DE NORMAALVORMEN VAN SOA

Kunt u ze nog herinneren? De normaalvormen? Weet u nog de definitie van de derde normaalvorm? Waarschijnlijk wel. De meeste databaseontwerpers weten hoe een relationele database ontworpen moet worden. Ze weten hoe de normaalvormen toegepast moeten worden. Sommigen staan er niet eens bij stil, ze passen de normaalvormen automatisch toe. Ondanks dat ze misschien onbewust gebruikt worden, het feit dat ze bestaan, helpt ons absoluut bij het ontwerpen van tabellen met degelijke structuren.

Door Rick van der Lans

In feite was het geweldig dat toen we aan het begin van de jaren tachtig relationele databases begonnen in te zetten, we richtlijnen hadden voor het ontwerpen van onze tabellen. Het gaf ons houvast. Feitelijk waren de richtlijnen, ofwel de normaalvormen, er eerder dan de technologie. En zo hoort het eigenlijk ook. Wat heeft het voor zin om technologie in te zetten als we niet weten wat de ontwerpregels zijn?
Hoe anders is dit voor Service Oriented Architectures. Ook al heeft u zelf nog nooit een SOA ontworpen, dan weet u ongetwijfeld dat de primaire bouwsteen van een SOA de service is. Simpel gesteld; elke service bevat een bepaalde functionaliteit en heeft een interface. Deze interface heeft een XML-structuur. Bijvoorbeeld, de ingaande interface van een service is een klantnummer en de uitgaande interface een klantdossier. Waarschijnlijk zijn beide brokken informatie, het klantnummer en het klantdossier, gevormd als XML-document. De SOA zelf is een netwerk van services waarbij de services aangeroepen worden door applicaties en services.
De interfaces van alle services moeten ontworpen worden. Dit geldt zeker als die interfaces 'breed' zijn, zoals bijvoorbeeld een klantdossier dat uit honderden kleine gegevenselementen kan bestaan. Nu zou u denken, dan hanteren we de normaalvormen voor services en zijn we klaar. Maar daar raken we een gevoelig punt: die zijn er namelijk (nog) niet. Normaalvormen voor services zouden nuttig kunnen zijn omdat interfaces op veel alternatieve manieren te ontwerpen zijn. En de vraag die dan ontstaat luidt; wat is het beste alternatief? We beschrijven enkele aspecten die er toe leiden dat veel alternatieve oplossingen mogelijk zijn.
Ten eerste, als we een hoeveelheid gegevenselementen in een XML-structuur moeten gieten, dan zijn daar vaak veel alternatieve vormen voor, van zeer platte tot zeer diepe structuren, met structuren van meer dan tien niveaus diep. Nemen we bijvoorbeeld een service die een klantdossier teruggeeft, wat is dan de ideale structuur voor dit grote XML-document?
Ten tweede, een vraag die beantwoord moet worden is of de service per se alle gegevens moet teruggeven. Neem weer de service die een klantdossier geeft. Het gehele klantdossier zal zeker adresgegevens, contactpersonen en verkoopcijfers bevatten. Maar is de aanroepende applicatie geïnteresseerd in alle gegevens? Misschien is het voldoende om eerst de adresgegevens terug te geven en pas als de applicatie ze werkelijk nodig heeft, de verkoopcijfers te geven. In feite kiezen we dan voor een REST-achtige oplossing.
En ten derde, dient de interface afhankelijk te zijn van de consument, ofwel de applicatie of service die de betrokken service aanroept? Bijvoorbeeld, dienen services anders ontworpen te worden als ze door een mashup aangeroepen worden, dan wel door een BPEL-engine? Kortom, veel alternatieven en veel vragen. Momenteel hopen we maar dat we services ontwerpen die de tand des tijds kunnen doorstaan en door allerlei applicaties efficiënt en eenvoudig aangeroepen kunnen worden.
Willen we SOA-projecten succesvol afronden, dan is het cruciaal dat we richtlijnen voor het ontwerpen van service interfaces gaan ontwikkelen. Het zal organisaties helpen om sneller de correcte service interfaces te ontwerpen. Het wordt dus tijd dat we hiervoor richtlijnen gaan opstellen. Stelt u zich eens voor dat we de normaalvormen niet zouden hebben. Hoe zouden dan de meeste databasestructuren er uit zien?

Rick van der Lans is zelfstandig IT-consultant.