Efficiënt wijzigingen doorgeven van bron- naar doeldatabase

Change Data Capture (2)

Toon Loonen

In deze artikelserie 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.

In dit en het volgende deel gaan we voornamelijk uit van de eerste optie van de in deel 1 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.

Tussenbestand

Voor de indeling van het tussenbestand hebben we de volgende alternatieven:

1. Een CSV (Tab gescheiden) ASCII-bestand per tabel of per set tabellen, bijvoorbeeld een bestand met alle wijzigingen op de Artikel-tabel, een tweede bestand met de wijzigingen op de Klanten en een derde voor de mutaties op Orders en Orderregels (met een record type als de eerste kolom in het bestand). CLOB- of BLOB- (Character of Binary Large Objects) kolommen, zoals een grote tekst of XML-kolom, een foto, een Word-bestand enzovoort) zijn hiermee niet mogelijk. Eventueel kunnen deze als separate files meegestuurd worden;
2. Een ASCII-bestand waarbij alle kolommen op een vaste positie beginnen. Dit betekent dat veel alfabetische kolommen met spaties aangevuld moeten worden tot de afgesproken lengte. Ook hier zijn geen CLOB- of BLOB-kolommen mogelijk;
3. Een XML-bestand. Deze optie vereist meer ruimte in verband met de tags, maar is eveneens meer flexibel dan de voorgaande optie. Hiermee kunnen eventueel ook CLOB- of BLOB-kolommen worden meegenomen in het tussenbestand zelf. Waarschijnlijk is het efficiënter om het XML-bestand te comprimeren voordat het via het netwerk wordt verstuurd;
4. Een (SOAP, XML) bericht. Deze optie heeft de voorkeur als batch of individuele transacties worden doorgegeven naar de bestemming met behulp van een ESB (Enterprise Service Bus);
5. Een werktabel in de brondatabase. Deze werktabel met de wijzigingen wordt door het doelsysteem direct gelezen via een database-to-database connectie [Ref 1];
6. Er is helemaal geen tussenbestand: Door een proces (periodiek lopend SQL script of een stored procedure) op de doeldatabase worden de wijzigingen op de brondatabase direct gelezen (via een database-to-database connectie) en in de doeldatabase verwerkt [Ref 1].

Als, bij een CSV-bestand, een Tab teken gebruikt wordt om de kolommen te scheiden, dan mag de inhoud van de kolommen zelf geen Tab (of Return of andere speciaal teken) bevatten. Als een ander teken gebruikt wordt, bijvoorbeeld een komma, en enkele kolommen kunnen een komma bevatten, dan moet de waarde van deze kolommen tussen (dubbele) quotes geplaatst worden en een eventuele quote in de tekst wordt vervangen door twee quotes, zie tabel 1. Op deze manier is de oorspronkelijke waarde altijd te herstellen.

Een alternatief, om de omvang van het tussenbestand te beperken, is om in het tussenbestand alleen de identificaties (primary key) en de actie (insert, update) van de gewijzigde objecten op te nemen. Het proces dat de wijzigingen in de doeldatabase verwerkt leest dan, aan de hand van deze identificerende waarden de records op de brondatabase via een database-to-database connectie en verwerkt de wijzigingen op de doeldatabase.

Met name voor LOB's zoals (Word of Excel) documenten of foto's van werknemers of Artikelen, is het interessant om een eigen timestamp van deze kolom bij te houden. Het is namelijk niet efficiënt om ook de foto van het Artikel naar een andere database over te halen als alleen de prijs van het Artikel is gewijzigd.

Verwerken van wijzigingen in doeldatabase

Een proces op de doeldatabase of in het doelsysteem pakt het tussenbestand of de berichten op en verwerkt de wijzigingen in de tabellen. In het algemeen zal een insert, update of delete op de bron gevolgd worden door eenzelfde actie op de bestemming, maar veel andere opties zijn mogelijk, vooral wanneer een datawarehouse de bestemming is:

- Als er geen verwijderingen zijn toegestaan op de doeldatabase, dan zal een verwijdering op de bron gevolgd worden door een update van de status van het betreffende record op de doeldatabase;
- Als de complete historie van een object op de doeldatabase wordt bewaard, dan zal elke update en verwijdering bij de bron leiden tot een toevoeging van een versierecord van het object in de doeldatabase;
- Het Klant record op de bron met Klantnummer, Naam, Adres en enkele andere attributen kan op de bestemming worden

Kolom in bron	Kolom in tussenbestand	Kolom in doel
AAA, AAA	“AAA, AAA”	AAA, AAA
AAA “AAA” AAA, AAA	“AAA ““AAA”” AAA, AAA”	AAA “AAA” AAA, AAA
AAA ““AAA”” AAA	“AAA ““““AAA”””” AAA”	AAA ““AAA”” AAA

Tabel 1

gesplitst over enkele tabellen, bijvoorbeeld de Klant tabel en de Postcode tabel (met straatnaam en woonplaats als attributen). Een mutatie op de Klant tabel leidt dan tot een mutatie op de Klant tabel op de doeldatabase en mogelijk ook nog een extra mutatie op de Postcode tabel;
- Sommige RDBMS producten hebben een MERGE statement. Aan de hand van de mutaties in het tussenbestand en de inhoud van de doeltabel zal het RDBMS zelf uitzoeken of een insert, update of delete (of mogelijk helemaal geen actie) nodig is. Als de primaire sleutel waarde al bestaat in de doeltabel, dan wordt een update gedaan, anders een insert. De waarde van de primaire sleutel mag hierbij natuurlijk nooit veranderen!

Wanneer twee of meer bronnen worden gebruikt om een tabel te vullen op de doeldatabase, bijvoorbeeld een nieuwe postcode kan komen van de Klanttabel of met een Postcode batch van het postkantoor, dan moet het systeem een keuze maken, bijvoorbeeld:
- Een Postcode, aangeleverd via het postkantoor wordt altijd toegevoegd; Indien deze Postcode al bestaat en de waarde van straat en/of stad verschilt van de huidige waarde in de database, dan worden de straatnaam en woonplaats bijgewerkt met de nieuw aangeleverde waarde;
- Een nieuwe Postcode, aangeleverd via de Klanttabel wordt ook toegevoegd;
- Voor een Postcode, aangeleverd via de Klanttabel die al bestaat in de database maar waarvan de straatnaam of de woonplaats anders zijn dan de huidige waarde in de database, zijn er twee mogelijkheden: is de huidige waarde in de database afkomstig van het postkantoor, dan wordt deze niet overschreven; maar is de huidige waarde afkomstig van de Klantentabel, dan wordt deze wel overschreven/bijgewerkt.

Het idee hierachter is dat de waarde, afkomstig van het postkantoor, de officiële naam is en dat daar geen schrijffouten in zitten. De meeste waarden zullen ook door het postkantoor aangeleverd worden. Maar deze werkwijze maakt het mogelijk om postcodes, die nog niet door het postkantoor zijn geleverd, toch in het bestand op te nemen.
Het advies is dus ook om wanneer er meer bronnen voor een object zijn:
- Eén van deze bronnen als primaire bron aan te wijzen [Ref 8];
- Aan te geven van welke bron de huidige waarde afkomstig is door middel van een bron_code als extra kolom in de doeltabel.

Het is mogelijk dat een wijziging twee keer wordt aangeleverd, bijvoorbeeld als de brondatabase is hersteld door het terugplaatsen van een backup van de dag ervoor. Het is dan mogelijk dat de mutatie opnieuw wordt verstuurd. Daarom moet het proces op het doelsysteem dat de mutaties verwerkt daar altijd op verdacht zijn en dubbele mutaties bijvoorbeeld negeren.
Een andere situatie doet zich voor als er meer wijzigingen na elkaar komen op dezelfde kolom van hetzelfde record. In feite zou dan alleen de laatste wijziging verwerkt behoeven te worden. Maar het is waarschijnlijk eenvoudiger om het doelrecord een paar keer te updaten.
Dit is ook het geval als er twee correcties op 1 record gedaan worden die elkaar opheffen, bijvoorbeeld: De naam van een Klant wijzigt kort na elkaar, binnen één mutatiebatch, van “Jansen” naar “Janssen” en weer terug naar “Jansen”. Een dubbele update is dan niet meer nodig, maar kan geen kwaad.

Audit trail

Vaak is het gewenst om van een object, een Klant of Artikel, in de doeldatabase te achterhalen waar deze gegevens vandaan komen, bijvoorbeeld voor onderzoek als de gegevens van dit object onwaarschijnlijk over komen. Aan de hand van een audit trail kan dan het spoor naar de bron worden gevolgd en kan men daar ook de gegevens van het object bekijken en daarmee mogelijk de oorzaak van een fout achterhalen.
Om een volledige audit trail van een doelsysteem naar de bron mogelijk te maken moet van elk gerepliceerd object (record) de bron worden aangegeven, dus het systeem of database waar dit object vandaan komt, de ID van het object in dat systeem en eventueel het volgnummer van de batch waarmee dit object is gecreëerd of bijgewerkt. Mogelijk staat bij de bron ook weer een verwijzing naar een daaraan voorafgaand bronsysteem.

Referentiële integriteit

Op de doeldatabase is het mogelijk dat fouten, bijvoorbeeld met verwijzingen of referentiële integriteit, ontstaan bij de verwerking van de mutaties. Bijvoorbeeld een nieuwe Order met enkele Orderregels is ontvangen, maar de bijbehorende Klant informatie staat nog niet in de doeldatabase. Dit kan voorkomen worden door de verwerking van de gegevens in de juiste hiërarchische volgorde uit te voeren. Maar dit zal niet altijd alle problemen voorkomen!
Bijvoorbeeld: stel dat we bij de bron een batchproces te starten. Dit proces zal eerst alle wijzigingen op de Klant tabel verzamelen, dan alle wijzigingen op de Artikel tabel en vervolgens de nieuwe of gewijzigde Orders met Orderregels. Indien, tijdens de verwerking van de Artikelen, een nieuwe Klant wordt toegevoegd en meteen ook een Order voor deze Klant, dan wordt deze

Order wel meegenomen naar het doelsysteem maar de bijbehorende Klant niet: immers de wijzigingen op deze tabel waren al verzameld. Dit zou het hiervoor genoemde probleem met de referentiële integriteit kunnen veroorzaken.
Deze situatie kan men voorkomen met de volgende werkwijze:
- Aan het begin van de batch voor het verzamelen van mutaties wordt de systeemtijd min 2 seconden (tijd Y) geregistreerd in een hulptabel;
- Uit deze hulptabel halen we de starttijd van de vorige batch op (tijd X);
- Vervolgens verzamelen we alle wijzigingen op de Klant tabel, de Artikel tabel en de Orders waarvan de tijd van wijziging ligt tussen X en Y (groter dan X en kleiner dan of gelijk aan Y);
- Vervolgens wordt de tijd Y als de starttijd voor de volgende batch in de hulptabel vastgelegd;
- Deze hulptabel kan één record bevatten met alleen de tijd van de laatste batch of de historie van alle mutatieruns met de betreffende tijd Y en eventueel andere kengetallen.

In afbeelding 1 (zie DB/M 4, juni 2011, pagina 36) zien we een voorbeeld voor deze hulptabel (tabel CDC Batches) waarin per batch wordt bijgehouden:
- Een ID, betekenisloos volgnummer;
- De starttijd en eindtijd van de batch waarin mutaties verzameld zijn;
- De begintijd en eindtijd van de selectie: de tijd X en Y van de beschrijving hierboven; X zal altijd gelijk zijn aan Y in het voorgaande record, deze kolom zou dus weggelaten kunnen worden;
- Het aantal Klanten, Artikelen en Orders dat verzameld en doorgegeven is.

Met deze werkwijze wordt het probleem van een Order zonder de bijbehorende Klant voorkomen: de voornoemde nieuwe Klant en Order worden beide niet geselecteerd in deze batch maar in de volgende batch. De 'systeemtijd minus 2 seconden' wordt hier gebruikt in plaats van de systeemtijd zelf om het volgende probleem te verhelpen:
- Een Klant wordt ingevoegd op tijd Z (ligt tussen X en Y), maar deze mutatie wordt nog niet meteen gecommit;
- Wanneer het batchproces de mutaties op de Klanttabel leest zal dit proces deze (niet committed) mutatie niet zien en dus ook niet meenemen;
- Ook in de volgende run wordt deze Klantmutatie ook niet meer meegenomen (de tijd Z is nu kleiner dan de nieuwe waarde van tijd X).

Daarom trekken we 2 seconden af van de systeemtijd zodat we zeker zijn dat intussen alle transacties gecommit zijn. Als er zeer lange transacties mogelijk zijn (dit liever voorkomen), neem dan een langere periode (10 seconden, 1 minuut) in plaats van de genoemde 2 seconden. Alleen in een zeer eenvoudige situatie waarbij alleen overdag (tussen 08.00 en 18.00 uur bijvoorbeeld) wijzigingen op de database worden aangebracht en de mutaties 's nachts worden verzameld, kan de werkwijze vereenvoudigd worden en kan men 's nachts gewoon alle mutaties van de afgelopen dag selecteren.
Het voorgaande geeft aan dat het selecteren van wijzigingen niet altijd zo triviaal is. Kimball is daarom zeer afwijzend om deze werkwijze met een datum en tijd te gebruiken [Ref 10].
Een andere mogelijkheid voor problemen met de referentiële integriteit ontstaat wanneer gegevens afkomstig zijn uit verschillende bronnen: De Klant gegevens zijn afkomstig uit het CRM-systeem en de Order gegevens zijn afkomstig uit het Order Entry-systeem. Als het proces voor het repliceren van de Klantgegevens is mislukt (of later draait), dan worden de Ordergegevens mogelijk wel verwerkt maar met een verwijzing naar een niet bestaande Klant. Dit zal in een volgende run wel weer worden rechtgetrokken.

De aanbeveling is hier afhankelijk van het type doelsysteem:
- Een andere productiesysteem: Facturering ontvangt gegevens van een Orderverwerkingssysteem;
- Een datawarehouse: ontvangt gegevens van diverse operationele systemen.

De interface naar een ander productiesysteem moet altijd volledig en juist zijn, anders worden mogelijk enkele Orders niet gefactureerd.

Voor een datawarehouse, dat gebruikt wordt voor statistische analyses, is het aanvaardbaar om een keer een mutatie te missen: De statistieken zullen nauwelijks wijzigen als er enkele records ontbreken. Van belang is dat tenminste de datatypes correct zijn (geldige datums en getallen enzovoort) en primaire sleutels uniek. De fouten kunnen gelogd worden en een beheerder kan deze logs beoordelen. Sommige fouten worden automatisch gecorrigeerd in de volgende run omdat de gegevens mogelijk in de verkeerde volgorde binnenkomen (zie hierboven). Andere fouten worden veroorzaakt door een fout in het bronsysteem en moeten worden gemeld aan de beheerder van dat systeem zodat ze bij de bron gecorrigeerd worden. De volgende replicatiebatch zal dan deze correcties meenemen naar de doeldatabase waarmee de inconsistentie wordt hersteld.
Wanneer er fouten of inconsistenties (zoals foreign keys zonder gerelateerde primary key) worden geaccepteerd tijdens het laadproces, dan moeten alle programma's en/of rapporten die deze gegevens gebruiken zo worden gebouwd dat zij niet vastlopen op deze inconsistenties. Daarnaast is de kwaliteit van de output (rapporten) altijd afhankelijk van de kwaliteit van de input, dus zowel van de kwaliteit van de gegevens in het bronsysteem alsook van de kwaliteit van de interface.

Controletellingen en datakwaliteit

Om zeker te zijn dat men geen mutaties mist kan men ook regelmatig (periodiek of bij elke mutatiebatch) een controletelling uitvoeren:

- Klopt het aantal Klanten, Orders en Artikelen in het bronsysteem met het doelsysteem;
- Klopt het totaal van de uitstaande saldo's op de Klantentabel in het bronsysteem met hetzelfde getal in het doelsysteem;
- Klopt de som van de voorraad van de Artikelen in de twee systemen ook?

Vaak wordt getracht om in de interface ook de kwaliteit van de gegevens te verbeteren, bijvoorbeeld dubbele personen één keer door te geven, te zorgen dat adressen kloppen enzovoort. Ik denk zelf dat het beter is om de kwaliteit van de gegevens direct in het bronsysteem te verbeteren dan om de interface hiermee te belasten.

Het kan zijn dat de bron en het doel verschillende tekensets hanteren, bijvoorbeeld ASCII of EBCDIC op de bron en UTF8 op het doelsysteem. Hiermee moet bij de conversie rekening gehouden worden [Ref 11].

Selectie op een beperkte set van tabellen

Bij ons voorbeeldmodel moeten we acht tabellen scannen op wijzigingen. In een echte productieomgeving zullen dit veel meer tabellen zijn. Een alternatief is om alleen de belangrijkste tabellen te scannen: Klanten, Orders, Artikel en BTW klasse.
Als bij een bestaande Order alleen een enkele Orderregel is gemuteerd (aantal besteld is gewijzigd) of toegevoegd dan wordt de timestamp van deze Orderregel bijgewerkt, maar ook de timestamp van de Order wordt bijgewerkt ook al waren er geen wijzigingen op het Orderrecord zelf. Als nu een bestaande Order is gewijzigd (ofwel de Order header of een of meer Orderregels), dan kan aan de hand van alleen de timestamp op de Order de wijziging geselecteerd worden en is geen separate scan op de Orderregel tabel meer nodig.

Het is belangrijk om de timestamp waarden van de laatste wijziging (de kolom Datumtijd_laatste_mutatie) hetzelfde te houden voor alle records binnen één transactie. Dit betekent:
- Aan het begin van een transactie wordt eerst deze timestamp (systeemtijd) opgehaald;
- Vervolgens wordt die tijd gebruikt als timestamp voor alle insert/update-acties binnen de transactie.

Dit voorkomt dat bij de verwerking van een Order met Orderregels, een deel van de Orderregels wordt gerepliceerd in een eerste batch en een tweede deel in een volgende.

Selecties op de brontabellen

Misschien moeten niet altijd alle records van een tabel worden gerepliceerd naar de doeldatabase. Bijvoorbeeld: het datawarehouse is alleen geïnteresseerd in actieve of afgesloten Klanten, niet in Klanten met de status van prospect. Dus de Klanten met deze status worden nooit geselecteerd voor replicatie. Zodra de status van een Klant wijzigt van prospect naar actief, dan wordt ook de timestamp bijgewerkt en wordt deze Klant geselecteerd in de volgende mutatiebatch.

Tools

Er is een divers aantal tools beschikbaar om de replicatie van wijzigingen van een brondatabase naar een doeldatabase uit te voeren:
- Standaard SQL scripts of stored procedures die de mutaties op de brondatabase via een database connectie lezen en direct verwerken in de doeldatabase;
- ETL (Extract, Transfer, Load) tools of data integratie tools van verschillende leveranciers [zie Ref 6, 7];
- Replicatie servers van de meeste RDBMS leveranciers.

De ETL-tools hebben een aantal voordelen boven SQL scripts, zoals:
- Een grafische user interface voor het tekenen/ontwerpen van de transformaties;
- Interfaces maken tussen diverse heterogene omgevingen, variërend van (object of relationele) databases van verschillende leveranciers, XML-berichten en gewone ASCII-bestanden, zowel aan de bron- als aan de doelkant;
- Het lezen van logbestanden van een RDBMS;
- Het lezen van gegevens uit (CRM, ERP, HRM, enz.) pakketten;
- Tekenset conversie;
- Monitoring en logging van de replicatieprocessen;
- Automatisch herstel na storingen enzovoort.

In een heterogene omgeving kan een ETL-tool dus veel voordelen hebben. In een omgeving met als bron- en doelomgeving een RDBMS van dezelfde leverancier, kan het net zo gemakkelijk en efficiënt zijn om SQL scripts te gebruiken voor het replicatieproces. Bij complexe conversies zal meestal ook een extra stukje SQL code of een stored procedure nodig zijn dat door het ETL-tool wordt aangeroepen.
Voor de selectie van een ETL-tool kan dit artikel een groot aantal requirements aanreiken, waaronder met name de hierboven genoemde voordelen boven standaard SQL scripts.

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: http://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.