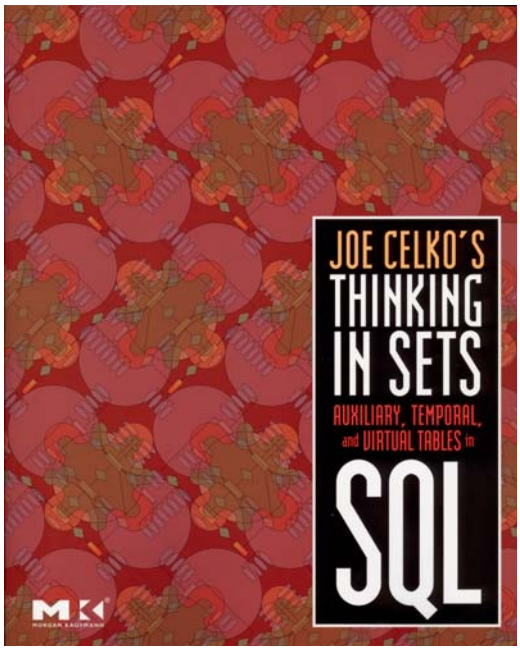
Joe Celko's Thinking in Sets

Rick van Rein

En alweer rolt een nieuwe introductie op SQL de markt op, is dat nog nodig? Waarschijnlijk, want de benadering van Joe Celko is nadrukkelijk anders dan de boeken die we allemaal kennen; en naar eigen zeggen is dat volgens zijn ervaring in het SQL-onderwijs hard nodig.



De basisgedachte achter Thinking in Sets is dat veel SQL-code met de verkeerde pet op wordt geschreven. Men denkt vanuit een programmeertaal die men goed kent, en mist de kans om SQL optimaal te benutten.

Paradigma's

Programmeertalen volgen een bepaald paradigma; de meest gangbare talen zijn procedurele ofwel imperatieve talen waarin commando's worden gegeven om variabele waarden te manipuleren. In objectgeoriënteerde talen sturen we boodschappen naar objecten met het verzoek om iets te doen als dat het ontvangende object uitkomt. Velen kennen Prolog's logica van horen zeggen. Talen als LISP, Haskell, ML en Miranda behoren weer tot de functionele talen, een subset van de applicatieve talen.
Al deze taalgroepen kennen een eigen paradigma, en SQL kent ook zijn eigen 'paradigma' gebaseerd op verzamelingenleer.
De these van Joe Celko is dat het van levensbelang is het paradigma eigen te maken in plaats van de syntax.
Een paradigma bepaalt de manier waarop men dient te denken als men met een programmeertaal werkt; die vertaalt dan vrij natuurlijk in de taalconcepten die daarop gebouwd zijn. Wie imperatieve gedachten heeft over SQL loopt onophoudelijk tegen vermeende beperkingen van de taal aan, en zal nooit op goede voet met zo'n taal komen te staan, terwijl iemand die over data leert denken in termen van verzamelingen snel het gemak van SQL voor datamanipulatie leert inzien.

Voorbeeld

Om dit punt kracht bij te zetten geven we een paar voorbeelden. Een paar jaar terug zag ik studenten die in termen van pre- en postcondities een sorteeralgoritme moesten specificeren. Ze waren zo gewend te denken in termen van implementaties dat ze vooral hoevragen hadden bij de formele taal Z waarin de specificaties werden gesteld. Ze vroegen me om hulp en ik stuurde hun denken aan door de vraag wat ze wisten over een lijst nadat die was gesorteerd. Uiteraard antwoordden ze dat die op volgorde stond. Na enig duwen en trekken kwam daar bij dat de lijst uit dezelfde elementen moest bestaan als de inputlijst. Klaar. "Klaar? Moeten we niet opschrijven welke stapjes gemaakt worden?" Ze stonden perplex – maar hadden wel geleerd dat ze hun problemen met het verkeerde paradigma benaderden.
Even onthand zijn doorgewinterde programmeurs die voor het eerst leren werken met een functionele programmeertaal. Dergelijke talen zijn applicatief, wat wil zeggen dat er geen assignment statement of andere vorm van side-effect is!
Daarnaast is een functie een datasoort die net als alledaagse data zoals integers en lijsten kan worden behandeld. Hoe zou een quicksort-routine in zo'n taal eruit zien? Zonder assignment is het immers niet mogelijk om elementen van plaats te verwisselen? Het antwoord is te werken via recursieve functieaanroepen, en verder de componenten te selecteren die bij die aanroepen worden meegegeven. Een eenvoudige implementatie is bijvoorbeeld:

```
sort []     = []
sort (x:xs) = sort (filter (<=x) xs) ++ [x] ++ sort (filter (>x) xs)
              where filter p []     = []
                    filter p (y:ys) = y: filter p ys, if p y
                                    =    filter p ys, otherwise
```

Hier zien we twee niet-overlappende definities van sort, de ene voor de lege lijst [] en de andere voor een lijst bestaande uit een eerste element x en de restlijst xs. We zien dat er geen assignment wordt gebruikt, maar concatenatie (via ++) van deellijsten; de eerste en laatste componenten zijn het resultaat van een recursief gesorteerde filtering op xs en de tweede is het eerste element van de lijst. In de notaties (<=x) en (>x) zien we functies die bestaan uit een niet-afgemaakte operatie. Het argument dat mist moet nog worden gegeven, maar dat belet een functionele programmeertaal niet om het als argument p in de aanroep van filter te gebruiken; de aanroep p y vult dat argument later wel in. We hebben ons in bovenstaande nog ingehouden om niet met de functie-compositie operator te werken, want ook daarmee zijn soms al prima resultaten te bereiken, vaak zonder dat de functie-argumenten zichtbaar worden. Een 'main function' heeft heel vaak dit formaat. Denk maar aan de pipeline in bash-scripts als voorbeeld van de kracht die dat formaat kent.

Alle deterministische software is in functionele termen te programmeren. Voor sommige toepassingen is dat handiger dan voor andere, maar het is een zeer charmante werkwijze die veel meer kan dan alleen wiskundige problemen oplossen.
Wie in Miranda (zoals hiervoor getoond) of Haskell gaat werken zonder het achterliggende paradigma te kennen zal echter snel verdwalen, of zich gaan ergeren. Want hoewel het mogelijk is om assignment te simuleren door elke functie aan te roepen met een evoluerende state, is men dan niet op een functionele manier bezig en zal men dus snel functionaliteit missen of onhandig vinden in de omgeving die de taal biedt.
Andersom kan het natuurlijk ook raar lopen – ik heb ooit de code mogen zien van een functionele programmeur die was overgestapt op Java – die definieerde om de haverklap inner classes met slechts één methode die als argument werd meegegeven aan een message van waaruit die methode als functie werd aangeroepen – precies de stijl die hierboven voor (<=x) en (>x) wordt gebruikt. Dat mag er handig en kort uitzien in Miranda, maar in Java is het een stuk minder elegant!

SQL op een presenteerblaadje

SQL is ook anders dan de meeste programmeertalen, door het op verzamelingenleer gebaseerde paradigma. Wie denkt in termen van stapjes maken, of assignments doen, zal bedrogen uitkomen, en vaak eindigt men dan met programma's die slechts losse records updaten. De ware kracht van SQL ligt natuurlijk in het updaten van hele sets tegelijk, en die wordt dan niet uitgebuit. Vandaar dat dit boek Thinking in Sets heet, en dat het nadrukkelijk uitlegt wat het idee is van de verzamelingenbasis van een database. Dat is een heel ander geluid dan de klassieke uitleg van een computer, data, de opslag in al dan niet gestructureerde vorm en dan via tabellen naar de eerste code. Deze standaard-uitleg leent zich prima voor misbegrip en het volharden in imperatieve programmeerstijl.
De aanpak van Celko rekent daar nadrukkelijk mee af. Een manier om dat te doen was geweest om in te steken vanuit de semantiek van SQL, maar aangezien relationele calculus (met wiskundige operatoren die werken op verzamelingen van records) niet voor iedereen licht verteerbaar is vermijdt Celko dat. In plaats daarvan geeft hij op heldere wijze pragmatisch beschrijvende uitleg van wat wel en niet het idee achter SQL is. Een groot deel van het boek bestaat uit kleine probleempjes die in SQL worden uitgedrukt, als oefening van het set-paradigma dat Celko voorstaat. Hij gebruikt hierbij ook onpraktische voorbeelden die niet echt in SQL thuishoren, maar dat neemt niet weg dat hij zijn standpunt met stapels voorbeelden onderbouwt, en gaandeweg de vruchtbare denkwijze binnen het set-paradigma bijbrengt. Wie de wiskunde bestudeert krijgt precies de grenzen van dat paradigma mee, maar je kunt het natuurlijk ook via een flinke kluit voorbeelden overdragen, en dat laatste is de aanpak van dit boek.

Goede wijn behoeft een passende zak

Er zijn zat boeken die SQL onderrichten, maar dit boek is in zoverre speciaal dat het afrekent met de invalshoek vanuit een fout paradigma. Wie SQL wil programmeren moet in SQL leren denken, ofwel moet leren denken vanuit dataverzamelingen, en niet vanuit lijsten die met een cursor kunnen worden doorlopen. En daarmee is dit boek een strak pleidooi voor de correcte stijl van werken in SQL. Voor wie telkens tegen 'de beperkingen van SQL' aanloopt is het boek zeker de moeite waard om eens te lezen.

Joe Celko's Thinking In Sets
Auteur: Joe Celko
Uitgever: Morgan Kaufmann
1e druk 2008
ISBN: 978 0123 74137 0

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