

1990-57

Faculteit der Economische Wetenschappen en Econometrie

ET

05348

Serie Research Memoranda

Generalisatie in een relationele omgeving:
een benadering gericht op het terugdringen van onderhoud

R.J. Veldwijk
M.V. van Dijk
M. Boogaard
E.R.K. Spoor

Research Memorandum 1990-57
November 1990



Generalisatie in een relationele omgeving

een benadering gericht op het terugdringen van onderhoud

R.J. Veldwijk¹
M.V. van Dijk
M. Boogaard
E.R.K. Spoor

Het onderhouden van informatiesystemen legt veelal beslag op een groot deel van de automatiseringscapaciteit van een organisatie. De oorzaak daarvan is terug te voeren op de starheid van deze systemen en wordt deels veroorzaakt door de eigenschappen van het onderliggende datamodel. Dit artikel introduceert een benadering, gebaseerd op integratie van data en meta-data, waarmee dit bezwaar wordt ondervangen. Deze dictionary-benadering is gebaseerd op relationele principes en is daardoor toepasbaar in RDBMS-omgevingen. De mogelijkheden van de benadering worden besproken aan de hand van het generalisatieprobleem. Deze veel voorkomende maar vaak niet als zodanig herkende structuur is binnen een relationeel kader niet goed te modelleren en leidt mede daardoor tot grote onderhoudsinspanningen. Aan de hand van een vergelijking met twee gangbare modellen wordt getoond dat een data dictionary model een betere oplossing biedt voor generalisatieproblemen. Bovendien wordt aannemelijk gemaakt dat een dergelijk model ook voor andere onderhoudsproblemen een oplossing kan bieden.

1 Inleiding

Informatiesystemen zijn hulpmiddelen van mensen en organisaties. Een systeem dat niet aansluit op de behoeften van zijn gebruikers heeft geen waarde. Dit onomstreden uitgangspunt geeft aanleiding tot ruime aandacht voor methoden en technieken waarmee deze aansluiting van informatiesystemen op informatiebehoeften kan worden gerealiseerd. De gangbare methodieken beschouwen systeemontwikkeling of pakketselectie als succesvol wanneer het eindresultaat binnen randvoorwaarden van kosten en doorlooptijd aansluit op de behoeften van de gebruikers [BEME87]. Helaas zijn die behoeften doorgaans aan moeilijk te voorspellen veranderingen onderhevig. Daarom hoeft een systeem dat initieel aansluit op de behoeften van zijn gebruikers nog geen goed systeem te zijn. De conclusie is dat een systeem *duurzaam* moet aansluiten op de onderkende informatiebehoeften.

Duurzame aansluiting van een systeem op veranderende informatiebehoeften vereist dat deze veranderingen ook in het systeem worden doorgevoerd. Een *open* systeem bezit zelf de flexibiliteit om zich aan te passen; bij een *gesloten* systeem is ingrijpen van buitenaf nodig [KRAM87].

Helaas zijn geautomatiseerde systemen in vergelijking tot systemen met een menselijke component sterk gesloten. Deze geslotenheid komt veelvuldig terug in methodologische en bedrijfseconomische discussies in begrippen als "software crisis" en "application backlog". Het onderhoudsprobleem is wellicht het belangrijkste onopgeloste probleem op het vakgebied van de informatica. Onlangs verwoordde prof. dr. M. Verhelst dit probleem als volgt: "In de informatiemaatschappij wordt flexibiliteit van bestuurlijke informatiesystemen een strategische succesfactor" [VERH90].

Als we deze stelling onderschrijven dan verdient het begrip flexibiliteit op zichzelf in elke fase van systeemontwikkeling aandacht. Flexibiliteit dient zodanig in een systeem te zijn "ingebakken" dat onderhoud door automatiseringsdeskundigen vergaand wordt teruggedrongen.

Een systeem dat in hoge mate aan deze eis voldoet is pas echt "open" ofwel *inherent flexibel*. Een informatiesysteem is inherent flexibel als het kan voorzien in gewijzigde informatiebehoeften zonder dat menselijk ingrijpen in de programmatuur en/of de logische database-structuur noodzakelijk is.

¹ Zie bijlage I voor een beschrijving van de onderzoekscontext.

Op het gebied van de methodologie bestaat vooral aandacht voor het voorkomen en het efficiënt doorvoeren van onderhoud. Het gaat hier om vormen van niet-inherente flexibiliteit. Middelen daartoe zijn: een adequate opzet van de automatiseringsorganisatie, een goede analyse van informatiebehoeften en een gestandaardiseerde en gedocumenteerde aanpak van de systeemontwikkeling [BEME87] [SLJT88]. Aan inherente flexibiliteit wordt weinig of geen aandacht besteed.

Het streven naar inherente flexibiliteit is wel waarneembaar op het gebied van de relationele database management systemen (RDBMS). RDBMSen ondersteunen *fysieke* gegevens-onafhankelijkheid zeer goed, waardoor veel applicatie-onderhoud overbodig wordt [DATE90a]. Het "tunen" van systemen als gevolg van wijzigingen in het systeemgebruik door middel van indexering of clustering leidt niet meer tot wijzigingen in de bronprogrammatuur.

Op het tussenliggende gebied van logisch database- en programma-ontwerp ontbreekt de belangstelling voor inherente flexibiliteit nagenoeg geheel. De belangstelling gaat hier veel meer uit naar benaderingen en hulpmiddelen die zich richten op het verhogen van de produktiviteit van systeemontwikkeling en systeemonderhoud door toepassing van 4e-generatie talen en diverse soorten CASE-hulpmiddelen. Aandacht voor *logische* gegevensonafhankelijkheid beperkt zich tot toepassing van views bij veranderingen op logisch niveau [DATE90a]. De mogelijkheden die het gebruik van views biedt zijn echter zeer beperkt.

De onevenredig grote aandacht van systeemontwikkelaars voor produktiviteit in vergelijking met inherente flexibiliteit is een vorm van suboptimalisatie die vooral schadelijk is op het terrein van management-informatievoorziening en beslissingsondersteuning. De reden daarvoor is dat deze gebieden zich kenmerken door frequente en ingrijpende structurele wijzigingen in de informatiebehoeften. Benaderingen die uitzicht bieden op een hoger niveau van logische gegevensonafhankelijkheid en daarmee van inherente flexibiliteit zijn vooral hier noodzakelijk.

In dit artikel introduceren wij aan de hand van een voorbeeld een benadering die uitzicht biedt op een verhoging van inherente flexibiliteit en daarmee op een vermindering van onderhoudskosten. Het uitgangspunt van deze benadering is dat het vermijden van "hard coding" in zowel de applicaties als in de database-structuur, door het gebruik van een geïntegreerde data dictionary, leidt tot inherent flexibele informatiesystemen. Een belangrijk kenmerk van de benadering is dat implementatie zonder problemen op bestaande RDBMSen kan plaatsvinden. Het blijkt dat nog lang niet alle mogelijkheden van het relationele model² zijn benut.

In de volgende paragrafen beperken wij de problematiek van flexibiliteit tot het gebied van generalisatie en specialisatie, ook wel aangeduid met de termen super- en subtyping. De reden daarvoor is tweeledig, namelijk (1) het generalisatieprobleem is een algemeen voorkomend probleem binnen informatiesystemen dat een belangrijke aanleiding vormt voor onderhoud van informatiesystemen en (2) het generalisatieprobleem vormt een goede basis voor het verkrijgen van inzicht in de dictionary-benadering.

In paragraaf 2 wordt het generalisatieprobleem aan de hand van een praktijkvoorbeeld geschetst. Tevens worden twee benaderingen besproken die normaliter fungeren als oplossing voor het generalisatieprobleem. In paragraaf 3 komen de tekortkomingen van deze twee benaderingen aan de orde. Paragraaf 4 introduceert een alternatieve benadering op basis van een geïntegreerde data dictionary, waarna in paragraaf 5 wordt aangetoond dat deze dictionary-benadering niet lijdt onder de tekortkomingen van de traditionele oplossingen. Paragraaf 6 bespreekt een tweetal problemen die voortvloeien uit de dictionary-benadering. Tenslotte wordt in paragraaf 7 kort aangegeven welke perspectieven een algemeen data dictionary model biedt voor andere modelleringsproblemen.

² In dit artikel spreken wij zowel over modellen zoals "het relationele model" als over modellen die automatiseerders opstellen en implementeren. Het onderscheid tussen deze modellen is belangrijk, maar blijkt uit de context.

2 Twee benaderingen van het generalisatieprobleem

2.1 Generalisatie en specialisatie

Bij het ontwerpen van een logisch gegevensmodel vindt men vaak verzamelingen van objecten die voor sommige informatiebehoeften als één geheel moeten worden beschouwd (generalisatie), terwijl voor andere informatiebehoeften slechts een deelverzameling (specialisatie) relevant is. Generalisatie/specialisatie ofwel supertyping/subtyping komt veelvuldig voor, vooral bij systemen die meerdere functionele gebieden bestrijken. De meer recente methoden van gegevensmodellering zoals RM/T [CODD79], NIAM [NIJS89] en de Object-georiënteerde aanpak [MEYE88] bevatten concepten waarmee de ontwerper generalisaties expliciet kan weergeven. Het relationele model en de daarop gebaseerde RDBMSen kennen deze concepten echter niet. Het omgaan met generalisatieproblemen wordt daarmee een taak van het applicatieprogramma. Dit geeft aanleiding tot veel programmeerwerk, zowel bij systeembouw als systeemonderhoud.

2.2 Een voorbeeld uit de praktijk

Er bestaan praktijksituaties waarbij het generalisatieprobleem schijnbaar onoverkomelijke problemen oplevert. Een karakteristiek voorbeeld bespreken wij in sterk vereenvoudigde vorm in dit artikel. Het betreft een generalisatieprobleem dat speelde bij het ontwerp, uitgevoerd door RAET NV, van een consolidatiesysteem voor een internationaal opererend concern. Aanleiding voor de ontwerpopdracht vormden de zeer hoge onderhoudskosten van een al bestaand consolidatiesysteem.

Het consolidatiesysteem bevat de balansen van alle dochters van het concern. Oppervlakkig beschouwd is er sprake van een zeer eenvoudig gegevensmodel³ waarin gegevens over bedrijven, soorten balansposten en balansbedragen van werkmaatschappijen worden vastgelegd:

BEDRIJF (BED#, NAAM, BEZIT)

BED#	NAAM	BEZIT
TOP	Topholding	100%
EURO	Holding Europa	100%
AZIE	Holding Azie	80%
THA	Bedrijf Thailand	60%

BP_SOORT (BP#, NAAM, DC)

BP#	NAAM	DC
GEB	Gebouwen	DEB
TER	Terreinen	DEB
OVA	Ov. activa	DEB
EVM	Eig. Verm.	CR
VVM	Vr. Verm.	CR

BALPOST (BED#, BP#, BEDRAG)

BED#	BP#	BEDRAG
THA	GEB	200
THA	TER	50
THA	OVA	150
THA	EVM	300
THA	VVM	100
....	...	...

Fig. 1: Basismodel van het consolidatiesysteem

³ In dit vereenvoudigde gegevensmodel ontbreken aspecten als historie, juridische verhoudingen, intercompany-verhoudingen, pro-memorie posten, resultatenrekeningen, etc.

De tabel **BEDRIJF** bevat gegevens over de dochterbedrijven die elk voor zich een jaarrekening opstellen, van waaruit de geconsolideerde jaarrekening wordt samengesteld. **BP_SOORT** bevat de balansposten waaruit elke bedrijfsbalans is opgebouwd. De balansbedragen van de bedrijven worden weergegeven door de tabel **BALPOST**. Voor elke balanspost van elk bedrijf wordt één tupel opgenomen.

Dit eenvoudige model is voldoende krachtig om de balansen van de verschillende bedrijven op te slaan. De eenvoud van de gegevensstructuur is terug te vinden in de structuur van de vele programma's die de balansgegevens verder verwerken (consolideren) tot management-informatie. Deze informatie bestaat veelal uit gecomprimeerde en geconsolideerde balansen. Het systeem kent ca. 150 detail-balansposten (**BP_SOORT**-tupels) en een zeer groot aantal bedrijven.

Helaas is dit model onvoldoende voor de financiële analisten. Voor deze gebruikers dient het gegevensmodel voor elke balanspost detailgegevens te bevatten. De gegevens die moeten worden vastgelegd zijn afhankelijk van de soort balanspost en veranderen regelmatig. We zullen het voorbeeld daarom uitbreiden met de veronderstelling dat we de onderstaande specifieke attributen moeten toevoegen:

voor "Gebouwen"	: Aankopen , Verkopen en Afschrijving
voor "Terreinen"	: Aankopen en Verkopen
voor "Overige Activa"	: niets
voor "Eigen Vermogen"	: Betaalde rente (= dividend)
voor "Vreemd Vermogen"	: Betaalde rente

Ook hier gaat het in werkelijkheid om veel meer gegevens. De ca. 150 soorten balansposten (hierboven 5) kunnen worden onderverdeeld in circa 42 groepen (hierboven: 4). Elke groep heeft een eigen set specifieke attributen, gemiddeld circa 5. Bepaalde attributen, zoals "Aankopen" en "Verkopen" in het bovenstaande voorbeeld komen voor in meerdere groepen.

Het beschreven gegevensmodel moet worden uitgebreid om in de behoeften van de analisten te kunnen voorzien. Daarvoor bestaan verschillende oplossingen die elk hun tekortkomingen hebben. In het vervolg van deze paragraaf bespreken wij de twee meest bekende.

2.3 Een primitieve benadering

Een mogelijke aanpak van het generalisatieprobleem is het negeren van het onderscheid tussen klassen en subklassen. Om deze reden kan deze benadering "primitief" worden genoemd. Bij de primitieve benadering heeft de ontwerper de volgende keuze: (1) alle subklassen worden beschouwd als losstaande klassen of (2) het bestaan van aparte subklassen wordt genegeerd.

De eerste oplossing houdt voor ons voorbeeld in dat de tabel **BALPOST** wordt gesplitst in de volgende vier tabellen:

```

BALPOST_GEB (BED#, BP#, BEDRAG, AANK, VERK, AFSCHR)
BALPOST_TER (BED#, BP#, BEDRAG, AANK, VERK)
BALPOST_OVA (BED#, BP#, BEDRAG)
BALPOST_VMO (BED#, BP#, BEDRAG, RENTE)

```

De consequentie hiervan is dat programma's die op de balans als één geheel werken nodeloos complex worden. Voor een tabel met veel subklassen zoals **BALPOST** is deze benadering niet acceptabel⁴. Wij zullen daarom verder uitgaan van de tweede oplossing volgens welke het bestaan van de subklassen wordt genegeerd. Deze oplossing leidt tot een uitbreiding van de tabel **BALPOST** met een aantal attributen:

⁴ Natuurlijk is het mogelijk om een view over de verschillende subklassen te leggen. Deze view legt een UNION over de gemeenschappelijke attributen van de verschillende subklasse-tabellen. Los van het feit dat er problemen kleven aan het gebruik van de UNION-operator kan worden opgemerkt dat een dergelijke oplossing niet aan de definitie "primitief" voldoet. In feite is het een implementatiewijze voor de klassieke benadering (zie paragraaf 2.4).

BALPOST (BED#, BP#, BEDRAG, AANK, VERK, AFSCHR, RENTE)

BED#	BP#	BEDRAG	AANK	VERK	AFSCHR	RENTE
THA	GEB	200	50	0	10	---
THA	TER	50	---	20	---	---
THA	OVA	150	---	---	---	---
THA	EVM	300	---	---	---	10
THA	VVM	100	---	---	---	20
....		...	...	...	...	...

Fig. 2: Primitief model van het consolidatiesysteem

Deze implementatiewijze vereist dat de attributen AANK, VERK, AFSCHR en RENTE de NULL-waarde mogen aannemen. Een NULL-waarde bij AANK voor de post "Overige Activa" heeft de betekenis "niet van toepassing", terwijl een NULL-waarde voor dit attribuut bij "Terreinen" de betekenis "onbekend" heeft. Het begrip NULL kan dus verschillende betekenissen hebben. De kennis met betrekking tot de betekenis van deze NULL-waarden kan niet aan het relationele model, laat staan aan een daarop gebaseerd RDBMS worden opgelegd en dit levert allerlei problemen op [DATE89]. Het afdwingen van de semantiek in het logische gegevensmodel moet daarom worden overgelaten aan de programmatuur⁵ in casu aan de programmeur. Dit brengt extra bouw- en onderhoudskosten met zich mee en draagt het risico van integriteitsschendingen in zich indien de programmatuur niet correct werkt. Niettemin is deze wijze van implementatie acceptabel⁶ wanneer:

- 1- er niet al te veel subklassen van eenzelfde objectklasse voorkomen met verschillende specifieke eigenschappen;
- 2- er niet veel attributen zijn die op meerdere subklassen van toepassing zijn (zoals in ons voorbeeld AANK, VERK en RENTE);
- 3- het aantal subklassen en de eigenschappen van die subklassen niet te sterk verandert;
- 4- de databaseontwerper alle NULL-regels opneemt in de beschrijving van zijn gegevensmodel.

De primitieve benadering is vermoedelijk de meest toegepaste. Er zijn weinig modellen waar deze vorm van generalisatie geheel ontbreekt. Modellen waarin de eerstgenoemde primitieve modellerwijze is toegepast (alle subklassen geheel uitsplitsen) duiden er vaak op dat de ontwerper de mogelijkheid van generalisatie over het hoofd heeft gezien.

2.4 De klassieke benadering

Het onderscheid tussen objectklassen en -subklassen kan ook op een meer expliciete manier worden weergegeven. Deze door Smith en Smith [SMIT77] geïntroduceerde en door vele andere auteurs van studieboeken [BEKK88] [LAAG87] overgenomen klassieke modellerwijze maakt het mogelijk om de subklassen expliciet in het model tot uitdrukking te brengen door voor elke subklasse⁷ een aparte tabel te introduceren waarin de voor die klasse geldende attributen

⁵ De volgende generatie RDBMSen zal de mogelijkheid bieden tot het specificeren van zogenaamde database-procedures. Daarmee kan meervoudig programmeerwerk worden voorkomen. Dat neemt niet weg dat de informatie dat het AFSCHR-attribuu alleen voor "gebouwen" van toepassing is niet als zodanig in het database-model kan worden opgenomen. Vergelijk dit met regels als "Elk bedrijf heeft een unieke code" en "elk balansbedrag hoort bij één bedrijf" die als het ware in het model zijn ingebakken.

⁶ Wij gaan niet in op het probleem van geheugenbeslag. Veelal speelt dit probleem in het geheel niet omdat het RDBMS ervoor zorg draagt dat er geen geheugenruimte wordt verspild.

⁷ Strikt genomen voor elke groep van subklassen met één en dezelfde set attributen.

worden opgenomen. De attributen die alle subklassen gemeen hebben blijven opgenomen in de oorspronkelijke tabel.

Op de klassieke wijze kan het consolidatievoorbeeld op de onderstaande wijze worden gemodelleerd:

BP_SOORT (BP#, NAAM, DC, BP_GROEP)
 BALPOST (BED#, BP#, BEDRAG)
 BALPOST_GEB (BED#, BP#, AANK, VERK, AFSCHR)
 BALPOST_TER (BED#, BP#, AANK, VERK)
 BALPOST_VMO (BED#, BP#, RENTE)

BALPOST

BED#	BP#	BEDRAG
THA	GEB	200
THA	TER	50
THA	OVA	150
THA	EVM	300
THA	VVM	100
....	...	...

BP_SOORT (BP#, NAAM, DC, BP_GROEP)

BP#	NAAM	DC	BP_GROEP
GEB	Gebouwen	DEB	BALPOST_GEB
TER	Terreinen	DEB	BALPOST_TER
OVA	Ov. activa	DEB	---
EVM	Eig. Verm.	CR	BALPOST_VMO
VVM	Vr. Verm.	CR	BALPOST_VMO

BALPOST_GEB

BED#	BP#	AANK	VERK	AFSCHR
THA	GEB	50	0	10
....	...	...	...	...

BALPOST_TER

BED#	BP#	AANK	VERK
THA	TER	---	20
....	...	...	...

BALPOST_VMO

BED#	BP#	RENTE
THA	EVM	10
THA	VVM	20
....	...	...

Fig. 3: Klassiek model van het consolidatiesysteem

De attributen BED# en BP# vormen nog steeds de primaire sleutel van de generalisatietabel BALPOST. Deze attributen vormen tevens de primaire sleutel van de drie specialisatietabellen en verwijzen tevens naar de BALPOST-tabel. De relaties tussen BALPOST enerzijds en BALPOST_GEB, BALPOST_TER en BALPOST_VMO anderzijds geven dus één-op-één verhoudingen weer. De specialisatietabel waarin de subklassegegevens zijn opgenomen is altijd bekend omdat de tabel BP_SOORT is uitgebreid met een attribuut BP GROEP. De waarden die dit attribuut mag aannemen, corresponderen met de namen van de specialisatietabellen. BALPOST_GEB-tupels die niet naar een BALPOST-tupel verwijzen kunnen niet voorkomen (referentiële integriteit). Voor een BALPOST-tupel waarvoor een BALPOST_GEB-tupel bestaat kan de ontwerper al dan niet specificeren dat de attributen AANK, VERK en AFSCHR moeten bestaan. Indien een dergelijk attribuut een NULL-waarde aanneemt dan kan de betekenis daarvan worden geïnterpreteerd als "onbekend". Ook maakt het model nu volstrekt duidelijk welke klassen van balanspostsoorten er bestaan. Per saldo levert deze modelleerwijze een betere weergave van de werkelijkheid in vergelijking met de weergave volgens de primitieve benadering.

3 Onopgeloste problemen

Zowel het primitieve als het algemeen aanbevolen klassieke model geven aanleiding tot een aantal problemen. De oorzaak daarvan ligt in het feit dat het relationele model niet krachtig genoeg is om generalisatieproblemen op een natuurlijke wijze weer te geven. Deze paragraaf bespreekt deze problemen, die deels al door Smith en Smith worden geconstateerd. Deze auteurs stellen voor om het relationele model uit te breiden teneinde de tekortkomingen op te heffen. Omdat de huidige generatie RDBMSen en de taal SQL gebaseerd zijn op het oorspronkelijke relationele model leidt elke vorm van generalisatie tot programmeerproblemen.

De zwaktes van het klassieke model hebben betrekking op integriteitsbewaking, programmeerinspanning en inflexibiliteit. In deze opzichten blijkt er geen wezenlijk verschil tussen het primitieve model van paragraaf 2.3 en het klassieke model van paragraaf 2.4.

3.1 Integriteitsbewaking

Het klassieke model wordt gezien als een verbetering in vergelijking met het primitieve model omdat het model kennis bevat over de subklassen die worden onderscheiden en de attributen die deze subklassen hebben. Bij het primitieve model van figuur 2 moet de programmatuur deze integriteitsregels afdwingen. Een implementatie van het model in figuur 3 staat echter nog steeds toe dat in ons voorbeeld: (1) een tuple in BALPOST GEB voor het attribuut BP# de waarde 'TER' heeft en (2) naar een BALPOST-tuple in meerdere specialisatie-tabellen wordt gerefereerd.

Smith en Smith stellen voor om dit probleem op te lossen door het relationele model zodanig uit te breiden dat het attribuut BP GROEP als waarde alleen de naam van een bestaande specialisatietabel mag aannemen. Een op dit uitgebreide relationele model gebaseerd RDBMS moet dan afdwingen dat de integriteit gehandhaafd blijft. Zolang deze uitbreiding niet is geïmplementeerd in een RDBMS zal de programmeur het werk moeten doen. Mede hierom vormt het klassieke model geen ideale oplossing voor de software-ontwikkelaar.

3.2 Programmeerinspanning

Zelfs als het RDBMS automatisch alle regels zou afdwingen leidt het model tot grote programmeerinspanning. Neem als voorbeeld de informatiebehoefte "Geef alle gegevens over de verschillende terreinen". Bij de primitieve modelleerwijze van figuur 2 lijkt dit geen problemen op te leveren.

In SQL:

```
SELECT * FROM BALPOST WHERE BP# = 'TER'
```

Resultaat:

BED#	BP#	BEDRAG	AANK	VERK	AFSCHR	RENTE
---	---	-----	----	----	-----	-----
THA	TER	50	---	20	---	---
...	...	...	...	...	...	...

Fig. 4A: Een SQL-query op één soort balanspost voor het primitieve model.

Dit resultaat is niet te interpreteren omdat irrelevante en ontbrekende gegevens worden vermengd. Al met al is er geen sprake van informatie op maat, zeker wanneer de gebruiker in de praktijk met meer dan honderd irrelevante attributen wordt geconfronteerd.

Bij klassieke model van figuur 3 liggen de zaken anders. De gebruiker moet nu twee SQL-statements opgeven om de informatie boven water te halen:

```

1: SELECT BP_GROEP FROM BP_SOORT WHERE BP# = 'TER'

Resultaat:
BP_GROEP
-----
BALPOST_TER

2: SELECT BALPOST.BED#, BALPOST.BP#, BEDRAG, AANK, VERK
   FROM BALPOST, BALPOST_TER
   WHERE BALPOST.BP# = 'TER'
         AND BALPOST.BP# = BALPOST_TER.BP#
         AND BALPOST.BED# = BALPOST_TER.BED#

Resultaat:
BED# BP# BEDRAG AANK VERK
---- --
THA  TER      50  ---   20
...  ...      ...  ...   ...

```

Fig. 4B: Een SQL-query op één soort balanspost voor het klassieke model

Het resultaat is verkregen door eerst de naam van de specialisatie-tabel BALPOST_TER op te vragen en deze vervolgens te gebruiken om de tweede query op te stellen. Omdat het gegevensmodel meer semantiek bevat dan de primitieve versie is de informatie nu op maat. De prijs daarvoor is echter het vervangen van één eenvoudig SQL-statement door twee minder eenvoudige⁸.

Een ernstiger interpretatieprobleem ontstaat bij de informatiebehoefte "Geef de aankoopbedragen van alle relevante balansposten". In het primitieve model lijkt het SQL-statement van figuur 5A correct, maar levert niet de gewenste uitkomst op. Het statement van figuur 5B verdient sterk de voorkeur:

⁸ Smith & Smith constateren zelf al dat er behoefte is aan een tweede orde querytaal. In een dergelijke taal is het mogelijk om de namen van tabellen en attributen te gebruiken als variabelen. In ons voorbeeld zou dat de volgende query opleveren:

```

SELECT BALPOST.* , &TABEL.*
FROM BP_SOORT, BALPOST, &TABEL
WHERE &TABEL      = BP_GROEP
      AND BP_SOORT.BP# = 'TER'
      AND BALPOST.BP# = 'TER'
      AND &TABEL.BP#  = 'TER'
      AND BALPOST.BED# = &TABEL.BED#

```

```

SELECT BED#, BP#, AANK
FROM BALPOST
ORDER BY BED#, BP#

Resultaat:
BED# BP# AANK
---- --- ----
THA  EVM  ---
THA  GEB  50
THA  OVA  ---
THA  TER  ---
THA  VVM  ---
...   ...  ---

```

```

SELECT BED#, BP#, AANK
FROM BALPOST
WHERE BP# = 'GEB'
      OR BP# = 'TER'
ORDER BY BED#, BP#

Resultaat:
BED# BP# AANK
---- --- ----
THA  GEB  50
THA  TER  ---
...   ...  ...

```

Fig. 5A en 5B: SQL-queries op één subklasse-attribuut volgens het primitieve model

Het resultaat van 5A is nauwelijks te interpreteren wanneer er veel soorten balansposten zijn en het attribuut AANK voor de meeste posten niet relevant is. De query van figuur 5B verdient daarom de voorkeur, al zal de WHERE-clause in de praktijk uit een groot aantal vergelijkingen bestaan.

Omdat in het klassieke model expliciet wordt vastgelegd voor welke soorten balansposten de eigenschap AANK relevant is, kan het resultaat van fig. 5A niet voorkomen. Dat neemt niet weg dat er ook hier een complexe query noodzakelijk is:

```

SELECT BED#, BP#, AANK
FROM BALPOST_GEB
UNION
SELECT BED#, BP#, AANK
FROM BALPOST_TER

Resultaat:
BED# BP# AANK
---- --- ----
THA  GEB  50
THA  TER  ---
...   ...  ...

```

Fig. 5C: Een SQL-query op één subklasse-attribuut volgens het klassieke model

De voorbeelden laten zien dat zowel het primitieve model als het klassieke model aanleiding geven tot complex programmeerwerk. Extrapoleren we ons voorbeeld naar de geschetste praktijksituatie dan beslaan de besproken SQL-queries vaak meer dan één pagina.

Bij het primitieve model ligt de meeste inspanning in het handhaven van de integriteit. Het klassieke model levert in dit opzicht minder problemen op, maar daar staat tegenover dat sommige informatiebehoeften in relatief complexe queries resulteren. Veel auteurs geven de voorkeur aan het klassieke model [BEKK88] [LAAG87] [DATE90a]. De meeste systeemontwikkelaars vermijden liever situaties waarbij namen van tabellen en velden als variabelen moeten worden gebruikt. Zij kiezen voor het primitieve model of nemen alle

subklasse-tabellen apart op zonder zich te bekommeren om generalisatie⁹.

3.3 Inflexibiliteit

Het blijkt dat de generalisatieproblematiek bij de huidige stand van de techniek tot veel inspanning leidt om de integriteit van de gegevens te waarborgen en om correcte queries te realiseren. Het klassieke model biedt geen soelaas zolang het relationele model en de SQL-taal niet worden uitgebreid. Welke implementatie men ook kiest, altijd is er sprake van relatief omvangrijke en complexe programma's. Dit resulteert op zichzelf al in hoge bouw- en onderhoudskosten in omgevingen waarin deze problematiek sterk speelt. Wie bestaande systemen bekijkt vanuit deze probleemstelling constateert al snel dat generalisatieproblemen zeer frequent voorkomen, vooral bij systemen die bedrijfsbreed worden gebruikt.

De problemen worden verergerd doordat generalisaties van nature instabiel zijn. Nieuwe informatiebehoeften leiden al heel snel tot veranderingen in het aantal subklassen of in de structuur van de subklassen¹⁰. Zo treden er in de praktijk van het consolidatiesysteem elk jaar wijzigingen op in het aantal subklassen en in de attributen die voor de subklassen worden vastgelegd als gevolg van veranderde informatiebehoeften van het topmanagement. Elke wijziging leidt tot ingrijpen in de structuur van de database en dus tot ingrijpen in de programmatuur. De aard van deze wijzigingen is steeds dezelfde. In figuur 6 is aangegeven welke consequenties veranderingen in de generalisatiestructuur hebben voor de database-structuur. In het primitieve model is het aantal tabellen constant, maar wijzigt de attribootstructuur van de generalisatietabel. In het klassieke model resulteert dit vaak in het veranderen van het aantal subklasse-tabellen.

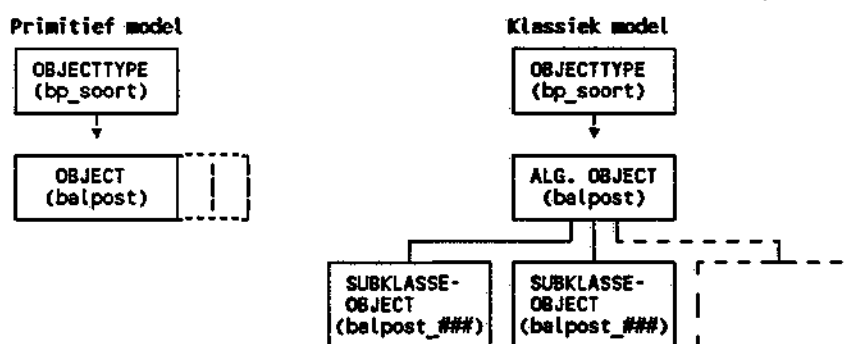


Fig. 6: Database-wijzigingen volgens het primitieve en het klassieke model

4 Een data dictionary model voor het generalisatieprobleem

Generalisatie is een probleem waarmee het relationele gegevensmodel geen raad weet. In dit licht is het te begrijpen dat er veel onderzoek wordt verricht naar alternatieven voor het relationele model. Sommige alternatieven, zoals de object-georiënteerde modellen [DATE90a] [MEYE88], beogen het relationele model te vervangen. Andere alternatieven, zoals RM/T en RM/V2 [CODD90] zijn uitbreidingen op het bestaande relationele model. Alle hebben ze gemeen dat generalisatie expliciet wordt onderkend en ondersteund. Omdat implementaties in commerciële (R)DBMSen nog niet gerealiseerd zijn, blijft de bruikbaarheid van de alternatieve modellen voorlopig beperkt tot ondersteuning van het ontwerpproces.

De hieronder te bespreken data dictionary benadering biedt een implementeerbare oplossing

⁹ De laatstgenoemde ontwerpkeuze komt zeer veel voor en kan worden herkend aan het grote aantal tabellen en het grote aantal programma-modules dat eenzelfde soort functionaliteit realiseert ("invoeren post gebouwen", "invoeren post terreinen" etc.).

¹⁰ Instabiliteit van gegevensmodellen wordt zelden als een probleem onderkend. Vrijwel algemeen gaat men uit van de fictie dat een gegevensgerichte modelleerwijze leidt tot een stabiel gegevensmodel. Wijzigingen zouden voornamelijk op het terrein van het omzetten van gegevens in informatie spelen en dus niet leiden tot wijzigingen op het gegevensmodel.

voor het generalisatieprobleem in de besproken vorm¹¹. De sleutel tot deze oplossing is de constatering dat noch het primitieve, noch het klassieke model informatie over zijn eigen structuur bevat. Dit gebrek aan "zelfkennis" leidt tot de besproken complexe SQL-queries en de gevoeligheid voor veranderingen in de structuur van de subklassen. De logische conclusie is dat de kennis over de structuur van de (sub)klassen niet moet worden weggestopt in de tabeldefinities. Als dergelijke structuurinformatie toegankelijk voor de programma's wordt gemaakt dan blijken de geconstateerde problemen te verdwijnen.

Op het eerste gezicht komt dit idee abstract over, maar op zichzelf bevat het niets nieuws. Een ontwerper die een BTW-percentage expliciet opneemt in een parametertabel in plaats van het weg te stoppen in de programmatuur doet in het klein wat ook op grotere schaal mogelijk is. Precies zoals een programma de actuele hoogte van de BTW uit de database haalt, is het mogelijk om de informatie over de gegevensstructuur uit de database te halen. Het idee om gegevens over de database-structuur op te nemen in een database is ook al niets nieuws. Elke data dictionary doet precies hetzelfde. Wat men zich meestal niet realiseert, is dat de *structuur* van de tabellen equivalent is met de *inhoud* van een data dictionary. Wijzigingen met behulp van DDL op de gegevensstructuur zijn derhalve equivalent met wijzigingen met behulp van DML op de dictionary-inhoud [LEK89]. Programma's die werken op de dictionary behoeven geen onderhoud bij mutaties op de inhoud van de dictionary-tabellen, net zoals een programma dat het BTW-percentage uit een tabel haalt ongevoelig is voor wijzigingen in het BTW-percentage. De sleutel tot de data dictionary benadering ligt in het integreren van de data dictionary met het te ontwikkelen systeem, in plaats van haar te beschouwen als een passieve opslag van ontwerpgegevens.

Bij de opzet van het consolidatiesysteem is op basis van deze uitgangspunten met succes gezocht naar een oplossing¹². Deze oplossing is gebaseerd op de gegevensstructuur die hieronder vereenvoudigd is weergegeven:

```

BALPOST      (BED# ,BP#)
BP_SOORT     (BP# ,NAAM , DC )
VAR_ATTRIBUUT (BP# ,VOLGNR, VAT_NM, OMSCHR, VERPL)
VAR_ATTRIBUUTTYPE (VAT_NM,TYPE , LENGTE)

```

BALPOST

BED#	BP#
THA	GEB
THA	TER
THA	OVA
THA	EVM
THA	VVM
...	...

BP_SOORT

BP#	NAAM	DC
GEB	Gebouwen	DEB
TER	Terreinen	DEB
OVA	Ov. activa	DEB
EVM	Eig.Verm.	CR
VVM	Vr. Verm.	CR

VAR_ATTRIBUUTTYPE

VAT_NM	TYPE	LENGTE
BEDRAG	NUM	4
AANK	NUM	3
VERK	NUM	3
AFSCHR	NUM	3
RENTE	NUM	3

¹¹ Er zijn ook andere vormen van generalisatie [BRAC83]. De in dit artikel besproken vorm is komt echter het meest voor.

¹² De eerste aanzet daarvoor is gegeven door de heer F. de Jel. De verdere uitwerking en de implementatie zijn verzorgd door R.J. Veldwijk.

VAR_ATTRIBUUT

BP#	VOLGNR	VAT_NM	OMSCHR	VERPL
GEB	1	BEDRAG	Balans gebouw	JA
GEB	2	AANK	Aankopen gebouw	NEE
GEB	3	VERK	Verkopen gebouw	NEE
GEB	4	AFSCHR	Afschr. gebouw	NEE
TER	1	BEDRAG	Balans terrein	JA
TER	2	AANK	Aankopen terrein	NEE
TER	3	VERK	Verkopen terrein	NEE
OVA	1	BEDRAG	Balans ov. activa	JA
EVM	1	BEDRAG	Balans EV	JA
EVM	2	RENTE	Dividend op EV	JA
VVM	1	BEDRAG	Balans VV	JA
VVM	2	RENTE	Rente op VV	JA

Fig. 7A: Alle structuurgegevens in een data dictionary

Net als het klassieke model beschrijft dit dictionary model welke attributen voor de (sub)klassen relevant zijn. Deze informatie is nu echter expliciet in de dictionary-tabellen VAR_ATTRIBUUT en VAR_ATTRIBUUTTYPE opgenomen en daarmee toegankelijk voor applicatieprogramma's. VAR_ATTRIBUUTTYPE beschrijft de klassen van attributen die bekend zijn, terwijl VAR_ATTRIBUUT aangeeft voor welke soorten balansposten deze relevant zijn. Het attribuut VERPL komt overeen met de NULL-clausule in het CREATE TABLE statement. Het is denkbaar dat een attribuuttype meer dan één keer voorkomt voor één soort balanspost (bijvoorbeeld "afschrijving tegen historische kosten" en "afschrijving tegen vervangingswaarde"). Om dit te ondervangen wordt het attribuut VOLGNR geïntroduceerd. Hoewel dit niet strikt nodig is, is het "vaste" attribuut BEDRAG van BALPOST ook in de dictionary opgenomen¹³.

Deze dictionary bevat alle relevante gegevens over de structuur van de generalisatie. Het is nu echter mogelijk geworden om de attributen van de subklassen te wijzigen door de dictionary te muteren. Uitbreiding van het model met gegevens over bijvoorbeeld de verkoopwaarde van de terreinen levert geen enkel probleem op. Zoals gezegd is een dergelijke dictionary niets nieuws. Het echte probleem dat moet worden opgelost is het weergeven van de inhoud van de verschillende tabellen. Dit blijkt mogelijk door het toevoegen van één enkele tabel waarin alle subklasse-waarden worden opgenomen:

¹³ Deze dictionary beschrijving vertoont grote gelijkenis met een beschrijving van de structuur van het relationele model zelf. Deze overeenkomst berust niet op toeval. Een nadere bespreking valt echter buiten het kader van dit artikel.

VAR_WAARDE (BED#, BP#, VOLGNR, WAARDE)

BED#	BP#	VOLGNR	WAARDE
THA	GEB	1	200
THA	GEB	2	50
THA	GEB	3	0
THA	GEB	4	10
THA	TER	1	50
THA	TER	2	---
THA	TER	3	20
THA	OVA	1	150
THA	EVM	1	300
THA	EVM	2	10
THA	VVM	1	100
THA	VVM	2	20
...	...	..	...

Fig. 7B: Alle attribuutwaarden in één tabel

Nu bevat het data dictionary model alle informatie die het klassieke model van figuur 3 bevat¹⁴. Weliswaar komt de tabel VAR_WAARDE ongestructureerd over, maar het model beschrijft evenals het klassieke model welke waarden zijn toegestaan doordat bij elk VAR_WAARDE-tupel precies één VAR_ATTRIBUUTTYPE-tupel behoort. Het data dictionary model is dus conceptueel ten minste even krachtig als het klassieke model¹⁵, maar is inherent flexibel in de zin van paragraaf 1. Figuur 8 geeft de gegevensstructuur voor het data dictionary model weer. In tegenstelling tot de modellen van figuur 6 komen structuurwijzigingen hier niet meer voor.

Data dictionary model

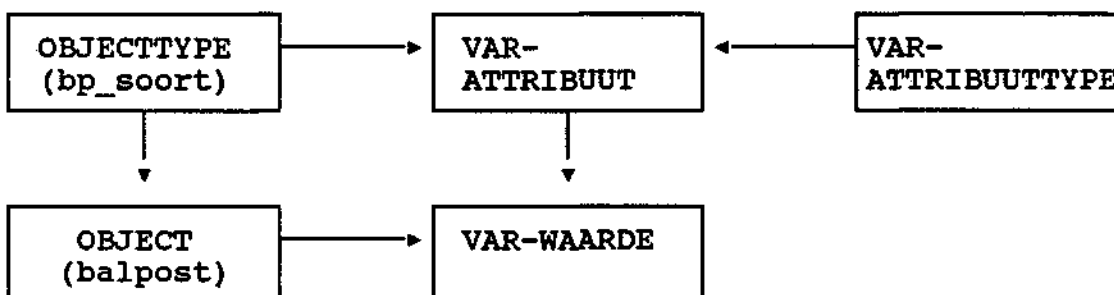


Fig. 8: Onveranderlijke structuur van het data dictionary model

¹⁴ Een goede manier om dit te verifiëren is het schrijven van een programma dat uitgaande van de dictionary-inhoud CREATE TABLE en INSERT-statements genereert. Bepaalde mutaties op de dictionary zijn equivalent met ALTER TABLE statements.

¹⁵ Dit betekent niet dat er helemaal geen problemen aan het model kleven. Deze problemen zijn echter merendeels van technische aard en komen in paragraaf 5 aan de orde.

5 Opgeloste problemen

Het data dictionary model biedt oplossingen voor de in paragraaf 3 besproken problemen van integriteitsbewaking, programmeerinspanning en inflexibiliteit.

5.1 Integriteitsbewaking

De grote zwakte van het primitieve model, het voorkomen van niet te interpreteren NULL-waarden, speelt noch in het klassieke model, noch in het data dictionary model. De tekortkoming van het klassieke model, het mogelijk voorkomen van balansposten die tot geen enkele of tot meerdere subklassen behoren, speelt evenmin. De in paragraaf 3.1 beschreven extra integriteitsregels zijn overbodig, de algemene referentiële integriteitsregel garandeert de integriteit van de dictionary. Daarmee verdwijnt ook de behoefte aan uitbreiding van het relationele model of van de relationele opvraagtafen. Elk RDBMS is dus in staat om het data dictionary model te ondersteunen al kent dit model twee regels die niet door het relationele model worden afgedwongen. De regels zijn: (1) de inhoud van het WAARDE-attriboot van elk VAR_WAARDE-tupel voldoet aan de formaatbeschrijving in het VAR_ATTRIBUTUTYPE-tupel waarnaar wordt verwezen en (2) VAR_WAARDE mag geen tupels bevatten met een NULL-waarde voor het attriboot WAARDE indien het attriboot VERPL van het bijbehorende VAR_ATTRIBUTUT-tupel de waarde 'JA' bevat. Deze regels zijn echter van een andere orde dan de twee regels van het klassieke model (zie figuur 3) die specificeren dat: (1) de inhoud van het attriboot BP GROEP in BP SOORT gelijk moet zijn aan de naam van één van de subklasse-tabellen en (2) voor elke tupel in BALPOST alleen in de subklasse-tabel waarnaar BP GROEP verwijst een tupel met dezelfde waarden voor BED# en BP# voor moet komen. Het verschil is dat een routine die de regels van het data dictionary model afdwingt ongevoelig is voor veranderingen in het aantal subklassen. Dat kan niet worden gezegd voor de regels van het klassieke model.

5.2 Programmeerinspanning

We bekijken opnieuw de twee informatiebehoeften die in paragraaf 3 zijn besproken. De eerste vraag was: "Geef alle gegevens over de verschillende terreinen". De SQL-queries van figuur 4 worden nu:

```
SELECT VW.BED#, VW.BP#, VA.OMSCHR, VW.WAARDE
FROM VAR_WAARDE VW, VAR_ATTRIBUTUT VA
WHERE VW.BP#      = 'TER'
      AND VA.BP#   = VW.BP#
      AND VA.VOLGMR = VW.VOLGMR
ORDER BY VA.VOLGMR
```

Resultaat:

BED#	BP#	OMSCHR	WAARDE
THA	TER	Balans terrein	50
THA	TER	Aankopen terrein	---
THA	TER	Verkopen terrein	20
...	...		...

Fig. 9: Een SQL-query op één soort balanspost voor het data dictionary model (vgl. fig. 4)

Hoewel de presentatie van het resultaat te wensen over laat leert vergelijking met de figuren 3A en 3B dat deze query in één keer de gevraagde informatie oplevert zonder dat er sprake is van

interpretatieproblemen¹⁶. Veel belangrijker is dat het SQL-statement geheel ongevoelig is voor de wijzigingen in de gegevens die over de terreinen worden vastgelegd. Bovendien leidt vervanging van de parameter 'TER' door een andere balanspost-code tot een geheel analoog resultaat. Hetzelfde SQL-statement kan derhalve steeds opnieuw worden gebruikt met andere waarden voor de variabelen.

Precies hetzelfde geldt voor de tweede informatiebehoefte: "Geef de aankoopbedragen van alle relevante balansposten". De SQL-queries van figuur 5 worden nu:

```
SELECT VW.BED#, VW.BP#, VW.WAARDE
FROM VAR_WAARDE VW, VAR_ATTRIBUUT VA
WHERE VA.VAT_NM = 'AANK'
      AND VA.BP# = VW.BP#
      AND VA.VOLGNR = VW.VOLGNR
ORDER BY VW.BED#, VW.BP#, VW.VOLGNR
```

Resultaat:

BED#	BP#	WAARDE
THA	GEB	50
THA	TER	---
...	...	...

Fig. 10: Een SQL-query op één subklasse-attribuut voor het data dictionary model (vgl. fig. 5)

Ook deze query is volledig ongevoelig voor wijzigingen in de structuur van de generalisatie. Bovendien hoeft de programmeur niet te weten voor welke soorten balansposten het gevraagde attribuut relevant is.

5.3 Inflexibiliteit

De in paragraaf 3.3 besproken onderhoudsproblemen verdwijnen geheel. De reden daarvoor is dat alle besproken vormen van onderhoud in het data dictionary model resulteren in wijzigingen in de inhoud van de tabellen in plaats van in de structuur ervan. Omdat de programmatuur uitgaat van de actuele status van de dictionary, werken veranderingen in de dictionary direct door in de programma's. Wij concluderen dat het data dictionary model leidt tot een hoger niveau van logische gegevensafhankelijkheid. Minstens zo interessant is dat de software in zeer hoge mate *herbruikbaar* wordt. De besproken SQL-queries voor het consolidatiesysteem zijn met kleine wijzigingen over te nemen naar andere generalisaties.

6 Het data dictionary model als praktisch en theoretisch alternatief

In de voorgaande twee paragrafen is het data dictionary model gepresenteerd als een praktisch toepasbare oplossing voor een probleem waarmee het relationele model geen raad weet. Dat neemt niet weg dat enkele praktische en theoretische vragen onbeantwoord zijn gebleven. De belangrijkste vraag vanuit de praktijk betreft de performance van een dictionary-implementatie. Een belangrijke vraag uit theoretisch oogpunt is of het data dictionary model in de toekomst een alternatief vormt voor benaderingen als RM/V2 en de objectgeoriënteerde aanpak die generalisatie expliciet ondersteunen. De vraag of de data dictionary benadering ook toepasbaar is op andere modelleringsproblemen komt in paragraaf 7 aan de orde.

¹⁶ In figuur 7B worden ook voor NULL-waarden tupels opgenomen in de tabel VAR_WAARDE. Indien gekozen wordt voor de meer elegante oplossing om deze loze tupels niet op te nemen dan zou gebruik van een OUTER-JOIN gemaakt moeten worden om een vergelijkbare uitkomst te krijgen. Een dergelijke keuze leidt ook tot herformulering van de tweede integriteitsregel in paragraaf 5.1.

6.1 Performance

Het data dictionary model is conceptueel sterker dan het primitieve en het klassieke model¹⁷ en het biedt een oplossing voor problemen die bij de huidige stand van de techniek tot veel programmeerwerk aanleiding geven. Programma's die op het data dictionary model werken bevatten veel minder programmaregels, zijn zeer onderhoudsongevoelig en zijn eenvoudig over te dragen naar andere systemen die een generalisatieproblematiek kennen.

Tegenover deze voordelen staat één groot nadeel: het data dictionary model leidt tot extra geheugengebruik en extra I/O's. Vooral het I/O-probleem kan leiden tot onacceptabel lange responsetijden. De reden daarvoor is dat elk veld wordt weergegeven als een tuple. In het gegeven voorbeeld wordt één BALPOST-tuple in het primitieve model weergegeven door 1 á 5 VAR_WAARDE-tupels in data dictionary model. In de praktijk zijn deze verhoudingen nog ongunstiger. Elk van deze tupels bevat behalve een attribuutwaarde de sleutel van de balanspost en de sleutel van het variabele attribuut.

Een gedeeltelijke oplossing hiervoor ligt in de clustering-technieken die sommige RDBMSen ondersteunen. Clustering maakt het mogelijk om te besparen op het geheugenbeslag van de sleutelattributen en om de gegevens over balansposten fysiek aansluitend op te slaan. Om deze redenen hoeft een implementatie volgens het data dictionary model niet te leiden tot een onacceptabele performance-verslechtering. In het algemeen kan men stellen dat het implementatie conform het data dictionary model meer de moeite waard wordt naarmate:

- 1- het generalisatieprobleem sterker speelt;
- 2- de verhouding tussen arbeidskosten (programmeurs) en kapitaalkosten (hardware en systeemsoftware) hoger is;
- 3- de database minder omvangrijk is;
- 4- de informatiebehoeften waarin het systeem voorziet meer gericht zijn op informatievoorziening voor het management dan op transactieverwerking;
- 5- de gebruikers minder goed weten wat zij willen (prototyping).

Onze conclusie is dat het in veel gevallen de moeite waard is om systemen die gekenmerkt worden door hoge onderhoudskosten nader te bekijken vanuit de optiek van het data dictionary model. Daarnaast is het zeer wel denkbaar dat deze benadering de toepassing van automatisering economisch haalbaar maakt op terreinen waar dat nu nog niet het geval is.

Al met al is de voorlopige conclusie dat het beschreven data dictionary model alleen een goed alternatief is in omgevingen die in hoge mate voldoen aan de genoemde vijf kenmerken. Naar onze mening mag dit echter geen eindconclusie zijn. Het data dictionary model is een *conceptueel* model. Het is mogelijk om het model één-op-één te implementeren, maar dat is niet de enig denkbare oplossing. Als performance belangrijk is dan is het mogelijk om het data dictionary model te combineren met het primitieve of met het klassieke model. In het geval van een combinatie met het primitieve model vervangt de tabel BALPOST van figuur 2 de tabel VAR_WAARDE van figuur 7B. We introduceren daarmee een stuk beheerste redundantie. De inhoud van de dictionary bevat immers alle informatie die terug te vinden is in de tabelstructuren van het primitieve of het klassieke model. Deze redundantie leidt ertoe dat wijzigingen met DML in de dictionary nu moeten leiden tot wijzigingen met DDL in de structuur van de BALPOST-tabel. Omdat deze structuur redundant is kunnen de benodigde ALTER TABLE statements geheel automatisch worden doorgevoerd.

Een ander gevolg van de redundante oplossing is dat de onderhoudsongevoeligheid van SQL-statements verdwijnt. Wijzigingen in de dictionary leiden niet alleen tot wijzigingen in de database-structuur maar ook tot wijzigingen in de programma's en in de layout van schermen en lijsten. De dictionary bevat echter voldoende informatie om SQL-statements op het data dictionary model automatisch te converteren in SQL-statements op het primitieve of op het klassieke model [VELD90]. Het is met andere woorden mogelijk om een precompiler-programma te schrijven dat het SQL-statement van figuur 10 converteert naar het SQL-statement van figuur 5B of 5C. Op dezelfde wijze is het mogelijk om vanuit de dictionary

¹⁷ Deze stelling is aannemelijk gemaakt door de besproken voorbeelden. Verificatie is mogelijk door algoritmen op te stellen die het ene model in het andere transformeren. Transformatie van het data dictionary model naar het primitieve of het klassieke model blijkt mogelijk, in tegenstelling tot alle andere transformaties.

scherm- en lijst-layouts aan te passen¹⁸.

Resumerend kunnen we stellen dat de performance-problemen volledig op te lossen zijn. In het meest ongunstige geval resulteert dit in het additioneel realiseren van programma's die:

- 1- vanuit de dictionary de database (her)genereren;
- 2- programma's geschreven op de dictionary (inclusief de VAR_WAARDE-tabel) omzetten in programma's op de vanuit de dictionary gegenereerde tabellen;
- 3- scherm- en lijstlayouts genereren.

Naar onze mening zijn soortgelijke implementaties nu al de moeite waard voor grotere systemen of voor programma-pakketten die voor de markt worden gemaakt. De overhead aan generatieprogramma's beperkt de voordelen van het data dictionary model bij de systeembouw, maar handhaaft alle voordelen bij systeemonderhoud. Bovendien hoeven de deze utility-programma's maar één keer te worden ontwikkeld om daarna algemeen bruikbaar te zijn.

6.2 Het data dictionary model en het relationele kader

De door ons voorgestelde oplossing van generalisatieproblemen past binnen het relationele kader en is dus te implementeren met behulp van de huidige RDBMSen. Daarmee is niet gezegd dat alle inspanning om generalisatie als zodanig op te nemen in een alternatief relationeel model, zoals RM/V2, of in een niet relationeel model, zoals het binaire of een object-georiënteerd model, nutteloos is. Generalisatie is net als "functionele afhankelijkheid", "entiteits-integriteit" en "referentiële integriteit" een basisconcept voor systeemontwerp. Smith en Smith merken reeds op dat dit concept binnen de administratieve automatisering vrijwel verwaarloosd wordt, terwijl het binnen de kunstmatige intelligentie een kernbegrip is. De conclusie is dat generalisatie net zo behandeld moet worden als deze andere begrippen, namelijk als basiselement (invariant) van het relationele model. Dit betekent ook dat DBMSen dit concept direct moeten ondersteunen zodat applicatie-ontwikkelaars het wiel niet telkens opnieuw uit hoeven te vinden.

Zo beschouwd is de door ons voorgestelde oplossing nogal ad hoc. Dat blijkt niet alleen uit het feit dat een performante implementatie aanleiding geeft tot het bouwen van generatoren, maar ook uit het feit dat de generalisatieproblematiek niet als zodanig is te herkennen. In het consolidatiesysteem speelt het generalisatieprobleem alleen voor het entiteitstype BALPOST. Indien het generalisatieprobleem ook zou spelen voor andere entiteitstypen dan zou dit telkens weer aanleiding geven tot een kopie van de dictionary-structuur¹⁹.

Onze conclusie is dat het inderdaad nodig is om het relationele model uit te breiden met concepten om generalisatie als zodanig op te nemen. Het is mogelijk om het data dictionary model zodanig te integreren in het relationele model dat alle besproken theoretische problemen overwonnen worden. Een beschrijving van de manier waarop dit kan gebeuren valt buiten het kader van dit artikel.

7 Verdergaande perspectieven

Uitgangspunt voor dit artikel vormde de constatering dat de technische vooruitgang niet geleid heeft tot een wezenlijke vergroting van de inherente flexibiliteit van geautomatiseerde systemen. Wij hebben laten zien dat de oorzaak van een categorie van problemen die aanleiding geeft tot starheid en daarmee tot hoge onderhoudskosten kan worden teruggevoerd op tekortkomingen van het relationele model. Het blijkt mogelijk te zijn om de besproken problemen binnen een relationeel kader uit de wereld te helpen, ook al laat de hier besproken oplossing nog het één en ander te wensen over.

¹⁸ De dictionary zal dan wel uitgebreid moeten worden met informatie over de positie van de attributen op de schermen. Een dergelijke uitbreiding valt buiten het kader van dit artikel, maar is niet moeilijk voor te stellen. Het resultaat is een nog grotere flexibiliteit. Scherm- en lijstlayouts kunnen worden aangepast zonder programmawijzigingen.

¹⁹ De constraints van paragraaf 5.1 kunnen daardoor niet door een RDBMS worden "herkend". Een van de belangrijkste taken van een DBMS is het afdwingen onveranderlijke regels.

Het belangrijkste kenmerk van de voorgestelde oplossing is het laten vallen van het harde onderscheid tussen tabelstructuur en tabelinhoud. Anders gezegd: de informatie over de structuur van het model wordt niet meer weggestopt in de structuur van de database of (erger nog) in de programma's maar wordt expliciet weergegeven als inhoud van een aantal dictionary tabellen. Het resultaat daarvan is een primitieve vorm van *zelfkennis* en deze zelfkennis ligt ten grondslag aan de flexibiliteit en de algemene toepasbaarheid van de gepresenteerde oplossing.

Ondanks dit alles hebben wij slechts voor één van de problemen waarmee het relationele model moeite heeft een oplossing geboden. Zelfs ten aanzien van de door ons besproken vorm van generalisatie kan worden gesteld dat het een speciaal (hoewel algemeen voorkomend) geval betreft. Eenzelfde opmerking kan gemaakt worden voor systeemonderhoud in het algemeen. De belangrijkste vraag is daarmee deze: "is een uitgebreide vorm van het data dictionary model krachtig genoeg om andere modelleringsconcepten expliciet te ondersteunen?". Indien dit het geval is dan biedt het model een alternatief voor de genoemde modellen. Het grote voordeel van het data dictionary model boven deze andere modellen ligt dan in de zeer hoge graad van logische gegevensafhankelijkheid en daarmee van inherente flexibiliteit die het model biedt.

Geraadpleegde literatuur:

- [BEKK88] Bekke J H ter, *Database Ontwerp*, Stenfert Kroese, Leiden, 1988
- [BEME87] Bemelmans T M A, *Bestuurlijke informatiesystemen en automatisering*, Stenfert Kroese, Leiden, 1987
- [BRAC83] Brachman J B, *What IS-A Is and Isn't: An Analysis of Taxonomic Links in Semantic Networks*, Computer, oktober 1983
- [CODD79] Codd, E F, *Extending the Database Relational Model to Capture More Meaning*, ACM Transactions on Database Systems 4, no. 4, december 1979
- [CODD90] Codd E F, *The Relational Model for Database Management: Version 2*, Addison-Wesley, Massachusetts, 1990
- [DATE89] Date C J, *Null Values in Database Management*, in "C J Date, Relational Database: Selected Writings", Addison-Wesley, Massachusetts, 1989
- [DATE90a] Date C J, *An introduction to database systems, volume I*, Addison-Wesley, Massachusetts, 1990
- [KRAM87] Kramer N J T A en Smit J de, *Systeemdenken*, Stenfert Kroese, Leiden, 1987
- [LAAG87] Laagland P T M, *Gegevensmodellen*, Kluwer, Deventer, 1987
- [LEK89] Lek H van der en Buitendijk R B, *Is 'CREATE TABLE' nodig ?*, Informatie, No 10, 1989, pp 721-804
- [MEYE88] Meyer B, *Object-oriented Software Construction*, Prentice Hall, New York, 1988
- [NIJS89] Nijssen G M, *Grondslagen van Bestuurlijke InformatieSystemen*, Nijssen Adviesbureau voor Informatica, Sittard, 1989
- [SIJT88] Sijtsma G L en Oud L C M, *Invloed van geautomatiseerde methoden en technieken op de flexibiliteit van informatiesystemen*, Informatie, No 2, 1988, pp 130-140
- [SMIT77] Smith J M en Smith D C P, *Database abstractions: Aggregation and Generalization*, ACM Transactions on Database Systems, Vol. 2, No. 2, Juni 1977, pp 105-133
- [VELD90] Veldwijk R J, Boogaard M, Dijk M V van, Spoor E R K, *EDSOs, implosion and explosion: concepts to automate a part of application maintenance*, Researchmemorandum 1990-25 van de Faculteit Der Economische Wetenschappen en Econometrie van de Vrije Universiteit Amsterdam (verschijnt in juni 1991 in "Information and Software Technology")
- [VERH90] Verhelst M, *Hoe systeemveroudering te vermijden in de jaren '90*, Syllabus NOVI-conferentie "Systeemveroudering: begeleiden of vermijden?" d.d. 3 oktober 1990

Inleiding.

Het MESDAG-project is een gezamenlijk project van de N.V. Nederlandse Spoorwegen, RAET N.V. en de Vrije Universiteit Amsterdam. Het MESDAG-project is in september 1990 gestart naar aanleiding van een onderzoek naar en implementatie van actieve data dictionary modellen in de tweede helft van 1989 bij RAET N.V.

De naam MESDAG is een afkorting van:

MEta Systems Design And Generation

Doelstellingen.

De doelstelling van het MESDAG-project is: "Het ontwikkelen van flexibele informatiesystemen".

Daarnaast heeft het MESDAG-project de volgende twee subdoelstellingen:

1. Onderzoek naar en implementatie van een actief, zelf-referentieel data dictionary model waarin zowel een beschrijving van de database-gegevens (meta-database-data) als een beschrijving van alle specificeerbare gegevens van het ontwerp van de applicaties (meta-applicatie-data) kunnen worden opgenomen. Dit data dictionary model dient voldoende semantische aspecten (zoals domeinen, constraints en tijdsaspecten) te bezitten voor het waarborgen van de integriteit, consistentie en validiteit van de (meta)data.
2. Onderzoek naar en implementatie van de mogelijkheden van data dictionaries in het algemeen en het, onder punt 1 genoemde, data dictionary model in het bijzonder, ten aanzien van de inbedding in en ontwikkeling van methoden, technieken, methodologische richtlijnen en geautomatiseerde hulpmiddelen voor het ontwerpen, implementeren en onderhouden van flexibele informatiesystemen.

Leden van de MESDAG-onderzoeksgroep.

1. Dr. E.R.K. Spoor.

Dr. E.R.K. Spoor is als universitair hoofddocent verbonden aan de Vrije Universiteit te Amsterdam. Hij doceert en adviseert op het gebied van de toepassing van gegevensbanken en gegevensmodellerings in organisaties. Gedurende zijn loopbaan in de automatisering was hij onder meer acht jaar verbonden aan NCR Nederland N.V. en zes jaar aan de Vrije Universiteit. Hij is als medeoprichter en commissaris betrokken bij twee automatiseringsbedrijven, te weten PSB en VDA, gevestigd respectievelijk in Amsterdam en Hilversum.

2. Drs. R.J. Veldwijk.

In oktober 1986 als Bestuurlijk Informatiekundige afgestudeerd aan de Vrije Universiteit Amsterdam en als project adviseur werkzaam bij RAET N.V. te Utrecht.

3. Drs. M. Boogaard.

In augustus 1990 als Bestuurlijk Informatiekundige afgestudeerd aan de Vrije Universiteit Amsterdam en sinds september 1990 als Assistent In Opleiding verbonden aan de Faculteit der Economische Wetenschappen en Econometrie, Vakgroep Bestuurlijke Informatiekunde van de Vrije Universiteit Amsterdam. Daarnaast is hij part-time werkzaam voor de N.V. Nederlandse Spoorwegen.

4. Drs. M.V. van Dijk.

In september 1990 als Bestuurlijk Informatiekundige afgestudeerd aan de Vrije Universiteit Amsterdam en sinds oktober 1990 als Assistent In Opleiding verbonden aan de Faculteit der Economische Wetenschappen en Econometrie, Vakgroep Bestuurlijke Informatiekunde van de Vrije Universiteit Amsterdam. Daarnaast is hij part-time werkzaam voor de N.V. Nederlandse Spoorwegen.

Correspondentie-adres.

Vrije Universiteit
Faculteit: FEWEC/BIK
De Boelelaan 1105
1081 HV Amsterdam

Tel.: 020-5487081
Fax.: 020-462645