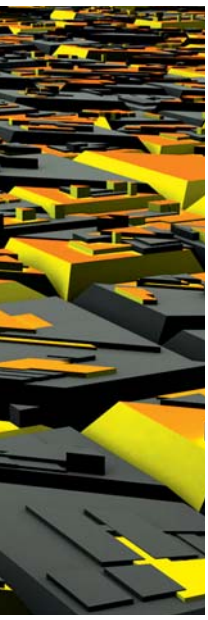


De HUB isolatieregel en normalisatie

Data Vault: gegevenskluizen?

Harm van der Lek

In DB/M 5, 2009 is de CIF-architectuur besproken. De conclusie was dat het zin heeft om het centrale datawarehouse genormaliseerd te ontwerpen. In dit artikel gaan we eens systematisch na wat dat laatste precies betekent.

Het blijkt dat we dan redelijk uitkomen op de methode die vandaag de dag bekend staat onder de naam Data Vault, onlosmakelijk verbonden met de bedenker ervan: Dan Linstedt. Met name wanneer we historie en normaliseren ook nog combineren met de gedachte van datawarehouse keys. De begrippen HUB- en satelliettabel komen dan vrij natuurlijk naar voren. De Linktabellen zijn wat lastiger in dit verhaal en daar blijken dan ook een paar pittige discussiepunten naar voren te komen.
Normaliseren is een woord dat door velen vrij gemakkelijk in de mond wordt genomen en eenieder die in dit vakgebied rondloopt wordt geacht ook meteen te begrijpen wat ermee wordt bedoeld. Een enkeling wil nog weleens toegeven niet helemaal (meer) te weten wat vierde en vijfde normaalvorm betekent, maar de derde dient toch gesneden koek te zijn. Laten we beginnen met vast te stellen dat het begrip zich (gelukkig) beperkt tot één tabel. Met andere woorden; van één tabel kan reeds worden vastgesteld of die genormaliseerd is of niet en een relationele database is dus genormaliseerd als alle tabellen dat zijn.
We beginnen uiteraard weer met een eenvoudig voorbeeld. In afbeelding 1 hebben we een tabel die deel uitmaakt van een kleine familiedatabase. Iedereen binnen deze clan kan worden geïdentificeerd door alleen al de (voor)naam. De eerste kolom 'Naam' is dus de primaire sleutel. Het voorbeeld lijkt te suggereren dat iedereen die in Den Haag woont ongehuwd is en iedereen in Leiden gehuwd. In dat geval zou men kunnen denken dat de kolom 'Burgerlijke Staat' functioneel afhankelijk is van de kolom 'Woonplaats'. Dat wil zeggen dat de waarden in de kolom 'Burgerlijke Staat' helemaal bepaald worden door de waarden in de kolom 'Woonplaats'. Als deze beperkingsregel zou gelden dan was deze tabel *niet* genormaliseerd. We zouden deze tabel kunnen normaliseren door de kolom 'Burgerlijke Staat' er uit te verwijderen. Om geen informatie te verliezen zouden we dan een extra tabel nodig hebben, waarin 'Plaats' dan de primaire sleutel is en 'Burgerlijke Staat' een attribuut van 'Plaats' wordt ("In Den Haag zijn de leden van onze familie ongehuwd"). Nu kunnen we ons deze herontwerpacties besparen, want de genoemde functionele afhankelijkheid is natuurlijk onzin. Al we er goed over nadenken (en indien nodig met materiedeskundigen er over praten), dan kunnen we concluderen dat alle kolommen in deze tabel alleen maar van de primaire sleutel afhangen. Dit laatste is exact de definitie van de derde normaalvorm (iets strenger en beter, maar complexer te formuleren, is de Boyce-Codd Normaal vorm).

Naam	Woonplaats	Burgerlijke staat
Jan	Den Haag	Ongehuwd
Piet	Leiden	Gehuwd

Afbeelding 1: Familiedatabase.

Historiseren

Als er nu iets verandert in deze database door een update, dan zijn we de waarde die er stond kwijt en dat willen we niet. We gaan het ontwerp dus wijzigen, zodat we de historie kunnen vasthouden. Hoe je dit doet is natuurlijk ook al lang bekend: je voegt aan de primaire sleutel van de tabel een datum (tijd) veld toe, bijvoorbeeld met de naam 'geldig van'. Afbeelding 2 geeft het resultaat, waarbij we dus zien dat Jan op dag 50 van Den Haag naar Leiden is verhuisd en op dag 70 is gehuwd. Met de saaie Piet is kennelijk nog niet veel gebeurd, althans niet volgens onze database.
Een interessante vraag is nu: is de tabel nog steeds genormaliseerd? Toen we wegens de verhuizing van Jan een nieuwe rij in de tabel moesten opnemen hebben we de waarde 'Ongehuwd' gekopieerd van de bestaande rij. Dit riekt naar redundantie en

Naam	Geldig van	Woonplaats	Burgerlijke staat
Jan	30	Den Haag	Ongehuwd
Jan	50	Leiden	Ongehuwd
Jan	70	Leiden	Gehuwd
Piet	10	Leiden	Gehuwd

Afbeelding 2: Datumveld toegevoegd.

had dat niet iets te maken met normalisatie? Ja en nee. Wat je kunt zeggen is dat als een database niet is genormaliseerd, dan kan er redundantie optreden en is er ook gevaar voor inconsistentie. Is dat hier het geval? Stel dat iemand de waarde 'Ongehuwd' in de tweede rij van afbeelding 2 verandert in 'Gehuwd', met andere woorden een afgeleide (want gekopieerde) waarde wordt gewijzigd; leidt dat niet tot inconsistentie? Nee! Stel namelijk dat we te horen krijgen dat Jan's huwelijksdatum wegens een foutje in eerdere berichten met terugwerkende kracht moet worden gewijzigd in dag 50, dan geeft de genoemde update het gewenst effect. Het is een beetje jammer dat de rijen 2 en 3 dan qua waarden van de echte attributen helemaal niet meer verschillen en dat dus rij 3 overbodig is (we noemen dit verschijnsel 'stotteren'), maar op inconsistenties is deze tabel niet te betrappen. In feite kan men het volgende beweren: als een database genormaliseerd is en zonder historie, dan kan men deze database historiseren door aan elke primaire sleutel een ingangsdatum (van de verandering) toe te voegen. Het resultaat is dan nog steeds een genormaliseerde database. In feite is dit de manier die Inmon in gedachte had toen hij voor het eerst een genormaliseerde vorm van het centrale datawarehouse voorstelde.

Ondanks het feit dat de database dus nog steeds genormaliseerd is kan men zich toch wat zorgen maken. Ten eerste is daar het feit dat men, zodra er maar een attribuut wijzigt, van de zeg 50, de andere 49 allemaal gekopieerd moeten worden. Maar wellicht nog veel erger blijkt het probleem waaraan we in het vorige artikel (DB/M 5, 2009) refereerden nog niet verdwenen. We hadden daar immers geconcludeerd dat juist het feit dat we historie stapelen in een tabel (daar ging het om de dimensie van een sterschema) tot inflexibiliteit leidt. We kwamen namelijk in allerlei problemen als we bijvoorbeeld een kolom aan zo'n tabel wilden toevoegen. Met onze genormaliseerde database is dat dus nog steeds niet opgelost, terwijl dat wel de belofte was.

Ankers?

Laten we eerst opmerken dat bovenstaande bezwaren wel worden ondervangen wanneer we wat extremer te werk gaan en simpelweg voor elk attribuut een eigen historietabel oprichten. In ons eenvoudige voorbeeld leidt dat dus tot de tabellen in afbeelding 3a/b. Dit ziet er goed uit: wanneer één attribuut wijzigt hoeft alleen in zijn eigen tabel een nieuwe rij te worden toegevoegd en daarbij worden waarden van andere attributen niet gekopieerd. Verder is dit extreem flexibel: het toevoegen van een attribuut is nu een kwestie van het creëren van een nieuwe tabel, zo simpel is dat. Deze manier van modelleren staat bekend onder de naam 'Anchor Modeling' (strikt genomen in combinatie met het idee van surrogaatsleutels) en kent fanatieke aanhangers. Ondanks dat deze bij hoog en laag volhouden dat dat (ook) goed is voor de performance heb ik daar toch mijn twijfels bij. Ik heb zelf ook wat testjes gedaan en het voeden van een datamart vanuit dergelijke structuren is erg complex en traag. Dat is ook wel begrijpelijk, want tientallen tabellen zullen moeten worden *gejoined* (ook nog via temporale joins) om bijvoorbeeld een gedenormaliseerde dimensietabel te voeden.

Goed, we laten deze 'Anchor Modeling' even voor wat het is en keren terug naar het historiseren van een genormaliseerde database. Voordat we richting Data Vault gaan koersen volgen nog enkele opmerkingen. De eerste betreft het verband tussen de tabellen: de verwijzende sleutels. Hier is natuurlijk wel een probleem mee, want alle primaire sleutels zijn uitgebreid met één kolom. Een voor de hand liggende gedachte is: dan breiden we alle foreign keys toch ook uit met een datumveld? Vanuit een rij (zeg rij A) verwijzen we dan naar een rij (zeg rij B) die een versie van een object voorstelt. Maar A is zelf ook een versie van een object en dit lijkt dus alleen maar goed te gaan als de tijdlijnen van deze beide objecten toevallig gelijk zijn. We gaan hier niet verder op door om twee redenen: 1. dit is al eens uitvoerig aan de orde gesteld door René Veldwijk en consorten in zijn artikelenreeks 'Tijd in de database'; 2. de introductie van datawarehouse sleutels vereenvoudigt dit probleem enigszins.

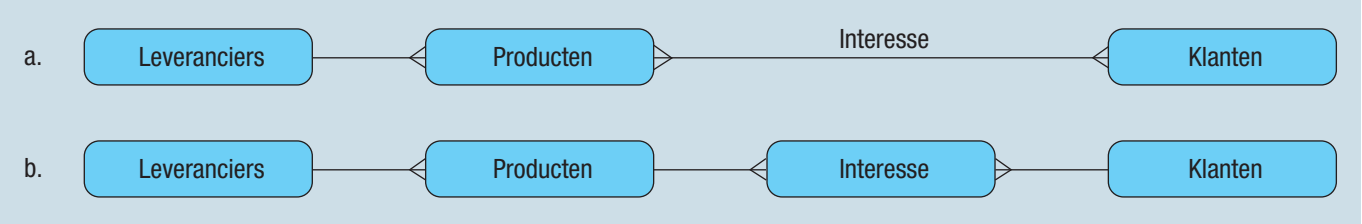
Kookboek

We gaan dit laatste (DWH Sleutels) dan ook doen en dit leidt tot een soort recept (men neme een business model ...) om van een entity relationship model een Data Vault model af te leiden. Belangrijk is om op te merken dat in dit model nog geen historie

Naam	Geldig van	Woonplaats
Jan	30	Den Haag
Jan	50	Leiden
Piet	10	Leiden

Naam	Geldig van	Burgerlijke staat
Jan	30	Ongehuwd
Jan	70	Gehuwd
Piet	10	Gehuwd

Afbeelding 3a en 3b: Elk attribuut krijgt een eigen historietabel.



Afbeelding 4: Entity relationship diagram.

is gemodelleerd. Dat gaan we immers bij de bereiding pas toevoegen:

1. Geef elk entiteittype een DWH Sleutel (DWH_KEY);
2. Maak per entiteittype één tabel om de één-op-één relatie tussen de business identificatie en de DWH Key op te slaan (de HUB-tabel);
3. Maak per entiteit een tabel waarin van de rest de historie wordt gestapeld (de Satelliettabel);
4. Wordt een entiteittype geïdentificeerd door relaties naar andere entiteittypen, maak dan een Linktabel.

Samenvattend: zodra we elke entiteit, naast de natuurlijke sleutel (die overigens ook complex kan zijn) binnen het datawarehouse ook gaan identificeren met een betekenisloos nummertje, dan hebben we in ieder geval één tabel nodig, waarin de één-op-één relatie tussen deze DWH Key en de business identificatie wordt vastgelegd (de HUB-tabel). We hebben dan 0 of meer satelliettabellen nodig om de historie van de overige attributen of vertrekkende relaties vast te houden. Punt 4 kan eigenlijk enigszins slordig worden geformuleerd als: veel-op-veel relaties leiden tot linktabellen.
Het wordt tijd om de procedure eens toe te passen op een voorbeeld. In afbeelding 4a is een ER-diagram gegeven. Wegens het voorkomen van de veel-op-veel relatie tussen 'Producten' en 'Klanten' (een klant kan interesse hebben getoond voor meer producten en voor één product kunnen natuurlijk meerdere klanten interesse hebben) is dit een echt ER-model en kan het niet de structuur van een RDBMS tonen. Wanneer men (zoals ik) een voorkeur heeft voor ER-diagrammen met alleen maar één-op-veel relaties, dan kunnen we dit diagram eenvoudig aanpassen door de veel-op-veel relatie 'Interesse' ook als een entiteittype te zien. Dit leidt tot afbeelding 4b. Het voordeel is dat het nu meteen ook te zien is als een diagram dat de structuur van tabellen en verwijzende sleutels weergeeft van een relationele database. Verder is het mijns inziens zo dat zuivere veel-op-veel relaties in de praktijk weinig voorkomen in die zin dat er vaak ook al erg interessante attributen bijhoren. In ons voorbeeld zou zo'n attribuut de mate van interesse kunnen aangeven (Piet is erg geïnteresseerd in een TV, ook wel in een radio maar dat is maar matig). Dergelijke attributen zouden dan bij de veel-op-veel relatie moeten worden genoteerd en er zijn inderdaad EAR-methoden die het zo moeilijk maken, maar dit is een andere discussie. Laten we daarom terugkeren naar ons kookboek. Het entiteittype 'Interesse' in afbeelding 4b wordt geïdentificeerd door de klant en het product. Met andere woorden: we kennen een 'interesse ding' zo gauw we de klant en het product kennen. De mate van interesse is dan een attribuut en dit kan in de loop van de tijd veranderen. Anders gezegd: als we de kookboekprocedure toepassen op afbeelding 4b dan krijgen we drie HUB-tabellen en één linktabel. Verder hangen we onder elke HUB één satelliettabel. Het resultaat is te zien in afbeelding 5. Merk op dat het ook is toegestaan om een satelliettabel te hebben onder een linktabel. Op deze wijze kunnen we bijvoorbeeld de historie van het attribuut 'mate van interesse' vastleggen (Piet was eerst maar matig geïnteresseerd in een radio, maar dat is veranderd: tegenwoordig wil hij er heel graag één hebben).

Een boeiende vraag is nu wat er gebeurd is met de één-op-veel relatie tussen Producten en Leveranciers uit afbeelding 4. Welnu, laten we veronderstellen dat we van alles per default historie willen bijhouden, dus ook van de leveranciers van een product in de loop van de tijd. Afbeelding 4 vertelt ons dat er op één moment in tijd hoogstens één leverancier is voor een gegeven product. Maar dat wil niet zeggen dat we in de loop van tijd niet af en toe switchen. De meest voor de hand liggende manier om dit te doen is om een verwijzende sleutel te maken van de tabel 'Producten SAT' naar 'Leveranciers HUB'. Met andere woorden: de satelliettabel van producten bevat een kolom die de DWH Key bevat van de leverancier die toen geldig was. Mocht dit veranderen dan moeten we dus een nieuwe rij opnemen, net zo goed als we dat moeten doen als een van de attributen van het product wijzigt.

De DV Standaard?

Het getoonde voorbeeld in afbeelding 5 voldoet echter niet aan de Data Vault Standaard[1]. Het is door Dan namelijk verboden om verwijzende sleutels te hebben vanuit een satelliettabel anders dan naar zijn eigen HUB-tabel. Laten we dit verbod even de *HUB isolatieregel* noemen. De achtergrond van deze regel is dat Dan een datawarehouse heel helder wil opdelen conform de HUB-tabellen. Dit zijn de schakels waarom het allemaal draait. Er zijn dus HUB-tabellen met satellieten. Dit zijn zelfstandige koninkrijkjes alleen met de buitenwereld verbonden via linktabellen. Op deze manier kan men de impact van veranderingen goed overzien. Betreft het een wijziging in één entiteittype dan beperkt de impact zich tot één zo'n koninkrijkje. Persoonlijk vind ik de regel discutabel. Wanneer men een Data Vault model modelgedreven genereert (zoals BIReady doet) dan is de impact

automatisch te overzien en af te handelen. Maar goed, laten we voor het moment gehoorzaam zijn en deze regel toepassen in onze volgende variant.
We gaan namelijk een tweede modelleringsvariant opzetten voor hetzelfde ER-voorbeeld uit afbeelding 4. We kunnen namelijk meer dan één satelliettabel hebben bij een gegeven HUB-tabel. Van de discussie in het begin van dit artikel weten we dat als het alleen maar om normalisatie gaat we mogen volstaan met hoogstens één satelliettabel per HUB-tabel. Er zijn echter verschillende redenen om er soms meer dan één te hebben. We zetten deze redenen even op een rijtje en wat mij betreft is dit ook de volgorde van belangrijkheid:

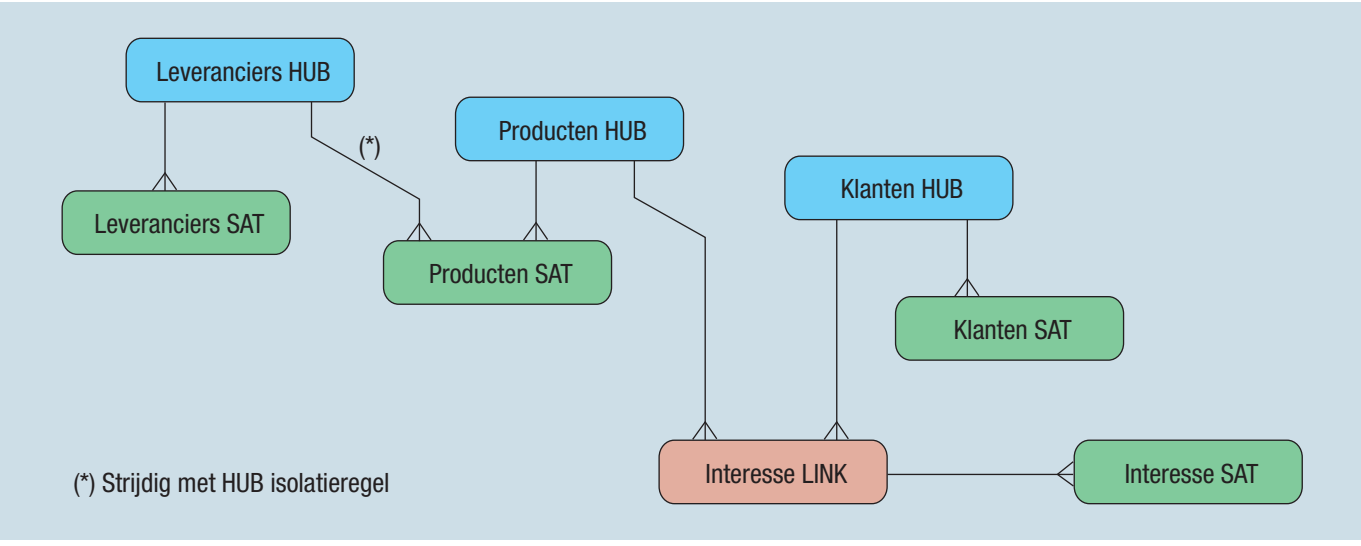
a. Snel veranderende versus langzaam veranderende attributen;
b. Uitbreiding datawarehouse;
c. Verschillende bronnen;
d. Voldoen aan de HUB isolatieregel (heet dan weer een linktabel).

We bespreken deze redenen even in het kort. Reden a zal duidelijk zijn. Als tengevolge van een paar vaak wijzigende attributen voortdurend kopieën gemaakt moeten worden van die attributen die bijna nooit wijzigen, dan is dat tamelijk onhandig. Het is dus duidelijk beter deze in aparte satellieten te hebben. Reden b is boeiend. Hierboven hadden we opgemerkt dat het probleem van het toevoegen van kolommen aan een historietabel, zoals in het eerdere artikel beschreven, nog steeds niet is opgelost. Welnu: don't! Indien men in een nieuwe release een aantal attributen wil toevoegen, voeg dan geen kolommen aan de bestaande satelliet toe maar richt gewoon een nieuwe op! Hiermee krijgen we dan wel wat meer satelliettabellen dan strikt nodig, maar zo erg als met de 'Anchor Modeling' methode is het in ieder geval nog niet. Op deze manier krijgen we de juiste mix tussen flexibiliteit (uitbreidbaarheid) en simpelheid. Mocht het datawarehouse door al deze uitbreidingen in de loop van de tijd wat versplinterd raken dan is het altijd nog mogelijk te zaak te reorganiseren. Het kost even tijd maar dan heb je ook wat. Dit is eigenlijk heel goed vergelijkbaar met het defragmenteren van uw vaste schijf als die wat traag is geworden. Er bestaat dus zoiets als het defragmenteren van uw datawarehouse. Punt c kan zijn voordelen hebben voor de laadperformance: de afzonderlijke satelliettabellen kunnen dan parallel worden geladen. Punt d tenslotte. Door de verwijzing naar een andere HUB-tabel op te nemen in zijn eigen satelliet, kunnen we voldoen aan de HUB isolatieregel. Wel moeten we deze satelliet dan een linktabel noemen.

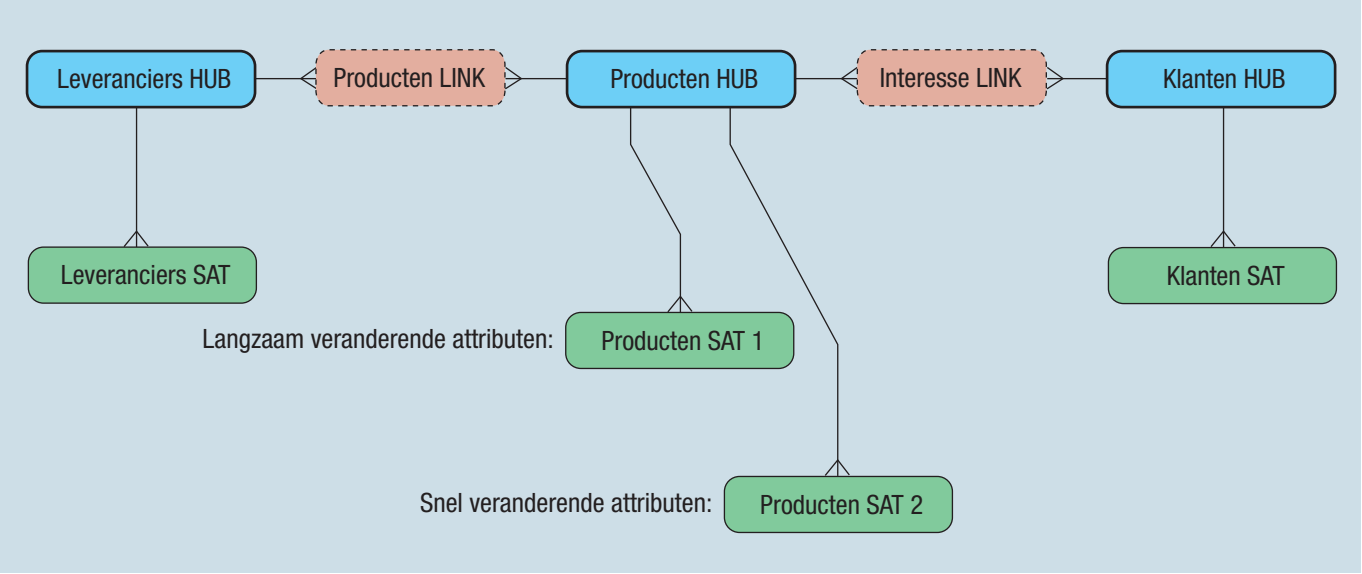
In afbeelding 6 is een tweede variant van een Data Vault model gegeven. Hierin hebben we meerdere satellieten onder de 'Producten HUB' gehangen (en er één weer linktabel genoemd). We hebben in deze variant dus voorbeelden van de toepassing van a en d.

Links: linke soep

Voldoet dit model nu wel aan de standaard? Oppervlakkig bezien wel, als we alleen naar de ER-structuur van afbeelding 6 kijken. Het lijkt erop dat beide linktabellen nu een veel-op-veel relatie representeren. Er is echter een belangrijk verschil. De 'Interesse LINK' representeert een veel-op-veel relatie, simpel omdat die dat altijd al was. De 'Producten LINK' is echter veel-op-veel geworden, omdat het tijdsaspect is toegevoegd. In de loop van de tijd kan er immers sprake zijn van meerdere leveranciers voor één product, terwijl dat volgens de afbeelding er op één moment in tijd maar hooguit één kan zijn. Dit verschil wordt dan ook pas duidelijk, als we in de interne structuur van de twee linktabellen kijken. De 'Producten LINK' bevat de volgende velden (primaire sleutel vetgezet):
PRODUCT_DWH_KEY (verwijzing naar de 'Producten HUB')
VANAF_DATUM
TOT_DATUM
LEVERANCIERS_DWH_KEY (verwijzing naar de 'Leveranciers HUB')



Afbeelding 5:Verwijzende sleutels.



Afbeelding 6: Tweede variant van een Data Vault model.

De interne structuur van 'Interesse LINK' kan er als volgt uit zien (primaire sleutel weer vetgezet en de alternatieve sleutel gecursiveerd):
INTERESSE_DWH_KEY (hiernaar wordt verwezen vanuit de Interesse satelliet)
PRODUCT_DWH_KEY (verwijzing naar de 'Producten HUB')
KLANTEN_DWH_KEY (verwijzing naar de 'Klanten HUB')

Nu blijkt dat dit volgens de Data Vault standaard niet mag. Een quote van Dan Linstedt zelf: "In the Data Vault standard there is only 1 type of Link table: a Link table". Met andere woorden; alleen de interne structuur zoals die van 'Interesse LINK' is toegestaan. Dit komt volgens de filosofie achter Data Vault voort uit het feit dat je zo weinig mogelijk 'business rules' moet gebruiken bij het ontwerp. Het idee is dat dergelijke regels ook kunnen veranderen in de loop van de tijd. Zo kunnen we op een gegeven moment besluiten dat we op één moment in tijd toch met meerdere leveranciers van één product willen gaan werken. Het 'business model' van afbeelding 4 wijzigt dus. Hoeven we ons Data Vault model toch maar mooi niet aan te passen. Zo lust ik er echter nog wel een paar en ik wil dit dan ook ter discussie stellen. Ten eerste moeten we ons realiseren dat we natuurlijk volop gebruik maken van bedrijfsregels zodra we een relationele database ontwerpen (en dat doen we als we een Data Vault model maken) of we het nu leuk vinden of niet. Als we geen beperkingsregels willen dan kunnen we alle relevante bedrijfsgegevens beter in een Word document opslaan, dat dwingt tenminste niets af, op de spellingchecker na. Laat ik een voorbeeld geven waar we in ieder geval al regels gebruiken. Eigenlijk is het hele concept van satelliet al gebaseerd op dit soort aannamen. Immers, als we de historie van bijvoorbeeld de postcode van een klant bijhouden in één van de satelliettabellen, dan bevat die tabel een begin/einddatum (tijd) constructie. We nemen dus kennelijk aan dat op één moment in tijd er hoogstens één postcode bij die klant hoort. Wat wanneer deze 'business rule' verandert?

Het lijkt erop dat men in Data Vault niet houdt van één-op-veel relaties, zo snel als het om de verbanden tussen de HUB's (zijnde de belangrijke business entiteittypen) gaat. Wanneer we echter denken aan de toepassingen en dus de datamarts die bovenop het datawarehouse moeten gaan komen, dan hebben we niet veel aan veel-op-veel. Als ik immers de totale omzet van klanten per regio wil weten, dan gaat dat toch echt alleen maar lukken als ik er zeker van kan zijn dat op één moment in tijd een klant zich slechts in één regio bevindt. Zelfs als er echt sprake is van een veel-op-veel relatie, bijvoorbeeld als er bij een verzekerde auto meerdere berijders staan vermeld, wil men naar één-op-veel. Je zou bijvoorbeeld graag willen weten hoeveel dure auto's er per leeftijdscategorie zijn verzekerd. In de praktijk kwamen we hier goed weg omdat we dankzij een indicatie 'hoofdberijder' toch een één-op-veel relatie konden realiseren van auto naar persoon (en dus leeftijd). Met andere woorden: OLAP en reporting houden van één-op-veel.

Samenvatting

We hebben gezien hoe we een Data Vault model kunnen afleiden uit een business model door de ideeën van datawarehouse sleutels en historiseren te combineren. De begrippen HUB- en satelliettabel komen daarbij vrij natuurlijk naar boven. Het begrip linktabel is wat lastiger en daarover is mijns inziens nog wel discussie mogelijk.[2]

Noten

1. Deze standaard is (nog) niet formeel publiekelijk beschreven.
2. Met dank aan de discussies met de Data Vault deskundigen van Nippur.

Harm van der Lek

Dr. H. van der Lek is onafhankelijk consultant en docent.