Automatiseren met Transformatie View Enhancer is effectief en flexibel

Scriptgenerator met ingebouwde intelligentie

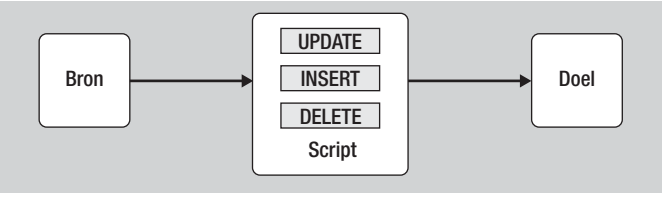
Marcel Witteveen

Database-systemen houden bij welke gegevens op welke wijze worden opgeslagen. Dankzij introspectie via metadata zijn deze systemen in staat om hun taken zo goed en efficiënt mogelijk uit te voeren. Dit artikel beschrijft hoe deze metadata ook gebruikt kunnen worden om op eenvoudige wijze het transport en de transformatie van gegevens efficiënt vorm te geven.

De beschreven functionaliteit is in het bijzonder geschikt voor het periodiek verwerken van gegevens in datawarehouse-omgevingen. Laden van gegevens in een database lijkt eenvoudiger dan het is. Een database is normaliter voorzien van allerlei beperkende maatregelen (constraints) die eisen stellen aan de volgordelijkheid van inlezen en de kwaliteit van de aangeboden gegevens. Moderne BI-omgevingen komen niet meer weg met het aloude credo 'garbage in, garbage out'. Typische datawarehouse-modelleringstechnieken, zoals snowflakes en starschemes (dimensionele modellen) vereisen bepaalde mate van intelligentie in het transport van gegevens. Dit is dan ook een van de vele uitdagingen voor een BI/Datawarehouse-team; het op intelligente wijze inlezen van vervuilde of incomplete gegevens en deze te transformeren naar bruikbare informatieproducten met behoud van historie.

Transparantie

Een deel van de problemen betreffende gegevenskwaliteit en de dimensionele modellen is te automatiseren. Door verdergaande automatisering kunnen de ontwerper en de ontwikkelaar zich meer concentreren op de bedrijfslogica, komen fouten minder vaak voor en is de oplossing zowel robuust als flexibel.
Een eenvoudige scriptgenerator, het gebruik van 'transformatie views', het generiek laden van gegevens en de uitbreiding hierop, de 'transformation view enhancer' zijn een uitstekende combinatie voor het weg-automatiseren van technische zaken die de ontwikkelaar afhoudt van de echte bedrijfslogica. Ook biedt het concept transparantie voor de eindgebruiker, die slechts op hoofdlijnen wil aangeven wat zijn kwaliteitscriteria zijn voor gebruik van behandeling van gegevens.
Het handmatig aanmaken van SQL-scripts is helaas nog steeds aan de orde van de dag. Door slim gebruik te maken van de informatie die een database zelf al verstrekt, is het mogelijk dit werk te minimaliseren door hiervoor automatisch code te laten genereren. Dit is het werk van de scriptgenerator. De scriptgenerator maakt hiervoor veelvuldig en dankbaar gebruik van de informatie die de onderliggende database biedt.



Afbeelding 1: Eenvoudige scriptgenerator.

De verschillende DML statements (INSERT, UPDATE en DELETE) zijn te genereren indien de bron en het doelobject bekend is. Het meest eenvoudige statement voor het kopiëren van gegevens uit twee identieke tabellen is het INSERT statement, waarbij een lege tabel gevuld wordt door een identieke brontabel. Deze transformatie heeft duidelijk niet veel om het lijf. Maar het gaat bijvoorbeeld direct mis als de twee objecten (klant en dimensie_klant), zoals in het volgende voorbeeld, niet dezelfde structuur hebben. Om de Dimensie_Klant tabel te verversen met gegevens uit de brontabel Klant is meer informatie nodig, zoals de structuur van de tabellen zelf en de opbouw van de unieke sleutel.

```
1  UPDATE    Dimensie_Klant
2  SET       naam = bron.naam, adres = bron.adres
3  FROM      klant_dimensie, klant
4  WHERE     Klant_dimensie.nummer = klant.nummer
```

De sleutel van de klant (nummer) komt niet voor in de SET-clause (2), maar wel in de WHERE-clause (4). Voor het insert-script geldt hetzelfde. Dit komt omdat het nummer in dit voorbeeld onderdeel is van de primary key. Hetzelfde is te zien bij het volgende INSERT statement. De volledige lijst van kolommen komt zowel

voor in regel 1 als in regel 2. Regel 4 gebruikt de enkelvoudige sleutelkolom 'nummer' om geen doublures toe te voegen.

```
1 INSERT INTO Dimensie_Klant (nummer, naam, adres)
2 SELECT      nummer, naam, adres
3 FROM        Klant
4 WHERE       Nummer not in (select nummer from
                                    klant_dimensie)
```

De systeemtabellen leveren voldoende informatie op voor het volledig automatisch genereren van deze scripts. In deze voorbeelden hoeft de generator slechts op de hoogte te zijn van de structuur van de bron- en doeltabellen en de opbouw van de unieke sleutel. De databases bieden deze informatie in de vorm van lijsten en tabellen. Door lijsten te combineren, bij elkaar op te tellen of de gezamenlijke elementen uit te sluiten zijn nieuwe lijsten (van kolommen) te genereren. De SET-clause bevat bijvoorbeeld geen kolommen die ook binnen de tabelsleutel voorkomen.
Door de elementen uit een lijst aan elkaar te plakken met een scheidingsteken, zoals een komma of het woord 'AND', zijn bruikbare tekstfragmenten op te bouwen. Door toepassing van zoek-en-vervang acties in voorgedefinieerde sjablonen, ontstaan vervolgens de juiste SQL scripts. Het gebruik van sjabonen blijkt uitermate effectief te zijn; ze voorzien in alle mogelijke SQL statements, database-dialecten en generieke scripts. De generator hoeft alleen nog maar tekstfragmenten te genereren en deze op de juiste plekken in de sjablonen te plaatsen.
In feite is elk soort script hiermee te genereren. De scripts hoeven zich dus niet te beperken tot de SQL statements Insert, Update en Delete. De uitkomst wordt bepaald door de scripts. Zo is het mogelijk om van te voren audit trails toe te voegen, zonder hiervoor intelligentie in de scriptgenerator in te bouwen. Het is allemaal een kwestie van de juiste sjablonen gebruiken.

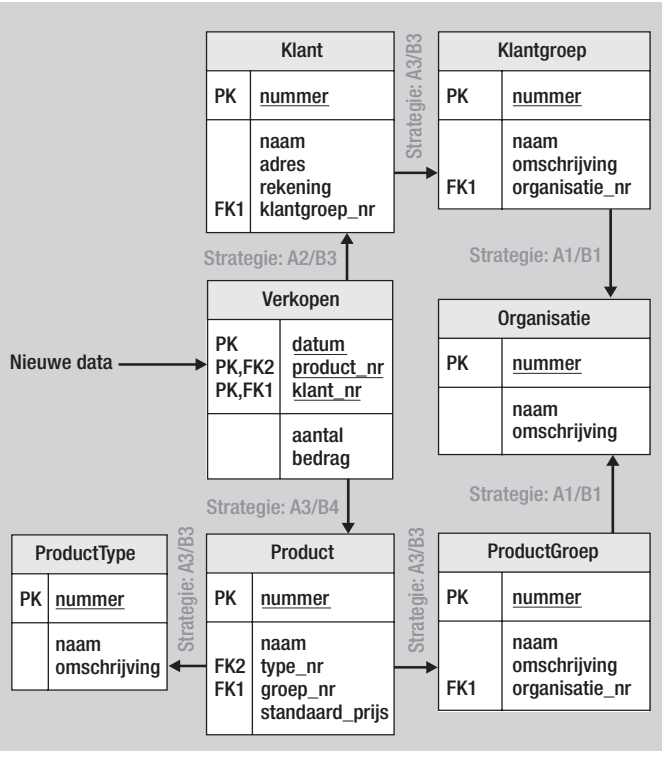
Transformatie views

De eenvoud van het voorgaande komt helaas niet vaak voor in de praktijk. Een tabel is slechts zelden een-op-een te 'mappen' naar een doeltabel. Om dit probleem te ondervangen volgt een korte beschrijving van het concept 'transformatie view'. Een transformatie view is niets meer of minder dan een view op nul of meerdere tabellen. De structuur van views lijkt sterk op die van tabellen en is op dezelfde manier opvraagbaar. Dit maakt het mogelijk views in plaats van tabellen als bron te gebruiken. De business-logica wordt geïmplementeerd binnen views.
Het concept van een transformatie view is niet nieuw. Vanwege de herbruikbaarheid worden ze al regelmatig toegepast in ETL-processen (mappings), zonder de naam 'transformatie view' te dragen. De informatie die de database verschaft over deze views gaan we echter veel verder exploiteren. Het gebruik van (transformatie) views kent wel een aantal nadelen. De systeembelasting komt bij de database te liggen en de mogelijkheden van tuning zijn beperkter dan die van ETL-tools. Hoewel metadata beschikbaar zijn, zijn deze beperkt (geen repository voor berekeningen) en uitsluitend technisch van karakter. Ook is het gebruik van set-

Viewnaam	Omschrijving	**Type**
Tr_Vul_Klant	Vult klanten dimensie met nieuwe klanten. Werkt stamgegevens bij van bestaande klanten.	UPDATE/INSERT
Tr_bepaal_omzet	Plaatst omzetgegevens van elke klant over de afgelopen periode. De view heeft twee kolommen; een klantnummer en een klantomzet.	UPDATE
Tr_segmenteer_klant	Werkt het segment van een klant bij op basis van de omzetgegevens van de klant.	UPDATE

Afbeelding 2. Drie-stappenplan.

operaties via SQL niet altijd de meest optimale manier om transport en transformatie van gegevens te ondersteunen.
Voordelen zijn er ook. Database views zijn gemeengoed en veel consultants zijn in staat deze te beheren en naar believen aan te passen. De business-logica in de views is wel degelijk opvraagbaar, (via het create statement van de view) en is als tekstformaat niet minder toegankelijk dan veel standaard ETL-omgevingen. Kennis over SQL en views is breed aanwezig op de markt. Views kunnen zowel opgeschaald als uitgeschaald (meerdere databases) worden. De complexiteit van het systeem als zodanig neemt daardoor af. Eventuele nadelen zijn voornamelijk gelieerd aan uitzonderingen in complexiteit en uitzonderingen in hoeveelheid. Het is altijd mogelijk voor deze uitzonderingen aparte oplossingen te implementeren.



Afbeelding 3: Doelmodel met zeven tabellen.

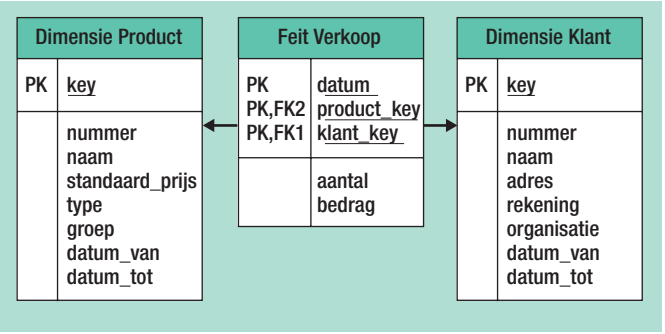
Een manier om de uitzonderlijk complexe transformaties te vermijden is om ze op te splitsen in meerdere kleinere transformaties met te 'behappen' complexiteit. Dit komt ook het beheer en de transparantie ten goede. De tabel in afbeelding 2 toont een driestappenplan om een klantenbestand bij te werken, omzetgegevens toe te voegen en klanten te segmenteren naar omzetklasse. Dit zou ook in een transformatie view kunnen, maar dat maakt het niet overzichtelijk en zeker niet snel qua verwerking. Functioneel bezien verandert de logica in de laatste view vaker dan de eerste. De opsplitsing in drie views is daarom verstandig.
De scriptgenerator en transformatie views leveren samen het gereedschap voor het efficiënt toepassen van gegevenslogistiek. Met deze twee componenten zijn nagenoeg alle ETL-processen in kleine tot middelgrote datawarehouses te implementeren.

Generiek laden van gegevens en relationele consistentie

Om gegevens in een relationeel model te wijzigen, te verwijderen of toe te voegen dient aan een aantal regels te worden voldaan.
De relationele consistentie is een van de meest belangrijke regels Aan de hand van een voorbeeld wordt uitgelegd hoe hiermee slim kan worden omgegaan. Het doelmodel bestaat daarbij uit zeven tabellen, zie afbeelding 3. Het gaat om omzetgegevens die worden bijgehouden op klant-, product- en dagniveau. Een klant behoort tot een klantgroep. Een product is van een producttype en behoort tot een productgroep. De klantgroep wordt bediend door een organisatieonderdeel, evenals de productgroep. Dit kunnen beide verschillende organisatieonderdelen zijn; bijvoorbeeld een helpdesk en een verkoopteam.
Dagelijks ontstaan nieuwe verkoopgegevens. Dit zou een eenvoudige toevoeging in de Verkopen-tabel betekenen, maar helaas worden, in verband met garanties en terugboekingen, verkopen met terugwerkende kracht geadministreerd (antidatering). Deze logica leidt dus tot zowel een UPDATE- als een INSERT-actie. Om een regel in de Verkopen-tabel toe te voegen, moeten de datum, het productnummer en het klantnummer ingevuld zijn. Daarnaast volgt nog een controle of het klantnummer ook daadwerkelijk bestaat in de Klant-tabel. Hetzelfde geldt voor product. Problemen gaan ontstaan wanneer een bronsysteem gegevens aanbiedt waarin onbekende producten geregistreerd zijn, of waarin regels voorkomen met omzetgegevens maar zonder een klantnummer.
Op voorhand is aan te geven hoe moet worden omgegaan met fouten in de brongegevens. Dit is in principe de taak van de opdrachtgever; welke kwaliteit van gegevens is gewenst en op welke wijze moeten kwaliteitsproblemen worden aangepakt? Het aantal mogelijke fouten en oplossingen met betrekking tot relationele consistentie is op een hand te tellen. Afbeelding 4 geeft een opsomming van mogelijke fouten en oplossingen, die met de 'Transformatie View Enhancer' te automatiseren zijn.
De aanpak van de hierboven geschetste consistentieproblemen wordt vanaf nu de consistentiestrategie genoemd. Het kan zijn dat de consistentiestrategie voor producten anders is dan die voor klanten. Klanten kunnen voor een markeringafdeling veel belangrijker zijn dan voor een productie-omgeving.

A	**Het aangeboden productnummer is leeg.**	
A1	Stop het proces met een foutmelding. Waarschuw de beheerder van het bronsysteem en van het datawarehouse.	Niets doen; een foutmelding zal vanzelf verschijnen bij verplichte relaties.
A2	Verwerk het record niet (sla het over) en markeer deze eventueel voor latere analyse. Ga verder met de resterende records.	Filter de brongegevens op lege nummers.
A3	Vervang het lege product_nr door het getal '-1'. Het product met productnummer '-1' is een zogenaamd dummy-product en vervangt een leeg productnummer.	Geef een alternatieve waarde (-1) indien het nummer leeg is.
B	**Het aangeboden productnummer is onbekend in de productentabel.**	
B1	Stop het proces met een foutmelding. Waarschuw de beheerder van het bronsysteem en van het datawarehouse.	Tijdens het uitvoeren van het script zal een foutmelding ontstaan.
B2	Verwerk het record niet (sla het over) en markeer deze eventueel voor latere analyse. Ga verder met de resterende records.	Voer een lookup uit in de productentabel. Sla het record over indien productnummer niet gevonden is.
B3	Vervang het onbekende product_nr door het getal '-2'. Het product met productnummer '-2' is een zogenaamd dummy-product en vervangt een onbekend productnummer.	Voer een lookup uit in de productentabel en, indien het record niet gevonden is, vervang deze door een alternatieve waarde (-2).
B4	Voeg het onbekende productnummer eerst toe aan de productentabel en verwerk het record opnieuw. Markeer het zojuist toegevoegde productnummer voor latere analyse. Geef dummy-waardes onbekend (-1) voor de kenmerken groep_nr en type_nr.	Genereer een nieuw INSERT-script voor de productentabel. Gebruik hiervoor de originele bron (met daarin het productnummer). Voer dit script uit alvorens de omzetgegevens bij te werken.
C	**Het kenmerk productnummer wordt niet meegeleverd in de bron.**	
C1	Stop het proces met een foutmelding. Waarschuw de beheerder van het datawarehouse.	Tijdens het uitvoeren van het script zal een foutmelding ontstaan.
C2	Voeg een dummy-productnummer toe met de waarde '-1'.	

Afbeelding 4: Fouten en oplossingen.



Afbeelding 5: Sterschema.

Dimensionele logica

Een ander terugkerend fenomeen in datawarehouses is het omgaan met dimensies en het onderhouden van historie. Kimbelliaanse technieken (sterschema's, snowflakes) lenen zich uitstekend voor verdere automatisering (dimensionele intelligentie). Wanneer het eerdere voorbeeld wordt omgezet naar een sterschema, is het voorbeeld zoals getoond in afbeelding 5 een uitkomst.

De brongegevens uit voorgaande teksten vermelden helaas geen *key*-velden, alleen product- en klantnummers. In de brongegevens dienen deze nummers daarom omgezet te worden naar de *dimensie keys*. Een lookup op basis van product- of klantnummer is hiervoor voldoende. Voor Slowly Changing Dimensions wordt dan ook nog een geldigheidsperiode meegenomen in de lookup-functie. Om het feit Omzet te vullen uit de oorspronkelijke tabel Staging_Omzet, moeten de klant- en productnummers uit de verkooptabel vervangen worden door de corresponderende surrogaat-sleutels. Dit heet in jargon 'dimension lookup' of 'surrogate key lookup'.

Naast het opzoeken van de juiste dimensiesleutels speelt ook het bijhouden van historie een rol. Stel dat de dimensie Klant een historie kent voor het kenmerk *organisatie*. Elke keer dat een klant door een ander organisatie-onderdeel wordt bediend, wordt de bestaande 'afgesloten' door een geldigheids-einddatum in te voeren en wordt een nieuwe klant toegevoegd. De nieuwe klant krijgt identieke kenmerken, behalve het kenmerk 'organisatie-onderdeel', de geldigheidsdatum en natuurlijk de betekenisloze sleutel (key). De scriptgenerator gebruikt voor slowly changing dimensions een ander sjabloon, zie afbeelding 6.

De Transformatie View Enhancer is een uitbreiding op de standaard transformatie view en is verantwoordelijk voor de consistentiestrategie en voor het omgaan met dimensionele modellen. De

	Standaard tabel	Dimensie tabel
1	-	Afsluiten gewijzigde records met historie (UPDATE)
2	Bijwerken bestaande gegevens (UPDATE)	Bijwerken bestaande gegevens zonder historie (UPDATE)
3	Toevoegen nieuwe gegevens (INSERT)	Toevoegen nieuwe gegevens (INSERT)

Afbeelding 6: Sjabloon voor slowly changing dimensions.

Enhancer genereert op basis de bron, het doel, het relationele model en de gedefinieerde strategie een nieuwe transformatie view, de Enhanced Transformatie View. De scriptgenerator ziet geen verschil tussen beide. De Transformatie View Enhancer voert een aantal functies uit, zie afbeelding 7. De Enhancer analyseert het bronobject (transformatie view), het doelmodel en voert vervolgens een aantal acties uit met betrekking tot relationele consistentie en dimensiesleutels.

Consistentiestrategieën

De Enhancer neemt een bronobject als uitgangspunt en genereert een nieuw SELECT statement, waarin de voorgedefinieerde consistentiestrategieën gestalte krijgen.

Het volgende fragment laat een implementatie zien voor strategie A1/B3, (vervang een lege waarde door een dummy -1 en genereer een foutmelding bij een onbekende waarde)

```
1 SELECT…, ISNULL(klant_nr, -1) AS klant_nr, … FROM …
```

Een standaard JOIN clause slaat lege en/of onbekende waardes over. In dit geval werkt de JOIN als filter.

```
2 SELECT … FROM Omzet JOIN Klant ON
                        Omzet.klant_nr = klant.nummer
```

Iets moeilijker wordt het als we de lege kolommen niet willen verwerken en de onbekende waardes vervangen door een dummy-waarde. De dummy-waarde in het voorbeeld is '-2'.

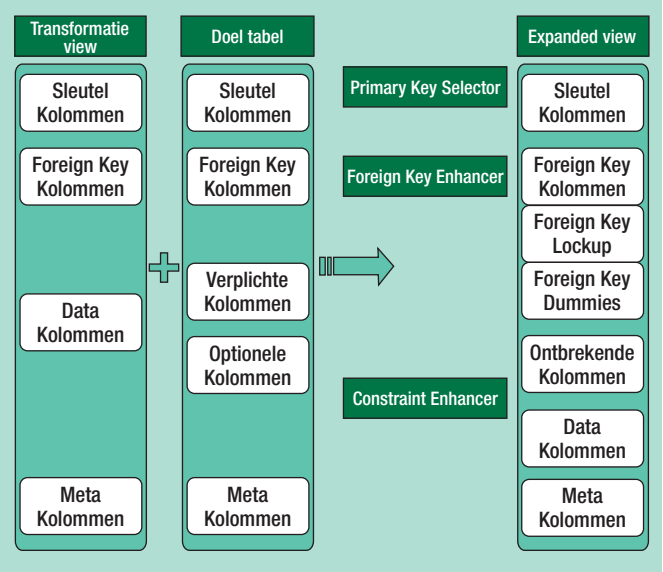
```
3 SELECT
4         ISNULL(klant.nummer, -2) AS klant_nr
5 FROM
6         Omzet
7         LEFT OUTER JOIN Klant ON Omzet.klant_nr
                                    = klant.nummer
8 WHERE
9         omzet.klant_nr IS NOT NULL
```

In eerste instantie worden de lege klantnummers uitgefilterd (9). Alleen de onbekende nummers blijven over. Hiervoor is een LEFT OUTER JOIN nodig, want de onbekende nummers mogen niet gefilterd (7) worden. Een niet gevonden klant impliceert een onbekende klant. Vervang dit door de dummy '-2' (regel 4).

In het laatste voorbeeld vervangt een -1 de lege waarde en -2 de onbekende waarde.

```
10 SELECT
11         ISNULL(klant.nummer, -2) AS klant_nr
12 FROM
13         Omzet
14         LEFT OUTER JOIN Klant ON ISNULL
                  (Omzet.klant_nr, -1) = klant.nummer
```

Elke consistentiestrategie is implementeerbaar door toevoegingen en wijzigingen in de SELECT, de FROM en de WHERE clause.



Afbeelding 7: Functies Transformatie View Enhancer.

Dit concept is zelfs nog verder uit te breiden. Ook kleine structuurwijzigingen in de brongegevens of doelmodellen zijn eenvoudig te ondervangen. In het volgende voorbeeld is in de doeltabel een nieuwe relatie *afdeling_nr* toegevoegd. De transformatielogica is hier nog niet aangepast. De Enhancer ondervangt dit door een dummy-waarde '-1 (leeg)' toe te voegen aan het bronobject Omzet.

```
15 SELECT
16         -1 AS afdeling_nr
17 FROM
18         Omzet
```

Het ondervangen van structuurwijzigingen is wellicht niet gewenst, maar is in parallelle ontwikkeltrajecten tijdbesparend. Het is eenvoudig om het ondervangen van structuurwijzigingen aan te zetten of uit te zetten.

Dimensionele Intelligentie

Een andere functie van de Enhancer ligt in de dimensionele intelligentie. Door zelf de dimensionele afhandeling uit te voeren, worden de ontwikkelaar en de ontwerper ontlast van zuiver technische zaken en treden er minder fouten op. Hier volgt weer een voorbeeld; voor het bijwerken van de Omzet-feit is de volgende transformatie view ontwikkeld.

```
1 SELECT datum, product_nr, klant_nr, aantal, excl
                                              as bedrag
2 FROM   Omzet
3 WHERE  datum > now() –30
```

Deze structuur past niet in de feitentabel. De feitentabel heeft wel een datumveld, maar geen product_nr en geen klant_nr. In plaats daarvan beschikt de tabel over product_key en klant_key.
De Enhancer zorgt voor automatische vertaling van de nummers naar de sleutels. Voorwaarde hiervoor is wel dat minimale informatie over de dimensies beschikbaar is.

```
1 SELECT datum
         , prd.key as product_key
         , klt.key as klant_key
2        , aantal, excl as bedrag
3 FROM   TransformatieView
4 WHERE  INNER JOIN dimensie_klant klt ON
5        klt.nummer = klant_nr
6        INNER JOIN dimensie_product prd ON
7        prd.nummer = product_nr
8        AND datum between prd.datum_van AND
                                      prd. datum_tot
```

De Enhancer gebruikt de in de database aanwezige informatie over de objecten (tabellen, views, sleutels, indexen) en gebruikt vervolgens extern aanwezige informatie dat meer over deze objecten zegt.
De Transformatie View Enhancer werkt op dezelfde manier als de scriptgenerator. Op basis van informatie over modellen, objecten, dimensies, sleutels en een sjabloon, genereert de view enhancer een nieuwe view; de Enhanced Transformation View. Vervolgens wordt deze nieuwe view aan de scriptgenerator aangeboden.
De Transformatie View Enhancer is in staat veel meer informatie op intelligente wijze in te zetten. Zo kunnen bijvoorbeeld incrementele wijzigingen in tabellen eenvoudig worden opgevangen. Hier moet echter de balans gezocht worden in voorspelbaarheid, complexiteit en beheersbaarheid. Teveel intelligentie leidt als snel tot een 'black box', te weinig leidt weer tot duur beheer. Door de enhancer te beperken in goed af te bakenen probleemdomeinen wordt maximaal voordeel gehaald uit minimale investeringen.

Conclusies

De generieke werking van de scriptgenerator, het gebruik van sjablonen en de ingebouwde intelligentie van de Transformatie View Enhancer vormen samen zeer efficiënt gereedschap om gegevens in databases te transformeren en te transporteren. De ontwikkelaar hoeft slechts op globaal niveau aan te geven hoe om te gaan met relationele gegevenskwaliteit en hoeft geen dimensionele intelligentie in te bouwen. De enige objecten die een ontwikkelaar implementeert zijn tabellen, constraints en transformatie views. De vermindering in het aantal te ontwikkelen objecten, de geautomatiseerde logica en de eenvoudige wijze waarmee relationele consistentie in ETL behandeld wordt, leidt niet alleen tot snelle oplossingen, maar vermindert voornamelijk het aantal fouten en daarmee de ROI. Metadata zijn nu niet de uitkomst van een ontwikkelproces, maar een sturende factor geworden in de problemen rondom de gegevenslogistiek.
Dit artikel heeft niet ten doel het bestaansrecht van professionele ETL-tools te ondergraven, maar is meer een pleidooi voor structureel en creatief gebruik van de informatie die al gratis voorhanden is. Een aantal voorbeeld scriptgeneratoren wordt gratis door Centennium beschikbaar gesteld. Deze zijn op het web-adres www.centennium.nl/DBM te vinden en te downloaden.

Marcel Witteveen is Business Intelligence consultant bij Centennium BI expertisehuis.