Drie componenttypen in de gegevenslaag

ETL-Generator (2)

Ronald Bijlhouwer, Richard Puijk en Vincent Wylenzek

De methode van ontwikkeling en bouw van het ETL-proces is aan het veranderen. Er is een duidelijke trend zichtbaar van automatisering van het ETL-bouwproces. Zoals wij in het eerste deel van dit drieluik al schreven, zijn er inmiddels al zo'n 15 partijen die een ETL-generator hebben ontwikkeld of hiermee bezig zijn. De early-adopters zijn erg enthousiast en wij verwachten dat de partijen met een goed functionerende ETL-generator voorlopig een aanzienlijk concurrentievoordeel kunnen behalen.

Wilt u, als zakelijke dienstverlener, de boot niet missen? Verdiep u dan in het genereren van ETL en lees dit artikel. De gegevenslaag bestaat uit drie componenttypen, namelijk de datastore, het datawarehouse (DWH) en één of meerdere (fysieke en/of logische) datamarts, zie afbeelding 1 in deel 1 (DB/M 5). In dit artikel wordt per component besproken in hoeverre deze gegenereerd kan worden.

Datastore

Een datastore heeft als belangrijkste functie het bieden van een online, direct bevraagbaar archief over alle aangeboden data. Wanneer niet alle data uit de aanleveringen in het datawarehouse worden opgenomen, kan dit archief inzicht verwerven in welke data nog meer beschikbaar zijn. Daarnaast kan de datastore voor auditdoeleinden gebruikt worden.
Er zijn diverse ontwerpkeuzes en vrijheidsgraden die bepalen hoe de uiteindelijke datastore er uit komt te zien. Bij de door ons gebruikte datastore gelden de volgende uitgangspunten:

1. De modellering van een datastore komt (zoveel mogelijk) overeen met de bron. Dit impliceert dat als een bron wijzigt, de datastore mee wijzigt;
2. Waar mogelijk worden maatregelen met betrekking tot de efficiëntie toegepast ten aanzien van de hoeveelheid dataopslag;
3. De kolomdefinities van aangeleverde interfaces worden één op één overgenomen in een datastoretabel (bron-centrisch);
4. Indien een interface wijzigt, wordt een nieuwe tabel aangemaakt en de oude afgesloten met de afsluitdatum als postfix in de tabelnaam, dit geldt tevens voor de metatabel.

Dit is deel 2 van een drieluik over het genereren van ETL. In deel 1 (DB/M 5) werd de realisatie van de ETL-generator toegelicht. Het ETL-proces van DWH naar datamart is onderdeel van deel 3.

Uitgangspunt twee in het bijzonder, ligt ten grondslag aan de volgende ontwerpkeuzes: de datastore bestaat uit twee typen tabellen, namelijk data- en metatabellen; elke datatabel heeft een corresponderende metatabel waarin bijgehouden wordt in welke runs, welk record wordt aangeleverd. Dit levert een significante besparing ten aanzien van de opslagruimte op ten opzichte van een datastore waarbij alle data, tijdens elk laadproces volledig worden bijgeladen.

De datatabel bevat alle data in hun ruwe vorm. De metatabel bevat niets anders dan het runnummer en een referentie naar het betreffende ID in de datatabel. In combinatie met de datatabel is het eenvoudig mogelijk om op te vragen in welke run een bepaald record aanwezig was. Zie afbeelding 1 voor een voorbeeld van een datatabel en bijbehorende metatabel.
Met de ETL-generator kan de gehele datastore gegenereerd worden. In de metadata database wordt bijgehouden welke versie van een bepaalde interface actief is. Indien de interfacedefinitie wordt gewijzigd met de GUI, impliceert dit een ophoging van het versienummer van de interface. Een dergelijke versieophoging initieert een procedure die ervoor zorgt dat de actieve data- en metatabel worden afgesloten met een einddatum in de objectnaam. Vervolgens wordt de nieuwe tabel, op basis van de nieuwe interfacedefinitie, aangemaakt.

Modelleringkeus datawarehouse

Zoals uit het logische architectuuroverzicht blijkt, is men vrij in de modelleringkeus van het datawarehouse. In de basis betreft het hier een keuze uit het relationele (3NF), het dimensionale of het Data Vault-model met elk hun eigen specifieke kenmerken. Wij hebben in onze eerste iteratie van de ETL-generator gekozen voor een uitwerking op basis van Data Vault. De keuze voor Data Vault is gebaseerd op een actuele klantvraag. Een ander voor-

deel van Data Vault is dat op basis van enkele modelleerafspraken, eenvoudig de bron- (stage) en doelvelden (DWH) gekoppeld kunnen worden. Er hoeft slechts per attribuut vastgelegd te worden of dit een detail-, sleutel- of linkattribuut betreft. Deze bepaling kan grotendeels geautomatiseerd verlopen, door bijvoorbeeld de datadictionary van een bron uit te lezen. In volgende iteraties worden de mogelijkheden voor het genereren van ETL voor het relationele en het dimensionale model bekeken. Als we de ETL vertalen naar een wat abstracter niveau, dan blijkt al snel dat er een generiek patroon te ontdekken is, welke onafhankelijk is van het gekozen modelleringtype. Hieronder volgt een uitleg van dit patroon.

Generiek patroon

Als het uitgangspunt is dat er al een datamodel aanwezig is voor het datawarehouse, dan kunnen we een generiek patroon ontwerpen voor de ETL. Vanaf de staginglaag naar het datawarehouse gelden dan de stappen zoals in afbeelding 2 te zien is.

Stap 1 Initialiseren. Initialiseren is volledig generiek, slechts de packagenaam en doeltabel zijn variabel. Voor logging wordt gebruik gemaakt van een combinatie tussen de SSIS-logging-component en een stored procedure. In deze stap wordt het logproces geïnitieerd. Het logproces wordt in deel 3 toegelicht.

Stap 2 Selecteren. Bij deze stap is de data source component generiek, de inhoudelijke query is specifiek en kan in de GUI worden ingevoerd. Lookups, joins en sorts worden in dit query statement opgenomen.

Stap 3 Bewerken. Bewerken is specifiek, maar met de juiste metadata wel (deels) te genereren. In de GUI kan men aangeven of een aanlevering vanuit de staginglaag moet worden ontdubbeld. Transformaties, filters, aggregaties en vaste extra attributen kunnen ook in de GUI worden opgevoerd. Hier komen we in de paragraaf over de techniek op terug.

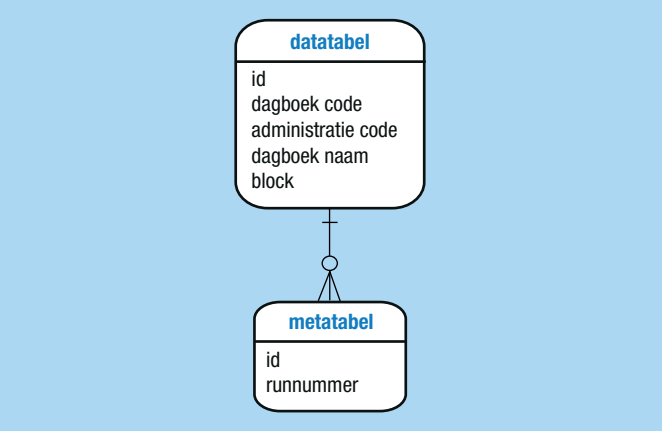
Stap 4 Splitsen. Splitsen is generiek, dit mechanisme bestaat altijd uit drie stromen, namelijk toevoegingen, aanpassingen en verwijderingen. In de GUI kan aangegeven worden of er historie bijgehouden moet worden over één of meerdere attributen. Wijzigingen op bestaande rijen worden gedetecteerd door afwijkende hashkeys.

Stap 5 Samenvoegen. Samenvoegen is generiek, alle toe te voegen records en gewijzigde records voor de datawarehousetabel worden met een union samengevoegd.

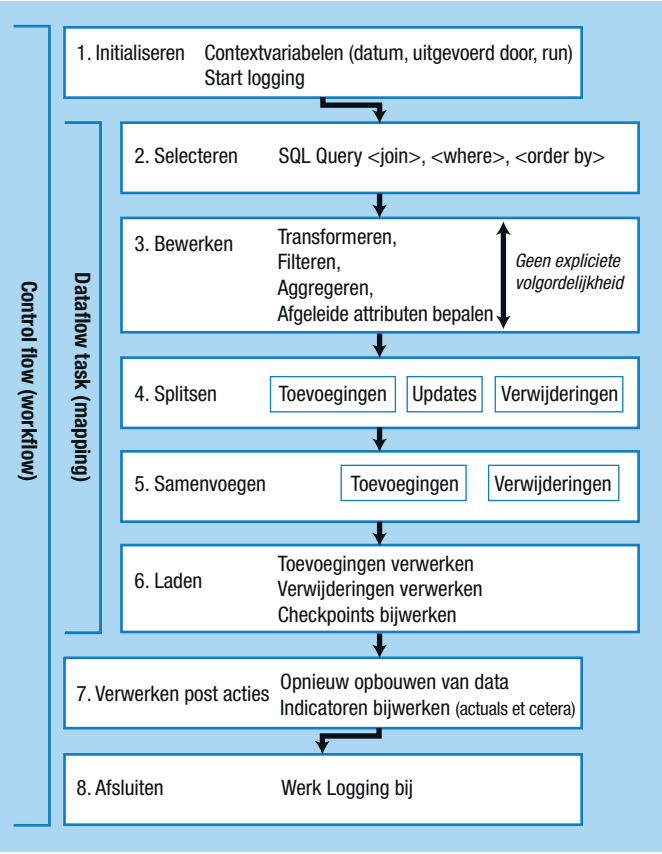
Stap 6 Laden. Laden is generiek, alle output uit stap 5 wordt geladen in de datawarehousetabel.

Stap 7 Verwerken post acties. Deze component is deels generiek, de bij te bewerken doeltabel en bijbehorende attributen worden geconfigureerd door middel van metadata. In deze stap worden procesafhankelijke gegevens bijgewerkt, bijvoorbeeld batchgewijs verwerken van wijzigingen en opnieuw opbouwen van de tabel met actuele gegevens.

Stap 8 Afsluiten. Afsluiten is generiek, slechts de packagename en de doeltabel zijn variabel. Deze stap werkt de logresultaten bij en sluit de logging af.



Afbeelding 1: Tabelstructuur datastore.



Afbeelding 2: Generiek patroon ETL.

De techniek

Uit de voorgaande paragrafen is duidelijk geworden dat de structuur van het ETL-proces naar het datawarehouse redelijk generiek is. De specifieke invulling van dit proces kan met behulp van de ETL-generator en metadata worden gerealiseerd. Met deze kennis kunnen we een volgende stap maken. Zoals we reeds in ons eerste artikel beschreven, maken we gebruik van templates. We hebben hier heel bewust voor gekozen, omdat in de praktijk blijkt dat het gewenste ETL-proces kan variëren per klant. Daarnaast kunnen de vereisten aan het ETL-proces in de loop der tijd wijzigen. Door gebruik te maken van templates

```
private void UpdateConnectionManagers()
{
        foreach (ConnectionManager connectionManager in package.Connections)
        {
                if (connectionManager.CreationName == "FLATFILE")
                {
                        connectionManager.ConnectionString =
TemplateParser.ParseTemplateString(connectionManager.ConnectionString);

                        if (connectionManager.Name.ToUpper().Contains("{ENTITY}"))
                        {
                                RuntimeWrapper.IDTSConnectionManagerFlatFile100 connectionFlatFile =
(RuntimeWrapper.IDTSConnectionManagerFlatFile100)connectionManager.InnerObject;

                                // verwijder de bestaande kolommen
                                foreach (RuntimeWrapper.IDTSConnectionManagerFlatFileColumn100 flatFileColumn in
connectionFlatFile.Columns)
                                {
                                        RuntimeWrapper.IDTSName100 columnName =
(RuntimeWrapper.IDTSName100)flatFileColumn;
                                        connectionFlatFile.Columns.Remove(flatFileColumn);
                                }

                                // aanmaken nieuwe kolommen op basis van de metadata
                                foreach (PackageColumn packageColumn in packageColumns)
                                {
                                        RuntimeWrapper.IDTSConnectionManagerFlatFileColumn100 flatFileColumn =
(RuntimeWrapper.IDTSConnectionManagerFlatFileColumn100)connectionFlatFile.Columns.Add();

                                        flatFileColumn.ColumnType = "Delimited";
                                        flatFileColumn.ColumnWidth = 0;
                                        flatFileColumn.MaximumWidth = (int)packageColumn.MaxLength;
                                        flatFileColumn.DataType = packageColumn.DataType;
                                        flatFileColumn.DataPrecision = (int)packageColumn.Precision;
                                        flatFileColumn.DataScale = (int)packageColumn.Scale;

                                        RuntimeWrapper.IDTSName100 columnName =
(RuntimeWrapper.IDTSName100)flatFileColumn;
                                        columnName.Name = packageColumn.ColumnName;
                                        if (packageColumn.Order == packageColumns.Count)
                                        {
                                                flatFileColumn.ColumnDelimiter = Environment.NewLine;
                                        }
                                        else
                                        {
                                                flatFileColumn.ColumnDelimiter =
packageDefinition.ColumnDelimiter;
                                        }
                                }
                        }
                }
        }
}
```

Code aanpassing connection manager van het type flatfile.

```
public class DataFlowTask
{
        const string FF_SRC_GUID = "{5ACD952A-F16A-41D8-A681-713640837664}";
        const string FF_DST_GUID = "{D658C424-8CF0-441C-B3C4-955E183B7FBA}";
        const string OLE_SRC_GUID = "{BCEFE59B-6819-47F7-A125-63753B33ABB7}";
        const string OLE_DST_GUID = "{5A0B62E8-D91D-49F5-94A5-7BE58DE508F0}";
        const string DCNV_GUID = "{BD06A22E-BC69-4AF7-A69B-C44C2EF684BB}";
        const string DER_GUID = "{2932025B-AB99-40F6-B5B8-783A73F80E24}";

        private void UpdateDataFlowComponents( IDTSComponentMetaData100 source,
                                               IDTSComponentMetaData100 destination,
                                               MainPipe dataFlowTask)
        {
                GenericComponentTypeProvider comp = null;

                switch (component.ComponentClassID)
                {
                        case FF_SRC_GUID:
                                comp = new FlatFileSourceComponent(package, packageDefinition, packageColumns);
                                comp.UpdateComponent(source, destination, dataFlowTask);
                                break;
                        case FF_DST_GUID:
                                comp = new FlatFileDestinationComponent(package, packageDefinition,
                                packageColumns);
                                comp.UpdateComponent(source, destination, dataFlowTask);
                                break;
                        case OLE_SRC_GUID:
                                comp = new OLESourceComponent(package, packageDefinition);
                                comp.UpdateComponent(source, destination, dataFlowTask);
                                break;
                        case OLE_DST_GUID:
                                ......
                }
        }
}
```

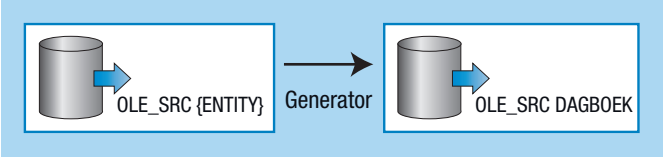
Codevoorbeeld actie per component.

heeft de betreffende consultant zelf de mogelijkheid om het ETL-proces aan te passen. Het is daarbij van belang dat de generator flexibel moet kunnen omgaan met allerlei templates. Om dit te bereiken wordt in de code niet meer aan specifieke componentnamen, maar aan componenttypes en placeholders gerefereerd. Hierdoor is het mogelijk om meerdere keren eenzelfde componenttype aan te roepen en kunnen nieuwe templates worden opgevoerd, zonder code-impact. Hiermee is tevens de beperking opgelost van de harde koppeling tussen code en template, die in het eerste artikel is beschreven.

Binnen de template kunnen placeholders op specifieke plaatsen aangebracht worden. De generator vervangt deze placeholders door gegevens uit de metadata. Deze metadatagegevens worden door middel van de GUI opgevoerd. Op deze manier wordt bijvoorbeeld de naam van de entiteit ingevuld, maar ook de specifieke eigenschappen van die entiteit, zoals bijvoorbeeld het query statement. Indien voor het maken van een extract altijd een volledige selectie van een brontabel wordt uitgevoerd, dan kan men de naam van de brontabel vervangen door een placeholder. Deze placeholder wordt ingevuld door de generator op basis van de ingevoerde metadata van bijvoorbeeld de brontabelnaam, zie afbeelding 3.

Van generiek patroon naar SSIS-componenten.

Zoals in het voorgaande deel besproken is, bestaat het ETL-proces naar het datawarehouse uit acht grotendeels generieke stappen. Voor deze stappen zijn SSIS-componenten benodigd. Voor elk benodigd SSIS-componenttype hebben we bepaald welke eigenschappen configureerbaar moeten zijn. Zie afbeelding 4 voor de stappen en benodigde componenten. Het wijzigen van een SSIS-component wordt door de generator uitgevoerd op basis van de ingevoerde metadata. Elke component heeft eigen specifieke eigenschappen, die de generator dient aan te passen. Afhankelijk van de opgevoerde template worden de diverse componenten geïnstantieerd. Een aantal acties moet echter altijd worden uitgevoerd: het aanmaken van een nieuwe package op basis van de geselecteerde template; het wijzigen van de connection managers; het bijwerken van de packagestructuur en het synchroniseren van de package pipeline. Zie afbeelding 5. Op het internet zijn inmiddels steeds meer voorbeelden van het genereren van SSIS-packages en daarbij tevens de eigenschappen van bepaalde SSIS-componenten te vinden. De volgende .NET- assembly's bevatten de specifieke SSIS classes en interfaces die men hiervoor kan gebruiken: Microsoft.SQLServer.DTSPipelineWrap, Microsoft.SQLServer.DTSRuntimeWrap,



Afbeelding 3: Placeholders in SSIS.

Stap	SSIS-Component(en)
Algemeen	Connection Manager Control Flow Task Data Flow Task
Initialiseren	Execute SQL Task
Selecteren	OLE DB Source Component Flat File Source Component
Bewerken	Derived Column Transformation Data Conversion
Splitsen	Conditional Split
Samenvoegen	Merge Join Union All
Laden	Flat File Destination Component OLE DB Destination Component
Verwerken post acties	Execute SQL Task
Afsluiten	Execute SQL Task

Afbeelding 4: ETL-stappen versus SSIS-componenten.

Microsoft.SQLServer.ManagedDTS. Een zoekactie met daarin één van deze assembly's als sleutelwoord levert veel relevante informatie op voor het genereren. We geven enkele codevoorbeelden voor het aanmaken van een nieuwe package, het aanpassen van de eigenschappen van de connection manager en het synchroniseren van de pipeline.

Aanmaken nieuwe package.

De eerste actie in de code betreft het instantiëren van een nieuwe package op basis van de bestaande template package. In onderstaand voorbeeld vormt een template package op het filesystem de basis voor de nieuwe package, dit kan uiteraard ook vanuit de SSIS-database of een andere database, waarin de template in het XML-datatype is opgeslagen.

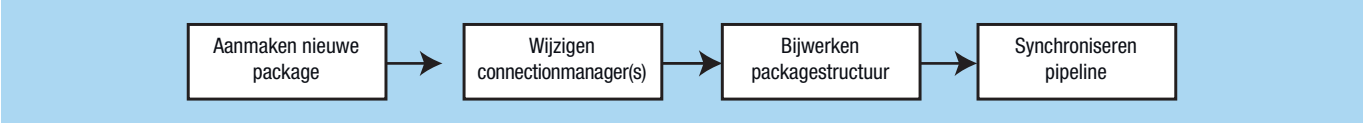
```
Application application = new Application();
Package package =
app.LoadPackage(@"D:\Templates\Template.dtsx",
null);
package.Name = "My Generated Package";
app.SaveToXml(@"D:\Packages\Package1.dtsx",
package, null);
```

Wijzigen connection manager(s).

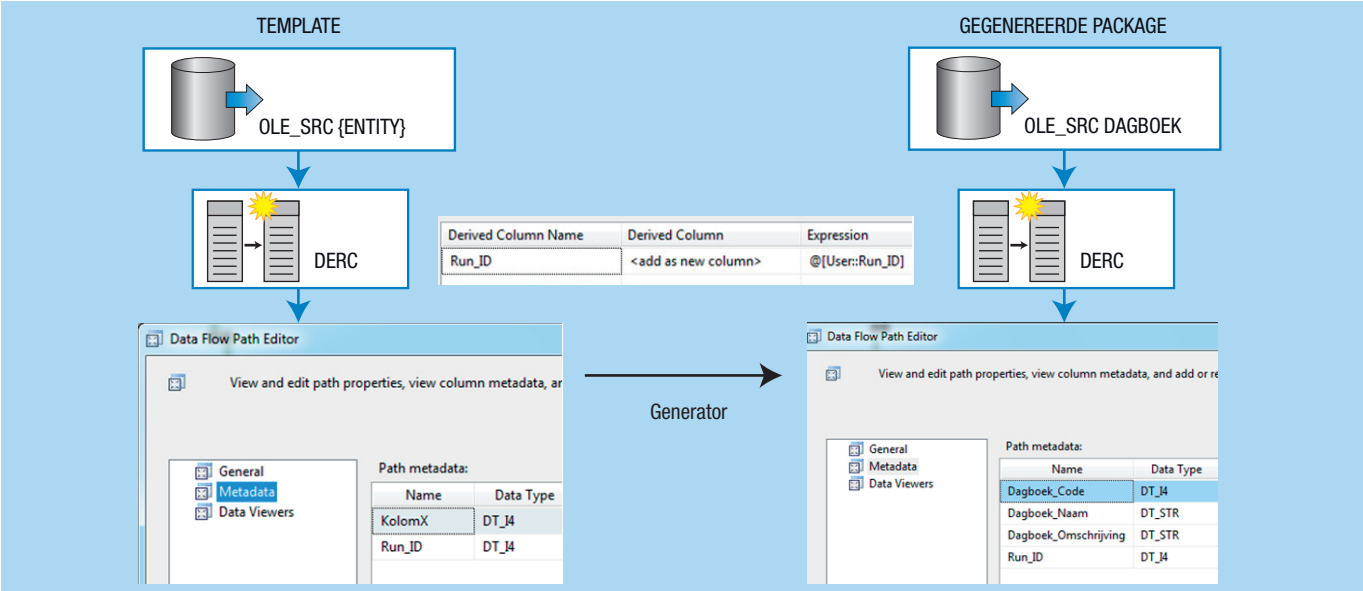
De tweede actie beteft het wijzigen van de connection manager. In het codevoorbeeld bovenaan op pagina 34 wordt een connection manager van het type flatfile aangepast. Hierbij moeten direct ook de package-metadata met betrekking tot de aanwezige kolommen en hun datatypes worden bijgewerkt.

Bijwerken structuur.

Na het aanpassen van de componenten in SSIS moet de onderliggende structuur ook worden gecreëerd. De onderliggende structuur wordt bepaald aan de hand van de ingevoerde metadata in combinatie met toegevoegde attributen uit de template(s). Een voorbeeld van een toevoeging in de template is een *derived*



Afbeelding 5:Vier vereiste stappen voor het genereren van een nieuwe package.



Afbeelding 6: Bijwerken structuur op basis van metadata en template.

column transformation die enkele auditgegevens toevoegt aan de pipeline en voor elke package identiek is, zie het Run_ID in afbeelding 6. Deze auditgegevens maken geen onderdeel uit van de ingevoerde metadata, maar dienen wel in de output te worden meegenomen. Dit wordt door de generator gerealiseerd. In het linkerdeel wordt de template afgebeeld, in de rechter de gegenereerde package. Het valt op dat alleen de kolommen gewijzigd zijn en het audit-gegeven (Run_ID), ongewijzigd is. Een andere component die invloed heeft op de structuur van de package is de *data conversion component*. Deze component wordt gebruikt om het datatype van een bronkolom om te zetten in een ander datatype. Hierdoor worden de package-metadata aangepast, de attributen binnen de pipeline worden immers omgezet in een ander datatype. In de GUI kan het datatype per attribuut worden opgegeven en via de data conversion component worden deze conversies (transformaties) doorgevoerd.

Synchroniseren.

Na bijwerken van de connection manager(s) en/of wijzigen van een component moeten de data- en de control-flow altijd worden gesynchroniseerd. Dit principe wordt standaard ook binnen de BIDS toegepast, denk aan de situatie waarbij een brontabel in SSIS is gewijzigd. Vervolgens dient de SSIS-package handmatig bijgewerkt te worden en ontstaan in de gehele data-flow waarschuwingen en/of foutmeldingen. Deze zijn alleen op te lossen door een zoek-en-vervang actie op de onderliggende XML of door het handmatig openen van elke gebruikte component. Het codevoorbeeld onderaan pagina 34 zet deze laatste actie per component automatisch in gang. De code herkent het componenttype aan het ComponentClassID.

Randvoorwaarden.

De generator is in staat om ETL-packages te genereren indien aan een aantal randvoorwaarden is voldaan:

1. Er is een functionerende SSIS-template beschikbaar. Voor alle te configureren componenttypes is sourcecode aanwezig. Indien dit niet het geval is, blijft de component ongewijzigd tijdens het genereren, met als gevolg dat de package faalt of een verkeerd resultaat oplevert;
2. De benodigde metadata zijn correct ingevoerd via de GUI;
3. Alle gebruikte database-objecten zijn reeds aanwezig in de database. Het genereren faalt wanneer deze objecten niet aanwezig zijn. De generator genereert de DDL indien het betreffende object niet aanwezig is. Indien deze DDL direct doorgevoerd moet worden, dan heeft de gebruiker schrijfrechten nodig op de onderliggende database(s).

Wordt vervolgd....

In het derde artikel worden de mogelijkheden met betrekking tot het genereren van het ETL-proces naar de datamart besproken. Daarnaast wordt ingegaan op enkele additionele eigenschappen van de GUI en wordt het loggingmechanisme besproken.

Ronald Bijlhouwer, **Richard Puijk** en **Vincent Wylenzek** zijn respectievelijk als Senior Developer, Medior Business Intelligence Consultant en Senior Business Intelligence Consultant werkzaam bij Ordina.