Domeinen in Oracle-applicaties

Niet meer telkens de bekende types definiëren

Bepaalde datatypes, zoals een postcode, een bedrag, een Boolean (J/N of 1/0 veld), een code met verplicht alleen hoofdletters of een datum zonder tijd komen vaak voor in de tabeldefinities bij administratieve applicaties. In dit artikel bespreken we het gebruik van domeinen om niet telkens de bekende types te definiëren.

In Oracle is het de gewoonte om deze definities telkens te definiëren in de bekende types zoals varchar2(6), number(10,2), varchar2(1) of date. Dus als een postcode bijvoorbeeld tien keer voorkomt in de database wordt even vaak gedefinieerd dat deze van het type varchar2(6) is en mogelijk met telkens de constraint dat deze uit 4 cijfers en 2 hoofdletters bestaat. Voor de datum (zonder tijd) wordt telkens aangegeven dat het tijdsdeel '00:00:00' moet zijn en voor de code dat deze alleen hoofdletters mag bevatten, zoals in onderstaand voorbeeld.

```
Create table testtypes
( Janee     number(1)
, Datum     date
, Landcode varchar2(3)
, Bedrag    number(10,2)
, Postcode varchar2(6)
);

Alter table testtypes add constraint booleantype  check (Janee in (0,
1));
Alter table testtypes add constraint landcodetype check (landcode =
upper(landcode));
Alter table testtypes add constraint datumtype    check (datum     =
trunc(datum));
Alter table testtypes add constraint bedragtype   check (bedrag    >=
0);
Alter table testtypes add constraint postcodetype check (
      Substr(postcode,1,1) between '1' and '9'
  And Substr(postcode,2,1) between '0' and '9'
  And Substr(postcode,3,1) between '0' and '9'
  And Substr(postcode,4,1) between '0' and '9'
  And Substr(postcode,5,1) between 'A' and 'Z'
  And Substr(postcode,6,1) between 'A' and 'Z'   );
```

Of, iets minder strict:

```
Alter table testtypes add constraint postcodetype check (
  Translate(postcode,
            '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ',
            '9999999999XXXXXXXXXXXXXXXXXXXXXXXXXX') = '9999XX');
```

Met goede insert:

```
Insert into testtypes values (0, trunc(sysdate), 'NL', 1234.56,
'6700AA');
```

En een foutieve insert:

```
Insert into testtypes values (2,       sysdate,  'nl', -234.56,
'AA6700');
```

Als bij dit soort kolommen telkens opnieuw het datatype en de constraints moeten worden gedefinieerd, dan leidt dit zeker tot veel extra werk en vaak ook tot inconsistenties en/of onvolledige definities. Nog vaker zullen de datatypes en constraints over systemen heen afwijken met problemen bij het uitwisselen van gegevens tussen deze systemen, bijvoorbeeld:

- Een naam is de ene keer als 20 de andere keer als 36 lang gedefinieerd;
- Een bedrag is de ene keer als number(10,2), de andere keer als number(12,2) gedefinieerd;
- Een boolean is de ene keer als number (1), de andere keer als een varchar2(1) gedefinieerd, mogelijk zonder verdere controle op de inhoud;
- Een datum waarbij de tijd altijd 00:00:00 moet zijn (ontvangstdatum van een brief of belastingaangifte) staat toch toe om een tijd op te slaan.

Om dit probleem met inconsistenties te voorkomen is het beter om hiervoor definities vast te leggen. Hiervoor bestaan de volgende alternatieven:

- Het create type statement in Oracle;
- Het create domain statement zoals dat in de ANSI standaards is gedefinieerd;
- Het definiëren van de types of domeinen in een data dictionary.

We zullen hieronder deze alternatieven nader onderzoeken. Wat zouden we hierbij willen? Wat zijn de eigenlijk de wensen of requirements voor de type- of domein- definities? Dit zijn:

- Eenmalig definiëren van deze types en hergebruik van deze definities in de kolomdefinities van de tabellen;
- Inclusief constraints, zoals: Bij de datum moet de tijd '00:00:00' zijn; De postcode moet het formaat 9999AA hebben;
- De insert, update, delete en select statements blijven hetzelfde (ofwel: even eenvoudig) als ze waren.

Het create type statement

Oracle kent al het begrip create type. Hieronder is dit uitgewerkt met een aantal types en een tabel waarin deze worden gebruikt.

```
Create type idtype         as object (id         number(10));
Create type booleantype    as object (boolean    number(1));
Create type naamtype       as object (naam       varchar2(80));
Create type postcodetype   as object (postcode   varchar2(6));
Create type huisnrtype     as object (huisnr     number(6));
create type telefoontype   as varray(5) of       varchar2(25);
Create type datumtype      as object (datum      date);
Create type datumtijdtype as object (datumtijd date);
Create type bedragtype     as object (bedrag     number(12,2));

Create type adrestype as object
( Straatnaam     naamtype
, huisnummer     huisnrtype
, postcode       postcodetype
, woonplaats     naamtype
);
```

Of:

```
Create type adrestyp2 as object
( Straatnaam     varchar2(80)
, huisnummer     number(6)
, postcode       varchar2(6)
, woonplaats     varchar2(80)
);
```

Omdat het PL/SQL betreft moet na elke statement nog een '/' komen!
Het telefoontype kan een lijst met 1 tot 5 telefoonnummers bevatten. Het adrestype bevat hier 4 attributen. Het definiëren van constraints, zoals hierboven per kolom, is bij het type zelf niet mogelijk.

Deze types kunnen weer in een tabeldefinitie worden gebruikt, bijvoorbeeld een tabel klant, met attributen:

- klantnummer (is ook primary key),
- klantnummer van de moederorganisatie (is een foreign key naar dezelfde tabel),
- klanttype (zakelijk of particulier),
- naam, adres,
- een lijstje van maximaal 5 telefoonnummers,
- maximum en huidig saldo,
- systeemdatum dat deze klant is aangemaakt,
- datum van de eerste en laatste aankoop.

```
Create table klant
( klantnummer       idtype     not null
, moederorganisatie idtype     null      -- klantnummer van de ..
, zakelijk     booleantype     not null -- 0 = NEE = particuliere klant
, naam                         naamtype       not null
, adres                        adrestyp2      not null
, telefoon                     telefoontype   not null
, max_saldo                    bedragtype     not null
, saldo                        bedragtype     not null
, dt_aangemaakt                datumtijdtype not null
, dd_eerste_aankoop            datumtype      null
, dd_laatste_aankoop           datumtype      null );
```

Het inserten van een record in de tabel klant ziet er nu als volgt uit:

```
Insert into klant values
( idtype(1)
, idtype(NULL)
, booleantype(1)
, naamtype('Janssen')
, adrestyp2('Damstraat', 123, '1234AB', 'Udam')
, telefoontype('012-1231230', '06-12345678')
, bedragtype(10000.00)
, bedragtype(9876.54)
, datumtydtype(sysdate)
, datumtype(to_date('2009-01-01', 'YYYY-MM-DD'))
, datumtype(to_date('2010-04-04', 'YYYY-MM-DD'))
);
Commit;
```

En het select statement ziet er uit als:

```
select * from klant;

KLANTNUMMER(ID)
MOEDERORGANISATIE(ID)
ZAKELIJK(BOOLEAN)
NAAM(NAAM)
ADRES(STRAATNAAM, HUISNUMMER, POSTCODE, WOONPLAATS)
TELEFOON
MAX_SALDO(BEDRAG)
SALDO(BEDRAG)
DT_AANGEMAAKT(DATUMTIJD)
DD_EERSTE_AANKOOP(DATUM)
DD_LAATSTE_AANKOOP(DATUM)
-----------------------------------------------------------

IDTYPE(1)
IDTYPE(NULL)
BOOLEANTYPE(1)
NAAMTYPE('Janssen')
ADRESTYP2('Damstraat', 123, '1234AB', 'Udam')
TELEFOONTYPE('012-1231230', '06-12345678')
BEDRAGTYPE(10000)
BEDRAGTYPE(9876.54)
DATUMTYDTYPE('2010-04-28 13:50:20')
DATUMTYPE('2009-01-01 00:00:00')
DATUMTYPE('2010-04-04 00:00:00')
```

Of:

```
select SALDO from klant where klantnummer = IDTYPE(1);

SALDO(BEDRAG)
--------------------
BEDRAGTYPE(9876.54)
```

We willen nu het bovenstaande bedrag maal 2 uit de tabel queriën. Helaas gaat dit niet zo gemakkelijk meer:

```
select SALDO*2 from klant where klantnummer = IDTYPE(1);
geeft:
ERROR at line 1:
ORA-00932: inconsistent datatypes: expected NUMBER got JJC.BEDRAGTYPE
```

We moeten daarvoor met een functie uit het type het betreffende bedrag halen:

```
Drop table klant;

Create or replace type bedragtype    as object
( bedrag     number(12,2)
, MEMBER FUNCTION get_bedrag(y bedragtype) RETURN NUMBER);
/

Create table klant . . . . . ;

CREATE or replace TYPE BODY bedragtype
IS
   MEMBER FUNCTION get_bedrag(y bedragtype)
   RETURN NUMBER
   IS
      x NUMBER;
   BEGIN
      x := y.bedrag ;
      RETURN (x);
   END;
END;
/

select K.SALDO.get_bedrag(K.saldo) * 2 as DUBBEL
from   klant K
where  klantnummer = IDTYPE(1);

    DUBBEL
----------
  19753.08
```

Dit werkt, maar handig is het niet echt. Mogelijk wordt het met een view allemaal wat bruikbaarder nadat we eerst voor alle types ook zo'n functie aan het type hebben toegevoegd?

```
Create or replace view klant_v00
as
select k.MAX_SALDO.get_bedrag(k.max_saldo) as MAX_SALDO
,      k.SALDO.get_bedrag(k.saldo)          as SALDO
--     etc voor alle andere kolommen
from   klant k;

select saldo * 2 from klant_v00;

 MAX_SALDO
----------
  19753.08
```

De tabel en de view zien er nu zo uit:

```
SQL> desc klant_v00
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------
 MAX_SALDO                                          NUMBER
 SALDO                                              NUMBER

SQL> desc klant
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------
 KLANTNUMMER                               NOT NULL IDTYPE
 MOEDERORGANISATIE                                  IDTYPE
 ZAKELIJK                                  NOT NULL BOOLEANTYPE
 NAAM                                      NOT NULL NAAMTYPE
 ADRES                                     NOT NULL ADRESTYP2
 TELEFOON                                  NOT NULL TELEFOONTYPE
 MAX_SALDO                                 NOT NULL BEDRAGTYPE
 SALDO                                     NOT NULL BEDRAGTYPE
 DT_AANGEMAAKT                             NOT NULL DATUMTYDTYPE
 DD_EERSTE_AANKOOP                                  DATUMTYPE
 DD_LAATSTE_AANKOOP                                 DATUMTYPE
```

Een andere beperking van deze object types is dat deze niet in een primary key of index mogen staan. Dit betekent dat alle primary keys (en dus ook de foreign keys die intrinsiek van hetzelfde type zijn) niet eenvoudig als type kunnen worden gedefinieerd:

```
Alter table klant add ( constraint Klant_pk
  primary key (klantnummer)  using index);

  primary key (klantnummer)
               *
ERROR at line 3:
ORA-02329: column of datatype ADT cannot be unique or a primary key

Create index Klant_I1 on klant (klantnummer);
ERROR at line 1:
ORA-02327: cannot create index on expression with datatype ADT
```

Erg handig is dit niet. Om eenduidige definities voor alle kolommen, zoals een bedrag, een boolean, een postcode, een naam en een datum, te krijgen lijkt het Oracle object 'type' niet erg geschikt.

Het is zeker wel bruikbaar voor een enkel specifiek datatype, zoals SDO_GEOMETRY in Oracle Spatial, maar voor de meeste kolommen zullen we in Oracle toch weer terugvallen op de bekende datatypes NUMBER, VARCHAR2, DATE, CLOB en BLOB.

De ANSI SQL standaard

De ANSI SQL standaard kent het begrip domein! Deze definitie is ook geïmplementeerd in PostgreSQL. Hierbij kunnen domeinen met constraints en default waarden gedefinieerd worden. Deze worden weer hergebruikt in tabeldefinities zonder bovenstaande overhead en complexe syntax bij insert en update.

De syntax is dan:

```
CREATE DOMAIN name [AS] data_type
    [ DEFAULT expression ]
    [ constraint [ ... ] ]
met
[ CONSTRAINT constraint_name ]
{ NOT NULL | NULL | CHECK (expression) }
```

Voor het hiervoor gebruikte voorbeeld zou dit er uit kunnen zien als:

```
Create domain idtype            as number(10);
Create domain booleantype       as number(1)
       default 0
       constraint booleantype   check (value in (0, 1));
Create domain naamtype          as varchar2(80);
Create domain huisnrtype        as number(6)
       constraint huisnrtype    check (value   >= 1);
create domain telefoontype      as varchar2(25);
Create domain datumtype         as date
       constraint datumtype     check (value = trunc(value));
Create domain datumtijdtype     as date;
Create domain bedragtype        as number(12,2)
       constraint bedragtype    check (value   >= 0);
Create domain postcodetype      as varchar2(6)
       constraint postcodetype check (
         Translate(value,
            '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ',
            '9999999999XXXXXXXXXXXXXXXXXXXXXXXXXX') = '9999XX');
```

De tabel KLANT kan dan worden gedefinieerd met:

```
Create table klant
( klantnummer       idtype         not null
, moederorganisatie idtype         null      -- klantnummer van de ..
, zakelijk     booleantype    not null -- 0 = NEE = particuliere klant
, naam              naamtype       not null
, Straatnaam        naamtype       not null
, huisnummer        huisnrtype     not null
, postcode          postcodetype   not null
, woonplaats        naamtype       not null
, telefoon          telefoontype   not null
, max_saldo         bedragtype     not null
, saldo             bedragtype     not null
, dt_aangemaakt     datumtijdtype  not null
, dd_eerste_aankoop datumtype      null
, dd_laatste_aankoop datumtype     null );
```

De insert, update, delete en select statements blijven geheel identiek aan deze statements zoals we ze kennen bij gebruik van de datatypes number, varchar2 of date. Ook de definitie van de primary key, foreign key of index wijzigt niet. Maar wel is er een grote hoeveelheid redundantie uit de definitie van het schema gehaald. Dat beogen we bij onze data, waarom dan niet ook bij de metadata?

Zoals gezegd, bij PostgreSQL is deze ANSI Standaard al geïmplementeerd. Daarnaast kent PostgreSQL ook een create type voor complexere objecten zoals in Oracle. Sybase en Micrsoft SQL Server hebben met sp_addtype een alternatief voor de domeinen maar met de beperking dat hier geen constraints mogelijk zijn.

Alleen in een repository

De hiervoor genoemde redundantie in de metadata kan ook worden voorkomen door gebruik te maken van een data dictionary of repository waarin de definitie van het schema geheel wordt vastgelegd. Door in de repository gebruik te maken van domeinen of datatypes wordt de definitie daar eenmaal vastgelegd. Bij het genereren van de tabeldefinities voor Oracle wordt deze typedefinitie voor elke kolom uitgegenereerd. Dit is gegenereerd (en wordt bij wijzigingen opnieuw gegenereerd) waardoor in elk geval het extra werk en de inconsistenties vermeden worden. Onder anderen biedt Power Designer deze mogelijkheid.

Conclusie

Door het introduceren van domeinen kan een grote hoeveelheid redundantie uit de definitie van het schema gehaald worden. Het vermeiden van redundantie beogen we al bij onze data, waarom dan niet ook bij de metadata?

Het zou dan ook een grote aanwinst voor Oracle zijn als in de volgende versie een domeindefinitie mogelijk is zoals gedefinieerd in de ANSI standaards. Daarnaast blijven de bestaande object types in Oracle nuttig voor zeer complexe objecten zoals SDO_GEOMETRY in Oracle Spatial. Deze types zijn echter niet geschikt voor de definitie van standaard kolommen in verband met de beperkingen (geen indexen en keys) en de complexe syntax bij queries.

Referenties

- *Loonen. Gegevensmodel van een distributed data dictionary. Database Magazine 1997/4.*
- *Loonen. Datum en tijd in het logisch en fysiek gegevensmodel. Database Magazine 2001/6*
- *Loonen. Kwaliteit van gegevens begint bij het begin. Database Magazine 2005/4,5*
- *De ANSI SQL 92 standaard, Zie http://savage.net.au/SQL/sql-92.bnf.html of http://www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt , hoofdstuk 4.7 Domains en 4.8 Columns*
- *Definitie van domeinen in PostgreSQL, Zie http://www.postgresql.org/docs/8.1/interactive/sql-createdomain.html*
- *Definitie van object types in Oracle, Zie http://download-east.oracle.com/docs/cd/B19306_01/server.102/b14200/statements_8001.htm*

Toon Loonen is werkzaam bij Capgemini en gespecialiseerd in (logisch en fysiek) gegevensmodellering. Hij is bereikbaar via e-mail: toon.loonen@capgemini.com of toon.loonen@inter.nl.net.