# Basis standalone

Realiseren

hoofdstuk

7

# Selecteren en weergeven van gegevens

# Algemene informatie

| Onderwerp | Selecteren en weergeven van gegevens |
|---|---|
| Leerdoel(en) | 1. De student kent het SQL SELECT commando en kan het toepassen.<br>2. De student kan scalaire functies toepassen<br>3. De student kan uitleggen wat een VIEW in SQL is<br>4. De student kan een view maken en aanpassen.<br>5. De student programmeert een Model<br>6. De student programmeert een Controller<br>7. De student programmeert een View<br>8. De student koppelt deze 3 componenten aan elkaar om een werkende applicatie te krijgen. |
| Vereiste voorkennis | 1. Student kent de stof uit readers 1 t/m 6 van thema 7. |
| Kwalificatiedossier | ☐ B1-K1-W1: Plant werkzaamheden en bewaakt de voortgang<br>☐ B1-K1-W2: Ontwerpt software<br>☒ B1-K1-W3: Realiseert (onderdelen van) software<br>☐ B1-K1-W4: Test software<br>☐ B1-K1-W5: Doet verbetervoorstellen voor de software<br><br>☐ B1-K2-W1: Voert overleg<br>☐ B1-K2-W2: Presenteert het opgeleverde werk<br>☐ B1-K2-W3: Reflecteert op het werk |

# Inhoudsopgave


Algemene informatie ..... 2
Inhoudsopgave ..... 3
Introductie ..... 4
Inhoud ..... 4
- Selecteren van gegevens ..... 4
- Scalaire functies ..... 6
  - Werken met datums ..... 6
- Views ..... 10
  - Het aanpassen van een view ..... 11
- Model bouwen ..... 12
- Controller bouwen ..... 12
  - Using System.Data.SqlClient ..... 14
  - Connectionstring via ConfigurationManager ..... 14
  - SQLConnection / Using ..... 15
  - SQLCommand ..... 16
  - ExecuteReader ..... 16
  - Wegschrijven naar een list ..... 17
- View bouwen ..... 20
  - List weergeven in een listview inclusief tag ..... 22

## Introductie

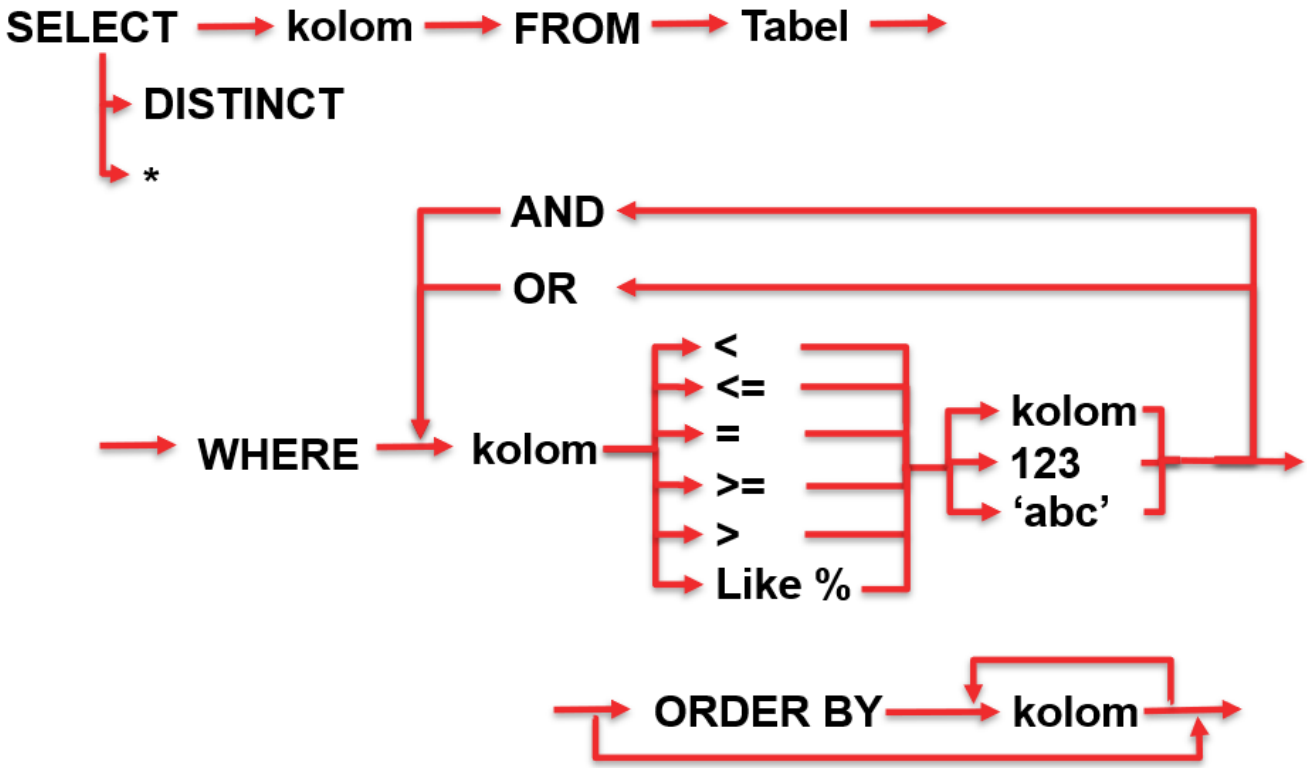
Binnen een applicatie maak je gebruik van gegevens die je wilt weergeven, toevoegen, aanpassen en/of verwijderen. In de praktijk wordt deze data opgeslagen in een database. Deze database dien je vervolgens aan te spreken in je code en vervolgens maak je de vertaalslag van de data die je binnen krijgt (via je controller) naar een weergave van deze data (ofwel de view) voor de gebruiker.

## Inhoud

### Selecteren van gegevens

In thema 4 hebben jullie voor het eerst kennis gemaakt met SQL-commando's. Een van de commando's die jullie daar hebben leren gebruiken is het SELECT statement. Dit (DML-statement) wordt gebruikt voor het raadplegen van gegevens uit een SQL-database. Elk statement heeft een syntax. Dit is de schrijfwijze van het statement. Hieronder staat de vereenvoudigde syntax van het select-statement weergegeven.

Select-statement




Gegevens selecteren doe je dus met het SELECT statement. Dat begint ALTIJD met het woordje SELECT. Vervolgens heb je wat keuzes (zoals je hierboven kunt zien). Je kan aangeven ALLE kolommen te willen zien door achter SELECT een * te plaatsen. Je kan aangeven dat je alleen unieke waarden wil door achter SELECT het woordje DISTINCT te plaatsen. Je kan aangeven

welke kolommen je wilt ophalen, door de kolomnamen achter SELECT te plaatsen. Een paar voorbeelden:

```
-- select alle kolommen uit de tabel users
SELECT *
FROM users

-- toon de unieke waarden uit de
-- kolom first_name van de tabel users
SELECT DISTINCT first_name
FROM users

-- geef de info van de kolommen
-- first_name en last_name uit de
-- tabel users
SELECT first_name, last_name
FROM users
```

Je kan informatie die je opvraagt ook filteren. Hiermee beperk je het aantal rijen in je resultaat. Je kan bijvoorbeeld alle klanten uit een tabel tonen, maar je kan er ook voor kiezen om alleen de klanten uit Den Bosch te tonen. Dit filteren doe je met de WHERE clausule in het statement.

Je kan hierbij een flink aantal opties gebruiken, zie ook de afbeelding hierboven. We gaan in deze reader er vanuit dat de optie =, <, <=, >, >= inmiddels duidelijk zijn. Kijk anders nog even de reader van thema 4 door. De optie LIKE herhalen we hier nog even. Met de optie LIKE kun je namelijk van zogenaamde wildcards gebruik maken.

Soms wil je zoeken naar records waarvan een deel van een kolomwaarde een bepaald woord bevat. Of je wil bijvoorbeeld alle achternamen die beginnen met een W in beeld krijgen. Hoe doe je dat dan? Daarvoor gebruik je LIKE en een wildcard. We gebruiken hier wildcard %.
Deze % vervangt tijdens het zoeken 0, 1 of meerdere karakters.

Dus:

`WHERE achternaam LIKE 'W%'` levert alle rijen op waarin de achternaam begin met een W. Immers, het % teken staat voor een aantal willekeurige karakters. Dit kan dus Willems zijn, maar ook W (al is dat natuurlijk een rare achternaam.

Voorbeeld van verschillende manieren waarop je een wildcard kunt gebruiken.

| | |
|---|---|
| `WHERE achternaam LIKE 'W%'` | Alle rijen waarbij de achternaam start met een W |
| `WHERE achternaam LIKE '%N'` | Alle rijen waarbij de achternaam eindigt op een N |
| `WHERE achternaam LIKE '%Q%'` | Alle rijen waar in de achternaam een Q zit |

 Je kunt nu **oefening 7.1** maken.

# Scalaire functies

We gaan in deze reader je SELECT statement enigszins uitbreiden. Dit gaan we doen met de zogeheten scalaire functies. Scalaire functies worden gebruikt om bewerkingen uit te voeren. Een scalaire functie heeft als invoer nul, één of meer zogenaamde parameters. De waarde van een scalaire functie is afhankelijk van de waarden van de parameters.

Het probleem van scalaire functies (kortweg "functies") is echter dat ze van platform tot platform verschillend kunnen zijn. Zo zal je bijvoorbeeld op een MySQL platform andere functies tegenkomen dan op een SQLServer platform.

We gaan in deze reader uit van de functies die je kunt gebruiken in een MS SQL Server. De voorbeelden hieronder zijn uitgewerkt op de Northwind database welke je bij de module computervaardigheden hebt gekregen.

## Werken met datums

In deze paragraaf ga je met een aantal functies werken die betrekking hebben op datum en tijd. Bij deze functies wordt soms gebruik gemaakt van de toevoegingen Cast en Convert.

 Ga voor meer informatie naar http://msdn.microsoft.com/en-us/library/ms187928.aspx

*GETDATE()*

De Getdate-functie geeft de systeemdatum van de server. Je moet gebruik maken van het Select-statement om dit zichtbaar te maken.

```
SELECT GETDATE()
```

Zo is het ook mogelijk om het verschil tussen twee datums te laten uitrekenen:

```
SELECT hiredate - GETDATE()
FROM employees
```

100 %

Results | Messages

| | (No column name) |
|---|---|
| 1 | 1870-02-10 15:23:48.820 |
| 2 | 1870-05-26 15:23:48.820 |
| 3 | 1870-01-11 15:23:48.820 |
| 4 | 1871-02-12 15:23:48.820 |
| 5 | 1871-07-29 15:23:48.820 |
| 6 | 1871-07-29 15:23:48.820 |
| 7 | 1871-10-14 15:23:48.820 |
| 8 | 1871-12-15 15:23:48.820 |
| 9 | 1872-08-26 15:23:48.820 |

Het resultaat is wellicht anders dan je had verwacht. SQL laat het verschil niet in dagen zien maar met een datum. Om dit probleem op te vangen moet je gebruik maken van Cast.

Als we de query nu zo aanpassen zoals hieronder is weergegeven en nogmaals uit voeren krijgen we de juiste resultaten. Het aantal dagen tussen de indienstdatum en de huidige datum. Omdat alle data hebben voor de huidige datum liggen zijn alle getallen negatief. Dit kun je oplossen door de som om te keren (dus `GETDATE() - hiredate`).

```
SELECT CAST(hiredate - GETDATE() as INT)
FROM employees
```

*DAY()*
Met de functie DAY() kun je de dag van de maand uit een datum halen.

```
SELECT DAY(hiredate), hiredate
FROM Employees
```

*MONTH()*
Met de functie MONTH() kun je het maandnummer uit een datum halen.

```
SELECT MONTH(hiredate), hiredate
FROM Employees
```

*DATENAME(Weekday, …)*
De functie datename(weekday, ...) levert de naam van de dag van de week op.

```
SELECT DATENAME(WEEKDAY, hiredate), hiredate
FROM Employees
```

Het eerste argument van deze functie (dit argument noem je datepart) kan naast weekday nog meer waardes bevatten. Deze vind je hieronder:

| **Waardes datepart** | **Afkorting** |
|---|---|
| year | yy, yyyy |
| quarter | qq, q |
| month | mm, m |
| dayofyear | dy, y |
| day | dd, d |
| week | wk, ww |
| weekday | dw, w |
| hour | hh |
| minute | mi, n |
| second | ss, s |
| millisecond | ms |
| microsecond | mcs |
| nanosecond | ns |
| TZoffset | tz |
| ISO_WEEK | ISOWK, ISOWW |

*YEAR()*
haalt het jaartal uit een gegeven datum

```
SELECT YEAR(hiredate), hiredate
FROM Employees
```

*CONVERT()*
Door gebruik te maken van de functie Convert, is het mogelijk om de opmaak van de datum aan te passen. Je zult wel gemerkt hebben dat de datumopmaak afhankelijk is van de landinstellingen van het operating system. Toch wil je vaak dat de datum altijd maar op één bepaalde manier wordt weergegeven.

```
SELECT CONVERT(VARCHAR(20),GETDATE(),103)
```

Je ziet dat de functie convert drie parameters heeft. De eerste is het datatype waarna er geconverteerd moet worden. In dit geval naar een tekst (varchar). Het tweede argument is de waarde die omgezet moet worden en als derde en laatste geven we het formaat door wat gebruikt moet worden. Een tabel met daarin mogelijke waardes voor de laatste parameter (die het formaat ingeeft) is te vinden in de link aan het begin van dit hoofdstuk.

Als we nu uit de Employee tabel van de Northwind database alle personeelsleden willen hebben met geboortedatum en indienstdatum maar dan in het Nederlands formaat kunnen we dat als volgt doen:

```
SELECT
    EmployeeID,
    LastName,
    FirstName,
    CONVERT(VARCHAR(10),BirthDate,105) as Birthdate,
    CONVERT(VARCHAR(10),HireDate,105) as Hiredate,
    HomePhone
FROM Employees
```

Results | Messages

| EmployeeID | LastName | FirstName | Birthdate | Hiredate | HomePhone |
|---|---|---|---|---|---|
| 1 | Davolio | Nancy | 08-12-1948 | 01-05-1992 | (206) 555-9857 |
| 2 | Fuller | Andrew | 19-02-1952 | 14-08-1992 | (206) 555-9482 |
| 3 | Leverling | Janet | 30-08-1963 | 01-04-1992 | (206) 555-3412 |
| 4 | Peacock | Margaret | 19-09-1937 | 03-05-1993 | (206) 555-8122 |
| 5 | Buchanan | Steven | 04-03-1955 | 17-10-1993 | (71) 555-4848 |
| 6 | Suyama | Michael | 02-07-1963 | 17-10-1993 | (71) 555-7773 |
| 7 | King | Robert | 29-05-1960 | 02-01-1994 | (71) 555-5598 |
| 8 | Callahan | Laura | 09-01-1958 | 05-03-1994 | (206) 555-1189 |
| 9 | Dodsworth | Anne | 27-01-1966 | 15-11-1994 | (71) 555-4444 |

Naast functies die betrekking hebben op data, zijn er ook scalaire functies die betrekking hebben op tekst. De namen van de functies spreken meestal voor zich en ken je wellicht ook al uit andere programmeertalen. Hieronder een overzicht en een kleine toelichting wat de functie doet.

*UPPER()*
Zet de gegeven tekst om in hoofdletters

*LOWER()*
Zet de gegeven tekst om in kleine letters

*LEN()*
Geeft als resultaat het aantal tekens in de meegegeven tekst

*REVERSE()*
Draait alle letters in de tekst om. Dus het laatste karakter komt als eerste, het een na laatste karakter als tweede etc.

*CHARINDEX()*
Geeft als waarde de index waar een karakter als eerste voorkomt. Dus bijvoorbeeld:

```
CHARINDEX(' ', 'Koning Willem 1 College')
```

Levert de waarde 7 op.

Terwijl

```
CHARINDEX('2', 'Koning Willem 1 College')
```

Levert de waarde 0 op, immers het karakter komt niet voor in de tekst.

*LEFT()*
Geeft als resultaat het aantal opgegeven tekens uit de tekst vanaf het begin van de tekst. Dus bijvoorbeeld:

```
LEFT('Koning Willem 1 College', 6)
```

Levert de waarde 'Koning' op.

RIGHT()

Zie de beschrijving bij LEFT, alleen begint hij nu aan het einde van de tekst en telt hij terug.

```
RIGHT('Koning Willem 1 College', 7)
```

Levert de waarde 'College' op.

*SUBSTRING()*
Geeft een gedeelte van een tekst terug. Dus bijvoorbeeld:

```
SUBSTRING('Koning Willem 1 College', 1, 6)
```

levert de waarde 'Koning' op.

Maar je kan deze functies natuurlijk ook combineren!

```
SUBSTRING('Koning Willem 1 College', CHARINDEX(' ', 'Koning Willem 1 College') + 1, LEN('Koning Willem 1 College'))
```

Levert de waarde 'Willem 1 College' op.

*IIF()*
Met de IIF functie kun je een if-then-else constructie opbouwen. Als de eerste paramater waar is, levert hij de twee paramater als resultaat, ander is het resultaat de derde parameter.

Dus bijvoorbeeld:

```
IIF(CHARINDEX(' ', 'Koning Willem 1 College')>0, 'Meerdere woorden', 'Een woord')
```

Levert de waarde 'Meerdere woorden' op.

 Je kunt nu **oefening 7.2** maken.

# Views

Een view is een virtuele dataset (ofwel een virtuele tabel). Een view is dus geen echte tabel, maar je kan er (net als bij een tabel) wel gegevens uit selecteren.

Views worden vaak toegepast in applicaties. Binnen de applicatie hoeven dan geen complexe queries te worden geschreven om bepaalde gegevens goed en eenvoudig op het scherm te krijgen. Daarnaast worden zo de originele tabellen afgeschermd van de gebruiker waardoor zowel de applicatie als de database beter te beveiligen zijn.

Een view maken is in feite een select-statement met een kleine toevoeging. Je moet hiervoor de onderstaande syntax gebruiken:

```
CREATE VIEW <viewnaam> AS
SELECT (DISTINCT) <veld1>, <veld2>, …, <veldN>
FROM <tabelnaam>
WHERE <veldnaam> operator <waarde>
```

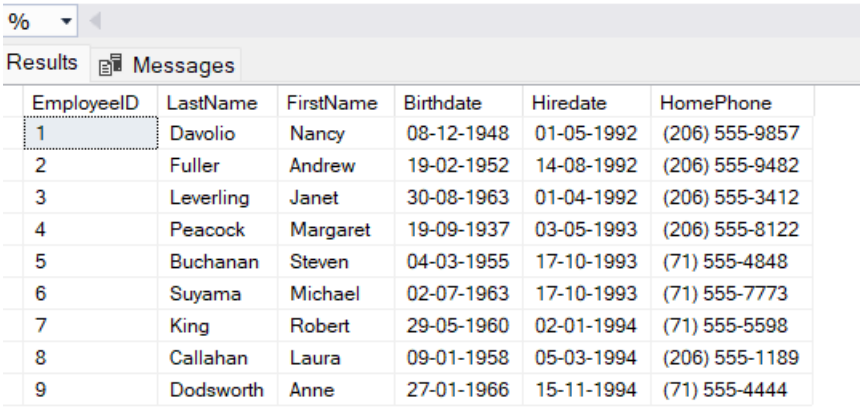
Vervolgens kun je de view gebruiken om gegevens uit te lezen met behulp van het SELECT statement. Dit SELECT statement heeft dezelfde mogelijkheden en syntax zoals je het zou gebruiken met een tabel.

Stel dat we de query uit het vorige hoofdstuk (waarin we de datums omzetten naar een Nederlandse notatie) zouden gebruiken om een view te maken, dan zou dat als volgt gaan:

```
CREATE VIEW vEmployee AS
SELECT
      EmployeeID,
      LastName,
      FirstName,
      CONVERT(VARCHAR(10),BirthDate,105) as Birthdate,
      CONVERT(VARCHAR(10),HireDate,105) as Hiredate,
      HomePhone
FROM Employees
```

Vervolgens kun je met een relatieve simpele query dezelfde resultaten ophalen als dat je met deze query zelf zou kunnen:

```
SELECT * FROM vEmployee
```



%

Results Messages

| EmployeeID | LastName | FirstName | Birthdate | Hiredate | HomePhone |
|---|---|---|---|---|---|
| 1 | Davolio | Nancy | 08-12-1948 | 01-05-1992 | (206) 555-9857 |
| 2 | Fuller | Andrew | 19-02-1952 | 14-08-1992 | (206) 555-9482 |
| 3 | Leverling | Janet | 30-08-1963 | 01-04-1992 | (206) 555-3412 |
| 4 | Peacock | Margaret | 19-09-1937 | 03-05-1993 | (206) 555-8122 |
| 5 | Buchanan | Steven | 04-03-1955 | 17-10-1993 | (71) 555-4848 |
| 6 | Suyama | Michael | 02-07-1963 | 17-10-1993 | (71) 555-7773 |
| 7 | King | Robert | 29-05-1960 | 02-01-1994 | (71) 555-5598 |
| 8 | Callahan | Laura | 09-01-1958 | 05-03-1994 | (206) 555-1189 |
| 9 | Dodsworth | Anne | 27-01-1966 | 15-11-1994 | (71) 555-4444 |

## Het aanpassen van een view

Achteraf kun je op dezelfde wijze een view aanpassen als dat met een tabel mogelijk is.

```
ALTER VIEW <viewnaam> AS
SELECT (DISTINCT) <veld1>, <veld2>, …, <veldN> FROM <tabelnaam> WHERE <veldnaam>
operator <waarde>
```

Stel dat we in de view van hierboven het veld Country willen toevoegen, doen we dat met het volgende commande:

```
ALTER VIEW vEmployee AS
SELECT
      EmployeeID,
      LastName,
      FirstName,
      CONVERT(VARCHAR(10),BirthDate,105) as Birthdate,
      CONVERT(VARCHAR(10),HireDate,105) as Hiredate,
      HomePhone,
      Country
FROM Employees
```

Het resultaat laat nu zien dat het veld land is opgenomen in de view.

```
SELECT * FROM vEmployee
```

Results | Messages

| EmployeeID | LastName | FirstName | Birthdate | Hiredate | HomePhone | Country |
|---|---|---|---|---|---|---|
| 1 | Davolio | Nancy | 08-12-1948 | 01-05-1992 | (206) 555-9857 | USA |
| 2 | Fuller | Andrew | 19-02-1952 | 14-08-1992 | (206) 555-9482 | USA |
| 3 | Leverling | Janet | 30-08-1963 | 01-04-1992 | (206) 555-3412 | USA |
| 4 | Peacock | Margaret | 19-09-1937 | 03-05-1993 | (206) 555-8122 | USA |
| 5 | Buchanan | Steven | 04-03-1955 | 17-10-1993 | (71) 555-4848 | UK |
| 6 | Suyama | Michael | 02-07-1963 | 17-10-1993 | (71) 555-7773 | UK |
| 7 | King | Robert | 29-05-1960 | 02-01-1994 | (71) 555-5598 | UK |
| 8 | Callahan | Laura | 09-01-1958 | 05-03-1994 | (206) 555-1189 | USA |
| 9 | Dodsworth | Anne | 27-01-1966 | 15-11-1994 | (71) 555-4444 | UK |

In de bovenstaande voorbeelden is de view steeds gebruikt met het Select-statement. Het is ook mogelijk om op een view ook de andere DML-statements (Insert, Delete, Update) toe te passen. De view gedraagt zich dan als een normaal tabel. Dit werkt echter alleen als je achter FROM maar één tabel hebt staan. We gaan in de komende weken leren dat je daar ook meer dan één tabel kunt plaatsen, dan werkt alleen SELECT op een VIEW.

 Je kunt nu **oefening 7.3** maken.

# Model bouwen

In de laag Model plaatsen we alle modellen (in de vorm van Classes) van onze entiteiten die we hebben kunnen vinden via de bekende modeleertechnieken. Vanuit je modelleer/data analyse proces heb je een overzicht van je modellen, alsmede een Entiteit Relatie Diagram (wat als basis dient voor je database). In het vorige hoofdstuk heb je geleerd hoe je een model maakt in C#. Zorg dat je jouw code voor de Modellen in en mapje Model zet binnen je project. Zo houd je overzicht!

Als we een model zouden bouwen op basis van de gegevens van een medewerker (employee) uit de view die we in het vorige hoofdstuk hebben aangemaakt, dan zou onderstaande code een voorbeeld kunnen zijn hiervan.

```
public class EmployeeModel
{
    0 references | 0 changes | 0 authors, 0 changes
    public int EmployeeID { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public string LastName { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public string FirstName { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public DateTime Birthdate { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public DateTime Hiredate { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public string HomePhone { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public string Country { get; set; }
}
```

# Controller bouwen

Een controller is de verbinding van je code naar je datalaag. In de vorige reader bestond deze datalaag uit een bestand wat je uitlas. In deze reader gaan we uitleggen wat je moet doen als deze datalaag bestaat uit een SQL Server. De verbinding met de SQL database voeren we uit met behulp van ADO.net.


In onze controllers zorgen ervoor dat minimaal de CRUD functies (zie afbeelding) zijn geïmplementeerd. In deze reader, dit hoofdstuk gaan we aan de slag met de Read, ofwel het uitlezen van gegevens. In onze controller kunnen we dus een methode aanmaken met de naam ReadAll. Deze methode heeft als resultaat een lijst met alle (de naam zegt het al) objecten, van een bepaald type, die uitgelezen worden uit de database.

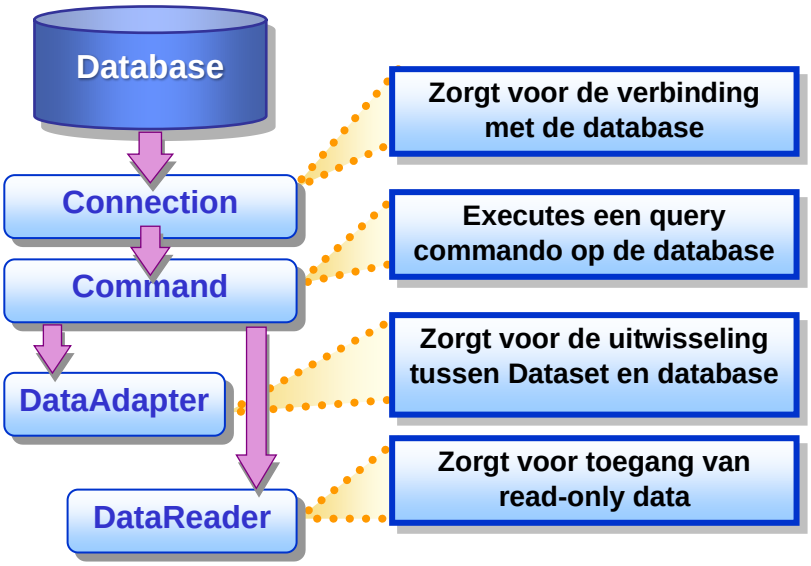
Vandaar dat de methode als volgt gedefinieerd wordt:

```
public List<EmployeeModel> ReadAll()
```

De onderstaande onderwerpen komen aan bod bij het werken met ADO.NET:

| Klasse | Doel |
|---|---|
| Connection | Zorgt voor de verbinding met een database. Aan de hand van parameters kun je opgeven welke database en hoe tot deze toegang kunt krijgen. |
| Command | Hierin komt het SQL-commando dat je wilt uitvoeren op de betreffende database. |
| Dataset | Hierin wordt tijdelijk (totdat het programma wordt afgesloten) de opgevraagde gegevens opgeslagen. |
| Datareader | Leest de gevraagde gegevens uit de database. |
| Dataadapter | Zorgt dat de klasse Command wordt uitgevoerd op de Connection. |

De relatie tussen de bovenstaande onderwerpen is in het onderstaande schema weergegeven:




Het ADO.Net objectenmodel gebruikt **Datasets** om gegevens te bewaren die uit een database zijn opgehaald. Een programma dat een Dataset-object gebruikt, kan SQL-instructies uitvoeren om deze met waarden te vullen en weer te geven.

Om een database via een select query uit te lezen, dien je de volgende zaken (in volgorde) te programmeren:

1. Using System.Data.SqlClient
2. **ConnectionString** definiëren.
3. **SqlConnection** definiëren en starten (via **.Open()** ).
4. **SqlCommand** definiëren (query komt hier), inclusief opgave **SqlParameters**
5. Database uitlezen via **.ExecuteReader()** en het **SqlDataReader** object vullen.
6. Door deze **SqlDataReader** itereren (loopen) via de methode **.Read()** om de rows één voor één uit te lezen en deze in een **List<>** met objecten te plaatsen.

We gaan deze stappen één voor één hieronder kort behandelen. Daarna ga je aan de hand van een voorbeeld deze theorie omzetten naar een werkende applicatie.

### Using System.Data.SqlClient

Alle functionaliteiten rondom het werken met ADO.NET zit verpakt in de library *System.Data.SqlClient*. Deze zit standaard meegeleverd met .NET en kunnen we daardoor gebruiken.

### Connectionstring via ConfigurationManager

Om een SQL Server te kunnen benaderen moet je programma/applicatie weten hoe deze te bereiken is. Dit kunnen we aangeven door middel van een zogenoemde Connectionstring. De term string (tekenreeks) moet je inmiddels bekend voorkomen. Een connectionsstring is dus niets meer of minder daan een tekenreeks met daarin informatie hoe de SQL server te vinden is, hoe er ingelogd dient te worden en welke database er gebruikt dient te worden. Het is cruciaal dat er in de ConnectionString geen fouten gemaakt worden, anders kan je applicatie niet inloggen op de database.

De settings die wij in de ConnectionString gaan gebruiken zijn de volgende:

| Key | Waarde (voorbeeld) | Opmerking |
|---|---|---|
| Data Source | .\SQLEXPRESS | Vertel hier waar de DB zich bevindt en hoe de instance heet. Wij geven in dit voorbeeld de computernaam van onze lokale PC op. |
| Initial Catalog | ExampleDatabase | De naam van de database die gebruikt dient te worden. |
| Integrated Security | TRUE | Integrated Security vertelt dat je via de “Windows Authentication” wilt inloggen. |

De connectionstring kun je in een variabele in jouw programmacode plaatsen. Bijvoorbeeld op deze manier:

```
private static string connectionString = @"Data Source=.\SQLEXPRESS;Initial
Catalog=ExampleDatabases;Integrated Security=True";
```

Een nadeel hiervan is echter dat deze connectie achteraf niet meer aan te passen is. Je moet dus bij het maken van de applicatie rekening houden met de positie van de database. Het is niet ongebruikelijk, sterker nog zeer waarschijnlijk, dat de database waarop die jij gebruikt bij het ontwikkelen een andere database is dan dat de gebruiker van je applicatie gebruikt. Wil je dus

een versie van jouw programma voor de gebruiker maken moet je eerst je programmacode aanpassen. Dat is onwenselijk!

Daarom plaatsen wij de connectionString in een resource van je project, namelijk de app.config.

De app.config is een XML bestand. Dat betekent dat je daarin nodes kun maken met informatie. Voor een connectionstring voeg je onderstaande info toe:

```
<connectionStrings>
  <add name="connectionStringNorthwind"
       connectionString="Data Source=.\SQLEXPRESS;Initial Catalog=Northwind;Integrated Security=True"
       providerName="System.Data.SqlClient"
  />
</connectionStrings>
```

Hiermee geef je aan dat er een connectionstring is met de naam connectionStringNorthwind. In de connectionstring is er een optie connectionString waarin je de verbinding aangeeft richting de database. In dit geval staat de database server op dezelfde machine (dit geeft de . aan) en heet de database instance SQLEXPRESS. De standaard database die geopend wordt is Northwind en er wordt ingelogd met behulp van Windows authenticatie (Integrated Security is True).

Om deze informatie uit de app.config te halen gebruik je het object ConfigurationManager. Die doe je op de volgende manier:

```
string connectionString =
ConfigurationManager.ConnectionStrings["connectionStringNorthwind"].ConnectionString;
```

De variabele connectionString van het type string vullen we met het attribuut ConnectionString uit de lijst ConnectionsStrings. Uit die lijst ConnectionStrings kiezen we de connectiestring met de naam connectionStringNorthwind.

## Die SQLConnection / Using

De connectionstring gebruiken we om, de naam zegt het al, een connectie met een database te maken. Deze connectie maken we met de C# class SQLConnection. Als je een nieuw object aanmaakt van de class SQLConnection geeft je als parameter namelijk de Connection string mee:

```
SqlConnection con = new SqlConnection(connectionString))
```

Met bovenstaand commando maakt je code een nieuw object aan. Object nemen ruimte in het geheugen in beslag tot ze worden weggegooid (disposed in programmeertermen). Dit gebeurt in principe automatisch als je in C# programmeert, maar je kan je programma een handje helpen door aan te geven wanneer het object verwijderd mag worden. Dit doe je met het commando using (lees hier meer). We kunnen bovenstaand commando dus aanpassen naar:

```
// Creeer een nieuw SqlConnection Object met de connectionString
using (SqlConnection con = new SqlConnection(connectionString))
{
}
```

Na de laatste accolade weet je programma dat het de variabele *con* mag opruimen. Dit noemen we in programmeertermen Garbage collection.

## SQLCommand

Nu is het tijd om een commando naar de database te gaan sturen. Een commando voor een database noemen we een query. Dit doen we middels een **SqlCommand.**

Een voorbeeld van een command ofwel query zou kunnen zijn:

```
SELECT * FROM vEmployee
```

Deze query is de eerste parameter als je een nieuw SqlCommand object aanmaakt. Naast de query geef je ook je aangemaakte SqlConnection object mee. Dat kan op deze manier:

```
SqlCommand command = new SqlCommand("SELECT * FROM vEmployee", con);
```

Echter als je op deze manier een langere query mee moet geven wordt het commando wat onoverzichtelijk. Daarom is het beter, overzichtelijker, netter om de query in een aparte variabele te zetten en vervolgens deze variabele mee te geven bij het aanmaken van een object:

```
string sqlQuery = "SELECT * FROM vEmployee"

SqlCommand command = new SqlCommand(sqlQuery, connection))
```

Net als met het connection object is het netjes om aan te geven tot wanneer je het object gebruikt, dus is het goed om ook bij het command een using commando te gebruiken.

```
using (SqlConnection con = new SqlConnection(connectionString))
{
      //Sql statement (select, update en insert)
      string sqlQuery = " SELECT * FROM vEmployee ";
      using (SqlCommand command = new SqlCommand(sqlQuery, con))
      {
      }
}
```

Let op dat je het using commando van je SqlCommand object binnen de using van de SqlConnection object plaatst. Het SqlCommand heeft deze SqlConnection namelijk nodig. Binnen de tweede accolades gaan we nu de code plaatsen waarmee we de gegevens uit de database gaan halen.

## ExecuteReader

We hebben inmiddels een object wat de verbinding naar de database verzorgd en een object wat een query naar een database kan sturen. Het wordt tijd dat we dat ook daadwerkelijk gaan doen. Om te beginnen gaan we zorgen dat het object wat de verbinding verzorgd ook daadwerkelijk die verbinding naar de database opent. Dat doen we met de methode Open(). Dus bijvoorbeeld zo:

```
con.Open();
```

Vervolgens willen we een SELECT statement uitvoeren. Die doen we door het aanroepen van de methode ExecuteReader() van het SqlCommand object. De methode geeft als resultaat gegevens van het datatype **SqlDataReader** terug. Dus maken voor een variabele van dit datatype aan en zetten we daar het resultaat in. Dus bijvoorbeeld zo:

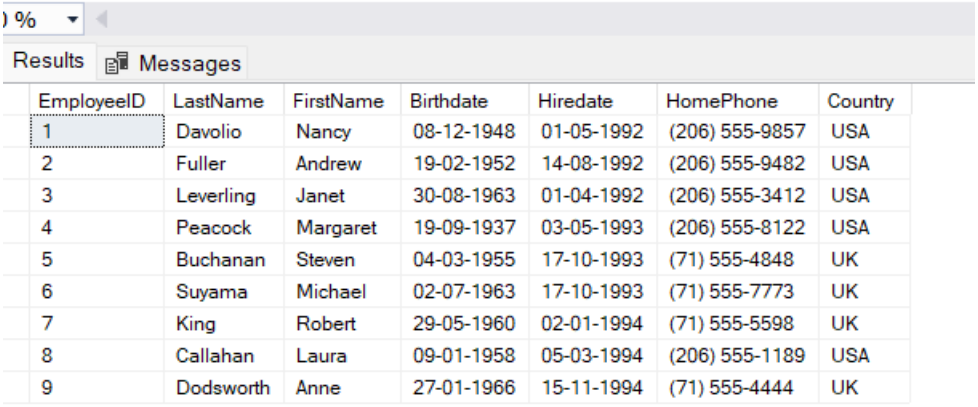
```
SqlDataReader reader = command.ExecuteReader();
```

## Wegschrijven naar een list

We hebben nu in ons object/variabele met de naam *reader* het resultaat van de SQL-query zitten. Het enige dat we nu nog moeten doen, is de resulaten uitlezen. Dit kunnen we doen via een while loop. Zo kunnen we de resultaten één voor één ophalen. Dit betekent dat we 0, 1 of meer rijen op gaan halen. Een datatype in C# dat we hiervoor kunnen gebruiken is de List. Een List is een lijst van objecten van een bepaald type. Dat kan dus een lijst van strings zijn (als volgt: List<string>) maar bijvoorbeeld ook een lijst van EmployeeModel objecten (als volgt: List<EmployeeModel>).

Onze voorbeeldquery vraagt alle rijen uit de view vEmployee op. Dat levert in de SQL Server Management Studio het volgende resultaat op:

```
SELECT * FROM vEmployee
```



Results | Messages

| EmployeeID | LastName | FirstName | Birthdate | Hiredate | HomePhone | Country |
|---|---|---|---|---|---|---|
| 1 | Davolio | Nancy | 08-12-1948 | 01-05-1992 | (206) 555-9857 | USA |
| 2 | Fuller | Andrew | 19-02-1952 | 14-08-1992 | (206) 555-9482 | USA |
| 3 | Leverling | Janet | 30-08-1963 | 01-04-1992 | (206) 555-3412 | USA |
| 4 | Peacock | Margaret | 19-09-1937 | 03-05-1993 | (206) 555-8122 | USA |
| 5 | Buchanan | Steven | 04-03-1955 | 17-10-1993 | (71) 555-4848 | UK |
| 6 | Suyama | Michael | 02-07-1963 | 17-10-1993 | (71) 555-7773 | UK |
| 7 | King | Robert | 29-05-1960 | 02-01-1994 | (71) 555-5598 | UK |
| 8 | Callahan | Laura | 09-01-1958 | 05-03-1994 | (206) 555-1189 | USA |
| 9 | Dodsworth | Anne | 27-01-1966 | 15-11-1994 | (71) 555-4444 | UK |

Je ziet dat er het resultaat bestaat uit 9 rijen (ook wel records genoemd) die elk bestaan uit 7 kolommen.

Om deze rijen uit te lezen starten we met het lezen van de eerste rij. Dat doen we met de methode Read(). Deze methode heeft als resultaat een boolean die aangeeft of er een rij is opgehaald. Dus als het resultaat true is, is er een rij opgehaald en kunnen we de kolommen van die rij uitlezen. Hier kunnen we dus een while loop voor maken:

```
while (reader.Read())

{

	// Code om de gegevens per rij/record uit te lezen…

}
```

Vervolgens kunnen we de kolommen aan door de variabele reader aan te spreken als een List, waarbij we als optie de naam van de kolom opgeven. Zo levert reader[“EmployeeID”] bij de eerste rij de waarde 1 op. En levert reader[“Lastname”] bij de derde rij de waarde Leverling op.

Per rij willen we de waardes natuurlijk opslaan in het geheugen. Dat doen we in een variabele van het type EmployeeModel (die we hebben aangemaakt als Model een paar bladzijdes terug. Dit model ziet er als volgt uit:

```
public class EmployeeModel
{
    0 references | 0 changes | 0 authors, 0 changes
    public int EmployeeID { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public string LastName { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public string FirstName { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public DateTime Birthdate { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public DateTime Hiredate { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public string HomePhone { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public string Country { get; set; }
}
```

We gaan nu per rij een nieuwe variabele van dit type aanmaken. Dat doen we als volgt:

```
EmployeeModel employee = new EmployeeModel();
```

Vervolgens kunnen we de attributen van dit object vullen. Bijvoorbeeld zo:

```
employee.EmployeeID = (int) reader["EmployeeID"];
```

Je ziet dat we het resultaat meteen 'casten' naar het juiste datatype door er (int) voor te zetten. In plaats van *int* kan hier ook b.v. *DateTime*, *string*, *decimal* of *bool* staan.

Zorg dat je op deze manier alle attributen van je object vult.

```
employee.EmployeeID = (int) reader["EmployeeID"];
employee.LastName = (string) reader["Lastname"];
employee.FirstName = (string) reader["Firstname"];
employee.Birthdate = (DateTime) reader["Birthdate"];
…
employee.Country = (string) reader["Country"];
```

Nu is het employee object gevuld met de gegevens uit de rij/record die we aan het inlezen zijn. Deze employee voegen we toe aan een lijst. Deze lijst wordt uiteindelijk het resultaat van de ReadAll methode! Zorg dus dat je in de start van je ReadAll methode een variabele van het type List<modelnaam> aanmaakt. In ons voorbeeld zouden we de volgende lijst aan moeten maken:

```
List<EmployeeModel> employeeList = new List<EmployeeModel>();
```

Aan deze lijst voegen we het zojuist gemaakte en gevulde object employee toe.

```
employeeList.Add(employee);
```

Nadat we alle items uit de database hebben gelezen (op dat moment zal reader.Read() als resultaat False opleveren) hoeven we alleen nog maar het resultaat van de methode gelijk te maken aan de inhoud van onze lijst met objecten.

```
return employeeList;
```

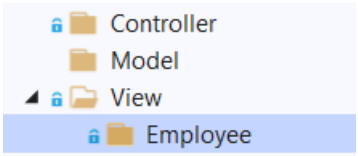
Als we dan globaal kijken hoe we de ReadAll() methode hebben opgebouwd gaat dat als volgt:

```
public List<EmployeeModel> ReadAll()
{
    // 1. Aanmaken List

    // 2. Ophalen connectionstring en maken van een SQLConnection object
    // en deze middels using gebruiken

    // 3. Aanmaken van een SQLCommand object en deze middels using gebruiken

    // 4. Middels while loop alle rijen van de database uitlezen
    // Maak een nieuw object aan en vul dit met de gegevens uit de datase
    // Voeg het object toe aan de List

    // Geef als resultaat van de methode de List uit stap 1 terug.
}
```

 Je kunt nu **oefening 7.4** maken.

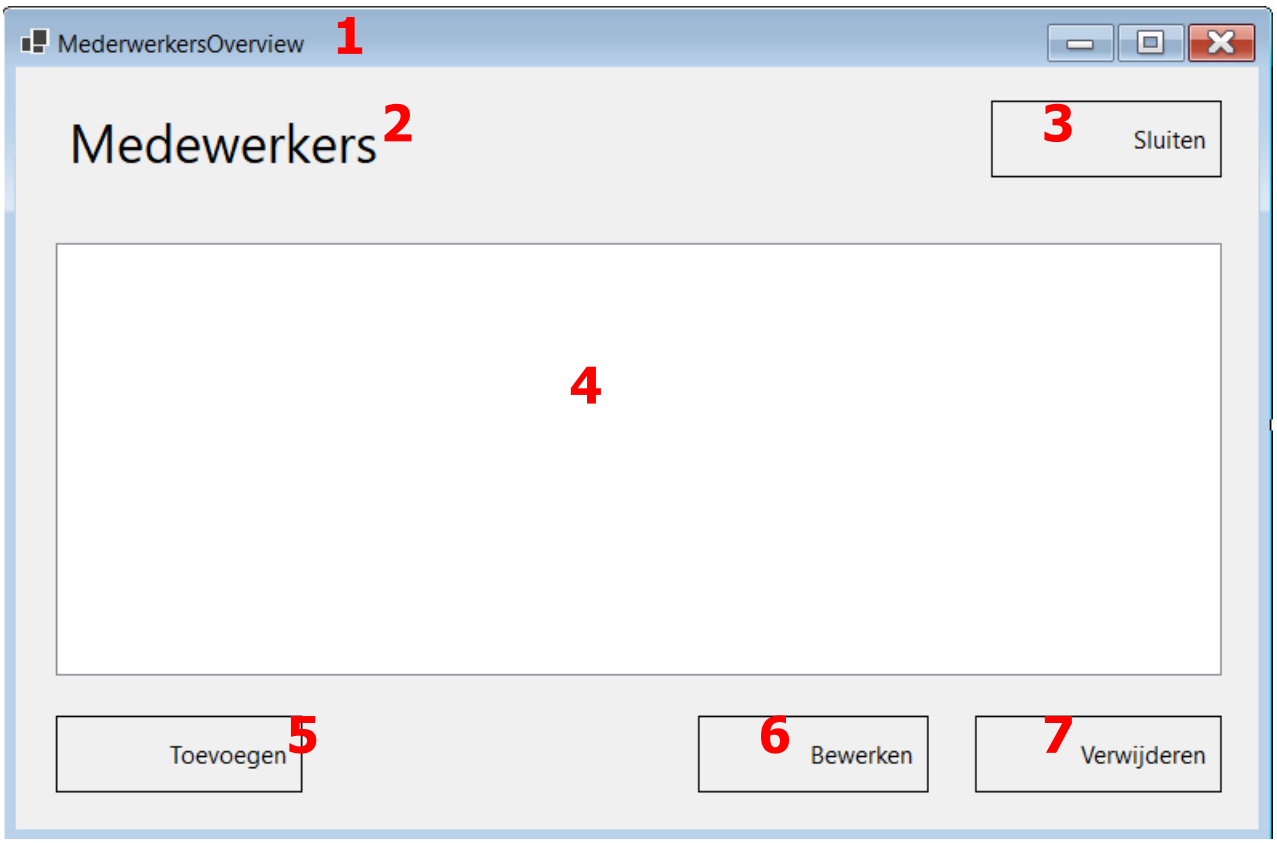
# View bouwen

In het vorige hoofdstuk heb je geleerd hoe je informatie vanuit een database in een variabele van het type List<modelnaam> moet plaatsen. Echter, voor je applicatie en de gebruiker van deze applicatie, is het natuurlijk wel nodig dat deze infomatie ook op het scherm zichtbaar word. Hiervoor maken we in onze applicatie een scherm aan. Schermen voor de gebruiker plaatsen we in de map view in ons project. Daarnaast is het voor het overzicht goed om per model een submapje te maken. Dan kun je alle schermen die met een specifiek model te maken hebben 'groeperen'. Maak dus onder het mapje view een mapje met de naam van het model aan (mocht dit nog niet gebeurd zijn).




In deze map maak je vervolgens een nieuw Form (Windows Forms) aan. Geef dit formulier de naam frm<modelnaam>Overview. Dus in ons voorbeeld zou het worden frmEmployeeOverview.

Binnen iedere applicatie of bedrijf maak je voor applicaties gebruik van een huisstijl. In een huisstijl heb je met elkaar, of bijvoorbeeld met de opdrachtgever, afgesproken hoe een applicatie er uit komt te zien. Binnen deze module hebben we ook een huisstijl. Deze ga je in de komende hoofdstukken steeds beter leren kennen. Voor nu gaan we er vanuit dat een overzicht scherm er op de volgende manier uitziet (de rode cijfers staan natuurlijk niet in je scherm maar worden gebruikt om hieronder instellingen toe te lichten):



Hieronder vind je per nummer de naam van het control wat gebruikt is en de instellingen die zijn aangepast in de standaard waarde van dit control.

| Nummer | Opmerkingen |
|---|---|
| 1 | De naam van het formulier. Geef hier duidelijk aan dat het een overview scherm betreft en wat hij laat zien (in dit geval medewerkers). |
| 2 | Control label<br>Text: Titel van het formulier (entiteit die wordt weergegeven)<br>Font: Segoe UI; 20 pt (die rond hij vaak wat af…)<br>Textalign: MiddleLeft<br>Autosize: True |
| 3 | Control Button<br>Name: btnClose<br>Flatsyle: Flast<br>Tekst: Sluiten<br>Textalign: MiddleRight<br>Width: 150<br>Height: 48 |
| 4 | Control Listview<br>Name: lv<modelnaam in meervoudsvorm> in dit voorbeeld lvEmployees<br>Headerstyle: none |
| 5 | Control Button<br>Name: btnAdd<br>Flatsyle: Flast<br>Tekst: Toevoegen<br>Textalign: MiddleRight<br>Width: 150<br>Height: 48 |
| 6 | Control Button<br>Name: btnEdit<br>Flatsyle: Flast<br>Tekst: Bewerken<br>Textalign: MiddleRight<br>Width: 150<br>Height: 48 |
| 7 | Control Button<br>Name: btnDelete<br>Flatsyle: Flast<br>Tekst: Verwijderen<br>Textalign: MiddleRight<br>Width: 150<br>Height: 48 |

## List weergeven in een listview inclusief tag

Als we deze controls hebben neergezet, hebben de Grafische User Interface (GUI) opgebouwd en is het om de programmacode van het formulier te gaan programmeren. We komen bij deze code door op F7 te drukken. Een andere optie is om met je rechtermuisknop op de header van het formulier te drukken (waar op de vorige pagina een rode 1 in staat) en te kiezen voor 'view code'. Wij gebruiker een derde manier. Hiervoor klikken we op het formulier, vervolgens in het properties venster op het knopje met de bliksem (  ) en dubbelklikken we achter het woordje Load.

Op deze manier maken we een Load event aan (hierover later in je opleiding meer) en krijgen het volgende scherm met code te zien:

```
public frmEmployeeOverview()
{
    InitializeComponent();
}

1 reference | 0 changes | 0 authors, 0 changes
private void frmEmployeeOverview_Load(object sender, EventArgs e)
{

}
```

We kunnen nu code gaan plaatsen in de methode waarvan de naam eindigt op _Load. Deze code wordt iedere keer als het formulier geladen wordt uitgevoerd.

Wat we hier doen zijn eenmalige instellingen. Deze instellingen hebben in dit formulier te maken met de listview. Een listview is een control wat rijen met informatie kan laten zien. Dit kan op heel veel manieren, waarvan wij er een gebruiken. De instellingen hiervoor doen we in deze methode.

Om te beginnen gaan we in de listview kolommen aanmaken. Ieder stukje informatie wat we willen laten zien kom in een eigen kolom. We maken in principe een kolom aan voor ieder attribuut van het model, behalve voor het ID. In dit geval doen we dat zo:

```
lvEmployees.Columns.Add("Achternaam");

lvEmployees.Columns.Add("Voornaam");

lvEmployees.Columns.Add("Geboortedatum");

…

lvEmployees.Columns.Add("Land");
```

Je ziet dat we voor ieder attribuut een kolom maken, maar dat we de benaming in het Nederlands doen. De namen van de kolommen is iets wat de gebruiker van de applicatie ziet en gaat lezen. Vandaar de keuze om hiervoor de Nederlandse taal te gebruiken.

We dienen nog een paar extra aanpassingen aan de listview doen.

```
lvEmployees.FullRowSelect = true;
lvEmployees.View = System.Windows.Forms.View.Details;
lvEmployees.HeaderStyle = ColumnHeaderStyle.Clickable;
```

We stellen in dat we telkens een hele rij willen selecten en dat we details willen laten zien. De headerstyle stellen we in zodat een gebruiker op de header kan klikken.

Vervolgens moeten we zorgen dat er gegevens in de listview komen. Dit vullen van de listview zal op meerdere plekken terugkomen. Daarom zetten we de code niet in deze methode maak maken we hier een aparte methode van. Deze methode gaan we als volgt definieren:

```
private void FillListView()
```

We maken de methode private, aangezien we hem alleen binnen dit formulier nodig hebben. De methode geeft verder geen resultaat terug (daarom void).

Wat moeten we nu doen om de informatie in de listview te krijgen? De informatie die in de listview moet komen halen we uit de database. Daarvoor hebben we hierboven de controller geprogrammeerd. Via de controller halen we de informatie uit de database. Dus hebben we in de functie een object nodig. Dit maken we als volgt aan:

```
EmployeeController employeeController = new EmployeeController ();
```

In deze controller hebben we een methode ReadAll gemaakt. Zoals je wellicht nog weet levert deze methode als resultaat een List op. In dit geval een List<EmployeeModel>. Deze List bevat de gegevens die we willen tonen in de Listview. Dus zorgen we ervoor dat we in deze methode een variabele hebben waarin we deze gegevens opslaan. Dit kan op de volgende manier:

```
List<EmployeeModel> employeeList = employeeController.ReadAll();
```

We maken dus een variabele employeeList van het type List<EmployeeModel> aan en vullen deze variabele met het resultaat van de ReadAll methode van het object employeeController. Het resultaat is dat in deze variabele alle gegevens uit de database komen te staan. Deze gaan we, item voor item, toevoegen aan de listview.

Voor we dat gaan doen, willen we eventueel bestaande gegevens in de listview webhalen. Dat kan met het volgende commando:

```
lvEmployees.Items.Clear();
```

LET OP! Als je in bovenstaande commande het woord Items weg laat, reset je de hele listview en ben je alle instellingen die in de de _Load hierboven hebt gedaan ook kwijt. Dat willen we natuurlijk niet.

Nu gaan we dus item voor item de gegevens uit de List<> toevoegen aan de listview. We weten dat alle variabele in de list hetzelfde type hebben. Daarom kunnen we gebruik maken van het C# commando foreach. Een foreach is een speciale loop waarmee je code uitvoert voor ieder element in een List<>. In ons voorbeeld kan dat als volgt:

```
foreach (EmployeeModel employee in employeeList)
{
      // Hier kun je code schrijven die met elke employee uit de employeeList
      // iets doet. Wij willen deze employee hier toevoegen aan de listview.
}
```

Item toevoegen aan een Listview doe je met een ListViewItem. Een ListViewItem aanmaken doe je door de volgende code uit te voeren:

```
ListViewItem lvItem = new ListViewItem(employee.LastName);
```

Hier maken we een variabele van het type ListViewItem aan die we lvItem noemen. Je ziet achteraan tussen haakjes een waarde meegegeven. Dit is de waarde dit in de eerste kolom van de listview komt te staan. Natuurlijk willen we in de volgende kolommen ook gegevens tonen. Dat gaat op de volgende manier:

```
lvItem.SubItems.Add(employee.FirstName);
```

Voor iedere extra kolom voeren we deze regel uit. Natuurlijk in de volgorde waarin de kolommen door de listview getoond worden. Let wel op. Zowel voor de eerste kolom, als voor alle andere kolommen mag je alleen strings toevoegen. In ons voorbeeld hierboven zijn zowel Lastname als Firstname al strings en gaat dit zonder problemen. Als je andere waarden hebt moeten deze dus nog worden omgezet naar een string waarde. Voor een datumwaarde zou dat als volgt kunnen:

```
lvItem.SubItems.Add(employee.Birthdate.ToString("dd-MM-yyyy"));
```

voor een waarde van het type in zou .ToString() al volstaan.

Voor het vervolg van de applicatie is het prettig als we de variabele van het type Employeemodel (in onze voorbeeld foreach is dat de variabele employee) kunnen benaderen. Dit kan! De variabele van het type ListViewItem heeft een attribuut met de naam Tag. Dit attribuut kan een object bevatten en ieder model wat we maken is een soort object. Dus kunnen we in het tag object ons object gewoon opslaan! In ons voorbeeld zouden we dat zo doen:

```
lvItem.Tag = employee;
```

Nu zijn we bijna klaar. Het enige wat nog moet gebeuren is ons zojuist aangemaakt en gevulde ListViewItem object toevoegen aan de listview:

```
lvEmployees.Items.Add(lvItem);
```

 Je kunt nu **oefening 7.5** maken.