Anchor Modeling

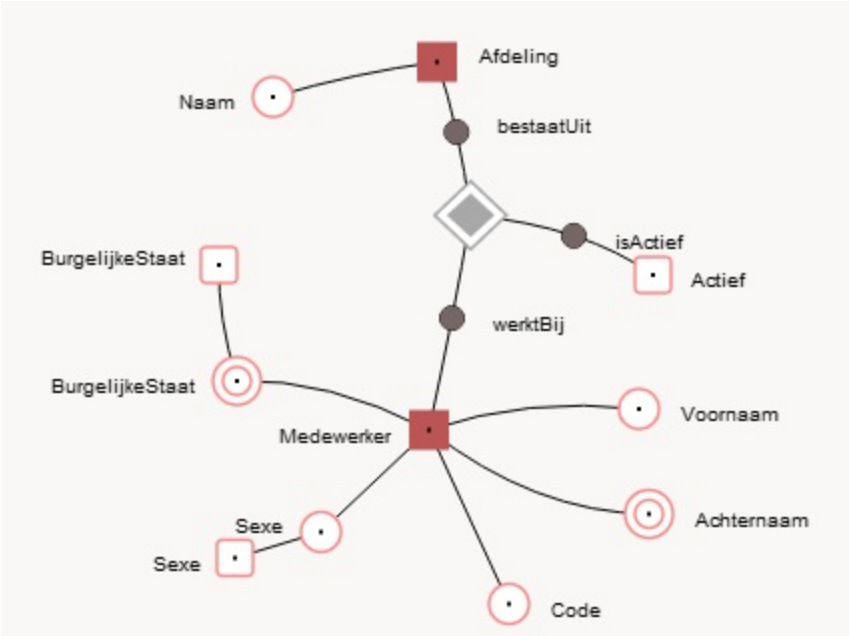
De sleutel voor een toekomstvast en flexibel datawarehouse is een goed doordacht informatiemodel. De beslissingen die genomen worden bij het maken van het informatiemodel zijn bepalend voor het slagen van het datawarehouseproject. De praktijk wijst echter uit dat het meestal niet mogelijk is om bij de start van een project een volledig dekkend informatiemodel te maken, en dat het model gedurende de levensduur van het datawarehouse regelmatig aangepast en uitgebreid zal moeten worden.

Dit is ook precies de reden dat 'agile datawarehousing'-methodieken in opkomst zijn - het informatiemodel moet gemakkelijk en snel uitbreidbaar zijn. De bekendste hiervan is Data Vault (DV) van Dan Linstedt, maar in dit artikel willen we aandacht besteden aan een andere recente en zeer veelbelovende methodiek: Anchor Modeling (AM) [1]. We zullen kort bespreken wat AM precies is en vervolgens ingaan op de voor- en nadelen.

Wat is Anchor Modeling?

Eind juni kwam Lars Rönnbäck naar Nederland om te vertellen over Anchor Modeling (AM): een modelleertechniek waarin het informatiemodel wordt genormaliseerd in de zesde normaalvorm. Simpel gezegd betekent de zesde normaalvorm dat elke tabel maar hooguit één domeingebonden attribuut kan bevatten. Een medewerker met een voornaam, achternaam, geslacht, burgerlijke staat en een unieke code (business key) wordt gemodelleerd als zes tabellen.

De eerste tabel, de anchor, geeft een surrogaatsleutel aan de medewerker, de zogenaamde identity. De overige vijf tabellen verwijzen elk één-op-één naar de anchor en bevatten de afzonderlijke attributen, eventueel nog aangevuld met een ValidDate die aangeeft vanaf wanneer het attribuut geldig is geworden. Business-keys uit het domein worden tevens als attribuut gemodelleerd en dus niet in de anchor zelf (zie afbeelding 1).



In de database ziet het er als volgt uit wanneer we alleen de anchor

Medewerker en het attribuut Sexe laten zien:

ME_ID
1
2
3

ME_ID	ME_SEX_Medewerker_Sexe
1	MAN
2	MAN
3	VROUW

Wanneer vaak dezelfde string herhaald wordt in een attribuut, dan kan er voor gekozen worden om een *knot* aan te leggen, vergelijkbaar met een lookup-tabel. De knot bevat een lijstje met opzoekwaardes, en de attribuuttabel bevat een foreign-keyverwijzing naar de knot-tabel. Dezelfde knot kan door meerdere attributen gebruikt worden. Een *knotted attribuut* ziet er als volgt uit:

ME_ID
1
2
3

ME_ID	SEX_ID
1	1
2	1
3	2

SEX_ID	SEX_Sexe
1	MAN
2	VROUW

Ook een knotted attribuut kan eenvoudig 'gehistoriseerd' worden door een ValidDate toe te voegen. Bijvoorbeeld de burgerlijke staat van een medewerker:

ME_ID	BST_ID	ValidDate
1	100	1-1-1900
2	100	1-1-1900
2	200	1-10-2010

BST_ID	BST_BurgerlijkeStaat
100	Alleenstaand
200	Getrouwd

Ties

Stel dat we willen vastleggen bij welke afdeling een medewerker werkt. Dit wordt gedaan door middel van een tie. Een tie bevat alleen de verwijzingen naar de anchors.

Bij het aanmaken van een tie moet ook een naam worden gegeven aan de rol tussen Medewerker en Afdeling (werktBij) en vice versa (bestaatUit). Laten we als voorbeeld een medewerker nemen met de naam Albert Schrijfgraag. Albert werkt voor Personeelszaken (PZ), maar werkt ook één dag per week voor de afdeling Finance. Er zullen twee records aanwezig zijn in de tie voor beide relaties. We willen ook weten hoe lang Albert al voor een afdeling werkt. Dit wordt gerealiseerd door de tie te 'historiseren' en een ValidDate toe te voegen. Albert werkt al sinds januari 2008 voor PZ en sinds januari 2009 tevens voor Finance. Per januari 2010 stopt Albert met het werken voor Finance. Om dit te kunnen modelleren maakt AM geen gebruik van een einddatum, maar van een ‘actief’-vlag die als knot wordt toegevoegd aan de tie. Afdeling 100 is PZ en afdeling 200 is Finance. Het interval van PZ is open en het interval van Finance is gesloten op januari 2010. De data in de tie komt er dan als volgt uit te zien:

ME_ID_werktBij	AF_ID_bestaatUit	ACT_ID_isActief	ME_werktBij_AF_bestaatUit ACT_isActief_ValidFrom
1	100	1	1-1-2008
1	200	1	1-1-2009
1	200	0	1-1-2010

Metadata
Elke anchor-, attribuut-, tie- en knot-tabel krijgt standaard een metadatakolom, die automatisch wordt gegenereerd door de modelleertool. In deze kolom staat een verwijzing naar een metadatatabel waarin gegevens kunnen worden opgeslagen over de batch, zoals het tijdstip van de batch, de uitvoerder en het bronsysteem.

Tijdlijnen
Een van de sterke punten van AM is dat het heel gestructureerd omgaat met verschillende tijdlijnen, en naadloos aansluit bij het 'bitemporal data'-concept [2, 3]. Bitemporal data gaat uit van twee tijdlijnen, namelijk Valid Time en Transaction Time. De Valid Time is de effectieve tijd waarop een attribuut of relatie geldig wordt, bijvoorbeeld de datum waarop een medewerker is getrouwd. De Transaction Time is een systeemtijd die aangeeft wanneer een attribuut of een relatie is ingevoerd of gewijzigd in het systeem (of is geladen in het datawarehouse), bijvoorbeeld de datum waarop de trouwdatum van de medewerker is ingevoerd. (Vaak wordt een ‘slowly changing’-dimensie ten onrechte gebaseerd op de Transaction Time, terwijl eindgebruikers meer geïnteresseerd zijn in de Valid Time; denk aan een query waarbij het aantal medewerkers moet worden afgezet tegen de burgerlijke staat).

In een AM-model wordt de Valid Time aangeduid als *changing time*, deze correspondeert met de ValidDate die in elke attribuut en tie kan worden opgenomen. De Transaction Time wordt aangeduid als *recording time*, deze correspondeert met de invoerdatum of met de laaddatum van de ETL-job die in het metadatarecord is opgeslagen. Een groot voordeel is dat per attribute en per tie een changing time kan worden opgegeven, dit in tegenstelling tot minder genormaliseerde modellen waarin voor een reeks attributen een changing time moet worden opgegeven, waardoor het niet duidelijk is welk attribuut is veranderd.

Naast changing time en recording time onderscheidt AM nog een derde type tijdlijn, namelijk *happening time*. Dit is het tijdstip van een eenmalige gebeurtenis, zoals de geboortedatum van een medewerker of de orderdatum.

Viewlaag
Er worden standaard voor iedere anchor drie typen views gegenereerd, die op een eenduidige manier rekening houden met het tijdsaspect:

1. De *latest view* van een anchor geeft een tabelformaat terug met de laatste versies van alle attributen van die anchor. Voor onze Medewerker-anchor ziet de latest view er als volgt uit (ervanuit gaande dat alleen achternaam en burgerlijke staat gehistoriseerd zijn):

```
CREATE VIEW lME_Medewerker
AS
SELECT  m.ME_ID, mv.ME_Medewerker_VNA_Voornaam, ma.ME_Medewerker_ANA_Achternaam,
        ms.ME_SEX_ID, s.SEX_Sexe, ms.ME_SEX_ValidDate,
        mb.ME_BST_ID, b.BST_BurgerlijkeStaat, mb.ME_BST_ValidFrom
FROM    ME_Medewerker m
        LEFT JOIN ME_Medewerker_VNA_Voornaam mv ON mv.ME_ID = m.ME_ID
        LEFT JOIN ME_Medewerker_ANA_Achternaam ma ON ma.ME_ID = m.ME_ID
        LEFT JOIN ME_Medewerker_SEX_Sexe ms ON ms.ME_ID = m.ME_ID
        LEFT JOIN SEX_Sexe s ON s.SEX_ID = ms.ME_SEX_ID
        LEFT JOIN ME_Medewerker_BST_BurgerlijkeStaat mb ON mb.ME_ID = m.ME_ID
        LEFT JOIN BST_BurgerlijkeStaat b ON b.BST_ID = mb.ME_BST_ID
```

```
WHERE   ma.ME_ANA_ValidDate = (SELECT MAX(ME_ANA_ValidDate)
        FROM ME_Medewerker_ANA_Achternaam) WHERE ME_ID = ma.ME_ID)
 AND    mb.ValidDate = (SELECT MAX(ME_BST_ValidDate)
        FROM ME_Medewerker_BST_BurgerlijkeStaat) WHERE ME_ID = mb.ME_ID)
```

2. De *point-in-time view* geeft hetzelfde tabelformaat terug, maar dan met de versies van de attributen op een gegeven tijdstip. Hiervoor kan een geparametriseerde view ofwel een table-valued function gebruikt worden, met als enige parameter @Date, het gevraagde tijdstip. Deze ziet er net zo uit als de latest view, behalve dat de subselect op ValidDate een extra conditie bevat:

```
CREATE FUNCTION pME_Medewerker (@timepoint datetime)
AS
-- (SELECT en FROM clause zijn identiek aan lME_Medewerker)
WHERE   ma.ME_ANA_ValidDate = (
                SELECT MAX(ME_ANA_ValidDate)
                FROM ME_Medewerker_ANA_Achternaam
                WHERE ME_ID = ma.ME_ID AND ME_ANA_ValidDate <= @timepoint)
 AND    mb.ValidDate = (
                SELECT MAX(ME_BST_ValidDate)
                FROM ME_Medewerker_BST_BurgerlijkeStaat
                WHERE ME_ID = mb.ME_ID AND ME_BST_ValidDate <= @timepoint)
```

3. De *interval view* geeft een tabelformaat met alle veranderingen van de attributen gedurende een bepaalde periode. In tegenstelling tot de latest view en de point-in-time view kan deze view meerdere records per medewerker bevatten, met verschillende op elkaar aansluitende tijdsintervallen.

De voor- en nadelen van Anchor Modeling

AM is een methode en een online modelleertool [1]. De methode formaliseert best practices die voor een deel al lang bekend zijn, maar die in de praktijk nogal willekeurig worden toegepast, afhankelijk van de ontwikkelaar. De tool dwingt de gebruiker om consistent de methode te volgen. Een metadataverwijzing wordt bijvoorbeeld automatisch gegenereerd en het is bovendien niet mogelijk om naamgeving te gebruiken die afwijkt van de conventie.

Doordat in de zesde normaalvorm wordt gemodelleerd wordt de ontwerper gedwongen om grondig na te denken over alle entiteiten en de relaties daartussen. In een minder genormaliseerd model wordt dit minder afgedwongen en wordt het verleidelijker om stukken van het datamodel van het bronsysteem over te nemen, met mogelijk als consequentie dat er business rules over het hoofd gezien worden (of juist onnodige velden worden overgenomen).

Laten we beginnen met het in onze ogen grootste voordeel: datakwaliteit, gevolgd door een mogelijk nadeel: gebruiksvriendelijkheid en performance van het datamodel.

Datakwaliteit
Kan het domein compleet en correct worden gerepresenteerd? Zijn meetwaarden herleidbaar tot brongegevens?

AM hanteert een zogenaamde ‘zero-update’-strategie. Dit betekent dat alles wordt gerealiseerd met insert statements. Er kan hooguit iets verwijderd worden wanneer zeker is dat de data

foutief is, maar er kan ook voor gekozen worden om de foutieve data te markeren in plaats van te verwijderen. De update-permissie kan volledig uitgezet worden op databaseniveau. Het gevolg hiervan is dat er geen fouten gemaakt kunnen worden met het updaten van data, zoals bij het updaten van een slowly changing-dimensie wel zou kunnen gebeuren. Wanneer een gebruiker de vraag stelt ‘Vorige week kwam er dit getal uit het rapport en nu dit getal. Hoe komt dit?’ dan kun je altijd de gegevens opvragen waarop het rapport een week geleden was gebaseerd. Ook het afsluiten van maanden wordt gemakkelijker omdat wijzigingen met terugwerkende kracht inzichtelijk worden (zie ook de alinea 'Tijdlijnen').

Het terugdraaien van een foutieve batch die updates uitvoert in het datawarehouse kan een behoorlijke uitdaging zijn. In een AM-model kunnen door middel van de metadata koppeling eenvoudig alle records worden verwijderd die relateren aan deze batch.

Gebruiksvriendelijkheid
Hoe makkelijk kunnen (ETL) ontwikkelaars en analisten ermee werken?

Een mogelijk probleem van AM is dat er een enorme hoeveelheid tabellen ontstaat. Het zal voor gebruikers van het datawarehouse lastiger zijn om overzicht te houden op databaseniveau en een ETL-ontwikkelaar zal veel meer tabellen moeten vullen en vergelijken. Dit probleem wordt echter aanzienlijk gereduceerd door de gebruiksvriendelijke visuele modelleertool die een mooie grafische weergave van het AM-model geeft, welke door domeinspecialisten gelezen en gevalideerd kan worden. De naamgevingsconventie zorgt er voor dat gerelateerde tabellen alfabetisch onder elkaar komen te staan, zodat een ontwikkelaar niet lang hoeft te zoeken naar de juiste tabel. De viewlaag op het AM-model zorgt er voor dat er niet telkens gejoind hoeft te worden met alle attribuuttabellen van een anchor (zie ook de alinea 'Viewlaag').

Om de gebruiksvriendelijkheid voor analisten te verbeteren, kan een datamartlaag met Kimball-stermodellen gebaseerd worden op de viewlaag. De slowly changing (type-2) dimensies kunnen dan opgebouwd worden met behulp van de interval-views (zie tabel 1).

Tabel 1: Slowly Changing Dimensie voor Medewerker

MedewerkerID	Voornaam	Achternaam	Geslacht	BurgerlijkeStaat	ValidDate
1	John	Dundee	Man	Alleenstaand	1-1-0001
2	Jan	Klaassen	Man	Alleenstaand	1-1-0001
2	Jan	Klaassen	Man	Getrouwd	1-10-2010
3	Marieke	Janssen	Vrouw	Alleenstaand	1-1-0001
3	Marieke	Klaassen	Vrouw	Getrouwd	2-2-2002

Performance en schaalbaarheid
Kunnen grote datasets met voldoende performance worden geladen en uitgelezen?

De eerste reactie van een BI-ontwikkelaar is vaak: **‘Dat kan toch nooit snel zijn? Als je alle medewerkergegevens opvraagt, dan moeten al die tabellen weer gejoind worden.’** Dit blijkt nogal mee te vallen vanwege drie redenen.

1. Allereerst hebben alle attribuuttabellen de clustered index op de anchor key, waardoor de joins erg efficiënt zijn. Bovendien bevatten de query-optimizers van vrijwel alle moderne RDBMS-systemen de ‘table elimination’-feature. Deze feature zorgt er voor dat indien er in de select clause van een query geen enkele kolom wordt opgevraagd uit een tabel die gejoind wordt, en die join ook geen invloed heeft op het aantal rijen, dat dan de join niet wordt uitgevoerd. Als je bijvoorbeeld de volgende query uitvoert dan zal de query optimizer alleen de join met de Voornaam en Achternaam attribuuttabellen uitvoeren, terwijl er in de view gejoind wordt met alle attribuuttabellen (zie de alinea 'Viewlaag' voor de definitie van de view lME_Medewerker).

```
SELECT      ME_ID, ME_Medewerker_VNA_Voornaam, ME_Medewerker_ANA_Achternaam
FROM        lME_Medewerker
```

2. Daarnaast is een AM-model optimaal in IO-gebruik in vergelijking met minder genormaliseerde modellen. Dit komt onder andere doordat in een anchor model geen NULL-waardes voorkomen. Wanneer bijvoorbeeld bij een medewerker alleen de achternaam wordt ingevuld, dan zal alleen de attribuuttabel voor achternaam een rij bevatten. In een minder genormaliseerd model zal een record worden aangemaakt waarbij ruimte wordt gereserveerd voor alle attributen, maar waarbij alleen achternaam een waarde bevat. Wat tabel 1 tevens goed laat zien, is dat de redundantie in een ‘slowly changing’ (type-2) dimensie behoorlijk kan oplopen omdat hier wijzigingen van een attribuut ‘gestapeld’ worden. Stapelen betekent dat de records van een tabel worden geversioneerd met een begin- en einddatum, zodat bij een gewijzigd attribuut een nieuw record wordt aangemaakt met een begindatum (en het oude record wordt afgesloten). Het gevolg hiervan is dat er een kopie wordt gemaakt van alle ongewijzigde attributen. In een AM model worden wijzigingen per attribuut opgeslagen en gebeurt het stapelen in de interval view.

3. Een full table scan zal over het algemeen sneller zijn in AM dan in een minder genormaliseerd model omdat de tabellen kleiner zijn. De geheugenpagina’s zijn hierdoor kleiner en de benodigde data is normaal gesproken minder verspreid over de harddisk. Wanneer vervolgens toch alle attributen worden opgevraagd, dan kunnen deze op basis van de primary key gejoind worden.

Enkele nadelen met betrekking tot performance:
1. De anchor keys worden in elke attribuuttabel herhaald.

2. De latest view en de point-in-time view maken gebruik van geneste queries, waarvan bekend is dat ze een mindere performance hebben. Volgens de auteur blijkt dit echter mee te vallen. En doordat de clustered index ligt op de anchor key in combinatie met de ValidDate wordt de data gesorteerd opgeslagen en zal de latest view relatief efficiënt zijn.

Flexibiliteit

Hoe makkelijk kunnen wijzigingen in het domein of in de bronsystemen worden doorgevoerd?

AM is 'domain-driven' en niet 'source-driven'. Dit betekent dat veranderingen in bronsystemen vaak alleen gevolgen hebben voor de ETL en niet voor het database schema en alles wat daarvan afhankelijk is.

Het voordeel van AM is dat er makkelijk een code-generatietool kan worden gebruikt. Voor een nieuw attribuut zal altijd een nieuwe tabel worden aangemaakt. Omdat de generatietool alleen maar tabellen toevoegt aan de database is er ook geen risico dat er iets verloren gaat tijdens het bijwerken van de database. Alle ETL-code en queries zullen gewoon blijven werken na het uitvoeren van een schema-update.

Levensduur
Wanneer is het DWH verouderd en dient het te worden vervangen (en waarom gebeurt dit niet in de praktijk)?

De claim die AM doet is dat de levensduur van een datawarehouse toeneemt. Dit komt met name omdat AM beter om kan gaan met veranderingen. Bijvoorbeeld veranderingen in bronsystemen of veranderingen in de requirements van eindgebruikers. Bij het modelleren worden zo min mogelijk assumpties gedaan over hoe het datawarehouse gebruikt wordt, maar wordt het domein gemodelleerd zoals het is. Deze claim kan helaas nog niet bevestigd worden omdat AM nog niet lang wordt toegepast.

Conclusie
Als datawarehouse-ontwerpers zijn wij zeer enthousiast over de voordelen die Anchor Modeling biedt. Qua datakwaliteit scoort AM zeer goed omdat meerdere tijdlijnen worden ondersteund en omdat gegevens nooit worden weggegooid. Doordat gemodelleerd wordt in de zesde normaalvorm wordt de ontwerper gedwongen om goed na te denken over alle entiteiten in het domein en de relaties daartussen. Bovendien heeft de zesde normaalvorm als voordeel dat er vrijwel geen redundantie is en dat het datamodel eenvoudig kan worden uitgebreid zonder dat tabellen hoeven te worden aangepast. Er ontstaan echter wel heel veel tabellen wat mogelijk lastig is in het gebruik, maar met de zeer gebruiksvriendelijke online modelleertool wordt het creëren van een nieuw datamodel en het wijzigen van een bestaand datamodel erg gemakkelijk gemaakt. Een ander voordeel van de modelleertool is dat best practices kunnen worden afgedwongen, zoals het bijhouden van metadata of het hanteren van een naamgevingsconventie.

De praktijk moet uitwijzen of met name de performance voldoende is of wellicht zelfs beter. Het is bekend dat IO vaak de bottleneck is in een datawarehouse-omgeving en op dit gebied zou AM beter moeten presteren. Ook in een cloud-omgeving zou AM interessant zijn omdat er minder data over de lijn gaat. In Zweden zijn er reeds een aantal succesvolle implementaties gedaan, maar in Nederland is het wachten nog op de eerste succesvolle implementatie. Met name voor grote organisaties en in situaties waar het informatiemodel steeds verder evolueert lijkt AM zeer geschikt.

Op het forum [1] vindt u interessante discussies over het gebruik van AM en mogelijke uitbreidingen.

Bas van den Berg (bas@c2h.nl) is freelance BI/DWH Consultant.
Jorg Jansen (jorg.jansen@kadenza.nl) is BI/DWH Consultant at Kadenza.

Bibliografie

1. **Rönnbäck, Lars.** Anchor Modeling. [Online] http://www.anchormodeling.com.
2. **Richard T. Snodgrass.** A Case Study of Temporal Data**.** [Online] http://temporaldata.com/forum_uploads/7037e26d-66a2-43bd-919a-e8e8d5b486c7.pdf
3. **Maarten Zaanen**. Historie in het platte vlak. Database Magazine, februari 2011.