Views als vereenvoudigde invalshoek op een database

Silly SQL (3): Kijken mag, aanraken niet

Rick van Rein

Bij het ontwerp van SQL heeft men met views geprobeerd om standaard query's voor te vormen. Met een beperkte invalshoek op een complexe database kon hiermee worden voorkomen dat de menselijke gebruiker blootgesteld zou worden aan de complexiteit die nodig was om een standaardvraag te stellen.

In plaats van meerdere tabellen te moeten koppelen, kon gewoon worden gewerkt met:

```
select address
from person
where lastname = 'Jansen'
```

Zoals eerder besproken vinden we de gedachte dat mensen dit zouden intikken lichtelijk achterhaald. Wat dat betreft zouden views dus ook niet meer voorkomen als SQL zou worden ontworpen met wat we nu van de programmeerwijze van databasetoepassingen weten. Maar vanuit een andere invalshoek kan een view wel weer een nuttige bijdrage betekenen.

Lussen

Wie een taal ontwerpt, die maakt een keuze van de concepten waarin de programmeur denkt als hij die taal gebruikt. In Pascal werden bijvoorbeeld lussen en procedures geïntroduceerd opdat de programmeur gestructureerd kon denken; een grote verbetering ten opzichte van een taal als Basic waarin het nodig was om dergelijke structuren uit te werken in de vorm van losse Goto statements. Evenzo is de overdracht van parameters in Pascal beter geregeld dan in Basic, waarin men dit meestal zelf moest uitwerken door rond een Gosub statement de nodige variabelen van waarden te voorzien.
Pascal en Basic kunnen ongeveer dezelfde programma's uitdrukken, maar in Pascal zijn de taalconcepten hoger. Dat wil zeggen, ze zijn specifieker geënt op een bepaalde manier van programmeren. Spaghettiprogrammeren is in Pascal vrijwel onmogelijk en zeker *not done,* terwijl het in Basic schering en inslag is. Basic is er voor knutselklusjes, Pascal voor gestructureerd programmeren. Deze vorm van luxe – de ondersteuning van de mentale concepten van een programmeur door directe ondersteuning ervan in de taal – bepaalt in de praktijk het model dat de programmeur ter beschikking staat. Views kunnen in dat opzicht erg nuttig zijn, doordat ze een vereenvoudigde invalshoek op een database bieden. Deze kunnen worden gebruikt om de mechanistische aspecten van de opslagstructuren te verbergen, en dus een logisch model te vormen bovenop het fysieke model van de databasetabellen.

SQL is een standaard, maar wel een oude. In een serie 'Silly SQL' artikelen bespreken we de dingen die, in retrospect bezien, een stuk handiger hadden gekund. Deel 2 is gepubliceerd in DB/M 4, 2006.

Mechanistisch

Wie kent niet het probleem van andermans database-ontwerp, waarin schijnbaar willekeurig met kolommen gesmeten is, en waarin de verbanden tussen de data nogal onduidelijk zijn? Door de afgevuurde SQL-fragmenten te lezen is wel te zien hoe de verbanden worden gelegd, maar dan nog is het niet zo helder welk logisch model de programmeur in gedachten had. Zonder dat model of ontwerp wroet je op Goto-niveau door een programma, en moet je zelf nog bepalen welke lusstructuren er zijn gebruikt.

> Een belangrijk probleem om op te lossen is dat een insert niet altijd alle waarden invult

Het zou in theorie mogelijk zijn om door views de modelconcepten in de database op te nemen, in plaats van alleen de fysieke opslagtabellen. Die fysieke tabellen zijn onderhevig aan de wetten van de normalisatie, die op hun beurt weer ten dienste staan van de snelle werking van de database. Snelle werking heeft echter niets te maken met een inzichtelijke structuur. Die inzichtelijkheid kan worden hervonden door views die de database vanuit de beoogde ontwerpdoelen weerspiegelen. Het is mogelijk doordat dezelfde data op allerlei plaatsen in meerdere views zichtbaar kunnen worden gemaakt. Dit is in strijd met normalisatieregels, maar die hinderen niet omdat een view uiteindelijk netjes vertaalt in gewone, fysieke database-tabellentoegang.

Tot zover het concept in theorie. De praktijk is op schrijnende wijze anders. De oorzaak is mechanistisch van aard, namelijk dat er niet zomaar in een view kan worden geschreven, of data toegevoegd. Dat betekent dat er geen sprake is van een normaal bruikbare conceptuele afbeelding van de fysieke data. Als je moet lezen vanuit de ene invalshoek en schrijven vanuit de andere, dan wordt de verwarring alleen maar groter en is het beter om het maar te houden op louter fysieke opslagtabellen. En dat gebeurt in de praktijk dus ook veelvuldig.

Views

Als SQL nu zou worden ontworpen, met deze bedoeling ter vereenvoudiging van database-toepassingen in gedachten, dan zou het eerder aantrekkelijk zijn om rechtstreekse toegang tot fysieke tabellen tot views te beperken, dan om views als read-only constructies op te zetten. En nu dat eenmaal is uitgesproken, is een oplossing ook vrij eenvoudig te bedenken. Immers, de formulering van een view beschrijft alleen het leesaspect:

```
create view person as
select nm.firstname, nm.lastname, fam.address
from name nm, family fam
where nm.family=fam.id
```

Dit is een logische view op een persoon met een adres, waarin niet tot uitdrukking komt dat het adres van die persoon gedeeld wordt met de rest van de familie. De problemen met updaten van views worden duidelijk als we ons bedenken wat er gebeurt wanneer een persoon verhuist. Als dat inhoudt dat die persoon de familie verlaat, dan moet de verborgen link met de familie worden verbroken; als een heel gezin verhuist is er echter sprake van een verandering van het adresveld voor de familie. Beide zijn zinnige updates, en de read-only beschrijving in create view geeft geen handvatten om te kiezen welke in een bepaald geval van toepassing is.
Nogmaals, een taalontwerper die het view-concept zou willen toevoegen om een compleet logische view op de database mogelijk te maken, zou dat probleem moeten oplossen. Het lijkt billijk om dat te doen als integraal onderdeel van het create view statement. Laten we eens een voorbeeld van een beter statement uitwerken.

Een belangrijk probleem om op te lossen is dat een insert niet altijd alle waarden invult. Voor view-kolommen die volgen uit berekeningen of uit een tabelkolom die geen defaultwaarde heeft, zou dat expliciet kunnen worden opgegeven. Daarnaast zou een defaultwaarde moeten worden gekozen voor de kolommen die door een view verborgen worden gehouden. Bijvoorbeeld:

```
create view person as
select nm.firstname default 'Jan',
       nm.lastname default 'Jansen',
       fam.address default NULL
from name nm, family fam
where nm.family=fam.id
default set nm.family=NULL, fam.id=NULL
```

Hiermee is het al mogelijk om een insert op de view toe te laten, zonder de gebruikelijke problemen als gevolg van het afkomstig zijn uit meerdere tabellen. Zonodig zou er een andere specificatie kunnen worden gegeven als geldt dat een person alleen toegevoegd kan worden op het adres van de moeder (van wier familie we even een unieke achternaam veronderstellen):

```
create view person as
select  nm.firstname default 'Jan',
        nm.lastname default 'Jansen',
        fam.address default NULL
from name nm, family fam
where nm.family=fam.id
default  set nm.family=(
        select distinct f.id
        from family f, name n
        where n.family=f.id
        and n.lastname=nm.lastname)
```

Met behulp van deze laatste vorm zijn al tamelijk vergaande berekeningen te maken voor defaultwaarden – en dat omvat het hete hangijzer, te weten de kolommen die door een view aan het oog worden onttrokken. Hiermee, of met iets soortgelijks, zou het dus mogelijk zijn geweest om inserts te doen in elke view.

> Inzichtelijkheid kan worden hervonden door views die de database vanuit de beoogde ontwerpdoelen weerspiegelen

Een verdere uitbreiding die nodig is, is de omgang met updates. Ook die hebben niet altijd een heldere betekenis. In het binnenwerk van een view kunnen immers meerdere stappen worden genomen om de informatie te vinden, en dat kan soms betekenen dat er meerdere fysieke updates mogelijk zijn, en ook zinvol zijn, in respons op een logische update op een view, zoals hierboven in het voorbeeld van de adreswijziging.

Wanneer meerdere fysieke updates inhoudelijk zinvol zouden zijn, is er wellicht ruimte voor een extra view. Voor de adreswijziging, die ofwel een enkele persoon betrof ofwel een heel gezin, zou er een extra view kunnen zijn die bijvoorbeeld familieadressen afbeeldt. Een update van die view zou het adres van een geheel gezin aanpassen, terwijl een update van de hier gebruikte voorbeeldview person zou worden opgevat als een update van alleen het persoonlijke adres. Dit is logisch, zoals het een logisch

model betaamt. Een voorbeeldsyntax voor de expliciete aanpassing van een los veld zou iets kunnen zijn als:

```
create view person as
select nm.firstname, nm.lastname, fam.address
from name nm, family fam
where nm.family=fam.id
to alter fam.address set fam.id=(
         select id
         from family
         where address=fam.address)
```

De andere view zou iets worden als:

```
create view familyaddress as
select unique(nm.lastname), fam.address
from family fam, name nm
where nm.family=fam.id
to alter fam.address set fam.adress=fam.address
```

Het lijkt niet onredelijk de laatste regel als defaultgedrag aan te merken. Met wijzigingen van deze aard, of cosmetisch fraaiere varianten erop, zou het mogelijk worden om views te gebruiken als representatie van logische modellen. Doordat dit ook voor schrijfacties zou gelden, zou het niet meer nodig zijn naar de fysieke opslagstructuren te kijken, behalve voor de aanmaak van nieuwe logische views. Het gevolg zou dan zijn dat bestaande databases veel leesbaarder worden, net als hun toepassingen die ook gemakkelijker te programmeren en onderhouden zouden zijn. Achteraf bezien zouden views dus nooit read-only tabelvervangers hebben mogen worden. Maar achteraf heb je natuurlijk makkelijk praten.

Rick van Rein
Dr. ir. H. van Rein (rick@openfortress.nl) is ontwikkelaar en beheerder bij OpenFortress Digital signatures.