Generalisatie als specialiteit

Subdomeinen en verwijzende sleutels

Frido van Orden

In het vorige artikel is het concept van hiërarchieën op domeinniveau, ofwel subdomeinen, al kort aangestipt. Een domein is de relationele term voor het meer algemene concept van een gegevenstype. Het definieert een verzameling toegestane waarden, zoals bijvoorbeeld 'alle karakterreeksen van maximaal 255 posities', 'alle gehele getallen groter of gelijk aan 0', of 'alle waarden X, Y of Z', maar ook 'alle afbeeldingen in het JPEG formaat'.

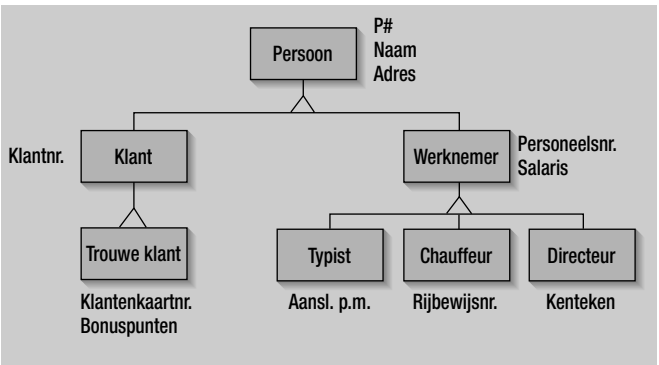
Omdat een domein overeenkomt met een type, is het ook mogelijk om subtypen te definiëren. De fundamentele eigenschap van een subtype S van supertype T zegt dat ieder object van type S tevens van type T is. Met andere woorden: als een bepaalde waarde in de verzameling toegestane waarden, gedefinieerd door subtype S, zit dan dient deze waarde ook in de verzameling toegestane waarden van supertype T te zitten. Het subtype 'alle karakterreeksen van maximaal 25 posities' is dus een subtype van 'alle karakterreeksen van maximaal 100 posities', terwijl 'alle karakterreeksen van maximaal 255 posities' dat niet is.

In gegevensmodellen is het subdomeinconcept op twee manieren van bijzonder belang. Allereerst is het in verwijzende sleutels mogelijk om een attribuut dat deel uitmaakt van een verwijzende sleutel, als domein een subdomein toe te kennen van het domein van het overeenkomstige attribuut in de primaire sleutel, waarnaar wordt verwezen. Neem bijvoorbeeld een gegevensmodel voor een verzekeringsmaatschappij, waarbij de primaire sleutel van de entiteit 'polis' bestaat uit een polistype (opstal, inboedel, auto enzovoort) en een polisnummer. Bepaalde andere entiteiten in het gegevensmodel zijn wellicht alleen van toepassing voor autopolissen. In dat geval kan de verwijzende sleutel vanuit die entiteiten nog steeds worden gedefinieerd over de attributen polistype en polisnummer, maar wordt het gegevenstype van het attribuut polistype in de verwijzende entiteit gewijzigd in een subtype van het algemene polistype. In informele syntax kan dat als volgt beschreven worden:

```
POLIS (POLIS_TYPE : POLIS_TYPE, POLISNR : POLISNR,
...)

POLIS_AUTO (POLIS_TYPE : POLIS_TYPE_AUTO,
POLISNR : POLISNR, ...)
```

Een andere toepassing van subdomeinen wordt gevonden in generalisatiehiërarchieën. Neem bijvoorbeeld het attribuut 'Salaris' van entiteit 'Werknemer' in de inmiddels bekende casus, zie figuur 1. Het gegevenstype van 'Salaris' beperkt de toegestane waarden tot positieve, numerieke waarden met een precisie van maximaal 8 cijfers, waarvan 2 cijfers achter de komma. Stel nu dat we een regel willen opnemen dat typisten een salaris van maximaal € 3000,- mogen verdienen. Feitelijk komt dit erop neer dat de voor Werknemers toegestane waarden voor Salaris in het geval van een Typist worden beperkt tot waarden <= 3000,00. Dit kan door het gegevenstype voor Salaris in de entiteit Typist te herdefiniëren (in OO-termen: *overriden*) met een sub-gegevenstype of subdomein dat het salaris maximeert op € 3000,-. Merk op dat de regel dat



Figuur 1: Objectmodel casus.

Het dbms voorbij (10)

Dit artikel vormt het laatste deel in het drieluik over de generalisatieproblematiek. Het concept 'subdomeinen', dat de vorige keer al kort werd geïntroduceerd, wordt nader onder de loep genomen. Frido van Orden besluit met een complete uitwerking van de casus conform een van de beschreven implementatiemodellen.

elke Typist tevens een Werknemer is, nog steeds geldt. Immers; elk Typist-salaris is tevens een geldig Werknemer-salaris.

Ter afsluiting van de behandeling van subdomeinen kan nog worden vermeld dat ook NOT NULL constraints op een lager niveau in de hiërarchie kunnen worden geherdefinieerd. Dit maakt het mogelijk om een attribuut dat in een superklasse optioneel is, in een subklasse alsnog verplicht te verklaren. Omgekeerd kan natuurlijk niet, omdat de waarde NULL voor het attribuut in de superklasse niet is toegestaan.

GENERALISATIEHIËRARCHIEËN

Voordat we de casus uitwerken, bekijken we eerst welke invloed generalisatiehiërarchieën hebben op de standaard relationele constraints voor domeinen, NOT NULL en primaire, unieke en verwijzende sleutels. Deze invloed is aanzienlijk, zoals blijkt uit de tabel in figuur 2.

Domein constraints worden in de meeste relationele dbms'en ondersteund door middel van check constraints. De specificatie van een dergelijke constraint luidt dan bijvoorbeeld 'SALARIS > 0'. In een generalisatiehiërarchie verandert dat in principe niet, tenzij er in subklassen sprake is van het herdefiniëren van het gegevenstype door middel van een subdomein. In dat geval ontstaat er in het geval van implementatie in één tabel per hiërarchie, een type-afhankelijke check constraint als 'SALARIS > 0 AND ((P_TYPE_CD='TYPIST' AND SALARIS <= 3000) OR P_TYPE_CD <> 'TYPIST')'.

Bij implementatie met behulp van één tabel per klasse ontstaat er een probleem. Om vast te kunnen stellen of sprake is van een bepaalde subklasse (in het voorbeeld: Typist) moet er in andere tabellen worden gekeken, en dat is iets wat in de huidige generatie rdbms'en niet mogelijk is, omdat de expressies in check constraints door vrijwel iedere leverancier worden beperkt tot expressies, die kunnen worden geëvalueerd op basis van een enkel record.

Not null constraints blijven (bij één tabel per hiërarchie) not null constraints voor zover ze zijn gedefinieerd op de hoogste entiteit in de hiërarchie. Not null constraints op subentiteiten worden check constraints in de vorm 'TYPE <> X OR ATTR IS NOT NULL'. Voor alle attributen in enige subentiteit geldt bovendien een 'null constraint', die afdwingt dat het attribuut verplicht leeg is als het record niet van een bepaald type is.

Bij implementatie in één tabel per klasse kan met standaard not null constraints worden volstaan, tenzij het een optioneel attribuut betreft dat in een subklasse alsnog verplicht wordt. In dat geval bieden check constraints wederom uitkomst.

Voor primaire sleutels gelden geen bijzondere beperkingen, behalve dan dat alle klassen in de hiërarchie dezelfde primaire sleutel dienen te hebben.

Unieke sleutels vormen een geval apart. Wanneer een unieke sleutel op een subklasse wordt gedefinieerd, dan betekent dit, in het geval van één tabel per hiërarchie, dat de sleutelrubrieken veelal optioneel zullen zijn. In onze casus is het attribuut 'Personeelsnummer' als unieke sleutel op de subklasse 'Werknemer' een goed voorbeeld. Personeelsnummer is in de hiërarchie-tabel null allowed, omdat bepaalde subklassen (bijvoorbeeld Klant) het attribuut Personeelsnummer in het geheel niet kennen. De vraag is nu wat een unieke sleutel op het attribuut Personeelsnummer in de hiërarchietabel betekent. Mogen er meerdere records met personeelsnummer NULL worden ingevoerd of niet? Deze semantiek is helaas leveranciersspecifiek.

In het geval van Oracle is het zelfs zo, dat het uitmaakt of de sleutel uit één of meerdere attributen bestaat. Een unieke sleutel op alleen 'Personeelsnummer' gaat naar wens: meerdere records (bijvoorbeeld Klanten) zonder personeelsnummer kunnen probleemloos worden ingevoerd. Bestaat de sleutel echter uit meerdere attributen (neem bijvoorbeeld een – in dit geval onzin-

	Standaard	Eén tabel per hiërarchie	Eén tabel per klasse
Domein	Afdwingen door middel van check constraints	Afdwingen door middel van standaard check constraints, tenzij overrides in subklassen, dan type-afhankelijke check constraints.	Afdwingen door middel van check constraints, overrides in subklassen niet ondersteund.
NOT NULL constraint	NOT NULL constraint	Verplichte attributen uit een subklasse worden null allowed in de hiërarchie-tabel met type-afhankelijke check constraints om het attribuut conditioneel verplicht te verklaren.	Geen bijzonderheden, tenzij overrides in subklassen, dan type-afhankelijke check constraints.
Primaire sleutel	PRIMARY KEY constraint	Alle tabellen in de hiërarchie dienen dezelfde primaire sleutel te hebben.	Alle tabellen in de hiërarchie dienen dezelfde primaire sleutel te hebben.
Unieke sleutel	UNIQUE KEY constraint	Semantiek van unieke sleutels i.c.m. optionele sleutelrubrieken is dbms- en situatie-specifiek.	Geen bijzonderheden.
Verwijzende sleutel	FOREIGN KEY constraint	Geen bijzonderheden.	Geen bijzonderheden.

FIGUUR 2: INVLOED VAN GENERALISATIEHIËRARCHIEËN OP STANDAARD RELATIONELE CONSTRAINTS.

nige – sleutel op Personeelsnummer en Salaris) dan worden deze records geweigerd!
Tot slot, voor verwijzende sleutels gelden geen bijzonderheden.

UITWERKING CASUS

Ter afsluiting van het onderwerp generalisatiehiërarchieën volgt een uitwerking van onze casus volgens de implementatievorm 'één tabel per hiërarchie'. We gebruiken hierbij de vertaling van relationele constructies zoals hierboven beschreven. Bij de uitwerking is uitgegaan van de SQL-variant van het Oracle-dbms. Voor andere dbms'en zullen de statements op enkele plaatsen moeten worden aangepast.

Om te beginnen definiëren we de hiërarchie-tabel, inclusief primaire sleutel, unieke sleutel op Personeelsnr en check-constraint op Salaris. Om te registreren van welk type (welke subklasse) een record precies is, wordt een attribuut P_TYPE_CD bijgehouden.

```
CREATE TABLE PERSOON_HIER (
P#                    NUMBER (10) NOT NULL,
NAAM                  VARCHAR2 (255) NOT NULL,
ADRES                 VARCHAR2 (255),
KLANTNR               NUMBER (10,0),
KLANTENKAARTNR        NUMBER (8,0),
BONUSPUNTEN           NUMBER (6,0),
PERSONEELSNR          NUMBER (8,0),
SALARIS               NUMBER (8,2),
AANSL_PM              NUMBER (3,0),
RIJBEWIJSNR           NUMBER (10,0),
KENTEKEN              CHAR (6),
P_TYPE_CD             CHAR (10) NOT NULL
CONSTRAINT PERS_P_TYPE_CD_CHK CHECK (P_TYPE_CD IN
('KLANT','TR_KLANT','TYPIST','CHAUFFEUR',
'DIRECTEUR')),
CONSTRAINT SALARIS_CHK CHECK (SALARIS > 0 AND
((P_TYPE_CD='TYPIST' AND SALARIS <= 3000) OR
P_TYPE_CD <> 'TYPIST'),
CONSTRAINT PK_PERSOON_HIER PRIMARY KEY ( P# ),
CONSTRAINT UNQ_PERSONEELSNR UNIQUE (PERSONEELSNR));
```

Vervolgens kunnen we de check constraints ter implementatie van NOT NULL regels toevoegen op de kolommen van de subklassen.

```
ALTER TABLE PERSOON_HIER ADD CONSTRAINT
KLANTNR_CHK
CHECK ((P_TYPE_CD IN ('KLANT','TR_KLANT') AND
KLANTNR IS NOT NULL) OR (P_TYPE_CD NOT IN
('KLANT','TR_KLANT') AND KLANTNR IS NULL));

ALTER TABLE PERSOON_HIER ADD CONSTRAINT
KLANTENKAARTNR_CHK
CHECK ((P_TYPE_CD = 'TR_KLANT' AND KLANTENKAARTNR
IS NOT NULL) OR (P_TYPE_CD <> 'TR_KLANT' AND
KLANTENKAARTNR IS NULL));

ALTER TABLE PERSOON_HIER ADD CONSTRAINT
BONUS-PUNTEN_CHK
CHECK ((P_TYPE_CD = 'TR_KLANT') OR (P_TYPE_CD <>
'TR_KLANT' AND KLANTENKAARTNR IS NULL));

ALTER TABLE PERSOON_HIER ADD CONSTRAINT
PERSONEELSNR_CHK
CHECK ((P_TYPE_CD IN ('TYPIST','CHAUFFEUR',
'DIRECTEUR') AND PERSONEELSNR IS NOT NULL) OR
(P_TYPE_CD NOT IN ('TYPIST','CHAUFFEUR',
'DIRECTEUR') AND PERSONEELSNR IS NULL));

ALTER TABLE PERSOON_HIER ADD CONSTRAINT
SALARIS_CHK
CHECK ((P_TYPE_CD IN ('TYPIST','CHAUFFEUR',
'DIRECTEUR') AND SALARIS IS NOT NULL) OR
(P_TYPE_CD NOT IN ('TYPIST','CHAUFFEUR',
'DIRECTEUR') AND SALARIS IS NULL));

ALTER TABLE PERSOON_HIER ADD CONSTRAINT
AANSL_PM_CHK
CHECK ((P_TYPE_CD = 'TYPIST' AND AANSL_PM IS NOT
NULL) OR (P_TYPE_CD <> 'TYPIST' AND AANSL_PM IS
NULL));

ALTER TABLE PERSOON_HIER ADD CONSTRAINT
RIJBEWIJSNR_CHK
CHECK ((P_TYPE_CD = 'CHAUFFEUR' AND RIJBEWIJSNR IS
NOT NULL) OR (P_TYPE_CD <> 'CHAUFFEUR' AND
RIJBEWIJSNR IS NULL));

ALTER TABLE PERSOON_HIER ADD CONSTRAINT
KENTEKEN_CHK
CHECK ((P_TYPE_CD = 'DIRECTEUR' AND KENTEKEN IS
NOT NULL) OR (P_TYPE_CD <> 'DIRECTEUR' AND
KENTEKEN IS NULL));
```

Het attribuut P# kunnen we als echte, onzichtbare object identity beschouwen, maar ook als normaal, zichbaar attribuut. In Oracle doen we dat met een sequence en een insert-trigger.

```
CREATE SEQUENCE PERSOON_HIER_SEQ;

CREATE OR REPLACE TRIGGER PERSOON_HIER_BRI BEFORE
INSERT ON PERSOON_HIER
FOR EACH ROW
WHEN (new.P# IS NULL)
BEGIN
  SELECT PERSOON_HIER_SEQ.NEXTVAL INTO :new.P#
FROM DUAL;
END;
/
```

Tot zover de implementatie 'onder de motorkap', buiten het zicht van de programmeur. De individuele klassen implementeren we door middel van views op de hiërarchietabel. Het is hierbij relevant om te weten welke subklassen er allemaal van een klasse zijn afgeleid:

```
CREATE OR REPLACE VIEW PERSOON ( P#, NAAM, ADRES )
AS SELECT P#, NAAM, ADRES
FROM PERSOON_HIER;

CREATE OR REPLACE VIEW KLANT ( P#, NAAM, ADRES,
KLANTNR ) AS SELECT P#, NAAM, ADRES, KLANTNR
FROM PERSOON_HIER
WHERE P_TYPE_CD IN ('KLANT','TR_KLANT');

CREATE OR REPLACE VIEW TR_KLANT ( P#, NAAM, ADRES,
KLANTNR, KLANTENKAARTNR, BONUSPUNTEN ) AS
SELECT P#, NAAM, ADRES, KLANTNR, KLANTENKAARTNR,
BONUSPUNTEN
FROM PERSOON_HIER
WHERE P_TYPE_CD = 'TR_KLANT';

CREATE OR REPLACE VIEW WERKNEMER ( P#, NAAM,
ADRES, PERSONEELSNR, SALARIS ) AS
SELECT P#, NAAM, ADRES, PERSONEELSNR, SALARIS
FROM PERSOON_HIER
WHERE P_TYPE_CD IN ('TYPIST', 'CHAUFFEUR',
'DIRECTEUR');

CREATE OR REPLACE VIEW TYPIST ( P#, NAAM, ADRES,
PERSONEELSNR, SALARIS, AANSL_PM ) AS
SELECT P#, NAAM, ADRES, PERSONEELSNR, SALARIS,
AANSL_PM
FROM PERSOON_HIER
WHERE P_TYPE_CD = 'TYPIST';

CREATE OR REPLACE VIEW CHAUFFEUR ( P#, NAAM,
ADRES, PERSONEELSNR, SALARIS, RIJBEWIJSNR) AS
SELECT P#, NAAM, ADRES, PERSONEELSNR, SALARIS,
RIJBEWIJSNR
FROM PERSOON_HIER
WHERE P_TYPE_CD = 'CHAUFFEUR';

CREATE OR REPLACE VIEW DIRECTEUR ( P#, NAAM,
ADRES, PERSONEELSNR, SALARIS, KENTEKEN) AS
select P#, NAAM, ADRES, PERSONEELSNR, SALARIS,
KENTEKEN
FROM PERSOON_HIER
WHERE P_TYPE_CD = 'DIRECTEUR';
```

Omdat de views gedefinieerd zijn als eenvoudige selecties op een enkele tabel, zijn ze automatisch updatable door Oracle. Wat echter niet automatisch gebeurt is het toekennen van de juiste waarde voor P_TYPE_CD. Bovendien is het zo, dat voor klassen die we niet rechtstreeks willen kunnen instantiëren (zogenaamde abstracte klassen) een insert expliciet moet worden verboden. Daarom definiëren we op alle views een instead-of-insert trigger:

Bestaat de sleutel echter uit meerdere attributen, dan worden deze records geweigerd!

```
CREATE TRIGGER PERSOON_II INSTEAD OF INSERT ON
PERSOON
FOR EACH ROW
BEGIN
    RAISE_APPLICATION_ERROR('-20000','Toevoegen
    van PERSOON is niet toegestaan, gebruik een
    concrete subklasse');
END;
/

CREATE TRIGGER KLANT_II INSTEAD OF INSERT ON KLANT
FOR EACH ROW
BEGIN
    INSERT INTO PERSOON_HIER ( P_TYPE_CD, P#,
    NAAM, ADRES, KLANTNR )
        VALUES ('KLANT', :new.P#, :new.NAAM,
        :new.ADRES, :new.KLANTNR);
END;
/

CREATE TRIGGER TR_KLANT_II INSTEAD OF INSERT ON
TR_KLANT
FOR EACH ROW
BEGIN
    INSERT INTO PERSOON_HIER ( P_TYPE_CD, P#,
    NAAM, ADRES, KLANTNR, KLANTENKAARTNR,
    BONUSPUNTEN )
        VALUES ('TR_KLANT', :new.P#, :new.NAAM,
        :new.ADRES, :new.KLANTNR, :new.KLANTEN
        KAARTNR, :new.BONUSPUNTEN);
END;
/

CREATE TRIGGER WERKNEMER_II INSTEAD OF INSERT ON
WERKNEMER
FOR EACH ROW
BEGIN
     RAISE_APPLICATION_ERROR('-20000','Toevoegen
     van WERKNEMER is niet toegestaan, gebruik een
     concrete subklasse');
END;
/
```

```
CREATE TRIGGER TYPIST_II INSTEAD OF INSERT ON
TYPIST
FOR EACH ROW
BEGIN
    INSERT INTO PERSOON_HIER ( P_TYPE_CD, P#,
    NAAM, ADRES, PERSONEELSNR, SALARIS, AANSL_PM )
        VALUES ('TYPIST', :new.P#, :new.NAAM,
        :new.ADRES, :new.PERSONEELSNR,
        :new.SALARIS, :new.AANSL_PM);
END;
/
```

Een simpel gegevensmodel leidt al snel tot een fors aandeel logica

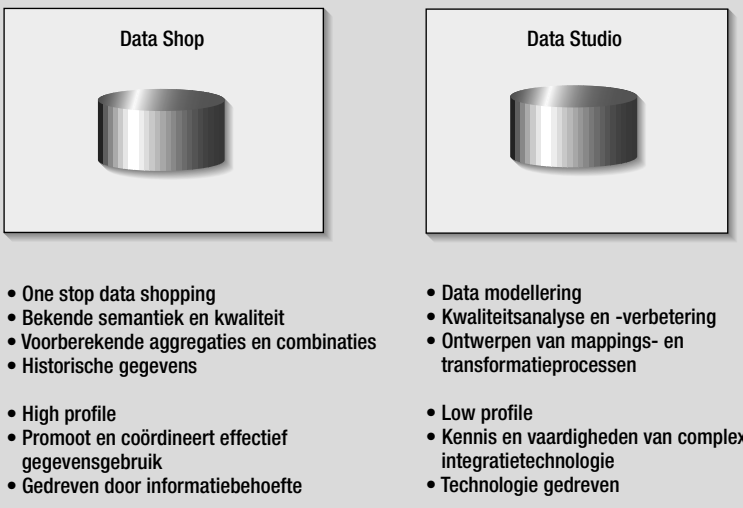
```
CREATE TRIGGER CHAUFFEUR_II INSTEAD OF INSERT ON
CHAUFFEUR
FOR EACH ROW
BEGIN
    INSERT INTO PERSOON_HIER ( P_TYPE_CD, P#,NAAM,
    ADRES, PERSONEELSNR, SALARIS, RIJBEWIJSNR )
        VALUES ('CHAUFFEUR', :new.P#, :new.NAAM,
        :new.ADRES, :new.PERSONEELSNR,
        :new.SALARIS, :new.RIJBEWIJSNR);
END;
/
```

```
CREATE TRIGGER DIRECTEUR_II INSTEAD OF INSERT ON
DIRECTEUR
FOR EACH ROW
BEGIN
    INSERT INTO PERSOON_HIER ( P_TYPE_CD, P#,NAAM,
    ADRES, PERSONEELSNR, SALARIS, KENTEKEN )
        VALUES ('DIRECTEUR', :new.P#, :new.NAAM,
        :new.ADRES, :new.PERSONEELSNR,
        :new.SALARIS, :new.KENTEKEN);
END;
/
```

TOT SLOT

De casus toont aan dat ook een simpel gegevensmodel met generalisatie-aspecten al snel leidt tot een fors aandeel modellering en logica in de database. Bovendien hebben eenvoudige wijzigingen in het objectmodel (toevoegen van een nieuwe subklasse, abstract maken van een niet-abstracte klasse en andersom) een veelvoud aan wijzigingen in constraints, views en triggers tot gevolg. Dit is onderhoudsintensief en foutgevoelig. Het verdient dan ook sterk de aanbeveling om de database-implementatie te laten genereren vanuit een dictionary, waarin de generalisatiehiërarchie netjes kan worden gemodelleerd zonder vervuiling met implementatie-aspecten. Een gegevensmodel voor een dergelijke dictionary is in het vorige artikel in DB/M 7 beschreven. ●

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



Gegevens-exploitatie

In DB/M nr. 6 is het artikel "Gegevens-exploitatie: voorbij Business Intelligence" van auteur Gerton Kuis gepubliceerd. In de weergave van figuur 4 is een fout geslopen. De teksten onder Data Shop horen onder Data Studio te staan, en andersom. Voor de volledigheid wordt de juiste figuur hier afgebeeld. Het complete gerectificeerde artikel is op internet te vinden. (http://www.array.nl/dbm/art0206/BI.pdf)