Access voor Beginners - Hoofdstuk 10 deel 2

Handleiding van Helpmij.nl

Auteur: OctaFish

Februari 2012

Formulieren: Werken met onafhankelijke Recordsets

In de vorige aflevering van de cursus zijn we begonnen met het maken van een formulier om met behulp van een timer tijden op te kunnen slaan in een tabel. Daarvoor hebben we een formulier gebruikt met een functie die gebruik maakt van de klok van de computer, om tijden in honderdsten van seconden te kunnen berekenen.

In dat hoofdstuk heb ik ook al aangekondigd dat we de gegevens die we genereren met de Timer-functie gaan opslaan in een onafhankelijke recordset. De reden daarvoor was, dat (schijf)bewerkingen tijd kosten, en die gaat ten koste van de nauwkeurigheid van de tijdregistratie. En die willen we zo zuiver mogelijk vastleggen en registreren. Daarom maken we dus in dit hoofdstuk gebruik van *virtuele* tabellen; tabellen die alleen in het geheugen van de computer bestaan. Een virtuele tabel heeft één nadeel: als je de pc uitzet, is de tabel ook verdwenen. We zullen de gegevens dus op enigerlei moment moeten opslaan om het verlies van de gegevens te voorkomen.

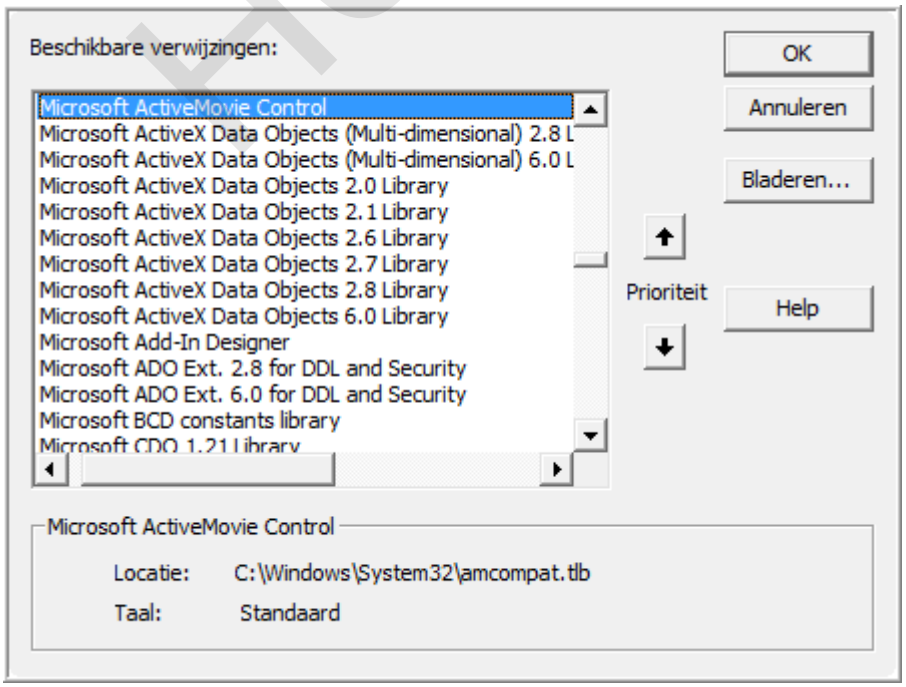
Voordat we een virtuele tabel gaan maken, kijken we eerst naar de methodes die we kunnen gebruiken om een recordset te maken. Access ondersteunt er namelijk een paar! En dat betekent, dat we een keuze zullen moeten maken voor de één of de ander. Welke smaken zijn er zoal?

ADO vs DAO

Nee, dit is geen voetbalwedstrijd tussen **A**lles **D**oor **O**verreding en **D**e **A**lmachtige **O**mkoopclub; ADO en DAO zijn afkortingen voor twee verschillende methodes om met recordsets te werken. De letters staan voor de volgende afkortingen:

DAO - **D**ata **A**ccess **O**bjects
ADO - **A**ctiveX **D**ata **O**bjects

Van de twee is DAO de oudste; dat maakt ADO derhalve de jongere van de twee. Microsoft heeft, qua ontwikkeling van zijn programma, min of meer gekozen voor ADO, want die versie wordt regelmatig verbeterd; bij elke nieuwe versie van Access zit wel een vernieuwde versie van ADO. Als je de bibliotheek met verwijzingen dan ook bekijkt, vind je een stevig aantal verwijzingen naar de verschillende ADO-versies, terwijl er van DAO meestal maar één versie in het programma zit. In het bijgaande plaatje zie je de versies zoals die beschikbaar zijn in Access 2010.



Het is altijd verstandig om de nieuwste versie van een bibliotheek te gebruiken, tenzij je vermoed dat de database ook in een oudere versie gebruikt gaat worden, waarin die specifieke bibliotheek niet aanwezig is. Zo zie je in dit plaatje een verwijzing naar ADO 6.0; deze bibliotheek zit niet bij Access 2003 (waar dit plaatje van afkomstig is), maar zit bij Access 2010. Als je een database maakt die ook in Access 2003 moet werken, dan is het dus verstandig om ADO 2.8 te gebruiken i.p.v. ADO 6.0. De bibliotheken zijn 'downward compatible' wat betekent dat alles wat in ADO 2.1 zit, ook in ADO 2.8 is te gebruiken. Omgekeerd hoeft dat uiteraard niet zo te zijn; Microsoft ontwikkelt zijn bibliotheken niet voor niets!

Het DAO-model

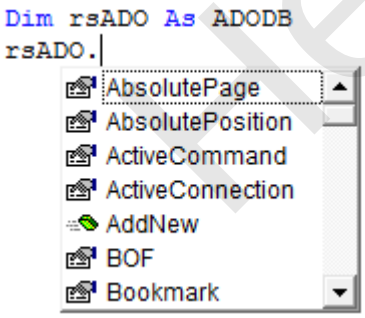
Zoals de naam al eigenlijk zegt, is het DAO-model ontwikkeld om Data toegankelijk te maken, en dan voornamelijk in Access. Omdat de bibliotheken ook in andere (Office) applicaties zijn te laden, kun je ook vanuit bijvoorbeeld Excel of Word een Access-tabel openen. Dat maakt het werken met het DAO-model bijzonder interessant.

Het ADO-model

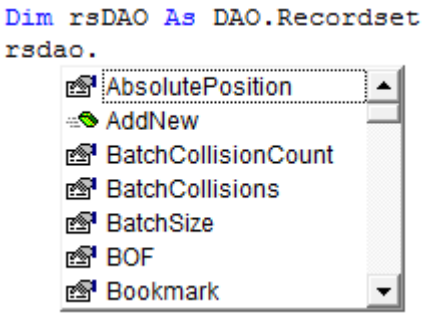
Dit model is later toegevoegd aan de Office suite, en beperkt zich niet alleen tot Access objecten. Het model is dus uitgebreider dan alleen tot databases. De nieuwste versies van ADO worden overigens ontwikkeld voor ADO.NET, wat weer een totaal andere bibliotheek is. Om het ADO-verhaal nog wat gecompliceerder te maken, is er ook een uitgebreide variant beschikbaar: ADOX. Zo kun je met ADO geen tabellen maken, maar met ADOX wel. Afhankelijk van de taken die je wilt uitvoeren, moet je dus eerst nadenken over de versie van ADO die je nodig hebt.

Verschillen tussen ADO en DAO

Als je met recordsets gaat werken, zul je merken dat beide varianten ongeveer dezelfde structuur hebben, maar zijn de commando's die je daarvoor nodig hebt nèt even wat anders. Ze gebruiken alle twee a.h.v. een eigen programmeertaal. Je kunt dan ook geen ADO opdrachten gebruiken als je met een DAO-recordset werkt, en omgekeerd.
Kijk maar eens naar de volgende voorbeelden; eerst de ADO-bibliotheek:



En hieronder de DAO-bibliotheek:



Een groot verschil is, dat Access zelf voor zijn objecten zoals formulieren DAO gebruikt. Als je dus de recordset (of een kopie daarvan) van een formulier opent, is dat standaard een DAO-object. Verder kun je met ADO geen nieuwe databases maken, en in de 'standaarduitvoering' ook geen tabellen. Voor het doel waarvoor wij recordsets gaan gebruiken, zijn de verschillen niet wezenlijk van belang.

Welke methode moet je gebruiken?

Als je wilt, kun je dagen doorbrengen op het internet om uit te zoeken welke variant nu gebruikt moet worden; op het ene forum lees je dat ADO het helemaal gaat maken en dat DAO op zijn laatste benen loopt, en op een ander forum wordt je overspoeld met de nadelen van ADO, en worden er weer vurige lansen gesmeed om toch maar vooral met DAO te blijven werken. Hetzelfde geldt voor boeken over Access en programmeren: je kunt stapels van gelijke hoogte maken van boeken uit beide kampen.

De verschillen zullen bij grote databases ongetwijfeld meetbaar zijn, en in het voordeel van een van de twee methodes werken. Voor het werk dat wij met een Access database doen, stellen we dat ze allebei even bruikbaar zijn. Al vermoed ik dat een discussie op Helpmij.nl vast en zeker verhitte voorstanders van de ene of de andere methode gaat opleveren!

Early Binding vs Late Binding

Bibliotheken, zoals in het vorige plaatje te zien waren, moeten worden geactiveerd. Dat doen we in het VBA-venster van Access. Je komt daar met de toetscombinatie <Alt>+<F11>. Vervolgens kies je in het menu <Extra> de optie <Verwijzingen>.

Maar voordat we met een bibliotheek activeren, bekijken we eerst of we gaan werken met *Late Binding*, of met *Early Binding*. Om het geheugen even op te frissen: het is mogelijk om in Access met objecten uit bibliotheken te werken die niet zijn geactiveerd in de bibliotheken. Het declareren van objecten gebeurt dan door ze als *Object* te declareren. Het voordeel van Late binding is, dat een andere gebruiker van het bestand (Access db, Excel spreadsheet) niet hoeft te controleren of de bibliotheken die gebruikt zijn wel op de lokale computer zijn geactiveerd. Met Late Binding gebeurt dat namelijk in de procedures zelf.

Dat klinkt als een 'offer you can't refuse', maar zoals algemeen bekend is: 'elk voordeel hep zun nadeel'… In dit geval is dat: de procedure wordt er langzamer op, en je kunt geen gebruik maken van de parameters die bij een object horen, en dat maakt het programmeren wat moeilijker. Kijk nog eens naar de voorbeelden uit de vorige plaatjes: en let dan op hoe de eigenschappen worden gebruikt.

In het tweede plaatje is het object rsDAO gedefinieerd als een DAO-recordset. Als je nu een methode wilt hangen aan het object, typ je ***rsdao*** gevolgd door een punt, waarna een keuzelijst met methodes en eigenschappen getoond wordt waaruit je kunt kiezen. Niet alleen handig, het voorkomt ook dat je typefouten maakt.

Overigens zie je in die plaatjes al gelijk dat er behoorlijke verschillen zijn tussen de twee methodes ADO en DAO. Zo begint de ADO-bibliotheek met *AbsolutePage*, een eigenschap die DAO duidelijk niet heeft. Wel hebben ze beide de eigenschap *AbsolutePosition*. Beide bibliotheken gebruiken dan

wel weer de methode *AddNew* en *BOF*.

Om weer terug te komen op Late Binding: objecten die op die manier worden gedeclareerd ontberen deze zinvolle hulp. Je kunt zelfs niet eens volstaan met het zelf typen van de methode of eigenschap, want je kunt alleen verwijzen naar het *nummer* van de optie uit de lijst.

Een voorbeeldje met een Outlook object gedefinieerd met Early binding:

olNs.GetDefaultFolder(olFolderCalendar)

Moet zo gemaakt worden als je de Outlook bibliotheek niet activeert:

olNs.GetDefaultFolder(9)

En dat is toch een stuk minder leesbaar!

Wat wordt het nu: ADO of DAO?

Die vraag is bijzonder makkelijk te maken, want één verschil tussen ADO en DAO is nog niet aangestipt: met DAO kun je helaas geen virtuele tabellen maken. En dat maakt de keuze natuurlijk gelijk heel simpel. Uit de keuze van 1 kiezen we dan uiteraard de eerste of laatste optie! Dus maken we de tijdelijke tabel in ADO. Als je een normale tabel kunt gebruiken in je eigen projecten, kun je uiteraard wel een DAO-recordset maken.

Dus samenvattend: een groot voordeel van Late binding is: onafhankelijke werking van de code. Voordelen van Early binding: de code werkt doorgaans wat sneller, het samenstellen van de commando's gaat een stuk intuïtiever, en de code is net wat leesbaarder. En we gaan de recordset maken met ADO.

Tip voor als je Late Binding nodig hebt: Maak je procedures eerst met Early Binding, en werk hem, als de db klaar is, later om naar Late Binding! Op die manier heb bij het ontwikkelen de volledige vrijheid die Early binding je biedt. Voor de cursus gebruik ik Early Binding, al was het maar omdat ik de nodige commando's beter kan omschrijven m.b.v. de parameters.

In het tijdschrijf formulier hebben we de recordset gedurende het gebruik van het formulier nodig. Het lijkt dan ook zinvol om de recordset maar gelijk bij het openen van het formulier aan te maken. Dat doen we bij de eigenschap Form_Load. De code ziet er zo uit:

```
Private Sub Form_Load()
   'Recordset klaarzetten voor de tijden
   Set rsADO = New ADODB.RecordSet
   With rsADO.Fields
      .Append "RitID", adInteger
      .Append "Ronde", adInteger
      .Append "SchaatserID", adInteger
      .Append "GeredenAfstand", adInteger
      .Append "Tijd", adVarChar, 12, adFldIsNullable
      .Append "Opmerking", adVarChar, 255, adFldIsNullable
   End With
   rsADO.Open
End Sub
```

We beginnen toewijzen van het Recordset object aan de variabele rsADO. Aan een recordset zonder velden hebben we uiteraard niks, dus in de volgende stap gaan we de velden toewijzen aan de recordset. Dat kan door elke regel als volledig commando uit te schrijven, zoals hier:

```
rsADO.Fields.Append "RitID", adInteger
rsADO.Fields.Append "Ronde", adInteger
rsADO.Fields.Append "SchaatserID", adInteger
```

Maar deze werkwijze is niet erg efficiënt, omdat je steeds opnieuw naar hetzelfde object of eigenschap moet wijzen. In dit geval moeten we elke keer het Fields eigenschap van het object Recordset (rsADO) openen. Met de instructie *With … End With* hoef je het object maar één keer te openen, en kun je verschillende instructies uitvoeren op een collectie. Je kunt hoeft de groep dus maar één keer te benoemen.

Eerst het voorbeeld met een enkelvoudig niveau, zoals hierboven gebruikt. Je begint de instructie met het bepalen van het object en/of de eigenschap waarop je wilt werken. Je ziet dat al heel snel, doordat de groepen die je kunt maken steeds herhaald worden aan het begin van de regel.

```
With rsADO.Fields
  .Append "RitID", adInteger
  …
  .Append "Opmerking", adVarChar, 255, adFldIsNullable
End With
rsADO.Open
```

Als je het voorbeeld goed bekijkt, zie je dat er nog een groep is te maken: de groep rsADO. Deze groep bevat twee eigenschappen: Fields en Open. Ook deze kun je in een groep openen, waarbij we de Append eigenschap uit het Fields object moeten nesten. Dat ziet er dan zo uit:

```
With rsADO
   With .Fields
      .Append "RitID", adInteger
      .Append "Ronde", adInteger
      …
      .Append "Opmerking", adVarChar, 255, adFldIsNullable
   End With
   .Open
End With
```

Hierbij word de groep rsADO geopend, vervolgens de groep Fields, waarna de velden via Append worden aangemaakt en de groep Fields wordt gesloten met End With. Daarna wordt de volgende regel uit de groep rsADO uitgevoerd (Open) waarna de groep rsADO wordt gesloten.

```
With rsADO
   With .Fields
      .Append "RitID", adInteger
      .Append "Ronde", adInteger
      .Append "SchaatserID", adInteger
      .Append "GeredenAfstand", adInteger
      .Append "Tijd", adVarChar, 12, adFldIsNullable
      .Append "Opmerking", adVarChar, 255, adFldIsNullable
   End With
   .Open
End With
```

Ik heb het nog niet over de veldtypen gehad die worden gemaakt. In de recordset zijn dat er twee: een getalveld en een tekstveld. Het getal is van het type Integer, en heet in ADO adInteger. Het tekstveld krijgt niet alleen een eigenschap, maar ook een grootte: het aantal tekens dat we willen opslaan. Dat doe je met de eigenschap adVarChar, 12, adFldIsNullable. Met adFldIsNullable geven we aan dat het veld ook nulwaarden mag bevatten. Andere gangbare veldtypen zijn: adBoolean, adDate, adDouble en adCurrency.

Records maken in een Virtuele tabel

Nu we de recordset hebben klaargezet, kunnen we hem gaan gebruiken. Dit doen we in combinatie met de timer functie die we in de vorige aflevering hebben gemaakt. Wat we gaan doen, is eigenlijk heel simpel: elke keer dat er op een knop wordt geklikt, en de tijd wordt gemeten, wordt het record opgeslagen in de recordset.

Laten we het proces eens van dichtbij bekijken…

Het formulier bestaat uit een aantal knoppen, en een aantal tekstvelden. Je ziet twee knoppen: <Start> en <Tijd>. De knop <Tijd> is nu nog uitgeschakeld, omdat de klok nog niet loopt, en je dus nog geen rondetijd kunt vastleggen. De tekstvelden worden gedurende het registreren gevuld. Dit kost uiteraard wat tijd, en dat zal ten koste gaan van de nauwkeurigheid, zoals je verderop zult merken aan de eindtijd. Zoals ik eerder al zei: hoe minder de computer hoeft te doen, hoe beter de tijd wordt vastgelegd.

De eerste actie is dus: de klok (en de tijdregistratie) starten met de knop <Start>. Die knop bevat de volgende code:

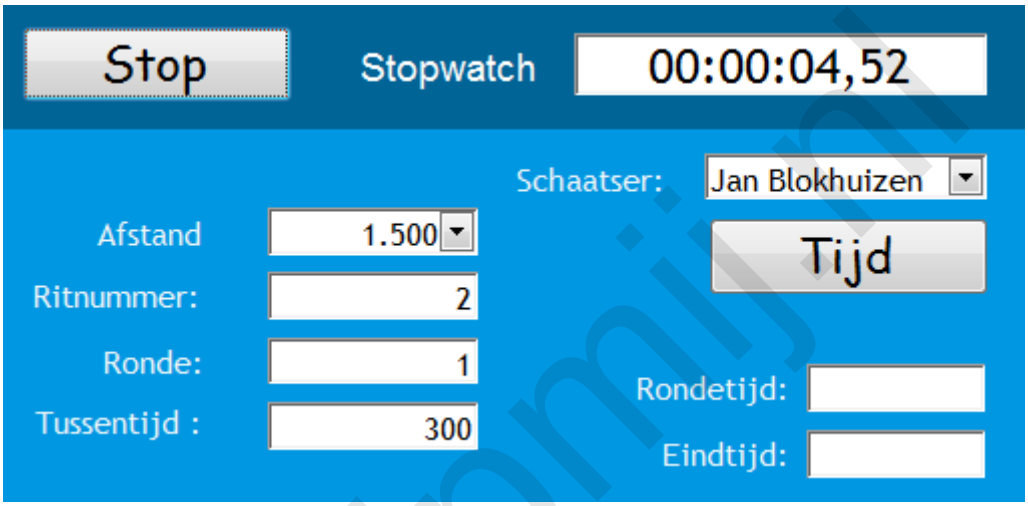
```
If Me.TimerInterval = 0 Then
   iGereden = CInt(Me.cboAfstand.Column(1)) Mod 400
   If iGereden = 0 Then Me.txtAfstand = 400 Else Me.txtAfstand = iGereden
   Me.txtRonde = 1
   Me.btnStartStop.Caption = "Stop"
   Me.btnReset.Enabled = False
   StartTickCount = GetTickCount()
   Me.TimerInterval = 15
Else
   TotalElapsedMilliSec = TotalElapsedMilliSec
      + (GetTickCount() - StartTickCount)
   Me.TimerInterval = 0
   Me!btnStartStop.Caption = "Start"
   Me!btnReset.Enabled = True
End If
```

De code is gedeeltelijk al eerder voorbijgekomen in de vorige aflevering, dus ik concentreer mij nu op twee nieuwe regels:

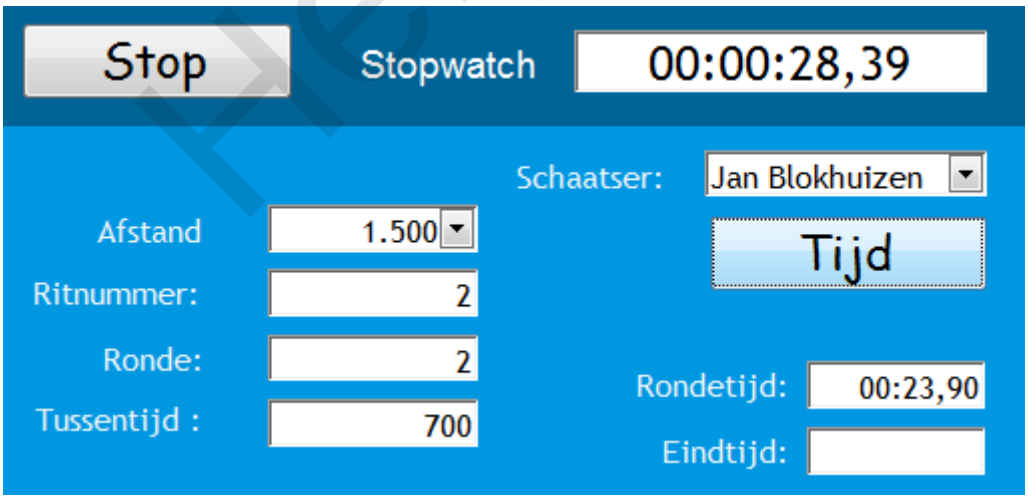
```
   iGereden = CInt(Me.cboAfstand.Column(1)) Mod 400
   If iGereden = 0 Then Me.txtAfstand = 400 Else Me.txtAfstand = iGereden
```

Iedereen die wel eens een langebaan schaatswedstrijd heeft gezien, weet dat de te rijden afstanden maar zelden een veelvoud zijn van de lengte van de schaatsbaan (die 400 m lang is). Daarom wordt er op verschillende plaatsen op de baan gestart, en is het eerste rondje meestal korter dan 400 m. Met behulp van de keuzelijst waarmee de ritlengte op het formulier wordt ingesteld kunnen we uitrekenen hoe lang het eerste rondje is. In het geval van het voorbeeld (1500 m) zouden 4 rondjes van 400 m een ritlengte van 1600 m opleveren. De eerste ronde is dus 300 m.

Hoe bereken je dat nu? Met de rekenkundige operator Mod; deze operator deelt een getal door een ander getal en geeft alleen de rest als resultaat. Als we dus 1500 Mod 400 uitrekenen, is het restant 300, zijnde de afstand van de eerste ronde. Is het resultaat 0, wat bijvoorbeeld bij de 10 km het resultaat is, dan is de eerste ronde exact 400 meter. En die check wordt dus in de tweede regel uitgevoerd.

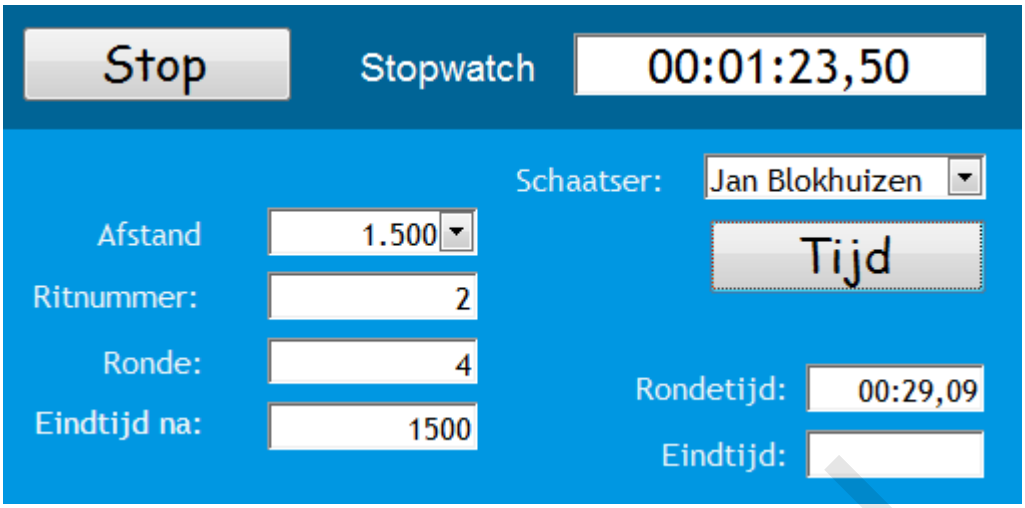


Als we dus op de <Start> knop hebben geklikt, gaat de klok lopen, en verschijnt de verstreken tijd in de stopwatch.



Je ziet de eerste ronde lopen, en de eerste gegevens in het formulier verschijnen. Zodra de renner over de streep komt, klik je op de knop <Tijd> (die nu actief is) en begint de klok voor de tweede ronde te lopen.

En bij de laatste ronde ziet het scherm er zo uit:



Het label bij de verreden afstand is veranderd, omdat de laatste ronde is ingegaan. Als de rijder over de streep gaat, wordt dan ook de laatste tijdmeting vastgelegd.

Functie Tijd

Om de code straks een beetje overzichtelijk te houden, maken we eerst van de routine die de verstreken tijd uitrekent een aparte functie die we op verschillende plekken kunnen aanroepen. Dat heeft als voordeel, dat de code een stuk overzichtelijker wordt. De functie ziet er zo uit:

```
Function Tijd(Waarde As Variant) As String
   Minutes = Format((Waarde 60000) Mod 60, "00")
   Seconds = Format((Waarde 1000) Mod 60, "00")
   MilliSec = Format((Waarde Mod 1000) 10, "00")
   Tijd = Minutes & ":" & Seconds & "," & MilliSec
End Function
```

De functie bevat een parameter die wordt meegegeven als we de functie aanroepen (Waarde), en de functie levert een Tekstreeks op als resultaat.

De Start/Stop knop bevat de code die we al kennen, en waarmee de timer wordt gestart. Voor de volledigheid geef ik die code nog een keer, maar ik zal hem verder niet behandelen.

```
Private Sub btnStartStop_Click()
   Me.cmdKlikken.Enabled = True
   If Me.TimerInterval = 0 And Me!btnStartStop.Caption = "Start" Then
      iGereden = CInt(Me.cboAfstand.Column(1)) Mod 400
      If iGereden = 0 Then Me.txtAfstand = 400 Else Me.txtAfstand = iGereden
      Me.txtRonde = 1
      Me.btnStartStop.Caption = "Stop"
      Me.btnReset.Enabled = False
      StartTickCount = GetTickCount()
      Me.TimerInterval = 15
   ElseIf Me.TimerInterval = 0 And Me!btnStartStop.Caption = "Stop" Then
      Me!btnStartStop.Caption = "Start"
      Resetten
   Else
      TotalElapsedMilliSec = TotalElapsedMilliSec + (GetTickCount() - StartTickCount)
      Me.TimerInterval = 0
      Me!btnStartStop.Caption = "Start"
      Resetten
   End If
```

```
End Sub
```

De knop voor de tijdregistratie is wat uitgebreider, en die wordt daarom in stukken geknipt.

```
Private Sub cmdKlikken_Click()
Dim sRondeTijd As String, sTussenTijd As String
    sTussenTijd = ""
   If CInt(Me.txtAfstand) = CInt(Me.cboAfstand.Column(1)) Then
```

De eerste controle die we doen is om te kijken of de afstand in zijn geheel is afgelegd of niet. In het eerste geval is de rit afgelopen, en kunnen we de data opslaan. In het tweede geval leggen we alleen de tussentijd vast. Omdat ik eerst wil laten zien hoe we de virtuele tabel werkt, sla ik dit deel over, en ga ik verder met het Else blok.

```
   Else
      RoundMilliSec = Nz(Me.txtElapsed, 0)
      Me.txtElapsed = ElapsedMilliSec
      TotalRoundMilliSec = ElapsedMilliSec - RoundMilliSec
      sRondeTijd = Tijd(TotalRoundMilliSec)
```

Het is wel handig om behalve de rondetijden ook de totaal verreden tijd straks op te slaan, zoals dat op de televisie ook gebeurt. Tenslotte gaat het uiteindelijk om de verreden tijd. Die totaaltijd houden we bij in een variabele (TotalRoundMilliSec) die wordt berekend op basis van de verstreken tijd.

```
      ElapsedMilliSec = (GetTickCount() - StartTickCount) + TotalElapsedMilliSec
      sTussenTijd = Tijd(ElapsedMilliSec)
      TotalElapsedMilliSec = TotalElapsedMilliSec + (GetTickCount() - StartTickCount)
      Me.txtRonde = Me.txtRonde + 1
      Me.txtRondeTijd = sRondeTijd
```

Daarna wordt de verstreken tijd van de ronde uitgerekend op basis van de Timer functie. De berekende rondetijd (sRondeTijd) en verstreken totale tussentijd (sTussenTijd) worden beide met de functie Tijd berekend. Zoals ik eerder al zei, gebruikt die functie een invoerwaarde. De rondetijd wordt berekend met de functieaanroep sRondeTijd = Tijd(TotalRoundMilliSec), en de tussentijd met sTussenTijd = Tijd(ElapsedMilliSec). De volgende stap is het opslaan van de gegevens.

```
      With rsADO
         .AddNew
         .Fields("RitID") = Me.Schaatser
         .Fields("Ronde") = iRonde
         .Fields("SchaatserID") = Me.cboSchaatser
         .Fields("GeredenAfstand") = Me.txtAfstand
         .Fields("Rondetijd") = sRondeTijd
         .Fields("Tussentijd") = sTussenTijd
         .Update
      End With
```

We hebben de Recordset al in een eerder stadium klaargezet. Nu kunnen we hem vullen. We gebruiken hier weer de **With…End With** constructie voor om de code overzichtelijk te houden. Dit hoeft dus niet, maar ik raad aan om, als dat mogelijk is, deze werkwijze zoveel mogelijk toe te passen.

Eerst wordt met **With rsADO.AddNew** de recordset als groep geopend, en wordt met **.AddNew** een nieuw record aangemaakt. Vervolgens worden alle records gevuld met de waarden uit het formulier, en de verschillende variabelen. Als alle velden zijn gevuld, moet het record nog worden opgeslagen, anders raken we de gegevens kwijt. Dat doen we met **.Update**

```
      Me.txtAfstand = Me.txtAfstand + 400
      If CInt(Me.txtAfstand) = CInt(Me.cboAfstand.Column(1)) Then
```

```
            Me.lblAfstand.Caption = "Eindtijd na:"
        End If
    End If
End Sub
```

De laatste stap is het aanpassen van de verreden afstand (eerste ronde dus flexibel, de overige ronden 400 meter). En dan begint het hele verhaal weer van voren af aan… Tot we bij de laatste tijdregistratie zijn aangeland: de code die ik met opzet even heb verplaatst naar het eind. De eerste code uit dat blok is min of meer hetzelfde als hierboven, want we moeten eerst de tijden van de laatste ronde vastleggen. En de knop om een tijd vast te leggen wordt uitgeschakeld. Dat gebeurt als volgt:

```
        Me.TimerInterval = 0
        RoundMilliSec = Nz(Me.txtElapsed, 0)
        Me.txtElapsed = ElapsedMilliSec
        TotalRoundMilliSec = ElapsedMilliSec - RoundMilliSec
        sRondeTijd = Tijd(TotalRoundMilliSec)

        ElapsedMilliSec = (GetTickCount() - StartTickCount) + TotalElapsedMilliSec
        Me.txtEindTijd = Tijd(ElapsedMilliSec)
        TotalElapsedMilliSec = TotalElapsedMilliSec + (GetTickCount() - StartTickCount)

        Me.cmdKlikken.Enabled = False
        With rsADO
            .AddNew
            .Fields("RitID") = Me.Schaatser
            .Fields("Ronde") = iRonde
            .Fields("SchaatserID") = Me.cboSchaatser
            .Fields("GeredenAfstand") = Me.txtAfstand
            .Fields("RondeTijd") = sRondeTijd
            .Fields("TussenTijd") = Me.txtEindTijd
            .Update
        End With
```

En nu dan het opslaan… We hebben dus voor elke ronde een record aangemaakt in de virtuele tabel. Die records moeten we nu uitlezen, en opslaan in een echte tabel. Voor het opslaan in de tabel [tRittijden] gebruik ik de DAO procedure; er is weinig op tegen om ADO en DAO door elkaar te gebruiken, zolang voor de procedure maar duidelijk is wat wat is. We beginnen met het openen van het eerste record van de tijdelijke tabel met **.MoveFirst**. Vervolgens gaan we met een lus door alle records heen: **Do While Not .EOF**. Hiermee zeggen we: loop door de records tot je bij het laatste record bent (**E**nd **O**f **F**ile)

```
With rsADO
    .MoveFirst
    Do While Not .EOF
```

We openen vervolgens de tabel [tRittijden]. Ook deze tabel moet worden geopend in de toevoegmodus. Dat doen we weer met **.AddNew**. Daarna worden de velden weer gevuld (*.Fields("RitID") = rsADO.Fields("RitID")* bijvoorbeeld)

```
        With CurrentDb.OpenRecordset("tRittijden")
            .AddNew
            .Fields("RitID") = rsADO.Fields("RitID")
            .Fields("SchaatserID") = rsADO.Fields("SchaatserID")
            .Fields("GeredenAfstand") = rsADO.Fields("GeredenAfstand")
            .Fields("RondeTijd") = rsADO.Fields("RondeTijd")
            .Fields("TussenTijd") = rsADO.Fields("TussenTijd")
```

In de laatste stappen slaan we met **.Update** het record op, en wordt met **.Close** de tabel gesloten. Omdat we het wegschrijven ook met een With groep doen, wordt vervolgens de groep gesloten met

End With. De laatste stap is het volgende record ophalen uit de tijdelijke tabel. Dat gebeurt met **.MoveNext**. En de lus wordt afgesloten met **Loop**.

```
            .Update
            .Close
        End With
        .MoveNext
    Loop
End With
```

De hele procedure, ziet er derhalve zo uit:

```
Private Sub cmdKlikken_Click()
Dim sRondeTijd As String, sTussenTijd As String
    sTussenTijd = ""
    If CInt(Me.txtAfstand) = CInt(Me.cboAfstand.Column(1)) Then
        Me.TimerInterval = 0
        RoundMilliSec = Nz(Me.txtElapsed1, 0)
        Me.txtElapsed1 = ElapsedMilliSec
        TotalRoundMilliSec = ElapsedMilliSec - RoundMilliSec
        sRondeTijd = Tijd(TotalRoundMilliSec)
        ElapsedMilliSec = (GetTickCount() - StartTickCount) + TotalElapsedMilliSec
        Me.txtEindTijd = Tijd(ElapsedMilliSec)
        TotalElapsedMilliSec = TotalElapsedMilliSec + (GetTickCount() - StartTickCount)
        Me.cmdKlikken.Enabled = False
        With rsADO
            .AddNew
            .Fields("RitID") = Me.Schaatser
            .Fields("Ronde") = iRonde
            .Fields("SchaatserID") = Me.cboSchaatser
            .Fields("GeredenAfstand") = Me.txtAfstand
            .Fields("RondeTijd") = sRondeTijd
            .Fields("TussenTijd") = Me.txtEindTijd
            .Update
        End With
With rsADO
    .MoveFirst
    Do While Not .EOF
        With CurrentDb.OpenRecordset("tRittijden")
            .AddNew
            .Fields("RitID") = rsADO.Fields("RitID")
            .Fields("SchaatserID") = rsADO.Fields("SchaatserID")
            .Fields("GeredenAfstand") = rsADO.Fields("GeredenAfstand")
            .Fields("RondeTijd") = rsADO.Fields("RondeTijd")
            .Fields("TussenTijd") = rsADO.Fields("TussenTijd")
            .Update
            .Close
        End With
        .MoveNext
    Loop
End With
    Else
        RoundMilliSec = Nz(Me.txtElapsed, 0)
        Me.txtElapsed = ElapsedMilliSec
        TotalRoundMilliSec = ElapsedMilliSec - RoundMilliSec
        sRondeTijd = Tijd(TotalRoundMilliSec)
        ElapsedMilliSec = (GetTickCount() - StartTickCount) + TotalElapsedMilliSec
        sTussenTijd = Tijd(ElapsedMilliSec)
        TotalElapsedMilliSec = TotalElapsedMilliSec + (GetTickCount() - StartTickCount)
        Me.txtRonde = Me.txtRonde + 1
```

```
        Me.txtRondeTijd = sRondeTijd

        With rsADO
            .AddNew
            .Fields("RitID") = Me.Schaatser
            .Fields("Ronde") = iRonde
            .Fields("SchaatserID") = Me.cboSchaatser
            .Fields("GeredenAfstand") = Me.txtAfstand
            .Fields("Rondetijd") = sRondeTijd
            .Fields("Tussentijd") = sTussenTijd
            .Update
        End With

        Me.txtAfstand = Me.txtAfstand + 400
        If CInt(Me.txtAfstand) = CInt(Me.cboAfstand.Column(1)) Then
            Me.lblAfstand.Caption = "Eindtijd na:"
        End If
    End If

End Sub
```

Opdracht

De procedure zoals hierboven beschreven is niet heel erg nauwkeurig; dat komt doordat er eigenlijk nog teveel handelingen plaatsvinden. Je kunt dus nog winst boeken door de hele procedure te stroomlijnen.

Kijk dus waar er in jouw ogen overbodige handelingen plaatsvinden, en probeer die te elimineren. Controleer vervolgens wat het effect van de aanpassingen is.

Samenvatting

In dit hoofdstuk hebben we een formulier gemaakt waarin met behulp van een tijdelijke recordset rondetijden worden opgeslagen, die na het afronden van de race worden opgeslagen in een vaste tabel. Hierbij worden twee soorten recordsets gebruikt: een ADO-recordset, en een DAO-recordset. Voor de meeste handelingen die je zult verrichten kun je ofwel een ADO-recordset maken, ofwel een DAO-recordset. Van de twee kun je alleen met ADO een virtuele tabel maken. Wil je een formulier koppelen aan een tijdelijke tabel, dan zul je dus altijd een ADO-recordset moeten gebruiken.