Kanttekeningen bij 'Het gaat over niets' uit DB/M 4

Much Ado About Nothing

Lex de Haan

"Ik heb het bovenstaande nummer van DB/M met plezier en aandacht gelezen, met name het tweetal artikelen over ontbrekende informatie en NULL-waarden," schrijft Lex de Haan. "Ik kon het niet nalaten om te reageren." Dit artikel is een samenvatting van zijn commentaar op het artikel van Toon Loonen en Nicolette Verdugt, reacties van de auteurs, en hier en daar een reactie op die reactie. 'Much ado about nothing', met een knipoog naar William Shakespeare.

In zijn inleiding van het artikel 'Het gaat over niets' schrijft Toon Loonen dat een RDBMS bij elke NOT NULL-kolom een extra byte opneemt dat aangeeft of de inhoud wel of niet van toepassing is. Ik vind dit in ieder geval volstrekt onwaar voor Oracle.
Loonen beaamt dat, maar riposteert dat Oracle voor elke kolom het lengte-attribuut oneigenlijk gebruikt om aan te geven dat een kolom de NULL waarde heeft (lengte is dan 0 bytes). Als Oracle het onderscheid tussen NULL en de lege string ('') wel gaat maken denkt hij dat Oracle ook een extra bit of byte nodig heeft. Ik vind Loonen's tegenwerping niet valide, omdat het lengte-attribuut niets met NULL values te maken heeft. (Loonen vindt dat dat in Oracle nu juist wel het geval is en dat is volgens hem het probleem.)

NULL in unieke indexen

In de paragraaf 'NULL in unieke indexen' stelt Loonen dat "het weinig zin heeft om niet ingevulde waarden mee te nemen in de index" en "slechts een heel enkele ongebruikelijke query zou daarvan profiteren". Ik denk dat dit soort fysieke implementatie-details op het logisch niveau niet erg relevant zijn, en een van de voordelen van het relationele model is toch juist dat *alle* query's kunnen worden geformuleerd? Wat maakt een query trouwens ongebruikelijk? En even verderop staat "Oracle zal de NULL-waarden hierbij niet opnemen in de index." Daaraan zou ik willen toevoegen: inderdaad, mits je alleen indexen beschouwt op een enkele kolom, en zolang je alleen over reguliere B*-Tree indexen praat. Niet echt relevant, want waar het hier om zou moeten gaan is hoe de diverse RDBMS-implementaties *omgaan* met UNIQUE constraints, en niet hoe ze die onder de motorkap implementeren.
Loonen zegt aangenaam verrast te zijn dat Oracle de NULL-waarden niet opneemt in de indexen. "Beschouw het als een compliment aan Oracle. Voor situaties als een index op meer kolommen en andere typen index zijn hier inderdaad nog nuanceringen voor aan te geven. Het was niet mijn bedoeling om een Oracle-cursus te geven maar het praktisch gebruik van NULL in de verschillende RDBMS-implementaties kort toe te lichten," aldus Loonens weerwoord.

WHERE-clause

Ik heb grote bezwaren tegen de voorstelling van zaken in de WHERE-clause paragraaf. Er wordt gesuggereerd dat 'onbekend' als 'onwaar' wordt behandeld. Ik citeer: "(...) ook onbekend en wordt door het RDBMS dus op onwaar gezet." Dit is volstrekte onzin, en dat valt ook gemakkelijk aan te tonen, door de WHERE-clause te ontkennen met de NOT-operator. De ontkenning van 'onwaar' is 'waar' en de ontkenning van 'onbekend' is 'onbekend'.
Loonen reageert hierop: "Bij een query worden de rijen geselecteerd waarvoor de conditie in de WHERE clause TRUE is. De rijen waarvoor de conditie FALSE is worden niet geselecteerd. Ook worden de rijen niet geselecteerd waarvoor de waarde van de conditie onbekend is, bijvoorbeeld omdat een kolom met de 'waarde' NULL wordt vergeleken met een constante, bijvoorbeeld 3. In dit opzicht wordt 'onbekend' hetzelfde behandeld als 'onwaar' en dit heb ik willen aangeven.

> De meer dan 20 jaar oude (+) syntax moet vanwege backward compatibility ondersteund worden

Met de conditie WHERE kolom IS NULL wordt er inderdaad een soort derde alternatief geboden. Als je dit 3-valued logic zou willen noemen, mij best," aldus Loonen.
Even verderop staat: "Een serie and's is onwaar als 1 van de condities onwaar is en dat is bij de conditie 'AND NOT A = NULL' het geval." Ook hier hebben de auteurs het mis; 'A = NULL' levert 'onbekend' op, dus 'NOT A = NULL' levert ook 'onbekend' op, en

een iterated AND-constructie levert alleen maar 'waar' op als alle componenten 'waar' opleveren. Omdat er in dit geval altijd (minstens) een component 'onbekend' oplevert, is het resultaat van de iterated AND dus 'onwaar' of 'onbekend'.

Outerjoin

Het is jammer dat in de Outerjoin-paragraaf de (+) syntax wordt gepresenteerd als 'de' Oracle-syntax. Oracle kan het ook niet helpen dat het al outerjoins ondersteunde toen er nog geen ANSI/ISO standaard outerjoin bestond. Mijn kopie van de documentatie van Oracle versie 3 (uit 1982) beschrijft deze syntax al. Verder vind ik de beschrijving aan het eind van de paragraaf "bij deze selectie worden de gevraagde kolommen (...) op NULL gezet", wat ongelukkig. Ik prefereer de uitleg waarbij aan een dergelijke rij van de klantentabel bij gebrek aan een match een kunstmatige 'all-NULL' landenrij wordt toegevoegd.
Muggenzifterij misschien, maar juist in deze materie kunnen we niet precies genoeg zijn, omdat de mogelijke misverstanden toch al voor het oprapen liggen. "Dit is (weer) geen kritiek op Oracle, maar een constatering," reageert Loonen. "Ik leg in het artikel het begrip 'outerjoin' uit en geef daarna drie voorbeelden met verschillende syntax. Bijna alle producten (in elk geval Oracle, Sybase en MS SQL server) liepen hier (ver) voor op de standaardisatie. Verder: ik wil ook graag precies zijn. Ik heb het resultaat van de outerjoin beschreven, onafhankelijk van hoe het wiskundig is gefundeerd of door het product is geïmplementeerd, namelijk een rij met klanten waarbij de kolommen uit de landentabel niet ingevuld/onbekend zijn (c.q. de NULL-waarde hebben etcetera)."
Hoewel ik veertien jaar voor Oracle heb gewerkt, voel ik me absoluut niet geroepen om als pleitbezorger van Oracle op te treden. Mijn punt van kritiek was alleen dat Oracle geen keus heeft: deze meer dan 20 jaar oude (+) syntax moet vanwege backward compatibility ondersteund worden. De ANSI/ISO OUTER JOIN-syntax wordt inmiddels al diverse jaren (en Oracle versies) ondersteund.

Codd en Date

In de paragraaf 'NULL volgens Codd en Date' schrijft Loonen: "Ook de logica die gebruikt wordt in de WHERE clause is zeker in 2-valued logic maar ook nog in 3-valued logic onvoldoende goed uitgewerkt." Ik kan me niet voorstellen dat dit aan Codd en/of Date is ontleend, omdat het refereren naar tweewaardige logica in de context van de SQL WHERE clause irrelevant is: de SQL WHERE clause hanteert een driewaardige logica. Bovendien, de tweewaardige logica bestaat al jaren, is wetenschappelijk (wiskundig) grondig onderzocht en onderbouwd, en mag gerust "sound and complete" worden genoemd. Dat er het nodige aan te merken valt op de implementatie van de driewaardige logica, daar zijn we het denk ik allemaal wel over eens.
"Ik denk dat Codd zich wel in de implementatie van NULL kan vinden," meent Loonen en verwijst naar het artikel van Frido van Orden in DB/M nr 4 van 2004 (pag. 32). "Maar ik citeer uit de handout van het Seminar 'Relational Remodeled' met Chris Date dat Array dit voorjaar heeft georganiseerd: *And SQL includes an imperfect implementation of 3-valued logic (3VL) – and introduces many additional flaws*. Wat ik in het kader heb willen aangeven is slechts dat ook bij als zeer deskundig bekend staande personen het begrip NULL nog ter discussie staat," aldus Loonen.
Natuurlijk staan NULL-values ter discussie, en ook ik heb beide seminars in Amsterdam bijgewoond. Mijn bezwaar betrof met name de toevoeging "en zeker in 2-valued logic." Chris Date spreekt dan ook expliciet over N-valued logic (N>2).

Verder worden alternatieven 1 en 2 wel erg gemakkelijk als 'onpraktisch' van de hand gedaan. Ten eerste heb je natuurlijk geen separate tabel nodig voor elk optioneel attribuut in combinatie met de primary key; dat is alleen maar nodig (en beslist de moeite van het overwegen waard) voor ieder subtype. Dat maakt een datamodel niet 'onnodig complex' maar juist 'accurater'; zodra een bepaald attribuut 'niet van toepassing' is, dan ben je dus attributen van een subtype aan een supertype aan het hangen. De optimizer zorgt wel voor adequate performance van de eventuele outer joins, en views kunnen voor de eindgebruiker (of applicatie) het interface desgewenst vereenvoudigen. Er gaat niets boven een goed database design, en er is later (in productie) niets zo rampzalig als een inferieur database design. Loonen beaamt dat van harte: "Goed database design is essentieel voor een goed systeem. Ik heb echter ook gezien dat het te ver doorvoeren van het principe waarbij elk NULL-attibuut werd vermeden, tot onpraktische en onnodig complexe gegevensmodellen leidt," meent Loonen, "Daarbij los je hiermee niets op omdat de NULL's in de outerjoins of views toch weer terug komen."

Leveranciers

Tot slot citeer ik uit de Conclusie en Samenvatting: "Maar zelfs als de syntax gelijk is kan het resultaat anders zijn, zoals te zien is bij unieke indexen met NULL-waarden en bij de sortering van NULL's." Het eerste punt is een non-issue, omdat dit zoals eerder aangegeven een puur fysieke aangelegenheid betreft: op het logisch niveau mogen we alleen maar over constraints praten, en niet over eventuele leverancier-afhankelijke implementaties. Leveranciers zijn hier namelijk volledig vrij in. Wat het tweede punt betreft: de ANSI/ISO standaard schrijft geen default sorteergedrag voor, maar biedt de NULLS {FIRST|LAST} syntax om precies aan te geven hoe je NULL-waarden gesorteerd wil zien, onafhankelijk van de ASC/DESC sorteervolgorde zelf.
"Ik geef in de conclusie aan dat een DBA of programmeur (of mogelijk zelfs een gebruiker die een query loslaat op de database) bij verschillende producten verdacht moet zijn op verschillend gedrag. Het is zeker geen waardeoordeel, alleen een constatering," aldus Loonen.

Lex de Haan is eigenaar van Natural Join B.V. Hij is al jaren betrokken bij de internationale SQL-standaard en lid van de Nederlandse 'national SQL-body'.