Cursus Access voor Beginners – Hoofdstuk 2

Handleiding van Helpmij.nl

Auteur: OctaFisH

April 2011

Cursus Access voor Beginners – Hoofdstuk 2

Auteur: OctaFisH

In deze aflevering gaan we verder met de theorie over het opzetten van een database. We bekijken het proces van het Normaliseren, en we gaan zien hoe onze database kan worden worden verbeterd volgens deze normen. We beginnen met het opzetten van de relaties tussen de tabellen. In Hoofdstuk 3 gaan we dan echt aan het werk, en maken we een begin met de Frontend en Backend versies van de database.

Normaliseren

Bij het opzetten van een database streven we er naar om zo min mogelijk gegevens te herhalen binnen de db. We noemden dit in het eerste hoofdstuk *Redundantie*. En het was iets dat zoveel mogelijk vermeden moest worden.

Wat is een Relationele database?

In het eerste hoofdstuk is het begrip *Relationele database* al een paar keer gevallen. Voordat we verder gaan, eerst een poging om dat iets dieper uit te leggen.

Een relationele database is in beginsel een database waarin *verzamelingen* worden gemaakt van gegevens. Het eerste basisingrediënt van een relatie noemen we een *Atribuut*, of *Kenmerk.* In de spreektaal binnen Access noemen we een attribuut doorgaans een *Veld*. Typische voorbeelden zijn: telefoonnummer, kleur, aantal etc. Attributen hebben Attribuuteigenschappen, die we in het databaseontwerp vastleggen. Een eigenschap van een attribuut is, dat het attribuut *atomair* is: elk attribuut mag maar één waarde bevatten.

Het tweede basisingrediënt is een *tupel*; dit is een geordend aantal attribuutwaarden. In Access noemen we een tupel meestal een Record.

Een *relatie* is een *verzameling tupels* van dezelfde attributen. Anders gezegd: een relatie is wat wij in de spreektaal een *tabel* noemen. De gegevens die binnen één regel bij elkaar horen; anders gezegd: hebben een *relatie* met elkaar. Binnen een relatie kunnen geen twee rijen identiek zijn. Duplicaten van een tupel zijn dus niet mogelijk. Een andere eigenschap van een relatie is dat er geen voorgedefinieërde volgorde is. Oftewel: een verzameling is niet gesorteerd.

In een database is het mogelijk om tabellen met elkaar te verbinden: er wordt een relatie gelegd tussen twee of meer tabellen. In de spreektaal heet zo'n verbinding jammer genoeg ook een relatie, waardoor het misverstand is ontstaan dat een relationele database een database is waarin je tabellen aan elkaar kunt koppelen. Dat is dus niet zo: een relationele database is gebaseerd op de relationele algebra, en heeft verder niets met tabelkoppelingen te maken.

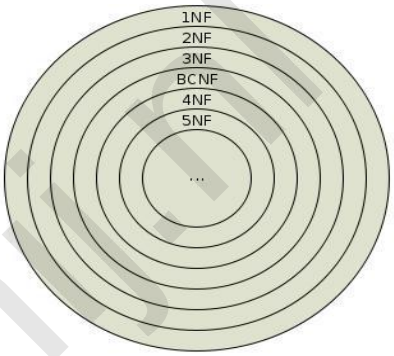
In het Engels is dat onderscheid een stuk duidelijker: daar heet een verzameling gegevens een *Relation*, en een koppeling tussen tabellen een *Relationship*. Twee verschillende begrippen dus!

In deze cursus zullen we het meestal hebben over records i.p.v. tupels, en over velden i.p.v. attributen, ook al zit er wel een (theoretisch) verschil tussen de verschillende begrippen. Je moet echter wel weten waar de begrippen voor staan, omdat ze in de theorie over normaliseren, en relationeel database ontwerp regelmatig worden gebruikt.

Het normaliseren van een database

Een proces om gegevensverzamelingen te optimaliseren is het *normaliseren* van de tabellen. Daarbij wordt per tabel bekeken welke gegevens we willen gebruiken, en hoe die gegevens zich verhouden tot de overige gegevens in dezelfde tabel. We onderscheiden daarbij een aantal niveau's:

- 1e Normaalvorm
- 2e Normaalvorm
- 3e Normaalvorm
- 4e Normaalvorm
- 5e Normaalvorm



Deze normalisatie vorm is vastgelegd in de Boyce-Codd normaalbeschrijving. In het lijstje geldt dat elk opvolgend niveau het vorige niveau overerft, en er nieuwe voorwaarden aan toevoegt.

De 2e normaalvorm voldoet dus aan de 1e normaalvorm en voegt nieuwe eisen toe; de 3e normaalvorm voldoet aan de 2e normaalvorm (en dus automatisch ook aan de 1e) en voegt nieuwe voorwaarden toe.

Meestal wordt een database genormaliseerd tot de 3e normaalvorm; dat is voor een gemiddelde database doorgaans genoeg. In deze cursus gaan we dan ook niet verder dan de 3e normaalvorm.

De 1e normaalvorm (1NV)

Een tabel staat in de eerste normaalvorm als:

- elk volledig record (tupel in de verzameling) uniek is
- Elk veld (attribuut in een tupel) een waarde bevat die niet-deelbaar (atomair) is
- Geen enkel attribuut wordt herhaald

Dat klinkt logisch, maar vereist toch enige uitleg. Je kunt bijvoorbeeld een niet-genormaliseerde tabel Artikelen hebben, met daarin de volgende gegevens:

Artikelnaam	Kleur1	Kleur2	Kleur3	Prijs	Magazijn	Voorraad
Jas	Marine	Beige	Grijs	120	1	200
Jas	Marine	Beige	Grijs	120	2	140
Broek	Zwart	Grijs	Navy	95	1	60
Broek	Zwart	Grijs	Navy	95	2	94

In deze tabel vinden we de artikelen Jas en Broek, die beide maar in drie kleuren te leveren zijn. Om een vierde kleur in het assortiment op te nemen, zou er een veld [Kleur4] aan de tabel moeten worden toegevoegd. En die handeling moet dan elke keer als er een nieuwe kleur van een product bijkomt worden uitgevoerd. Hierbij bepaalt het artikel met de meeste kleuren uiteindelijk dus de grootte van de tabel.
Deze situatie is niet optimaal. Het is beter om voor de kleuren één veld te hebben, zoals in onderstaand voorbeeld. Hierin is ook een veld gemaakt voor de verschillende Maten waarin een kledingstuk verkrijgbaar is, want daarvoor geldt hetzelfde probleem als voor de kleuren.

Artikelnaam	Kleur	Maat	Prijs	Magazijn	Voorraad
Jas	Marine, Beige	36;38;40;42;44;46	120	1	200
Jas	Marine, Beige	36;38;40;42;44;46	120	2	140
Broek	Zwart, Grijs	S;M;L;XL	95	1	60
Broek	Zwart, Grijs	S;M;L;XL	95	2	94

Ook deze tabel kunnen we niet genormaliseerd noemen. We hebben nu namelijk twee velden (Kleur, Maat) die meerdere gegevens bevatten, namelijk voor elk kledingstuk verschillende kleuren en maten. De velden [Kleur] en [Maat] zijn dus deelbaar, en dat houdt gelijk in dat deze tabel niet genormaliseerd is. Ook deze tabel staat in de *0e normaalvorm*. Om deze tabel te normaliseren naar de 1e normaalvorm, moeten we de gegevens in de velden [Kleur] en [Maat] dus opsplitsen. Omdat één van de regels is dat een attribuut niet mag worden herhaald, mogen we maar één veld Kleur gebruiken, en één veld Maat. Dat houdt in, dat we voor elke combinatie van kleur en maat aparte tupels (records) maken.

De (verkorte) tabel ziet er dan zo uit:

Artikelnaam	ArtGroep	Kleur	Maat	Prijs	Magazijn	Voorraad	Waarde
Jas	A100	Marine	36	120	1	6	720
Jas	A100	Beige	36	120	1	15	1800
Jas	A100	Marine	38	120	1	15	1800
Jas	A100	Beige	38	120	1	11	1320
Jas	A100	Marine	42	120	1	16	1920
Jas	A100	Beige	42	120	1	14	1680
Jas	A100	Marine	36	120	2	17	2040
Jas	A100	Beige	36	120	2	9	1080
Jas	A100	Marine	38	120	2	4	480
…							
Broek	A112	Zwart	XL	95	2	12	1140
Broek	A112	Grijs	XL	95	2	8	760

Zoals je in dit voorbeeld kunt zien, moeten ook de voorraadgetallen worden aangepast; in de eerste tabel werd voor elk artikel in het magazijn één record gemaakt met dus ook één waarde voor de voorraad. De voorraad voor het artikel was een totaal van alle voorraden voor alle kleuren en maten van het artikel. Op het moment dat we de tabel gaan normaliseren naar de 1e normaalvorm, zijn er meer records nodig om alle artikelen te beschrijven, namelijk één record voor elke kleur en maat van elk artikel. Tevens kunnen we nu voor elke kleur en maat van elk artikel apart de voorraad bijhouden, wat uiteraard een groot voordeel is, want we kunnen nu veel gerichter artikelen bijbestellen.

Een belangrijk doel van normaliseren is om in een tabel een *Sleutel* te kunnen definiëren. Zoals we in het niet-genormaliseerde voorbeeld zagen, kon één artikel, zoals een Jas in de kleur Marine van maat 42 niet specifiek worden opgezocht in de tabel. Dat kwam doordat de maten en kleuren bij elkaar waren gevoegd. Door deze gegevens te splitsen, is het wèl mogelijk om binnen de artikelentabel één record apart op te zoeken. Daarvoor hebben we vier gegevens nodig: de velden [Artikelnaam], [Kleur], [Maat] en [Magazijn]. Deze combinatie definiëren we derhalve als de Primaire sleutel binnen de tabel: elke combinatie van [Artikelnaam], [Kleur], [Maat] en [Magazijn] levert altijd één record op: niet meer, en niet minder.

Omdat deze tabel lastig is te onderhouden (zeker later, in combinatie met andere tabellen) is het verstandig om een extra veld aan de tabel toe te voegen met daarin een Artikelnummer. De tabel ziet er dan zo uit:

ArtNo	**ArtGroep**	***Artikelnaam***	***Kleur***	***Maat***	***Prijs***	***Magazijn***	***Voorraad***	**Waarde**
A12	A100	Jas	Marine	**36**	120	**1**	6	720
A12	A100	Jas	Marine	**36**	120	**2**	17	2040
A13	A100	Jas	Beige	**36**	120	**1**	15	1800
A13	A100	Jas	Beige	**36**	120	**2**	9	1080
A12	A100	Jas	Marine	**38**	120	**1**	15	1800
A12	A100	Jas	Marine	**38**	120	**2**	4	480
A13	A100	Jas	Beige	**38**	120	**1**	11	1320
A13	A100	Jas	Beige	**38**	120	**1**	14	1680
		…						
A71	A112	Broek	Zwart	**XL**	95	**2**	12	1140
A72	A112	Broek	Grijs	**XL**	95	**2**	8	760

Hierbij is voor artikel artikel+kleur een aparte code gemaakt. Hoewel de oorspronkelijke sleutel ([Artikelnaam], [Kleur], [Maat] en [Magazijn]) nog steeds geldt, en dus ook nog steeds gebruikt kan worden, is er nu een kortere sleutel te bedenken: [ArtNo], [Maat] en [Magazijn]. De oorspronkelijke sleutel is nu ‘gedegradeerd’ tot *alternatieve sleutel* (zie hoofdstuk 1). De nieuwe sleutel is nu de primaire sleutel. In het plaatje zijn deze velden geel gemarkeerd.

De 2e normaalvorm (2NV)

In de tweede normaalvorm kijken we naar de mate waarin de verschillende attributen (velden) afhankelijk zijn van de sleutel van de tabel. In de tabel mogen alleen attributen zitten, die *volledig afhankelijk zijn van de sleutel*.

Als we de tabel bekijken, dan constateren we dat de tabel in zijn huidige vorm de voorraden van de verschillende artikelen in de verschillende magazijnen bijhoudt. In een tabel die voldoet aan de 2^{e} normaalvorm, zitten alleen attributen die afhankelijk zijn van de sleutel. Zoals het veld [Voorraad]: dit bepaalt namelijk hoeveel exemplaren van elk artikel (per maat) in voorraad is. Voor de voorraad is het echter niet belangrijk om de *kleur* van het artikel te weten: we hebben namelijk voor elk artikel in een bepaalde kleur een eigen artikelnummer gemaakt. Hetzelfde geldt voor het veld [Prijs]: de prijs is voor elk artikelnummer hetzelfde, ongeacht de voorraad, of in welk magazijn het artikel ligt.

Oftewel: om de tabel naar de 2e normaalvorm te promoveren, moeten er velden uit de tabel worden verwijderd. En wel de velden die *niet* afhankelijk zijn van de sleutel van de tabel. We moeten de tabel (in dit geval) opsplitsen in twee tabellen: een tabel [Artikel_Voorraad] en een tabel [Artikel]. De nieuwe tabellen zien er dan ongeveer zo uit:

Tabel **[Artikelen]**

ArtNo	**Maat**	**ArtGroep**	**Artikelnaam**	**Kleur**	**Prijs**
A12	36	A100	Jas	Marine	120
A13	36	A100	Jas	Beige	120
A12	38	A100	Jas	Marine	120
A13	38	A100	Jas	Beige	120
A12	40	A100	Jas	Marine	120
A13	40	A100	Jas	Beige	120
…					
A71	XL	A112	Broek	Zwart	95
A72	XL	A112	Broek	Grijs	95

En de tabel **[Artikel-Voorraad]**

ArtNo	**Maat**	**Magazijn**	**Voorraad**	**Mutatiedatum**	**Waarde**
A12	36	1	6	11-11-2010	720
A12	36	2	17	03-05-2010	2040
A13	36	1	15	04-09-2010	1800
A13	36	2	9	29-12-2010	1080
A12	38	1	15	13-06-2010	1800
A12	38	2	4	21-11-2010	480
A13	38	1	11	21-10-2010	1320
A13	38	2	11	21-10-2010	1320
…					
A71	XL	2	12	08-07-2010	1140
A72	XL	2	8	09-11-2010	760

De tabellen krijgen een relatie op basis van de velden [ArtNo] en [Maat]; deze twee velden komen in beide tabellen voor. In de tabel [Artikelen] is de combinatie uniek: elke samenstelling van een artikelnummer en een kledingmaat levert één uniek Tupel (record) op. Hiermee is deze veldencombinatie de primaire sleutel van de tabel.

Dezelfde velden vormen in de tabel [Artikel-Voorraad] geen unieke combinatie, omdat er meerdere magazijnen zijn waarin de artikelen voorkomen. Om een sleutel te kunnen definiëren, is dus nog een extra veld nodig: het veld [Magazijn]. Dit is dezelfde sleutel als in de tabel toen die nog in de 1e normaalvorm stond. Het enige verschil met die tabel is, dat de attributen die niet afhankelijk waren van de sleutel, uit de tabel zijn verwijderd. Daarmee staan de twee tabellen dus allebei in de 2e normaalvorm.

In het voorbeeld is nog een extra veld toegevoegd: [Mutatiedatum]. Omdat dit veld volledig afhankelijk is van de primaire sleutel (het zegt iets over de datum waarop de laatste

artikelmutatie in een magazijn heeft plaatsgevonden) heeft het toevoegen van dit veld geen consequenties voor de normaalvorm van de tabel; dit is nog steeds een tabel in de 2e normaalvorm.

De 3e normaalvorm (3NV)

In de derde normaalvorm zijn alle attributen (velden) afhankelijk van de primaire sleutel, en zijn er geen gegevens die zijn af te leiden uit andere gegevens in de tabel; elk (niet-sleutel) attribuut is volledig functioneel onafhankelijk van alle overige attributen (m.a.w. er is géén indirecte sleutelafhankelijkheid)

De tabel [Artikel-Voorraad] voldoet na de aanpassing dus nog niet aan de 3e normaalvorm, omdat weliswaar alle velden afhankelijk zijn van de sleutel, maar het veld [Waarde] kan worden afgeleid uit de artikelprijs en de voorraadwaarde. Om deze tabel in de 3e normaalvorm te zetten, hoeven we alleen maar het veld [Waarde] te verwijderen.

ArtNo	Maat	Magazijn	Voorraad	Mutatiedatum
A12	36	1	6	11-11-2010
A12	36	2	17	03-05-2010
A13	36	1	15	04-09-2010
A13	36	2	9	29-12-2010
A12	38	1	15	13-06-2010
A12	38	2	4	21-11-2010
A13	38	1	11	21-10-2010
…				
A71	XL	2	12	08-07-2010
A72	XL	2	8	09-11-2010

Ook de tabel [Artikelen] bevat nog een aantal velden waarvoor dit geldt dat ze weliswaar iets zeggen over de sleutel, maar die afhankelijk zijn van een niet-sleutelgegeven, zoals het veld [Prijs]. Om dat duidelijk te maken, kijken we naar het veld [Artikelgroep]:

ArtNo	ArtGroep	Maat	Artikelnaam	Kleur	Prijs
A12	A100	36	Jas	Marine	120
A13	A100	36	Jas	Beige	120
A12	A100	38	Jas	Marine	120
A13	A100	38	Jas	Beige	120
A12	A100	40	Jas	Marine	120
A13	A100	40	Jas	Beige	120
…					
A71	A112	XL	Broek	Zwart	95
A72	A112	XL	Broek	Grijs	95

Als in dit voorbeeld de prijs van de jas moet worden aangepast, betekent dit dat van alle artikelen die vallen in de artikelgroep A100 de prijs moet worden aangepast. De prijs is dus niet afhankelijk van de primaire sleutel ([ArtNo]+[Maat]), maar van het veld [ArtGroep]. In een tabel in de 3e normaalvorm worden de gegevens die afhankelijk zijn van een niet-

sleutelveld uit de tabel gehaald, en in een aparte tabel gezet. De tabellen komen er dan zo uit te zien:

Tabel [**Artikelen**]

ArtNo	**ArtGroep**	**Maat**	**Artikelnaam**	**Kleur**
A12	A100	36	Jas	Marine
A13	A100	36	Jas	Beige
A12	A100	38	Jas	Marine
A13	A100	38	Jas	Beige
…				
A71	A112	XL	Broek	Zwart
A72	A112	XL	Broek	Grijs

En de nieuwe tabel [**ArtikelPrijzen**]:

ArtGroep	**Prijs**
A100	120
A109	80
A112	95
A115	175

De tabellen worden aan elkaar gekoppeld op basis van het veld [ArtGroep].

Als er nu een prijs moet worden aangepast, hoeven we alleen maar in de tabel [ArtikelPrijzen] de prijs te veranderen. Doordat de tabellen aan elkaar zijn gekoppeld, is de nieuwe artikelprijs nu eenvoudig af te leiden door de prijs op basis van het veld [ArtGroep] op te halen uit de tabel [ArtikelPrijzen]. Tevens is, na een prijsaanpassing, de nieuwe waarde van de artikelvoorraad ook bekend, want ook die is af te leiden uit de nieuwe tabel; we kunnen namelijk de prijs opzoeken op basis van het Artikelnummer in de tabel [ArtikelVoorraad].

Samenvatting

Een *relationele database* is een database waarin gegevens worden geplaatst in *Verzamelingen*. Een verzameling bestaat uit een eenduidige verzameling *Attributen* met bepaalde *Eigenschappen*. Elke unieke combinatie van een rij met attributen heet een *tupel*. Tussen verzamelingen kunnen koppelingen worden gemaakt.

In Microsoft Access vertalen we *Verzameling* naar *Tabel*, het begrip *Attribuut* naar *Veld*, en het begrip *Tupel* naar *Record*. Tussen tabellen kunnen Koppelingen worden gelegd, die in Access *Relaties* worden genoemd.

Bij het normaliseren van een database proberen we een database zodanig in te richten, dat de gegevens zo efficiënt mogelijk worden opgeslagen, en gegevensredundantie zoveel mogelijk wordt vermeden. We proberen daarbij onze gegevens zoveel mogelijk bij elkaar te groeperen in verschillende tabellen op basis van logische afhankelijkheden. Zo sla je in een tabel

Artikelen alleen relevante artikelgegevens op, en in een tabel Klanten alleen zinvolle klantgegevens.

Elke tabel krijgt een primaire sleutel, waarmee een record kan worden geïdentificeerd en opgehaald. Daar waar combinaties van gegevens worden opgeslagen, zoals in een tabel Bestellingen waarin een klant een artikel kan bestellen, worden in die tabel alleen de koppelwaarden zoals (uniek) Artikelnummer en (uniek) klantnummer opgeslagen. Elk record in zo'n combinatietabel is ook weer uniek.

Het normaliseren bestaat uit een aantal stappen, waarbij elke volgende stap de instellingen uit de vorige stap overneemt.

- **1e normaalvorm**
  Verwijder alle herhalende kolommen uit de tabel
  Maak aparte tabellen voor elke groep van bij elkaar horende gegevens.
  Geef elke tabel een veld of combinatie van velden die uniek is; de primaire sleutel
- **2e normaalvorm**
  Verwijder (groepen van) gegevens die herhaaldelijk terugkomen in de tabel, en plaats ze in een eigen tabel. Leg een koppeling tussen de nieuwe tabel en de oorspronkelijke tabel
- **3e normaalvorm**
  Verwijder gegevens die afhankelijk zijn van andere gegevens in de tabel, en die berekend kunnen worden.
  Verwijder gegevens die afhankelijk zijn van een niet-sleutelveld in de tabel, en plaats ze in een eigen tabel. Leg een koppeling tussen de nieuwe tabel en de oorspronkelijke tabel.

Hoewel het in beginsel verstandig is om een database zoveel mogelijk volgens de 3e normaalnorm te bouwen, kan het soms zinvol zijn om van deze principes af te wijken. Zo is één van de regels van de 3e normaalvorm dat gegevens die berekend kunnen worden niet worden opgeslagen in een tabel. In een tabel met factuurgegevens kan het echter toch zinvol zijn om de uiteindelijke bedragen (al dan niet gedeeltelijk) op te nemen. Zo zijn verkoopprijzen doorgaans niet gefixeerd, en veranderen ze in de loop van de tijd. Als de verkoopprijs op het moment van aankoop niet is vastgelegd in de tabel, zou een berekening van de totaalprijs anders kunnen zijn als de gegevens worden afgedrukt voor bijvoorbeeld een jaaroverzicht. Deze situatie is uiteraard niet wenselijk.

Een oplossing voor dit probleem zou kunnen zijn om de prijsmutaties bij te houden in een tabel, waarbij een ingangsdatum wordt gebruikt voor de prijzen. De juiste prijs kan dan worden opgezocht a.d.h.v. deze tabel. Deze oplossing is goed te maken, maar vereist meer kennis dan een oplossing waarbij de actuele prijs in de verkooptabel wordt opgeslagen.

Een ander voorbeeld kan zijn, dat in de Artikelen tabel een aantal afbeeldingen wordt toegevoegd voor de producten. Hiervoor zou (volgens de 1e normaalvorm) een aparte tabel moeten worden gemaakt, zodat het aantal afbeeldingen niet aan een limiet is gebonden. Als je maar een maximum van vier afbeeldingen wilt gebruiken, is er weinig op tegen om daarvoor vier velden in de tabel op te nemen. Deze oplossing maakt het artikelenformulier veel eenvoudiger te maken en onderhouden, en dat vergroot dus het gemak voor de ontwerper dus aanzienlijk.

Conclusie: het is zonder meer aan te bevelen om een database te ontwerpen volgens de Boyce-Codd normaalvorm. Het kan echter voorkomen dat het beter is om van deze normen af te wijken. Dat kan zijn om de prestaties binnen de database te verhogen, of omdat het ontwerpen van formulieren en rapporten daardoor eenvoudiger wordt.

Opdracht

De opdracht bij deze les heeft uiteraard te maken met het normaliseren van tabellen.

- Controleer of de tabellen die je hebt gemaakt in de vorige les voor de Duikclub voldoen aan tenminste de 3e Normaalvorm
- Als je denkt dat een (of meer) tabel(len) niet voldoen aan de 3e normaalvorm, splits de betreffende tabel dan op in twee tabellen, waarbij je de nieuwe tabellen met koppelingen aan elkaar relateert, waarbij Referentiële Integriteit wordt afgedwongen
- Als je met extra tabellen wilt oefenen: op het HelpMij forum is een aparte Topic gemaakt voor deze cursus, waarop oefenbestanden staan: http://www.helpmij.nl/forum/showthread.php/591915-Vragen-Opmerken-Reacties-en-Tips-voor-de-Access-cursus

Volgende Aflevering

In de volgende aflevering behandelen we het principe van een Frontend-Backend database. We maken daarbij van onze Duikdatabase een gesplitste database. Ook beginnen we met het maken van een connectie kan maken met een database met behulp van VBA.