Access voor beginners - hoofdstuk 25

Handleiding van Helpmij.nl

Auteur: OctaFish

Oktober 2014

Werken met Klassemodules

Tot nu toe heb ik in de cursus Access veel gewerkt met formulieren, en met procedures en functies. Die functies stonden dan ofwel in de module die bij het formulier hoorde, ofwel in een aparte module. De werking van de functies was doorgaans gelijk, met als verschil dat een functie die op een formuliermodule staat alleen binnen dat formulier gebruikt wordt, en een functie op een algemene module overal kan worden aangeroepen.

Ik vertel het een beetje gechargeerd, want het is niet helemaal correct, maar in de praktijk is het wel de meest gangbare manier van werken.

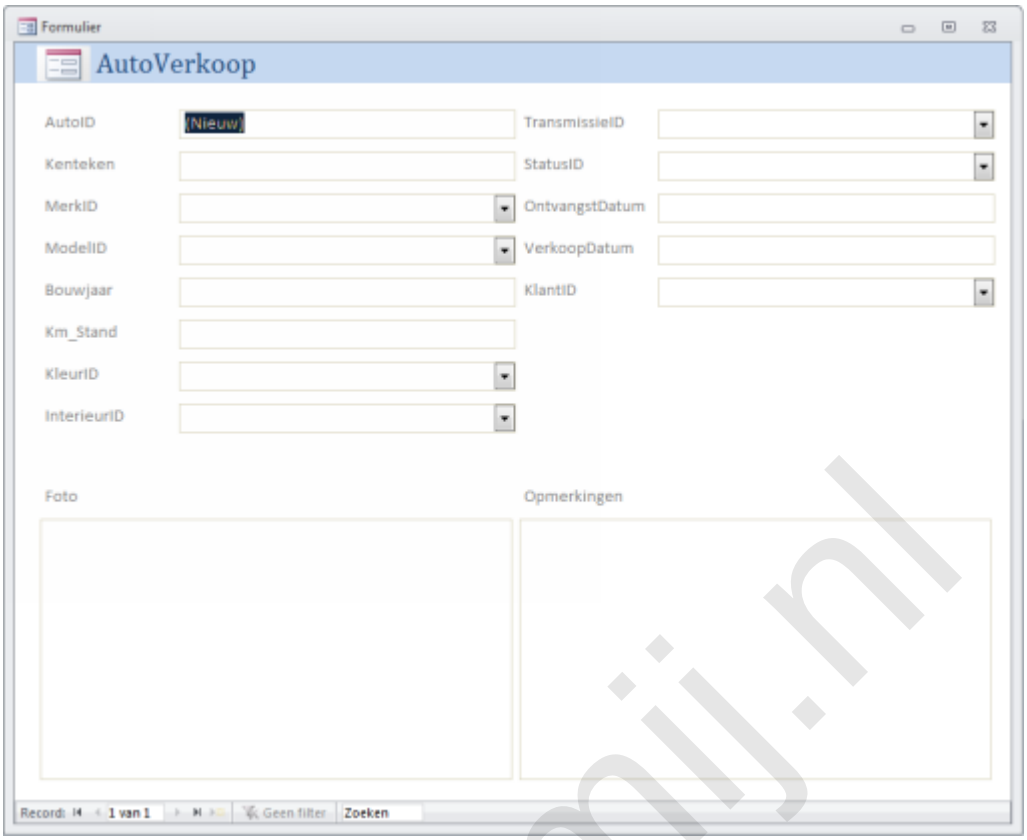
Maar er is nog een derde module variant: de *klassemodule*. En daar gaat dit hoofdstuk (en de komende hoofdstukken) over. Voordat ik daarmee begin geef ik eerst een korte samenvatting van de programmeertechnieken welke we tot nu toe hebben gebruikt.

Wat was het geval? We maakten een (al dan niet gebonden) formulier, en op dat formulier zetten we velden, keuzelijsten en knoppen neer. Vervolgens maakten we gebeurtenissen die bepaalde acties uitvoerden. In dat proces gebruikten we *objecten* (*objects*) zoals het formulier zelf, een keuzelijst met invoervak, een knop en een tekstvak. Al die verschillende objecten hadden *eigenschappen (properties)* en *gebeurtenissen (events)*. Met die eigenschappen deden we dan iets: aan bijvoorbeeld de gebeurtenis <Bij wijzigen> hingen we een actie die een eigenschap van een ander object veranderde.

Tijdens het programmeren hadden we dan veel baat bij het gebruik van de punt om een methode of eigenschap te kiezen. Met Me.RecordSource kon je bijvoorbeeld de Recordbron van een formulier of rapport veranderen. Door het typen van **Me.** kreeg je een lijst met beschikbare *methods* en *properties* die bij het object hoorden. Door na de punt de letter ‘r’ te typen sprong de cursor naar de eerste optie in de lijst die met de letter ‘r’ begon, en daarna kon je de zoekactie verder verfijnen door een volgende letter te typen. Het spreekt bijna vanzelf dat een formulierobject daarbij andere methodes en eigenschappen kent, en laat zien, dan een tekstvak of keuzelijst.

Maar daar houdt het gebruik van objecten niet op: je kunt ook *eigen* objecten definiëren in Access, en die kun je daarna op dezelfde manier gebruiken als de ingebouwde objecten van Access. Je kunt die eigen objecten vervolgens allerlei specificaties meegeven, en eigen gebeurtenissen creëren voor die objecten. Kortom: het is de juiste tijd om aandacht te schenken aan het gebruik van klassemodules!

Als voorbeeld gaan we een klassemodule maken voor een tweedehands autodealer, die (logisch) auto’s verkoopt, en die transacties bijhoudt in een database. Daarbij gebruikt hij een formulier om de verkopen bij te houden. Dat ziet er ongeveer zo uit:



Klassediagrammen

Bij het ontwerpen van klassemodules is het belangrijk om goed bij te houden wat er allemaal moet gebeuren op welke velden in welke tabellen. Daar zijn klassediagrammen een prima hulpmiddel bij. Voor de tabel Verkoop, waar bovenstaand formulier op is gebaseerd, zou het klassediagram er zo uit kunnen zien:

Boven de streep zie je de velden van de tabel tAutoVerkoop met een sleuteltje bij de primaire sleutel. Onder de streep zie je de methodes die je nodig hebt om de tabel te onderhouden. Voor elke methode heb je aparte procedures nodig, die we later behandelen. Zo gebruik je de methode Retrieve als je een formulier vult met gegevens en de methode Update als je een record bijwerkt.

Werken met Properties

Elk object heeft eigenschappen, die we *properties* noemen. Dus ook objecten die we zelf aanmaken. In de objecten die we gaan aanmaken gebruiken we Public variabelen die we moeten kunnen manipuleren. Daarvoor gebruiken we drie technieken: *Property Let*, *Property Get* en *Property Set*.

Property Let

De Property Let procedure is bedoeld om te bepalen wat er moet gebeuren als een waarde wordt toegewezen aan een property. Hierbij kunnen ook bepaalde checks worden uitgevoerd om te controleren of de ingevoerde waarde voldoet aan de gestelde eisen. Bij een telefoonnummer kun je bijvoorbeeld controleren of er geen tekst of spaties zijn ingevoerd. En bij een tekstveld kun je controleren of de ingevoerde tekst niet te lang is.

Property Get

Met Property Get wordt bepaald wat er moet gebeuren als een waarde is opgehaald uit de property. Meestal wordt de waarde aan een variabele toegewezen.

Property Set

De property Set methode wordt alleen gebruikt om te specificeren wanneer een bepaalde waarde is toegewezen aan de property. Deze methode heb je minder vaak nodig, want je zult doorgaans genoeg hebben aan de Propery Let methode.

Als je Properties wilt gebruiken die alleen voor lezen zijn bedoeld, dan heb je aan Property Get genoeg.

Een klassenmodule invoegen

Het invoegen van een klassemodule gaat op dezelfde manier als het invoegen van een normale module. In het VBA venster van je database (<Alt>+<F11>) klik je op <Invoegen>, en vervolgens op <Klassenmodule>. Het beste kun je de klassenmodule vervolgens een herkenbare naam geven, want straks komt die naam terug bij het gebruik van de klasse. En dan is het wel zo handig dat je de klasse daaraan kunt herkennen.

Dat hernoemen van de klassenmodule doe je in de <Eigenschappen> van de module. Als je het Eigenschappen venster niet ziet, druk dan op <F4> of kies in het menu <Beeld> de optie <Venster Eigenschappen>. In het tekstvak <Name> kun je dan de naam veranderen. In dit hoofdstuk gebruik ik de naam clsCar.

Declareren van de variabelen

Het maken van een Klassemodule begint, zoals we inmiddels wel gewend zijn, met het declareren van de variabelen. Omdat we die in een aparte module vastleggen, zijn ze automatisch lokaal. Dat is prima, want de properties worden public gedeclareerd, dus het gebruik van de variabelen komt toch niet verder dan de module.

```
(Algemeen)
Option Compare Database
'------------------------------------------------------
'Declareren van de variabelen
'------------------------------------------------------
Public Event ActionSuccess(strMessage As String)

Dim dblAutoIdNummerVal As Double
Dim intMerkIdVal As Integer
Dim intModelIdVal As Integer
Dim intModelJaarVal As Integer
Dim intKmVal As Integer
Dim intColorIdVal As Integer
Dim intInteriorTypeIdVal As Integer
Dim intInteriorColorIdVal As Integer
Dim intTransmissionTypeIdVal As Integer
Dim intStatusIdVal As Integer
Dim dtDatumOntvangenVal As Date
Dim dtVerkoopDatumVal As Date
Dim intKlantnummerIdVal As Integer
Dim strPhotoVal As String
Dim strSpecialFeaturesVal As String

Public Enum SpecialFeaturesList
    Zonnedak
    Schuifdak
    Boordcomputer
    NavigatieSysteem
    TelefoonAansluiting
    USBAansluiting
    Overig
End Enum
```

Bovenin zie je de declaratie voor de Public Event *ActionSuccess*. Deze wordt later behandeld. En onderin vind je de declaratie voor de Enum *SpecialFeaturesList*. Ook die komt later aan bod. We concentreren ons nu eerst op de properties Let en Get.

Objecten beheren

Voor het object gebruiken we, net als overigens voor alle volgende objecten, de Let en Get properties.

```
Public Property Get AutoIdNummer() As Double
    AutoIdNummer = dblAutoIdNummerVal
End Property

Public Property Let AutoIdNummer(ByVal Value As Double)
    dblAutoIdNummerVal = Value
End Property
```

Je ziet dat de *Property Get* wordt gedefinieerd als Double; dit is logisch, want de variabele is ook als Double gedefinieerd. Wat er gebeurt is eigenlijk heel simpel: de property krijgt de waarde die in de variabele dblAutoIdNummerVal is vastgelegd.

Iets vergelijkbaars zien we bij de *Property Let* gebeuren: hier wordt de variabele dblAutoIdNummerVal gevuld met wat in de property AutoIdNummer is vastgelegd. Verschil met de property Get is, dat de

property Let een parameter heeft (Byval Value As Double) en dat is ook wel logisch, want de property moet zijn waarde natuurlijk ergens vandaan halen.

Laten we er nog een paar bekijken:

```
Public Property Get MerkId() As Integer
    MerkId = intMerkIdVal
End Property

Public Property Let MerkId(ByVal Value As Integer)
    intMerkIdVal = Value
End Property

Public Property Get ModelID() As Integer
    ModelID = intModelIdVal
End Property

Public Property Let ModelID(ByVal Value As Integer)
    intModelIdVal = Value
End Property

Public Property Get ModelJaar() As Integer
    ModelJaar = intModelJaarVal
End Property

Public Property Let ModelJaar(ByVal Value As Integer)
    intModelJaarVal = Value
End Property

Public Property Get KmVal() As Integer
    KmVal = intKmVal
End Property

Public Property Let KmVal(ByVal Value As Integer)
    intKmVal = Value
End Property
```

Het principe is, zoals je kunt zien, voor alle properties gelijk. We hebben variabelen gedeclareerd, en die worden met Let gevuld, en met Get uitgelezen.

Je ziet ook wat het voordeel is van een consequente werkwijze; het voorvoegsel bij de variabele geeft aan wat voor soort variabele het is (dbl = Double, int = Integer) en de naam van de property wordt ook gebruikt in de variabele. Zo kun je altijd zien wat bij elkaar hoort.

De overige properties zien er dan ook als volgt uit:

```
Public Property Get ColorID() As Integer
    ColorID = intColorIdVal
End Property

Public Property Let ColorID(ByVal Value As Integer)
    intColorIdVal = Value
End Property

Public Property Get InteriorTypeID() As Integer
    InteriorTypeID = intInteriorTypeIdVal
End Property

Public Property Let InteriorTypeID(ByVal Value As Integer)
    intInteriorTypeIdVal = Value
End Property

Public Property Get TransmissionTypeID() As Integer
    TransmissionTypeID = intTransmissionTypeIdVal
End Property

Public Property Let TransmissionTypeID(ByVal Value As Integer)
    intTransmissionTypeIdVal = TransmissionTypeID
End Property

Public Property Get StatusID() As Integer
    StatusID = intStatusIdVal
End Property

Public Property Let StatusID(ByVal Value As Integer)
    intStatusIdVal = StatusID
End Property

Public Property Get DatumOntvangen() As Date
    DatumOntvangen = dtDatumOntvangenVal
End Property

Public Property Let DatumOntvangen(ByVal Value As Date)
    dtDatumOntvangenVal = Value
End Property

Public Property Get DatumVerkoop() As Date
    DatumVerkoop = dtDatumVerkoopVal
End Property

Public Property Let DatumVerkoop(ByVal Value As Date)
    dtDatumVerkoopVal = Value
End Property

Public Property Get KlantId() As Integer
    KlantId = intKlantIdVal
End Property

Public Property Let KlantId(ByVal Value As Integer)
    intKlantIdVal = Value
End Property

Public Property Get Photo() As String
    Photo = strPhotoVal
End Property

Public Property Let Photo(ByVal Value As String)
    strPhotoVal = Value
End Property
```

De procedures

De volgende stap is het vastleggen van de uit te voeren procedures. Zoals we in het ontwerp al gezien hebben, zijn dat er vier: Retrieve, Update, Add en Delete. De code daarvoor ziet er voor het moment even zo uit:

```
Public Function Retrieve(ByVal AutoIdNummer) As ADODB.Recordset
    Debug.Print "clsCar.Retrieve methode"
    Debug.Print "Auto ID: " & AutoIdNummer & " is opgehaald"
End Function
```

```
Public Sub Add()
    Debug.Print "clsCar.Add methode"
    Debug.Print "Nieuw record voor AutoIdNummer: " _
    & Me.AutoIdNummer
End Sub
```

```
Public Sub Update()
    Debug.Print "clsCar.Update methode"
    Debug.Print "Record voor AutoIdNummer: " _
        & Me.AutoIdNummer & " is bijgewerkt."
    RaiseEvent ActionSuccess("Updates voor bestaand " _
        & "record successvol bewaard!")
End Sub
```

```
Public Sub Delete()
    Debug.Print "clsCar.Delete methode"
    Debug.Print "Record AutoIdNummer: " _
        & Me.AutoIdNummer & " is verwijderd."
End Sub
```

Voorlopig zie je nog geen echte functionaliteit in de methoden. Voor het moment worden ze alleen alvast aangemaakt, en worden ze getest met het venster Direct waarin je kunt zien wat het resultaat is van de acties.

De functie Retrieve wordt gedefinieerd als een ADODB recordset. We gaan uiteindelijk de gegevens manipuleren, en daarvoor hebben we een connectie nodig met de gegevens. Retrieve heeft (uiteraard) input nodig, want moet wel weten wát er moet worden opgehaald. In dit geval is dat de property AutoIDNummer.

Gebeurtenis declareren

Net als variabelen moet je een gebeurtenis ook declareren. Dat gebeurt, omdat de gebeurtenis overal bruikbaar moet zijn, in het algemene deel van de klassenmodule. De code staat in de afbeelding bovenaan, en luidt:

```
Public Event ActionSuccess(strMessage As String)
```

In de procedure Update zie je dat het event wordt aangeroepen met de code

```
RaiseEvent ActionSuccess("Updates voor bestaand " _
    & "record successvol bewaard!")
```

Volgende keer

In het volgende hoofdstuk ga ik uiteraard verder met de klassenmodule, want op dit moment doet het nog niet veel. We gaan de klasse dan aan een formulier hangen, en code schrijven voor de verschillende gebeurtenissen.