Normaliseren voor Dummies

Waarom normaliseren?

"Gegevensredundantie leidt tot gegevensinconsistentie!" Dit cryptisch antwoord betekent het volgende: indien men dezelfde gegevens onnodig herhaaldelijk opslaat (= gegevensredundantie = gegevensovertolligheid), zullen de gegevens vroeg of laat niet meer in overeenstemming zijn met elkaar (=gegevensinconsistentie). Voorbeeld: indien men adresgegevens op verschillende plaatsen bijhoudt en er is een adreswijziging, dan is het niet ondenkbeeldig dat het adres op één van de plaatsen niet wordt bijgewerkt. Op dat ogenblik hebben we twee verschillende adressen voor dezelfde entiteit (persoon, bedrijf, …): één correct, één foutief.

Wat is normaliseren?

Normaliseren is een formele techniek, die in een aantal stappen (= Normaalvormen) de gegevensredundantie wegwerkt door de gegevens volgens vaste regels af te splitsen.

Begrippen.

Voor dat we verder gaan is het aangewezen een duidelijke definitie te hebben van een aantal begrippen.

Gegevens, informatie en kennis.

Niettegenstaande deze drie begrippen door elkaar gebruikt worden, zijn ze strikt genomen totaal verschillend.

Gegevens (*data*): zijn naakte feiten: 150 euro, 9000 Gent, zwart, metaal, 15x20x30, …

Informatie (*information*): zijn bruikbare gegevens (in bedrijfseconomische zin voor het nemen van beslissingen of om te kunnen overgaan tot actie):

- het goedkoopste product vind ik bij Y (product A kost 250 euro bij X en 240 bij Y),
- ik moet tanken (ik heb nog 9 liter brandstof, ik ben op 50 km van een tankstation en ik verbruik 11 liter per 100 km), ...

Kennis (*knowledge*): weten welke acties bij welke informatie passen:

- bij gelijke kwaliteit, koop ik het goedkoopste product,
- rekening houdend met een zekere marge, heb ik nog net voldoende brandstof om tot het volgende tankstation te geraken en ga ik daar dus tanken, …

Wij gaan ons hier beperken tot de gegevens.

Gestructureerd en ongestructureerd.

Gestructureerde gegevens halen hun betekenis uit hun plaats binnen een structuur. Voorbeeld: in een journaalpost weten we dat een bedrag gedebiteerd wordt omdat dit bedrag in de debetkolom staat.

Ongestructureerde gegevens halen hun betekenis uit de context, uit de andere gegevens die er bij staan. Voorbeeld: in de zin “Uw postcode is 9000”, weten we dat 9000 een postcode is uit de rest van de zin.

Wij gaan ons hier beperken tot gestructureerde gegevens.

Database, tabel en rij.

Een tabel (of gegevensstructuur) is een gestructureerde verzameling van gegevens en is ingedeeld in rijen en kolommen. Elke kolom bevat gelijksoortige gegevens en heeft een naam (kolomnaam of veldnaam). In een adressentabel kan men een kolom “gemeente” hebben. Deze kolom zal dan uitsluitend namen van gemeenten bevatten.

Een rij van een tabel bevat alle gegevens van een welbepaalde entiteit. In een adressentabel bevat een rij alle gegevens van een bepaalde persoon (naam, straat, postcode, gemeente, telefoon, …)

Een database tenslotte is een verzameling van tabellen die onderling verbonden zijn d.m.v. relaties. Wij kennen drie soorten relaties:
Één-op-één relatie: één rij van de ene tabel komt precies overeen met één rij van de andere tabel. (komt relatief weinig voor)
Één-op-veel relatie: één rij van de ene tabel heeft een relatie met meerdere rijen van de andere tabel, voorbeeld: één klant heeft meerdere facturen, één factuur heeft meerdere factuurregels, … (meest voorkomende relatie)
Veel-op-veel relatie: meerdere rijen van de ene tabel hebben een relatie met meerdere rijen van de andere tabel, voorbeeld: één auteur heeft meerder boeken geschreven en één boek is geschreven door meerdere auteurs. (deze relatie kan als dusdanig niet geïmplementeerd worden. In de praktijk is een veel-op-veel relatie een combinatie van twee één-op-veel relaties met één gemeenschappelijke tabel – telkens de “veel” kant van de relatie)

Wij beperken ons verder tot de Relationele database. Het relationeel model is momenteel het meest voorkomende model. Alle hier gebruikte terminologie is afgestemd op dit model.

Sleutels

Wij kennen 2 soorten sleutels:

Primaire sleutel (Primary Key, PK): is een gegeven (enkelvoudig of samengesteld, dus één of meerdere kolommen) die op unieke wijze de overige gegevens van een tabel identificeren. Voorbeeld: een klantennummer laat ons toe de overige gegevens van een bepaalde klant op een ondubbelzinnige wijze in een tabel terug te vinden. Een PK moet dus per definitie uniek zijn (er mogen geen dubbels zijn). Om bruikbaar te zijn dient een PK daarenboven niet te lang te zijn. In de mate van het mogelijke zal men kiezen voor een natuurlijke sleutel (PK nemen uit de beschikbare gegevens). Is er geen natuurlijke sleutel, dan moet men een kunstmatige sleutel invoeren (gegeven dat men toevoegt aan de bestaande gegevens) [de PK gaan we aanduiden door ze vetjes op te maken]

Verwijssleutel (Foreign Key, FK): is een gegeven in een tabel die in een andere tabel een primaire sleutel is. Voorbeeld: in een tabel facturen zal het veld klantennummer een verwijssleutel zijn (die verwijst naar het klantennummer in de tabel klanten).[de FK gaan we aanduiden door ze te onderstrepen]

PK's en FK's spelen een heel belangrijke rol in databases. Het is dankzij die sleutels dat we relaties kunnen leggen tussen de tabellen.

Functioneel afhankelijk

Gegevens zijn functioneel afhankelijk als je kan stellen dat er een relatie is tussen de gegevens. Het eenvoudigst is één van de gegevens (meestal de PK) te laten samenvallen met een object of entiteit en zich af te vragen of het ander gegeven een kenmerk of eigenschap is van dat object/entiteit. Is dit het geval, dan kan men zo goed als zeker spreken van een functionele afhankelijkheid tussen deze gegevens.

Normaliseren

Aan de hand van volgend voorbeeld gaan we de verschillende normaalvormen bespreken.

Een vereniging kent een aantal commissies, zoals de Technische commissie, de Barcommissie en de Excursiecommissie. Stel de nodige tabellen op om te voldoen aan de volgende informatiebehoefte:

Commissie: **BC, Barcommissie**		
Lidnummer	**Naam**	**Telefoon**
034	L. Ketelaar	(09)125 00 91
122	V. Bottelaar	(03)236 15 28
307	M. Vervaet	(059)17 28 99

0e Normaalvorm (0NF)
We stellen eerst een ongenormaliseerde basistabel op door alle gegevens tussen haakjes te plaatsen. Deze tabel krijgt een naam en de gegevensgroepen die meer dan eenmaal kunnen voorkomen plaatsen we nogmaals tussen haakjes om aan te duiden dat dit herhalingsstructuren zijn. Vervolgens bepalen we een/de uniek(e) gegeven(s) die de PK zullen vormen (in voorkomend geval voeren we een nieuw gegeven in als PK)

0NF	Stel de ongenormaliseerde tabel op en bepaal de primaire sleutel.

Uit de opgave kunnen we halen dat een commissie een code en een naam heeft. Een commissie bestaat verder uit een aantal leden met elk hun nummer, naam en telefoon. Het aantal leden van een commissie ligt niet vast en kan dus gaan van geen, één tot meerdere leden. Een commissie kan dus meerdere leden hebben. We plaatsen de leden dan ook in een herhalingsstructuur (tussen haakjes). Vervolgens bepalen we welk gegeven uniek en bruikbaar is als PK. Zowel de code als de naam van de commissie kunnen als PK gebruikt worden (dit zijn de kandidaatsleutels).

Namen zijn meestal geen goede PK's: te lang en grote kans van dubbels. We nemen dus de code van de commissie als PK. Dit alles geeft ons de volgende structuur:

COMMISSIE (**Code**, ComNaam, (LidNr, LidNaam, LidTel))

Uitgewerkt in tabelvorm (voor alle duidelijkheid uitgebreid met de Technische en de Excursie commissie):

Tabel **COMMISSIE**

Code	ComNaam	LidNr	Lidnaam	LidTel	LidNr	Lidnaam	LidTel	LidNr	Lidnaam	LidTel
BC	BarCommissie	034	Ketelaar	091250091	122	Bottelaar	032361528	307	Vervaet	059172899
TC	Techniische C.	122	Bottelaar	032361528						
EC	Excursie C.	034	Ketelaar	091250091	307	Vervaet	059172899			

1e Normaalvorm (1NF)

Het probleem met herhalingsstructuren is o.a. dat bij implementatie (het uiteindelijk maken van de tabellen) men niet op voorhand weet hoeveel ruimte men moet voorzien. In ons voorbeeld: hoeveel leden moeten we maximaal voorzien voor een commissie?
Om dit op te lossen, gaan we de herhalingsstructuur afsplitsen in een nieuwe tabel. In deze nieuwe tabel nemen we de oorspronkelijke PK over als FK, dit om een relatie te behouden met de oorspronkelijke tabel (zodat we weten tot welke commissie de leden behoren).

1NF	Splits de herhalingsstructuren af.

De gegevens van de commissieleden plaatsen we in een nieuwe tabel samen met de oorspronkelijke PK Code die hier een FK wordt (onderstreept). In de nieuwe tabel moeten we vervolgens een PK bepalen. Als we er vanuit gaan dat een lid in verschillende commissie kan zitten, dan kunnen we niet anders dan een combinatie nemen van Code en LidNummer. Dit alles levert ons de volgende situatie:

COMMISSIE (**Code**, ComNaam)

COMMISSIELEDEN (**Code, LidNr**, LidNaam, LidTel)

Uitgewerkt in tabelvorm:

Tabel **COMMISSIE**

Code	ComNaam
BC	BarCommissie
TC	Techniische C.
EC	Excursie C.

Tabel **COMMISSIELEDEN**

Code	LidNr	Lidnaam	LidTel
BC	034	Ketelaar	091250091
BC	122	Bottelaar	032361528
BC	307	Vervaet	059172899
TC	122	Bottelaar	032361528

EC	034	Ketelaar	091250091
EC	307	Vervaet	059172899

2e Normaalvorm (2NF)
Gegevens in een tabel die niet volledig functioneel afhankelijk zijn van de volledige PK leiden tot redundantie (gegevensovertolligheid), hetgeen moeilijker is om te onderhouden en dus vroeg of laat zal leiden tot inconsistentie (niet-overeenstemming van de gegevens). Bepaalde leden komen in de tabel CommissieLeden meer dan éénmaal voor. Als gegevens van een lid wijzigen (vb. telefoonnr), dan moet dit gegeven in verschillende rijen aangepast worden. Wordt dit ook maar in één rij vergeten, dan zitten we met verschillende telefoonnummers voor hetzelfde lid (= gegevensinconsistentie).

2NF	Splits alle niet van de volledige sleutel afhankelijke gegevens af:

Plaats de kolommen die niet volledig van de sleutel afhangen in een afzonderlijke tabel. De sleutel van deze nieuwe tabel blijft in de oorspronkelijke tabel over als FK.

COMMISSIE (**Code**, ComNaam)

COMMISSIELEDEN (**Code, LidNr**)

LEDEN (**LidNr**, LidNaam, LidTel)

Uitgewerkt in tabelvorm:

Tabel **COMMISSIE**

Code	**ComNaam**
BC	BarCommissie
TC	Techniische C.
EC	Excursie C.

Tabel **COMMISSIELEDEN**

Code	**LidNr**
BC	034
BC	122
BC	307
TC	122
EC	034
EC	307

Tabel **LEDEN**

LidNr	**Lidnaam**	**LidTel**
034	Ketelaar	091250091
122	Bottelaar	032361528
307	Vervaet	059172899
122	Bottelaar	032361528
034	Ketelaar	091250091
307	Vervaet	059172899

3e Normaalvorm (3NF)
Gegevens in een tabel die functioneel afhankelijk zijn én van de PK én van een ander niet-sleutel gegeven zijn transitief afhankelijk en leiden eveneens tot gegevensovertolligheid. De gevolgen zijn analoog met de problematiek van de 2NF en worden dan ook op dezelfde wijze behandeld.

3NF Splits alle transitieve afhankelijkheden af:

De transitief afhankelijke kolommen worden in een nieuwe tabel geplaatst samen met hun PK, PK die in de oorspronkelijke tabel als FK overblijft.

In ons voorbeeld hebben we geen transitieve afhankelijkheid en we zeggen dan dat 3NF = 2NF

Denormalisatie
We hebben gezien dat bij elke normalisatiestap we telkens gegevens in nieuwe tabellen afsplitsen, m.a.w. het aantal tabellen neemt bij elke normaalvorm toe. Een probleem dat kan ontstaan is dat het beheer van de tabellen complexer wordt. Als regel kunnen we stellen dat de complexiteit van het beheer van tabellen exponentieel stijgt met hun aantal. Voor dat we de tabellen dan ook effectief gaan implementeren, kan het de moeite lonen om na te gaan of er geen tabellen kunnen worden samengevoegd (omdat hun structuur quasi gelijk is) of in de limiet zelfs gewoon geschrapt worden (omdat de gegevens irrelevant of onrechtstreeks in andere tabellen ook aanwezig zijn).
Als we ervoor zorgen dat:
- er geen informatie verloren gaat en
- er geen redundantie opnieuw wordt ingevoerd

Dan en enkel dan, is het verantwoord om het aantal tabellen te verminderen en het beheer dus te vereenvoudigen.