Gebruikers toegang bieden én het ETL-ontwerpproces vereenvoudigen

Het dilemma tijdens het ontwerpen

Michael Schmitz

In datawarehousing wordt vaak de vraag gesteld of het de gebruiker gemakkelijker gemaakt moet worden om toegang tot de informatie te verkrijgen, of dat gemakkelijke ontwikkeling en onderhoud van ETL-functies de voorkeur verdient.

Bij dit dilemma spelen diverse mogelijke, en onderling tegenstrijdige, gevolgen een rol. IT-organisaties zouden deze grondig tegen elkaar moeten afwegen alvorens een beslissing te nemen. Om de business user in staat te stellen een datawarehouse te raadplegen, is de aanwezigheid van schema's vereist die eenvoudig te gebruiken zijn en goede resultaten geven. Het ontwerp moet aan de volgende eisen voldoen:

- leesbaar voor de gebruiker;
- korte, voorspelbare respons-tijd;
- ondersteuning van zowel voorziene als onvoorziene toegangspaden, vaak over grote hoeveelheden data;
- inherente correctheid;
- contextgebonden history beschikbaar;
- flexibel en uitbreidbaar;
- goed te onderhouden.

De ETL-processen die deze schema's laden en onderhouden, moeten:

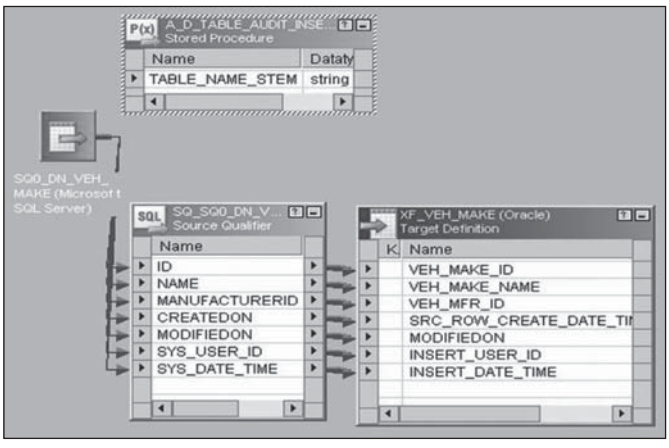
- efficiënt werken en goed schaalbaar zijn (kunnen omgaan met mogelijk grote hoeveelheden data);
- snelle ontwikkeling ondersteunen;
- datavalidatie bieden;
- goed te onderhouden zijn;
- herstartbaar zijn en foutafhandeling en auditing bieden.

Hoe meer nadruk er wordt gelegd op de schemadoelstellingen, hoe complexer het ETL-proces wordt. De leesbaarheid voor de gebruiker en de snelle werking betekenen dat het schema zo eenvoudig mogelijk moet zijn. Het verband tussen eenvoud en bruikbaarheid is vanzelfsprekend, maar waarom is eenvoud ook nodig voor snelheid? Eenvoud is van belang voor de snelheid, omdat het degene die de database optimaliseert in staat stelt de structuur te doorgronden en daardoor de beste optimalisatie-algoritmes te kiezen. Het bewerkstelligen van een dergelijke eenvoud vergt wel het een en ander van de ontwikkelaar, en de resulterende structuren compliceren ETL-ontwikkeling veelal aanzienlijk. Alleen geconcentreerde inspanning leidt tot ware eenvoud.

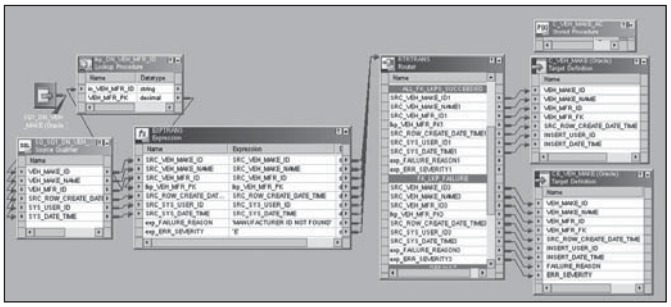
De twee lagen-oplossing

Er bestaan tegenwoordig veel oplossingen waarin een gegevensarchitectuur bestaande uit twee lagen wordt toegepast, waarbij de basislaag zeer algemeen wordt gehouden door een hoge mate van abstractie. Dit vereenvoudigt de ontwikkeling van de eerste laag en bevordert de ontwikkeling van generieke gegevensarchitecturen en ETL-processen. Hoe meer abstractie er wordt ingebouwd in de datamodellering en de ETL-processen, hoe gemakkelijker het wordt ze te onderhouden. Daar is op zich niets tegen in te brengen; het probleem is echter dat men hiervoor een prijs betaalt op het vlak van bruikbaarheid en prestatie. Om dit te voorkomen wordt daarom een presentatielaag gebouwd ten behoeve van gebruik en performance.

Het uiteindelijke resultaat is dat degenen die verantwoordelijk zijn voor ontwikkeling van de basislaag, een eenvoudige taak hebben om de benodigde processen te ontwikkelen en onderhouden, terwijl het team dat de presentatielaag ontwikkelt de zwaarste lasten draagt. Ik ben van mening dat deze aanpak de plank misslaat. In feite worden de problemen verschoven van de ene naar de andere groep en ontstaat er al met al zelfs meer werk. De bouw van de basislaag gaat inderdaad sneller, maar voordat de gebruiker aan de slag kan moet wel eerst de tweede laag worden ontwikkeld. Ik heb voor verschillende klanten gewerkt waar de uiteindelijke oplevering aanzienlijke vertraging opliep door deze aanpak met het bouwen van twee lagen.
Een betere benadering is om een gegevensarchitectuur bestaande



Afbeelding 1: Extractie.



Afbeelding 2: Ontdekken van veranderingen.

uit een enkele laag te gebruiken, die efficiënt werkt en die bovendien effectief gebruik door zakelijke gebruikers mogelijk maakt. Maar voor niets gaat de zon op en in dit geval bestaan de kosten uit moeilijker te ontwikkelen ETL-processen.

Een elegante oplossing

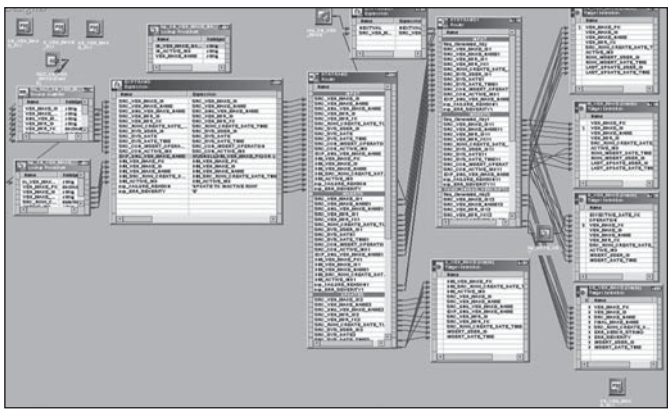
Mijn aanbeveling is een gegevensarchitectuur met een enkele laag te gebruiken: de 'Dimensional Normal Form'. Deze gegevensarchitectuur biedt de gebruiker eenvoud en goede prestaties. Het vergt wel een hogere mate van ETL-complexiteit. Die complexiteit kan men te lijf gaan met behulp van dimensie onderhouds-templates, die minder code behoeven en een gestandaardiseerde aanpak voor het onderhoud van dimensietabellen bieden. Met deze templates kunnen op basis van metadata-definities zowel de ondersteunende database-scripts als de tool-specifieke ETL-processen automatisch worden gegenereerd, hetgeen goed is voor circa 90 procent van de totale ontwikkeling van dimensie onderhoudsprocessen. Een kort overzicht volgt van gebruikte methoden, metadata en templates.
We delen dimensie onderhoud op in vier mogelijke stappen, waarbij we gebruik maken van tussenliggende tabellen ten behoeve van snelle ontwikkeling en herstart- en testpunten. Na het testen en debuggen kunnen sommige tussenliggende tabellen worden verwijderd met het oog op een hogere verwerkingssnelheid, maar de beste herstartpunten moeten worden behouden. Deze templates bieden efficiënte ondersteuning voor zowel de initiële als voor latere laadprocessen, en voor zowel volledige als incrementele extractieprocessen.
Volledige extractie vindt plaats wanneer incrementele extractie onmogelijk is, of wanneer er niet zonder meer van uit kan worden gegaan dat de incrementele extracties alle veranderingen weergeven. Bij volledige extractie worden stappen 1 tot en met 4 doorlopen, bij incrementele extractie alleen stappen 2 tot en met 4.

Stap 1 (alleen bij volledige extractie): Extraheer de brondata, zet deze over naar het ETL-platform en zet ze daar klaar voor gebruik. Aan de geëxtraheerde data wordt zo weinig mogelijk gerekend, totdat er veranderingen worden ontdekt. Het is een verspilling van rekenkracht om onveranderde data te checken en te valideren, die slechts zullen worden weggegooid.
Stap 2: Detecteer welke data nieuw of veranderd zijn (dit is al gedaan bij een incrementele extractie) en check de integriteit ervan (zowel op entiteitsniveau als referentieel), waarbij deze in een tussenliggende tabel worden geplaatst. Hierdoor kan het initiële laadproces bijzonder efficiënt verlopen, aangezien alle data nieuw zijn en als bulk in de tussenliggende tabel worden geplaatst. Er vindt geen logging plaats en alle constraints worden parallel toegepast.
Stap 3: Voeg, opnieuw door middel van in bulk laden, nieuwe rijen toe aan de dimensie en plaats de updates. Het scheiden van het update-proces van het invoegproces heeft twee voordelen. Ten eerste kunnen data al worden geladen en voor prototyping gebruikt voordat de complexe update-afhandeling is gecodeerd. Ten tweede wordt zo het bulk laden van de dimensie mogelijk gemaakt. Weliswaar is bulk laden voor kleine dimensietabellen niet nodig, maar deze aanpak is gelijk voor zowel kleine als grote dimensies en is daarmee dus gestandaardiseerd.
Stap 4: Voer het update-proces van de dimensietabel uit.
Er worden zes verschillende methoden ondersteund om de history van dimensie-attributen te volgen.

Afbeeldingen 1 tot en met 4 tonen de vier processen zoals ze zijn geïmplementeerd in Power Mart van Informatica. Deze templates zijn gerealiseerd met de volgende ETL-producten: Informatica, Data Warehouse Builder van Oracle en DTS van Microsoft.

Metadata

Om de algemene database data definition language (DDL) en data manipulation language (DML) en de ETL-processen te kunnen genereren, zijn metadata nodig. Voor iedere kolom in de doeltabel zijn uit de brontabel nodig: schema-eigenaar, tabelnaam en kolomnaam. Daarnaast zijn uiteraard ook uit de doeltabel de schema-eigenaar en de stam van de tabelnaam nodig.
De stam wordt gebruikt met gestandaardiseerde prefixen die de tussenliggende, history-, error- en dimensietabellen aanwijzen.
De history-methode voor doeltabel kolommen moet aangewezen worden samen met aanwijzen van logische primaire sleutels, unieke en verwijzende sleutels. De kolom attribuut-history wordt met het oog op audits altijd ondersteund door een Dimension History Tabel, dit staat van oudsher bekend als Methode 5.
Er bestaan nog vijf andere methoden die de attribuut-history

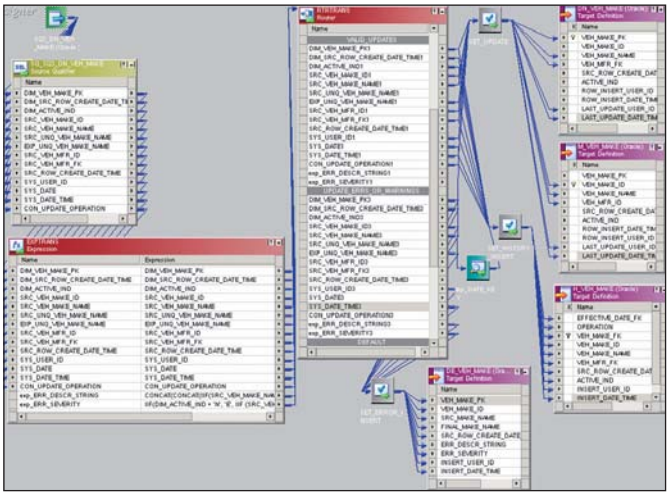


Afbeelding 3: Afhandelen van inserts (stap 3).

ondersteunen voor analytische doeleinden. Methode 1 vervangt eenvoudig de waarde, zonder een analytische history te produceren. Methode 2 ondersteunt de history door bij iedere verandering van de kolom een nieuwe versie van de dimensie te genereren. Methode 3 maakt gebruik van extra kolommen in de tabel om een beperkte hoeveelheid history bij te houden. Methode 4 haalt een of meer kolommen uit de dimensietabel, een sleutel genererend voor iedere unieke combinatie, en deze sleutel wordt direct in gerelateerde feitentabellen geplaatst; waarbij de rijen van de feitentabel worden verbonden met de kolomwaarden op het moment van activiteit in de feitentabel. Methode 6 zorgt ervoor dat unieke identifiers transparant blijven met behulp van een bijbehorende transparantie-tabel.

Conclusie

Het gebruik van de templates kan een aanzienlijke besparing van ontwikkelingstijd en -arbeid opleveren, waarbij bovendien een schaalbare, gestandaardiseerde aanpak voor de ontwikkeling van een onderhoudsmethode voor dimensietabellen ontstaat. Hierdoor kan men met inzet van een enkellaags-gegevensarchitectuur ('Dimensional Normal Form') ontwerpen maken gericht op bruikbaarheid en snelheid en bovendien, zonder een tweede laag in de gegevensarchitectuur in te bouwen, de complexere ETL-processen toch snel, efficiënt en volledig blijven ontwikkelen.
Kortom: deze benadering stelt ons in staat om de business toegang te geven tot informatie en daarbij de mate van complexiteit te sturen bij het bouwen van de ondersteunende ETL-processen.



Afbeelding 4: Afhandelen van updates (stap 4).

Michael Schmitz is onafhankelijk consultant, docent en eigenaar van Business Knowledge Professionals Inc, te Oregon USA (www.bk-pro.com).

De oorspronkelijke onvertaalde tekst, die tevens voorbeeldcode bevat, kunt u als PDF-document vinden op www.dbm.nl