Autorisatieprincipes in het gegevensdomein (2 en slot)

Modelleren en horizontaal programmeren

Frido van Orden

Autorisatieregels lijken op validatieregels, maar helaas niet genoeg. Met de kennis die we daarover hebben opgedaan (zie het vorige nummer van DB/M) bouwen we een gegevensmodel. En we sluiten dit tweeluik af met een onverwacht theoretische exercitie over het fenomeen horizontale autorisatie.

Beginnende met een gegevensmodel voor autorisatiemanagement dient direct een oude belofte te worden ingelost: in het vorige artikel heb ik u beloofd nader in te gaan op het gebruiker/rol-model zoals dat in de SQL wereld wordt gehanteerd voor modellering van gebruikers en gebruikersgroepen.

ROLLENSPEL

In SQL is een rol niets anders dan een verzameling rechten, bijvoorbeeld het recht een bepaalde tabel te raadplegen of de database te starten en te stoppen.
'Gebruiker' is een identificatie van de persoon die of het programma dat de database benadert. Een gebruiker heeft een vaste rol die hem een aantal standaard rechten verschaft, bijvoorbeeld het recht gegevens te raadplegen en te muteren in tabellen waarvan hijzelf de eigenaar is. Daarnaast kan hij worden geautoriseerd voor een bepaalde rol, door middel van het SQL-statement:

```
GRANT TheRole TO TheUser
```

Bijzonder aan dit statement is dat de gebruiker na uitvoering ervan niet direct de rechten heeft die aan de rol zijn toegekend. Daarvoor dient in een sessie eerst het statement:

```
SET ROLE TheRole
```

te worden uitgevoerd. Merk hierbij op dat het niet mogelijk is op enig moment meer dan één rol te activeren. Dit levert problemen op indien bij de inrichting van rollen wordt gekozen voor fijn-granulaire rollen die elk een beperkt aantal rechten verschaffen, bijvoorbeeld het raadplegen van adresgegevens of het muteren van budgetgegevens. Een relatief eenvoudige businesstransactie raakt echter al snel meer dan één rol, waardoor driftig met SET ROLE-statements moet worden gestrooid. Echt vervelend wordt het pas als men selecties moet maken waarvoor de rechten van verschillende rollen nodig zijn. De enige oplossing ligt dan in de definitie van een samengestelde rol die de rechten van meerdere rollen in zich verenigt. En de enige rechtvaardiging voor een dergelijke kunstgreep zou kunnen zijn dat het recht op het gebruik van zo'n speciale samengestelde rol is voorbehouden aan een bepaald gedeelte van de businesslogica[1].

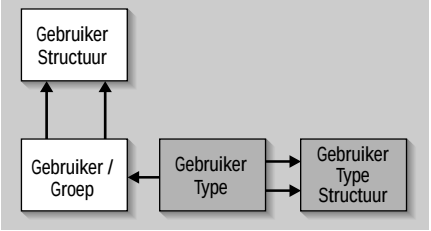
Maar SQL richt zich puur op gegevens, niet op functionaliteit. De lasten zonder lusten van het rollenmodel, gecombineerd met de problematiek van identificatie van de 'echte' gebruiker zoals die in het vorige artikel is besproken; de gebruiker waarmee de database wordt benaderd is niet dezelfde als de daadwerkelijke applicatiegebruiker. Dit is dan ook een van de belangrijkste oorzaken dat autorisatie in het dbms in de praktijk weinig wordt gebruikt.

Is er een uitweg mogelijk uit deze klucht van rollenspellen? Ja, namelijk door als relationelen voor even leentjebuur te spelen bij onze objectgeoriënteerde broeders en het begrip 'overerving' toe te passen. Het idee hierbij is dat de aan een gebruikersgroep toegekende rechten automatisch gelden voor de gebruikers en/of gebruikersgroepen die 'eronder hangen'. Dit laat zich vangen in het gegevensmodel van figuur 1.

Net zoals bij eerdere gegevensmodellen in deze serie zijn de 'grijze' tabellen weer tabellen met meta-metagegevens. In GEBRUIKER_TYPE leggen we de soorten gebruikers vast die we onderkennen. Voorlopig zijn dit er twee: Groep en Gebruiker. Merk op dat de term 'Gebruiker' hier voor tweeërlei uitleg vatbaar is; strikt

Het dbms voorbij (6)

In het vorige artikel is een begin gemaakt met de behandeling van de verschillende vormen van autorisatieregels die zich voordoen in het gegevensdomein. Frido van Orden liet zien dat de regels voor autorisatie en validatie onvoldoende op elkaar lijken. Vervolgens heeft hij verschillende vormen van autorisatie in detail de revue laten passeren.
In dit tweede en laatste artikel over autorisatie balt de auteur de vergaarde kennis samen in een gegevensmodel. Hij sluit af met een beschouwing over horizontale autorisatie.



FIGUUR 1: GEBRUIKERS EN GEBRUIKERSGROEPEN.

genomen ware 'AUTORISATIE_SUBJECT' een betere term voor wat in dit gegevensmodel 'GEBRUIKER' heet. In GEBRUIKER_TYPE_STRUCTUUR (tabel 3 en 4) leggen we vast op welke wijze deze gebruikerstypen mogen worden gerelateerd.
De voor de hand liggende vulling is dat Groepen onder Groepen mogen hangen en Gebruikers ook onder groepen (maar niet andersom). In de tabel GEBRUIKER (tabel 5) leggen we de gebruikers en gebruikersgroepen vast die we onderkennen.
In GEBRUIKER_STRUCTUUR (tabel 6) ten slotte leggen we de relaties tussen gebruikers en gebruikersgroepen vast. Natuurlijk is er weer een user defined constraint (hier niet verder uitgewerkt), die afdwingt dat de vulling van deze tabel overeenstemt met wat we hebben vastgelegd in GEBRUIKER_TYPE_STRUCTUUR. Merk op dat we dit hadden kunnen afdwingen door middel van een foreign key, als de primary key van GEBRUIKER ook het attribuut Gebruiker_Type omvatte.

GEBRUIKER_TYPE			
Gebruiker_Type	Varchar(10)	Not Null	— Gebruiker type code
Omschrijving	Long	Not Null	— Omschrijving (proza)

GEBRUIKER_TYPE	**OMSCHRIJVING**
GROEP	Gebruikersgroep
GEBRUIKER	Gebruiker

TABEL 1 EN 2: META-METAGEGEVENS OVER CATEGORIEËN GEBRUIKERS.

GEBRUIKER_TYPE_STRUCTUUR			
Gebruiker_Type_Hoog	Varchar(10)	Not Null	— Gebruiker type code hoog
Gebruiker_Type_Laag	Varchar(10)	Not Null	— Gebruiker type code laag

GEBRUIKER_TYPE_HOOG	**GEBRUIKER_TYPE_LAAG**
GROEP	GROEP
GROEP	GEBRUIKER

TABEL 3 EN 4: VASTLEGGING VAN DE STRUCTUUR VAN HET GEBRUIKERSTYPE.

GEBRUIKER			
Gebruiker_Naam	Varchar(30)	Not Null	— Gebruikernaam
Gebruiker_Type	Varchar(10)	Not Null	— Gebruiker type code
Omschrijving	Long	Not Null	— Omschrijving (proza)

TABEL 5.

GEBRUIKER_STRUCTUUR			
Gebruiker_Naam_Hoog	Varchar(30)	Not Null	— Gebruikernaam hoog (groep)
Gebruiker_Naam_Laag	Varchar(30)	Not Null	— Gebruikernaam laag (groep/gebruiker)

TABEL 6.

Brengen we nog even de verschillende vormen van autorisatie in gedachten die zijn besproken in het vorige artikel (zie tabel 7).

FLASHBACK

Voor het modelleren van autorisatievorm 1 (verticale gebruiker-gegevensautorisatie) zijn we met zojuist beschreven gebruikersmodel en een aantal onderdelen van het gegevensmodel voor validatie[2] al een aardig eind op weg. We herhalen hier eenvoudigweg de tabeldefinities voor de entiteiten TABEL, ATTRIBUUT en DML_TYPE (tabellen 8, 9 en 10), met dien verstande dat het attribuut DML uit laatstgenoemde tabel (tabel 10) in het kader van autorisatie ook de waarde 'Select' moet kunnen bevatten. Nu hebben we nog een tabel nodig die TABEL/ATTRIBUUT (samengevat onder de term DATABASE_OBJECT), DML_TYPE en GEBRUIKER aan elkaar koppelt en klaar is ons volgende stuk gegevensmodel. Deze tabel noemen we VERTICALE_GEGEVENS_AUTORISATIE en ziet eruit als tabel 11. De kolom Attribuut_Naam is optioneel; voor de DML-acties Insert en Delete is dit attribuut immers niet relevant. Daarom moeten we aan de tabel een surrogaat primaire-sleutelattribuut-ID toevoegen. De attributen Tabel_Naam, Attribuut_Naam, DML en Gebruiker_Naam vormen samen een alternatieve sleutel. De user defined constraint die afdwingt dat Attribuut_Naam is ingevuld indien DML = Update of Select en anders leeg is, had u zelf natuurlijk ook direct al bedacht.

Minder voor de hand liggend is het attribuut Ind_Auth, dat op het eerste gezicht redundant is: als voor zekere tabel/attribuut/dml/gebruiker combinatie een tupel in VERTICALE_GEGEVENS_AUTORISATIE is te vinden, betekent dit dat de betrokken gebruiker rechten heeft de betreffende DML-actie op betreffende tabel/attribuut uit te voeren, anders niet. Toch heeft de toevoeging van dit attribuut een doel, namelijk dat we een hoger niveau overerfde rechten weer teniet kunnen doen, zonder ergens in de autorisatiehiërarchie een groep te hoeven toevoegen

met als vrije definitie 'alles van groep G1 behalve dit en dat'.

Objectgeoriënteerde puristen zullen deze vorm van overerving volstrekt verderfelijk vinden. Daar valt tegenin te brengen dat de semantiek van de gegevens in VERTICALE_GEGEVENS_AUTORISATIE volstrekt eenduidig blijft, als we deze als volgt definiëren:

- Gebruiker/groep G heeft rechten tot het uitvoeren van DML-actie D op tabel/attribuut O indien in VERTICALE_GEGEVENS_AUTORISATIE een relevant tupel voorkomt met Ind_Auth = true;
- Gebruiker/groep G1 heeft rechten tot het uitvoeren van DML-actie D op tabel/attribuut O indien in VERTICALE_GEGEVENS_AUTORISATIE een relevant tupel voor groep G2 voorkomt met Ind_Auth = true en G1 in GEBRUIKER_STRUCTUUR direct of indirect overerft van G2;
- Gebruiker/groep G1 heeft géén rechten tot het uitvoeren van DML-actie D op tabel/attribuut O indien in VERTICALE_GEGEVENS_AUTORISATIE een relevant tupel voor groep G2 voorkomt met Ind_Auth = false en G1 in GEBRUIKER_STRUCTUUR direct of indirect overerft van G2, tenzij zich tevens de situatie voordoet als hierboven vermeld;
- Gebruiker/groep G heeft géén rechten tot het uitvoeren van DML-actie D op tabel/attribuut O indien in VERTICALE_GEGEVENS_AUTORISATIE een relevant tupel voorkomt met Ind_Auth = false, onafhankelijk van enig ander tupel.

Kort samengevat komt deze semantiek erop neer dat een negatieve autorisatie altijd sterker geldt dan een van een *hoger* niveau overerfde positieve autorisatie en altijd zwakker geldt dan een van *hetzelfde* niveau overerfde positieve autorisatie.

Nu we de verticale gebruiker-gegevensautorisatie op poten hebben staan, is autorisatievorm 3 (verticale functie-gegevensautorisatie: wat doet een functie met de database?) een peulenschil. Wat we natuurlijk nodig hebben, is een entiteit waarin de verschillende functies zijn vastgelegd. Daar kunnen we ons voor dit doel eenvoudig vanaf maken met tabel 12. De functie-gegevensautorisatie zelf kunnen we dan kwijt in een entiteit die we kortweg CRUD noemen (tabel 13) en die als twee druppels water lijkt op VERTICALE_GEGEVENS_AUTORISATIE (tabel 11).

Het enige attribuut dat hier nadere toelichting verdient is Ind_Auth_Nodig. Dit attribuut kan ons van pas komen als we de gebruiker-gegevensautorisatie en de functie-gegevensautorisatie aan elkaar willen relateren. De sleutel daartoe is gelegen in autorisatievorm 5: de aloude gebruiker-functieautorisatie. Deze kunnen we simpelweg vangen in de entiteit FUNCTIONELE_AUTORISATIE (tabel 14). In essentie is dit niet meer dan een koppeltabel tussen GEBRUIKER en FUNCTIE, gecombineerd met het Ind_Auth attribuut dat we al kennen uit VERTICALE_GEGEVENS_AUTORISATIE. Figuur 2 geeft het gegevensmodel weer dat nu is ontstaan.

Ogenschijnlijk is dit een model zonder veel poespas: een simpele registratie waarin we vastleggen wat we in ons systeem wel en niet toelaten. Toch zijn met dit een-

	ACTOR	ACTIE	OBJECT
1	Medewerkers P&O	Raadplegen	Salaris van een medewerker
2	Gebruiker 'Storm'	Muteren	Budget afdeling 'Debiteuren/crediteuren'
3	Functie 'Raadplegen Afdelingen'	Raadplegen	Attributen van afdelingen
4	Functie 'Muteren medewerkers'	Muteren	Medewerkers behalve de directie
5	Gebruiker 'Orden'	Uitvoeren	Functie 'Raadplegen Afdelingen'

Tabel 7: Vormen van autorisatie.

TABEL:				
Tabel_Naam	Varchar(30)	Not Null	—	Tabelnaam
Omschrijving	Long	Not Null	—	Tabel omschrijving (proza)
Tabel_Type	Char(1)	Not Null	—	Base table of View
...	...	...	—	...

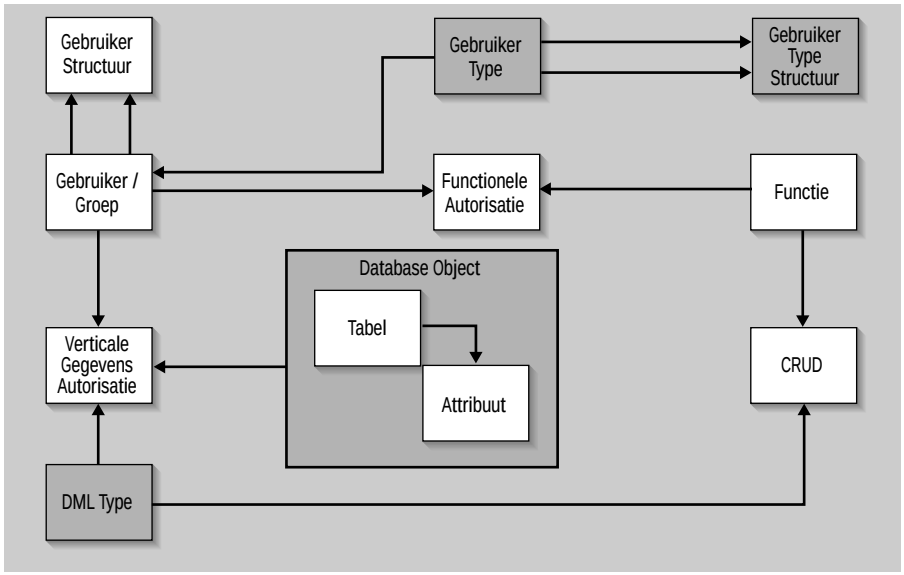
ATTRIBUUT:				
Tabel_Naam	Varchar(30)	Not Null	—	Tabelnaam
Attribuut_Naam	Varchar(30)	Not Null	—	Attribuutnaam
Omschrijving	Long	Not Null	—	Attribuut omschrijving (proza)
Domein_Naam	Varchar(30)	Not Null	—	Referentie naar domein
Ind_Verplicht	Char(1)	Not Null	—	Null allowed of niet?
...	...	...	—	...

DML_TYPE:				
DML	Char(1)	Not Null	—	Insert, Update, Delete of Select actie
Omschrijving	Varchar(10)	Not Null	—	DML omschrijving (proza)

Tabel 8, 9 en 10: Tabeldefinities voor de entiteiten TABEL, ATTRIBUUT en DML_TYPE.

ID	Autonumber	Not Null	—	ID
Tabel_Naam	Varchar(30)	Not Null	—	Tabelnaam
Attribuut_Naam	Varchar(30)	Null	—	Attribuutnaam
DML	Char(1)	Not Null	—	Insert, Update, Delete of Select actie
Gebruiker_Naam	Varchar(30)	Not Null	—	Gebruikernaam
Ind_Auth	Boolean	Not Null	—	Indicatie geautoriseerd of niet

Tabel 11: Verticale gegevensautorisatie.



Figuur 2: Verticale autorisatievormen.

voudige model al een hoop leuke dingen mogelijk. Wat te denken van het combineren van de gebruiker-functieautorisatie met de CRUD? Als we weten wie welke functie mag uitvoeren en wat die functie met de database doet, is het niet moeilijk de gebruiker-gegevensautorisatie -een ondergeschoven kindje in menig systeem- door middel van een enkel SQL-statement precies op maat te bepalen:

```
  INSERT INTO verticale_gegevens_autorisatie
 (Tabel_Naam, Attribuut_Naam, DML, Gebruiker_Naam, Ind_Auth)
  SELECT c.Tabel_Naam, c.Attribuut_Naam, c.DML, f.Gebruiker_Naam, true
  FROM crud c, functionele_autorisatie f
  WHERE c.Functie_Naam = f.Functie_Naam
```

Stel nu dat we deze vulling inderdaad met een *one shot generator* genereren en er vervolgens aan gaan sleutelen, bijvoorbeeld door het verwijderen van de update-autorisatie van gebruiker Jansen voor het attribuut Salaris in tabel Medewerker. Wat nu te doen met een denkbeeldig scherm 'Onderhouden en raadplegen Medewerkergegevens', dat de mogelijkheid biedt het salaris van een medewerker te muteren? Mag de gebruiker dit scherm nog opstarten? Kunnen we het scherm 'Onderhouden en raadplegen Medewerkergegevens' nog steeds als zodanig beschouwen, ook als het veld Salaris niet muteerbaar is?

De meeste lezers zijn vermoedelijk van mening dat een flexibel systeem de gebruiker gewoon toegang zou moeten bieden tot het gewraakte scherm en het veld Salaris read-only zou moeten maken. Maar wat nu indien de naam van de medewerker niet meer mag worden getoond, omdat de gebruiker daarop geen Select-privilege meer heeft? Of erger nog, de code van de medewerker die als primaire sleutel fungeert in de tabel Medewerker?

Technisch gezien kan veel: het niet mogen zien van de naam van de medewerker is geen enkel probleem, het niet mogen zien van de primaire sleutel leidt in het ergste geval tot een scherm waarin helemaal niet meer gemuteerd kan worden - al is in dat geval de naam '*Onderhouden* en raadplegen' discutabel geworden. Maar in de praktijk zal men een scherm waarin salarissen kunnen worden gewijzigd niet acceptabel vinden zonder visuele controle op de juistheid van de ingetoetste medewerkercode, door middel van naamcontrole.

Zie hier het nut van het attribuut Ind_Auth_Nodig in de CRUD-tabel: met dit attribuut kan worden aangegeven of het noodzakelijk is dat de gebruiker bepaalde rechten heeft voordat hem toegang tot de functie wordt geboden -al heeft hij dat toegangsrecht volgens de gebruiker-functieautorisatie! Door middel van Ind_Auth_Nodig definieert de applicatiebeheerder dus eigenlijk wat de essentie van een functie is: een scherm 'Onderhouden en raadplegen Medewerkergegevens' hoeft geen salarissen te kunnen muteren, maar moet wel de code en naam van de medewerker tonen. Dit betekent feitelijk dat geen enkel beletsel hoeft te bestaan om 'plat administratieve' schermen een ruim-

FUNCTIE:

Functie_Naam	Varchar(30)	Not Null	— Functienaam
Omschrijving	Long	Not Null	— Functie omschrijving (proza)

Tabel 12: Vastlegging verschillende functies.

CRUD:

ID	Autonumber	Not Null	— ID
Tabel_Naam	Varchar(30)	Not Null	— Tabelnaam
Attribuut_Naam	Varchar(30)	Null	— Attribuutnaam
DML	Char(1)	Not Null	— Insert, Update, Delete of Select actie
Functie_Naam	Varchar(30)	Not Null	— Functienaam
Ind_Auth_Nodig	Boolean	Not Null	— Indicatie autorisatie nodig of niet

Tabel 13.

FUNCTIONELE_AUTORISATIE

Functie_Naam	Varchar(30)	Not Null	— Functienaam
Gebruiker_Naam	Varchar(30)	Not Null	— Gebruikernaam
Ind_Auth	Boolean	Not Null	— Indicatie geautoriseerd of niet

Tabel 14.

hartige gebruiker-functieautorisatie te geven: de wal keert het schip hier toch wel. Voor batchachtige programmatuur ligt dat natuurlijk heel anders.

HORIZONTAAL

Wat nog ontbreekt aan ons model is de ondersteuning voor de beide horizontale vormen van autorisatie. Konden we de tot nu toe besproken autorisatievormen relatief makkelijk in een gegevensmodel gieten, bij horizontale autorisatie neemt het aantal vrijheidsgraden enorm toe. Onze voorbeeldregel 'Gebruiker Storm mag het Budget van de Afdeling 'Debiteuren/Crediteuren' wijzigen' is nog relatief eenvoudig, maar het kan natuurlijk veel ingewikkelder.

Wellicht komt deze situatie u nog bekend voor van de bespreking van user defined constraints in de artikelen over validatie eerder in deze serie. U herinnert zich dan ook dat deze constraints in het gegevensmodel voor validatie zijn opgenomen via een attribuut dat een SQL-expressie kan bevatten. Deemoedig is toen erkend dat dit een flagrante schending is van het eerste der *Tien geboden voor goed database-ontwerp*. Alternatieven waren en zijn echter niet voorhanden. Weliswaar bestaan diverse frequent voorkomende prototypen van user defined constraints, bijvoorbeeld 'indien attribuut A de waarde X bevat, moet attribuut B verplicht een waarde hebben'. Maar bij enig doordenken neemt deze verzameling prototypen zo snel toe, dat het einde zoek is.

Met horizontale autorisatie zitten we nu in hetzelfde schuitje. De restrictie die aan te raadplegen of te muteren tupels kan worden opgelegd, wordt in potentie arbitrair complex. En dit betekent dat we in elk geval de mogelijkheid zullen moeten bieden deze restrictie als SQL-expressie in ons gegevensmodel op te nemen. Anders dan bij user defined constraints komen drie prototypen restricties echter bovengemiddeld vaak voor:

- een restrictie aan het waardebereik van een attribuut, bijvoorbeeld 'geslacht = 'Man'' of 'Budget < 1 miljoen';
- een verwijzing naar een restrictie op een tabel waaraan via een verwijzende sleutel wordt gerefereerd, bijvoorbeeld 'Medewerkers werkzaam op een Afdeling met een Budget 1 miljoen';
- samenstellingen van bovenstaande twee, eventueel uitgebreid met een extra 'niet-standaard' restrictie, 'Mannelijke medewerkers werkzaam op een Afdeling met een Budget 1 miljoen die een auto van de zaak rijden'.

Willen we deze prototypen restricties expliciet opnemen in het gegevensmodel, dan hebben we behoefte aan modellen voor het begrip 'waardebereik van een attribuut' en 'verwijzende sleutel'. Die herkent u natuurlijk direct als de constraint-typen 'Domein' en 'Foreign key' uit het gegevensmodel voor validaties[2] (zie ook figuur 3). Wel hebben we nog een mechanisme nodig om te garanderen dat de restrictie aan het waardebereik van een attribuut ook bestaanbaar is. We moeten garanderen dat de waarden in het gerestricteerde waardebereik een deelverzameling zijn van de waarden in het 'normale' waardenbereik van een attribuut. Dit kunnen we doen door constraints van het type 'Domein' een optionele verwijzing naar zichzelf te geven. Een user defined constraint moet garanderen dat deze verwijzing bestaanbaar is; gegevenstypen zijn gelijk, eventuele waardenbereiken vormen een deelverzameling. De constraint 'Domein Budget < 1 miljoen' (datatype = NUMBER(8,2), waarde > 0) kunnen we nu definieren als een subdomein van constraint 'Domein Budget' (datatype NUMBER(12,2), waarde > 0).

RESTRICTIE:

Tabel_Naam	Varchar(30)	Not Null	— Tabelnaam
Restrictie_Naam	Varchar(30)	Not Null	— Restrictienaam
Omschrijving	Long	Not Null	— Restrictie omschrijving (proza)

TABEL 15.

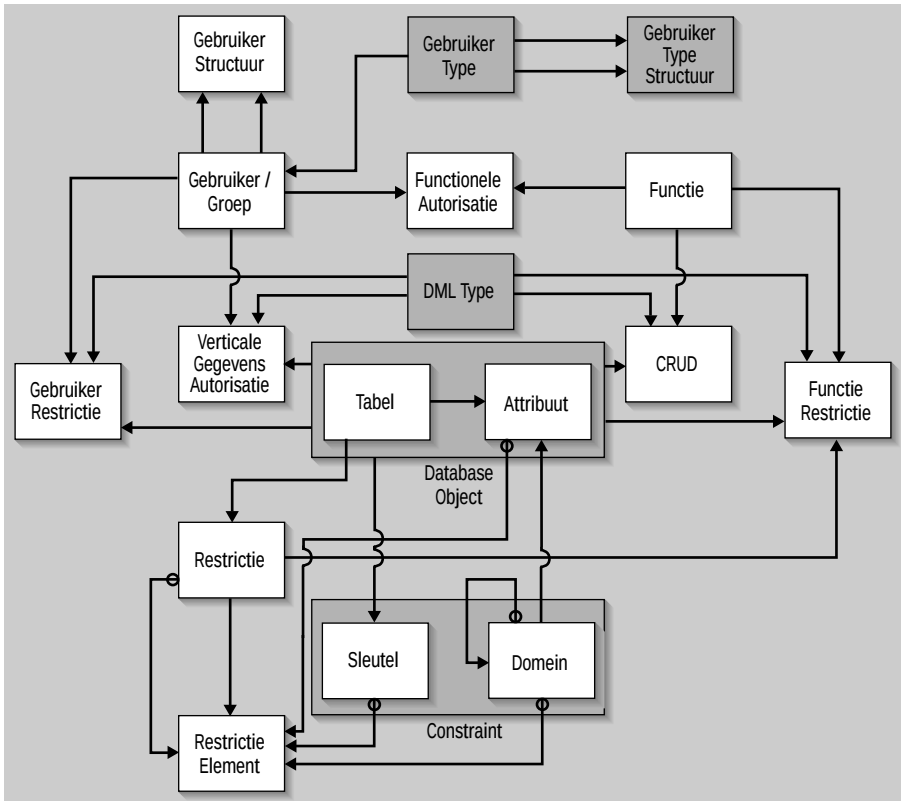
RESTRICTIE_ELEMENT:

Tabel_Naam	Varchar(30)	Not Null	— Tabelnaam
Restrictie_Naam	Varchar(30)	Not Null	— Restrictienaam
Volgnr	Number(2)	Not Null	— Volgnummer
Conditie	Long	Null	— SQL restrictie
Attribuut_Naam	Varchar(30)	Null	— Naam van attribuut met subdomeinrestrictie
Subdomein_Naam	Varchar(30)	Null	— Domein van attribuut met subdomeinrestrictie
Sleutel_Naam	Varchar(30)	Null	— Sleutelnaam voor restrictieverwijzing
Tabel_Naam_Verw	Varchar(30)	Null	— Tabelnaam voor restrictieverwijzing
Restr_Naam_Verw	Varchar(30)	Null	— Restrictie verwijzing

TABEL 16.

TABEL_NAAM	RESTRICTIE_NAAM	OMSCHRIJVING
Afdeling	Budget_Onder_Miljoen	Budget < 1 miljoen Euro
Medewerker	Mannen	Mannelijke medewerkers
Medewerker	50plus	50 plussers
Medewerker	Afd_Budget_Onder_Miljoen	Werkzaam op afdeling met budget < 1 miljoen Euro
Medewerker	Auto_Zaak	Rijdende in een auto van de zaak
Medewerker	Outplacement	Mannelijke medewerkers, ouder dan 50, werkzaam op afdeling met budget < 1 miljoen Euro en toch rijdende in een auto van de zaak

TABEL 17.



FIGUUR 3: OVERALL GEGEVENSMODEL.

Met deze toevoeging aan het constraintmodel kunnen we nu een gegevensmodel voor modellering van de horizontale autorisatierestricties opstellen. In de tabel RESTRICTIE (tabel 15) beschrijven we het bestaan van een restrictie, altijd gekoppeld is aan een tabel. In de tabel RESTRICTIE_ELEMENT (tabel 16) beschrijven we de bouwstenen van de restrictie. Als er meerdere elementen bestaan, is de totale restrictie de logische AND van de individuele elementen.

Een element wordt gedefinieerd door middel van een SQL-restrictie (attribuut Conditie ingevuld, rest leeg), een subdomeinrestrictie op een attribuut (attribuut Attribuut_Naam en Subdomein_Naam ingevuld, rest leeg) of een verwijzing naar een andere restrictie via een foreign key (attribuut Sleutel_Naam, Tabel_Naam_Verw en Restr_Naam_Verw ingevuld, rest leeg). Natuurlijk zijn er ook weer een aantal user defined constraints:

- een restrictie-element bestaat uit hetzij een SQL-conditie of een subdomeinrestrictie of een verwijzing naar een andere restrictie;
- het domein van een subdomeinrestrictie moet een subdomein zijn van het domein van het attribuut;
- de restrictie waarnaar wordt verwezen in een restrictieverwijzing moet zijn gedefinieerd op de tabel waarnaar de foreign key verwijst.

Een vulling van deze tabellen zou eruit kunnen zien als tabel 17 en 18; de definitie van de subdomeinen is niet opgenomen. Nu moeten we de restricties nog integreren met de rest van het gegevensmodel. Voor wat betreft de horizontale gebruiker-gegevensautorisatie doen we dit door middel van de tabel GEBRUIKER_RESTRICTIE (tabel 19). De horizontale functie-gegevensautorisatie lijkt hier weer sprekend op: zie tabel 20. Het geïntegreerde gegevensmodel is weergegeven in figuur 3. De vetcursieve entiteiten zijn overgenomen uit het model voor validatie[2].

EIND GOED AL GOED?

Met het gegevensmodel zijn we aan het eind gekomen van de verhandeling over autorisatie. We zijn eenvoudig begonnen, hebben een voor velen wat minder bekende gedachtengang doorgemaakt rondom horizontale autorisatie, maar zijn er uiteindelijk toch in geslaagd het geheel in een gegevensmodel te gieten dat bij nadere bestudering eenvoudiger in elkaar steekt dan op het eerste gezicht lijkt.

Eind goed al goed, zult u zeggen. Helaas niet, want donkere wolken pakken zich boven ons samen. Het duveltje dat we in huis hebben gehaald luistert naar de naam horizontale selectie-autorisatie. We herhalen nog eens kort ons voorbeeld gegevensmodel dat als leidraad in deze serie dient:

TABEL_NAAM	RESTRICTIE_NAAM	VOLGNR	CONDITIE	ATTRIBUUT_NAAM	SUBDOMEIN_NAAM	SLEUTEL_NAAM
Afdeling	Budget_Onder_Miljoen	1		Budget	BEDR_KL_1MLN	
Medewerker	Mannen	1		Geslacht	GESLACHT_M	
Medewerker	50plus	1		Leeftijd	LEEFT_GR_50	
Medewerker	Afd_Budget_Onder_Miljoen	1				MED_AFD
Medewerker	Auto_Zaak	1	exists (...)			
Medewerker	Pervers	1		Geslacht	GESLACHT_M	
Medewerker	Pervers	2		Leeftijd	LEEFT_GR_50	
Medewerker	Pervers	3				MED_AFD
Medewerker	Pervers	4	exists (...)			

TABEL 18.

```
Medewerker(Med#, Naam, Sexe,
Salaris, Afd#)
Afdeling(Afd#, Naam, Med#_Mgr,
Budget, Afd#_Hoger)
```

Laten we nu eens de restrictie 'Afdelingen met een budget kleiner dan een miljoen euro' van toepassing verklaren op raadpleegacties op die tabel door gebruiker 'Orden'. Het effect hiervan is feitelijk dat elke query op de tabel Afdeling een extra restrictie 'Afdeling.Budget < 1000000' krijgt. Zolang we alleen met de tabel Afdeling te maken hebben, gaat dit prima. Maar wat gebeurt er als we de tabel Medewerker willen raadplegen? Stel dat er een afdeling A1 is met een budget dat hoger is dan 1 miljoen euro. Deze afdeling verschijnt niet in het resultaat van de query:

```
select * from Afdeling where
Afd#='A1'
```

Maar de query:

```
select * from Medewerker where
Afd#='A1'
```

levert mogelijk wel tupels op. Hoe zit dat met de referentiële integriteit? Hebben we zojuist een bom onder een van de grondvesten van het relationele model gelegd?

Ja, feitelijk wel. Willen we het goed doen, dan zullen we de restrictie op het afdelingsbudget ook van toepassing moeten verklaren op Medewerker. In feite is dit de automatische toepassing van het mechanisme van verwijzing naar een restrictie op een tabel via een foreign key dat we zojuist in het gegevensmodel hebben gemodelleerd.

Maar nu is het hek van de dam. Want als we geen medewerkers mogen zien die werken op een afdeling met een budget

TABEL_NAAM_VERW	RESTR_NAAM_VERW
Afdeling	Budget_Onder_Miljoen
Afdeling	Budget_Onder_Miljoen

GEBRUIKER_RESTRICTIE:

Gebruiker_Naam	Varchar(30)	Not Null	— Gebruikernaam
Tabel_Naam	Varchar(30)	Not Null	— Tabelnaam
Attribuut_Naam	Varchar(30)	Null	— Attribuutnaam
DML	Char(1)	Not Null	— Insert, Update, Delete of Select actie
Restrictie_Naam	Varchar(30)	Not Null	— Restricctienaam

GEBRUIKER_RESTRICTIE:

Functie_Naam	Varchar(30)	Not Null	— Functienaam
Tabel_Naam	Varchar(30)	Not Null	— Tabelnaam
Attribuut_Naam	Varchar(30)	Null	— Attribuutnaam
DML	Char(1)	Not Null	— Insert, Update, Delete of Select actie
Restrictie_Naam	Varchar(30)	Not Null	— Restricctienaam

TABEL 19 EN 20: GEBRUIKER-RESTRICTIE EN HORIZONTALE FUNCTIE-GEGEVENSAUTORISATIE.

>= 1 miljoen euro, dan mogen we ook geen afdeling zien die gemanaged worden door medewerkers die werken op een afdeling met een budget >= € 1 mln, en ook geen afdelingen die via-via-via vallen onder een hogere afdeling met een budget van € 1 mln +. Zelfs in ons uiterst eenvoudige voorbeeld gegevensmodel loopt de introductie van een eenvoudige selectierestrictie al volledig uit de hand. En dan te bedenken dat een foreign key slechts een bijzondere variant is van een constraint die op meer dan 1 tupel betrekking heeft. Met evenveel recht zouden we de relaties, geïntroduceerd door een willekeurige user defined constraint die op meer dan 1 tupel betrekking heeft, onderwerp kunnen maken van automatische overerving van restricties.

Moeten we dan maar helemaal afzien van horizontale selectierestricties? Natuurlijk niet! De restricties komen niet uit ons hoofd, maar uit wensen vanuit de praktijk. Uit eigen ervaring blijkt het volgende mechanisme goed te werken. Er vindt geen automatische overerving van restricties via foreign keys of wat dan ook plaats. Wel worden bij query's die vanuit een scherm worden uitgevoerd daarnaast de restricties van de tabellen overgenomen waarnaar direct naar verwezen via een foreign key. Een query op alle afdelingen levert dan effectief de volgende SQL op:

```
select a1.*
from Afdeling a1, Afdeling a2
where a1.Budget < 1000000
      and a1.Afd#_Hoger = a2.Afd#
      and a2.Budget < 1000000
```

EEN VRAAG TOT SLOT

We sluiten af met een vraag voor het slimste jongetje of meisje van de klas. In het gegevensmodel ziet u dat de tabellen GebruikerRestrictie en FunctieRestrictie een koppeling vormen tussen Gebruiker resp. Functie, Database Object en DML Type. Hetzelfde geldt voor de tabellen VerticaleGegevensAutorisatie en CRUD. Waarom is gekozen voor deze modellering en niet voor een rechtstreekse foreign key vanuit GebruikerRestrictie naar Verticale GegevensAutorisatie resp. van Functie Restrictie naar CRUD? Tip: het heeft te maken met een ander theoretisch probleem rond horizontale selectie-autorisatie. ●

Noten:

1. Deze filosofie ligt bijvoorbeeld ten grondslag aan het security-mechanisme van de Java-programmeertaal. Hierbij kan de gebruiker van een softwarecomponent rechten toekennen aan die component. De meest bekende toepassing is het 'sandbox'-model, dat ervoor zorgt dat in een webbrowser draaiende Java-applets niet zomaar allerlei rechten op de client-pc hebben.
2. Zie Het dbms voorbij (3): Naa een model voor constraint management, DB/M 6/2001 (oktober).

Frido van Orden (fridoo@faapartners.com) is partner bij FAA Partners.