Set-level denken vergt een wat andere manier van denken

Data subtaal: geen alomvattende taal

Frido van Orden

Zoals vorige keer al besproken vormen de regels 4 tot en met 12 eerder het voorportaal van de relationele tempel, waar de regels 0 tot en met 3 met recht het 'heilige der heiligen' zouden kunnen worden genoemd. Reden om zo nu en dan in een aflevering twee regels tegelijk te bespreken, in deze aflevering regel 5 en 7.

Wees niet bang, we zijn regel 6 niet vergeten. De eis over view-mutaties uit regel 6 hangt echter nauw samen met de concepten van fysieke en logische gegevensonafhankelijkheid uit de regels 8 en 9, vandaar dat we deze de volgende keer samenhangend zullen behandelen. Deze keer gaat het over de taal waarin de database wordt aangesproken en het set-level karakter van het relationele model. Twee ogenschijnlijk 'open deuren', die in de praktijk echter nogal eens dicht blijken te zitten.

Data subtaal

De vijfde regel, die over de data subtaal handelt, is vrij uitgebreid geformuleerd. Bij het lezen van de naamgeving valt echter al gelijk 1 ding op: het gebruik van het woord 'subtaal'. 'Subtaal' suggereert dat er ook zoiets is als een 'supertaal' of 'hoofdtaal', en dat is dan ook precies wat Codd beoogde. Het relationele model is een (meta-)gegevensmodel, het is een alomvattende theorie over gegevens en de manipulatie daarvan. Het zegt niets over processen, hardware of zelfs maar fysieke bestandsindeling, terwijl er geen enkel werkend informatiesysteem gebouwd kan worden zonder rekening te houden met de zaken. Codd is echter de schoenmaker die zich bij zijn leest houdt: hij beoogt geen alomvattende (programmeer)taal te specificeren, maar geeft wel precies aan wat de taal die het relationele DBMS ondersteunt voor mogelijkheden zou mogen bieden.
Die mogelijkheden vallen uiteen in vijf eenvoudige categorieën: het definiëren van gegevensstructuren (het gegevensmodel); het raadplegen en muteren van gegevens uit het gegevensmodel; het beschermen van die gegevens door definitie van beveiliging- en integriteitregels en tot slot het waarborgen van een goed transactiebeheer, volgens de bekende ACID principes (Atomicity, Concurrency, Integrity en Durability). Niets meer en niets minder. Voor alle andere zaken kan de programmeur terecht in een andere programmeertaal, die in Codd's tijd in veel gevallen Cobol, Algol, Fortran of C zal hebben geheten.

De moderne systeemontwikkelaar, die zijn hand niet omdraait voor applicaties geschreven in een combinatie van Visual Basic, Java, SQL, XML en Unix-scripttaal zal het bestaan van een specifieke data subtaal de normaalste zaak van de wereld vinden. In Codd's tijd dacht men daar echter heel anders over. De wereld van administratieve applicaties, de beoogde doelgroep van het relationele model, werd beheerst door Cobol, COmmon Business Oriented Language. De beoogde doelstelling van Cobol was diametraal anders: het bieden van een gezamenlijke, alomvattende taal voor ontwikkeling van administratieve applicaties. Wie overweg kon met Cobol kon (en kan nog steeds) complete applicaties schrijven, van het bewandelen van fysieke index-structuren tot het presenteren van de gebruikers-interface.

Relational Rules (7)

De vorig jaar overleden dr. E.F. Codd werd wereldberoemd met zijn serie publicaties over een gegevens(meta)model dat later bekend zou worden onder de naam 'Relationeel Model'. De eerste uit die serie publicaties was een intern IBM Research Report dat uitkwam in 1969. 2004 zal dus gelden als een lustrum – 35 jaar Relationeel Model.
Frido van Orden schrijft in Database Magazine een serie artikelen over de betekenis en de erfenis van het gedachtengoed van Codd. In deze zevende aflevering worden regel 5 en 7 gecombineerd besproken.
Regel 5: De veelomvattende data subtaal-regel.
Het systeem dient minstens 1 relationele taal te ondersteunen die:

a. een lineaire syntaxis heeft;
b. zowel interactief gebruikt kan worden als binnen applicatieprogramma's;
c. ondersteuning biedt voor datadefinitie-operatoren (inclusief view-definities), data-manipulatie-operaties (mutaties en raadplegingen), beveiliging- en integriteitregels en transactiebeheer-operaties (begin, commit en rollback).

Regel 7: Set-level insert, update, en delete.
Het systeem dient set-level INSERT, UPDATE, en DELETE operatoren te ondersteunen.

Hoe anders is dit vergeleken bij de situatie in vele project-teams van vandaag, waar Microsoft-specialisten ruziën met Java-adepten over de gebruikte interface-XML en de DBA zich beklaagt over de niet-performante SQL die wordt uitgespuugd, terwijl de Unix-script goeroe met briljante cron-scripts de boel in de lucht mag zien te houden. Het idee van de schoenmaker die zich bij zijn leest houdt is goed, maar met een schoenmaker en een kleermaker moet u toch een eind kunnen komen. Met een compleet team van eigenzinnige kleding en image-adviseurs bereikt u op z'n best een suboptimale productiviteit.

Tot zover Codd's eisen aan de mogelijkheden van de data subtaal. De vijfde regel stelt echter nog twee andere eisen. De eis van 'lineaire syntaxis' schenk ik u, ik vind het zelf geen bijster interessante eis en zoekactie op Internet levert weinig meer op dan letterlijke citaten van Codd's regel. Een oproep aan de linguïsten onder de DB/M lezers derhalve om deze lacune in te vullen.

> Het benaderen van gegevens stond gelijk aan het schrijven van een programma

Wel interessant is de eis dat de data subtaal zowel interactief als binnen applicatieprogramma's moet kunnen worden gebruikt. Ongetwijfeld vindt u het de normaalste zaak van de wereld dat u SQL statements niet alleen in uw applicatieprogramma kunt opnemen maar ook rechtstreeks aan de database kunt aanbieden. Als u consequent bent vindt u het dan belachelijk dat u dit niet kunt met query's geschreven in bijvoorbeeld EJB-QL, de objectgeoriënteerde query-taal die wordt gebruikt in de Enterprise JavaBeans-architectuur? Helemaal schandelijk is het dan natuurlijk dat data-manipulatie (toevoeging, wijziging en verwijdering) in die omgeving alleen maar kan door middel van het aantrappen van een Java-methode, die dan nog niet eens te gebruiken is vanuit een eenvoudig Java-testprogramma maar alleen via een speciaal stuk middleware, de EJB-container!
In de ontstaanstijd van het relationele model was het al niet veel beter: het benaderen van gegevens stond gelijk aan het schrijven van een programma. Dat was in de begintijd van de automatisering onvermijdelijk, het onderscheid tussen het werken met gegevens en datgene wat we tegenwoordig 'business logica' noemen was simpelweg nog niet bedacht. Later werd het niet onderscheiden zelfs tot credo verheven (zie Cobol!). Codd was de eerste die het belang inzag van het wel maken van het onderscheid. In de eerste paar jaar dat hij over het relationele model schreef, had hij het meermalen over de doelstelling dat het relationele model en relationele technologie eenvoudig genoeg zou moeten zijn voor eindgebruikers.

Belachelijk denk u misschien, met de verwende GUI-gebruikers van vandaag in het achterhoofd, die gruwen bij een degelijk 80x25 'green screen'. Met 'eindgebruikers' werden in die tijd echter geen kantoormedewerkers bedoeld, maar applicatie-ontwikkelaars en automatiseerders die rapportages moesten produceren. In de donkere pre-relationele tijd waren dergelijke lieden overgeleverd aan de nukken van hun collega's die het voor het zeggen hadden over de database, die toen overigens nog niet zo werd genoemd. Het gerealiseerd krijgen van een simpele query vergde een inspanning vergelijkbaar met het krijgen van een pers-accreditatie voor filmen in Noord-Korea.

Hoe spijtig is het dan ook te moeten constateren dat het tegenwoordig vaak niet veel anders is, bang als DBA's zijn voor het openstellen van hun kaartenbakken (beveiliging en integriteit in de applicatie) voor vreemde SQL statements, die de boel ook nog eens platleggen omdat vrijwel geen DBMS standaard functionaliteit heeft om query's die niet performen of veel te veel rijen ophalen, op basis van simpele regels automatisch af te breken. Nou ja, u hoeft tenminste geen Cobol-programma meer te schrijven.

Set level insert, update en delete

Weinig aspecten van het relationele model worden in de dagelijkse praktijk zo weinig op waarde geschat als het intrinsieke set-level karakter van het relationele model. Het kernbegrip van het relationele model, de relatie, is gedefinieerd als een verzameling van tupels, die weer bestaan uit waarden (ik gebruik verder de engelse term 'set' in plaats van het eigenlijk juistere Nederlandse 'verzameling'). Alle relationele operatoren zijn gedefinieerd als operatoren op relaties die ook weer relaties als resultaat opleveren. Het is zelfs niet mogelijk om een enkel tupel (laat staan een losse waarde) op te vragen, er wordt altijd gevraagd om een relatie, die dan uit een enkel tupel bestaat, eventueel slechts een enkele waarde bevattend. De vertaalslag van relatie naar tupel en vandaar naar waarden vindt plaats buiten de wereld van het relationele model! Dat scheelt meteen weer een hoop zaken die buiten de data subtaal gehouden kunnen worden. Laat u niet van de wijs brengen door SQL-constructies als

```
SELECT v, (SELECT x FROM y WHERE y.b = a.b)
FROM a
```

waarbij het resultaat van een genest SELECT statement (wat een relatie zou moeten zijn) ineens als waarde in de SELECT clausule wordt gebruikt, daarbij impliciet een relatie, bestaande uit 1 tupel dat 1 waarde bevat, converterend naar die enkele waarde.

In de pre-relationele wereld is de gedachte precies omgekeerd. Het 'model', voor zover dat bestaat, gaat uit van records die via pointers aan andere records zijn gerelateerd. Indien de vraag die wordt gesteld, of de manipulatie die moet worden uitgevoerd, over meerdere records gaat, worden de record-level operaties

opgenomen in lusstructuren, waardoor uiteindelijk hetzelfde resultaat wordt bereikt.
De voordelen van de relationele set-level selectie boven de non-relationele lusstructuren en 'pointer chasing', zijn inmiddels tot vrijwel iedereen doorgedrongen. Zelfs de objectgeoriënteerden, anders toch trouwe volgelingen van de pre-relationele record-level filosofie, gebruiken hun eigen set-level query-talen, waarbij 'simpelheid' het helaas echter meestal wint van expressiekracht.
Bij mutatie van gegevens is de situatie echter helaas anders. Velen die de SELECT vrezen, met al die moeilijke joins en zo, slaken een een zucht van verlichting als er gegevens gewijzigd moeten worden. Er zijn drie eenvoudige statements:

```
INSERT INTO tabel(kolom1, kolom2, kolom3)
VALUES(waarde1, waarde2, waarde3);

UPDATE tabel SET kolom=waarde WHERE CURRENT OF
                                          cursor;

DELETE FROM tabel WHERE CURRENT OF cursor;
```

De 'where current of cursor' wordt dan nog wel eens vervangen door de selectie op primaire sleutelwaarden maar moeilijker dan dat wordt het meestal niet. Wordt er manipulatie van meerdere records tegelijk gevraagd, dan noemen we het 'business logica' en mogen we het in Java schrijven, moet het op de 'middle tier' draaien voor maximale schaalbaarheid of verzinnen we een ander excuus om toch maar vooral geen extra, moeilijke SQL te hoeven schrijven.

Iedereen kan u vertellen dat set-level data-manipulatie zeker bij grote aantallen een veel betere performance oplevert dan record-level, zelfs als u het programma met lusstructuren binnen-in de database server als stored procedure laat draaien. Dat alleen al zou u moeten overtuigen van het nut van set-level werken.
Het relationele model zegt echter niets over implementatie, dus moet er ook een logisch argument voor zijn. Dat logische argument is natuurlijk precies hetzelfde als het argument waarom set-level selecties beter zijn dan 'uitgeprogrammeerde' record-level equivalenten.
Met een een set-level operatie specificeer je wat je wilt, niet hoe je dat wilt. Dat is eenvoudiger, biedt minder kans op fouten en meer kans op een optimale werking onder verschillende omstandigheden, aangezien het DBMS allerlei additionele informatie, bijvoorbeeld statistieken, kan gebruiken om de specificatie te vertalen in een uitvoeringsstrategie.
Toegegeven, het set-level denken vergt soms een wat andere manier van denken. Zo werk ik momenteel op een project waarin een financiële applicatie wordt ontwikkeld. Een onderdeel van de applicatie is het uitspuwen van grote hoeveelheden correspondentie, zoals facturen, brieven en rapportages. De bedoeling daarbij is dat de applicatie panklare 'records' klaarzet, waarna een separaat print-pakket aan de slag gaat om de records te 'mail-mergen' met templates om de uiteindelijke output te produceren. De elementen die op de correspondentie moeten komen zijn zaken als naam- en adresgegevens, datums, factuurnummers en factuurregels.
Om een en ander flexibel te houden willen we een duidelijke scheiding tussen het algemene mechanisme dat bepaalt welke correspondentie wanneer de deur uitmoet, en de specifieke programmatuur die de variabele informatie 'berekent'.
In eerste instantie ging daarom de gedachte uit naar een klein programmaatje of SQL statement dat per soort informatie (bijvoorbeeld naam, woonplaats, factuurnummer) de waarde berekent, totaal zo'n 150 stuks. Per brief worden deze programmaatjes dan een voor een aangeroepen om het gewenste output record voor de mail-merge samen te stellen. Met gemiddeld enkele tientallen elementen per brief en een beoogde output van vele honderdduizenden brieven per maand betekent dat goede zaken voor de hardware-leveranciers.

> Met een een set-level operatie specificeer je wat je wilt, niet hoe je dat wilt

Terug aan de tekentafel leerde analyse dat het niet moeilijk is welke brieven elke dag moeten worden afgedrukt. Ook bleek dat de variabele informatie op de correspondentie vaak gecombineerd kan worden in samenhangende groepen, die ook in de onderliggende database vaak uit samenhangende tabellen komen, zoals de verschillende velden voor naam-, adres- en woonplaatsinformatie. Daarbij kwam nog dat er weliswaar enkele honderden verschillende soorten output bestaan, maar dat veel groepen van variabele informatie telkens weer voorkomen, met wederom de naam-, adres- en woonplaatsinformatie als mooi voorbeeld. Het bleek derhalve mogelijk om de dagelijkse (of beter: nachtelijke) output te genereren met een handvul bulk-inserts en -updates, zonder dat de beoogde flexibiliteit geweld aan werd gedaan.
Het aantal specifiek te schrijven programma's is vele malen lager dan in de oorspronkelijke oplossing, terwijl deze individuele programma's nauwelijks complexer van aard zijn dan de 150 programma's uit de oorspronkelijke oplossing. Dat de performance subliem is hoeft geen betoog.

Vooruitblik

Zoals beloofd gaan we het de volgende keer hebben over views en logische en fysieke gegevensonafhankelijkheid. Ik kan u alvast verklappen dat het een pijnlijke confrontatie voor de heersende relationele orde gaat worden.

Frido van Orden (frido.van.orden@faapartners.com) is partner bij FAA Partners.