In groter tempo veranderingen doorvoeren in een datamodel

Beheersbare database-evolutie met deltasql

Rick van Rein

Meestal beschouwen we databases als stabiele wijzen van opslag voor onze data. In de realiteit van alledag ondergaat software echter versies en revisies, met ook regelmatig invloed op de databaseschema's. Hoog tijd voor een tool die dit proces beter beheersbaar maakt.

De meeste software wordt vanuit een heldere visie geprogrammeerd en dient vaak een enkel doel. De praktijk is echter dat gebruikers een zekere invloed gaan uitoefenen, met argumentatie als "ik mis nog dat ene stukje, kan dat er bij?" In het commerciële traject zegt een verkoper zoiets regelmatig toe; in de open source wereld zijn het regelmatig de developers zelf. Feit is dat gebruikers bijna altijd patronen aandragen die bij de oorspronkelijke ontwikkeling van een stuk software niet voorzien waren. Bovendien is ook bekend dat innovatie vaak een traject doorloopt van statisch en vastomlijnd naar dynamische structuren. Beide zijn er de oorzaak van dat de conceptuele modellen van software veranderen terwijl die tot volle wasdom groeit.
Op zich is software nog wel te upgraden, maar alles wat persistent is moet met zorg worden overgezet op nieuwe versies. Dat geldt voor vrije vorm bestanden zoals de configuratie van een stuk software, en dat verklaart modulaire en uitbreidbare mechanismen als XML en een Windows repository. Maar de applicatiedata vormen een probleem op zichzelf; die zullen ook mee moeten bij een update van de software. Precies daarvoor bestaat deltasql.

Nieuw schema

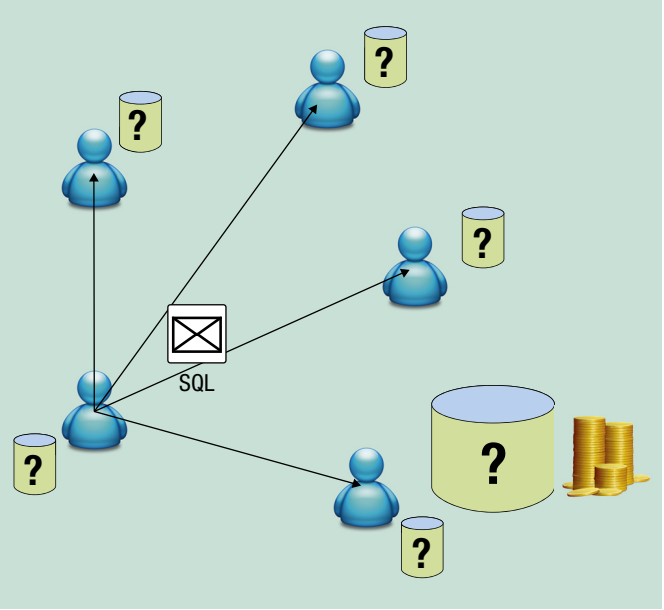
Een ontwikkelaar heeft op zijn ontwikkelmachine een omgeving waarin een database draait die voldoet aan de nieuwste structuren die zijn ontwikkelende applicatie nodig heeft. Hierin probeert hij, ontdekt fouten en repareert ze. Natuurlijk is al dit werk incrementeel; software wordt gemaakt door aanpassingen opeen te stapelen, en ook de ontwikkeldatabase wordt zo beheerd.
Dus in plaats van telkens een verse database in te laden, zal de ontwikkelaar instructies geven zoals ALTER TABLE en CREATE TABLE. Daarnaast kan het ook nodig zijn om gegevens te transformeren met een UPDATE, of in een VIEW samen te vatten, enzovoort.
Als op een gegeven moment het nieuwe model stabiel genoeg is voor een release, dan zal deze ontwikkelaar het script aanpassen waarmee een verse database kan worden aangemaakt. Los daarvan ziet hij zich vaak genoodzaakt een transitiescript te maken waarmee de gebruiker kan upgraden van een oude versie van de database naar een nieuwe.
Toch heeft hij in wezen al dit werk al gedaan op zijn ontwikkelsysteem. Al die ALTER TABLE en CREATE TABLE statements op zijn ontwikkeldatabase zouden hergebruikt kunnen worden voor het transitiescript. Sterker nog, beginnend met een lege database zouden zulke stappen achter elkaar gedraaid kunnen worden om een initiële database aan te maken voor nieuwe gebruikers. Immers, efficiëntie is niet echt belangrijk voor dergelijke eenmalig draaiende scripts. Voor de ontwikkelaar van een applicatie is deltasql overigens op nog een andere manier nuttig; hij kan samenwerken met andere ontwikkelaars, en de updates van die ontwikkelaars ook in zijn ontwikkeldatabase doorvoeren wanneer hij de software met een upgrade bijwerkt.
Dit heeft als voordeel dat zijn database niet gedeeld hoeft te worden met die van andere ontwikkelaars. Dat is erg prettig, omdat ontwikkelomgevingen voortdurend fluctueren tussen stabiel en *crashend*, en als zo'n omgeving gedeeld zou worden door ontwikkelaars, dan zou een ontwikkelomgeving ontstaan die alleen stabiel is als alle developers tegelijk hun deel stabiel hebben. Dat geeft noodzakelijke afhankelijkheden tussen ontwikkelaars die ten koste gaan van de voortvarendheid van de verdere ontwikkeling. De mogelijkheid tot integratie van wijzigingen door afzonderlijke ontwikkelaars is dan wel handig, en ook dit kan deltasql regelen.

Uitvoering

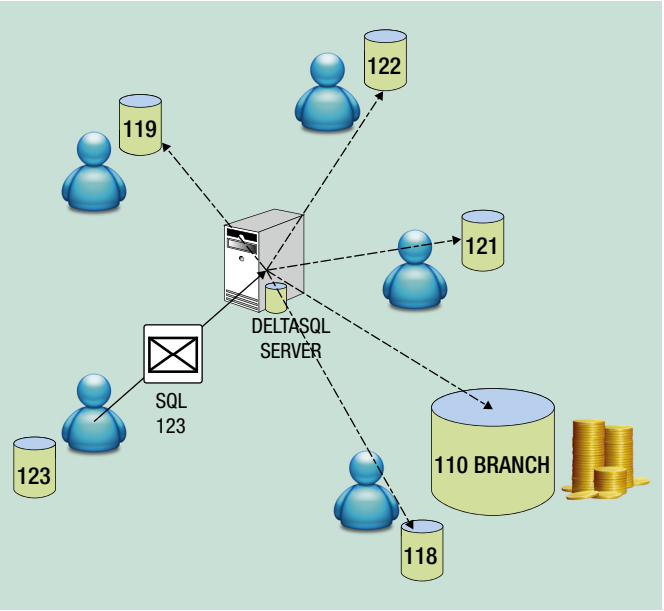
Feitelijk is deltasql een centrale opslagstructuur voor de stukjes developer script die de structuur van de database aanpassen. De developer hangt versienummers aan de structuur van de database, en vervolgens kan iedereen die dat nodig heeft op basis van de huidige en gewenste versienummers een script downloaden dat hun database opwaardeert. Dat wil zeggen, een script dat speciaal voor de persoonlijke situatie geldt, zodat altijd van een oudere versie naar een nieuwe kan worden overgegaan.
Het gevolg van deze flexibiliteit is dat veel gemakkelijker aanpassingen in het databasemodel gemaakt kunnen worden. In een normaal ontwikkelproject krijgt de database al snel een soort heilige status, juist vanwege de problematiek van het updaten van de versie. Dit gaat zo ver dat soms alleen op *major* nieuwe

versies van het databasemodel wordt veranderd. Maar met dit redelijk automatisch verlopende systeem voor upgrades, wordt het geen moeite om na een installatie te zeggen "zorg even dat de database in versie X is". Dit kan elke installatie dan afzonderlijk regelen.
De normale interactie met deltasql is over een webinterface, waar een script in SQL wordt afgeleverd op basis van de oorspronkelijke versie van de database en de gewenste nieuwe versie. Deze draait men dan handmatig op de database. Dit sluit redelijk aan bij mensen die zelf source code bouwen, en zich bewust zijn van de noodzaak van deze stap.

Er zijn echter ook genoeg gebruikers die een kant-en-klaar pakketje downloaden en installeren, en er zijn ook heel veel toepassingen waarvoor dat noodzakelijk is, omdat ze niet tot de core business van een onderneming behoren. Voor zulke gevallen is er ook automatische support in de vorm van *client modules*. Deze dienen toegang te hebben tot deltasql om updates te downloaden. Het is goed denkbaar dat een product waarvan de database onder deltasql valt, een server bereid houdt voor het upgraden van de database bij een klant. Sterker nog, het klinkt als een bijzonder slim plan om dit te doen voor commerciële roll-outs omdat het veel problemen en frustratie bij de gebruiker van een pakket kan helpen voorkomen.
De automatische downloads zijn met name ook bereikbaar vanuit shell scripts, wat ze prima geschikt maakt voor toepassing binnen installers zoals een .deb of .rpm of setup.exe; bovendien is deze variant voor het benaderen van deltasql heel geschikt voor 'continu' bouwen van de software, wat in de praktijk neerkomt op eens per nacht en dan als het even kan voor een veelheid van operating systemen. De op deze wijze verkregen input over bouwbaarheid op allerhande systemen helpt de ontwikkelaar aan waardevolle praktische informatie die lang niet altijd in generieke vorm beschikbaar is.
Vanzelfsprekend is het met deltasql mogelijk om gebruikers met verschillende toegangsrechten te definiëren. Voor bovenstaande suggestie van downloaders is er een gast-account, waarmee men kan downloaden zonder veranderingen te mogen aanbrengen. Niettegenstaande de commerciële potentie is deltasql een stuk software dat onder de GPL beschikbaar is, ofwel het is open source code. Zoals we wel meer zien in deze wereld wordt hier een essentieel probleem onderkend en opgelost en zoals ook vaker opvalt is het gevolg dat niet overal het wiel uitgevonden hoeft te worden. Zo bezien zou je open source zelfs een belangrijke aanjager van de wereldeconomie kunnen noemen!



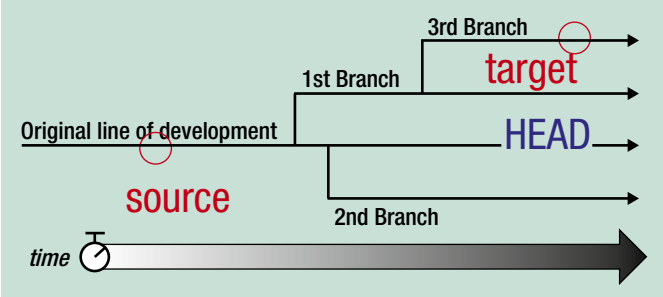
Afbeelding 1: Wanneer een ontwikkelaar aanpassingen in een datamodel doet dan dient dit met elke gebruiker uitgewisseld te worden. En dat roept onherroepelijk klantvragen op. (Bron: www.deltasql.org).



Afbeelding 2: Door deltasql te gebruiken, kan de ontwikkelaar zijn aanpassingen uitdragen via een centraal systeem, waardoor ieder op zijn eigen wijze de aanpassingen op het datamodel kan binnenhalen en doorvoeren. (Bron: www.deltasql.org).

Structuren

De interne organisatie van deltasql bestaat uit projecten met daarbinnen modules. Een project is meestal een stuk software voor intern of extern gebruik; een model is een database-model, en zal meestal slaan op de structuur van een bepaalde applicatiedatabase. Elke database bevat een tabel TBSYNCHRONIZE met daarin een attribuut VERSIONNR. Hieruit moet voor een upgrade de hoogste waarde worden opgehaald, en ingevoerd in deltasql als de versie van waaruit men naar een bepaalde, door de software bepaalde nieuwe versie dient te gaan. Uiteraard wordt de tabel TBSYNCHRONIZE aan het eind van een upgrade bijgewerkt.
Tenslotte is deltasql zich bewust van de praktijk van branching: het afsplitsen van de historie van bepaalde softwareversies, waarna die in afzondering worden onderhouden. Dit is gebruike-



Afbeelding 3: Versiebeheersystemen werken met branches, en deltasql kan daar (in beperkte mate) mee omgaan. (Bron: www.deltasql.org).

lijk bij releases, waarna een versiebeheersysteem wordt gebruikt om de software in die versie rotsvast te houden, terwijl de ontwikkelversie zich een weg baant over kronkelpaadjes langs kloven en afgronden. Waar nodig kunnen ontdekte fouten worden gerepareerd in de ontwikkelversie en 'gebackport' worden naar zo'n stabiele versie, zodat deze niet helemaal blijft stilstaan.
Wat deltasql doet ter ondersteuning hiervan is bewust maken van de mogelijkheid tot branchen, en welke van de aanpassingen van het databasemodel op welke branches van toepassing zijn. Bij het upgraden van de databasemodellen wordt hiertoe gekeken welke aanpassingen die bij de gewenste versie van een branch horen al aanwezig zijn, en de nodige aanpassingen worden daarmee in lijn doorgevoerd.

Beperkingen

Hoewel deltasql in al zijn conceptuele eenvoud een echte schepper van mogelijkheden is, zijn er toch ook wat beperkingen die het noemen waard zijn. Want de integratie met versiebeheersystemen laat op een aantal punten te wensen over. Let wel, door niet met een bepaald systeem te integreren is de toepasbaarheid breder, maar het iets te algemene karakter heeft ook enige nadelen. Allereerst wordt voor deltasql een afzonderlijke (versie) administratie gevoerd. Dat kan ook gevolgen hebben voor regelmatig terugkerende handelingen zoals packaging; het kan minder automatisch of eenvoudig worden om *tarballs* te draaien, bijvoorbeeld.

Belangrijker echter, is de inherente eigenschap van deltasql om alleen vooruit te bewegen. Waar een patch op een stuk source code doorgaans in twee richtingen (heen en terug) kan worden gebruikt, is dat met deltasql niet mogelijk.
De gevolgen hiervan zijn dat een ontwikkelaar niet soepel tussen branches kan schakelen. Waar een krachtig versiebeheersysteem bereid is om een set van niet-ingecheckte wijzigingen te behouden bij de overschakeling op een andere branch (door terug te gaan naar de nieuwste gemeenschappelijke versie en daarna langs de nieuwe branch vooruit te bewegen naar de gewenste versie) is dat met deltasql niet mogelijk.
De oorzaak is overigens heel praktisch: een ontwikkelaar checkt de wijzigingen op de database in, maar doorgaans niet de omgekeerde bewerking. Het ligt ook niet voor de hand dat te vragen omdat het de hoeveelheid werk verdubbelt zonder dat dit de release versnelt. De tijdsdruk die geregeld achter ontwikkeling zit maakt het dan ook niet waarschijnlijk dat een ontwikkelaar daadwerkelijk zulke wijzigingen zou doorvoeren.
Wat dan zou resten is een eventuele mogelijkheid om bewerkingen automatisch, dus met kennis van de betekenis van SQL, om te keren. Hierbij ontstaat dan wel de behoefte om te weten welke database is gebruikt, en precies de interpretatie van SQL voor die database toe te passen. Dat kan op zijn beurt weer beperkend werken.
Bovendien zijn niet alle wijzigingen omkeerbaar. Het zou best doenlijk zijn om de toevoeging van een attribuut terug te draaien, naar de verwijdering van dat attribuut. Of andersom, als een attribuut overbodig bleek te zijn dan zou de verwijdering daarvan tot een toevoeging kunnen leiden. Soms zijn er zelfs omzettingen zoals van coördinatenstelsels in geografische systemen die omkeerbaar zijn.

Maar er zijn ook gevallen waarin er keihard data verdwijnen. Zelfs als je de structuur omkeerbaar zou kunnen aanpassen, dan nog is het weghalen van data onomkeerbaar. En zeker bij subtiele bewerkingen zoals het overstappen naar een andere branch, waarbij je niet altijd bewust bent dat, of hoe ver de historie teruggerold wordt, zou het kunnen gebeuren dat een schokkende hoeveelheid data verdwijnt. Dit zou bijvoorbeeld noodzakelijk kunnen zijn als dezelfde bewerking op de oude en nieuwe branch zou zijn toegepast, maar als voor de branchwissel eerst teruggegaan wordt naar een gemeenschappelijke versie die stamt van voor deze gemeenschappelijke wijziging.
De werkwijze van deltasql is dus weliswaar enigszins beperkt in verhouding tot wat met softwareversies kan, maar dat is op de keper beschouwd eigenlijk wel een verstandige opzet.

Samenvatting

Met deltasql is het mogelijk om met meer ontwikkelaars en in een groter tempo veranderingen door te voeren in een datamodel. Zelfs klanten en eindgebruikers kunnen gebruik maken van automatische updates vanaf hun oude model naar een nieuw. Dit automatiseert een hoop problemen, en heeft als enige voorwaarde een server die wijzigingsscripts kan ophoesten.
Het is met deltasql niet mogelijk om terug te gaan in de tijd, en dat lijkt ook verstandig voor het behoud van data. Voor elke afzonderlijke database wordt daarom een branch gekozen, en daar wijkt hij wat deltasql betreft niet meer van af, en daarover beweegt hij slechts vooruit qua datamodel. Dit zijn in principe beperkingen, maar ze hoeven niet fataal te zijn want ze sluiten aan bij wat toch al gebruikelijk is.

Rick van Rein
Dr. ir. H. van Rein (rick@openfortress.nl) is ontwikkelaar en beheerder bij OpenFortress Digital signatures.