Werken met Recordsets (deel 1)

Al een aantal keren heb ik aangegeven dat ik het in de cursus zou gaan hebben over het koppelen van een Front-end database aan een Backend middels Recordsets. En daar ga ik nu (eindelijk) mee beginnen! Er is namelijk een hele goede reden om een Frontend database te maken die je met behulp van Recordsets laadt vanuit de Backend. Laten we er daar eens een paar van onder de loep nemen:

Belasting van de Backend

Zoals je misschien wel weet, kun je met een beperkt aantal gebruikers in één en dezelfde database werken zonder dat je veel problemen gaat krijgen met de performance van die database. Dat aantal gebruikers ligt tussen de 5-10 personen. En dan liever 5 gebruikers dan 10. Heb je een database waar meer mensen tegelijk in moeten werken, dan is het verstandiger om de database te splitsen, en elke gebruiker een eigen Frontend te geven. Hiermee vergroot je het aantal gebruikers dat tegelijkertijd in de database kan werken aanzienlijk en het is ook gelijk veel makkelijker om elke gebruiker een op maat gesneden werkomgeving te geven. Hij/zij krijgt dan een Frontend met exact die formulieren die hij/zij nodig heeft, en meer niet.

In deze constructie heb je nog steeds gebruikers die continu verbonden zijn aan de database, en dat levert toch een bepaalde druk op in de database. Die druk kun je omlaag brengen door de Backend database met recordsets te benaderen op het moment dat het echt nodig is.

Verschillende Access versies

Sinds Microsoft Access 2007 heeft uitgebracht, is er nogal wat veranderd in de database structuur. Zelfs met de vorige versies, die allemaal van hetzelfde bestandsformaat (mdb) gebruik maakten, was er maar moeizaam uitwisseling mogelijk met vorige versies. Dat wil zeggen: het lukt meestal nog wel om een oudere database in te lezen in een nieuwe versie, maar omgekeerd lukt dat dus niet: je kunt geen Access 2003 database inlezen in Access 2000, tenzij deze in de 2003 versie is gemaakt in het 2000 format. En dat kan behoorlijke problemen opleveren.

Als je een (backend) database benadert met ADO of DAO, dan maakt het niet zoveel uit met welke Access versie de backend is gemaakt; je leest namelijk rechtstreeks uit de tabellen. En die kun je dus ook vanuit een oudere database benaderen.

Zijn er beperkingen?

De versieverschillen in Access hebben meestal te maken met specifieke aspecten als formulieren. Als je een koppeling legt vanuit een oudere database, heb je met formulieren e.d. weinig te maken, want je kijkt alleen naar de tabellen. Je krijgt dan hooguit te maken met velden die meervoudige waarden bevatten (Access 2007 en hoger) en berekende velden; veldtypes die in de oudere databases niet bestaan.

Over “velden met meervoudige waarden” heerst wellicht verwarring bij gebruikers, want eigenlijk is dat iets wat niet thuis hoort in een fatsoenlijke database. Zelf ben ik er dan ook geen voorstander van om deze veldtypes te gebruiken. Eén van de normalisatieregels stelt dat je in één veld precies één waarde opslaat. En als je dus in een veld meerdere waarden zet, dan ligt die regel ergens buiten op straat, maar niet meer in de db. Exact de reden dat ik eerst een absoluut tegenstander van dit veldtype was.

Microsoft heeft in zijn nieuwere versies, waar dit veldtype dus in zit, een vorm bedacht waarin de opgeslagen waarden wel degelijk genormaliseerd kunnen worden. Access maakt hierbij gebruik van interne tabellen, en daarmee kun je toch genormaliseerd werken. Jammer genoeg werkt dit niet als je via een Recordset verbinding maakt met zo’n veld. Je krijgt dan alle opgeslagen waarden in één string te zien, en je moet daar dus zelf een oplossing voor vinden om dat veld weer te normaliseren.

Een andere regel stelt dat gegevens die afgeleid kunnen worden van andere gegevens óók niet

worden opgeslagen in de tabel. Daar verdwijnt dus het “Berekende veld”, want dat doet precies dát: een waarde opslaan die is gebaseerd op waarden in dezelfde record. Deze waarden zijn gelukkig dan wél hard opgeslagen in het veld, en kunnen dus gewoon worden uitgelezen. Dat het veel beter is om de waarden zelf te berekenen, lijkt mij duidelijk. Het is de enige garantie dat de uitkomst ook echt klopt.

ADO of DAO?

Microsoft biedt al jaren de mogelijkheid om ofwel via ADO, ofwel via DAO met recordsets te werken. Of, mocht je dat willen, met allebei. Al raad ik dat eigenlijk niet aan, omdat de structuur van de twee methodes nogal overeenkomt, maar ook wezenlijke verschillen kent. En het is lastig om die te omzeilen als je met beiden tegelijk aan het werken bent. Bovendien zit er nauwelijks verschil in qua performance.

Waarom twee methodes?

Goede vraag, en hij verdient dan ook een goed antwoord. Microsoft heeft min of meer vanaf het begin DAO ondersteund, en doet dat de laatste jaren met versie 3.6. Deze versie vind je dan ook in veruit de meeste Access versies. Daarnaast wordt al jaren het einde van DAO aangekondigd door Microsoft, en wordt beweerd dat DAO ook niet verder meer ontwikkeld wordt. Toch heeft Microsoft in Access 2010 DAO 3.6 vervangen door een nieuwere versie: Microsoft Office x.x Access Database Engine Object Library. In Access 2010 is dat: “Microsoft Office 14.0 Access Database Engine Object Library”.

Dit dus in tegenstelling tot ADO, dat met elke nieuwe Access versie verder stijgt in versienummer. Waar de laatste ‘oude’ Access versie (2003) nog werd geleverd met ADO 2.8, is in de voorlaatste versie (2010) ADO gestegen naar 6.1.

Er blijken nog heel veel gebruikers van DAO te zijn, en omdat er wel degelijk praktische verschillen tussen de methodes zitten, kan Microsoft het blijkbaar niet over zijn hart verkrijgen om de ondersteuning voor DAO stop te zetten. Daarom kunnen we nog steeds met twee technieken werken.

Welke versie moet je gebruiken?

Die vraag is niet zo een-twee-drie te beantwoorden, en hangt af van de databases waar je mee werkt. Als het om snelheid gaat, kun je beide varianten gebruiken, want je zult met de huidige generatie computers nauwelijks verschillen merken tussen DAO en ADO. Maar werk je met verschillende database versies, dan is het wellicht verstandig om met DAO te gaan werken, omdat dat in alle Access uitvoeringen met dezelfde versie werkt: 3.6 namelijk. Dat gezegd hebbende: het wordt lastig om vanuit een Access 2003 database met DAO een verbinding te maken met een Access 2007/2010 database, omdat in de nieuwere versie een andere DAO engine wordt gebruikt. De Access 2003 kan daar niet mee overweg. Er is dus een duidelijke scheidslijn van wat wel kan, en wat niet.

De ADO machine is door de jaren heen behoorlijk veranderd. Niet alleen in versienummer, maar ook in de onderliggende database engine. Tot en met versie 2003 gebruikte Microsoft de Jet engine voor de connectie, maar met ingang van 2007 (en dus ook de hogere versies) wordt met ACE gewerkt. Dat maakt voor de opdrachten niet eens zoveel uit, maar het leggen van de connectie zelf moet gebeuren met een specifieke verwijzing naar de Engine, dus je krijgt met twee verschillende connectiestrings te maken. Die strings zien er bijvoorbeeld zo uit:

cnnDB.Provider = "Microsoft.Jet.OLEDB.4.0"

Voor de JET engine, en

cnnDB.Provider = "Microsoft.ACE.OLEDB.12.0"

Voor de ACE engine.

Heb je eenmaal bepaald met welke database je een connectie wilt maken, dan kun je dus verder

gewoon werken met de databases, maar je moet daar dus wel rekening mee houden. Met DAO speelt dit allemaal niet.

In het eerste voorbeeld gaan we een verbinding maken met een externe database vanuit een bestaande database. In de volgende hoofdstukken gaan we een Frontend database bouwen, maar nu kun je de voorbeelden in elke willekeurige database uitproberen.

Verbinding maken met DAO

Om verbinding te maken met een andere database moet er eerst een *Workspace* worden vastgelegd. Werkgeheugen dus voor de te openen database. Workspaces zijn een eigenschap van het applicatieobject *DBEngine* en de dbEngine is het hoogste niveau in het DAO datamodel. Je begint dus altijd vanuit de DBEngine te bouwen. De DBEngine zelf is weer een eigenschap van de *Applicatie*, en dat is in essentie dan weer de Access database van waaruit je start.

De regels waarmee we de Workspace definiëren zien er zo uit:

```
Sub DAO_Access()

Dim ws As DAO.Workspace

Dim db As DAO.Database

Dim rs As DAO.Recordset

Dim fld As DAO.Field

Dim sDb As String, sSQL As String

Dim sMsg As String
```

We beginnen weer, zoals gewoonlijk, met het declareren van variabelen. In mijn test database heb ik zowel ADO als DAO al geladen, dus ik geef specifiek aan dat het om het DAO datamodel gaat. Gebruik je alleen DAO of alleen ADO, dan kun je dat eventueel weglaten, al is het altijd beter om de verwijzingen volledig te gebruiken. Dan weet je ook altijd wat je precies hebt gedeclareerd.

We definiëren dus een variabele als Workspace, een variabele als Database en een variabele als Recordset. Dat laatste hebben we al eerder gedaan in de cursus, dus dat type zou je bekend voor kunnen komen. Daarnaast definiëren we nog twee strings voor het pad naar de database, en voor de uit te voeren SQL instructie op de database.

```
    sDb = "D:Documents_HelpMijMs Accessbetalingen.mdb"

    sDb = "D:Documents_HelpMijMs Accessbetalingen.accdb"
```

Ik heb alvast twee strings gemaakt, één voor een Access 2003 database, en één voor een 2010 database. Uiteraard kan sDb maar één waarde bevatten, dus de eerste string wordt in deze code vervangen door de tweede. Om de andere variant uit te proberen kun je de volgorde omwisselen, of één van de twee opmaken als commentaar.

```
    Set ws = DBEngine.Workspaces(0)

    Set db = ws.OpenDatabase(sDb)
```

Hier gebeurt het echte werk. Met Set ws wordt aan de variabele ws een Workspace toegevoegd. Je kunt meerdere Workspaces tegelijk declareren, zodat je in één keer verschillende databases kunt openen, maar erg zinvol zal dat meestal niet zijn. De eerste Workspace wijs je toe met de waarde 0

zoals in de code. De opdracht Set ws1 = DBEngine.Workspaces(1) zou je dus ernaast kunnen declareren als je een tweede Workspace nodig hebt.

De opdrachtregel Set db = ws.OpenDatabase(sDb)opent vervolgens de database in de aangemaakte Workspace. Daarvoor wordt de opdracht OpenDatabase gebruikt.

```
    sSQL = "SELECT * FROM tblKlanten"

    Set rs = db.OpenRecordset(sSQL)

    For Each fld In rs.Fields

        If Not sMsg = vbNullString Then sMsg = sMsg & vbLf

        sMsg = sMsg & fld.Name

    Next fld

    MsgBox sMsg
```

In het volgende stuk wordt in de variabele rs een recordset geladen, in mijn voorbeeldje de tabel [tblKlanten]. Gebruik je een andere database, dan kun je hier uiteraard een andere tabel gebruiken.

Om te kijken of de code werkt (wat je overigens gauw genoeg merkt als je de procedure uitvoert), doen we nu niet veel meer dan in een lus door de veldnamen van de tabel lopen en de veldnamen in een Messagebox laten zien. Als dat lukt, dan weet je dat de code goed werkt.

```
    rs.Close

    Set rs = Nothing

    db.Close

    Set db = Nothing

    ws.Close

    Set ws = Nothing

End Sub
```

In het laatste stuk worden alle objecten gesloten. Dat is in beginsel niet nodig, want als een procedure wordt beëindigd worden ook alle variabelen opgeruimd. Het is echter netter om het wel zo te doen, zeker als je de ‘opruimcode’ combineert met een foutprocedure, zodat de variabelen ook echt opgeruimd worden, ook als de procedure vastloopt.

Verbinding maken met ADO

ADO werkt niet met Workspaces, maar met *Connections*. Je moet dus eerst een connectie maken met een database voordat je er mee kunt werken. Heb je eenmaal een connectie gemaakt, dan kun je weer recordsets openen binnen de connectie.

```
Sub ADO_XLAccess2003()

Dim cnnDB1 As ADODB.Connection
```

```
Dim rs1 As New ADODB.Recordset

Dim fld As ADODB.Field

Dim sDb1 As String, sSQL As String

Dim sMsg As String
```

Het gebruikelijke verhaal, eerst weer de variabelen declareren! Ik laat de code twee keer zien; één keer gesplitst voor een 2003 connectie en daarna in zijn geheel voor een Access 2010 database.

```
    sDb1 = "D:Documents_HelpMijMs Accessbetalingen.mdb"

    Set cnnDB1 = New ADODB.Connection
```

We hebben de variabele cnnDB1 ingesteld als ADODB.Connection, en hij wordt ingesteld met Set.

```
    With cnnDB1

        .Provider = "Microsoft.Jet.OLEDB.4.0"

        .CursorLocation = adUseClient

        .Open sDb1

    End With
```

Een connectie heeft, zoals de meeste objecten, een aantal eigenschappen die je kunt instellen. In dit voorbeeld gebruiken we er drie: Provider, CursorLocation en Open. Daarvan is de laatste logisch, want het is nu eenmaal de bedoeling om een database te openen. De parameter die *Open* krijgt is dan ook de volledige verwijzing naar de te openen database.

De eigenschap Provider

De eigenschap *Provider* wordt gebruikt om de database engine te duiden waarmee de connectie moet worden gemaakt. Zoals ik al eerder zei: de nieuwere versies van Access (2007/2010/2013) gebruiken een andere engine, en de provider moet dan ook specifiek voor de database worden ingesteld.

De eigenschap CursorLocation

De eigenschap CursorLocation is een interessante, want hiermee geef je aan wat er met de opgehaalde gegevens moet gebeuren. Er zijn twee varianten: *adUseClient* en *adUseServer*.

Gebruik je *adUseClient*, dan wordt de volledige dataset waarmee gewerkt gaat worden overgehaald naar de computer waar de gebruiker aan werkt. Dit heeft voor- en nadelen. Voordeel is dat je voor de verwerking van de gegevens niet afhankelijk bent van de server waarvandaan je de gegevens ophaalt. Je hebt dus een hele snelle dataverwerking, want alle gegevens heb je lokaal staan. Nadeel is, dat het wat tijd kost om de complete dataset over te halen; elke keer als je gegevens ophaalt of wegschrijft, gaat er veel data heen en weer naar de databases.

Met *adUseServer* gebeurt precies het omgekeerde. Nu blijft alle data zoveel mogelijk op de server staan, en wordt alleen datgene naar de client gehaald wat bewerkt moet worden. Voordeel hiervan is, dat er relatief weinig dataverkeer is op het netwerk. De verbinding tussen de databases is dus razendsnel. Nadeel kan zijn dat met veel gebruikers de resources van de databases in de knel kunnen komen, en er dus aan de server kant performance wordt ingeleverd. Het is dus zaak om een goede afweging te maken tussen een Client cursor en een Server cursor.

```
    sSQL = "SELECT * FROM tblKlanten"

    rs1.Open sSQL, cnnDB1
```

In de variabele sSQL staat de querystring die we willen openen. Die wordt vervolgens in de recordset geopend, waarbij de recordset dus als parameter de SQL string krijgt, en de juiste Connector. Ik heb al eerder aangegeven dat je met DAO makkelijk meerdere Workspaces tegelijk open kunt zetten, en voor ADO geldt hetzelfde, maar dan voor meerdere connecties. Je geeft dus in de Recordset aan in welke connector hij moet worden geopend.

```
    For Each fld In rs1.Fields

        If Not sMsg = vbNullString Then sMsg = sMsg & vbLf

        sMsg = sMsg & fld.Name

    Next fld

    MsgBox sMsg
```

Als de recordset is geopend, kun je er van alles mee doen. In dit geval test ik alleen of de constructie werkt, en dat gebeurt met een lus die door de Fields collectie van de recordset loopt, en alle veldnamen in een string zet. Deze wordt na het voltooien van de lus dan in een Messagebox getoond.

```
    rs1.Close

    Set rs1 = Nothing

    cnnDB1.Close

    Set cnnDB1 = Nothing

End Sub
```

En om alles netjes af te sluiten, eindigt de procedure met het afsluiten van de variabelen.

Hieronder vind je de volledige code voor het benaderen van een Access 2010 database. Die is identiek aan de code voor de Access 2003 database, met alleen een andere Provider string.

```
Sub ADO_XLAccess2010()

Dim cnnDB2 As ADODB.Connection

Dim fld As ADODB.Field

Dim rs2 As New ADODB.Recordset

Dim sDb2 As String, sSQL As String

Dim sMsg As String

    sDb2 = "D:Documents_HelpMijMs Accessbetalingen.accdb"

    Set cnnDB2 = New ADODB.Connection
```

```
    With cnnDB2
        .CursorLocation = adUseServer
        .Provider = "Microsoft.ACE.OLEDB.12.0"
        .Open sDb2
    End With
    sSQL = "SELECT * FROM tblKlanten"
    rs2.Open sSQL, cnnDB2
    For Each fld In rs2.Fields
        If Not sMsg = vbNullString Then sMsg = sMsg & vbLf
        sMsg = sMsg & fld.Name
    Next fld
    MsgBox sMsg
    rs2.Close
    Set rs2 = Nothing
    cnnDB2.Close
    Set cnnDB2 = Nothing
End Sub
```

Samenvatting

Dit is pas deel 1 van het maken van een verbinding met een andere database. Je hebt gezien dat als je vanaf een Access 2003 database een verbinding wilt maken met een andere (2003 of oudere) database, je prima met DAO kunt werken. Het is vrij simpel om daarmee verbindingen te maken.

Wil je echter vanuit Access 2003 gegevens ophalen uit een 'hogere' database versie, dan ben je eigenlijk al aangewezen op ADO, omdat Access in de nieuwe versies gebruik maakt van nieuwere engines. En DAO kan die niet gebruiken, en ADO wel. De keus is dan dus simpel.

Volgende keer gaan we een onafhankelijk Front-end formulier maken, waarbij we de tabellen dus niet koppelen in de database, maar middels een Verbinding zoals dat hier beschreven staat.