De jaren zeventig op hun paasbest

Veertig jaar SQL-cultuur

Rick van Rein

SQL gaat al enige decennia mee. Terugkijkend op die tijd is er ontstellend veel gebeurd. En toch gebruiken we nog altijd SQL. Is de taal zo goed, of zitten we vastgeroest?

SQL is ontworpen in de jaren zeventig, een tijd waarin het normaal was om op cursus te gaan voor je een computer aanraakte, en waarin de machine bepaalde wat mogelijk was, in plaats van de gebruiker. SQL is overduidelijk een kind van die tijd; tekst als interface en als er iets fout gaat dan ligt dat natuurlijk aan de mens. En die mens, die is natúúrlijk technisch aangelegd.
Het relationele model is in juni 1970 door Codd gepubliceerd, en daarna heeft IBM er de taal SQL voor ontworpen. De eerste commerciële implementatie was in 1979 door Oracle, dat toen Relational Software heette.
Er is enorm veel gebeurd in de jaren sinds deze eerste relationele database het licht zag. En dat is niet beperkt gebleven tot de intrede van kleur op onze beeldschermen!

Alles is een taal

In de klassieke informaticaopleiding heerst een impliciet geloof dat alles in een taal kan worden uitgedrukt. Een computertaal, wel te verstaan. Feit is dat die houding ons software heeft gegeven die op een zeer consistente wijze de (niet altijd correcte) code van een mens uitvoert, keer op keer. En dat werkt uitstekend – het werkelijke debuggen is het vinden van onze eigen denkfouten of verkeerde formuleringen, maar we kunnen normaal gesproken aannemen dat de computer foutloos werkt.
Modernere stromingen zien alles als een model – dus een samenspel van concepten volgens vooraf bepaalde relaties. We hebben het dan bijvoorbeeld over objectgeoriënteerde talen, maar zeker ook over ER-diagrammen. Hoewel dat nog altijd een taal kan worden genoemd, is het minder rigide dan een programmeertaal met een lineaire syntaxis, met name omdat het niet langer nodig is om alles hiërarchisch te ordenen. Dat geeft bijvoorbeeld vrijheid om langs verschillende routes door de data te leveren. Wat dat betreft is het een beetje jammer dat een moderne taal als XML dat hiërarchische structuurvoorschrift weer invoerde, en vervolgens speciale constructies nodig heeft om dat laveren weer zo algemeen te maken dat het praktisch bruikbaar is.
Maar dit schetst wel een beetje waarom SQL een tekstuele taal is. Op zich was het in de jaren zeventig enorm hip dat het interactief was en subiet de data uit een database kon opvissen. Maar de interface is dus wel echt gericht op computerverwerking, met een stilistische knipoog naar de Engelse taal.
Toch heeft juist die constructie als taal een stempel gezet op wat we met databases kunnen. De achterliggende constructies van SQL zijn namelijk de verzamelingenleer, met doorsnede en vereniging als basis; en waarin een SELECT-statement meer lijkt op formule 1 dan op een taalconstructie:

$$\{p \in \text{Person} \mid \text{Name}(p) = \text{'John'}\}$$

Die wiskunde bestaat uit operatoren die, vergelijkbaar met de gewone rekenkundige operatoren, verzamelingen met elkaar kunnen combineren tot andere verzamelingen. Dat idee is bijzonder krachtig, maar juist door SQL te beperken tot iets wat op een natuurlijk taal lijkt, zitten we vast aan structuren die sterker zijn voorgeschreven dan op grond van de verzamelingenleer nodig is.

$$\text{Uitkomst} = \{X_{\text{select}}(\varkappa) \mid \varkappa \in S_{\text{from}} \wedge C_{\text{where}}(\varkappa)\}$$

De vorm van formule 2 is nog flink algemeen, maar als we gaan uitbreiden met unions en volgordes dan wordt het al een stuk lastiger. Bijvoorbeeld wanneer we twee geneste selecties met elkaar willen koppelen, iets wat in de algemene verzamelingenleer niet zo'n punt zou zijn.
De structurering als natuurlijke taal heeft hiermee een limiterende uitwerking op de uitdrukkingskracht van SQL. Er zijn allerlei dingen die je zou willen kunnen zeggen, maar daarvoor bestaat

geen formulering in de taal, of alleen een onhandige. Hoe wrang dus, dat we deze taal tegenwoordig genereren uit grafische software die de moderne gebruiker wil bedienen! Want het genereren van de soms geforceerde notatie van SQL is een enorme omweg, die zou kunnen worden opgelost met een directere toegang tot de verzamelingenlogica; een omgeving als Monet maakt dat ook zeer duidelijk. Deze omweg is vooral lastig als het gaat om een stuk software dat allerhande vormen van query's kan opvragen; daarin kan niet zomaar even een generieke codegenerator worden gebouwd, omdat combinaties wel logisch kunnen lijken maar niet zomaar in SQL kunnen worden opgeschreven.

Conceptueel denken ... iemand?

Belangrijker dan de uitdrukkingskracht is misschien wel of een taal aansluit op de belevingswereld van gebruikers. Anders was de innovatie immers wel gestopt na de ontwikkeling van de Turing Machine, een denkbeeldige machine die door wiskundigen is bedacht om precies alles te kunnen doen wat berekenbaar is, maar die in niets lijkt op iets waar een programmeur blij van wordt.
Naast uitdrukkingskracht kan ook de implementeerbaarheid op praktische systemen een rol spelen. Dit is beslist een argument om gegevens als verzamelingen te behandelen, want daarmee blijken hoge doorvoersnelheden te behalen te zijn. Hoewel het sociaal niet altijd acceptabel is om processen in bulk te behandelen, denk bijvoorbeeld aan bankoverschrijvingen of andere klantinteracties, is het toch een nuttig principe om veel voorkomend werk in regelmatige batches uit te laten voeren, omdat daarmee de schaalbaarheid sterk verbetert.
Maar het benaderen van data als verzamelingen lijkt wel wat minder logisch voor de gebruiker van data. Veel moderne software spreekt SQL namelijk aan met één record tegelijk, zonder gebruik te maken van de krachtige verzamelingprimitieven die SQL zo nuttig maken. Soms komt dat voort uit een 'eerste versie' die zonder herontwerp wordt verscheept, om pas veel later te laten zien dat het niet schaalbaar is om zo te werken.

De interface naar de gebruikers heeft hier natuurlijk veel mee te maken, want die bepaalt het soort vragen dat door opdrachtgevers wordt aangedragen. Men is gewend geraakt aan onmiddellijke verwerking, en gebruikt vaak een grafische desktop waarmee men niet te hard hoeft na te denken bij wat er technisch gebeurt. Dat is ook te zien aan het gebruik; veel werktijd wordt verspild aan herhalend werk. In plaats van eenmalig na te denken over hoe een operatie massaal kan worden toegepast op alle data, is het veel gebruikelijker om op zo'n interface hetzelfde soort werk herhaaldelijk te laten gebeuren. Als een gebruiker al zo weinig respect heeft voor de eigen tijdsbesteding, dan geldt dat natuurlijk in nog sterkere mate voor de software die eronder zwoegt.
De houding van de desktopgebruiker is doorgaans iets als "de computer is een gereedschap en moet het gewoon doen". Als je kijkt hoe star dat wordt doorgevoerd dan lijkt het aannemelijk dat dezelfde mensen ook een nijptang gebruiken om schroeven in een plank te timmeren. De juiste tool op de juiste plaats, dat werkt uiteindelijk een stuk soepeler. En het bedienen van een tool is en blijft een ambacht; en dat wordt er echt niet minder op naarmate de tool er complexer op wordt! De selectie van tools, en dat betreft zelfs de meest gebruikte, wordt meestal gedaan op basis van een snelle initiële leercurve, in plaats van op uiteindelijk gebruiksgemak zoals de haalbare efficiëntie.
In zo'n wereld misstaat het enigszins om datastructuren te representeren of van mensen te vragen na te denken over efficiënte bulkverwerking; men is nog net aan lijstjes gewend, maar gekker dan dat moet het niet worden. Als ontwerper moet je dan wel vrij sterk in je schoenen staan als je toch een degelijk conceptueel ontwerp wilt maken.

Een voorbeeld van deze neiging is het populaire gebruik van PHP en MySQL. Deze tools worden (onder andere) gebruikt door het slag code-beuners dat net genoeg SQL kent om er niet in te verzuipen, maar een database-ontwerp maken is vaak te hoog gegrepen. Het is in PHP-kringen heel gebruikelijk om databases te zien als opslagstructuren van enkelvoudige 'objecten'. Geen wonder dus dat MySQL meeliftte op de populariteit van PHP – de ongeoptimaliseerde benadering werkt alleen op een database waarop query's bijna zonder overhead draaien; sterker nog, het werkt daarvoor beter dan een database als Oracle die juist veel tijd investeert in de optimalisatie van query's.

Misschien is het voor zulke toepassingen veel handiger om gebruik te maken van key-value databases, zoals de DBM familie. Het enige nadeel van deze is de afwezigheid van een manier om lijstjes te maken, maar aangezien een SQL wrapper zoals ODBC het nodig maakt om lussen te programmeren, zou men evengoed in een lus door zo'n DBM structuur kunnen lopen. Hoewel deze redenatie niet altijd opgaat, zou hij best vaker mogen worden overwogen als het doel is om snel bij enkelvoudige objecten te komen.

Voorbij XML

Hoewel niet onverdienstelijk als uitwisselingsstandaard voor data, is het verwonderlijk hoeveel bijval XML krijgt. Het scheelt natuurlijk dat er standaard parsers voor zijn, maar die moeten op metaniveau en dynamisch opereren, en vormen soms dus een performance bottleneck. Dat hoeft geen probleem te zijn voor de uitwisseling van data, maar het maakt het beslist ongeschikt als primaire taal om data in te beschrijven.
Veel toepassingen van XML specificeren het datamodel niet precies, laat staan dat de betekenis van die data wordt gedocumenteerd; XML is immers alleen een syntaxis standaard en moet dus worden aangevuld met een beschrijving van de verschillende voorkomens van de gegevens. Die stap van data naar informatie wordt nogal eens overgeslagen in de haast om een product af te krijgen, met de nodige hoofdbrekens op een later tijdstip. Wat dat betreft is er weinig verbeterd ten opzichte van SQL!

De datastructuren in XML zijn ook nogal strak voorschrijvend. Alles is een hiërarchie, en als je het daar niet mee eens bent ... nou ja, dan definieer je de data maar als een onsamenhangende rij van lijsten met gelijkvormige data. Grappig genoeg heb je daar geen XML voor nodig, want je bent dan SQL aan het nabootsen, en dat was ook al een standaard die nog meer van de betekenis van de notatie vastlegt op de koop toe.
Minstens zo onhandig is dat XML geen concept van verzamelingen kent; alles is een lijst, ofwel een verzameling met een volledig gespecificeerde volgorde. Wat dat betreft zijn het twee extremen, waarbij SQL de minste ordeningsinformatie biedt en XML de meeste. Die informatie is juist niet in relationele databases aangebracht om het mogelijk te maken de query te optimaliseren, doordat de verzameling een eleganter wiskundig concept is dan een lijst.
Tussen deze twee extremen ligt het wiskundige concept van een partiële ordening. Dit is een lastig concept om efficiënt te implementeren in een computeromgeving, maar het zou heel erg in de stijl van een optimaliserende database passen om een handige keuze uit alternatieve algoritmen te maken.
De reden om de partiële ordening hier te noemen is dat het heel erg veel voorkomt in de alledaagse realiteit; SQL dwingt ons tot het afzien van enige volgorde, terwijl XML dwingt om juist een volledige ordening aan te brengen en daarmee juist meer informatie te geven dan in het gemodelleerde domein aanwezig is. Beide zijn een tikje onnatuurlijk, hoewel het zowel in SQL als in XML nagebootst zou kunnen worden als we dat echt zouden willen.
Een partiële ordening is een verzameling van objecten met tussen sommige objecten een volgorderelatie, maar niet tussen alle. De enige randvoorwaarden zijn dat er geen cyclische volgordes in mogen zitten (dus; Jan is ouder dan Marie, die ouder is dan Piet, die ouder is dan Jan is ongeldig) en dat we mogen aannemen dat de volgorde rechtstreeks geldt als die indirect geldt (als Jan ouder is dan Marie, en Marie is ouder dan Piet, dan kunnen we concluderen dat Jan ouder is dan Piet).

Deze structuur komt ongelofelijk veel voor in de realiteit. Het is de moeite waard hem eens goed te begrijpen en dan op zoek te gaan naar voorbeelden. Maar pas op, dit werkt wel enigszins frustrerend, want het laat zien hoe beperkt onze datastructureringsprincipes in de programmeerpraktijk van alledag zijn!
In SQL zou de nabootsing van deze structuur neerkomen op het toevoegen van een extra tabel die van twee keys aangeeft dat de

ene voor de andere komt. Vervelend in een taal als SQL is dan om consistent om te gaan met indirect afleidbare volgordes, want dat soort dingen kan in SQL niet bijster handig worden opgeschreven. In XML zit je al met een voorgeschreven volgorde, dus je zou ermee moeten beginnen die te negeren en vervolgens alle volgorde die wel in het probleemdomein bestaat expliciet toe te voegen, op vergelijkbare wijze als in SQL. Zowel de ongeordende verzameling van SQL, als de volledige volgorde van XML zijn uit te drukken in partieel geordende verzamelingen. Het is dus best raar dat dit principe, dat algemener is en beter aansluit bij de gemodelleerde realiteit, niet in onze modelleringtalen zit.

Ondergrondse

Hoewel SQL oorspronkelijk ontworpen is als een taal die door mensen zou moeten worden ingevoerd, wordt het tegenwoordig nog alleen als onderlaag gebruikt, als funderend datamodel waarop dan een applicatie gebouwd wordt. Eigenlijk is het heel vreemd dat we daarbij vasthouden aan SQL als taal, want daarvoor is de taal niet zo heel geschikt; het zou veel handiger zijn een API te hebben waarover generieke operaties uit de relatie-algebra zouden kunnen worden uitgevoerd. Zo'n API zou ook een prima plek zijn om betere modelleringconcepten zoals partiële verzamelingen een plek te geven, tezamen met de optimaler te implementeren extreme vormen van de ongeordende verzameling en de volledig geordende lijst.

Samenvattend

SQL bestaat al decennia en wordt tegenwoordig een stuk anders gebruikt dan oorspronkelijk voorzien. We gebruiken het als automatiseringsonderlaag, waarvoor de syntaxis eigenlijk heel onhandig is. Maar vooral belangrijk is dat de mens lang niet altijd bereid is om gebruik te maken van de grootste kracht van een relationele database, namelijk om efficiënte massa-query's op grote datasets toe te passen. De enorme potentie aan schaalbaarheid die uitgaat van een relationele database wordt in de praktijk maar weinig gebruikt; het is wat dat betreft gezond dat we nu allerlei alternatieven kennen, ook buiten SQL om, om zulke vereenvoudigde toepassingen te bouwen.

Rick van Rein

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