**Databases ontwerpen en normaliseren**

**oktober 2021**

INHOUD

[1 Introductie 3](#_Toc21990315)

[2 Databasegeschiedenis 3](#_Toc21990316)

[3 Voordelen en mogelijkheden van relationele databases 6](#_Toc21990317)

[3.1 Tabellen en de primaire sleutel 7](#_Toc21990318)

[3.2 Gegevens koppelen 8](#_Toc21990319)

[3.3 De een op veel relatie 9](#_Toc21990320)

[3.4 De veel op veel relatie 10](#_Toc21990321)

[3.5 De één op één relatie 12](#_Toc21990322)

[4 Database normaliseren 13](#_Toc21990323)

[5 Uitgewerkte methodiek met voorbeeld 14](#_Toc21990324)

[5.1 Nulde Normaalvorm (0 NV) 15](#_Toc21990325)

[5.2 Van de 0 NV naar de Eerste Normaalvorm 1 NV 16](#_Toc21990326)

[5.3 Van de 1 NV naar de Tweede Normaalvorm (2 NV) 19](#_Toc21990327)

[5.4 Van de 2 NV naar de Derde Normaalvorm (3 NV) 22](#_Toc21990328)

[5.5 Begrippen 25](#_Toc21990329)

# Introductie

In deze cursus **leer je een database ontwerpen**. Databases zijn programma's die gegevens opslaan en beschikbaar maken. Veel programma's maken gebruik van een database.

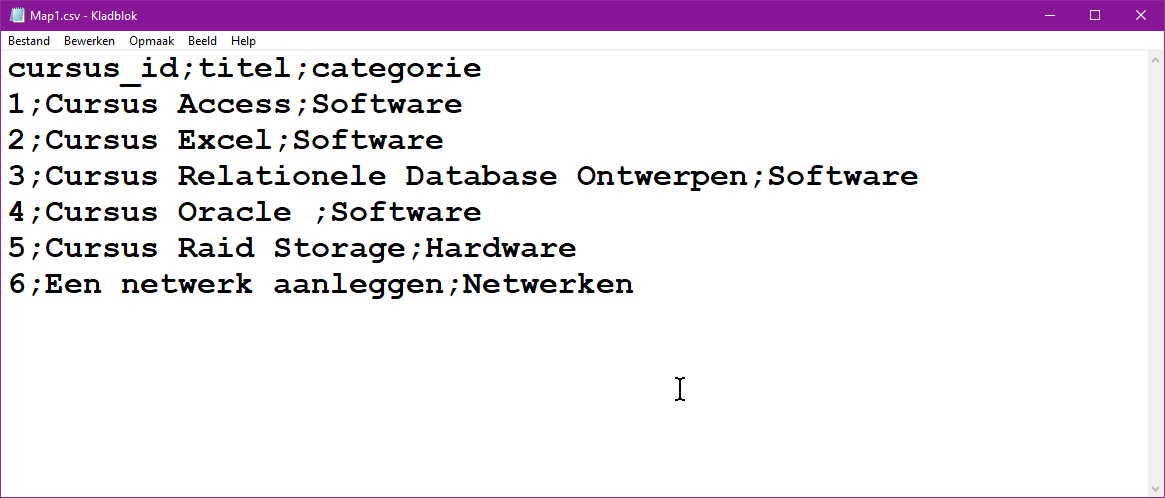
Je maakt zelf waarschijnlijk dagelijks gebruik van databases. **Als je geld opneemt bij de pinautomaat** raadpleeg je de database van de bank, als je **websites op internet** bezoekt raadpleeg je een database waar (o.a.) **paginateksten** in opgeslagen zijn, als je **een nieuw paspoort** aanvraagt wordt ongetwijfeld een database met persoonsgegevens geraadpleegd, etc.

In deze cursus relationele database ontwerpen leer je zelf een database ontwerpen volgens het **relationeel model**.

# Databasegeschiedenis

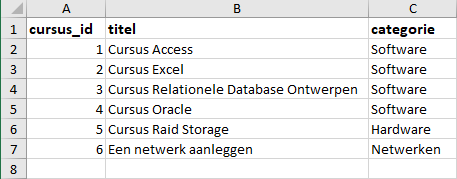
Vroeger, toen informatici nog stoffige jasjes en grote brillen droegen, werden gegevens opgeslagen in **'platte bestanden'**, in het Engels **'flat files'**. Flat files zijn tekstbestanden waarin gegevens gescheiden door komma's, tabs of nog andere scheidingstekens of *delimiters* zijn opgenomen.

Met behulp van een programma kunnen deze gegevens uit het bestand gelezen worden. Het programma herkent de scheidingstekens en kan zo gegevens uit het bestand selecteren.



Deze flat files doen hun werk, maar **erg efficiënt zijn ze niet** als ontwerp voor een database.

Om bijvoorbeeld de 'Cursus Relationele Database Ontwerpen' te selecteren in bovenstaande flat file moet het programma alle regels stuk voor stuk afzoeken totdat de cursus gevonden is.



Door in deze tabel naar een **cursus\_id** te zoeken in de cursus\_id-kolom kunnen snel de bijhorende **titel** en **categorie** opgevraagd worden. Dat gaat veel sneller dan door een tekstbestand heen wandelen totdat je de gewenste informatie gevonden hebt. Bij een relationele database kunnen gegevens uit specifieke rijen, kolommen en zelfs uit verschillende tabellen gecombineerd opgevraagd worden.

Je kunt het **cursus\_id** (de 'primaire sleutel') gebruiken om de cursus te koppelen aan gegevens in andere tabellen. Hierover later meer.

Het concept van de relationele database waarin gegevens zijn geordend in tabellen is bedacht in de jaren 70 door **Ted Codd**. De relationele database is een zeer krachtige methode voor het opslaan van gegevens gebleken.

Tegenwoordig gebruiken heel veel programma's een relationele database om gegevens in op te slaan.

Tegenwoordig zijn er tal van verschillende **Relationele Database Management Systemen (RDBMS)** in gebruik.

Tot de toppers van de database-industrie behoren:

* **Oracle**. Oracle wordt in de regel gebruikt voor grotere professionele applicaties
* **Microsoft SQL server**. Het professioneel RDBMS van Microsoft wordt gebruikt voor alle soorten applicaties, van studentenapplicaties tot professionele applicaties met veel gebruikers.
* **MySQL:** Uit de officiële site (<www.mysql.com>): *Many of the world's largest and fastest-growing organizations including Facebook, Google, Adobe, Alcatel Lucent and Zappos rely on MySQL to save time and money powering their high-volume Web sites, business-critical systems and packaged software.* **MySQL** is gratis (of beter gezegd 'vrij') verkrijgbaar.



Een **databaseontwerp** maak je **niet** voor een bepaald RDBMS. Het ontwerp van de database is onafhankelijk van het databasesysteem, zolang je een relationeel databasesysteem (RDBMS) gebruikt.

# Voordelen en mogelijkheden van relationele databases

Naast het snel opzoeken van informatie heeft een relationele database nog een aantal **voordelen:**

* Een relationele database kan gegevens in verschillende tabellen relateren door het gebruik van **sleutels**. Daardoor kunnen gerelateerde gegevens uit verschillende tabellen tegelijk opgevraagd worden.
* **Efficiënte opslag van gegevens**. Gegevens worden maar één keer en op één plek opgeslagen. Dit zorgt ervoor dat een wijziging of verwijdering van informatie altijd maar op één plek hoeft te gebeuren.
* Met een relationele database kan je **regels** opstellen voor het **soort gegevens** dat in een veld opgeslagen kan worden. Zo kun je onder andere datumvelden, tekstvelden en numerieke velden aanmaken.
* **Integriteit van gegevens**. Door het formuleren van de juiste relaties van tabellen kun je de integriteit (correctheid, betrouwbaarheid) van de gegevens die zijn opgeslagen beter garanderen.
* De meeste relationele databasesystemen kennen een **rechtenstructuur**, waarmee aan verschillende gebruikers verschillende rechten toegekend kunnen worden. Zo kun je het recht hebben om gegevens uit de database te op te vragen, maar niet om nieuwe gegevens in te voeren.
* Relationele databases zijn geavanceerde programma's die **geoptimaliseerd zijn voor bepaalde taken**, zoals bijvoorbeeld het zoeken door numerieke velden, data sorteren, etc. Deze technologie maakt het mogelijk om zeer snel gegevens te zoeken en te presenteren.
* Voor het bevragen van een relationele database is een krachtige 'query taal' beschikbaar met de naam **'Structured Query Language'** (SQL). Deze taal maakt het onder andere mogelijk om zeer geavanceerde gegevensselecties te maken uit de database.
* **Standaardisering** maakt het mogelijk om gegevens uit de ene database vaak relatief eenvoudig naar de andere database over te zetten.

## Tabellen en de primaire sleutel

Een relationele database bestaat uit **tabellen**. In deze tabellen zijn gegevens van dezelfde soort in **rijen** of in vaktaal **records** opgeslagen.

Om bijvoorbeeld klanten uniek te kunnen identificeren moet elke klant gekoppeld zijn aan een uniek stukje informatie, bijvoorbeeld een klantnummer. Deze unieke informatie wordt de **primaire sleutel** **(primary key)** genoemd.

Een van de belangrijkste regels van het relationeel model is dat alle rijen in een tabel **uniek** te identificeren zijn door middel van de primaire sleutel.

Er zijn tal van **voorbeelden** te bedenken van nummers en codes uit het dagelijks leven die waarschijnlijk dienen als primaire sleutel in een database.

* Een bestelnummer
* Een rekeningnummer
* Een factuurnummer
* Een klantnummer
* Een productnummer

Wat hebben al deze codes gemeen?

* Ze zijn allemaal **uniek**. Jouw rekeningnummer bestaat altijd maar één keer, net als een factuurnummer, je sofinummer, etc.
* Ze werken allemaal als toegangsweg naar meer informatie.
* Aan een factuurnummer is een datum, een bedrag, etc. gekoppeld.
* Aan een productnummer kan een productbeschrijving, een plaatje, etc. gekoppeld zijn.

**De primaire sleutel** wordt dus gebruikt om rijen in tabellen **uniek te identificeren** en om **gegevens in verschillende tabellen aan elkaar te koppelen.**

Primaire sleutels zijn zoals gezegd altijd uniek en mogen daarom maar 1 keer voorkomen in de kolom. Het is in onderstaande tabel dus niet mogelijk om een nieuwe rij met klantnummer 1 toe te voegen. Geef je aan dat het klantnummer de primaire sleutel is, dan zal het databasesysteem er automatisch voor zorgen dat klantnummers in die kolom uniek zijn.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **klantnummer** | **voornaam** | **tussenvoegsel** | **achternaam** | **e-mailadres** | **telefoonnummer** |
| **1** | Jan |  | Jansen | janjansen@hotmail.com | 612345678 |
| **2** | Klaas | van | Baalen | kvb@kvbmail.be | 20123456 |

## Gegevens koppelen



Op elke rij van deze **contactmomententabel** staat informatie over een uniek contactmoment. Elk contactmoment is uniek identificeerbaar door de primaire sleutel, het **contactmoment\_id** (eerste kolom). Bovendien is elk contactmoment gekoppeld aan een klant uit de klanttabel door middel van het **klantnummer** in de tweede kolom.

Met behulp van het klantnummer kunnen bij elke klant in de klanttabel de contactmomenten opgezocht worden en andersom bij elk contactmoment de betreffende klant uit de klanttabel. Er is hier sprake van een **'1 op veel' relatie** tussen de klant en zijn contactmomenten.

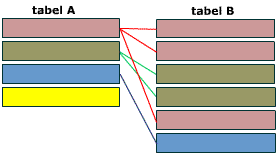
Nu je een voorbeeld gezien hebt van een database is het tijd om in te gaan op de **relaties** die kunnen bestaan tussen tabellen. Het voorbeeld van de klanttabel en de contactmomententabel is een 'één op veel relatie'. Voor elke klant (1) kunnen meerdere (veel) contactmomenten opgeslagen zijn.

Bij het ontwerpen van relationele databases onderscheiden we **3 soorten relaties** tussen tabellen:

* **De één op veel relatie**
* **De veel op veel relatie**
* **De één op één relatie**

## De een op veel relatie

Bij de **één op veel relatie** is 1 ding gekoppeld aan 0, 1 of meer andere dingen. Je zag al een voorbeeld met klanten en contactmomenten. 1 klant kan 0, 1 of meerdere contactmomenten hebben en een contactmoment is altijd gekoppeld aan 1 klant.



Hetzelfde geldt voor de **relatie tussen moeders en hun kinderen**. Elk kind heeft maar 1 moeder, maar een moeder kan geen, één of meerdere kinderen hebben. De één op veel relatie modelleer je in een relationele database (meestal) als **twee tabellen**. Elke rij in tabel A correspondeert met **0, 1 of meerdere** rijen uit tabel B. (zie afbeelding).

De klanttabel speelt de rol van tabel A en de contactmomententabel die van tabel B. Elke rij in tabel A (ofwel, elke klant) is gekoppeld aan 0,1 of meerdere rijen (ofwel, contactmomenten) in tabel B.

Het klantnummer is de **primaire sleutel** (**primary key**) in tabel A, het klantnummer in tabel B wordt de **vreemde sleutel** (**foreign key**) genoemd. **Een vreemde sleutel is een veld in een tabel dat verwijst naar de primaire sleutel van een andere tabel.** Het klantnummer in onze database is de primaire sleutel van de klanttabel. In andere tabellen is het klantnummer een vreemde sleutel.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **klantnummer** | **voornaam** | **tussenvoegsel** | **achternaam** | **e-mailadres** | **telefoonnummer** |
| **1** | Jan |  | Jansen | janjansen@hotmail.com | 612345678 |
| **2** | Klaas | van | Baalen | kvb@kvbmail.be | 20123456 |



Klantnummer is de **primaire sleutel (ps)** in de **klanttabel**

Klantnummer is de **vreemde sleutel (vs)** in de **contactmomententabel**. Hij verwijst naar het klantnummer in de klanttabel. Het contactmoment\_id is de **primaire sleutel (ps)** in de contactmomententabel.

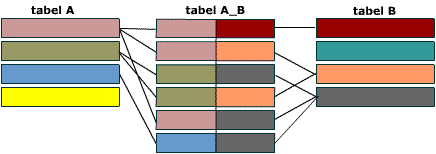
## De veel op veel relatie

De **veel op veel relatie** is een relatie waarbij meerdere rijen uit tabel A gekoppeld kunnen zijn aan meerdere rijen uit tabel B.

Een voorbeeld is een school, waarbij **docenten les geven aan studenten**. Elke docent geeft les aan 0, 1 of meer studenten. Andersom kan elke student les krijgen van 0, 1 of meerdere docenten.

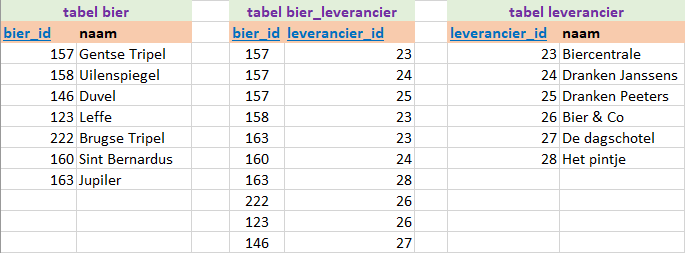
Een ander voorbeeld is de relatie tussen **biermerken en leveranciers**. Elke leverancier levert meerdere biermerken, maar elk biermerk kan ook door meerdere leveranciers geleverd worden.

De veel op veel relatie modelleer je met drie tabellen. **Twee brontabellen en één koppeltabel**.



Tabel A en tabel B zijn beide tabellen met een primaire sleutel. Tabel A\_B verbindt records uit tabel A en B en heet een **'koppeltabel'**.

Een koppeltabel bestaat uit maar twee kolommen die allebei een **vreemde sleutel** bevatten. De vreemde sleutel in de linkerkolom verwijst naar de primaire sleutel uit tabel A en die in de rechterkolom naar de primaire sleutel uit tabel B. De primaire sleutel in koppeltabel A\_B is samengesteld uit de twee vreemde sleutels. Je ziet in het voorbeeld dan ook dat de combinatie van de twee velden uniek moet zijn in koppeltabel A\_B. In het klantcontactsysteem komt (vooralsnog) geen veel op veel relatie voor, daarom staat hieronder een voorbeeld met bieren en hun leveranciers.



Bovenstaande tabellen koppelen bieren en leveranciers in een veel op veel relatie. Eén bier kan door 0, 1 of meerdere leveranciers geleverd worden:

* **Gentse Tripel (157)** wordt bijvoorbeeld geleverd door de leveranciers 23, 24 en 25, respectievelijk **Biercentrale**, **Dranken Janssens** en **Dranken Peeters**
* Andersom levert **Bier &Co** 2 bieren uit de bierentabel, te weten **Brugse Tripel** en **Leffe**

Merk op dat in bovenstaande tabellen de primaire sleutelvelden blauw en onderstreept zijn weergegeven. **In modellen van relationele databases worden primaire sleutels vaak onderstreept.**

Zoals je ziet is de primaire sleutel van de koppeltabel 'bier\_leverancier' samengesteld uit twee velden. Elke rij in de koppeltabel bestaat uit de combinatie van een bier\_id met een leverancier\_id.

Deze tabel bestaat zogezegd uit een **samengestelde primaire sleutel**. Primaire sleutels moeten uniek zijn, dus de **combinaties moeten uniek zijn** in de tabel. Het heeft geen zin om tweemaal dezelfde combinatie op te geven: dit levert geen nieuwe informatie op.

Een ander goed gebruik is om in databasemodellen **relaties te benoemen**. Je zou bij de hierboven afgebeelde relatie in een model de opmerkingen "levert" kunnen zetten. Elke rij in de tabel zegt namelijk "bier X wordt geleverd door leverancier Y", of "Leverancier Y levert bier X".

Het benoemen van relaties maakt een databasemodel leesbaarder. Deze benoeming komt niet terug in de eigenlijke database of in de software die gebruik maakt van de database.

## De één op één relatie

De **één op één relatie** tussen tabellen komt niet heel vaak voor. Bij de één op één relatie heeft elke rij in tabel A 0 of 1 corresponderende rij in tabel B. Deze relatie wordt weleens gebruikt om de tekortkomingen van een databasesysteem te omzeilen door bijvoorbeeld een tabel op te delen om prestatiewinst te behalen.

Merk op dat elke individuele tabel vol zit met 1 op 1 relaties: de relatie tussen de primaire sleutel en de rest van de gegevens op die rij is een 1 op 1 relatie, of laat ik zeggen zou dat moeten zijn als je het relationeel model helemaal volgt.

*Een voorbeeldje: in de biertabel op de vorige pagina is “Gentse Tripel” in een 1 op 1 relatie gekoppeld aan de primaire sleutel 157. Dit is precies de reden dat 1 op 1 relaties tussen tabellen niet vaak voorkomen.*

**Een relationeel databaseontwerp is zoals je hebt gelezen een verzameling van “entiteiten” of tabeldefinities die relaties hebben met elkaar.**

Voor het kiezen van de juiste relaties tussen gegevens formuleert het relationeel model een aantal 'normaalvormen'. In het volgende hoofdstuk ga ik in op het doel van normaliseren en de taken die hierbij horen.

# Database normaliseren

De regels voor goed relationeel databaseontwerp zijn samengevat in 5 **'normaalvormen'**, waarbij de eerste normaalvorm de laagste en de vijfde de hoogste – meest genormaliseerde – is. Deze normaalvormen zijn **richtlijnen** voor het juist ontwerpen van een relationele database.

Normaliseren heeft een aantal **doelen**:

* **Flexibiliteit**: de genormaliseerde structuur van de database zorgt ervoor dat gegevens op veel verschillende manieren opgevraagd en bijgewerkt kunnen worden
* **Integriteit**: in een genormaliseerde database komen geen anomalieën voor zoals: 2 verschillende adressen voor dezelfde persoon (terwijl dat in werkelijkheid niet het geval is), *wees-records* zoals detaillijnen van bestelbons die geen link hebben naar de bestelbon zelf, enz.
* In een genormaliseerde database worden gegevens **maar op 1 plek opgeslagen**. Als je data wil invoeren, aanpassen of verwijderen hoef je dat dus maar op 1 plek te doen.

Het normaliseren van een database schijnt voor veel mensen taaie materie te zijn. Het normaliseren van een database komt eigenlijk neer op het nastreven van de volgende zaken en die zijn met een beetje oefening en puzzelen vaak redelijk gemakkelijk te realiseren.

* Het verdelen van gegevens in **logische samenhangende groepen**.
* Het **minimaliseren van de hoeveelheid data die dubbel opgeslagen is**, ofwel het voorkomen van **'redundancy'** (overtollige informatie).
* De gegevens zo organiseren dat het aanpassen of verwijderen van een gegeven **altijd maar op één plek** hoeft te gebeuren.
* Gegevens zo organiseren dat ze **snel en efficiënt op te vragen** zijn.

De meeste applicaties gebruiken databases die zijn genormaliseerd tot de 1ste, de 2de of de 3de normaalvorm.

De 4de en 5de normaalvorm zie je zelden. In deze cursus databases wordt daarom alleen de eerste, tweede en derde normaalvorm besproken.

# Uitgewerkte methodiek met voorbeeld

Als je een database ontwerpt is het belangrijk dat die aan een aantal eisen voldoet. Eén daarvan is de eis dat het bestand optimaal ontworpen is. Dit betekent dat er geen **redundantie** mag zijn.

Er zijn meerdere stappen om naar deze goede database te komen. Deze stappen zijn **normaalvormen**.

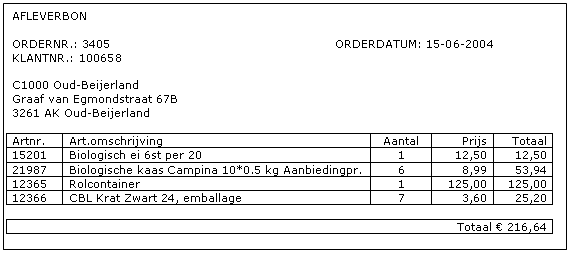
* 0 NV (Nulde Normaalvorm - Inventarisatie)
* 1 NV (Eerste Normaalvorm)
* 2 NV (Tweede Normaalvorm)
* 3 NV (Derde Normaalvorm)

Verder zijn er ook nog de BCNV (Boyce Codd Normaalvorm), 4NV en 5NV. Op deze laatste normaalvormen gaan we niet in omdat deze bijna nooit gebruikt worden.

Uitgangspunt voor het normaliseren is steeds de **informatiebehoefte** van de toekomstige gebruiker van de database. De indeling en inhoud van de tabellen wordt bepaald door de informatie die de gebruiker wenst te zien.

## Nulde Normaalvorm (0 NV)

Om tot de Nulde Normaalvorm te komen moeten we de informatiebehoefte gaan **inventariseren**. Bij wijze van voorbeeld vertrekken we van onderstaande leveringsbon. Wanneer een bedrijf goederen of producten aflevert, bij de klant, krijgt de klant hiervan een exemplaar.



Als we deze leveringsbons in een databank willen opslaan, beginnen we met alle gegevens te **inventariseren**. Dit betekent dat we alle gegevens op de afleverbon netjes onder elkaar gaan zetten. We krijgen dan het volgende:

**0 NV**

|  |
| --- |
| **ORDERS**  ordernr orderdatum klantnr klantnaam adres postcode plaats artnr artomschrijving aantal prijs regeltotaal eindtotaal |

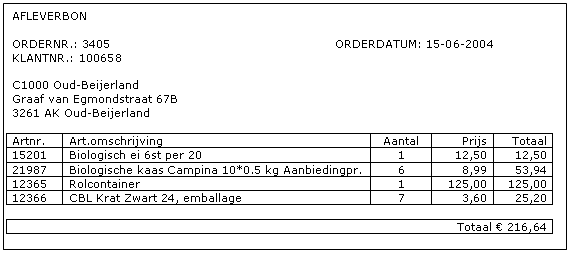
Zoals je ziet heb ik de inventarisatie ook een **naam** gegeven. Die staat in hoofdletters: **ORDERS**.

## Van de 0 NV naar de Eerste Normaalvorm 1 NV

Na het inventariseren is het de bedoeling dat we naar de Eerste Normaalvorm moeten gaan. Dit doen we altijd met de volgende stappen:

1. **Verwijder alle procesgegevens (berekende gegevens)**
2. **Geef de sleutel van de groep aan**
3. **Geef de deelverzameling aan die een herhaald aantal keren voorkomt t.o.v. de primaire sleutel**
4. **Herhaal de sleutelgegevens van de oorspronkelijke groep samen met de gegevens van de zich herhalende deelverzameling als een nieuwe groep**
5. **Verwijder de zich herhalende deelverzameling uit de oorspronkelijke groep**

Toegepast op ons voorbeeld geeft dat het volgende:



1. **Verwijder alle procesgegevens.**

In dit voorbeeld is het natuurlijk overduidelijk wat de procesgegevens zijn. Het **regeltotaal** wordt berekend uit **aantal \* prijs** en het **eindtotaal** wordt berekend als som van alle regeltotalen. Deze twee strepen we dus weg.

1. **Geef de sleutel van de groep aan**

In ons voorbeeld is de sleutel **ordernr**, immers een klant kan meerdere keren een order plaatsen, maar de **orders kunnen nooit hetzelfde nummer** **hebben**. De sleutel geven we aan door hem te onderstrepen.

1. **Geef de deelverzameling aan die een herhaald aantal keren voorkomt t.o.v. de primaire sleutel.**

Deze deelverzameling noemt men ook wel eens de **Repeterende Groep (RG)**. Dit zijn de gegevens die vaker voorkomen. In ons voorbeeld is dit het tabelletje met de bestelde goederen. Geef deze gegevens aan in je inventarisatie. We hebben nu dus het volgende:

**0NV**

|  |
| --- |
| **ORDERS**  ordernr orderdatum klantnr klantnaam adres postcode plaats **RG artnr RG artomschrijving RG aantal RG prijs** ~~X regeltotaal (procesgegeven~~) ~~X eindtotaal (procesgegeven)~~ |

1. **Herhaal de sleutelgegevens van de oorspronkelijke groep (ordernr) samen met de gegevens van de zich herhalende deelverzameling (RG) als een nieuwe groep. Geef deze nieuwe groep een goed gekozen naam.**

**0 NV**

|  |  |
| --- | --- |
| **ORDERS** ordernr orderdatum klantnr klantnaam adres postcode plaats **RG artnr RG artomschrijving RG aantal RG prijs** | **BESTELDE\_ARTIKELEN** ordernr RG artnr RG artomschijving RG aantal RG prijs |

Zoals je ziet heeft de tweede groep nog geen **sleutel**. De sleutel moet je zodanig kiezen dat er zo min mogelijk herhaalde groepen voorkomen t.o.v. deze sleutel. Het liefst natuurlijk geen herhaalde groepen meer, anders moet je de vorige stappen nog een keer herhalen.

Meestal kun je een combinatie nemen van de sleutel van de oorspronkelijke groep en het gegeven dat in de Repeterende Groep de sleutelrol vervult. De sleutel wordt in dit geval een **combinatie van ordernr en artnr**.

1. **Verwijder de zich herhalende deelverzameling uit de oorspronkelijke groep.**

Hier moeten we de RG dus weer opruimen en dan hebben we de Eerste Normaalvorm:

**1NV**

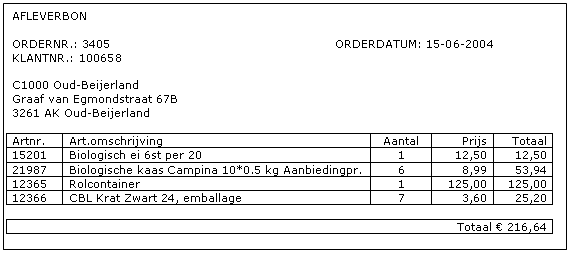
|  |  |
| --- | --- |
| **ORDERS**  ordernr orderdatum klantnr klantnaam adres postcode plaats | **BESTELDE\_ARTIKELEN** ordernr artnr artomschrijving aantal prijs |

## Van de 1 NV naar de Tweede Normaalvorm (2 NV)

Nu is het tijd om naar de Tweede Normaalvorm te gaan. Dit doen we ook weer met een aantal vaste stappen:

1. **Geef de attributen aan die niet functioneel afhankelijk zijn van de volledige sleutel.**
2. **Maak een aparte groep voor ieder deel van de sleutel waarvan de attributen functioneel afhankelijk zijn.**
3. **Neem in iedere groep de attributen met het bijbehorende sleuteldeel op en wijs de primaire sleutel aan.**
4. **Verwijder deze attributen uit de oorspronkelijke groep.**

Het voorbeeld:



1. **Geef de attributen aan die niet functioneel afhankelijk zijn van de volledige sleutel** als er een samengestelde sleutel is = een sleutel die uit meer dan 1 gegeven bestaat

We herhalen nog even de Eerste Normaalvorm:

**1NV**

|  |  |
| --- | --- |
| **ORDERS**  ordernr orderdatum klantnr klantnaam adres postcode plaats | **BESTELDE\_ARTIKELEN**  ordernr artnr artomschrijving aantal prijs |

De eerste groep **ORDERS** komt niet in aanmerking voor deze stap **omdat die niet beschikt over een samengestelde sleutel**.

Binnen de andere groep **BESTELDE\_ARTIKELEN** zijn er wel gegevens die functioneel afhankelijk zijn van een **deel van de sleutel**. Kijk maar eens naar **artomschrijving** en **prijs**. Blijkbaar zijn deze afhankelijk van **artnr** en niet van **ordernr**. Je kunt dit nagaan door te kijken wat er veranderd als het **artnr** gewijzigd wordt.

We geven dit zo aan in het voorbeeld:

**1NV**

|  |  |
| --- | --- |
| **ORDERS**  ordernr orderdatum klantnr klantnaam adres postcode plaats | **BESTELDE\_ARTIKELEN** ordernr > artnr > artomschrijving aantal > prijs |

Let erop dat **aantal** niet afhankelijk is van **artnr**. Het aantal per artikel kan namelijk verschillen per order.

**2. Maak een aparte groep voor ieder deel van de sleutel waarvan deze attributen functioneel afhankelijk zijn**

Het kan gebeuren dat een samengestelde sleutel in meerdere delen gesplitst kan worden en dat van ieder deel afzonderlijk attributen functioneel afhankelijk zijn. Er moeten dan meerdere groepen gevormd worden. In het voorbeeld ontstaat slechts één nieuwe groep: **ARTIKELEN**.

**3. Neem in iedere groep de attributen met het bijbehorende sleuteldeel op en wijs de primaire sleutel aan.**

**1NV**

|  |  |  |
| --- | --- | --- |
| **ORDERS**  ordernr orderdatum klantnr klantnaam adres postcode plaats | **BESTELDE\_ARTIKELEN**  ordernr > artnr > artomschrijving aantal > prijs | **ARTIKELEN**  artnr artomschrijving prijs |

**4. Verwijder deze attributen uit de oorspronkelijke groep.**

Na deze stap hebben we onze Tweede Normaalvorm en begint het al lekker op te schieten:

**2 NV**

|  |  |  |
| --- | --- | --- |
| **ORDERS**  ordernr orderdatum klantnr klantnaam adres postcode plaats | **BESTELDE\_ARTIKELEN** ordernr artnr aantal | **ARTIKELEN**  artnr artomschrijving prijs |

## Van de 2 NV naar de Derde Normaalvorm (3 NV)

De laatste Normaalvorm die ik behandel in deze tutorial is de Derde Normaalvorm.

Hiervoor hebben we de volgende stappen:

1. **Geef de niet-sleutel attributen aan die functioneel afhankelijk zijn van andere niet-sleutel attributen.**
2. **Maak een aparte groep voor ieder attribuut of combinatie van attributen, waar andere attributen functioneel van afhankelijk zijn.**
3. **Neem in iedere groep de attributen met bijbehorende sleutel op en wijs de primaire sleutel aan.**
4. **Verwijder de attributen van de nieuwe groep(en) uit de oorspronkelijke groep.**

Ik begin nog maar eens met de uitkomst van de Tweede Normaalvorm:

**2 NV**

|  |  |  |
| --- | --- | --- |
| **ORDERS**  ordernr orderdatum klantnr klantnaam adres postcode plaats | **BESTELDE\_ARTIKELEN** ordernr artnr aantal | **ARTIKELEN**  artnr artomschrijving prijs |

**1. Geef de niet-sleutel attributen aan die functioneel afhankelijk zijn van andere niet-sleutel attributen.**

Als we kijken in de tabel **ORDERS** zien we dat er een aantal gegevens zijn die niet afhankelijk zijn van **ordernr**. Deze zijn **klantnaam**, **adres**, **postcode** en **plaats**. Deze zijn afhankelijk van het **klantnr**.

**Klantnr** is in deze tabel een niet-sleutelattribuut, dus deze stap is makkelijk uit te voeren: (de afhankelijkheid is aangegeven met een A)  
  
**2NV**

|  |  |  |
| --- | --- | --- |
| **ORDERS** ordernr orderdatum A klantnr A klantnaam A adres A postcode A plaats | **BESTELDE\_ARTIKELEN** ordernr artnr aantal | **ARTIKELEN** artnr artomschrijving prijs |

**2. Maak een aparte groep voor ieder attribuut of combinatie van attributen, waar andere attributen functioneel van afhankelijk zijn.**

Vrij vertaald moeten we voor de groep die we net aangegeven hebben een nieuwe groep (**KLANTEN**) maken:  
 **2NV**

|  |  |  |  |
| --- | --- | --- | --- |
| **ORDERS** ordernr orderdatum A klantnr A klantnaam A adres A postcode A plaats | **KLANTEN** klantnaam adres postcode plaats | **BESTELDE\_ARTIKELEN** ordernr artnr aantal | **ARTIKELEN** artnr artomschrijving prijs |

**3. Neem in iedere groep de attributen met bijbehorende sleutel op en wijs de primaire sleutel aan.**

We moeten dus de sleutel waarvan de nieuwe groep afhankelijk is in de nieuwe groep plaatsen, daarna moeten we de sleutel aangeven. De nieuwe groep ziet er dan zo uit:

**KLANTEN**  
klantnr  
klantnaam  
adres  
postcode  
plaats  
  
Let er wel op dat **klantnr** niet zomaar uit **ORDERS** gehaald kan worden, dan zouden de orders namelijk nooit aan een klant gekoppeld kunnen worden.

**4. Verwijder de attributen van de nieuwe groep(en) uit de oorspronkelijke groep.**

Als we deze stap hebben gedaan zijn we **klaar met het normaliseren**. Dit ziet er dan zo uit:  
  
**3NV**

|  |  |  |  |
| --- | --- | --- | --- |
| **ORDERS** ordernr klantnr orderdatum | **KLANTEN** klantnr klantnaam adres postcode plaats | **BESTELDE\_ARTIKELEN** ordernr artnr aantal | **ARTIKELEN** artnr artomschrijving prijs |

## Begrippen

* **Redundantie**: Het dubbel opslaan van gegevens.
* **Gegevens**: Feiten of gebeurtenissen die op een bepaalde manier vastgelegd zijn.
* **Informatie**: De betekenis die aan gegevens ontleend wordt.
* **Procesgegevens**: gegevens die berekend worden uit andere gegevens
* **Record**: Rij in een tabel
* **Attribuut**: Kolom of veld in een tabel
* **Functionele afhankelijkheid**: We zeggen dat een gegeven B functioneel afhankelijk is van een ander gegeven A als er bij een waarde A één (en niet meer dan één) waarde van B kan voorkomen (naam van een artikel is functioneel afhankelijk van het artnr).