Carel-Jan Engel over ervaringen rond het Oracle-rdbms

D'r beur'n rare ding'n

Guus kom noar huus want de koei'n stoan op spring'n
De vark'ns mot'n vrett'n en 't hooi mot van 't laand
Guus kom noar huus want d'r beur'n rare ding'n
Dit kan toch zo niet doorgoan, Guus wat is er aan de haand.

In 1975 had Alexander Curly een nummer 1-hit waarin dit refrein voorkwam. Voor hen die het lied niet kennen in twee zinnen het verhaal: veehouder Guus Uut'nwaard gaat met flink wat geld naar de grote stad om een nieuwe tractor te kopen. Guus wordt overweldigd door de verleidingen in de grote stad en zijn geld vloeit richting vrouwelijk schoon in plaats van naar het landbouwmechanisatiebedrijf.

Wat heeft dit nu met Oracle te maken? Niets. Toch is het verhaal dat bij dit liedje hoort illustratief voor de situatie die zich geregeld voordoet bij het bouwen van een applicatie die gebruik maakt van een Oracle-omgeving. Een zo kort mogelijke time-to-market, verkeerde gebruikte

Afblijven!
Never Change A Running System!

extreme programming en tekort aan ervaren personeel leidt regelmatig tot het bouwen van applicaties die op z'n zachtst gezegd *database-unaware* of in ieder geval 'Oracle-unaware' zijn. Nu kun je dat die bouwers niet kwalijk nemen. Vaak zijn ze ontoerekingsvatbaar: zij krijgen geen tijd voor opleiding, hun baas schermt ze niet af van de grillen van de opdrachtgever, er moet te veel in te weinig tijd gebeuren. Verder is mijn ervaring als dba in omgevingen waar extreme programming wordt toegepast, dat het vooraf ontwerpen van een datamodel strijdig wordt verklaard met de methode. Iedere ontwikkelaar heeft de volledige vrijheid columns of tables aan het datamodel toe te voegen als dat noodzakelijk is voor het stukje functionaliteit dat hij aan het bouwen is. Dat dit niet leidt tot een optimaal datamodel laat zich raden.

LEK

Nu ben ik zeker niet zo dogmatisch ingesteld dat voor het kleinste systeempje een volledig traject van CMM level 4 moet worden doorlopen, integendeel. Voor een klein systeem met een voorzienbare korte lifecycle kan dit een perfecte methode zijn. Maar voor een systeem waarvan de gegevens naar verwachting jarenlang raadpleegbaar moeten zijn, is een doordacht en toekomstgericht datamodel onontbeerlijk. Helaas is dit in de praktijk vaak anders en wordt het datamodel ondergeschikt gemaakt aan de functionaliteit. Ik hoef op deze plaats niet te verdedigen dat dit de onderhoudskosten van een systeem extreem laat oplopen.

Een ander probleem dat zich op dit vlak voordoet is de multi-vendor-omgeving waarin de Oracle-database wordt geplaatst.
Op basis van 'best-of-breed' (wie bepaalt wat *best* inhoudt? Bestaan daarvoor objectieve criteria?) wordt een grote diversiteit aan tools en platforms naar binnen gesleept en vervolgens aan elkaar geknoopt. Als het eenmaal werkt: *afblijven! Never Change A Running System!*

Kort na het actief worden van de applicatie gaat alles nog prima. De database is vrijwel leeg -de gegevens pas-

sen volledig in memory- en de databaseserver verwerkt de gegevens met twee vingers in de de neus... totdat de business een succes wordt en enige vulling van de tabellen wordt bereikt. Dan blijkt het schip lek; maar voordat je dat lek boven water hebt...!

DE KLOMPEN VAN GUUS

Zo werd ik kort na de liberalisering van de mobiele telefoonmarkt bij het systeem voor customer care & billing van een van de nieuwe aanbieders geroepen: ze konden al ettelijke weken niet meer factureren. Dat was op zijn minst problematisch voor de business. De situatie vond haar oorzaak in twee elkaar versterkende problemen: de database was heel erg langzaam en de factureringssoftware was niet stabiel. Dat betekende dat een eventueel mislukte batchrun moest kunnen worden teruggedraaid. Het window voor de batch lag tussen 22.00 en 07.00 uur. In die tijd moest een export worden gedraaid, de batch worden verwerkt en eventueel weer een (gedeeltelijke) import kunnen worden uitgevoerd. Omdat tegelijkertijd ook andere batches liepen, was het gebruik van backup/restore niet mogelijk. Een tweede probleem, overdag, was de traagheid van het systeem: als een klant belde, duurde het ettelijke minuten voordat de agent in het callcenter de historie van de klant voor zich had.

Het systeem bestond uit een Cobol-backend en een Windows-gebaseerde GUI. Oorspronkelijk was een en ander opgezet voor een C-Isam-bestandsstructuur. De ontwikkelomgeving kende een eigen repository. Het geheel was voorzien van middleware die de C-Isam-omgeving compatible maakte met een Oracle 7-database.

Zoiets kan niet goed gaan - dat voel je zelfs op Guus' klompen aan. Op één na werkten alle leveranciers van componenten van het systeem mee aan het onderzoek. Helaas was de uitzondering uitgerekend de leverancier van de middleware. Deze was van mening dat het niet aan zijn software kón liggen. Aan de hardware kon ook het niet liggen: die bestond uit de zwaarste DEC Alpha-omgeving die op dat moment beschikbaar was, met een zeer grote geheugencapaciteit en veel snelle processoren.

Naspeuringen met tuning-tools leerde dat één tabel wel heel erg frequent werd geraadpleegd. Deze tabel bevatte een 'long raw'-kolom. De tabel bevatte ruim 1 miljoen rijen en de inhoud bleek zeer divers. Op zichzelf hoeft een miljoen rijen geen enkel probleem te zijn. Alle raadplegingen op deze tabel verliepen echter

Productie en repository waren dus niet gescheiden opgeslagen

via een full-table-scan. Uiteindelijk konden we over deze tabel de volgende conclusies trekken: hij vormde het centrale opslagmedium voor álle kolommen van het type 'long (raw)' die in de productiegegevens én de repository voorkomen. Productiedata en repository waren dus niet gescheiden opgeslagen. Vanuit de tabellen waarin de gegevens eigenlijk thuishoorden, werd met een foreign key naar deze centrale tabel verwezen. De omvang werd veroorzaakt doordat de klanthistorie per contact in een long raw-kolom (in deze tabel) werd opgeslagen. Daar is op zichzelf niets mis mee. Maar op het moment dat een klant belde en contact kreeg met een van de driehonderd medewerkers in het callcenter werd zijn historie opgehaald, met een full-table-scan. Deze werd veroorzaakt doordat het gegenereerde SQL-statement geen type-conversie toepaste op de zoekcriteria voor de primary key-kolom: deze was van het type 'raw', en er werd gezocht met een 'char'. Dit leidde tot de full-table-scan: de automatische type-conversie van de databasekolom maakte de index onbruikbaar. Het bewuste SQL-statement werd gegenereerd door... juist, de middleware, waaraan het niet kón liggen.

'SQL IS SQL'

Een tweede probleem ontstond doordat alle SQL-statements als 'platte string' werden aangeboden aan de database. Het gevolg is dat Oracle ieder statement apart moet parsen en valideren. Beter is het gebruik van bind-variabelen in de statements en deze door

Helaas weten de mensen vaak niet beter

middel van een prepare/execute-mechanisme aan de database aan te bieden. Oracle hoeft eerder aangeboden SQL-statements niet opnieuw te parsen als een identiek statement in de SQL-cache kan worden teruggevonden. Metingen wezen uit dat tot 98 procent van de totale uitvoeringstijd van een SQL-statement dat één rij moet opleveren, werd besteed aan het parsen van het statement. Het gebruik van 'prepared' statements levert tot vijftig keer snellere applicaties op! Eenzelfde manier van aanbieden van SQL zien we bij de huidige generatie software (vaak Java en JDBC) ook vaak voorkomen.

Helaas weten de mensen vaak niet beter. Net als boer Uut'nwaard werd overweldigd door de indrukken die hij opdeed in de grote stad zien veel ontwikkelaars door de bomen het bos niet meer, als zij met de uitgebreide mogelijkheden van een Oracle-rdbms worden geconfronteerd. 'SQL is SQL', is het adagium; dat kan niet moeilijk zijn. Het is ook niet moeilijk, maar goede begeleiding is essentieel. Het is van groot belang dat een ontwikkelteam wordt ondersteund door een ervaren dba, die thuis is in de gebruikte database. Op ons rust de schone taak projectmanagers en teamleiders van dit belang te doordringen. Deze dba moet dan niet vlak voor de ingebruikname van de applicatie worden ingeschakeld, maar vanaf het begin! ●

Carel-Jan Engel (cjpengel@ease.nl) is technisch directeur en senior problem solver bij Ease Automation b.v.