

## 9 OPGAVE 8: LIJSTEN OPHALEN

### 9.1 Toepassing op...

- DataReader

### 9.2 Opdracht

Maak een nieuw WPF-Window met een uitvallijst en een gewone lijst. In de uitvallijst toon je alle soorten (alfabetisch). Als de gebruiker in deze lijst een soort aanklikt, toon je de bijhorende planten in de tweede lijst (alfabetisch):



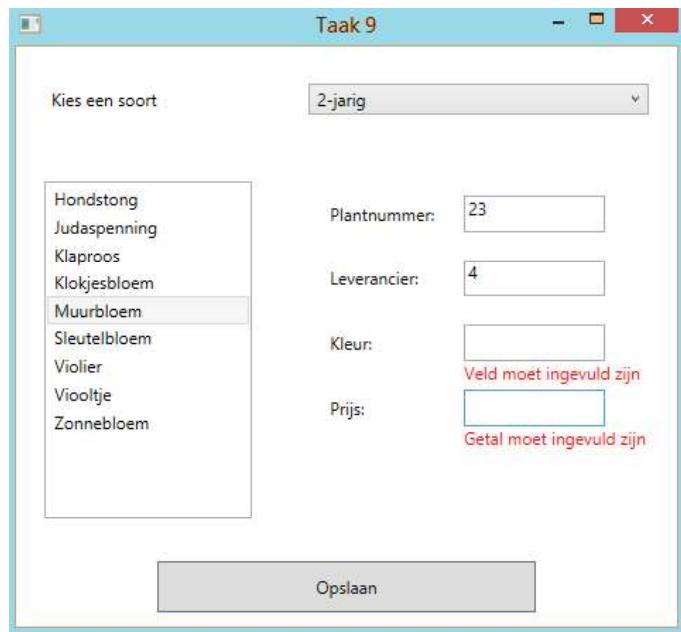
## 10 OPGAVE 9: DATABINDING

### 10.1 Toepassing op...

- Data Binding

### 10.2 Opdracht

Je past het programma van opgave 8 als volgt aan:



- Als de gebruiker één van de planten aanklikt in de lijst, geef je de plantgegevens in tekstboxen naast de lijst. Dit moet gebeuren via DataBinding.
- De Prijs wordt getoond als een valutawaarde.
- Zorg ervoor dat de PlantNr en de LeveranciersNr niet gewijzigd kunnen worden (ReadOnly), de kleur en de prijs wel
- Voeg een knop toe: Opslaan  
→ De vraag wordt gesteld om alle veranderde planten van die soort in de database op te slaan.



- Bij het veranderen van een soort wordt gevraagd om alle veranderde planten van de huidige soort op te slaan alvorens de nieuwe soort te tonen.
- Je voegt een validatie toe voor de kleur (dit veld moet ingevuld zijn) en voor de prijs (dit veld moet een getal groter dan nul zijn)  
→ bij een foute validatie :
  - wordt een foutmelding onder de *TextBox* getoond
  - kan men niet naar een andere plant of andere soort gaan, en de huidige plant zeker niet opslaan.

## 11 OPGAVE 10: DATABINDING

### 11.1 Toepassing op...

- Data Binding

### 11.2 Opdracht WPF

Stel, gebruik makend van een ObjectDatasource, alle leveranciers voor in een grid. We willen alle gegevens zien. De leveranciersnr mag niet gewijzigd worden.

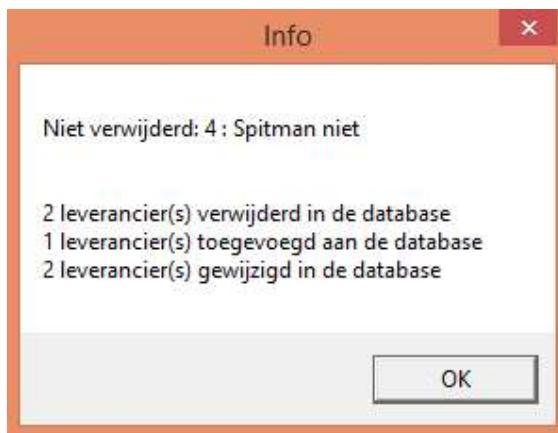
Boven de DataGrid wordt er een ComboBox voorzien waar alle gebruikte postcodes instaan, en de mogelijkheid om alles te kiezen. Deze Combobox wordt **NIET** vanuit de database opgevuld.

Leveranciers				
Kies postnummer		alles		
Lev Nr	Naam	Adres	Post Nr	Woonplaats
2	Baumgarten	Takstraat 13	8500	Kortrijk
3	Struik	Bessenlaan 1	8560	Wevelgem
4	Spitman	Achtertuin 9	8930	Menen
5	Dezaaier	De Gronden 11	8560	Wevelgem
6	Mooiweer	Verlengde zomerstraat 24	8930	Menen
7	Bloem	Linnaeushof 17	8500	Kortrijk
8	De Plukker	Koeleplekstraat 10	8560	Wevelgem
9	Erica	Berkenweg 87	8930	Menen
10	De groene kas	Glasweg 1	8930	Menen
11	Flora	Oeverstraat 76	8930	Menen

Werk het wijzigen, toevoegen en verwijderen van een leveranciersrecord uit, en zorg ervoor dat bij het sluiten van de applicatie de vraag wordt gesteld om alles weg te schrijven naar de database.



Indien alles is uitgevoerd, wordt er een overzichtsscherm getoond met de status van alle verrichtingen:



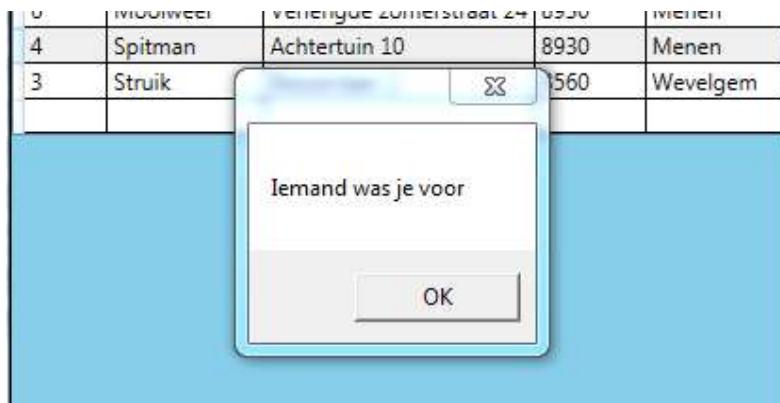
## 12 OPGAVE 11: MULTI-USER PROGRAMMA

### 12.1 Toepassing op...

- Multi-user programma's

### 12.2 Opdracht

Pas de Opgave 10 aan, zodat ze geschikt is voor een multiuser omgeving.



## 13 UITWERKING OPGAVE 1: CONNECTIE MET EEN SQL SERVER DATABASE

### 13.1 De SqlServer database Tuincentrum creëren

Het oefenmateriaal bevat een script (*createTuincentrum.sql*) die deze database aanmaakt.

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

- start deze tool.
- tik in het venster *Connect to Server .\sqlexpress*, als je SQL Express gebruikt, of kies je volwaardige SQL Server uit de lijst.
- laat de keuze bij *Authentication* op *Windows Authentication*.
- Klik op *Connect*.
- kies in het menu *File* de opdracht *Open* en de vervolgopdracht *File*.
- selecteer *CreateTuincentrum.sql / Open*.
- kies in de knoppenbalk de opdracht *Execute*.
- klik in het linkerdeel met de rechtermuisknop op de map *Databases* en kies de opdracht *Refresh*.

### 13.2 De database bekijken in de Server Explorer

- Schakel over naar de Server Explorer door:  
ofwel op het tabblad van de Server Explorer te klikken  
ofwel in het menu te kiezen voor View – Server Explorer  
ofwel Ctrl-Alt-S of Ctrl-W-L te drukken
- Klik op het icoontje of klik met de rechtermuistoets op Data Connections en kies Add Connection...
- Klik in het dialoogvenster Add Connection onder Data source: op de knop Change... en kies voor Microsoft SQL Server.
- Kies onder Server name de gewenste SQL Server. (.\\sqlexpress)
- Als je voor Windows Authentication kiest dan klik je op Use Windows Authentication, kies je voor SQL Server Authentication dan klik je op Use SQL Server Authentication en vul je een gebruikersnaam en wachtwoord in.
- Onder Select or enter a database name kies je voor Tuincentrum.
- Test eventueel de connectie en klik op OK.
- Controleer in de Server Explorer of er onder je Sql Server een database bijgekomen is die de naam draagt die je zonet hebt ingevuld
- Klap de database open door op het voor de naam van de database te klikken.
- Klap nu ook de lijst met tabellen open door op het voor de tekst Tables te klikken.
- Rechtsklik op één van de tabellen en kies in het snelmenu voor Show Table Data.

## 14 UITWERKING OPGAVE 2: CONNECTIES TESTEN

### 14.1 De lay-out

- Maak een nieuw project van het type (template) *WPF Application (AdoWPF)* in een nieuwe solution. (**AdoTaken**)
- Voeg een Button (**buttonTest**) en een Label (**labelStatus**) toe aan het WPF-Window. (**WPFOpgave2**)

### 14.2 DLL project

- Voeg een nieuw Project van het type *Class Library* toe aan de Solution en geef het een naam (bijv. **TakenGemeenschap**).
- Verwijder de Class die al toegevoegd is en voeg een class *TuinDbManager* toe door in de Solution Explorer met de rechtermuistoets te klikken op de naam van het project en vervolgens te kiezen voor *Add – Class*.
- Voeg een reference toe naar *System.Configuration*.
- Voeg bovenaan in de code *using System.Configuration* en *using System.Data.Common* toe aan dit project.
- Build het project.

### 14.3 References en includes

- Selecteer in de Solution Explorer de WPF Application.
- Klik met de rechtermuistoets op het project en kies *Add Reference...*
- Op het tabblad *Solution/Projects* selecteer je de Class Library die je net aangemaakt hebt door de checkbox aan te klikken.
- Klik op *Ok*.
- Voeg in de WPF-applicatie ook een include (using) toe van de Class Library die je aangemaakt hebt en van *System.Data*.

### 14.4 Het configuratiebestand

- Pas in de WPF-applicatie het bestand *App.config* als volgt aan opdat het zou werken met de SQL Server database:

Als je met Windows authentication werkt is de connectionstring als volgt :

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <connectionStrings>
        <add name="Tuin" connectionString=
            "server=.\sqlexpress;database=TuinCentrum;integrated security=true"
            providerName="System.Data.SqlClient"/>
    </connectionStrings>
</configuration>
```

Als je met SQL Server authentication werkt is de connectionstring als volgt :

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <connectionStrings>
        <add name="Tuin" connectionString=
            "server=.\sqlexpress;database=TuinCentrum;
             user id=sa;password=pi3.14"
            providerName="System.Data.SqlClient"/>
    </connectionStrings>
</configuration>
```

## De code

Eerst voorzie je de Dll van twee private shared variabelen en een functie die een connectie retourneert.

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

namespace TakenGemeenschap
{
    public class TuinDbManager
    {
        private static ConnectionStringSettings conTuinSettings =
            ConfigurationManager.ConnectionStrings["Tuin"];
        private static DbProviderFactory factory =
            DbProviderFactories.GetFactory(conTuinSettings.ProviderName);

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

In de Form schrijf je code voor de Button:

```
private void buttonTest_Click(object sender, EventArgs e)
{
    try
    {
        using (var conTuin = new TuinDbManager().GetConnection())
        {
            conTuin.Open();
            labelStatus.Content = "Tuincentrum geopend";
        }
    }
    catch (Exception ex)
    {
        labelStatus.Content = ex.Message;
    }
}
```

Je maakt een nieuwe instantie van de class TuinDbManager. Dit is de class uit het Dll-project. Vervolgens roep je de functie aan uit deze class die een connectie retourneert. Het resultaat ken je toe aan een DbConnection.

In het try-block open je de connectie en geef je een mededeling via het label.

In het Catch-block toon je een foutbericht via het Label.

In het WPF-project vind je App.xaml

Daarin wordt aangegeven met welke pagina de WPF-applicatie start.

```
<Application x:Class="AdoWPF.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    StartupUri="WPFOpgave2.xaml">
    <Application.Resources>
    </Application.Resources>
</Application>
```

## 15 UITWERKING OPGAVE 3: EXECUTENONQUERY

### 15.1 De lay-out

- Voeg een nieuw WPF-Window toe aan het project
- Plaats vier Labels (Naam, Adres, Postcode en Woonplaats) met bijhorende TextBox op de Form
- Plaats daaronder twee Buttons en één Label voor de mededelingen.

### 15.2 De Stored Procedures

In deze oefening gebruik je een aantal Stored Procedures om records toe te voegen, te verwijderen of te wijzigen.

Maak een Stored Procedure *LeverancierToevoegen*:

```
CREATE PROCEDURE LeverancierToevoegen(
    @Naam nvarchar(30),
    @Adres nvarchar(30),
    @PostNr nvarchar(10),
    @Woonplaats nvarchar(30)      )
AS
    INSERT INTO Leveranciers(naam, adres, postnr, woonplaats)
    VALUES (@Naam, @Adres, @PostNr, @Woonplaats)
```

Maak een Stored Procedure *Eindejaarskorting*:

```
CREATE PROCEDURE Eindejaarskorting
AS
    UPDATE Planten
    SET VerkoopPrijs=VerkoopPrijs*0.75
```

### 15.3 De code

#### 15.3.1 Code voor het toevoegen van een leverancier

Je maakt in het Dll-project een class die een Leverancier voorstelt:

```
public class Leverancier
{
    public int LevNr { get; set; }
    public string Naam { get; set; }
    public string Adres { get; set; }
    public string PostNr { get; set; }
    public string Woonplaats { get; set; }
}
```

Je schrijft in het Dll-project een method voor het toevoegen van een leverancier in de database. Als parameter geef je een Leverancier-object mee. Maak hiervoor een nieuwe class in de TuinDbManager. (bovenaan voeg je `using System.Data;` toe)

```
public void LeverancierToevoegen(Leverancier eenLeverancier)
{
    var manager = new TuinDbManager();

    using (var conTuin = manager.GetConnection()) (1)
    {
        using (var comToevoegen = conTuin.CreateCommand()) (2)
        {
            comToevoegen.CommandType = CommandType.StoredProcedure; (3)
            comToevoegen.CommandText = "LeverancierToevoegen"; (4)
        }
    }
}
```

```

var parNaam = comToevoegen.CreateParameter();
parNaam.ParameterName = "@Naam";
parNaam.Value = eenLeverancier.Naam;
comToevoegen.Parameters.Add(parNaam);

var parAdres = comToevoegen.CreateParameter();
parAdres.ParameterName = "@Adres";
parAdres.Value = eenLeverancier.Adres;
comToevoegen.Parameters.Add(parAdres);

var parPostNr = comToevoegen.CreateParameter();
parPostNr.ParameterName = "@PostNr";
parPostNr.Value = eenLeverancier.PostNr;
comToevoegen.Parameters.Add(parPostNr);

var parWoonplaats = comToevoegen.CreateParameter();
parWoonplaats.ParameterName = "@Woonplaats";
parWoonplaats.Value = eenLeverancier.Woonplaats;
comToevoegen.Parameters.Add(parWoonplaats);

conTuin.Open();                                     (6)
comToevoegen.ExecuteNonQuery();
}
}

```

- (1) Je maakt een connectie met de database. Daarvoor hergebruik je de functie die we eerder maakten.
- (2) Via de CreateCommand functie verkrijgen je van de connectie een command.
- (3) Het CommandType wordt StoredProcedure.
- (4) De naam van de Stored Procedure is LeverancierToevoegen
- (5) Je voegt vier parameters toe. Eerst laat je de command een parameter aanmaken, vervolgens geef je deze een naam en een waarde en je voegt de parameter toe aan de command.
- (6) Tenslotte open je de connectie en je voert de command uit.

### 15.3.2 Code voor de eindejaarskorting

In het DLL-project voeg je een functie toe voor het doorvoeren van een prijswijziging.

```

public int EindejaarsKorting()
{
    var manager = new TuinDbManager();

    using (var conTuin = manager.GetConnection())          (1)
    {
        using (var comKorting = conTuin.CreateCommand())   (2)
        {
            comKorting.CommandType = CommandType.StoredProcedure;
            comKorting.CommandText = "EindejaarsKorting";      (3)
            conTuin.Open();
            return comKorting.ExecuteNonQuery();                (4)
        }
    }
}

```

- (1) Je maakt een connectie met de database.
- (2) Via de CreateCommand functie verkrijgen je van de connectie een command.

- (3) Het CommandType is StoredProcedure en de naam van de Stored Procedure is Eindejaarskorting. Er zijn geen parameters.
- (4) Je opent de connectie en voert de command uit. Het resultaat van de command (het aantal gewijzigde records) is de returnwaarde van de functie.

### 15.3.3 Code in de Form

Nu de code voor het toevoegen van een leverancier en voor de eindejaarskorting opgenomen is in het Dll-project, kan je code toevoegen aan de Buttons in de Form.

Voor het toevoegen van een leverancier maak je in een try-block eerst een nieuwe instantie van de TuinDbManager, dit is de class die je aanmaakte in het Dll-project.

Daarna maak je een Leverancier object. Je vult de properties van dit object in met de inhoud van de TextBoxen. Daarna roep je de functie LeverancierToevoegen op met als parameter het Leverancier object. In het label laat je zien of de toevoeging al dan niet succesvol was.

```
private void buttonToevoegen_Click(object sender, EventArgs e)
{
    try
    {
        var manager = new TuinDbManager();
        var deLeverancier = new Leverancier();
        deLeverancier.Naam = textBoxNaam.Text;
        deLeverancier.Adres = textBoxAdres.Text;
        deLeverancier.PostNr = textBoxPostcode.Text;
        deLeverancier.Woonplaats = textBoxWoonplaats.Text;
        manager.LeverancierToevoegen(deLeverancier);
        labelStatus.Content = "nieuwe leverancier is toegevoegd";
    }
    catch (Exception ex)
    {
        labelStatus.Content = ex.Message;
    }
}
```

Voor de eindejaarskorting maak je eerst een nieuwe instantie van de TuinDbManager. Daarna roep je de functie Eindejaarskorting op. Het resultaat van deze functie is het aantal aangepaste records. Om dit getal aan een stukje tekst te plakken gebruik je de ToString-functie.

```
private void buttonEindejaarskorting_Click(object sender, EventArgs e)
{
    try
    {
        var manager = new TuinDbManager();
        labelStatus.Content = manager.EindejaarsKorting().ToString()
            + " plantenprijzen aangepast";
    }
    catch (Exception ex)
    {
        labelStatus.Content = ex.Message;
    }
}
```

## 16 UITWERKING OPGAVE 4: TRANSACTIONS

### 16.1 De lay-out

- Voeg een extra Button toe op het WPF-Window uit de vorige oefening

### 16.2 De Stored Procedures

In deze oefening gebruik je een Stored Procedure om het leveranciersnummer in de tabel planten te wijzigen in een ander nummer en daarnaast een Stored Procedure om een leverancier te wissen uit het leveranciersbestand.

Maak een stored procedure *LeverancierWijzigen*:

```
CREATE PROCEDURE LeverancierWijzigen(@OudLevNr int, @NieuwLevNr int)
AS
UPDATE Planten
SET Levnr = @NieuwLevNr
WHERE Levnr = @OudLevNr
```

Maak een stored procedure *LeverancierVerwijderen*:

```
CREATE PROCEDURE LeverancierVerwijderen (@LevNr int)
AS
DELETE FROM Leveranciers
WHERE LevNr = @LevNr
```

### 16.3 De code

#### 16.3.1 Code in het dll project

Je schrijft in het Dll-project een procedure voor het wijzigen van het leveranciersnummer in de tabel planten en voor het verwijderen van de oude leverancier in de tabel leveranciers.

```
public void VervangLeverancier(int oudLevNr, int nieuwLevNr)
{
    var manager = new TuinDbManager();
    using (var conTuin = manager.GetConnection())
    {
        conTuin.Open();
        using (var traVervangen =
conTuin.BeginTransaction(IsolationLevel.ReadCommitted))
        {
            using (var comWijzigen = conTuin.CreateCommand())
            {
                comWijzigen.Transaction = traVervangen;
                comWijzigen.CommandType = CommandType.StoredProcedure;
                comWijzigen.CommandText = "LeverancierWijzigen";

                var parOudLevNr = comWijzigen.CreateParameter();
                parOudLevNr.ParameterName = "@OudLevNr";
                parOudLevNr.Value = oudLevNr;
                comWijzigen.Parameters.Add(parOudLevNr);

                var parNieuwLevNr = comWijzigen.CreateParameter();
                parNieuwLevNr.ParameterName = "@NieuwLevNr";
                parNieuwLevNr.Value = nieuwLevNr;
                comWijzigen.Parameters.Add(parNieuwLevNr);
            }
        }
    }
}
```

```
if (comWijzigen.ExecuteNonQuery() == 0)
{
    traVervangen.Rollback();
    throw new Exception("Leverancier " + oudLevNr + " kon niet
    vervangen worden door " + nieuwLevNr); (5)
}
}
using (var comVerwijderen = conTuin.CreateCommand())
{
    comVerwijderen.Transaction = traVervangen;
    comVerwijderen.CommandType = CommandType.StoredProcedure;
    comVerwijderen.CommandText = "LeverancierVerwijderen";

    var parLevNr = comVerwijderen.CreateParameter();
    parLevNr.ParameterName = "@LevNr";
    parLevNr.Value = oudLevNr;
    comVerwijderen.Parameters.Add(parLevNr);

    if (comVerwijderen.ExecuteNonQuery() == 0)
    {
        traVervangen.Rollback();
        throw new Exception("Leverancier " + oudLevNr + " kon niet
        verwijderd worden");
    }
    traVervangen.Commit(); (7)
}
}
}
}
```

- (1) We maken een connectie met de database en openen deze
- (2) We beginnen een nieuwe transactie en kiezen voor het isolationlevel ReadCommitted
- (3) We maken een command voor de stored procedure LeverancierWijzigen en voegen die toe aan de transactie
- (4) We voegen twee parameters toe aan de command, één voor het oude leveranciersnummer en één voor het nieuwe
- (5) Als de wijziging geen resultaten oplevert stoppen we de transactie en werpen we een exception
- (6) Zelfde werkwijze voor de tweede command maar deze keer is er slechts één parameter
- (7) Als alles probleemloos verlopen is doen we een commit

### 16.3.2 Code in de form

In de form voeg je onderstaande code toe aan de nieuwe button:

Eerst maak je een nieuwe instantie van de class *TuinDbManager*. Je roept daarna de method *VervangLeverancier* op, met als parameters 2 en 3. Afhankelijk van het resultaat wordt er een mededeling of een foutbericht weergegeven in het label.

```
private void buttonVervang_Click(object sender, RoutedEventArgs e)
{
    try
    {
        var manager = new TuinDbManager();
        manager.VervangLeverancier(2, 3);
        labelStatus.Content =
            "Leverancier 2 is verwijderd en vervangen door 3";
    }
    catch (Exception ex)
    {
        labelStatus.Content = ex.Message;
    }
}
```

## 17 UITWERKING OPGAVE 5: EXECUTESCALAR

### 17.1 De lay-out

- Voeg een nieuw WPF-Window toe aan het project en stel die in als startform.
- Plaats een TextBox (textBoxSoort) met bijhorend Label, een Button (buttonGemiddelde) en een Label (voor de uitvoer)(labelResultaat) op de Form.

### 17.2 De Stored Procedures

In deze oefening gebruik je één Stored Procedure om de gemiddelde prijs van alle planten van één soort te berekenen.

Maak een stored procedure *GemiddeldePrijsVanEenSoort*:

```
CREATE PROCEDURE GemiddeldePrijsVanEenSoort
(@Soort nvarchar(10))
AS
SELECT Avg(VerkoopPrijs) AS GemiddeldeVerkoopPrijs
FROM Soorten INNER JOIN Planten ON Soorten.SoortNr = Planten.SoortNr
WHERE Soorten.Soort=@Soort;
```

### 17.3 De code

#### 17.3.1 Code voor het berekenen van het gemiddelde

Je schrijft in het DLL-project een functie voor het berekenen van de gemiddelde prijs.

```
public Decimal GemiddeldePrijsVanEenSoort(String soort)
{
    var manager = new TuinDbManager();
    using (var conTuin = manager.GetConnection()) (1)
    {
        using (var comGemiddelde = conTuin.CreateCommand())
        {
            comGemiddelde.CommandType = CommandType.StoredProcedure; (2)
            comGemiddelde.CommandText = "GemiddeldePrijsVanEenSoort";

            var parSoort = comGemiddelde.CreateParameter(); (3)
            parSoort.ParameterName = "@soort";
            parSoort.Value = soort;
            comGemiddelde.Parameters.Add(parSoort);

            conTuin.Open();
            var resultaat = comGemiddelde.ExecuteScalar(); (4)
            if (resultaat == DBNull.Value) (5)
            {
                throw new Exception("Soort bestaat niet");
            }
            else
            {
                return (Decimal)resultaat;
            }
        }
    }
}
```

(1) Je maakt een connectie met de database en ook een command

(2) Het CommandType is Stored Procedure

(3) Je voegt een parameter toe: Soort.

- (4) Je opent de connectie, voert de command uit
- (5) Als de command een resultaat oplevert converteer je het naar een decimal, zoniet throw je een exception

### 17.3.2 Code in de Form

In de Form voeg je onderstaande code toe aan `buttonGemiddelde`.

Eerst maak je een nieuwe instantie van de class `TuinDbManager`. Je roept daarna de functie `GemiddeldePrijsVanEenSoort` op, met als parameter de soort. Het resultaat van de functie `GemiddeldePrijsVanEenSoort` is een decimal. Om deze decimal netjes als een geldbedrag tonen, kan je gebruik maken van de `String.Format` functie die met `{0:C}` het getal omzet naar een currency (valuta) waarde.

```
private void buttonGemiddelde_Click(object sender, EventArgs e)
{
    try
    {
        var manager = new TuinDbManager();

        labelResultaat.Content = String.Format("Gemiddelde prijs : {0:C}",
            manager.GemiddeldePrijsVanEenSoort(textBoxSoort.Text));
    }
    catch (Exception ex)
    {
        labelResultaat.Content = ex.Message;
    }
}
```

## 18 UITWERKING OPGAVE 6: MEERDERE GEGEVENS OPZOEKEN

### 18.1 De lay-out

- Voeg een nieuwe Form toe aan het Project
- Plaats een TextBox met bijhorend Label, een Button en 5 keer Labels (voor de opzoekresultaten) op de Form.

### 18.2 De Stored Procedure

De code voor de Stored Procedure *PlantenGegevens*:

```
CREATE PROCEDURE PlantenGegevens(
    @plantnr      int,
    @plantnaam    nvarchar(30)  OUTPUT,
    @soort        nvarchar(10)  OUTPUT,
    @leverancier  nvarchar(30)  OUTPUT,
    @kleur        nvarchar(10)  OUTPUT,
    @kostprijs    money        OUTPUT )
AS
Select @plantnaam=Planten.Naam, @soort=Soort, @kleur=Kleur,
       @leverancier=Leveranciers.Naam, @kostprijs=VerkoopPrijs
from Soorten inner join
     (Planten inner join Leveranciers on Planten.Levnr=Leveranciers.LevNr)
     on Soorten.SoortNr=Planten.SoortNr
where PlantNr=@plantnr
```

- (1) Er is één invoerparameter: het nummer van de plant.  
Daarnaast zijn er 5 outputparameters
- (2) Je selecteert vijf velden en kent ze toe aan de outputparameters
- (3) Een dubbele join: tussen de haakjes join je de tabel Planten met de tabel Leveranciers op basis van het veld LevNr in beide tabellen. Dit resultaat join je met de tabel Soorten op basis van het veld SoortNr uit beide tabellen
- (4) Het resultaat van de join is een aantal records waarvan je enkel deze wilt waarbij het plantnummer gelijk is aan de invoerparameter.

## 18.3 De code

### 18.3.1 Code voor het opzoeken van de plantgegevens

De code voor het opzoeken van de plantgegevens stop je in een functie in het Dll-project. Om het opzoekresultaat, dat uit vijf gegevens bestaat, te bundelen maak je eerst een nieuwe class aan met de naam PlantGegevens, met Getters en Setters voor de 5 Properties en een Constructor.

```
public class PlantGegevens
{
    public string Naam { get; set; }
    public string Soort { get; set; }
    public string Leverancier { get; set; }
    public string Kleur { get; set; }
    public decimal Kostprijs { get; set; }

    public PlantGegevens(string nNaam, string nSoort, string
nLeverancier,
        string nKleur, decimal nKostprijs)
    {
        Naam = nNaam;
        Soort = nSoort;
        Leverancier = nLeverancier;
        Kleur = nKleur;
        Kostprijs = nKostprijs;
    }
}
```

In TuinDbManager.cs

```
public PlantGegevens PlantGegevensOpzoeken(int plantNr) (1)
{
    var manager = new TuinDbManager();

    using (var conTuin = manager.GetConnection())
    {
        using (var comPlantGegevens = conTuin.CreateCommand())
        {
            comPlantGegevens.CommandType = CommandType.StoredProcedure; (2)
            comPlantGegevens.CommandText = "PlantenGegevens";

            var parPlantNr = comPlantGegevens.CreateParameter(); (3)
            parPlantNr.ParameterName = "@plantnr";
            parPlantNr.Value = plantNr;
            comPlantGegevens.Parameters.Add(parPlantNr);

            var parPlantNaam = comPlantGegevens.CreateParameter(); (4)
            parPlantNaam.ParameterName = "@plantnaam";
            parPlantNaam.DbType = DbType.String;
            parPlantNaam.Size = 30;
            parPlantNaam.Direction = ParameterDirection.Output;
            comPlantGegevens.Parameters.Add(parPlantNaam);

            var parSoort = comPlantGegevens.CreateParameter();
            parSoort.ParameterName = "@soort";
            parSoort.DbType = DbType.String;
            parSoort.Size = 10;
            parSoort.Direction = ParameterDirection.Output;
            comPlantGegevens.Parameters.Add(parSoort);
        }
    }
}
```

```
var parLeverancier = comPlantGegevens.CreateParameter();
parLeverancier.ParameterName = "@leverancier";
parLeverancier.DbType = DbType.String;
parLeverancier.Size = 30;
parLeverancier.Direction = ParameterDirection.Output;
comPlantGegevens.Parameters.Add(parLeverancier);

var parKleur = comPlantGegevens.CreateParameter();
parKleur.ParameterName = "@kleur";
parKleur.DbType = DbType.String;
parKleur.Size = 10;
parKleur.Direction = ParameterDirection.Output;
comPlantGegevens.Parameters.Add(parKleur);

var parKostprijs = comPlantGegevens.CreateParameter();
parKostprijs.ParameterName = "@kostprijs";
parKostprijs.DbType = DbType.Currency;
parKostprijs.Direction = ParameterDirection.Output;
comPlantGegevens.Parameters.Add(parKostprijs);

conTuin.Open();                                     (5)
comPlantGegevens.ExecuteNonQuery();
if (parPlantNaam.Value.Equals(DBNull.Value))        (6)
{throw new Exception("Plantgegevens niet gevonden");}

return new PlantGegevens(parPlantNaam.Value.ToString(),
parSoort.Value.ToString(), parLeverancier.Value.ToString(),
parKleur.Value.ToString(), (Decimal)parKostprijs.Value); (7)
}
}
}
```

- (1) Het resultaat van de functie is een object van het type PlantGegevens
- (2) Je maakt een connectie met de database en ook een Command. Het type is een Stored Procedure, genaamd PlantenGegevens.
- (3) Je maakt een (invoer-)parameter parPlantNr...
- (4) ...en vijf uitvoerparameters. Je geeft deze een naam, een DbType en eventueel een lengte, de direction wordt output.
- (5) Je opent de connectie en voert de command uit.
- (6) Als er geen geldige plantnaam wordt gevonden throw je een exception
- (7) Is er wel een geldig resultaat, dan returnt de functie een instantie van de class PlantGegevens, opgevuld via de constructor met de waarden van de 5 outputparameters.

### 18.3.2 Code in de Form

Als de gebruiker in de Form op ButtonOpzoeken klikt, voer je volgende code uit:

```
private void buttonOpzoeken_Click(object sender, EventArgs e)
{
    try
    {
        var manager = new TuinDbManager();                               (1)
        var resultaat = manager.PlantGegevensOpzoeken(
Convert.ToInt16(textBoxPlantNummer.Text));                      (2)
        labelNaam.Content = resultaat.Naam;
        labelSoort.Content = resultaat.Soort;
        labelLeverancier.Content = resultaat.Leverancier;
```

```
    labelKleur.Content = resultaat.Kleur;
    labelKostprijs.Content= String.Format("{0:C}",resultaat.Kostprijs); (3)
}
catch (Exception ex)
{
    labelNaam.Content = ex.Message;
    labelSoort.Content = String.Empty;
    labelLeverancier.Content = String.Empty;
    labelKleur.Content = String.Empty;
    labelKostprijs.Content = String.Empty;
}
}
```

- (1) Je maakt een nieuwe instantie van de class TuinDbManager aan.
- (2) Dan roep je de opzoekfunctie op en bewaart het resultaat in een instantie van de class Gemeenschap.PlantGegevens.
- (3) Je vult de Labels met de onderdelen van het resultaat. Enkel de kostprijs formatteer je in currency-notatie
- (4) Als er geen resultaat is, toon je een foutbericht en maak je de andere Labels leeg.

## 19 UITWERKING OPGAVE 7 : AUTONUMBER-VELDEN

### 19.1 De lay-out

- Aan de lay-out hoeft niks te veranderen

### 19.2 De Stored Procedure

Maak een Stored Procedure *AutoNumberOphalen*:

```
CREATE PROCEDURE AutoNumberOphalen
AS
SELECT @@identity
```

### 19.3 De code

#### 19.3.1 Code voor het toevoegen van een leverancier

Je wijzigt in het Dll-project de sub voor het toevoegen van een leverancier.

```
public void LeverancierToevoegen(Leverancier eenLeverancier)
{
    var manager = new TuinDbManager();
    using (var conTuin = manager.GetConnection())
    {
        using (var comToevoegen = conTuin.CreateCommand())
        {
            comToevoegen.CommandType = CommandType.StoredProcedure;
            comToevoegen.CommandText = "LeverancierToevoegen";

            var parNaam = comToevoegen.CreateParameter();
            parNaam.ParameterName = "@Naam";
            parNaam.Value = eenLeverancier.Naam;
            comToevoegen.Parameters.Add(parNaam);

            var parAdres = comToevoegen.CreateParameter();
            parAdres.ParameterName = "@Adres";
            parAdres.Value = eenLeverancier.Adres;
            comToevoegen.Parameters.Add(parAdres);

            var parPostNr = comToevoegen.CreateParameter();
            parPostNr.ParameterName = "@PostNr";
            parPostNr.Value = eenLeverancier.PostNr;
            comToevoegen.Parameters.Add(parPostNr);

            var parWoonplaats = comToevoegen.CreateParameter();
            parWoonplaats.ParameterName = "@Woonplaats";
            parWoonplaats.Value = eenLeverancier.Woonplaats;
            comToevoegen.Parameters.Add(parWoonplaats);

            using (var comAutoNumber = conTuin.CreateCommand()) (1)
            {
                comAutoNumber.CommandType = CommandType.StoredProcedure;
                comAutoNumber.CommandText = "AutoNumberOphalen";

                conTuin.Open();
                comToevoegen.ExecuteNonQuery();
                eenLeverancier.LevNr =
                    Convert.ToInt32(comAutoNumber.ExecuteScalar()); (2)
            }
        }
    }
}
```

```
        }
    }
}
```

(1) Je maakt een nieuwe command aan : gekoppeld aan een stored procedure met de naam AutoNumberOphalen.

(2) Je voert de command uit. Het resultaat stop je in het veld levnr.

### 19.3.2 Code in de Form

Ook de code die uitgevoerd wordt bij het klikken van de button passen we aan :

```
private void buttonToevoegen_Click(object sender, EventArgs e)
{
    try
    {
        var manager = new TuinDbManager();
        var deLeverancier = new Leverancier();
        deLeverancier.Naam = textBoxNaam.Text;
        deLeverancier.Adres = textBoxAdres.Text;
        deLeverancier.PostNr = textBoxPostcode.Text;
        deLeverancier.Woonplaats = textBoxWoonplaats.Text;
        manager.LeverancierToevoegen(deLeverancier);
        labelStatus.Content = "Leverancier met nummer "+  

        deLeverancier.LevNr+" is toegevoegd"; (1)
    }
    catch (Exception ex)
    {
        labelStatus.Content = ex.Message;
    }
}
```

(1) Je voert het nieuwe leveranciersnummer uit.

## 20 UITWERKING OPGAVE 8: LIJSTEN OPHALEN

### 20.1 De lay-out

- Voeg een nieuw WPF-Window toe aan het Project.
- Plaats er een ComboBox met bijhorend Label en een ListBox op.

### 20.2 De code

#### 20.2.1 Voorbereidende code in dll-project

In het gemeenschappelijk dll-project maak je een class Soort waarin je de naam van de soort en het bijhorende nummer kan bewaren. Van zodra je over een dergelijke class beschikt, kan je Soort-objecten bewaren in de ComboBox.

```
public class Soort
{
    public int SoortNr { get; set; }
    public string SoortNaam { get; set; }

    public Soort(String nSoort, int nSoortNr)
    {
        SoortNaam = nSoort;
        SoortNr = nSoortNr;
    }

    public override String ToString()
    {
        return SoortNaam;
    }
}
```

Voeg een method GetSoorten() toe aan het dll-project. Deze method levert een list van soorten op.

```
public List<Soort> GetSoorten()
{
    var soorten = new List<Soort>();
    var manager = new TuinDbManager();
    using (var conTuin = manager.GetConnection())
    {
        using (var comSoorten = conTuin.CreateCommand())
        {
            comSoorten.CommandType = CommandType.Text;
            comSoorten.CommandText =
                "select SoortNr, Soort from Soorten order by Soort";
            conTuin.Open();
            using (var rdrSoorten = comSoorten.ExecuteReader())
            {
                var soortPos = rdrSoorten.GetOrdinal("soort");
                var soortnrPos = rdrSoorten.GetOrdinal("soortnr");
                while (rdrSoorten.Read())
                {
                    soorten.Add(new Soort(
                        rdrSoorten.GetString(soortPos), rdrSoorten.GetInt32(soortnrPos)));
                }
            }
        }
    }
    return soorten;
```

}

Voeg een method GetPlanten() toe aan het dll-project. Deze method levert een lijst met plantennamen op.

```
public List<String> GetPlanten(int soortnr)
{
    var planten = new List<String>();
    var manager = new TuinDbManager();
    using (var conTuin = manager.GetConnection())
    {
        using (var comPlanten = conTuin.CreateCommand())
        {
            comPlanten.CommandType = CommandType.Text;
            comPlanten.CommandText = "select naam from planten where
                soortnr=@soortnr order by naam";
            var parSoortNr = comPlanten.CreateParameter();
            parSoortNr.ParameterName = "@soortnr";
            parSoortNr.Value = soortnr;
            comPlanten.Parameters.Add(parSoortNr);
            conTuin.Open();
            using (var rdrPlanten = comPlanten.ExecuteReader())
            {
                while (rdrPlanten.Read())
                {
                    planten.Add(rdrPlanten["naam"].ToString());
                }
            }
        }
        return planten;
    }
}
```

### 20.2.2 Het opvullen van de combobox in het WPF-project

In de Window\_Loaded event van de Form voeg je volgende code toe waarin je de ComboBox opvult met alle soortgegevens:

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    try
    {
        var manager = new TuinDbManager(); (1)
        comboBoxSoorten.DisplayMemberPath = "SoortNaam"; (2)
        comboBoxSoorten.SelectedValuePath = "SoortNr";
        comboBoxSoorten.ItemsSource = manager.GetSoorten(); (3)
    }
    catch (Exception ex)
    { MessageBox.Show(ex.Message); }
}
```

- (1) Je maakt een nieuwe instantie van de class TuinDbManager aan.
- (2) Je zorgt ervoor dat de SoortNaam getoond wordt in de combobox en bepaalt dat de waarde van een geselecteerd item SoortNr is.
- (3) Je vraagt alle soorten op en verbindt deze lijst met de combobox.

### 20.2.3 Het opvullen van de listBox in het WPF-project

Telkens de gebruiker een (andere) soort kiest in de ComboBox, werk je de ListBox met planten bij. Je voegt volgende code toe aan de ComboBox:

```
private void comboBoxSoorten_SelectionChanged(object sender,
                                              SelectionChangedEventArgs e)
{
    try
    {
        listBoxPlanten.Items.Clear();                                (1)
        int soortNr = Convert.ToInt32(comboBoxSoorten.SelectedValue); (2)
        var manager = new TuinDbManager();
        var allePlanten = manager.GetPlanten(soortNr);
        foreach (var eenPlant in allePlanten)
            { listBoxPlanten.Items.Add(eenPlant); }                      (3)
    }
    catch (Exception ex)
    {   MessageBox.Show(ex.Message); }
}
```

- (1) Je maakt de ListBox leeg.
- (2) Je cast de SelectedValue naar een integer.
- (3) Je vult de ListBox met de namen van de planten die tot de gekozen soort behoren.

## 21UITWERKING OPGAVE 9: DATABINDING

### 21.1 De lay-out

Zorg ervoor dat de details van een plant naast de ListBox worden weergegeven, en dat de Opslaan-button onderaan komt te staan.

### 21.2 De code

#### 21.2.1 De class Plant

Nu je alle gegevens van een plant nodig hebt, kan je best niet alleen de namen van de planten inlezen in de listBox, maar een gans Plant-object.

In het gemeenschappelijk dll-project maak je hiervoor een class Plant

```
public class Plant
{
    public bool Changed { get; set; } (1)
    public String PlantNaam{ get; set; }
    public int PlantNr{ get; set; }
    public int LevNr{ get; set; }

    private Decimal prijs;
    public Decimal Prijs
    {
        get { return prijs; }
        set { prijs = value;
              Changed = true;
        }
    }

    private String kleur;
    public String Kleur
    {
        get { return kleur; }
        set { kleur = value;
              Changed = true;
        }
    }

    public Plant(String nPlantNaam, int nPlantNr, int nLevNr,
    Decimal nPrijs, String nKleur)
    {
        PlantNaam = nPlantNaam;
        PlantNr = nPlantNr;
        LevNr = nLevNr;
        Prijs = nPrijs;
        Kleur = nKleur;
        Changed = false;
    }
    public Plant() { }
}
```

- (1) De property Changed gaat dienen om wijzigingen in de plant te kunnen detecteren. Bij de Setter van een property die kan wijzigen wordt deze op true gezet, bij het creëren van het object op false.

## 21.2.2 De ListBox vullen met objecten

Je past eerst de method GetPlanten() aan in het DLL-project.

```
public List<Plant> GetPlanten(int soortnr)
{
    var planten = new List<Plant>();
    var manager = new TuinDbManager();
    using (var conTuin = manager.GetConnection())
    {
        using (var comPlanten = conTuin.CreateCommand())
        {
            comPlanten.CommandType = CommandType.Text;
            comPlanten.CommandText = "select * from planten where
                soortnr=@soortnr order by naam"; (1)
            var parSoortNr = comPlanten.CreateParameter();
            parSoortNr.ParameterName = "@soortnr";
            parSoortNr.Value = soortnr;
            comPlanten.Parameters.Add(parSoortNr);
            conTuin.Open();
            using (var rdrPlanten = comPlanten.ExecuteReader())
            {
                var plantNaamPos = rdrPlanten.GetOrdinal("Naam"); (2)
                var plantNrPos = rdrPlanten.GetOrdinal("plantnr");
                var levnrPos = rdrPlanten.GetOrdinal("levnr");
                var prijsPos = rdrPlanten.GetOrdinal("verkoopprijs");
                var kleurPos = rdrPlanten.GetOrdinal("kleur");
                var soortPos = rdrPlanten.GetOrdinal("soortnr");
                while (rdrPlanten.Read())
                {
                    var eenPlant = new Plant(
                        rdrPlanten.GetString(plantNaamPos),
                        rdrPlanten.GetInt32(plantNrPos),
                        rdrPlanten.GetInt32(levnrPos),
                        rdrPlanten.GetDecimal(prijsPos),
                        rdrPlanten.GetString(kleurPos));
                    planten.Add(eenPlant); (3)
                }
            }
        }
    }
    return planten;
}
```

- (1) Het SQL-statement bevat deze keer alles
- (2) Je houdt de kolomnummers van de velden (binnen een record) bij.
- (3) Je maakt een nieuwe instantie van de class Plant aan je vult ze op.  
Deze keer voeg je een Plant-object toe aan de ListBox en geen string

In de eerste versie van deze oefening vulde je de ListBox met strings, de namen van de planten. Deze keer zal je objecten in de ListBox opnemen van de class Plant. Omdat we de gegevens moeten kunnen overlopen, gaan we een List<Plant> aanmaken die we koppelen aan de ListBox.

```
private List<Plant> listBoxPlantenlijst = new List<Plant>(); (1)
private string GeselecteerdeSoortNaam;
```

- (1) Om de huidige soortnaam bij te houden die je nodig hebt bij de vraag of je ze wil opslaan

```

private void comboBoxSoorten_SelectionChanged(object sender,
                                             SelectionChangedEventArgs e)
{
    GeselecteerdeSoortNaam = ((Soort)comboBoxSoorten.SelectedItem).SoortNaam;
    try
    {
        var manager = new TuinDbManager();
        listBoxPlantenlijst =
            manager.GetPlanten(Convert.ToInt32(comboBoxSoorten.SelectedValue));
        listBoxPlanten.ItemsSource = listBoxPlantenlijst;
        listBoxPlanten.DisplayMemberPath = "PlantNaam";
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

```

### 21.2.3 De gegevens weergeven

Door de DataBinding worden de TextBoxen gevuld. Om het decimaal teken als een komma te zien, maken we gebruik van de parameter ConvertureCulture en voor het €-teken voegen we de parameter StringFormat toe.

```

...
<StackPanel Grid.Column="1" Grid.Row="1" Margin="20"
            DataContext="{Binding ElementName=listBoxPlanten, Path=SelectedItem}">
    <StackPanel Orientation="Horizontal" Margin="10">
        <Label Width="100" Content="Plantnummer:"/></Label>
        <TextBox Width="100" Name="textBoxPlantNr" IsReadOnly="True"
                Text="{Binding PlantNr}"/></TextBox>
    </StackPanel>
    <StackPanel Orientation="Horizontal" Margin="10">
        <Label Width="100" Content="Leverancier:"/></Label>
        <TextBox Width="100" Name="textBoxLeverancier" IsReadOnly="True"
                Text="{Binding LevNr}"/></TextBox>
    </StackPanel>
    <StackPanel Orientation="Horizontal" Margin="10">
        <Label Width="100" Content="Kleur:"/></Label>
        <TextBox Width="100" Name="textBoxKleur" Text="{Binding Kleur}"/></TextBox>
    </StackPanel>
    <StackPanel Orientation="Horizontal" Margin="10">
        <Label Width="100" Content="Prijs:"/></Label>
        <TextBox Width="100" Name="textBoxPrijs" Text="{Binding Prijs,
ConverterCulture={x:Static glob:CultureInfo.CurrentCulture},
StringFormat=c}"/></TextBox>
    </StackPanel>
</StackPanel>
<Button Name="buttonOpslaan" Grid.Column="0" Grid.ColumnSpan="2" Grid.Row="2"
       Content="Opslaan" Margin="100,10" Click="buttonOpslaan_Click"/>

```

### 21.2.4 De wijzigingen opslaan

De mogelijkheid om wijzigingen te kunnen opslaan, moeten we voorzien op 2 plaatsen: bij het klikken op de button en bij het veranderen van een soort.

Daarom maken we een aparte procedure die dit voor zijn rekening neemt:

```
private void WijzigingenOpslaan()
{
    List<Plant> gewijzigdePlanten = new List<Plant>(); (1)

    foreach (Plant p in listBoxPlantenlijst) (2)
    {
        if (p.Changed == true)
        {
            gewijzigdePlanten.Add(p);
            p.Changed = false;
        }
    }

    if ((gewijzigdePlanten.Count > 0) && (3)
        (MessageBox.Show("Gewijzigde planten van soort "
        + GeselecteerdeSoortNaam
        +" opslaan ?", "Opslaan", MessageBoxButtons.YesNo,
        MessageBoxIcon.Question, MessageBoxResult.Yes) == MessageBoxResult.Yes))
    {
        var manager = new TuinDbManager();
        try
        {
            manager.GewijzigdePlantenOpslaan(gewijzigdePlanten);
            MessageBox.Show("Planten opgeslagen", "Opslaan",
            MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
        catch (Exception ex)
        {
            MessageBox.Show("Er is een fout opgetreden: " + ex.Message,
            "Opslaan", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
    }
}
```

- (1) Een List<Plant> voor de wijzigingen wordt gecreëerd
- (2) Er wordt door de lijst van planten in de ListBox gegaan om te zien dewelke zijn gewijzigd
- (3) Als er gewijzigde planten zijn EN op de vraag om ze op te slaan wordt positief geantwoord, dan worden ze doorgegeven naar de database.

De wijzigingen effectief opslaan in de database moet worden geïmplementeerd in de TuinDbManager:

```
public void GewijzigdePlantenOpslaan(List<Plant> planten)
{
    var manager = new TuinDbManager();
    using (var conPlant = manager.GetConnection())
    {
        using (var comUpdate = conPlant.CreateCommand())
        {
            comUpdate.CommandType = CommandType.Text;
            comUpdate.CommandText = "update planten set
Kleur=@kleur, VerkoopPrijs=@prijs where PlantNr=@plantnr";
            var parKleur = comUpdate.CreateParameter();
            parKleur.ParameterName = "@kleur";
            comUpdate.Parameters.Add(parKleur);
```

```
var parPrijs = comUpdate.CreateParameter();
parPrijs.ParameterName = "@prijs";
comUpdate.Parameters.Add(parPrijs);

var parPlantNr = comUpdate.CreateParameter();
parPlantNr.ParameterName = "@plantrn";
comUpdate.Parameters.Add(parPlantNr);

conPlant.Open();

foreach (Plant p in planten)
{
    parKleur.Value = p.Kleur;
    parPrijs.Value = p.Prijs;
    parPlantNr.Value = p.PlantNr;
    if (comUpdate.ExecuteNonQuery() == 0)
        throw new Exception(p.PlantNaam + " opslaan mislukt");
}
}
```

De code-behind van de Opslaan-button is het aanroepen van vorige procedure:

```
private void buttonOpslaan_Click(object sender, RoutedEventArgs e)
{
    WijzigingenOpslaan();
}
```

Bij het wijzigen van een soort (SelectionChanged), moeten wijzigingen in de huidige planten ook eventueel worden weggeschreven. Als eerste coderegel implementeren we:

```
WijzigingenOpslaan();
```

### 21.2.5 Validatie toevoegen

Het maken van de *ValidationRules*:

```
public class VeldMoetIngevuldZijn : ValidationRule
{
    public override ValidationResult Validate(object value,
        CultureInfo cultureInfo)
    {
        if (value == null || (((String)value).Length == 0))
        {
            return new ValidationResult(false, "Veld moet ingevuld zijn");
        }
        return ValidationResult.ValidResult;
    }
}

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

```
NumberStyles style = NumberStyles.Currency;
if (value == null || value.ToString() == string.Empty)
{
    return new ValidationResult(false, "Getal moet ingevuld zijn");
}
if (!decimal.TryParse(value.ToString(), style, cultureInfo, 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;
}
```

Implementeren in de XAML-code:

Binnen in de Window-tag:

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

De *TextBox*-tag van de *Kleur* wordt veranderd in:

```
<TextBox Width="100" Name="textBoxKleur" Validation.ErrorTemplate="{StaticResource ErrorTemplate}">
    <TextBox.Text>
        <Binding Path="Kleur" ValidatesOnDataErrors="True" UpdateSourceTrigger="PropertyChanged" ValidatesOnNotifyDataErrors="True">
            <Binding.ValidationRules>
                <local:VeldMoetIngevuldZijn/>
            </Binding.ValidationRules>
        </Binding>
    </TextBox.Text>
</TextBox>
```

De *TextBox*-tag van de *Prijs* wordt veranderd in:

```
<TextBox Width="100" Name="textBoxPrijs" Validation.ErrorTemplate="{StaticResource ErrorTemplate}">
    <TextBox.Text>
        <Binding Path="Prijs" ConverterCulture="{x:Static glob:CultureInfo.CurrentCulture}" StringFormat="c" ValidatesOnDataErrors="True" UpdateSourceTrigger="PropertyChanged" ValidatesOnNotifyDataErrors="True">
            <Binding.ValidationRules>
                <local:GetalGroterDanNul/>
            </Binding.ValidationRules>
        </Binding>
    </TextBox.Text>
</TextBox>
```

In de *ListBox*- en de *ComboBox*-tag worden volgende events toegevoegd: (om te voorkomen dat men naar een andere *plant* kan switchen terwijl er nog fouten zijn)  
*PreviewMouseDown* en de *PreviewKeyDown*

In de code-behind:

```
private void buttonOpslaan_Click(object sender, RoutedEventArgs e)
{
    if (!PlantHasErrors())
        WijzigingenOpslaan();
}

private bool PlantHasErrors()
{
    bool foutGevonden = false;
    if (Validation.GetHasError(textBoxKleur)) foutGevonden = true;
    if (Validation.GetHasError(textBoxPrijs)) foutGevonden = true;
    return foutGevonden;
}

private void listBoxPlanten_PreviewMouseDown(object sender, MouseButtonEventArgs e)
{
    if (PlantHasErrors()) e.Handled = true;
}

private void listBoxPlanten_PreviewKeyDown(object sender, KeyEventArgs e)
{
    if (PlantHasErrors()) e.Handled = true;
}

private void comboBoxSoorten_PreviewMouseDown(object sender, MouseButtonEventArgs e)
{
    if (PlantHasErrors()) e.Handled = true;
}

private void comboBoxSoorten_PreviewKeyDown(object sender, KeyEventArgs e)
{
    if (PlantHasErrors()) e.Handled = true;
}
```

## 22 UITWERKING OPGAVE 10: DATABINDING

### 22.1 De Data Source

- Voeg een nieuw WPF-Window toe aan het Project en stel het in als startpagina.
- Maak het Data Sources-venster zichtbaar. (View/Other windows/Data Sources).  
Kies voor Add New Data Source (  ) en kies voor Object
- In een lijst zie je de naam van je gemeenschappelijke DLL-project staan. Klap de lijst open en kies voor de class Leverancier (uit opgave 3).
- Klik op Finish.  
Sleep uit het venster Data Sources, Leverancier naar het WPF-Window. Default worden de CollectionViewSource leverancierViewSource en de DataGrid leverancierDataGrid aan de form toegevoegd.

### 22.2 De Layout

Naar analogie met de cursus kunnen we het volgende invullen in de xaml code.

```
<Window
...
<Window.Resources>
    <CollectionViewSource x:Key="leverancierViewSource"
        d:DesignSource="{d:DesignInstance {x:Type local:Leverancier},
        CreateList=True}"/>
</Window.Resources>

<StackPanel Orientation="Vertical"
    DataContext="{StaticResource leverancierViewSource}">
    <StackPanel Orientation="Horizontal">
        <Label Content="Kies postnummer" Width="150"/></Label>
        <ComboBox Name="comboBoxPostnummers" Width="100"
            SelectionChanged="comboBoxPostnummers_SelectionChanged">
        </ComboBox>
    </StackPanel>
    <DataGrid x:Name="leverancierDataGrid"
        AutoGenerateColumns="False"
        EnableRowVirtualization="True"
        ItemsSource="{Binding}"
        RowDetailsVisibilityMode="VisibleWhenSelected"
        Width="auto">
        <DataGrid.Columns>
            <DataGridTextColumn x:Name="levNrColumn"
                Binding="{Binding LevNr}" Header="Lev Nr" Width="1*"
                IsReadOnly="True"/>
            <DataGridTextColumn x:Name="naamColumn"
                Binding="{Binding Naam}" Header="Naam" Width="2*"/>
            <DataGridTextColumn x:Name="adresColumn"
                Binding="{Binding Adres}" Header="Adres" Width="3*"/>
            <DataGridTextColumn x:Name="postNrColumn"
                Binding="{Binding PostNr}" Header="Post Nr" Width="2*"/>
            <DataGridTextColumn x:Name="woonplaatsColumn"
                Binding="{Binding Woonplaats}" Header="Woonplaats"
                Width="2*"/>
        </DataGrid.Columns>
    </DataGrid>
</StackPanel>
</Window>
```

## 22.3 De leveranciers ophalen

De class Leverancier die je eerder al gebruikte in opgave 3 beschikt nog niet over een geparametriseerde constructor. Voeg deze eerst toe.

```
public Leverancier(int nLevNr, String nNaam, String nAdres, String
                     nPostNr, String nWoonplaats)
{
    LevNr = nLevNr;
    Naam = nNaam;
    Adres = nAdres;
    PostNr = nPostNr;
    Woonplaats = nWoonplaats;
}
public Leverancier() { }
```

Voeg een functie GetLeveranciers toe aan de class TuinDbManager :

```
public ObservableCollection<Leverancier> GetLeveranciers()
{
    ObservableCollection<Leverancier> leveranciers = new
    ObservableCollection<Leverancier>();
    var manager = new TuinDbManager();

    using (var conPlanten = manager.GetConnection())
    {
        using (var comLeveranciers = conPlanten.CreateCommand())
        {
            comLeveranciers.CommandType = CommandType.Text;
            comLeveranciers.CommandText = "select * from Leveranciers";
            conPlanten.Open();
            using (var rdrLeveranciers = comLeveranciers.ExecuteReader())
            {
                Int32 leverancierNrPos = rdrLeveranciers.GetOrdinal("LevNr");
                Int32 naamPos = rdrLeveranciers.GetOrdinal("Naam");
                Int32 adresPos = rdrLeveranciers.GetOrdinal("Adres");
                Int32 postcodePos = rdrLeveranciers.GetOrdinal("PostNr");
                Int32 gemeentePos = rdrLeveranciers.GetOrdinal("Woonplaats");
                while (rdrLeveranciers.Read())
                {
                    leveranciers.Add(
                        new Leverancier(rdrLeveranciers.GetInt32(leverancierNrPos),
                            rdrLeveranciers.GetString(naamPos),
                            rdrLeveranciers.GetString(adresPos),
                            rdrLeveranciers.GetString(postcodePos),
                            rdrLeveranciers.GetString(gemeentePos)));
                }
            } // using rdrLeveranciers
        } // using comLeveranciers
    } // using conPlanten
    return leveranciers;
}
```

## 22.4 De leveranciers weergeven

Om de leveranciers te kunnen bewerken, is het handig de verzameling in een ObservableCollection bij te houden. In de Window\_Loaded wordt deze dan gekoppeld aan de CollectionViewSource.

```
public ObservableCollection<Leverancier> leveranciersOb = new
ObservableCollection<Leverancier>();

private void Window_Loaded(object sender, RoutedEventArgs e)
{
    System.Windows.Data.CollectionViewSource leverancierViewSource =
((System.Windows.Data.CollectionViewSource)(this.FindResource("leverancier
ViewSource")));
    var manager = new TuinDbManager();
    leveranciersOb = manager.GetLeveranciers();
    leverancierViewSource.Source = leveranciersOb;
}
```

## 22.5 Leveranciers toevoegen en verwijderen

Omdat we werken met een ObservableCollection, is het mogelijk om via de geïmplementeerde interface INotifyCollectionChanged de OnCollectionChanged procedure uit te voeren.

Voeg onderaan in de Window\_Loaded de onderstaande regel toe:

```
leveranciersOb.CollectionChanged += this.OnCollectionChanged;
```

Telkens het aantal objecten in de collection verandert, wordt deze procedure aangeroepen.

```
public List<Leverancier> OudeLeveranciers = new List<Leverancier>();
public List<Leverancier> NieuweLeveranciers = new List<Leverancier>();

void OnCollectionChanged(object sender, NotifyCollectionChangedEventArgs e)
{
    if (e.OldItems != null)
    {
        foreach (Leverancier oudeLev in e.OldItems)
        {
            OudeLeveranciers.Add(oudeLev);
        }
    }
    if (e.NewItems != null)
    {
        foreach (Leverancier nieuweLev in e.NewItems)
        {
            NieuweLeveranciers.Add(nieuweLev);
        }
    }
}
```

De functie om de leveranciers te verwijderen (in TuinDbManager.cs)

```
public List<Leverancier> SchrijfVerwijderingen(List<Leverancier> leveranciers)
{
    var manager = new TuinDbManager();
```

```
List<Leverancier> nietLevs = new List<Leverancier>();

using (var conTuin = manager.GetConnection())
{
    using (var comDelete = conTuin.CreateCommand())
    {
        comDelete.CommandType = CommandType.Text;
        comDelete.CommandText = "delete from leveranciers where LevNr = @levnr";
        var parLevnr = comDelete.CreateParameter();
        parLevnr.ParameterName = "@levnr";
        comDelete.Parameters.Add(parLevnr);
        conTuin.Open();
        foreach (Leverancier eenLeverancier in leveranciers)
        {
            try
            {
                parLevnr.Value = eenLeverancier.LevNr;
                if (comDelete.ExecuteNonQuery() == 0)
                {
                    nietLevs.Add(eenLeverancier);
                }
            }
            catch (Exception)
            {
                nietLevs.Add(eenLeverancier);
            }
        } // foreach
    } // comDelete
} // conTuin
return nietLevs;
}
```

Deze functie geeft een fout bij het verwijderen van een leverancier die nog gelinkt is aan planten.

De functie om de leveranciers toe te voegen (in TuinDbManager.cs)

```
public List<Leverancier> SchrijfToevoegingen(List<Leverancier> leveranciers)
{
    var manager = new TuinDbManager();
    List<Leverancier> nietLevs = new List<Leverancier>();

    using (var conTuin = manager.GetConnection())
    {
        using (var comInsert = conTuin.CreateCommand())
        {
            comInsert.CommandType = CommandType.Text;
            comInsert.CommandText = "Insert into leveranciers (Naam, Adres, PostNr, Woonplaats) values(@naam, @adres, @postcode, @gemeente)";
            var parNaam = comInsert.CreateParameter();
            parNaam.ParameterName = "@naam";
            comInsert.Parameters.Add(parNaam);

            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);

conTuin.Open();

foreach (Leverancier eenLeverancier in leveranciers)
{
    try
    {
        parNaam.Value = eenLeverancier.Naam;
        parAdres.Value = eenLeverancier.Adres;
        parPostcode.Value = eenLeverancier.PostNr;
        parGemeente.Value = eenLeverancier.Woonplaats;
        if (comInsert.ExecuteNonQuery() == 0)
        {
            nietLevs.Add(eenLeverancier);
        }
    }
    catch (Exception)
    {
        nietLevs.Add(eenLeverancier);
    }
} // foreach
} // comInsert
} // conTuin
return nietLevs;
}
```

Deze functie geeft een fout als je een leverancier toevoegt waarbij je het postnummer niet hebt ingevuld.

## 22.6 Leveranciers wijzigen

Om bij te houden welke leveranciers er gewijzigde properties hebben, voegen we een extra property toe : Changed. Deze wordt bij creatie op false gezet, bij de setters van de andere properties wordt deze op true gezet.

```
public string Naam
{
    get { return naamValue; }
    set { naamValue = value;
        Changed = true;
    }
}

public string Adres
{
    get { return adresValue; }
    set { adresValue = value;
        Changed = true;
    }
}

public string PostNr
{
    get { return postNrValue; }
    set { postNrValue = value;
        Changed = true;
    }
}
```

```
public string Woonplaats
{
    get { return woonplaatsValue; }
    set { woonplaatsValue = value;
        Changed = true;
    }
}

public Leverancier(int nLevNr, String nNaam, String nAdres, String
                    nPostNr, String nWoonplaats)
{
    LevNr = nLevNr;
    Naam = nNaam;
    Adres = nAdres;
    PostNr = nPostNr;
    Woonplaats = nWoonplaats;
    Changed = false;
}
```

### De functie om de leveranciers te wijzigen (in TuinDbManager.cs)

```
public List<Leverancier> SchrijfWijzigingen(List<Leverancier> leveranciers)
{
    var manager = new TuinDbManager();
    List<Leverancier> nietLevs = new List<Leverancier>();

    using (var conLeveranciers = manager.GetConnection())
    {
        using (var comUpdate = conLeveranciers.CreateCommand())
        {
            comUpdate.CommandType = CommandType.Text;
            comUpdate.CommandText = "update leveranciers set Naam = @naam,Adres =
@adres, PostNr = @postnr, Woonplaats = @woonplaats where LevNr = @levnr";

            var parNaam = comUpdate.CreateParameter();
            parNaam.ParameterName = "@naam";
            comUpdate.Parameters.Add(parNaam);

            var parAdres = comUpdate.CreateParameter();
            parAdres.ParameterName = "@adres";
            comUpdate.Parameters.Add(parAdres);

            var parPostNr = comUpdate.CreateParameter();
            parPostNr.ParameterName = "@postnr";
            comUpdate.Parameters.Add(parPostNr);

            var parWoonplaats = comUpdate.CreateParameter();
            parWoonplaats.ParameterName = "@woonplaats";
            comUpdate.Parameters.Add(parWoonplaats);

            var parLevNr = comUpdate.CreateParameter();
            parLevNr.ParameterName = "@levnr";
            comUpdate.Parameters.Add(parLevNr);

            conLeveranciers.Open();
            foreach (var eenLeverancier in leveranciers)
            {
                try
                {
                    parNaam.Value = eenLeverancier.Naam;

```

```
        parAdres.Value = eenLeverancier.Adres;
        parPostNr.Value = eenLeverancier.PostNr;
        parWoonplaats.Value = eenLeverancier.Woonplaats;
        parLevNr.Value = eenLeverancier.LevNr;
        if (comUpdate.ExecuteNonQuery() == 0)
        {
            nietLevs.Add(eenLeverancier);
        }
    }
    catch (Exception)
    {
        nietLevs.Add(eenLeverancier);
    }
} // foreach
} // comUpdate
} // conLeveranciers
return nietLevs;
}
```

Deze functie geeft een fout als je een leverancier wijzigt waarbij je het postnummer leegmaakt.

## 22.7 Doorvoeren naar de database

Alles doorvoeren naar de database doen we bij de Closing\_Window. Dit event wordt bij om het even welke manier van sluiten altijd doorlopen.

Voeg eerst in de code de lijst GewijzigdeLeveranciers toe, waar je ook OudeLeveranciers en NieuweLeveranciers hebt gedeclareerd.

```
public List<Leverancier> GewijzigdeLeveranciers = new List<Leverancier>();
```

Om zeker te zijn dat de huidige Row (die eventueel een nieuwe rij kan zijn) met zijn laatste gegevens te bewaren, moet je een *CommitEdit* voor de huidige rij doorvoeren.

Op het moment dat alles naar de database moet worden geschreven, gaan we de toegevoegde en verwijderde gegevens gebruiken die we hebben bijgehouden in de aparte lijsten. Voor de gewijzigde gegevens gaan we de lijst van leveranciers overlopen en testen op de Changed property.

Na het opslaan wordt de lijst terug opgeladen vanuit de database om ze onder andere de LevNr van de nieuwe leveranciers te tonen.

```
private void Window_Closing(object sender, System.ComponentModel.CancelEventArgs e)
{
    if (MessageBox.Show("Wilt u alles wegschrijven naar de database ?", "Opslaan",
    MessageBoxButton.YesNo, MessageBoxImage.Question, MessageBoxResult.Yes) ==
    MessageBoxResult.Yes)
    {
        leverancierDataGrid.CommitEdit(DataGridEditingUnit.Row, true);
        var manager = new TuinDbManager();
        List<Leverancier> resultaatLevs = new List<Leverancier>();
        StringBuilder nietgoed = new StringBuilder();
        StringBuilder welgoed = new StringBuilder();

        if (OudeLeveranciers.Count > 0)
        {
            resultaatLevs = manager.SchrijfVerwijderingen(OudeLeveranciers);
            if (resultaatLevs.Count > 0)
            {
```

```
        foreach (var l in resultaatLevs)
        {
            nietgoed.Append("Niet verwijderd: " + l.LevNr + " : " + l.Naam +
" niet\n");
        }
    }
    welgoed.Append(OudeLeveranciers.Count - resultaatLevs.Count + "
leverancier(s) verwijderd in de database\n");
}

resultaatLevs.Clear();
if (NieuweLeveranciers.Count > 0)
{
    resultaatLevs = manager.SchrijfToevoegingen(NieuweLeveranciers);
    if (resultaatLevs.Count > 0)
    {
        foreach (var l in resultaatLevs)
        {
            nietgoed.Append("Niet toegevoegd: " + l.LevNr + " : " + l.Naam +
" niet\n");
        }
    }
    welgoed.Append(NieuweLeveranciers.Count - resultaatLevs.Count + "
leverancier(s) toegevoegd aan de database\n");
}

foreach (Leverancier l in leveranciersOb)
{
    if ((l.Changed == true) && (l.LevNr != 0))
    {
        GewijzigdeLeveranciers.Add(l);
        l.Changed = false;
    }
}

resultaatLevs.Clear();
if (GewijzigdeLeveranciers.Count > 0)
{
    resultaatLevs = manager.SchrijfWijzigingen(GewijzigdeLeveranciers);
    if (resultaatLevs.Count > 0)
    {
        foreach (var l in resultaatLevs)
        {
            nietgoed.Append("Niet gewijzigd: " + l.LevNr + " : " + l.Naam +
" niet\n");
        }
    }
    welgoed.Append(GewijzigdeLeveranciers.Count - resultaatLevs.Count + "
leverancier(s) gewijzigd in de database\n");
}
}

MessageBox.Show(nietgoed.ToString() + "\n\n" + welgoed.ToString(), "Info",
MessageBoxButton.OK);

OudeLeveranciers.Clear();
NieuweLeveranciers.Clear();
GewijzigdeLeveranciers.Clear();

System.Windows.Data.CollectionViewSource leverancierViewSource =
((System.Windows.Data.CollectionViewSource)
(this.FindResource("leverancierViewSource")));
leveranciersOb = manager.GetLeveranciers();
leverancierViewSource.Source = leveranciersOb;
}
```

}

## 22.8 Een filter voor de postcodes

Met een combobox in de toolbar willen we enkel de leveranciers tonen met de geselecteerde postcode. Als de combobox “alles” weergeeft, tonen we alle leveranciers.

In het Window\_Loaded event voegen we onderaan de code toe om de ComboBox op te vullen:

```
var nummers = (from l in leveranciersOb orderby l.PostNr
               select l.PostNr.ToString()).Distinct().ToList();
nummers.Insert(0, "alles");
comboBoxPostnummers.ItemsSource = nummers;
comboBoxPostnummers.SelectedIndex = 0;
```

Als er een verandering in de ComboBox is, wordt de Filter al naar gelang de keuze aangepast:

```
private void comboBoxPostnummers_SelectionChanged(object sender,
SelectionChangedEventArgs e)
{
    if (comboBoxPostnummers.SelectedIndex != 0)
        leverancierDataGrid.Items.Filter =
            new Predicate<object>(PostnummerFilter);
    else
        leverancierDataGrid.Items.Filter = null;
}

public bool PostnummerFilter(object lev)
{
    Leverancier l = lev as Leverancier;
    bool result = (l.PostNr == comboBoxPostnummers.SelectedValue.ToString());
    return result;
}
```

## 23 UITWERKING OPGAVE 11: MULTI-USER PROGRAMMA

Voeg aan de tabel Leveranciers van de Sql Server Database TuinCentrum een kolom Versie toe met als DataTypeTimeStamp.

Klik daarvoor rechts op de tabel Leveranciers in de Server Explorer van de Visual Studio. Kies voor Open Table Definition. Voeg een kolom Versie toe met als DataTypeTimeStamp.

### 23.1 Een eerste WPF-applicatie

Het timestamp datatype is een array van bytes. (byte[])

In onze class Leverancier.cs vangen we dit op in een object.

```
public class Leverancier
{
    private int LevNrValue;
    private String NaamValue;
    private String AdresValue;
    private String PostNrValue;
    private String WoonplaatsValue;
    private Object versieValue;

    ...

    public Object Versie
    {
        get { return versieValue; }
        set { versieValue = value; }
    }
    ...

    public Leverancier(int nLevNr, String nNaam, String nAdres,
                       String nPostNr, String nWoonplaats, Object nVersie)
    {
        LevNr = nLevNr;
        Naam = nNaam;
        Adres = nAdres;
        PostNr = nPostNr;
        Woonplaats = nWoonplaats;
        Changed = false;
        Versie = nVersie;
    }
    ...
}
```

In de class TuinDbManager wordt de method GetLeveranciers() :

```
public ObservableCollection<Leverancier> GetLeveranciersBeginNaam(String beginNaam)
{
    ...
    using (var rdrLeveranciers = comLeveranciers.ExecuteReader())
    {
        Int32 leverancierNrPos = rdrLeveranciers.GetOrdinal("LevNr");
        Int32 naamPos = rdrLeveranciers.GetOrdinal("Naam");
        Int32 adresPos = rdrLeveranciers.GetOrdinal("Adres");
        Int32 postcodePos = rdrLeveranciers.GetOrdinal("PostNr");
        Int32 gemeentePos = rdrLeveranciers.GetOrdinal("Woonplaats");
        var versiePos = rdrLeveranciers.GetOrdinal("Versie");
        while (rdrLeveranciers.Read())
```

```
        {
            leveranciers.Add(new Leverancier
(rdrLeveranciers.GetInt32(leverancierNrPos),
rdrLeveranciers.GetString(naamPos),
rdrLeveranciers.GetString(adresPos),
rdrLeveranciers.GetString(postcodePos),
rdrLeveranciers.GetString(gemeentePos),
rdrLeveranciers.GetValue(versiePos));
        }
    } // using rdrLeveranciers
} // using comLeveranciers
} // using conPlanten
return leveranciers;
}
```

Als een record gewijzigd wordt, wordt de Versie automatisch verhoogd. In de method SchrijfWijzigingen verandert dus enkel de CommandText.

```
comUpdate.CommandText = "update leveranciers set Naam=@naam,Adres=@adres,
PostNr=@postnr, Woonplaats=@woonplaats where LevNr=@levnr and
Versie=@versie";
```

De parameter wordt ook toegevoegd:

```
...
var parVersie = comUpdate.CreateParameter();
parVersie.ParameterName = "@versie";
comUpdate.Parameters.Add(parVersie);

conLeveranciers.Open();
foreach (var eenLeverancier in leveranciers)
{
    try
    {
        parNaam.Value = eenLeverancier.Naam;
        .
        .
        .
parVersie.Value = eenLeverancier.Versie;
        if (comUpdate.ExecuteNonQuery() == 0)
    }
}
```

## 23.2 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 *WPFOpgave10.xaml* van *AdoWPF* in de Solution Explorer en kies voor *Copy*.

Klik met de rechtermuisknop op *AdoWPF2* en kies *Paste*.

Klik met de rechtermuisknop op de map *Images* 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 het DLL-project.

Klik met de rechtermuisknop op *AdoWPF2* in de Solution Explorer, *Add, Reference*. Kies bij Solutions, Projects voor het DLL-project en klik op OK.

In App.xaml van *AdoWPF2* zet je *WPFOpgave10.xaml* als startscherm.

In App.config voeg je de connectionString toe:

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

Zet AdoWPF2 als StartUp Project en test even uit.

### 23.3 Exceptions

In de method SchrijfWijzigingen in de TuinDbManager wijzig je de code als volgt:

```
...
conLeveranciers.Open();
foreach (var eenLeverancier in leveranciers)
{
    parNaam.Value = eenLeverancier.Naam;
    parAdres.Value = eenLeverancier.Adres;
    parPostNr.Value = eenLeverancier.PostNr;
    parWoonplaats.Value = eenLeverancier.Woonplaats;
    parLevNr.Value = eenLeverancier.LevNr;
    parVersie.Value = eenLeverancier.Versie;
    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.

### 23.4 Conflicten

Klik rechts 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.

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 foutbericht.

## 24 COLOFON

**Sectorverantwoordelijke:**

**Cursusverantwoordelijke:** Jean Smits

**Medewerkers:** Chris Van Loon

Veerle Smet

Steven Lucas

Hans Desmet

**Versie:** September 2017