De hoeksteen van het relationele model

# Regel 1: De informatie-regel

Frido van Orden

**De informatie-regel vereist simpelweg dat alle informatie in de database wordt gerepresenteerd op één, en slechts één, manier, namelijk door middel van waarden op kolomposities binnen rijen van tabellen.**

De vorige keer is uitgebreid stilgestaan bij 'Regel nul', die vreemde eend in de bijt waarachter twee fundamentele gedachten van het relationele model bleken schuil te gaan: de scheiding van model en implementatie en de essentialiteit (of zo men wil: minimaliteit) van concepten en constructies.
Op die laatste gedachte bouwt de eerste 'echte' van de 12 regels eigenlijk voort. De informatie-regel, zoals hij ook wel wordt genoemd, is eigenlijk in de bespreking van regel 0 al om de hoek gekomen.
Daarbij is een eenvoudig Employee-Department gegevensmodel behandeld in zowel een relationele als (fictieve) netwerk-variant. Er bleek dat het concept van een 'CODASYL set' – kernconcept in het netwerkmodel – helemaal niet essentieel is. Men kan het gebruik van pointer-kettingen in een CODASYL database (zoals CA IDMS of Digital DBMS) altijd vermijden door het opnemen van een extra attribuut in de member dat verwijst naar de owner: het concept dat in de relationele wereld bekend staat als een verwijzende sleutel.
Het netwerkmodel is dus een relationeel model met een extra keuzemogelijkheid. Of anders gezegd is het relationele model het netwerkmodel minus een overbodige constructie. De kracht van het relationele model is dat het niet meer maar juist minder kan dan concurrerende modellen.

## Atomaire gegevens

De informatie-regel komt op het eerste gezicht eenvoudig, eenduidig en logisch over, een karakteristiek die overigens ook voor de meeste andere regels geldt. Het overzien van de gevolgen is echter heel wat minder eenvoudig, het interpreteren en op juiste waarde schatten en het consequent toepassen vormen nog weer een ander verhaal.
Zo bestaat er bijvoorbeeld een directe relatie tussen de informatie-regel en de eerste normaalvorm uit de normalisatietheorie (zoals u inmiddels weet ook van Codd's hand, zij het van iets later datum). Die eerste normaalvorm stelt dat de database uitsluitend atomaire gegevens mag bevatten. Atomair wil in dit verband zeggen dat de gegevens, wat het database management-systeem betreft (beter: wat het database-model betreft), atomair van karakter zijn en dus geen interne structuur kennen. Tragischerwijs echter is dit ooit geïnterpreteerd als 'alleen eenvoudige datatypen als tekst en numerieke waarden': een vergissing met verstrekkende gevolgen.

De gedachtegang vindt zijn oorsprong begin jaren negentig met de opkomst van object-oriëntatie. De functionele behoefte aan object-oriëntatie komt onder andere voort uit de hoek van geografische en multimediale informatieverwerking. De eerste toepassingen die gebruik maakten van object-oriëntatie richtten zich dan ook op die toepassingsgebieden. Toen de behoefte ontstond aan het vastleggen van dergelijke complexe soorten gegevens, werd natuurlijk in eerste instantie gekeken naar de mogelijkheden van de toenmalige producten, die inmiddels als relationeel waren te betitelen (relationeel volgens de definities van de 12 regels waren zij overigens geenszins). Die mogelijkheden beperkten zich in vrijwel alle gevallen tot tekst, numerieke waarden en datum- en tijdinformatie, waarop de conclusie werd getrokken dat relationele producten niet aan de eisen van 'moderne object-georiënteerde applicaties' voldeden.
Zoals zo vaak werd vervolgens de scheiding tussen implementatie en model moeiteloos overschreden en werd met verve verkondigd dat het relationele model niet krachtig genoeg was voor alles wat verder ging dan simpele 'text and numbers'. Vervolgens volgde natuurlijk krachtige promotie voor object-georiënteerde databases. Die werden geen doorslaand succes, maar merkwaardig genoeg voelde iedere zichzelf serieus nemende relationele database-

## Relational Rules (3)

De vorig jaar overleden dr. E.F. Codd werd wereldberoemd met zijn serie publicaties over een gegevens(meta)model dat later bekend zou worden onder de naam 'Relationeel Model'. De eerste uit die serie publicaties was een intern IBM Research Report dat uitkwam in 1969. 2004 zal dus gelden als een lustrum – 35 jaar Relationeel Model.
Frido van Orden schrijft in Database Magazine een serie artikelen over de betekenis en de erfenis van het gedachtengoed van Codd en komt nu toe aan de eerste 'echte'regel; de informatie-regel.

leverancier zich vervolgens geroepen om zijn relationele product op te tuigen met allerlei zogenaamd object-georiënteerde features. Voor wat betreft het uitbreiden van gegevenstypen voorbij tekst en numerieke waarden, is dat op zich nog als vooruitgang te beschouwen, maar voor het overige is het niet minder dan een ramp (waarover straks meer).
Terug naar het begrip 'atomair'. Wat is er atomair aan een plaatje, een geluidsfragment, een driedimensionaal vectordiagram of een videofilm? Passen we de definitie van atomair toe, dan moet het plaatje, het geluidsfragment enzovoort, intern geen structuur bevatten wat de database/het model betreft, met andere woorden: niets meer zijn dan een BLOB (Binary Large Object), een brok binaire gegevens *as is*. Maar natuurlijk wil men ook kunnen zoeken op plaatjes van bijvoorbeeld Saddam Hoessein, geluidsfragmenten in WAV formaat, vectordiagrammen van de Marslander of videofilms van minder dan 60 minuten speelduur. Is dat dan nog atomair? Het antwoord lijkt nee, zeker als men zich realiseert dat om efficiënt te kunnen zoeken, men waarschijnlijk ook specifieke indexeringstechnieken zal willen gebruiken. En zijn tekst en numerieke waarden eigenlijk wel atomair? Als we kunnen zoeken met 'medewerker.naam LIKE 'Jans%e%' om iedereen met een naam die enigszins lijkt op Jansen te kunnen lokaliseren, kijkt men toch ook naar de interne structuur? En een index op een numerieke kolom geeft bij het doorlopen toch zelfs andere resultaten dan diezelfde index op een tekstkolom met identieke waarden? Zo beschouwd is een tekstveld net zo non-atomair of complex als een veld waarin een plaatje kan worden opgeslagen.

> Het netwerkmodel is een relationeel model met een extra keuzemogelijkheid

## Datatypen

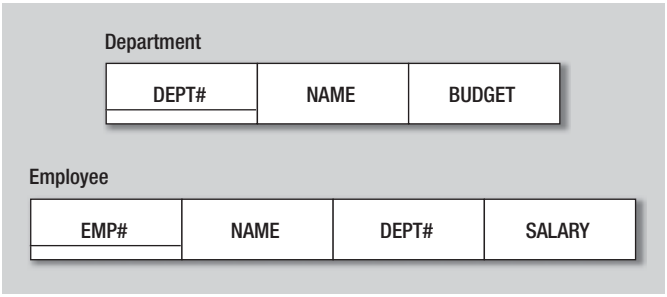
Voor de verstokte relationelen wellicht schokkend: dit raakt hier een lacune in het relationele model die in de object-georiënteerde wereld veel beter wordt gefaciliteerd. Het gaat hier over het concept 'datatype', in de relationele wereld veelal 'domein' genoemd. Het domeinconcept in het relationele model is erg rudimentair uitgewerkt. Strikt genomen beschrijft het relationele model niet eens welke domeinen/gegevenstypen er bestaan. Over het algemeen wordt er van een beperkte set domeinen uitgegaan, gebaseerd op tekst en numerieke gegevens, en met een standaard verzameling operatoren (gelijk aan, groter dan, plus etcetera).
De object-georiënteerde visie op datatypen voegt enkele concepten toe, die bepaald niet redundant zijn:

- Interne structuur (bijvoorbeeld een datatype 'punt' met een x- en een y-coördinaat). Let op: deze structuur is vanuit het database-model gezien nog steeds atomair! De database 'weet' niet dat een 'punt' waarde intern structuur heeft (evenmin als de database weet dat een tekstwaarde intern de structuur heeft van een reeks karakters);
- Operatoren/methodes (bijvoorbeeld een '=' operator op datatype 'punt' die true teruggeeft indien x- en y-coördinaat van de beide operanden identiek zijn en anders false). Een operator heeft altijd een resultaattype (in het geval van de '=' operator is dat boolean, het enige fundamentele datatype dat de relationele theorie vereist).

De eerder genoemde 'like' operator is bijvoorbeeld te beschouwen als een bijzondere operator of methode van het gegevenstype 'tekst'.

Een goede definitie van datatypen geeft enorm krachtige mogelijkheden. In een voorraadadministratie-database bijvoorbeeeld kan een datatype 'aantal' worden gedefinieerd (intern bestaand uit één numerieke waarde >= 0) en een datatype 'prijs' (intern bestaand uit 1 numerieke waarde). We kunnen dan de operator '*' definiëren met als operanden aantal en prijs en resultaattype prijs. Query's als 'SELECT * FROM artikel WHERE voorraad = prijs' kunnen worden verboden, aangezien er geen operator '=' zal zijn gedefinieerd met parameters voorraad en prijs.
'SELECT voorraad*prijs FROM artikel' kan daarentegen weer wel. Het type van de enige kolom in de resultaatrelatie is dan 'prijs', en niet NUMBER(10,2) of iets dergelijks.

Gebruikt men de object-georiënteerde visie op datatypen voor invulling van het relationele concept 'domein' (nogmaals: in het relationele model sec niet uitgewerkt!), dan blijken de puzzelstukken prachtig in elkaar te passen. Gaan we terug naar de oorspronkelijke vraagtekens die we bij atomiciteit plaatsten ('geluidsfragmenten in WAV formaat', 'LIKE 'Jans%e%') dan is te zien dat alles wat interne kennis over de gegevens vereist kan worden 'weggestopt' in operatoren: men krijgt dan relationele condities als 'sound.getFormat() = 'WAV'' en 'naam.like('Jans%e%')'. De database hoeft geen kennis te hebben van de interne structuur van een geluidsfragment of een tekst maar alleen van de operatoren die via het datatype ter beschikking worden gesteld. Dit ligt dan heel dicht in de buurt van wat in de object-georiënteerde wereld bekend staat als 'encapsulatie': de interne structuur van een datatype is afgeschermd voor de buitenwereld en alleen bekend binnen de operatoren die op het datatype zijn gedefinieerd.




**Afbeelding 1:** Relationeel model.

Tot slot van de discussie over atomiciteit nogmaals een verwijzing naar 'The Third Manifesto' van Chris Date en Hugh Darwen. Onmisbare kost voor wie dit onderwerp interessant vindt.

## Wij willen pointers!

Naast de misleide discussie over atomiciteit is er nog een discussie waarin geknaagd wordt aan de informatie-regel. Niet verrassend wordt ook deze discussie met name vanuit de object-georiënteerde hoek gevoerd. Twee citaten:

- 'Foreign keys zijn de pointers van het relationele model';
- 'Om in het relationele model twee entiteiten aan elkaar te koppelen moet je altijd een join gebruiken. Dat is nodeloos complex, want je gebruikt immers toch elke keer dezelfde join'.

Samenvattend: de informatie-regel is misschien theoretisch leuk maar in de praktijk is het ontbreken van pointers toch vooral erg vervelend. Was Codd dan slechts een studeerkamergeleerde? Geenszins! Natuurlijk was er een theoretisch argument voor de informatie-regel (zie de discussie over essentialiteit), maar anderzijds handelde Codd ook duidelijk vanuit een gebruikersperspectief. Zoals iedere programmeercursist kan vertellen is het werken met pointers zeer krachtig, maar lang niet voor iedereen intuïtief.

Codd wilde dat het relationele model gegevenstoegang dichter bij eindgebruikers zou brengen en uit het gesloten wereldje van automatiseringskenners zou trekken. In plaats van pointers gaf hij daarom de voorkeur aan operatoren die iedereen intuïtief kan toepassen: het vergelijken van waarden. Voor wie verwend is met sexy 'drag & drop' query tools is het misschien moeilijk voor te stellen, maar Codd meende 30 jaar geleden oprecht met de relationele joins à la 'EMP.DEPT# = DEPT.DEPT#' de eindgebruiker een dienst te bewijzen! Zo bezien is het niet merkwaardig dat nu net de 'echte mannen' (sorry dames) onder de programmeurs, ofwel de object-georiënteerden, zich beklagen over het ontbreken van de pointer.
Maar er is meer aan de hand dan alleen een verschil in focus of een cultuur-clash. In de hele discussie tot nu toe is het concept 'foreign key', of algemener, 'integriteitsregel', nog niet aan de orde geweest. In Regel 10 wordt er pas voor het eerst iets over gezegd. Wel hebben we het uitgebreid gehad over waarden en operatoren, en in het bijzonder de vergelijkingsoperatoren.

Teruggaand naar het standaard Afdeling/Medewerker-voorbeeld is zo een aantal query's te verzinnen waarin de join 'EMP.DEPT# = DEPT.DEPT#' voorkomt. Maar waarom kan men deze join eigenlijk leggen, en waarom is de join 'EMP.DEPT# = DEPT.BUDGET' bijvoorbeeld onzinnig, hoewel beide attributen mogelijk identiek zijn geïmplementeerd als NUMBER(10)? Het antwoord is natuurlijk dat EMP.DEPT# en DEPT.DEPT# gebaseerd zijn op hetzelfde domein, ofwel dat beide attributen hetzelfde datatype hebben.
Een vergelijking 'EMP.SALARY < DEPT.BUDGET' is ook mogelijk als beide attributen hetzelfde datatype toegekend wordt, hetgeen goed voorstelbaar maar beslist niet noodzakelijk is.
Daarnaast kent het gegevensmodel de regel dat medewerkers alleen op een bestaande afdeling mogen werken (de foreign key EMP.DEPT# => DEPT.DEPT#) en natuurlijk is het zo dat deze foreign key alleen gelegd kan worden bij gratie van het feit dat beide attributen hetzelfde datatype hebben. Maar het al dan niet aanwezig zijn van de foreign key heeft geen enkel effect op de query's die men op het model kan formuleren (wel natuurlijk op de mogelijke uitkomsten van die query's, immers, met de foreign key beperkingsregel kan er wat betreft tabelinhoud minder dan zonder).
Breidt men het gegevensmodel uit door in Department een attribuut DEPT#_SUP op te nemen, waarin vastgelegd is onder welke hogere afdeling een afdeling valt, dan zijn joins in de vorm EMP.DEPT# = DEPT.DEPT#_SUP mogelijk, terwijl deze join ook in het geval van netjes gedefinieerde foreign keys niet 1:1 op een foreign key betrekking heeft.

> De kracht van het relationele model is dat het juist minder kan dan concurrerende modellen

Zijn foreign keys dus nodeloos complexe relationele pointers? Nee! Natuurlijk is het zo dat joins vaak over een foreign key gelegd worden en zeker indien die foreign key is samengesteld uit meerdere attributen zou een verwijzing naar de complete join over alle sleutelattributen een handige en fouten voorkomende shortcut zijn. Maar het is vast geen toeval dat juist in de object-georiënteerde wereld weinig gebruik gemaakt wordt van de kracht van relationele joins, en al helemaal niet van alle join-varianten die niet 1:1 met een pointer (pardon, foreign key) corresponderen. Blijft staan het laatste argument voor pointers in plaats van foreign keys: relationele joins performen niet. Even nadenken: wat hadden we eerder ook weer gezegd over 'theoretisch' en 'implementatievrij'? En vraag gerust uw DBA om ontzenuwing van dit argument.

## Conclusie

De informatie-regel is een van de hoekstenen van het relationele model. Vele aanvallen op het relationele model, waarvan er enkele hier genoemd zijn, hebben in de kern uiteindelijk betrekking op de informatie-regel, en veelal op een slecht begrip ervan.
De volgende keer gaat het verder over primaire sleutels en komt nog zo'n heilig huisje uit de object-oriëntatie aan bod: de oid.

**Frido van Orden** (frido.van.orden@faapartners.com) is partner bij FAA Partners.