Een introductie tot rule-enriched gegevensmodellen

De Business Rules Approach

Marc Dierick

De laatste tijd is er heel wat te doen over Business Rules en de Business Rules Approach. Vele lezers van DB/M zullen denken: de zoveelste hype, en weer één die niet overeenstemt met ons relationele denken. De Business Rules Approach staat echter niet haaks op het relationele model.

Integendeel, het gegevensmodel vormt – samen met het procesmodel en het bedrijfsregelmodel – een belangrijk onderdeel bij het toepassen van de Business Rules Approach, en wel zo dat het gegevensmodel verrijkt wordt met een aantal bedrijfsregelspecifieke gegevenselementen. Dit om bedrijfsregels die betrekking hebben op gegevens en gegevensmutaties aanschouwelijker te maken en het afdwingen ervan te vereenvoudigen. Een dergelijk model wordt een *rule-enriched gegevensmodel* genoemd[1]. Doel van dit artikel is aan te geven wat en uit te leggen hoe men een relationeel gegevensmodel dient uit te breiden om tot een model te komen dat toepasbaar is binnen de rules-oriënted wereld.

Een voorbeeld

Het gaat hierbij om een variant van een al eerder in DB/M gepubliceerd voorbeeld[2] betreffende een organisatiestructuur, waarbij de organisatie bestaat uit afdelingen en diensten. Medewerkers kunnen tewerkgesteld worden op niveau van de organisatie, binnen afdelingen en binnen diensten, waarbij het mogelijk is tegelijkertijd op verschillende niveaus tewerkgesteld te worden. Een voorbeeld van een tewerkstelling op het niveau van de organisatie is die van de algemene directeur, maar ook diens secretaresse fungeert op dat niveau. Op niveau van afdeling hebben we de afdelingschef, maar ook daar diens secretaresse. Hetzelfde geldt voor een werknemer die toegewezen wordt aan een dienst, deze verricht prestaties op niveau van die dienst. Uiteindelijk betreft het hier het merendeel van de werknemers die werkzaam zijn binnen de organisatie.
Bekijk de tabellen in afbeelding 1. Uiteraard definiëren we een aantal relaties tussen deze tabellen en komen zo tot het gegevensmodel zoals in afbeelding 2 te zien is. Een medewerker mag een maximaal aantal uren per week werken. Dit aantal uren wordt afgesproken tussen de personeelsdirecteur en de medewerker en wordt opgeslagen in de tabel MEDEWERKER. We kunnen deze bedrijfsregel controleren door het uitvoeren van het volgende SQL-statement, dat het afgesproken maximaal aantal uren vergelijkt met het totaal aantal uren dat de medewerker is tewerkgesteld op niveau van de organisatie, afdelingen en diensten samen:

```
select * from MEDEWERKER MED
where sum( "AantalUren" ) > MED.MAX_AANT_UREN
from
(
        select sum( AANT_UREN ) "AantalUren"
        from MED_ORG_TEWERKSTELLING MOTWS
        where MOTWS.MED_ID = MED.ID
        union
        select sum( AANT_UREN ) "AantalUren"
        from MED_AFD_TEWERKSTELLING MATWS
        where MATWS.MED_ID = MED.ID
        union
        select sum( AANT_UREN ) "AantalUren"
        from MED_DNST_TEWERKSTELLING MDTWS
        where MDTWS.MED_ID = MED.ID
) // Dit statement retourneert medewerkers die te
                                          veel werken
```

Binnen een relationeel gegevensmodel worden bedrijfsregels afgedwongen door middel van beperkingregels, eventueel uitgedrukt in SQL. De zojuist gedefinieerd beperkingregel dient gecontroleerd te worden in die gevallen waarbij de kans bestaat dat hij overtreden wordt:
Bij het wijzigen van MEDEWERKER.MAX_AANTAL_UREN;
Bij het toevoegen van een MED_ORG_TEWERKSTELLING-tupel;
Bij het wijzigen van MED_ORG_TEWERKSTELLING.AANTAL_UREN;

Bij het wijzigen van MED_ORG_TEWERKSTELLING.MED_ID;
Bij het toevoegen van een MED_AFD_TEWERKSTELLING-tupel;
Bij het wijzigen van MED_AFD_TEWERKSTELLING.AANTAL_UREN;
Bij het wijzigen van MED_AFD_TEWERKSTELLING.MED_ID;
Bij het toevoegen van een MED_DNST_TEWERKSTELLING-tupel;
Bij het wijzigen van MED_DNST_TEWERKSTELLING.AANTAL_UREN;
Bij het wijzigen van MED_DNST_TEWERKSTELLING.MED_ID.

Al met al lijkt de creatie van zo'n monolithische beperkingregel behoorlijk eenvoudig, maar het onderhoud ervan is problematischer. Hoogstwaarschijnlijk, en helaas echter, is er een groot aantal van deze complexe beperkingregels terug te vinden binnen het merendeel van de administratieve informatiesystemen.

Problemen met deze wijze van modelleren

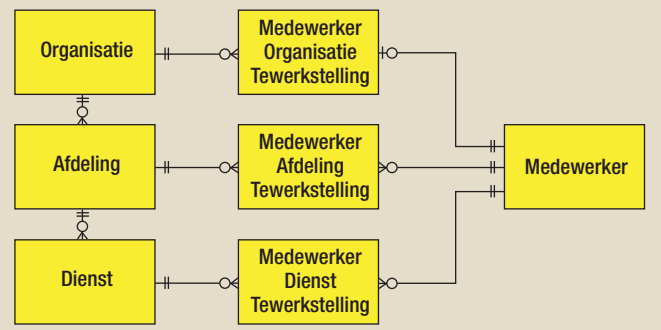
Bij een wijzigende bedrijfsorganisatie (bijvoorbeeld het toevoegen van een niveau holding of departement) is het dan moeilijk uit te maken welke beperkingregels gewijzigd dienen te worden en hoe. Veldwijk draagt een oplossing aan die dit probleem oplost. Abstracter modelleren is de boodschap. Terecht. Maar in dit artikel wil ik een ander probleem belichten, namelijk dat van de zogenaamde mismatch tussen ICT en de business.
Voor ICT-mensen die ietwat begaan zijn met het relationele model, is er geen probleem met het bouwen van een gegevensmodel, het definiëren van beperkingen in de vorm van SQL-statements en het aangeven van de gevallen waar deze beperkingen overtreden kunnen worden. En voor OO'ers zijn er al evenmin problemen indien men het model uit het voorbeeld omzet in een klassenmodel waarbij de beperkingregels dan afgedwongen worden in methodes die geïmplementeerd worden binnen deze klassen.
Maar de mensen uit de business – waaronder de eindgebruikers – blijven in de kou staan. Hen uitleggen dat het informatiesysteem de nodige bedrijfsregels daadwerkelijk afdwingt blijkt een hele klus indien men dit doet aan de hand van relationele modellen en objectgeoriënteerde modellen. Een stap in de goede richting is de beperkingregels uit te drukken in een voor eindgebruikers gangbare taal, zoals overigens voorgeschreven wordt in de Business Rules Approach. De beperkingregel uit het voorbeeld kunnen we immers opschrijven als bedrijfsregel:

Business Rules Aanpak

Heeft de Business Rules Aanpak gevolgen voor het implementeren van relationele databases? Bij de Business Rules Aanpak vormen bedrijfsregels een hoofdelement bij het bouwen van software. Een bedrijfsregel is een atomair stukje herbruikbare bedrijfslogica dat op declaratieve wijze gespecificeerd wordt. Bij administratieve informatiesystemen die gebruik maken van relationele databases is het eigenlijk niet anders. Dat een uitgaande factuur opgemaakt wordt voor een klant en dat we voor een klant meerdere uitgaande facturen kunnen hebben zijn twee uitspraken die we in een database declaratief kunnen opnemen, namelijk met behulp van een verwijzende sleutel die vastgelegd wordt in de catalogue. Dezelfde code in de database engine wordt uitgevoerd bij het afdwingen van om het even welke verwijzende sleutel. Herbruikbaarheid ten top! Jammer genoeg voorzien de meeste relationele databases niet in de mogelijkheid complexere bedrijfsregels dan verwijzende sleutels te implementeren op declaratieve wijze. Al snel komt men in de sfeer van SQL, triggers en stored procedures. En dat is procedurele applicatiecode, met als gevolg dat bedrijfsregels verborgen raken in die applicatiecode.
Met de Business Rules Aanpak wordt getracht hier iets aan te doen door gelijkaardige bedrijfsregels te classificeren. Dit leidt tot patronen die opgenomen kunnen worden in een uitgebreide relationele catalogue. Een uitgebreide database engine, in de vorm van generieke applicatiecode, zorgt ervoor dat de gegevens consistent blijven met de regels. Eén van deze patronen, het sommeringpatroon, wordt beschreven in dit artikel.

Tabelnaam en Kolommen (primaire sleutel onderlijnd)	Korte naam	Beschrijving
ORGANISATIE (ID, NAAM)	ORG	Een organisatie heeft een id en een naam. Aangezien de organisatie de scope van het systeem bepaalt mag deze tabel slechts één tupel bevatten.
AFDELING (ID, NAAM, ORG_ID)	AFD	Een afdeling heeft een id en een naam, en maakt deel uit van de organisatie.
DIENST (ID, NAAM, AFD_ID)	DNST	Een dienst heeft een id en een naam, en maakt deel uit van een afdeling.
MEDEWERKER (ID, NAAM, VOORNAAM, SALARIS, MAX_AANT_UREN)	MED	Een medewerker heeft een id, een naam, een voornaam en een salaris, en hij/zij mag een maximaal aantal uren werken per week.
MED_ORG_TEWERKSTELLING (MED_ID, ORG_ID, AANT_UREN)	MOTWS	Elk tupel representeert de tewerkstelling van een medewerker op niveau van de organisatie waaraan hij/zij een aantal uren per week toegewezen wordt.
MED_AFD_TEWERKSTELLING (MED_ID, AFD_ID, AANT_UREN)	MATWS	Elk tupel representeert de tewerkstelling van een medewerker op niveau van een afdeling waaraan hij/zij een aantal uren per week toegewezen wordt.
MED_DNST_TEWERKSTELLING (MED_ID, DNST_ID, AANT_UREN)	MDTWS	Elk tupel representeert de tewerkstelling van een medewerker op niveau van een dienst waaraan hij/zij een aantal uren per week toegewezen wordt.

Afbeelding 1.



Afbeelding 2: Gegevensmodel.

Het aantal uren dat een medewerker tewerkgesteld wordt mag een tussen de personeelsdirecteur en die medewerker afgesproken maximum niet overschrijden.

De eerder vermelde problemen bij wijzigende organisatiestructuur blijven evenwel bestaan. Het is een heel karwei uit te zoeken welke bedrijfsregels aangepast dienen te worden en de aanpassingen aan de tekstuele beschrijvingen van deze bedrijfsregels zijn ook niet te onderschatten.

En hoe kunnen we uitleggen aan de eindgebruikers dat de bedrijfsregels door het systeem worden afgedwongen? Niet dus! Men stelt zich over het algemeen tevreden met het opstellen van *use cases*. Bij elke use case tracht men dan aan te geven welke de bedrijfsregels zijn die overtreden kunnen worden, en hoe het systeem hierop dient te reageren. Hierop kan men dan test cases afstemmen. Indien deze succesvol worden doorlopen gaat men ervan uit dat het systeem correct geïmplementeerd is.

Kan het dan anders?

Complexe monolithische beperkingregels dienen eerst genormaliseerd te worden om zo enkel *atomaire* regels over te houden. Het komt erop neer dat een complexe beperkingregel wordt opgesplitst in een aantal samenhangende kleinere en beter herbruikbare regels.

Dit wordt gerealiseerd door het relationele gegevensmodel uit te breiden met een aantal tabellen en/of kolommen die geen basisinformatie bevatten, maar wel daarvan afgeleide kennisinformatie. Het opbouwen van deze kennisinformatie gebeurt aan de hand van kennisregels die als een speciaal geval van bedrijfsregels gezien kunnen worden.

We verrijken het gegevensmodel met zaken die we weten over werknemers, met andere woorden met kennis over werknemers. Deze kennis geeft aanleiding tot bijkomende kolommen in tabel MEDEWERKER, die in afbeelding 3 weergegeven worden met vierkante haakjes.

Let wel, deze kolommen dienen *niet persistent* opgenomen te worden in de database. Het niet opnemen van deze kolommen in de database heeft wel tot gevolg dat de waarden berekend dienen te worden bij elke keer dat men een MEDEWERKER-tupel benadert, wat aanleiding kan geven tot performanceproblemen. Het wel opnemen van kennisinformatie in de database houdt dan

Tabelnaam en Kolommen (primaire sleutel onderlijnd)	Korte naam	Beschrijving
MEDEWERKER (ID, NAAM, VOORNAAM, SALARIS, MAX_AANT_UREN, [AANT_UREN_ORG], [AANT_UREN_AFD], [AANT_UREN_DNST], [TOTAAL_AANT_UREN])	MED	Een medewerker heeft een id, een naam, een voornaam en een salaris, en hij/zij mag een maximaal aantal uren werken per week. Hij/zij werkt een aantal uren per week op niveau van de organisatie. Hij/zij werkt een aantal uren per week op niveau van afdelingen. Hij/zij werkt een aantal uren per week op niveau van diensten. Hij/zij werkt een totaal aantal uren, op om het even welk niveau.

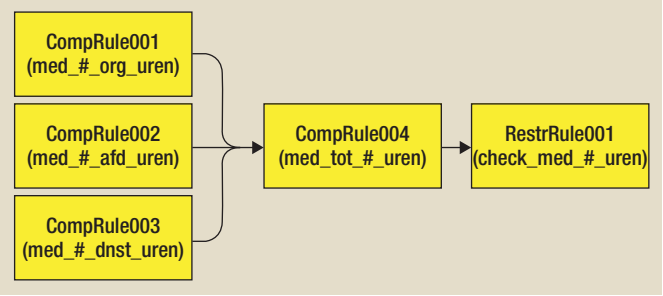
Afbeelding 3.

Kennisregels	
CompRule001	Gegeven een MEDEWERKER-tupel, dan is AANT_UREN_ORG = de som van AANT_UREN van alle MED_ORG_TEWERKSTELLING-tupels horende bij dat MEDEWERKER-tupel.
CompRule002	Gegeven een MEDEWERKER-tupel, dan is AANT_UREN_AFD = de som van AANT_UREN van alle MED_AFD_TEWERKSTELLING-tupels horende bij dat MEDEWERKER-tupel.
CompRule003	Gegeven een MEDEWERKER-tupel, dan is AANT_UREN_DNST = de som van AANT_UREN van alle MED_DNST_TEWERKSTELLING-tupels horende bij dat MEDEWERKER-tupel.
CompRule004	Gegeven een MEDEWERKER-tupel, dan is TOTAAL_AANT_UREN = AANT_UREN_ORG + AANT_UREN_AFD + AANT_UREN_DNST

Afbeelding 4.

Beperkingregels	
Restr001	Gegeven een MEDEWERKER-tupel, indien TOTAAL_AANT_UREN > MAX_AANT_UREN dan geef melding “Het maximum aantal uren voor de medewerker is overschreden”.

Afbeelding 5.

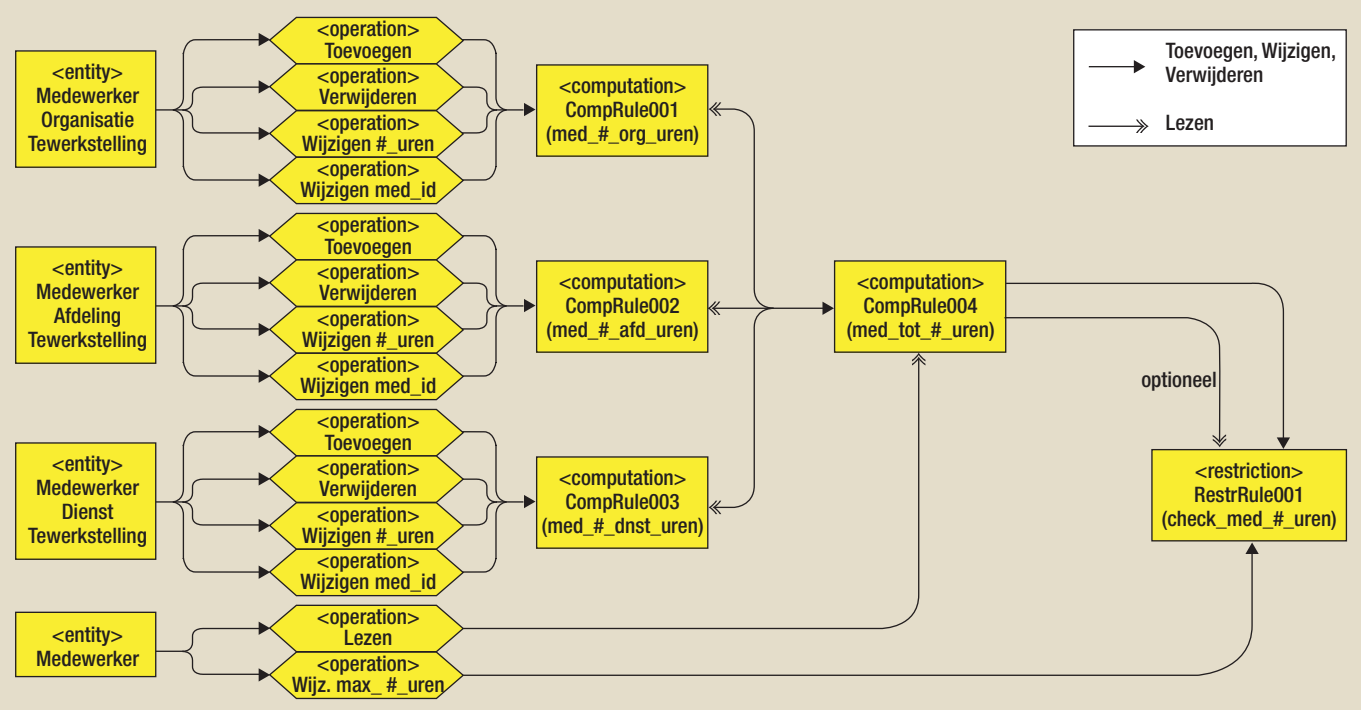


Afbeelding 6: Rules dependency diagram.

weer in dat men ervoor dient te zorgen dat de basisinformatie en de kennisinformatie te allen tijde synchroon blijven.
Op basis van deze toegevoegde kolommen definiëren we nu een aantal regels die toelaten de monolithische in SQL uitgedrukte beperkingregel uit het voorbeeld op te splitsen in kleinere samenhangende regels. We beginnen met de kennisregels, zie afbeelding 4. Het afdwingen van deze kennisregels bestaat uit een stukje proces uitvoeren dat ervoor zorgt dat de regel niet wordt overtreden. Dit stukje proces kan bijvoorbeeld geïmplementeerd worden aan de hand van triggers en stored procedures. Het uitvoeren ervan wordt verder in dit artikel beschouwd als het uitvoeren van de regel. Wat betreft de regels CompRule001, CompRule002 en CompRule003 kunnen we stellen dat ze erg op elkaar lijken, met andere woorden dat ze voldoen aan hetzelfde patroon. In dit geval gaat het om een sommeringpatroon.
En dan de beperkingregel waar alles om draait, zie afbeelding 5. Het afdwingen van deze beperkingregel bestaat erin een foutboodschap te geven en de transactie terug te draaien.
Nadat de regels gedefinieerd zijn, worden de onderlinge afhankelijkheden bepaald. Zo zal beperkingregel RestrRule001 pas gecontroleerd kunnen worden nadat kennisregel CompRule004 is uitgevoerd. En kennisregel CompRule004 wordt pas uitgevoerd nadat de kennisregels CompRule001, CompRule002 of CompRule003 zijn uitgevoerd. Zo komen we tot een *rules dependency diagram*, zie afbeelding 6.
CompRule001, CompRule002 en CompRule003 zijn niet afhankelijk van elkaar. Wijzigingen aan één van deze regels zullen geen gevolgen hebben voor de andere. Wijzigende kennis- en bedrijfsregels vergen minder onderhoud en zijn beter herbruikbaar indien ze genormaliseerd en dus atomair zijn. Uiteraard dienen we nu nog aan te geven wanneer welke beperkingregels overtreden kunnen worden en wanneer welke kennisregels uitgevoerd dienen te worden. Per regel geven we aan wanneer hij rechtstreeks overtreden of uitgevoerd kan worden door toedoen van een gebruiker. We doen dit aan de hand van de CRUD-operaties die rechtstreeks door gebruikers doorgevoerd kunnen worden, zie afbeelding 7. Zo komen we tot een *rule invocation diagram*, dat een totaalbeeld geeft wanneer welke regels uitgevoerd en/of gecontroleerd dienen te worden, zie afbeelding 8. We kunnen in dit diagram bijvoorbeeld zien dat bij het toevoegen van een MED_AFD_TEWERKSTELLING-tupel kennisregel CompRule002 uitgevoerd wordt. Daarna wordt kennisregel CompRule004 uitgevoerd. Bij het uitvoeren van kennisregel CompRule004 wordt echter een MEDEWERKER-tupel gelezen. Dit geeft aanleiding tot het uitvoeren van CompRule001 en CompRule003. Pas dan wordt beperkingregel RestrRule001 gecontroleerd.
De uitleg aan een eindgebruiker wordt er eenvoudiger op. Het toevoegen van een MED_AFD_TEWERKSTELLING-tupel komt er immers op neer dat de desbetreffende medewerker vanaf dan een aantal uren werkt voor de desbetreffende afdeling. Tegelijkertijd verhoogt het totaal aantal uren dat die medewerker werkt op niveau van afdelingen (kennis die afgeleid wordt met behulp van CompRule002). En het totaal aantal uren dat de medewerker tewerkgesteld wordt verhoogt eveneens (kennis die afgeleid wordt met behulp van CompRule004). Uiteindelijk wordt gecontroleerd of het maximaal aantal uren dat de medewerker mag werken niet overschreden wordt (beperkingregel RestrRule001).

Regel	Controle/Uitvoering
CompRule001 Berekenen van het # uren dat de medewerker tewerkgesteld is op niveau van de organisatie.	Bij het toevoegen van een MED_ORG_TEWERKSTELLING-tupel
	Bij het wijzigen van MED_ORG_TEWERKSTELLING.AANT_UREN
	Bij het wijzigen van MED_ORG_TEWERKSTELLING.MED_ID
	Bij het verwijderen van een MED_ORG_TEWERKSTELLING-tupel
CompRule002 Berekenen van het # uren dat de medewerker tewerkgesteld is op niveau van afdelingen.	Bij het toevoegen van een MED_AFD_TEWERKSTELLING-tupel
	Bij het wijzigen van MED_AFD_TEWERKSTELLING.AANT_UREN
	Bij het wijzigen van MED_AFD_TEWERKSTELLING.MED_ID
	Bij het verwijderen van een MED_AFD_TEWERKSTELLING-tupel
CompRule003 Berekenen van het # uren dat de medewerker tewerkgesteld is op niveau van diensten.	Bij het toevoegen van een MED_DNST_TEWERKSTELLING-tupel
	Bij het wijzigen van MED_DNST_TEWERKSTELLING.AANT_UREN
	Bij het wijzigen van MED_DNST_TEWERKSTELLING.MED_ID
	Bij het verwijderen van een MED_DNST_TEWERKSTELLING-tupel
CompRule004 Berekenen van het totaal aantal uren dat de werknemer is tewerkgesteld.	Bij het lezen van een MEDEWERKER-tupel
RestrRule001 Controleren of het maximaal aantal uren van tewerkstelling niet overschreden wordt.	Bij het wijzigen van MEDEWERKER.MAX_AANT_UREN

Afbeelding 7.



Afbeelding 8: Rule invocation diagram.

Beperkingregels	
Restr002	Gegeven een MEDEWERKER-tupel, indien AANT_UREN_ORG > 0 en (AANT_UREN_AFD > 0 of AANT_UREN_DNST >0) dan geef melding "De medewerker die tewerkgesteld is op niveau van de organisatie mag niet tewerkgesteld worden op niveau van afdelingen of diensten".

Afbeelding 9.

Bij het lezen van een MEDEWERKER-tupel wordt eerst getracht kennisregel CompRule004 uit te voeren. Om deze kennisregel uit te kunnen voeren dienen echter eerst de kennisregels CompRule001, CompRule002 en CompRule003 uitgevoerd te worden. Pas dan wordt beperkingregel RestrRule001 uitgevoerd, indien gewenst (bijvoorbeeld bij het toevoegen van een nieuwe regel of het wijzigen van een bestaande regel kan het handig zijn bestaande gegevens tegen de gewijzigde kennis- en beperkingregels aan te houden).

Het hergebruik van regels wordt in de hand gewerkt door gebruik te maken van atomaire regels. Stel dat iemand die tewerkgesteld is op niveau van de organisatie niet tewerkgesteld mag worden op niveau van een afdeling of dienst. Dan krijgen we een bijkomende beperkingregel RestrRule002, zie afbeelding 9. Hierbij worden de kennisregels CompRule001, CompRule002 en CompRule003 integraal hergebruikt wat leidt tot minder en makkelijker onderhoud. In SQL zou dit aanleiding geven tot een complex statement in de aard van het voorbeeld bij het begin van dit artikel, waarbij gedeeltes van beide SQL-statements sterk gelijken op elkaar, maar toch tweemaal neergeschreven dienen te worden.

Gevolgen voor software-ontwikkeling

Als we tijdens de implementatie van een administratief informatiesysteem gebruik maken van use cases, dan volstaat het hierin te verwijzen naar de tabellen en CRUD-operaties die van toepassing zijn bij het uitvoeren van elke use case. Welke regels dan uitgevoerd en/of gecontroleerd dienen te worden volgt uit het rule dependency diagram en het rule invocation diagram. Een OO-implementatie of implementatie binnen een RDBMS met behulp van triggers en stored procedures op basis van deze diagrammen is redelijk eenvoudig te realiseren. Als men ook nog gebruik maakt van patronen zoals het sommeringpatroon, kan men ervoor zorgen dat één stuk applicatiecode hergebruikt wordt bij het afdwingen van verschillende op elkaar gelijkende bedrijfsregels. En test cases bedenken wordt verheven tot een quasi-wiskundige bezigheid, wat de kwaliteit van informatiesystemen ten goede komt.

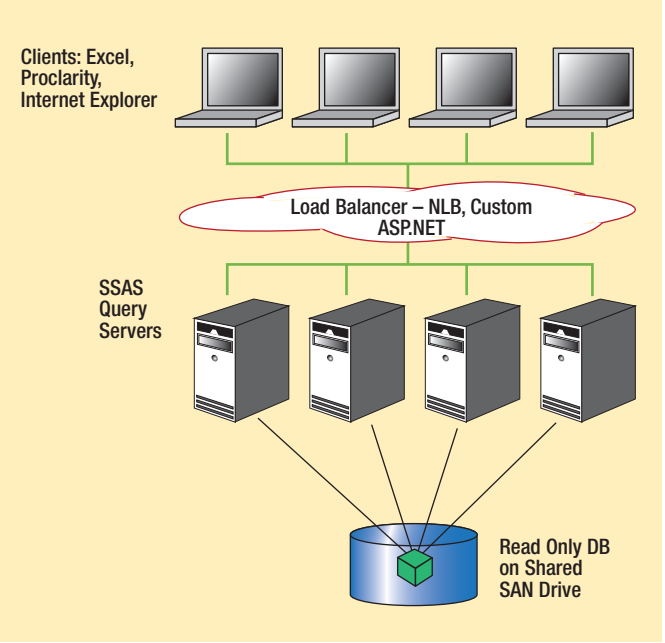
Nog eenvoudiger wordt het als men gebruik maakt van een *data-centric rules engine* zoals beschreven door C.J. Date.[3] Zo'n engine detecteert op basis van CRUD-operaties de noodzakelijke regels die uitgevoerd en/of gecontroleerd moeten worden, en dit zonder de volgorde van regels aan te hoeven geven, wat de rule dependency en rule invocation diagrams overbodig maakt. Een rules engine wordt immers in staat geacht deze volgorde zelf te bepalen. Bovendien laat zo'n rules engine toe bedrijfsregels declaratief te definiëren op basis van patronen zoals het sommeringpatroon.

Vervolg op pagina 41

mate van beschikbaarheid te bieden die we in het kort zullen benoemen (in een vervolgartikel gaan we uitvoerig in op de Always On Technologie en de praktische implementatie daarvan). De in SQL Server 2005 ondersteunde Database Mirroring is in SQL Server 2008 uitgebreid met compressie en men hoeft een database niet langer te herstarten ingeval van een handmatige failover. Ook de noodzaak komt te vervallen om een drive letter voor elke SQL Server instance te moeten toewijzen, wat het aantal mogelijke instances bij een geclusterde oplossing verhoogt. SQL Server 2008 introduceert verbetering aan de peer-to-peer replicatie, door nodes te kunnen toevoegen terwijl het replicatieproces online is. Door SQL Server 2008 te combineren met Windows Server 2008 wordt een solide HA-oplossing voor een totale SQL Server instance geboden.

Epiloog

Met de uiteindelijke release van SQL Server 2008 denkt Microsoft een database te kunnen bieden die geschikt is voor het datawarehouse, transaction processing en business-critical applicaties in een high-end data center. Op het gebied van beveiliging, beschikbaarheid, consolidatie en prestaties zijn daartoe de nodige aanpassingen en verbeteringen aangebracht. Naast deze, voor een datacenter database, belangrijke eisen, zijn er ook op het gebied van programmering, beheer, datatypen, analyse, datapresentatie en query allerlei nieuwe features geïmplementeerd. Het door Microsoft genoemde 'Data Platform' komt daarmee aan de vraag tegemoet om de toenemende verscheidenheid aan datatypen in een relationele database te kunnen opslaan, waaronder XML, e-mail, tijd/datum, geospatial, enzovoort. Met de komst van SQL Server 2008 denkt Microsoft een database te kunnen bieden die alle eigenschappen heeft die een onderneming vraagt voor het bieden van een continue toegang tot data, waarbij beveiliging, betrouwbaarheid en schaalbaarheid voor mission critical applicaties hoog in het vaandel staan.



Afbeelding 5: Schaalbare, gedeelde database voor AS (bron Microsoft).

SQL Server 2008 RTM is te downloaden vanaf: www.microsoft.com/sqlserver/2008/en/us/trial-software.aspx

Bram Dons is onafhankelijk IT consultant.

Thema Business Rules

Vervolg van pagina 31

Beter voldoen

Bedrijfsregels in de vorm van complexe volzinnen worden omgezet in kleinere samenhangende kennisregels en beperkingregels. Elk van deze regels kan makkelijk uitgelegd worden aan de eindgebruikers. Zo ook de samenhang tussen deze regels. En op welke informatie de regels inspelen en wanneer. Er wordt dus voldaan aan drie belangrijke aspecten die de Business Rules Approach oplegt, namelijk het begrijpelijk maken van de bedrijfsvoering voor de eindgebruiker, de traceerbaarheid van waar en wanneer bedrijfsregels uitgevoerd en/of gecontroleerd worden, en de scheiding van gegevens, processen en bedrijfsregels. Bovendien zijn deze kleine samenhangende regels beter onderhoudbaar dan grote complexe volzinnen, en komt men gemakkelijker tot hergebruik van regels.

Als we de aanhangers mogen geloven, zal deze aanpak zorgen voor systemen die beter voldoen aan de noden van de eindgebruikers. En het creëren van applicaties zou minder tijd in beslag nemen dan nu het geval is. Om maar te zwijgen over het onderhoud aan systemen. Daar zou een winst van meer dan 50 procent geen utopie zijn. En dat is geen overbodige luxe bij de huidige dynamiek die bedrijven en organisaties nastreven bij hun bedrijfsvoering.

Noten

1. *Barbara von Halle, Business Rules Applied, Building Better Systems Using the Business Rules Approach.*
2. *R.J. Veldwijk, 10 Geboden voor Goed Database-Ontwerp.*
3. *C.J. Date, What Not How: The Business Rules Approach to Application Development.*

Marc Dierick (marc@mardinfo.be) is onafhankelijk IT-consultant bij Mardinfo.