Achouffe-Wibrin	10000	

Vooraan de rij met de validatiefout wordt een rood uitroepteken getoond.

### 11.8.3 RowValidation in Datagrid

Het is bij een *DataGrid* ook mogelijk om niet per *Cell* maar per *Row* een validatie te doen. Er wordt dan pas gevalideerd als je de rij verlaat.

Om dit te illustreren ga je de *PostcodeRangeRule* aanpassen:

```
public class PostcodeRangeRule : ValidationRule
{
 public override ValidationResult Validate(object value,
 CultureInfo cultureInfo)
 {
 int postcode = 0;
 if (value is BindingGroup)
 postcode = ((Brouwer)(value as BindingGroup).Items[0]).Postcode;
 else
 {
 if (!int.TryParse(value.ToString(), out postcode))
 {
 return new ValidationResult(false, "Gelieve een getal in te geven");
 }
 }
 if ((postcode < 1000) || (postcode > 9999))
 {
 return new ValidationResult(false,
 "De postcode moet tussen 1000 en 9999 liggen");
 }
 else
 {
 return ValidationResult.ValidResult;
 }
 }
}
```

- (1) Als de validatie vanuit de *DataGrid* wordt aangeroepen, dan is de *value* een *BindingGroup*. Dit komt overeen met een record uit de Binding wat in dit geval een *Brouwer* is. Het eerste object (*Items[0]*) is de huidige rij. De rest van de validatie blijft behouden.



Onder `DataGrid.Columns` geven we aan welke `RowValidationRule` we gaan gebruiken:

```
<DataGrid.RowValidationRules>
 <local:PostcodeRangeRule ValidationStep="UpdatedValue"/>
</DataGrid.RowValidationRules>
```

Nu moet er enkel nog toegevoegd worden, hoe de fouten moeten getoond worden bij een foute validatie. We doen dit met een template. Dit komt onder de `RowValidationRules` tag.

```
<DataGrid.RowValidationErrorTemplate>
 <ControlTemplate>
 <Grid Margin="0,-2,0,-2" (1)
 ToolTip="{Binding RelativeSource={RelativeSource FindAncestor,
 AncestorType={x:Type DataGridRow}},
 Path=(Validation.Errors)[0].ErrorContent}"> (2)
 <Ellipse StrokeThickness="0" Fill="Red"
 Width="{TemplateBinding FontSize}"
 Height="{TemplateBinding FontSize}" /> (3)
 <TextBlock Text="!" FontSize="{TemplateBinding FontSize}"
 FontWeight="Bold" Foreground="White"
 HorizontalAlignment="Center" /> (4)
 </Grid>
 </ControlTemplate>
</DataGrid.RowValidationErrorTemplate>
```

- (1) Bepaalt de plaatsing van het icoontje in de nulde kolom van de grid. Die kolom komt voor de echte kolommen.
- (2) Maakt dat de error tekst in de tooltip terechtkomt, op diezelfde plaats in de grid.
- (3) De fout wordt weergegeven met een rood cirkeltje, waarin het teken verschijnt dat je in de `TextBlock Text` opgeeft bij (4)

Tijd om uit te testen. Om alleen het effect te zien van de *RowValidation*, **zet je eerst de *ValidationRule* van de *Binding* zelf in commentaar.**

Nu zal er bij een foute ingave niets gebeuren totdat je naar een andere *Row* wil veranderen. De tooltip zie je als je met de muisaanwijzer over het fout icoontje beweegt.

Als je de *ValidationRule* van de *Binding* uit commentaar haalt, krijg je, van zodra een fout optreedt, een reactie van zowel de *Cell* als de *Row*.

Brouwernr	Naam	Adres	Postcode	Gemeente
1	Achouffe	Route du Village 32	666	Achouffe-Wibrin
2	Alken	Stationstraat 2	3570	Alken
De postcode moet tussen 1000 en 9999 liggen		ale 45	6953	Ambly-Nassogne

#### 11.8.4 Een *ValidationRule* in code wijzigen

Het kan gebeuren dat bijvoorbeeld bij het maken van een object andere *ValidationRules* gelden dan bij het wijzigen ervan, of dat er bepaalde opties bepalen welke *ValidationRules* er momenteel gelden.

Daardoor wordt het interessant om in code te bepalen welke *ValidationRules* er van toepassing zijn.

Ter illustratie maken we een *ValidationRule* voor de postcode bij waarin het getal 0 ook een geldige waarde is. Kopieer hiervoor *PostcodeRangeRule* naar *PostcodeRangeRule0* en verander ook de *Class-naam*.

Voeg de extra test

```
if (postcode == 0)
 return ValidationResult.ValidResult;
```

toe na het eerste if-then-else statement.

Om de verschillende mogelijkheden te testen in onze applicatie voeg bovenaan je een *CheckBox* toe achter de *Button* van het Zoeken:

```
<CheckBox Name="checkBoxPostcode0" Content="Poscode 0 ok" VerticalAlignment="Center"
Click="checkBoxPostcode0_Click" ></CheckBox>
```

In het *Click*-event wordt er getest of de postcode 0 toegelaten is of niet, en de bijhorende *ValidationRule* ingesteld.

```
private void checkBoxPostcode0_Click(object sender, RoutedEventArgs e)
{
 Binding binding1 = BindingOperations.GetBinding(postcodeTextBox,
 TextBox.TextProperty);
 binding1.ValidationRules.Clear();

 var binding2 = (postcodeColumn as DataGridBoundColumn).Binding as Binding;
 binding2.ValidationRules.Clear();

 brouwerDataGrid.RowValidationRules.Clear();

 switch (checkBoxPostcode0.IsChecked)
 {
 case true:
 binding1.ValidationRules.Add(new PostcodeRangeRule0());
 binding2.ValidationRules.Add(new PostcodeRangeRule0());
 brouwerDataGrid.RowValidationRules.Add(new PostcodeRangeRule0());
 break;
 case false:
 binding1.ValidationRules.Add(new PostcodeRangeRule());
 binding2.ValidationRules.Add(new PostcodeRangeRule());
 brouwerDataGrid.RowValidationRules.Add(new PostcodeRangeRule());
 break;
 default:
 binding1.ValidationRules.Add(new PostcodeRangeRule());
 binding2.ValidationRules.Add(new PostcodeRangeRule());
 brouwerDataGrid.RowValidationRules.Add(new PostcodeRangeRule());
 break;
 }
}
```

(1) De *GetBinding* method geeft de *Binding* van een bepaalde property (dat een *DependencyObject* moet zijn=*TextProperty*) van bepaald object (dat een *DependencyObject* moet zijn=*PostcodeTextBox*) terug.

(2) Alle bestaande *ValidationRules* van deze *Binding* worden verwijderd.

- (3) De *DataGridTextBoxColumn* *postcodeColumn* is afgeleid van de basisklasse *DataGridBoundColumn* die een *Binding* aan een column van een *DataGrid* verbindt (via de basisklasse *BindingBase*).
- (4) Alle bestaande *ValidationRules* van deze *Binding* worden verwijderd.
- (5) Aan de *RowValidationRules* van de *DataGrid* kan je rechtstreeks aan, en dus worden ook deze *Rules* verwijderd.
- (6) Als het getal 0 is toegelaten voeg je de *PostcodeRangeRule0* toe aan de bindings, en anders de *PostcodeRangeRule*.  
Let op: een *CheckBox.IsChecked* heeft ook nog een derde geldige waarde namelijk *null*.

De *ValidationRule* heeft 2 properties die eventueel nog ingesteld kunnen worden nl. *ValidationStep* en *ValidatesOnTargetUpdated*. Indien deze properties een bepaalde waarde moeten hebben, dan kan je dit ook via code instellen.

Bijvoorbeeld naar het uitgewerkte voorbeeld toe kan je onderaan bij de *CheckBox Click* volgende code toevoegen:

```
binding1.ValidationRules[0].ValidatesOnTargetUpdated = true;
binding1.ValidationRules[0].ValidationStep = ValidationStep.UpdatedValue;

binding2.ValidationRules[0].ValidatesOnTargetUpdated = true;
binding2.ValidationRules[0].ValidationStep = ValidationStep.UpdatedValue;

brouwerDataGrid.RowValidationRules[0].ValidatesOnTargetUpdated = true;
brouwerDataGrid.RowValidationRules[0].ValidationStep = ValidationStep.UpdatedValue;
```

Je kan de applicatie uitproberen !

### 11.8.5 Reageren op een foute validatie

Tot hiertoe geen probleem in het detailgedeelte van de geselecteerde brouwer. Maar de *DataGrid* heeft geen weet van een eventuele foutieve validatie in het detail-gedeelte, dus als je nu in de *DataGrid* op een andere brouwer klikt, dan kan je zomaar naar een andere brouwer switchen zonder de huidige brouwer in orde te maken.

Om dit te voorkomen kan je in het *PreviewMouseDown*-event testen of de validatie van de details in orde is.

Omdat de *DataGrid* een complexe control is, moet je de *Preview....*-events gebruiken (die op high-level van toepassing zijn) in plaats van het gewone *MouseDown* event omdat deze events low-level plaatsvinden en door de complexiteit van de *DataGrid*-control teniet worden gedaan.

Je kan rechtstreeks de control gaan testen waarop de validatie gebeurt. Vooreerst voeg je aan het *DataGrid* een *PreviewMouseDown* event toe (de naam wordt algemeen gekozen omdat dit ook voor de andere *Controls* gebruikt kan worden):

```
private void testOpFouten_PreviewMouseDown(object sender, MouseButtonEventArgs e)
{
 if (Validation.GetHasError(postcodeTextBox)) e.Handled = true;
}
```

Je sluit op deze manier de handeling van de muis kort door te zeggen dat dit event is afgehandeld : *e.Handled = true*;

Als er meerdere controls gevalideerd worden, wordt het een ganse opsomming van het aftesten van deze controls. Is dit het geval, dan kan je deze testen veralgemenen door de omvattende tag te testen, wat in dit geval een *Grid* is. Door de *Children* van deze control af te gaan, kan je bepalen of er ergens nog een foute validatie aanwezig is.

Verander de code in:

```
private void testOpFouten_PreviewMouseDown(object sender, MouseButtonEventArgs e)
{
 bool foutGevonden = false;
 foreach (var c in gridDetail.Children)
 {
 if (Validation.GetHasError((DependencyObject)c)) (1)
 {
 foutGevonden = true;
 }
 }
 if (foutGevonden) e.Handled = true; (2)
}
```

- (1) Je converteert eerst de control naar een elementair object namelijk *DependencyObject*. De meeste controls zijn hiervan afgeleid. Van dit object wordt de gevraagd of er een validatie-error aanwezig is.
- (2) Indien er ergens een fout in een control is vastgesteld, dan wordt de handeling als afgehandeld beschouwd, en dus de klik kortgesloten.

Nu kan er nog een probleem optreden als er een *AdornerDecorator* gebruikt is bij bepaalde controls. Dit geeft momenteel geen fout terug als de ingesloten control een foutieve validatie heeft omdat je alleen de *Children* test en niet de onderliggende controls.

Om dit op te vangen kan je voorgaande procedure uitbreiden met:

```
private void testOpFouten_PreviewMouseDown(object sender, MouseButtonEventArgs e)
{
 bool foutGevonden = false;
 foreach (var c in gridDetail.Children)
 {
 if (c is AdornerDecorator) (1)
 {
 if (Validation.GetHasError(((AdornerDecorator)c).Child))
 {
 foutGevonden = true;
 }
 }
 else if (Validation.GetHasError((DependencyObject)c))
 {
 foutGevonden = true;
 }
 }
 if (foutGevonden) e.Handled = true;
}
```

- (1) Als het een *AdornerDecorator* is, test dan het omsloten object, wat overeenkomt met het *Child*. Een *AdornerDecorator* kan maar één *Child* hebben.

Dit principe kan je nu ook toepassen op de *Grid* met de details en de *Listbox*. Als er in de *DataGrid* een foute validatie is, mag je niet meer naar de andere controls kunnen gaan. Hiervoor kan je procedure nog uitbreiden:

```
private void testOpFouten_PreviewMouseDown(object sender, MouseButtonEventArgs e)
{
 bool foutGevonden = false;
 foreach (var c in gridDetail.Children)
 {
 if (c is AdornerDecorator)
 {
 if (Validation.GetHasError(((AdornerDecorator)c).Child))
 {
 foutGevonden = true;
 }
 }
 else if (Validation.GetHasError((DependencyObject)c))
 {
 foutGevonden = true;
 }
 }

 foreach (var c in brouwerDataGrid.ItemsSource)
 {
 var d = brouwerDataGrid.ItemContainerGenerator.ContainerFromItem(c);
 if (d is DataRow)
 {
 if (Validation.GetHasError(d))
 {
 foutGevonden = true;
 }
 }
 }

 if (foutGevonden) e.Handled = true;
}
```

Natuurlijk moet je ook de *PreviewMouseDown* van de *Grid* en de *ListBox* invullen met deze procedure.

Om te voorkomen dat er met het toetsenbord naar een andere control wordt gegaan, kan je in het *PreviewKeyDown* event gelijkaardige testen inbouwen. Eventueel bepaalde toetsen testen (zoals TAB of Enter) want elke toetsaanslag zou tot het blokkeren van het programma kunnen leiden.

In praktijk ga je dus alle mogelijkheden om van brouwer te veranderen moeten aanpakken om te voorkomen dat er naar een andere brouwer wordt gegaan terwijl de huidige brouwer fouten bevat.

In het geval van het voorbeeld zou je nog de zelfgemaakte menubalk de navigatieknoppen en de andere mogelijkheden moeten voorzien van deze test en voorkomen dat de rest van de code wordt uitgevoerd.

## 11.9 Conversie van gegevens

### 11.9.1 Conversie van control naar property in object

Als de gebruiker een waarde intikt in een cel van een *DataGrid* of in een *TextBox* is deze waarde initieel van het type *String*.

Het data-binding-mechanisme converteert deze String automatisch naar het type van de property van het interne geheugen-object dat bij de DataGrid-cel of de TextBox hoort.

Als de gebruiker een waarde intikt in de DataGrid-cel die hoort bij de Postcode, converteert het data-binding-mechanisme de waarde automatisch naar Int16 (het type van de property Postcode in de class Brouwer).

Als de conversie niet lukt, werpt het data-binding-mechanisme een fout die je kan opvangen en aan de eindgebruiker tonen als een foutmelding.

Soms is deze ingebouwde conversie niet voldoende voor jouw behoeften. Bij de class Brouwer kan de property Omzet ofwel een getal, ofwel de waarde *null* bevatten.

De waarde *null* is een interne geheugen-waarde die de eindgebruiker niet kan intikken. De gebruiker zal de cel in de DataGrid leeg maken of de TextBox leeg maken. Dit geeft een lege String. Het data-binding-mechanisme probeert deze waarde om te zetten naar een Int16, wat niet lukt.

Daarom voegen we het volgende toe aan de xaml-code:

```
<DataGridTextColumn Header="Omzet"
 Binding="{Binding Omzet, UpdateSourceTrigger=PropertyChanged,
 TargetNullValue={x:Static sys:String.Empty}}" />
```

Nu wordt *sys*: niet herkend. Daarom moet er aan het begin van de xaml-code nog het volgende worden bijgevoegd:

```
xmlns:sys="clr-namespace:System;assembly=mscorlib"
```

mscorlib: even opzoeken op google levert het volgende resultaat:

"When Microsoft first started working on the .NET Framework, MSCorLib.dll was an acronym for Microsoft Common Object Runtime Library. Once ECMA started to standardize the CLR and parts of the FCL, MSCorLib.dll officially became the acronym for Multilanguage Standard Common Object Runtime Library."

Bij de TextBox bij simple data binding voegen we het volgende toe, om daar ook de waarde null te kunnen ingeven:

```
Text="{Binding Path=Omzet, TargetNullValue={x:Static sys:String.Empty},
 . . .
```

Als je het programma uitvoert, krijg je geen foutmelding meer bij het intikken van een lege String in de cel van de DataGridView die hoort bij de omzet en krijg je ook geen foutmelding meer als je een lege String intikt in de TextBox die hoort bij de omzet.

### 11.9.2 Tonen van een decimale waarde als een valutawaarde (currency)

De omzet van de brouwer is eigenlijk een valutawaarde (= getal met een munteenheid) die momenteel als een gewoon getal getoond wordt.

Eigenlijk zijn er Windows-instellingen die het formaat vastleggen van een valutawaarde : Control Panel : Clock, Language and Region (=Culture).

Je moet dus van het getal gaan specificeren dat het een valuta-getal is (=StringFormat) en welke windows-instelling van toepassing moet zijn (=Culture). Standaard gaat XAML ervan uit dat je de standaardinstelling wil gebruiken en die is "us-en", waardoor je een \$-teken in je resultaat te zien krijgt.

Zowel in de DataGrid als in de TextBox kan je het formaat van de Omzet gaan instellen in het Binding-gedeelte.

Je kan een *ConverterCulture* attribuut toevoegen waarmee je aangeeft welke *Culture* je wil gebruiken. Hiervoor dient de **namespace xmlns:glob="clr-namespace:System.Globalization;assembly=mscorlib"**.

**Gebruik zeker het .NET Framework 4.6 om deze instellingen te kennen** (die vind je terug in de properties van het project) !

Door te specificeren *ConverterCulture*={x:Static glob:CultureInfo.CurrentCulture}, stel je de huidige Windows-instellingen (CurrentCulture) in.

Een tweede attribuut bepaalt dat het getal een valutawaarde is: *StringFormat=c*

De c staat voor Currency. Een lijst van mogelijkheden vind je terug op

<https://msdn.microsoft.com/en-us/library/dwhawy9k.aspx>.

Toegepast op de huidige XAML-code wordt dit:

```
<DataGridTextColumn x:Name="omzetColumn" Header="Omzet" Binding="{Binding
Omzet,UpdateSourceTrigger=PropertyChanged,TargetNullValue={x:Static
sys:String.Empty}, ConverterCulture={x:Static
glob:CultureInfo.CurrentCulture}, StringFormat=c}"/>
```

Deze werkwijze kan je ook toepassen voor de Omzet bij de brouwer-details.

```
<TextBox Grid.Column="3" Grid.Row="2" Height="23"
HorizontalAlignment="Left" Margin="3" Name="omzetTextBox"
Text="{Binding Path=Omzet, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged,
ValidatesOnExceptions=true, NotifyOnValidationError=true,
TargetNullValue={x:Static sys:String.Empty}, ConverterCulture={x:Static
glob:CultureInfo.CurrentCulture}, StringFormat=c}"
VerticalAlignment="Center" Width="120" />
```

Probeer maar uit.

### 11.9.3 Opmaak en validatie

Als je deze valuta- opmaak in combinatie met een *ValidationRule* gaat gebruiken, kan je wel in de problemen komen om de geldigheid te testen.

Je gaat een *ValidationRule* opstellen voor de *Omzet* bij de details namelijk: als de omzet is ingevuld, dan moet deze groter zijn dan 0.

Het opstellen van de voorwaarde is geen probleem:

```
public class IngevuldGroterDanNul : ValidationRule
{
 public override ValidationResult Validate(object value,
System.Globalization.CultureInfo cultureInfo)
 {
 decimal getal;
```



```

 if (value == null || value.ToString() == string.Empty)
 {
 return ValidationResult.ValidResult;
 }

 if (!decimal.TryParse(value.ToString(), out getal))
 {
 return new ValidationResult(false, "Waarde moet een getal zijn");
 }
 if (getal <= 0)
 {
 return new ValidationResult(false, "Getal moet groter zijn dan nul");
 }
 return ValidationResult.ValidResult;
}
}

```

In de XAML-code haal je de Text-property uit de TextBox-tag om de *ValidationRule* zelf toe te voegen, en voeg je in de TextBox-tag de *Validation.ErrorTemplate* en de *Style* toe:

```

<TextBox Grid.Column="3" Grid.Row="2" Height="23"
 HorizontalAlignment="Left" Margin="3" Name="omzetTextBox"
 VerticalAlignment="Center" Width="120"
 Validation.ErrorTemplate="{StaticResource validationTemplate}"
 Style="{StaticResource textBoxInError}">
 <TextBox.Text>
 <Binding Path="Omzet"
 ValidatesOnDataErrors="True"
 UpdateSourceTrigger="PropertyChanged"
 ValidatesOnNotifyDataErrors="True"
 Mode="TwoWay"
 ConverterCulture="{x:Static
glob:CultureInfo.CurrentCulture}"
 StringFormat="c"
 TargetNullValue="{x:Static sys:String.Empty}">
 <Binding.ValidationRules>
 <local:IngevuldGroterDanNul/>
 </Binding.ValidationRules>
 </Binding>
 </TextBox.Text>
</TextBox>

```

Als je dit uitprobeert, zal je merken dat je bij het tikken van de omzet vanaf het tweede getal een foutieve validatie krijgt (ook al is dit eigenlijk ok).

Reden: bij het testen wordt het euro-teken als celinhoud beschouwd en mits dit geen getal is, wordt er fout gevalideerd. Om er voor te zorgen dat bij de validatie rekening gehouden wordt met bepaalde opmaak, kan men gebruik maken van de *CultureInfo* die als parameter bij de validatie binnenkomt, alsook in het bijvoorbeeld omzetten naar een getal een *style* doorgeven zodat bepaalde karakters genegeerd worden.

Pas de *ValidationRule* aan als volgt:

```

public override ValidationResult Validate(object value,
 System.Globalization.CultureInfo cultureInfo)
{

```



```
decimal getal;
NumberStyles style = NumberStyles.Currency; (1)

// mag niet ingevuld zijn
if (value == null || value.ToString() == string.Empty)
{
 return ValidationResult.ValidResult;
}

if (!decimal.TryParse(value.ToString(), style, cultureInfo, out getal)) (2)
{
 return new ValidationResult(false, "Waarde moet een getal zijn");
}
}
if (getal <= 0)
{
 return new ValidationResult(false, "Getal moet groter zijn dan nul");
}
return ValidationResult.ValidResult;
}
```

- (1) *NumberStyles* is een *Enumeration* die je, eventueel in combinaties (door te “or”en = || ) kan meegeven in een *Parse* of *TryParse* functie. Door hier te zeggen dat het *NumberStyles.Currency* is, wordt er geen rekening gehouden met het valuta-symbool.
- (2) In de *TryParse* wordt de *NumberStyle* doorgegeven alsook de *CultureInfo* die van toepassing is. In dit geval geeft dit als resultaat dat het valuta-symbool € wordt genegeerd.

Je kan nog meer op zeker spelen door in de XAML-code bij de *ValidationRule* gebruik te maken van verschillende parameters:

```
<local:IngevuldGroterDanNul ValidatesOnTargetUpdated="True"
 ValidationStep="RawProposedValue"/>
```

*ValidatesOnTargetUpdated* = de validatie wordt uitgevoerd als de andere kant van de *Binding* wordt gewijzigd

*ValidationStep* = het moment dat er gevalideerd moet worden. Dit is een *Enumeration* met als defaultwaarde *RawProposedValue* (= er is nog niets met de ingegeven waarde gebeurd).

## 11.10 Gegevens sorteren

Sorteren van gegevens staat default op true in een *DataGrid* in WPF. Klikken op een kolomhoofding sorteert de gegevens volgens het veld van die kolom. Als je dat niet wilt, kan je die eigenschap op false zetten.

## 11.11 Gegevens filteren

Als je in de *DataGrid* niet alles van de *Collection* wil laten zien, kan je een filter gebruiken om alleen te tonen wat je wil zonder de database te consulteren. Hiervoor moet je gebruik maken van de *Filter*.

Naast de *CheckBox* van de postcode gaan we een *ComboBox* plaatsen die we opvullen met in de database gebruikte postcodes. Voeg na de tag van de Zoeken-button volgende tags toe:

```
<Label Content="Postcode filter:"></Label>
<ComboBox Name="comboBoxPostCode"></ComboBox>
```

Om deze *ComboBox* op te vullen, kan je via bv. een *StoredProcedure* de lijst opvragen in de database, maar om het aantal database-acties te limiteren, kan je dit beter doen door middel van een Linq-query gebaseerd op de *ItemsSource* van de *DataGrid*.

Dit alles moeten we in de code-behind bij het *Window\_Loaded* event doen, zodat vanaf het begin deze lijst is opgevuld. Als extra gaan we ook de mogelijkheid om “alles” te tonen, toevoegen.

Voeg volgende code onderaan in dit event erbij:

```
var nummers = (from b in brouwersOb orderby b.Postcode select
b.Postcode.ToString()).Distinct().ToList(); (1)
nummers.Insert(0, "alles"); (2)
comboBoxPostCode.ItemsSource = nummers; (3)
comboBoxPostCode.SelectedIndex = 0; (4)
```

(1) Haal uit *brouwersOb* alle postcodes:

gesorteerd : *orderby* *b.Postcode*  
elke postcode maar éénmaal : *.Distinct()*  
maak er een lijst van : *.ToList()*

(2) Voeg als eerste *Item* (0) de tekst “alles” toe

(3) Gebruik deze lijst als *ItemsSource* van de *ComboBox*

(4) Zorg dat de mogelijkheid “alles” in het begin geselecteerd staat

Bij het *Selection\_Changed* event van de *ComboBox* gaan we ervoor zorgen dat er een filter geactiveerd wordt als een specifieke postcode gekozen wordt, en alles getoond wordt als “alles” gekozen wordt.

Om de Filter in te stellen, moeten we gebruikmaken van een delegate method *Predicate<T>(function)* waarbij T het soort object is dat je doorgeeft, en *function* de functienaam is die aangeroepen wordt.

In de functie wordt een element (van dat type object) getest op een criteria. Als die voldoet, wordt er *true* teruggegeven, anders *false*.

```
private void comboBoxPostCode_SelectionChanged(object sender,
SelectionChangedEventArgs e)
{
 if (comboBoxPostCode.SelectedIndex == 0) (1)
 brouwerDataGrid.Items.Filter = null;
 else
 brouwerDataGrid.Items.Filter = new Predicate<object>(PostCodeFilter); (2)
 goUpdate(); (3)
 labelTotalRowCount.Content = brouwerDataGrid.Items.Count;
}

public bool PostCodeFilter(object br) (4)
{
 Brouwer b = br as Brouwer; (5)
 bool result = (b.Postcode == Convert.ToInt16(comboBoxPostCode.SelectedValue)); (6)
 return result;
}
```

- (1) Als de `SelectedIndex = 0`, dan wild it zeggen dat “alles” gekozen is, en de Filter wordt afgezet.
- (2) Je geeft objecten van het type *object* door aan de functie *PostCodeFilter* die elk object gaat valideren.
- (3) Zet alle navigatie bovenaan het scherm juist
- (4) De functie *PostCodeFilter* heeft als parameter een *object*
- (5) Dit *object* moet eerst gecast worden naar een *Brouwer* om aan de verschillende velden te kunnen
- (6) Omdat de *SelectedValue* van de ComboBox een object is, wordt deze eerst geconverteerd naar een *Int16* (hetzelfde type object als de *PostCode* van de *Brouwer*) om daarna vergeleken te worden met de *PostCode* van de *Brouwer*.

## 11.12 Verwijderen en toevoegen (OnCollectionChanged)

Als een ganse rij (record) wordt verwijderd of toegevoegd in de DataGrid, dan wijzigt de collectie die hiermee verbonden (Binding) is.

Momenteel is dit een `List<Brouwer>`, maar er bestaat een verzameling `ObservableCollection` die de Interface `INotifyCollectionChanged` implementeert waardoor er wordt bijgehouden wat wordt verwijderd (`OldItems`) en wat wordt toegevoegd (`NewItems`). Telkens de collection in aantal verandert, wordt de method `OnCollectionChanged` uitgevoerd.

### 11.12.1 Objecten verwijderen

Om een brouwer te verwijderen: klik op het grijze vierkantje voor de brouwer in de DataGrid zodat de brouwer volledig geselecteerd is en druk op de Delete toets van je toetsenbord.

Dit verwijdert de brouwer uit de datagrid maar de brouwer staat nog steeds in de database. Bij herstarten van je applicatie verschijnt de brouwer opnieuw in de lijst.

Je moet dus zorgen dat er een mogelijkheid is om de verwijderingen door te voeren naar de database. In dit geval kiezen we voor een Save-Button die alle wijzigingen (verwijderingen, toevoegingen en veranderingen in records) doorvoert.

Voeg bovenaan naast de buttonZoeken een button toe met de volgende XAML code: (SaveHL.bmp vind je terug in de oefenbestanden)

```
<Button Height="28" Name="buttonSave" Width="28">
 <Image Source=".\\Images\\SaveHL.bmp"></Image>
</Button>
```

De *Binding* van de datagrid moet aangepast worden zodat deze reageert op de *OnCollectionChanged* method:

Bovenaan de code verander je de *List* van de brouwers naar een *ObservableCollection*:

```
public partial class OverzichtBrouwers : Window
{
 private CollectionViewSource brouwerViewSource;
```

```
public ObservableCollection<Brouwer> brouwersOb =
 new ObservableCollection<Brouwer>();
```

Vergeet niet bovenaan `using System.Collections.ObjectModel;` toe te voegen

In de `VulGrid()` heeft deze aanpassing een foutmelding bij het oproepen van `GetBrouwersBeginNaam` tot gevolg omdat ze tothiertoe een `List` teruggeeft, die nu veranderd moet worden in een *ObservableCollection*.

Dus in `BrouwerManager.cs`

```
public ObservableCollection<Brouwer>
 GetBrouwersBeginNaam(string beginNaam)
{
 ObservableCollection<Brouwer> brouwers =
 new ObservableCollection<Brouwer>();
 var manager = new BierenDbManager();
 using (var conBieren = manager.GetConnection())
 { . . .
```

Vergeet ook hier `using System.Collections.ObjectModel` niet.

Onderaan in de `VulDeGrid()` voeg je een extra regel toe:

```
brouwersOb.CollectionChanged += this.OnCollectionChanged;
```

Als er nu iets verandert aan de collectie (de Binding), wordt de event `OnCollectionChanged` getriggerd. Die stuurt alle verwijderde items mee in een lijst `OldItems`. Die items vangen we op in een lijst `OudeBrouwers`.

Voeg eerst in de code de lijst `OudeBrouwers` toe, waar je ook `brouwersOb` hebt gedeclareerd:

```
public List<Brouwer> OudeBrouwers = new List<Brouwer>();

void OnCollectionChanged(object sender, NotifyCollectionChangedEventArgs
e)
{
 if (e.OldItems != null)
 {
 foreach (Brouwer oudeBrouwer in e.OldItems)
 {
 OudeBrouwers.Add(oudeBrouwer);
 }
 }
 labelTotalRowCount.Content = brouwerDataGrid.Items.Count;
}
```

`NotifyCollectionChangedEventArgs` heeft  
`using System.Collections.Specialized;` nodig.

Het totaal aantal brouwers wordt in de teller ook aangepast.

Nu is er dus een lijst met de verwijderde brouwers. Die moet worden aangeboden aan de `BrouwerManager` waar een functie moet bestaan om die brouwers effectief te verwijderen.

We zorgen er ook voor dat het resultaat van deze functie een lijst is met alle mislukkingen (dus alle brouwers die uiteindelijk niet verwijderd zijn), zodat we kunnen melden welke brouwers niet verwijderd zijn.

```
public List<Brouwer>SchrijfVerwijderingen(List<Brouwer> brouwers)
{
 List<Brouwer> nietVerwijderdeBrouwers = new List<Brouwer>();
 var manager = new BierenDbManager();
 using (var conBieren = manager.GetConnection())
 {
 conBieren.Open();
 using (var comDelete = conBieren.CreateCommand())
 {
 comDelete.CommandType = CommandType.Text;
 comDelete.CommandText = "delete from brouwers where BrouwerNr = @brouwerNr";
 var parBrouwerNr = comDelete.CreateParameter();
 parBrouwerNr.ParameterName = "@brouwerNr";
 comDelete.Parameters.Add(parBrouwerNr);
 foreach (Brouwer eenBrouwer in brouwers)
 {
 try
 {
 parBrouwerNr.Value = eenBrouwer.BrouwerNr;
 if (comDelete.ExecuteNonQuery() == 0)
 nietVerwijderdeBrouwers.Add(eenBrouwer);
 }
 catch (Exception)
 {
 nietVerwijderdeBrouwers.Add(eenBrouwer);
 }
 } // foreach
 } // comDelete
 } // conBieren
 return nietVerwijderdeBrouwers;
}
```

- (1) De ExecuteNonQuery geeft als resultaat het aantal rijen terug die aangepast, bijgevoegd of verwijderd zijn. Dus als dit nul is, dan is de brouwer niet verwijderd, en wordt deze in de resultaatlijst bijgevoegd
- (2) Als het uitvoeren van het commando op zich een Exception veroorzaakt, bijvoorbeeld als er nog bieren gelinkt zijn, dan komt de betreffende brouwer ook in de resultaatlijst terecht.

We gaan deze SchrijfVerwijderingen oproepen bij het *Click*-event van de Save-button op het WPF-Window. Dubbelklik op de button en in vul het volgende in:

```
private void buttonSave_Click(object sender, RoutedEventArgs e)
{
 List<Brouwer> resultaatBrouwers = new List<Brouwer>();
 var manager = new BrouwerManager();
 if (OudeBrouwers.Count() != 0)
 {
 resultaatBrouwers = manager.SchrijfVerwijderingen(OudeBrouwers);
 if (resultaatBrouwers.Count > 0)
 {
 StringBuilder boodschap = new StringBuilder();
 boodschap.Append("Niet verwijderd: \n");
 foreach (var b in resultaatBrouwers)
 {
 boodschap.Append("Nummer: "+b.BrouwerNr+" : "+b.BrNaam + " niet\n");
 }
 MessageBox.Show(boodschap.ToString());
 }
 }
}
```

```

 MessageBox.Show(OudeBrouwers.Count - resultaatBrouwers.Count +
 " brouwer(s) verwijderd in de database", "Info", MessageBoxButton.OK,
 MessageBoxImage.Information);
 }

 OudeBrouwers.Clear();
}

```


(3)

- (1) Als het verwijderen uit de database resultaatbrouwers teruggeeft, wil dit zeggen dat het verwijderen van deze brouwers mislukt is
- (2) Een boodschap wordt samengesteld met de mislukte verwijderingen, en wordt ook als messagebox getoond
- (3) Er wordt getoond hoeveel brouwers effectief verwijderd zijn

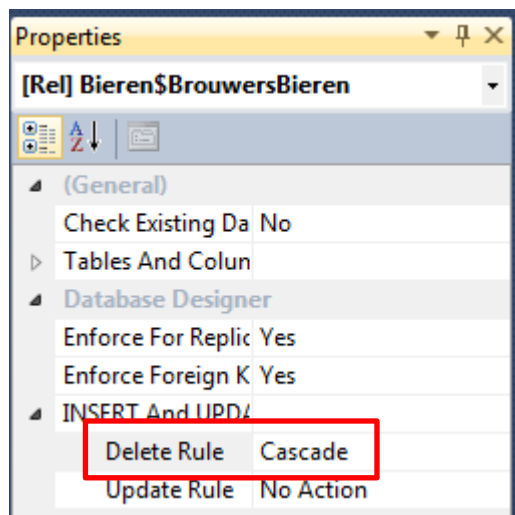
Probeer nu eens een brouwer te verwijderen. Na het klikken op buttonSave moet de brouwer ook uit de database verdwenen zijn.

Je kan er voor te zorgen dat brouwers samen met hun respectievelijke bieren verwijderd worden. Hiervoor met je de relatie-definities tussen beide wijzigen:

Open hiervoor de SQL Server Management Studio. Open de database Bieren. Klik met de rechtermuisknop op de map Database Diagrams en kies voor een nieuw database diagram. Voeg de 3 tabellen toe en klik op de relatie tussen brouwers en bieren.

Klik bij de properties op  bij insert and update specifics en wijzig bij Delete Rule de waarde 'no action' in 'cascade'.

Let hiermee wel goed op, want nu verdwijnen ook alle bieren die de brouwer leverde.



### 11.12.2 Objecten toevoegen

Data Binding kan slechts een nieuwe brouwer toevoegen als de class Brouwer een default constructor heeft.

Je voegt deze constructor toe aan de class Brouwer:

```
public Brouwer() {}
```

Je kunt een brouwer toevoegen in de lege rij onderaan de DataGridView.

Opmerking: als je nu op het laatste record gaat staan en de knoppen Volgende en Laatste uitprobeert, zal je merken dat het al dan niet beschikbaar zijn van deze knoppen niet meer klopt. Dit is echter snel op te lossen:

```
private void goUpdate()
{
 goToPreviousButton.IsEnabled = !(brouwerViewSource.View.CurrentPosition == 0);

 goToFirstButton.IsEnabled = !(brouwerViewSource.View.CurrentPosition == 0);

 goToNextButton.IsEnabled =
 !(brouwerViewSource.View.CurrentPosition == brouwerDataGrid.Items.Count - 2);

 goToLastButton.IsEnabled =
 !(brouwerViewSource.View.CurrentPosition == brouwerDataGrid.Items.Count - 2);

 . . .
}
```

Ook bij de Postcode\_SelectionChanged method moet een kleine aanpassing gebeuren om het juiste aantal te tonen:

```
labelTotalRowCount.Content = brouwerDataGrid.Items.Count - 1;
```

Gelijkaardig aan het verwijderen van een record, gaan we nu het toevoegen van een Brouwer implementeren.

Voeg in de code de lijst NieuweBrouwers toe, waar je ook brouwersOb hebt gedeclareerd:

```
public List<Brouwer> NieuweBrouwers = new List<Brouwer>();
```

OnCollectionChanged wordt nu:

```
void OnCollectionChanged(object sender, NotifyCollectionChangedEventArgs e)
{
 if (e.OldItems != null)
 {
 foreach (Brouwer oudeBrouwer in e.OldItems)
 {
 OudeBrouwers.Add(oudeBrouwer);
 }
 }
 if (e.NewItems != null)
 {
 foreach (Brouwer nieuweBrouwer in e.NewItems)
 {
 NieuweBrouwers.Add(nieuweBrouwer);
 }
 }
 labelTotalRowCount.Content = brouwerDataGrid.Items.Count;
}
```

Nu is er dus een lijst met de toegevoegde brouwers. Die moet nu worden aangeboden aan de BrouwerManager waar een functie moet bestaan om die brouwers effectief toe te voegen.

```
public List<Brouwer> SchrijfToevoegingen(List<Brouwer> brouwers)
{
 List<Brouwer> nietToegevoegdeBrouwers = new List<Brouwer>();
 var manager = new BierenDbManager();
 using (var conBieren = manager.GetConnection())
 {
 using (var comInsert = conBieren.CreateCommand())
 {
 comInsert.CommandType = CommandType.Text;
 comInsert.CommandText = "Insert into brouwers (BrNaam, Adres, Postcode,
Gemeente, Omzet) values (@brnaam, @adres, @postcode, @gemeente, @omzet)";
 var parBrNaam = comInsert.CreateParameter();
 parBrNaam.ParameterName = "@brnaam";
 comInsert.Parameters.Add(parBrNaam);

 var parAdres = comInsert.CreateParameter();
 parAdres.ParameterName = "@adres";
 comInsert.Parameters.Add(parAdres);

 var parPostcode = comInsert.CreateParameter();
 parPostcode.ParameterName = "@postcode";
 comInsert.Parameters.Add(parPostcode);

 var parGemeente = comInsert.CreateParameter();
 parGemeente.ParameterName = "@gemeente";
 comInsert.Parameters.Add(parGemeente);

 var parOmzet = comInsert.CreateParameter();
 parOmzet.ParameterName = "@omzet";
 comInsert.Parameters.Add(parOmzet);

 conBieren.Open();

 foreach (Brouwer eenBrouwer in brouwers)
 {
 try
 {
 parBrNaam.Value = eenBrouwer.BrNaam;
 parAdres.Value = eenBrouwer.Adres;
 parPostcode.Value = eenBrouwer.Postcode;
 parGemeente.Value = eenBrouwer.Gemeente;
 if (eenBrouwer.Omzet.HasValue)
 { parOmzet.Value = eenBrouwer.Omzet; }
 else
 { parOmzet.Value = DBNull.Value; }
 if (comInsert.ExecuteNonQuery() == 0)
 nietToegevoegdeBrouwers.Add(eenBrouwer);
 }
 catch (Exception)
 {
 nietToegevoegdeBrouwers.Add(eenBrouwer);
 }
 } // foreach
 } // comInsert
 } // conBieren
 return nietToegevoegdeBrouwers;
}
```



We roepen deze methode op via onze buttonSave

```
private void buttonSave_Click(object sender, RoutedEventArgs e)
{
 brouwerDataGrid.CommitEdit(DataGridEditingUnit.Row, true); (1)

 List<Brouwer> resultaatBrouwers = new List<Brouwer>();
 var manager = new BrouwerManager();
 if (OudeBrouwers.Count() != 0)
 {
 resultaatBrouwers = manager.SchrijfVerwijderingen(OudeBrouwers);
 if (resultaatBrouwers.Count > 0)
 {
 StringBuilder boodschap = new StringBuilder();
 boodschap.Append("Niet verwijderd: \n");
 foreach (var b in resultaatBrouwers)
 {
 boodschap.Append("Nummer: " + b.BrouwerNr + " : " + b.BrNaam
 + " niet\n");
 }
 MessageBox.Show(boodschap.ToString());
 }
 }
 MessageBox.Show(OudeBrouwers.Count - resultaatBrouwers.Count +
 " brouwer(s) verwijderd in de database", "Info", MessageBoxButton.OK,
 MessageBoxImage.Information);

 resultaatBrouwers.Clear();
 if (NieuweBrouwers.Count() != 0)
 {
 resultaatBrouwers = manager.SchrijfToevoegingen(NieuweBrouwers);
 if (resultaatBrouwers.Count > 0)
 {
 StringBuilder boodschap = new StringBuilder();
 boodschap.Append("Niet toegevoegd: \n");
 foreach (var b in resultaatBrouwers)
 {
 boodschap.Append("Nummer: " + b.BrouwerNr + " : " + b.BrNaam
 + " niet\n");
 }
 MessageBox.Show(boodschap.ToString());
 }
 }
 MessageBox.Show(NieuweBrouwers.Count - resultaatBrouwers.Count +
 " brouwer(s) toegevoegd aan de database", "Info", MessageBoxButton.OK,
 MessageBoxImage.Information);

 VulDeGrid();

 OudeBrouwers.Clear();
 NieuweBrouwers.Clear();
}
```

- (1) Als je op een nieuwe rij staat op het moment dat je wil opslaan, moet er een *CommitEdit* gebeuren van de nieuwe rij waarop je staat. Door de *Edit* af te sluiten, wordt deze ingevuld doorgegeven.
- (2) We gaan de DataGrid terug opvullen om bij de nieuwe brouwers de echte BrouwerNr te tonen.

De code kan uitgetest worden.

## 11.13 Veranderen van gegevens

Bij het gebruik van de interface `INotifyCollectionChanged` wordt alleen de `OnCollectionChanged` getriggert als het aantal objecten in de collection verandert. Bij het veranderen van gegevens in bestaande objecten wordt dit dus niet gedaan.

Om toch te weten te komen welke objecten gewijzigd zijn, gaan we een extra property in de Brouwer klasse toevoegen (`Changed`), die bij creatie op `False` gezet wordt, maar bij de setter van een property naar `True` verandert:

Hiervoor veranderen we deze klasse als volgt :

```
public class Brouwer
{
 private Int32 brouwersNrValue;
 private String brNaamValue;
 private String adresValue;
 private Int16 postcodeValue;
 private String gemeenteValue;
 private Int32? omzetValue;

 public bool Changed { get; set; } (1)

 public Int32 BrouwerNr
 { get{return brouwersNrValue;}}

 public String BrNaam
 {
 get { return brNaamValue; }
 set { brNaamValue = value;
 Changed = true; } (2)
 }

 public String Adres
 {
 get { return adresValue; }
 set
 {
 adresValue = value;
 Changed = true;
 }
 }

 public Int16 Postcode
 {
 get { return postcodeValue; }
 set
 {
 postcodeValue = value;
 Changed = true;
 }
 }

 public String Gemeente
 {
 get { return gemeenteValue; }
 set
 {
 gemeenteValue = value;
 Changed = true;
 }
 }
}
```

```
public Int32? Omzet
{
 get { return omzetValue; }
 set {
 if (value.HasValue && Convert.ToInt32(value) < 0)
 { throw new Exception("Omzet moet positief zijn"); }
 else
 {
 omzetValue = value;
 Changed = true;
 }
 }
}

public Brouwer(Int32 brNr, String brNaam, String adres, Int16 postcode,
 String gemeente, Int32? omzet)
{
 brouwersNrValue = brNr;
 this.BrNaam = brNaam;
 this.Adres = adres;
 this.Postcode = postcode;
 this.Gemeente = gemeente;
 this.Omzet = omzet;
 this.Changed = false;
}

public Brouwer() { }
}
```

(3)

- (1) De extra property Changed van het type boolean
- (2) Bij het toewijzen van een waarde aan een property wordt de Changed property op True gezet.
- (3) Als laatste regel in de constructor wordt de Changed expliciet op False gezet.

Op het moment dat alles moet doorgevoerd worden naar de database wordt er op de buttonSave geklikt. De veranderingen in de bestaande *Binding* (brouwersOb) moeten eerst nog worden verzameld in een *List* die we dan aanbieden aan de BrouwerManager om dit effectief ook in de database door te voeren.

In de BrouwerManager wordt volgende method voorzien:

```
public List<Brouwer> SchrijfWijzigingen(List<Brouwer> brouwers)
{
 List<Brouwer> nietDoorgevoerdeBrouwers = new List<Brouwer>();
 var manager = new BierenDbManager();
 using (var conBieren = manager.GetConnection())
 {
 using (var comUpdate = conBieren.CreateCommand())
 {
 comUpdate.CommandType = CommandType.Text;
 comUpdate.CommandText = "update brouwers set BrNaam=@brnaam, Adres = @adres, Postcode = @postcode, Gemeente = @gemeente, Omzet = @omzet where BrouwerNr = @brouwernr";
 var parBrNaam = comUpdate.CreateParameter();
```

```

parBrNaam.ParameterName = "@brnaam";
comUpdate.Parameters.Add(parBrNaam);

var parAdres = comUpdate.CreateParameter();
parAdres.ParameterName = "@adres";
comUpdate.Parameters.Add(parAdres);

var parPostcode = comUpdate.CreateParameter();
parPostcode.ParameterName = "@postcode";
comUpdate.Parameters.Add(parPostcode);

var parGemeente = comUpdate.CreateParameter();
parGemeente.ParameterName = "@gemeente";
comUpdate.Parameters.Add(parGemeente);

var parOmzet = comUpdate.CreateParameter();
parOmzet.ParameterName = "@omzet";
comUpdate.Parameters.Add(parOmzet);

var parBrouwerNr = comUpdate.CreateParameter();
parBrouwerNr.ParameterName = "@brouwernr";
comUpdate.Parameters.Add(parBrouwerNr);

conBieren.Open();
foreach (Brouwer eenBrouwer in brouwers)
{
 try
 {
 parBrNaam.Value = eenBrouwer.BrNaam;
 parAdres.Value = eenBrouwer.Adres;
 parPostcode.Value = eenBrouwer.Postcode;
 parGemeente.Value = eenBrouwer.Gemeente;
 if (eenBrouwer.Omzet.HasValue) { parOmzet.Value =
eenBrouwer.Omzet; }
 else { parOmzet.Value = DBNull.Value; }
 parBrouwerNr.Value = eenBrouwer.BrouwerNr;

 if (comUpdate.ExecuteNonQuery() == 0)
 nietDoorgevoerdeBrouwers.Add(eenBrouwer);
 }
 catch (Exception)
 {
 nietDoorgevoerdeBrouwers.Add(eenBrouwer);
 }
} // foreach
} // comUpdate
} // conBieren
return nietDoorgevoerdeBrouwers;
}

```

In de code van de Window voorzien we eerst een *List* om de gewijzigde Brouwers te verzamelen:

Voeg in de code de lijst *GewijzigdeBrouwers* toe, waar je ook *brouwersOb* hebt gedeclareerd:

```
public List<Brouwer> GewijzigdeBrouwers = new List<Brouwer>();
```

De code van het *Click*-event van de *buttonSave* breiden we onderaan voor de *MessageBox* uit als volgt:

```

foreach (Brouwer b in brouwersOb)
{

```

```

 if ((b.Changed == true) && (b.BrouwerNr != 0)) GewijzigdeBrouwers.Add(b);
 b.Changed = false;
}

resultaatBrouwers.Clear();
if (GewijzigdeBrouwers.Count() != 0)
{
 resultaatBrouwers = manager.SchrijfWijzigingen(GewijzigdeBrouwers);
 if (resultaatBrouwers.Count > 0)
 {
 StringBuilder boodschap = new StringBuilder();
 boodschap.Append("Niet gewijzigd: \n");
 foreach (var b in resultaatBrouwers)
 {
 boodschap.Append("Nummer: " + b.BrouwerNr + " : " +
 b.BrNaam + " niet\n");
 }
 MessageBox.Show(boodschap.ToString());
 }
}
MessageBox.Show(GewijzigdeBrouwers.Count - resultaatBrouwers.Count +
 " brouwer(s) gewijzigd in de database", "Info", MessageBoxButton.OK,
 MessageBoxImage.Information);

VulDeGrid();

OudeBrouwers.Clear();
NieuweBrouwers.Clear();
GewijzigdeBrouwers.Clear();

```

- (1) De Changed property wordt terug op false gezet anders blijft dit object aangeduid als veranderd. Bij een nieuwe Brouwer wordt bij het invullen van de gegevens de Changed-property op true gezet waardoor dat deze nieuwe Brouwer ook bij de gewijzigde Brouwers terechtkomt. Je kan door de testen of het Levnr verschillend is van nul testen of het al dan niet om een nieuwe Brouwer gaat.

Het programma kan uitgevoerd worden.



Maak uit de oefenmap opgave 9 en 10

## 12 MULTI-USER PROGRAMMA'S

### 12.1 Algemeen

Als meerdere users een database aanpassen, kan het gebeuren dat ze hetzelfde record aanpassen. Als de eerste gebruiker het record aanpast, dan zou een update van de tweede gebruiker de eerste update overschrijven. Misschien is dat wel nodig, maar dan moet de tweede gebruiker op de hoogte zijn van de veranderingen aan de record vooraleer hij zijn update doorvoert.

Een record toevoegen levert geen problemen op, want er wordt een nieuw record gevormd met een nieuw autonumber. Ook de delete-opdracht vormt geen probleem. Enkel de update geeft problemen.


Alle kolommen vergelijken is te veel werk, dus moet er een andere oplossing gevonden worden.

### 12.2 De Database strips

Maak een database strips in de SQL Server.

- Open hiervoor de SQL Server Management Studio.
- Klik op je computernaam in het linkervenster en klik met de rechtermuisknop op Databases.
- Kies voor New Database. Geef de Database de naam Strips en klik op OK.
- Nu vind je in het linkervenster je database terug onder Databases
- Klik op de + voor de database strips. Klik met de rechtermuisknop op Tables en kies voor Table.

Nu geef je de drie velden in:

- **ID** (int, vink allow nulls af), klik rechts op de zwarte pijl aan het begin van de rij en kies voor Set Primary Key  
Bij de Column Properties onderaan, klik je op de  voor Identity Specification. Bij (Is Identity) vul je de waarde yes in. Dit maakt van dit veld een autonumber
- **Naam** (varchar (50), vink Allow Nulls af)
- **Versie** (timestamp, vink Allow Nulls af)
- Save de tabel. Geef de tabel de naam Figuren.
- Refresh Tables zodat deze tabel zichtbaar wordt
- Klik met de rechtermuisknop op de tabel en kies voor Edit Top 200 Rows. Geef een aantal stripfiguren in. Je kent er vast wel enkele.  
(toch enkele tips: Suske, Wiske, Kiekeboe, Jommeke, ...)  
Het timestamp-veld hoeft je niet in te vullen, dat gebeurt automatisch.

SQL Server heeft dus een datatype timestamp, dat automatisch wordt ingevuld en dat wordt gewijzigd bij elke aanpassing van het bijhorende record.

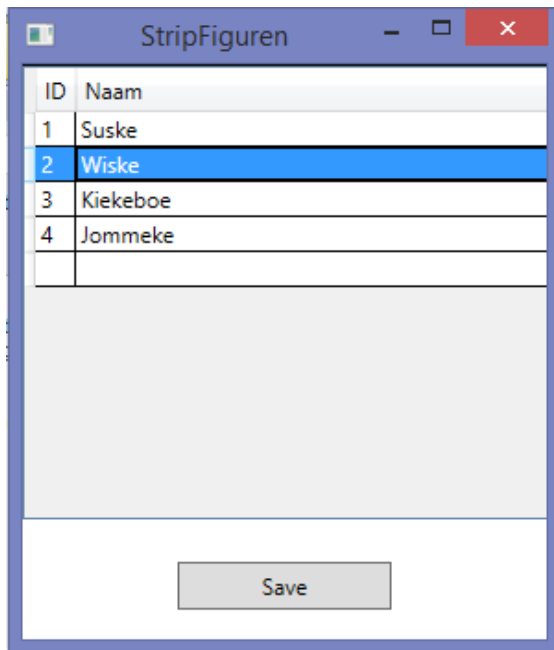
We voegen in de app.config van ons project AdoWPF de nodige connectionstring toe.

App.config:

```
<connectionStrings>
 <add name="strips" connectionString=
 "server=.\sqlexpress;database=strips;integrated security=true"
 providerName="System.Data.SqlClient"/>
</connectionStrings>
```

## 12.3 Conflicten opvangen.

### 12.3.1 WPF-Window StripFiguren



Aan de AdoGemeenschap voegen we een class Figuur toe, met de volgende code:

```
public class Figuur
{
 private Int32 IDValue;
 private String naamValue;
 private System.Object versieValue;

 public Int32 ID
 {
 get{return IDValue;}
 set{IDValue=value;}
 }
 public String Naam
 {
 get{return naamValue;}
 set { naamValue=value;}
 }

 public Object Versie
 {
 get { return versieValue; }
 set { versieValue = value; }
 }
 public Figuur(Int32 id, String naam, Object versie)
 {

```

```

 this.IDValue=id;
 this.Naam=naam;
 this.Versie=versie;
 }
 public Figuur() {}
}

```

Het timestamp datatype is een array van bytes. (byte[]).Daarom vangen dat op in een object.

Voeg aan AdoGemeenschap een class StripManager toe.

Vergeet bovenaan de file de volgende lijnen niet toe te voegen:

```

using System.Configuration;
using System.Data.Common;

public class StripManager
{
 private staticConnectionStringSettings conStripSetting =
 ConfigurationManager.ConnectionStrings["strips"];
 private static DbProviderFactory factory =
 DbProviderFactories.GetFactory(conStripSetting.ProviderName);

 public DbConnection GetConnection()
 {
 var conStrip = factory.CreateConnection();
 conStrip.ConnectionString = conStripSetting.ConnectionString;
 return conStrip;
 }
}

```

En ook nog een class FiguurManager

De methode GetFiguren() haalt alle figuren uit de tabel Figuren en plaatst ze in de list figuren.

```

using System.Data;

public class FiguurManager
{
 public List<Figuur> GetFiguren()
 {
 List<Figuur> figuren = new List<Figuur>();
 var manager = new StripManager();
 using (var conStrip = manager.GetConnection())
 {
 using (var comFiguren = conStrip.CreateCommand())
 {
 comFiguren.CommandType = CommandType.Text;
 comFiguren.CommandText = "select * from Figuren";

 conStrip.Open();
 using (var rdrFiguren = comFiguren.ExecuteReader())
 {
 Int32 IDPos = rdrFiguren.GetOrdinal("ID");
 Int32 NaamPos = rdrFiguren.GetOrdinal("Naam");
 Int32 VersiePos = rdrFiguren.GetOrdinal("Versie");

 while (rdrFiguren.Read())
 {
 figuren.Add(new Figuur(rdrFiguren.GetInt32(IDPos),
 rdrFiguren.GetString(NaamPos),
 rdrFiguren.GetValue(VersiePos)));
 }
 }
 }
 }
 }
}

```



```

 } // do while
 } // using rdrFiguren
 } // using comFiguren
 } // using conStrip
 return figuren;
}
}

```

Kies in het menu *Build* de opdracht *Build Solution* om de code eens te compileren.

Voeg een WPF-Window *StripFiguren* toe aan het project *AdoWPF* en stel dit Window in als opstartWindow van het project.

Of dat betekent in *App.xaml* :

```

<Application x:Class="AdoWPF.App"
 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 StartupUri="StripFiguren.xaml">
 <Application.Resources>
 </Application.Resources>
</Application>

```

Kies in het menu *View* voor *Other Windows* en daarin *Data Sources* om het venster met *Data Sources* te zien.

Voeg een nieuwe *Data Source* toe. Kies in het eerste venster voor object en klik op *Next*. Kies voor *AdoGemeenschap*, *AdoGemeenschap*, *Figuur*. De *DataSource* verschijnt nu in het *DataSources* Venster.

Sleep de *Data Source* naar het Window *StripFiguren*

We passen de layout wat aan:

```

<DockPanel LastChildFill="True">
 <Border Background="PapayaWhip" BorderBrush="Black"
 BorderThickness="1" DockPanel.Dock="Top">
 <DataGrid DataContext="{StaticResource figuurViewSource}"
 AutoGenerateColumns="False"
 EnableRowVirtualization="True"
 Height="200" HorizontalAlignment="Left"
 ItemsSource="{Binding}"
 Name="figuurDataGrid"
 RowDetailsVisibilityMode="VisibleWhenSelected"
 VerticalAlignment="Top" Width="250">
 <DataGrid.Columns>
 <DataGridTextColumn x:Name="idColumn"
 Binding="{Binding Path=ID}" Header="ID"
 Width="SizeToHeader" />
 <DataGridTextColumn x:Name="naamColumn"
 Binding="{Binding Path=Naam}"
 Header="Naam" Width="*" />
 </DataGrid.Columns>
 </DataGrid>
 </Border>
</DockPanel>

```

Bij het loaded-event van het WPF-Window, voeg je volgende code toe:

```

private List<Figuur> figuren = new List<Figuur>();
private CollectionViewSource figuurViewSource;

private void Window_Loaded(object sender, RoutedEventArgs e)

```

```
{
 figuurViewSource = ((System.Windows.Data.CollectionViewSource)
 (this.FindResource("figuurViewSource")));
 FiguurManager manager = new FiguurManager();
 figuren = manager.GetFiguren();
 figuurViewSource.Source = figuren;
}
```

Vergeet using AdoGemeenschap niet

Nu kan je het project eens uittesten. De stripfiguren worden ingevuld in de DataGridView.

Nu moeten we de waarden nog kunnen wijzigen in de Database. Daarvoor moeten we een extra property Changed in de Figuur klasse aanmaken, en deze in de constructor op false zetten. Bij het “setten” van de Naam, wordt de Changed op True gezet:

```
...
public String Naam
{
 get { return naamValue; }
 set { naamValue = value;
 Changed = true;
 }
}

public Figuur(Int32 id, String naam, Object versie)
{
 this.IDValue = id;
 this.Naam = naam;
 this.Versie = versie;
 this.Changed = false;
}

public bool Changed { get; set; }
...

```

Plaats een button (Name: buttonSave, Content: Save) onder de afsluiting van de Border van de Datagrid.

Klik dubbel op die button en vul het Click-event in met volgende code:

```
private void buttonSave_Click(object sender, RoutedEventArgs e)
{
 List<Figuur> GewijzigdeFiguren = new List<Figuur>();
 foreach (Figuur f in figuren)
 {
 if (f.Changed == true)
 GewijzigdeFiguren.Add(f);
 f.Changed = false;
 }

 if (GewijzigdeFiguren.Count != 0)
 {
 FiguurManager manager = new FiguurManager();
 manager.SchrijfWijzigingen(GewijzigdeFiguren);
 }
 GewijzigdeFiguren.Clear();
}
```

- (1) Er wordt een nieuwe List van Figuur objecten aangemaakt om de wijzigingen bij elkaar te brengen.
- (2) Als er gewijzigde Figuren zijn, worden ze in de lijst opgenomen en hun Changed property terug op false gezet.

De method SchrijfWijzigingen komt in de FiguurManager van AdoGemeenschap

```
public void SchrijfWijzigingen(List<Figuur> figuren)
{
 var manager = new StripManager();
 using (var conStrip = manager.GetConnection())
 {
 using (var comUpdate=conStrip.CreateCommand())
 {
 comUpdate.CommandType = CommandType.Text;
 comUpdate.CommandText="update figuren set
 Naam=@naam where ID=@id and Versie=@versie";

 var parNaam = comUpdate.CreateParameter();
 parNaam.ParameterName = "@naam";
 comUpdate.Parameters.Add(parNaam);

 var parVersie = comUpdate.CreateParameter();
 parVersie.ParameterName = "@versie";
 comUpdate.Parameters.Add(parVersie);

 var parID = comUpdate.CreateParameter();
 parID.ParameterName = "@id";
 comUpdate.Parameters.Add(parID);

 conStrip.Open();
 foreach (var eenFiguur in figuren)
 {
 parNaam.Value = eenFiguur.Naam;
 parVersie.Value = eenFiguur.Versie;
 parID.Value = eenFiguur.ID;
 comUpdate.ExecuteNonQuery();
 }
 }
 }
}
```

Dit kan al eens uitgetest worden, door één van de namen van de stripfiguren te wijzigen en te kijken of die ook gewijzigd werd in de database.

### 12.3.2 Een tweede WPF-applicatie

We maken een tweede WPF-applicatie: klik met de rechtermuisknop op de Solution in de Solution Explorer, Add, New Project. Kies voor een WPF Application en geef als naam: AdoWPF2

Klik met de rechtermuisknop op *StripFiguren.xaml* van *AdoWPF* in de Solution Explorer en kies voor *Copy*.

Klik met de rechtermuisknop op *AdoWPF2* en kies *Paste*.

We moeten nog een referentie leggen van AdoWPF2 naar AdoGemeenschap.

Klik met de rechtermuisknop op *AdoWPF2* in de Solution Explorer, *Add, Reference*. Kies bij Solution, Projects voor *AdoGemeenschap* en klik op OK.

In App.xaml van *AdoWPF2* zet je *StripFiguren.xaml* als startscherm.

In App.config voeg je de connectionString toe:

```
<connectionStrings>
 <add
 name="strips"
 connectionString="server=.\sqlexpress;database=strips;integrated security=true"
 providerName="System.Data.SqlClient"/>
</connectionStrings>
```

Zet *AdoWPF2* als StartUp Project en test even uit: wijzig een Naam en klik op Save.

### 12.3.3 Exceptions

In de method *SchrijfWijzigingen* in de *FiguurManager* wijzig je de code als volgt:

```
...
conStrip.Open();
foreach (var eenFiguur in figuren)
{
 parNaam.Value = eenFiguur.Naam;
 parVersie.Value = eenFiguur.Versie;
 parID.Value = eenFiguur.ID;

 if (comUpdate.ExecuteNonQuery() == 0)
 throw new Exception("Iemand was je voor");
}
```

Als de rij niet veranderd is, doordat de versie niet klopte, wordt een exception gecreëerd.

Die vangen we op in de eventhandler *buttonSave\_Click* in de code van het WPF-Window *StripFiguren*

```
...
}
if (GewijzigdeFiguren.Count != 0)
{
 FiguurManager manager = new FiguurManager();
 try
 {
 manager.SchrijfWijzigingen(GewijzigdeFiguren);
 }
 catch (Exception ex)
 {
 MessageBox.Show(ex.Message);
 }
}
GewijzigdeFiguren.Clear();
...
```

Dit doen we zowel in onze *AdoWPF*-applicatie als in onze *AdoWPF2*-applicatie.

### 12.3.4 Conflicten

Klik met de rechtermuisknop op de Solution en kies voor *Set Startup Projects*

Kies voor Multiple Startup Projects en verander bij de projecten de waarde None door Start bij de AdoWPF en AdoWPF2.

Start de applicatie. Nu starten het AdoWPF-Window en het AdoWPF2-Window.

Wijzig een record via het AdoWPF-Window, maar save het nog niet.

Wijzig nu ook hetzelfde record via het AdoWPF2-Window en klik nu wel op Save.

Sla nu ook de record op in het AdoWPF-Window.

Wat is het resultaat?

Bij het AdoWPF-Window klopt het versienummer niet meer, want dat was ondertussen verhoogd via het AdoWPF2-Window. Daardoor gebeurt de update niet en krijg je de foutboodschap.

Probeer het ook eens omgekeerd.



Maak uit de oefenmap opgave 11

## 13 COLOFON

**Sectorverantwoordelijke:**

**Cursusverantwoordelijke:**

Jean Smits

**Medewerkers:**

Veerle Smet

Chris Van Loon

Steven Lucas

Hans Desmet

**Versie:**

Juni 2016

**Nummer dotatielijst:**