Fraaie truc is nodig om backup te vereenvoudigen

Silly SQL (4): De batchy bitch

Rick van Rein

In de tijd dat SQL werd ontworpen was het heel wat om een computer te hebben in je bedrijf. Het was een enorm bakbeest dat bijna evenveel kon als een AT-type PC, en daar moesten dan natuurlijk flink wat mensen tegelijk op werken vanwege de prijs. Het was de tijd waarin onderzocht werd of het ergonomischer was om groene of amberkleurige tekst-terminals te gebruiken.

Nu is er niets wezenlijk mis met de ergonomie van een tekstscherm – het is een veel efficiëntere interface tussen mens en machine dan wanneer er om de haverklap een muis aan te pas moet komen. Vandaar dat reisbureaus nog steeds met een tekstueel systeem werken, gewoon omdat dat sneller gaat.
En vandaar ook dat schrijver dezes zijn grafische desktop vooral gebruikt om een aantal tekstvensters naast elkaar te plaatsen.
Het enige probleem aan tekst-interfaces is dat ze nogal wat kennis van de bedienende mens verlangen. Het is niet iedereen gegeven om in een Unix-shell door een file-systeem te navigeren, laat staan op een MSDOS-prompt. Maar wie het kán, die werkt wel tien keer sneller dan iemand die alleen maar klikt. Het punt van een grafische interface is alleen dat het minder inleertijd vergt, en deze productiviteit op de korte termijn verwart men doorgaans met gebruikersvriendelijkheid.
SQL is gemaakt op een redelijk kennisniveau van het bedienende personeel, dat wil zeggen; men werd geacht om instructies in een Engels-gebaseerde taal te kunnen componeren. Men werd geacht om de uitvoer te kunnen lezen, en ook nog te begrijpen hoe de platte data gerelateerd waren aan de werkelijkheid die erin vervat lag. SQL is beslist niet voor domme mensen gemaakt.

Meteen maar een hele doos

Geheel in lijn met de noodzaak om er toch even goed voor te gaan zitten om een query te formuleren, is het mogelijk om de operaties lekker generiek te maken. In plaats van handmatige aanpassingen aan vele losse records, is het met SQL immers mogelijk om meteen alles aan te passen wat aan vooraf bepaalde randvoorwaarden voldoet. In vergelijking met mechanischer werk is dat een enorme tijdwinst: om een doos met honderd pingpongballen rood te verven heb je een honderdste van de tijd nodig die geldt voor een doos met tienduizend, maar in SQL is het allemaal evenveel werk. De tijd gaat zitten in de composite van precies de juiste query, niet in afhandeling van elk los record.
De semantiek (ofwel de precieze betekenis) van SQL is gedefinieerd in termen van operaties op verzamelingen, en dat is heel duidelijk voelbaar: een database rolt in hoog tempo over een hele tabel (of record-verzameling) en doet daar zijn ding.

Ook DBMS-implementaties volgen deze batch-gewijze aanpak. Men steekt graag moeite in slimme overhead die een query vereenvoudigt, vooral als deze een operatie in een lus stopt zodat dezelfde verzameling minder vaak afgelopen hoeft te worden.
Op die manier maken databases het mogelijk om vele duizenden records per seconde te behandelen, maar dan wel allemaal binnen dezelfde query; SQL wordt een flink stuk trager als de aanpassingen van de afzonderlijke records in afzonderlijke query's worden gestopt.

> Geen enkel bedrijf kan uitkomen door alleen vaste acties massaal te doen op vaste tijdstippen

De batch-gewijze aanpak van SQL heeft haar vruchten afgeworpen qua productiviteit. Grote corporaties die consumenten massaal voor zich proberen te winnen kunnen niet goed functioneren zonder massaal databeheer. Helaas maken diezelfde corporaties vaak de fout om geen persoonlijke aspecten meer te kunnen beheren, wat ze een slechte naam bezorgt. Gezien de gelijke kosten, maar kleinere baten van een persoonlijke benadering ten opzichte van een massale, levert dat winst op korte termijn.

Botsing met de realiteit

Geen enkel bedrijf kan uitkomen door alleen vaste acties massaal te doen op vaste tijdstippen en verder niets. Vragen en wijzigingen stromen voortdurend binnen. De werkelijkheid is dus niet zo batch-gewijs als SQL ons wil doen geloven. De gevolgen zien we overal. Denk maar aan fouten in rekeningen, correcties daarop, en dan vooral de uitzonderingen 'die het systeem niet aankan'.
Denk maar aan 'u krijgt deze maand nog wel een foute rekening,

maar volgende maand krijgt u het bedrag dan terug'. We schieten soms wat door in onze wens om de realiteit met batch-processen te synchroniseren.
De werkelijkheid bevindt zich buiten de computer. Daar geven we aan de juiste partijen een paar stukjes kennis, met als bedoeling dat die partij daar dan anders door gaat handelen. Als we een statiegeldfles bij een winkel inleveren dan krijgen we gewoon ons geld terug, zonder te moeten wachten op de één of andere Maandelijkse Cyclus van Creditering. Als we een afspraak maken om de volgende dag samen te lunchen dan hoeven we ons niet te haasten om op tijd te zijn voor de Synchroniserende Ronde der Secretaresses. Data worden gewoon doorgegeven en verwerkt zodra ze ontstaan. Klaar.
Een computermodel dat hier op lijkt is dat van de dataflow-machine. Zo'n machine beheert stukjes data (noem ze voor mijn part object) en notificeert andere stukjes data in diezelfde machine bij veranderingen. De notificaties bewegen dan door een berekening heen en leveren een golffront aan veranderingen op. Een spreadsheet geeft een heel aardig voorbeeld van dit principe. Op die manier beheerd kunnen verse data onmiddelijk verwerkt worden, en hoeven geen raar 'getimede' batch-procedures te worden afgewacht. Maar het is geen efficiënte, massale verwerking meer.

Dichter bij huis

Dichter bij huis; voor een database-beheerder is dit probleem ook goed herkenbaar. De intrinsieke afweging die in elke backup-policy wordt gemaakt is dat er niet te vaak een backup moet worden gemaakt omdat dat het DBMS te veel zou belasten; maar het mag ook niet te weinig gebeuren omdat dan mogelijk data verloren gaan bij een grote crash.
Er zijn voor file-systemen hele fraaie trucs beschikbaar om backup te vereenvoudigen. Het rsync-commando – te vinden op http://rsync.samba.org/ – is bijvoorbeeld een prachtig systeem dat razendsnel door files heen scant en daarin de wijzigingen vindt. Alleen de wijzigingen worden doorgezonden. De vergelijking vindt praktisch plaats zonder overdracht van data, zodat eigenlijk alleen veranderingen worden overgezet. Vaker rsync'en kost dus nauwelijks iets extra.
Kijk, en zoiets zoeken we dus ook voor SQL-databases: een manier om snel twee tabellen op verschillende machines te vergelijken, en de wijzigingen over te pompen. Liefst niet op file-niveau omdat dat kan veranderen bij gelijkblijvende data, maar gewoon op het niveau van SQL. Het transactiemechanisme, en daaruit met name atomiciteit, zorgt ervoor dat de data altijd in een consistente toestand worden aangeboden, dus niet halverwege een transactie die tegelijkertijd draait. Ideaal dus: je kunt vaak een backup maken en dan ook nog eens op elk moment dat het je uitkomt. Zodat je backup vrijwel altijd 'up to date' blijft, en vrijwel zonder bloedsporen de originele database kan overnemen. Nadeel aan rsync is natuurlijk dat het afzonderlijke records overzendt, terwijl het soms efficiënter is om de SQL-instructie over te zetten. Voordeel is echter dat het mislukte overzenden van een update er niet toe doet – de backup is na een geslaagde uitwisseling helemaal bij, doordat het verschil wordt gladgestreken. Als bandbreedte niet het probleem is, dan is dit dus veel handiger omdat het beheer veel eenvoudiger is.

Een streep door de rekening is alleen dat SQL batch-gewijs werkt. Het is binnen de standaard al niet mogelijk om te vragen of de database elke wijziging wil doorzenden, zodat een backup dat instant kan oppakken. Maar het is door het niet-interactieve vraag- en antwoordspel van SQL ook onmogelijk om tabellen in de backup met die van de originele server te vergelijken via de efficiënte rsync-methode. Dergelijke oplossingen worden nu dus overgelaten aan de uitwerking in specifieke DBMS-producten, en dat is dus niet goed overdraagbaar. Het koppelen van twee verschillende types database is daardoor ondenkbaar, zodat een selectie uit de databases van leveranciers niet zonder meer overgepompt kan worden in een integraal bestelsysteem. Zeker niet live.
Achteraf bezien had SQL nooit als batch-systeem ontworpen mogen zijn. Maar achteraf heb je natuurlijk makkelijk praten.

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