Generalisatie als specialiteit

Een oplossing voor problemen die er niet zijn?

Frido van Orden

In het vorige nummer hebben we een aanvang gemaakt met het verkennen van objectgeoriënteerde features in de wereld van het (relationele) dbms. Een actueel onderwerp, getuige het artikel 'Problemen die er niet zijn' van Patrick Savalle in hetzelfde nummer. Hij kiest een modelmatig uitgangspunt en komt tot de conclusie dat het relationele model krachtig genoeg is voor gegevensmodellering en dat er geen behoefte bestaat aan object-dbms'en. Een conclusie die overeenkomsten vertoont met de inleidende paragrafen van het vorige artikel in deze serie. Bij nadere beschouwing blijken de verschillen echter groter dan de overeenkomsten.

Het vorige artikel begonnen we met het benoemen van subtypering als een van de weinige features van objectoriëntatie die in de, nog steeds door relationelen gedomineerde, wereld van gegevensmodellering school hebben gemaakt. Het is interessant om te zien hoe het overzichtje van OO-features en relationele varianten daarop (of, zo u wilt, workarounds) aansluit bij het artikel van Savalle.

De eerste feature, het ondersteunen van complexe gegevenstypen als beeld en geluid, is strict genomen geen OO-feature maar meer een toepassing van andere OO-features, met name 'eenheid van gegevens en logica'. Het opslaan van multimediagegevens is immers niet veel meer dan het opslaan van een reeks bytes, gecombineerd met geavanceerde logica, met name zoekmechanismen, om in deze ongestructureerde informatie toch de gewenste gegevens te kunnen vinden. Met gegevensmodellering heeft het verder niet zo heel veel te maken, en het is dan ook niet verbazingwekkend dat Savalle er geen aandacht aan besteed.

OO feature	(Relationele) praktijk-oplossing	Savalle
Ondersteuning voor beeld, geluid etc	gebruik van BLOB velden	-
Object identiteit	Primary key, autonumber velden	-
Eenheid van gegevens en logica (attributen en methoden)	Architectuur scheidt gegevens en logica. Dbms alleen voor data opslag	Abstraheren van 'actieve aspecten'
Encapsulatie van gegevens	Wordt niet als interessant ervaren	Abstraheren van 'actieve aspecten'
Ondersteuning van typehiërarchieën	Hard uitmodelleren	'Notatietruc'
Goede integratie met OO-applicaties	Applicatieservers en mapping tools	'Een RDBMS sluit beter aan op OO-software dan een ODBMS'

Dat generalisatie een non-issue in gegevensmodellering zou zijn is een conclusie die veel te ver gaat

Evenmin geldt dat voor het gebrek aan aandacht voor het tweede OO-feature, namelijk dat van de object-identiteit. Voor vrijwel iedereen is het concept van primaire sleutels, in de praktijk veelal onterecht synoniem verklaard met een enkel numeriek geïndexeerd veld, meer dan afdoende. Het door OO-puristen gepropageerde concept van identiteit, onafhankelijk van attribuutwaarden, en globaal (of moeten we zeggen: universaal) uniek, wijkt hier weliswaar op diverse theoretische aspecten van af, maar blijkt in de dagelijkse praktijk geen wezenlijke voordelen te bieden. Fascinerend is het overigens om te constateren dat het concept van meerdere unieke sleutels in OO-modellering geen plaats heeft.

Ook bij de volgende feature, de eenheid van logica en gegevens, sluit het betoog van Savalle aan op de eigen bevindingen. De toegevoegde waarde van eenheid van logica en gegevens is groter

Het dbms voorbij (9)

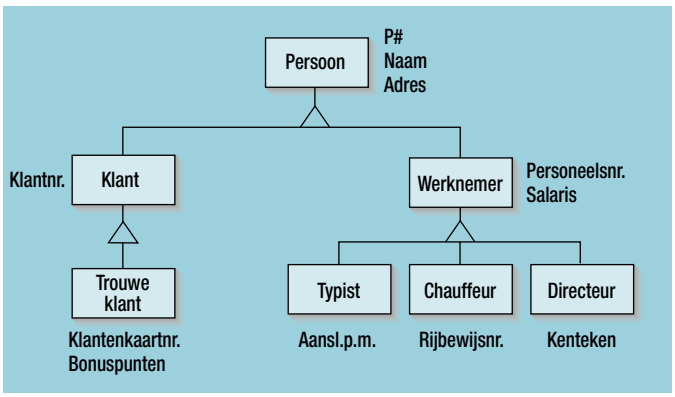
In het vorige nummer ging de aflevering over OO-features (in het bijzonder de modelondersteuning voor generalisatiehiërarchieën) in de wereld van relationele dbms'en. In hetzelfde nummer verscheen een artikel van Patrick Savalle waarin OO-databases overbodig worden verklaard. Frido van Orden pakt de toegeworpen handschoen op en reageert in deze aflevering op de stellingen van Savalle.

naarmate de logica meer met de gegevens verweven is. De heersende teneur is echter dat zelfs de meest elementaire gegevensgerelateerde logica als het afdwingen van verplichtingregels (in relationele termen: NOT NULL constraints) regelmatig wordt bestempeld als zijnde 'business logica' en daarmee veroordeeld wordt tot een implementatie ver weg van de gegevenslaag. Wie 'Het dbms voorbij' al langer volgt weet dat de auteurs dezes daar andere inzichten over hebben. In een van de komende artikelen komen we dan ook nader terug op dit aspect van object oriëntatie. Over encapsulatie van gegevens kunnen we kort zijn: encapsulatie is niets meer of minder dan een bijzonder vorm van eenheid van logica en gegevens, die ervoor zorgt dat toegang tot de attributen van een object (zowel lezen als schrijven) altijd via een stuk, meest elementaire, logica verloopt (in de volksmond vaak 'getters' en 'setters' genoemd).

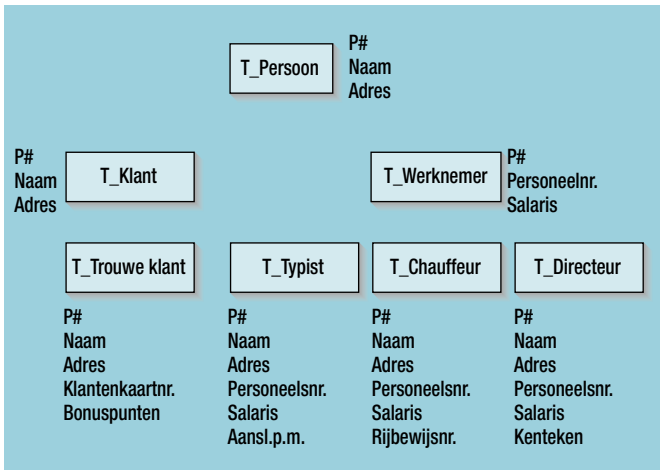
GENERALISATIE

Nu over naar het oorspronkelijke onderwerp van het vorige artikel in deze serie: de ondersteuning van typehiërarchieën. De fascinerende conclusie van Savalle hier is dat overerving beschouwd moet worden als een notatietruc. Er wordt ook beweerd dat problemen kunnen worden vermeden door het opstellen van class-modellen zonder generalisatie. In zijn algemeenheid is deze uitspraak zeker waar. Veel OO-ers beschouwen generalisatie als wondermiddel om stukken model aan elkaar te plakken, waar in veel gevallen compositie (het samenstellen van objecten middels (verwijzingen naar) andere objecten) een beter alternatief is. Dat daarmee generalisatie een non-issue in gegevensmodellering zou zijn is een conclusie die veel te ver gaat. Een treffende illustratie vormt de casus die we in het vorige artikel hebben gebruikt en die hierbij nogmaals kort is opgenomen.

Natuurlijk is het zo dat deze generalisatiestructuur valt te implementeren in een relationeel gegevensmodel. In het vorige artikel kwamen we hierbij tot twee basale oplossingsvarianten: een die uitgaat van één tabel per generalisatiehiërarchie en een die uitgaat van één tabel per klasse. Wanneer we de redeneringen van Savalle doortrekken (iets wat in zijn vertaaldiagrammen helaas niet buiten het domein van klassendiagrammen wordt gehaald) ontstaat nog een derde variant: die van één tabel per klasse, maar



FIGUUR 1: OBJECTMODEL CASUS.



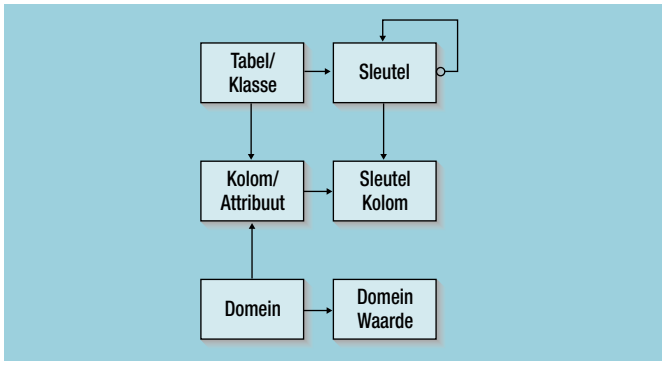
FIGUUR 2: RELATIONEEL MODEL CASUS VOLGENS 'METHODIEK' SAVALLE.

dan zonder onderlinge relaties. Immers: 'generalisatierelaties mogen niet worden overgenomen in de database', 'objecten hebben geen relatie, ook als classes die wel hebben' en 'abstracte elementen (hieronder verstaat Savalle ook generalisatierelaties) worden zorgvuldig verwijderd middels defactorisatie' (met de laatste term wordt het 'platslaan' van de generalisatiehiërarchie naar het niveau van de meest specifieke klasse bedoeld). Volgen we zijn instructies, dan ontstaat het relationele gegevensmodel uit figuur 2. Savalle's doelstelling is bereikt: onze een-uit-tien objectdatabase kent geen generalisatierelaties tussen records. Het nut van deze doelstelling ontgaat mij echter volledig: in plaats van een pleidooi te voeren voor relationele databases ten nadele van objectgeoriënteerde nieuwigheid, wordt een verkapte lans gebroken voor het geautomatiseerde equivalent van de aloude kaartenbak!

De rascynicus zal zeggen dat Savalle een treffende beschrijving van de huidige automatiseringspraktijk weet te geven. Voorlopig houd ik het erop dat we hier decennia terug in de tijd worden geworpen, een uitspraak die wat mij betreft tot op redelijke hoogte overigens ook voor state-of-the-art object-dbms'en geldt (en als u nu roept dat de titel van deze artikelserie misleidend is en niet het woord 'dbms' maar 'rdbms' had moeten bevatten dan geef ik u groot gelijk). Ik daag Patrick Savalle echter van harte uit om mij van het tegendeel te overtuigen. Laat het debat in DB/M leven!

GEGEVENSMODEL

Terug naar ons oorspronkelijke onderwerp. De vorige keer zijn we op het punt aangeland van de voor- en nadelen van de twee gangbare vormen van modellering van generalisatiehiërarchieën in rdbms'en. Ook hebben we aandacht besteed aan het afschermen van implementatie-specifieke aspecten onder databaseviews en triggers. Tot slot werden we jaloers op de gebruikers van DB2 die van IBM kant-en-klare ondersteuning voor generalisatiehiërarchieën krijgen aangeboden. Wat u nog tegoed heeft is een gegevensmodel van waaruit we de databaseobjecten (views, triggers en



FIGUUR 3: BASISMODEL VOOR EEN RELATIONELE CATALOGUS.

vooral niet te vergeten: constraints) automatisch kunnen laten genereren. Daarvoor gaan we eerst terug naar een oud gegevensmodel, dat de vaste lezers zich wellicht nog herinneren uit de 'tien geboden'-serie en de kern van een relationele catalogus beschrijft. Dit model is weergegeven in figuur 3.

In het model leggen we de definities vast van tabellen (of, zo u wilt, entiteiten of klassen) met hun bijbehorende kolommen danwel attributen. Van elk attribuut leggen we de definitie van de waarden die het attribuut mag aannemen, in een domein of gegegevenstype[1]. Op tabellen kunnen we sleutels definiëren, zowel unieke (primaire en alternatieve) sleutels als verwijzende sleutels, die ver-

Voorlopig houd ik het erop dat we hier decennia terug in de tijd worden geworpen

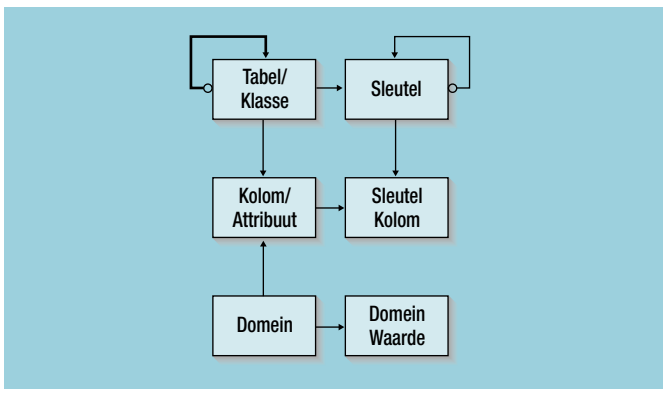
wijzen naar unieke sleutels. Een sleutel wordt gevormd door een of meerdere attributen uit de tabel waartoe de sleutel behoort, waarbij geldt dat de kolommen van verwijzende sleutels qua gegevenstype (domein) overeen moeten komen met het gegevenstype van de kolommen van de unieke sleutel waarnaar de verwijzende sleutel verwijst. Op basis van dit model is het mogelijk om database creatiescripts te genereren die in de database tabellen, kolommen, sleutels en (domein)constraints genereren. Het model is tevens aanhaakpunt voor de eerder in deze artikelserie beschreven gegevensmodellen betreffende validatie en autorisatie.

Het uitbreiden van dit gegevensmodel om generalisatiehiërarchieën te kunnen modelleren begint en eindigt ook direct met de mogelijkheid om, populair relationeel gezegd, tabellen onder tabellen te kunnen 'hangen'. Dat komt in nette OO-termen neer op klassen afleidbaar maken van andere klassen. In het gegevensmodel kunnen we dit mogelijk maken door een verwijzing van de tabel 'Tabel' naar zichzelf op te nemen, met als betekenis 'de tabel/klasse waarvan deze tabel/klasse is afgeleid'.

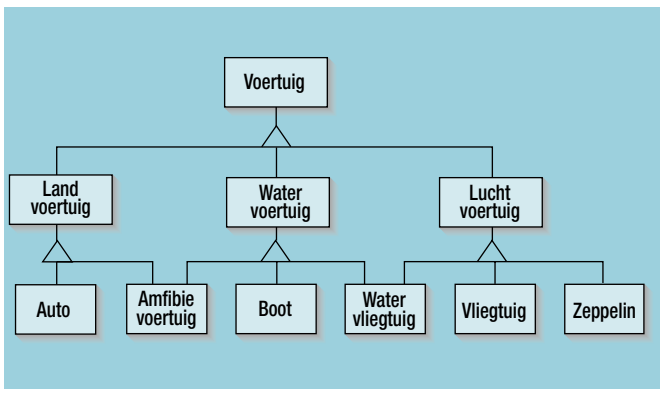
MEERVOUDIGE OVERERVING

Met het definiëren van een tabelhiërarchie verandert ook de semantiek van de tabellen 'Kolom' en 'Sleutel'. Zo verandert de semantiek van 'Kolom' van 'de kolommen van een tabel' in 'de kolommen van een tabel die toegevoegd worden aan de kolommen die de tabel overerft van de hoger in de hiërarchie gelegen tabellen' en geldt voor sleutels een vergelijkbare semantiek. Belangrijk is ook om te beseffen dat deze definitie van de tabel 'Kolom' betrekking heeft op het gegevensmodel 'zoals de programmeur dat ziet' (ik vermijd met opzet termen als 'conceptueel model' of 'logisch model' omdat deze termen in de praktijk voor zeer diverse doeleinden worden gebruikt). Het gegevensmodel zegt nog niets over de vorm waarin het gegevensmodel uiteindelijk middels relationele structuren wordt geïmplementeerd in de database. De beide varianten die in het vorige artikel zijn besproken blijven mogelijk.

Iets wat in ons model niet mogelijk is, is meervoudige overerving, waarbij een klasse van meerdere klassen overerft. Meervoudige overerving is een van de hete hangijzers in de objectoriëntatie, waar zowel theoretisch als praktisch een hoop haken en ogen aan zitten. In de praktijk wordt meervoudige overerving in programmeertalen weinig ondersteund en nog minder gebruikt. Bij OO-opleidingen wordt steevast teruggegrepen op de casus van modellering van verschillende soorten voertuigen in een generalisatiehiërarchie, waarbij de klassen 'watervliegtuig' en 'amfibievoertuig' het nut van meervoudige overerving moeten illustreren.



FIGUUR 4: BASISMODEL, UITGEBREID MET ONDERSTEUNING VOOR GENERALISATIE.



FIGUUR 5: DE VOERTUIGCASUS.

Stel dat we toch vasthouden aan het mogelijk maken van meervoudige overerving, dan is het interessant te zien hoe dit uitwerkt in de twee implementatievarianten. In de variant met één tabel per hiërarchie is er ogenschijnlijk weinig aan de hand, maar dient men te beseffen dat door een welgeplaatste meervoudige overerving hiërarchieën samen kunnen gaan vallen. In de praktijk zorgt dit overigens zelden voor problemen, zie de voertuig-casus waarin zowel 'watervliegtuig' als 'amfibievoertuig' ook zonder meervoudige overerving al in de hiërarchie 'voertuig' zouden zitten. In de variant met één tabel per klasse verandert er ook niet veel. De impact van samenvoeging van twee hiërarchieën uit zich hier in het niet samenvallen van de sleutelattributen waarin de meervoudig overervende tabel naar de hoger in de hiërarchie gelegen tabellen verwijst. In de voertuigcasus vallen deze sleutelattributen juist wel samen: in de tabel 'Amfibievoertuig' verwijst het sleutelattribuut 'Voertuig-ID' zowel naar de tabel 'Landvoertuig' als naar de tabel 'Watervoertuig'.

Meervoudige overerving is een van de hete hangijzers in de objectoriëntatie

BLIK VOORUIT

In het volgende nummer ronden we de bespreking van de generalisatieproblematiek af. We gaan het dan hebben over hiërarchieën op domeinniveau (subdomeinen) en bespreken nog enkele addertjes onder het gras bij het implementeren van generalisatiehiërarchieën in een relationeel DBMS. We sluiten af met een uitwerking van de casus conform de implementatievariant van één tabel per generalisatiehiërarchie. ●

Ir. F.G.W. van Orden (fridoo@faapartners.com) is managing partner van FAA Partners BV

Noot:

1. Interessant in dit kader is dat het concept 'gegevenstype' weer grote conceptuele overeenkomsten vertoont met het OO-concept 'klasse'. Er is dan ook een stroming in databaseland die de veel voorkomende mapping van het relationele concept 'tabel' op het OO-concept 'klasse' verkettert en van mening is dat het enig juiste relationele equivalent van 'klasse' 'domein' is. De bekendste vertegenwoordiger van deze stroming is Chris Date, die hierover een uitgebreide verhandeling in boekvorm heeft gepubliceerd onder de titel 'The third manifesto'. Verplichte kost voor alle relationelen, objectgeoriënteerden en alles wat daar tussenin zit.