Efficiënt wijzigingen doorgeven van bron- naar doeldatabase

Change Data Capture (1)

Toon Loonen

In deze artikelreeks bekijken we de mogelijkheden om replicatieprocessen efficiënt uit te voeren en met name hoe we op de brondatabase de wijzigingen kunnen selecteren die naar de doeldatabase doorgegeven moeten worden.

Een groot deel van de gegevensverwerking in administratieve systemen bestaat simpelweg uit het schuiven (ofwel kopiëren of repliceren) van gegevens van het ene systeem of database naar het andere, bijvoorbeeld:
- Vanuit een administratief systeem (de bron) naar een ander administratief systeem (het doel of de bestemming), bijvoorbeeld de Klantgegevens worden gekopieerd van het CRM-systeem naar het Order Entry-systeem; de Ordergegevens worden van het Order Entry-systeem gekopieerd naar het financieel systeem;
- Vanuit diverse OLTP-systemen naar het datawarehouse of archief [Ref 4]: gegevens van zowel het CRM-systeem, het Order Entry-systeem en mogelijk nog enkele andere bronnen (financiële administratie, HRM enzovoort) worden gekopieerd naar het datawarehouse.

In dit artikel bekijken we de mogelijkheden om deze replicatieprocessen efficiënt uit te voeren en met name hoe we op de brondatabase de wijzigingen kunnen selecteren die naar de doeldatabase doorgegeven moeten worden. De nadelen van al dit kopiëren van gegevens zijn evident:
- Het kost extra veel schijfruimte;
- Het kost tijd om de gekopieerde gegevens bij te werken;
- De gekopieerde gegevens worden later bijgewerkt dan de bron en lopen dus uit de pas met de bron;
- Er is een kans op inconsistenties of fouten.

Maar vaak is er geen andere optie. Want diverse systemen van een organisatie maken gebruik van dezelfde gegevens, bijvoorbeeld de personeelsgegevens staan in het HRM (personeel en salaris) systeem, een systeem voor het plannen en uitvoeren van de werkzaamheden, het toegangspasjessysteem en het e-mailsysteem, ofwel: de office automatisering. Het is geen optie om daarvoor 1 geïntegreerd systeem met 1 database te kiezen. Of voor overzichten en statistieken moeten gegevens uit diverse systemen gecombineerd worden. De oplossing is dan meestal: kopieer de gewenste gegevens uit al deze bronsystemen naar een datawarehouse [Ref 9] en maak de overzichten of analyses vanuit dit datawarehouse.

Als de gegevens niet vanuit een primaire bron naar de andere systemen gerepliceerd zouden worden, dan zouden deze gegevens in elk van deze systemen handmatig ingevoerd moeten worden met, naast de hierboven genoemde nadelen, ook nog grotere kans op fouten.

Naast replicatie tussen systemen binnen een organisatie kennen we ook het versturen van gegevens tussen organisaties:
- Een order van een klant naar een leverancier wordt elektronisch verstuurd;
- Een opdracht naar de bank voor het doen van een aantal betalingen;
- Een belastingaangifte naar het belastingkantoor;
- Het via een webservice opvragen van gegevens van een persoon (bij het GBA), een bedrijf (bij de Kamer van Koophandel) of een adres (bij de Post) enzovoort.

In dit artikel bekijken we de mogelijkheden voor deze replicatie, primair tussen systemen binnen één organisatie. Onder replicatie verstaan we hier: de gegevens worden in 1 systeem ingebracht door een (apparaat of) gebruiker: dit is ofwel een medewerker van het bedrijf ofwel de klant via een web interface; deze wijzigingen worden doorgegeven naar de andere systemen binnen de organisatie die ook in deze gegevens geïnteresseerd zijn.
Als voorbeeld gebruiken we een systeem met Klanten, Orders en Artikelen met het gegevensmodel zoals in afbeelding 1 is te zien. De tabel CDC Batches wordt in een volgend artikel toegelicht.
Als dit systeem het doelsysteem zou zijn, dan zouden er bij elk record ook nog kolommen opgenomen kunnen zijn voor (uitleg volgt later):
- De bron: de code voor het bronsysteem;
- De identificatie (primary key) van dit object in het bronsysteem indien deze afwijkt van de identificatie in het doelsysteem;
- De datum en tijd dat dit object in het doelsysteem is toegevoegd of gemuteerd.

Referentiegegevens en transactiegegevens

In dit model zien we gegevens die een lang leven leiden, de Referentie Gegevens of Masterdata zoals Klant en Artikel, en

gegevens die een (relatief) kort bestaan leiden, de Transactiegegevens, zoals de Order en Orderregel.
De referentiegegevens zullen, in de loop van de tijd, wijzigingen ondergaan:
- Een Klant verhuist, krijgt een andere status enzovoort.
- Een Artikel krijgt een andere prijs of BTW-code, de voorraad wijzigt voortdurend.

Voor de transactiegegevens (Order en Orderregel) is het aantal wijzigingen minimaal of soms zelfs helemaal niet van toepassing. Een ander onderscheid is dat het belang van transactiegegevens vaak tot één systeem beperkt blijft: zij worden mogelijk nog naar een datawarehouse gekopieerd maar niet naar andere systemen. De masterdata worden wel vaak in meer systemen gebruikt: bijvoorbeeld de Klantgegevens worden, naast het Ordersysteem ook gebruikt in het CRM-systeem (marketing) en het financieel systeem (facturering) en natuurlijk ook in het datawarehouse.

> In plaats van een kolom met de timestamp van de laatste wijziging kan men ook gebruik maken van een vlag

Dit betekent dat deze masterdata van een primair systeem, waar ze worden ingevoerd, naar andere systemen gerepliceerd moeten worden. Masterdata Management [Ref 8] is de discipline die tracht na te streven:
- Uniformiteit van de definities van deze masterdata over de systemen heen;
- Eén bron waar de gegevens primair ingevoerd worden of tenminste zekerheid bieden dat de gegevens gelijk zijn over de systemen heen.

We zullen vaak terugvallen op deze twee begrippen (Referentie Gegevens of Masterdata respectievelijk Transactiegegevens) omdat deze twee typen gegevens vaak andere eisen stellen of mogelijkheden bieden bij de replicatie [Ref 8].

Het principe van replicatie

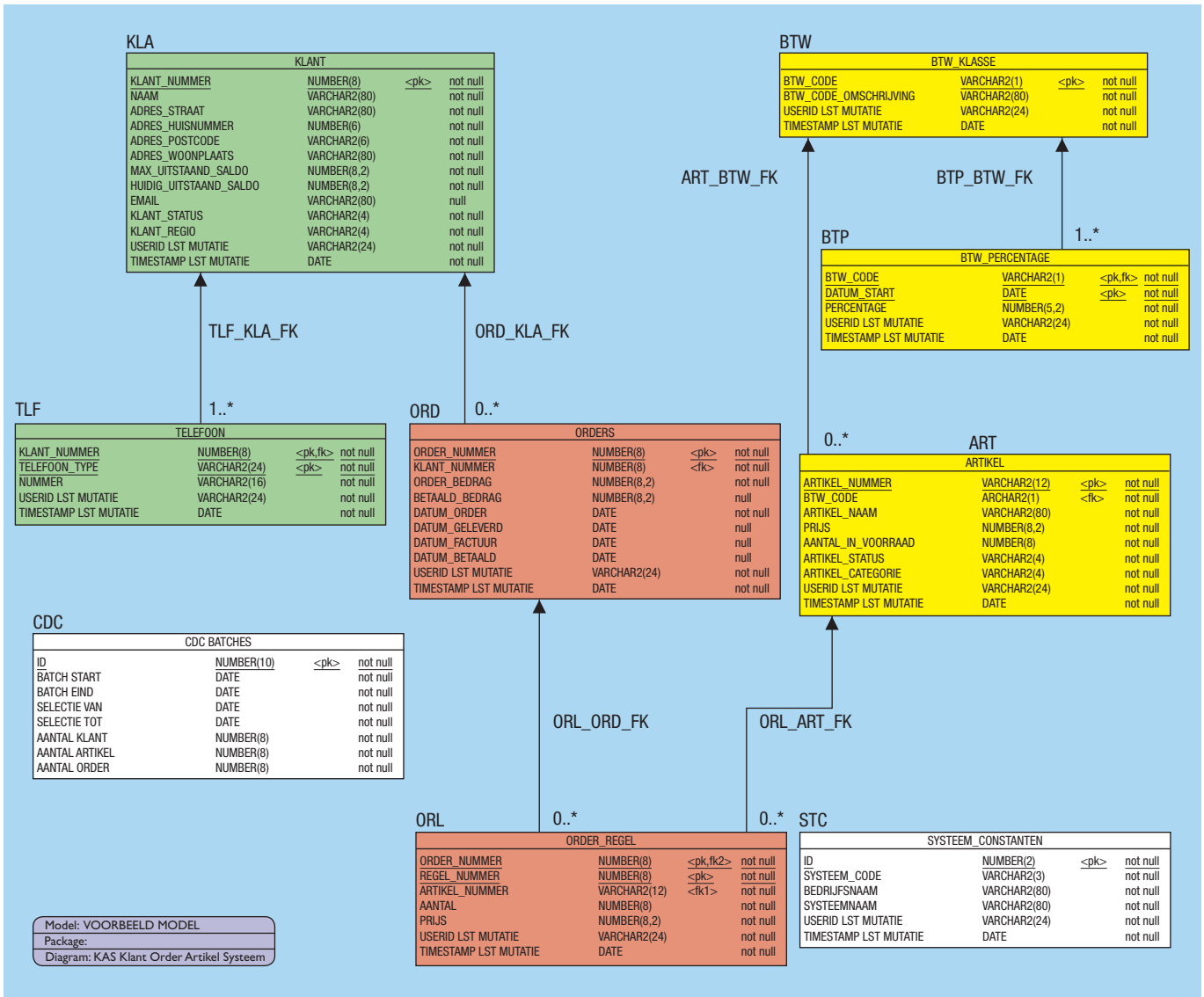
Er zijn verschillende manieren om een replicatie uit te voeren. De meest eenvoudige methode is om volledige tabellen van de brondatabase te kopiëren naar de doeldatabase, al of niet met behulp van een tussenbestand. Op de doeldatabase, kan men:
- De betreffende tabellen leeg maken en weer vullen met de inhoud van het tussenbestand;
- Per tabel de inhoud van de tabel en het tussenbestand samenvoegen (mergen): nieuwe records worden in de doeltabel toegevoegd (insert), gewijzigde records worden bijgewerkt (update) en records die zijn verwijderd op de bron worden ook verwijderd op de doeltabel of de status van deze records wordt gezet op 'verwijderd'.

Deze merge-optie impliceert het vergelijken van de inhoud van het tussenbestand met de inhoud van de tabel. Voor grote tabellen kan dit veel werk zijn. Ook het selecteren van alle rijen uit de brontabel, het vullen van het tussenbestand en het versturen van dit bestand over het netwerk naar het doelsysteem, is allemaal erg inefficiënt als slechts een klein percentage van de brongegevens is gewijzigd. In dit geval is het beter om alleen de gewijzigde records te selecteren aan de bron en naar het doelsysteem te versturen. Als er gekeken wordt naar welke gegevens nieuw of gewijzigd zijn dan is er nog een extra uitdaging om verwijderde gegevens te detecteren, wanneer deze gegevens ook fysiek zijn verwijderd; immers verwijderde records zijn er niet meer, dus daar valt niets meer aan te zien.

We zullen hier alleen ingaan op de replicatie van wijzigingen van de ene database of applicatie naar de andere. We zullen *niet* bespreken:
- De processen om een kopie van een database up-to-date te houden voor terugvaldoeleinden, dus om op door te draaien als de eerste database uitvalt. Dit kan vaak al gedaan worden met de hulpmiddelen van het RDBMS of van de leverancier van een SAN;
- De processen waarbij tabellen uit twee databases (A en B) over en weer informatie uitwisselen en database A kan worden bijgewerkt door database B en omgekeerd;
- Publish/subscribe: de mogelijkheid dat vele uiteenlopende bestemmingen geïnteresseerd zijn in een deelverzameling van de wijzigingen bij de bron. Bijvoorbeeld: De gemeente heeft een database met alle personen die in deze plaats wonen (de Gemeentelijke Basis Administratie of GBA). Het leger, als afnemer van deze gegevens, is alleen geïnteresseerd in wijzigingen bij mannelijke personen onder een bepaalde leeftijd. De fiscus is alleen geïnteresseerd in personen boven een bepaalde leeftijd. Een verzekeringsmaatschappij is alleen geïnteresseerd in personen die een verzekering hebben lopen bij deze maatschappij.

Dit laatste kan als zo'n verzekeraar specifiek aangeeft voor welke personen hij de wijzigingen wil hebben. En mogelijk is deze verzekeraar ook alleen geïnteresseerd in de wijzigingen van bepaalde attributen en zijn wijzigingen in andere attributen niet interessant en hoeven die ook niet doorgegeven te worden.

Verder is het mogelijk dat de functionaliteit in de twee systemen (bron en doel) verschillend gemodelleerd is:
- Het doelsysteem kan een datawarehouse zijn waar ook de historie van de gegevens wordt bewaard, terwijl op de bron alleen de huidige/laatste stand opgeslagen is;
- De bron kan een aantal details bevatten waarin het doelsysteem niet geïnteresseerd is. Bijvoorbeeld: in het bronsysteem kunnen van een Klant meerdere Telefoonnummers worden bijgehouden (zie ons voorbeeldmodel) terwijl het doelsysteem alleen geïnteresseerd is in één Telefoonnummer, waardoor ook de noodzaak voor een extra tabel vervalt.



Afbeelding 1: Voorbeeld gegevensmodel.

Alleen de wijzigingen doorgeven

Zoals hiervoor al gesteld: het is efficiënter om (periodiek of direct) alleen de wijzigingen door te geven dan telkens hele (soms zeer grote) tabellen of bestanden over te halen naar het doelsysteem. Dit kan gedaan worden op verschillende manieren:

1. Voor elke tabel in het bronsysteem definiëren we een kolom met de tijd wanneer dit record is toegevoegd of bijgewerkt. Periodiek (iedere minuut, uur, dag, week of maand, afhankelijk van de eisen) draait er een proces dat de wijzigingen oppakt sinds het begin van de vorige run. Dit proces creëert vervolgens een tussenbestand (of per tabel een bestand) en stuurt dit bestand naar het doelsysteem alwaar het in de database wordt verwerkt [Ref 2];
2. Een database trigger wordt gebruikt om de veranderingen naar het tussenbestand te schrijven. Waarschijnlijk is dit tussenbestand hierbij ook een database (hulp)tabel, die later wordt omgezet naar een fysiek bestand of een bericht. Maar het is ook mogelijk dat een proces op het doelsysteem direct deze hulptabel leest via een database-to-database connectie [Ref 1];
3. Het logbestand van de database wordt gebruikt om de veranderingen op de database-tabellen te detecteren en in het tussenbestand vast te leggen. Zie voor uitdagingen hierbij [Ref 6];
4. Met behulp van partitionering: voor grote tabellen met dagelijkse transacties die na het inserten niet meer veranderen, bijvoorbeeld een tabel met de verkoopdetails (Orders en Orderregels, Call Detail Records), is het mogelijk om dagelijkse partities te definiëren. Deze partities bevatten de transactiegegevens per dag. Na deze dag zullen er geen nieuwe records worden toegevoegd aan deze partitie. Een dagelijkse batch kan dan alle rijen van deze partitie snel en efficiënt lezen en deze verzenden naar de bestemming.

De laatste (vierde) optie geeft de beste performance, maar is alleen haalbaar voor (grote) tabellen met partities en met transactiegegevens die (bijna) nooit veranderen. Het is niet zinvol voor de master/referentiegegevens.

De derde optie met het logbestand geeft geen overhead op de database server omdat er geen selecties op de database zelf worden uitgevoerd. Het is echter niet mogelijk voor alle bronnen/DBMS-producten en vergt speciale kennis van de structuur van het logbestand.

Wanneer een ETL-tool (Extract, Transfer, Load) wordt gebruikt om de gegevens/wijzigingen van de bron te selecteren en als deze ETL-tool het logbestand kan lezen, dan is dit een aanbevolen optie. Merk op dat de structuur van het logbestand niet gestandaardiseerd is en dat DBMS-leveranciers deze structuur ook van de ene versie naar de andere kunnen wijzigen. De leverancier van een ETL-tool zal zijn product up-to-date houden met deze versies van het DBMS.
De tweede optie, het gebruik van database triggers, heeft extra overheadkosten aan de kant van de (bron)database server, net als alle trigger-verwerking. Het voordeel is dat de veranderingen direct na de wijziging kunnen worden verzonden naar de bestemming. Voor het real-time up-to-date houden van de doeldatabase is dit de beste optie. Ook voor fysieke deletes is dit een goede optie als de derde optie niet mogelijk is.
De eerste optie is waarschijnlijk het meest eenvoudig te implementeren, mits de brontabellen allen deze timestamp hebben inclusief een index op deze kolom. Deze optie is, net als de laatste optie, niet haalbaar voor real-time updates van de doeldatabase. Het kan alleen worden gedaan in periodieke batches.

Er zijn verschillende manieren om een replicatie uit te voeren

Een andere optie voor de selectie van de mutaties is dat deze niet verzameld worden uit de *brondatabase* maar dat de *bronapplicatie* verantwoordelijk is voor het verzamelen van de mutaties en het samenstellen van het tussenbestand. Dit is echter een foutgevoelige werkwijze: fouten in de programmatuur leiden snel tot gemiste gegevens in het doelsysteem. Het is alleen zinvol als alle inserts en updates op de database worden gedaan via een algemene routine die ook verantwoordelijk is voor het doorgeven van de wijzigingen naar het tussenbestand.
Wanneer het bronsysteem een pakket is met een zeer complexe datastructuur die afwijkt van de logische datastructuur, dan zal een functie of service van het pakket gebruikt moeten worden om de wijzigingen te kopiëren naar het tussenbestand.
In plaats van een kolom met de timestamp van de laatste wijziging kan men ook gebruik maken van een vlag (replicatie_status_vlag) met de waarden 0 (voor nog niet gerepliceerd) en 1 (voor wel gerepliceerd). Bij de replicatie worden alle records geselecteerd waarvoor deze vlag de waarde 0 heeft. De geselecteerde gegevens worden naar het tussenbestand geschreven en de vlag wordt op 1 gezet. Deze optie heeft echter enkele nadelen ten opzichte van de timestamp:
- De timestamp geeft meer informatie die ook voor andere doeleinden gebruikt kan worden, zoals auditing en concurrent update detectie [Ref 5];
- De update van de vlag van 0 naar 1 geeft veel overhead.

Verder is er natuurlijk de vraag welke optie in ieder geval mogelijk is. Bij een pakket zal het vaak niet mogelijk of toegestaan zijn om zelf het gegevensmodel te wijzigen om een timestamp aan elke tabel toe te voegen of om zelf triggers of partitionering op de tabellen aan te brengen.
Afhankelijk van de eisen en mogelijkheden zal een van deze opties of een mix van opties gekozen moeten worden. De aanbevolen volgorde van de opties is:
- Voor real-time replicatie is de tweede optie (triggers) de meest haalbare;
- Voor near real-time zal de derde optie (log-bestand) waarschijnlijk ook goed bruikbaar zijn;
- Voor grote tabellen met de dagelijkse partities en een dagelijkse batch is optie vier (dagelijkse partities) aan te bevelen;
- Voor andere batch-interfaces is de derde optie (het log-bestand) de beste keuze, indien mogelijk, anders wordt de eerste optie (timestamp) aanbevolen;
- Voor kleine tabellen met referentiegegevens is het volledig overhalen van deze gegevens (al of niet gewijzigd) in plaats van alleen de wijzigingen soms ook een optie: het is eenvoudiger te bouwen en de (hier) kleine overhead compenseert een ingewikkelder selectieproces.

In het volgende deel gaan we voornamelijk uit van de eerste optie van de hierboven genoemde alternatieven, dus de timestamp op elke record. Maar bij de andere opties zal het meeste nog steeds van toepassing zijn of kan het op een vergelijkbare manier geïmplementeerd worden.

Literatuur

1. *Loonen. Gegevenskoppelingen in een 4GL/RDBMS omgeving. Database Magazine 1996/8.*
2. *Loonen. Mutatierapportage, de tijdgeest van de database. Database Magazine 1999/3.*
3. *Loonen. Performance. Database Magazine 2002/1,2,3.*
4. *Loonen. Wat moeten we archiveren en op welk formaat. Database Magazine 2003/1.*
5. *Loonen. Foutafhandeling in SQL coding. Database Magazine 2003/2.*
6. *Wikipedia: http://en.wikipedia.org/wiki/Change_data_capture*
7. *Informatica: www.informatica.com*
8. *Wikipedia: http://en.wikipedia.org/wiki/Master_data_management*
9. *Wikipedia: http://nl.wikipedia.org/wiki/Datawarehouse*
10. *Kimball: www.rkimball.com/html/designtipsPDF/DesignTips2005/DTKU63BuildingAChangeDataCaptureSystem.pdf*
11. *Loonen. Gebruik van speciale tekens in de database. Database Magazine 2002/8.*

Toon Loonen (toon.loonen@capgemini.com) is werkzaam bij Capgemini.