Regel 4: Actieve online catalogus gebaseerd op het relationele model

De relationele catalogus: een handige faciliteit

Frido van Orden

Na het conceptuele geweld van de eerste regels wordt het wat rustiger. Deze keer staat een regel centraal die in vrijwel heel zich relationeel noemend database-land op correcte wijze nageleefd wordt, al begrijpen helaas weinigen waarom.

Alvorens op de relationele catalogus in te gaan volgt echter eerst nog even een terugblik op de vorige aflevering, die over de problematiek ging van ontbrekende informatie, beter bekend als 'NULL-waarden'. Het blijkt dat er maar liefst twee onjuistheden in het artikel zaten, beide ontdekt door lezer Lex de Haan.
Als eerste geeft De Haan aan dat het onderscheid dat Codd maakt tussen de verschillende soorten NULL-waarden, pas geïntroduceerd is in RM/V2, zijn verbeterde versie van het relationele model daterend uit 1990 en derhalve van later datum dan de regels waarover we het in deze serie hebben. Sommigen zijn dan ook van mening dat Codd's regels zijn laatste echt goede werk over het relationele model waren en dat hij daarna is gaan dwalen. Tot mijn spijt heb ik niet de beschikking over Codd's originele artikel; schandelijk genoeg zijn de meeste van zijn publicaties zeer moeilijk te bemachtigen.
De verwoording die in deze artikelserie gehanteerd wordt is dan ook niet van Codd's eigen hand, maar die van ... Chris Date! Normaal gesproken is hij een strijder aan Codd's zijde, behalve op het gebied van verschillende soorten NULL-waarden. Ik zal de laatste zijn om Date van het incorrect weergeven van Codd's bedoelingen te beschuldigen, zeker zonder de originele bron ter beschikking te hebben, maar pikant is het natuurlijk wel.
Verder geeft Lex de Haan aan dat de beschrijving van het gedrag van NULL in SQL niet correct is weergegeven. In SQL is een boolean niet tweewaardig (true of false) maar driewaardig (true, false of unknown). Alle booleaanse vergelijkingen en operatoren met NULL leveren in SQL als resultaat Unknown op, en niet NULL zoals onterecht in de vorige aflevering stond vermeld.
Een betere illustratie van de ellende van het gebruik van NULL door SQL dan het erin sluipen van deze fout is nauwelijks mogelijk.

De correcte versie moet zijn:
- NULL = anything ➔ Unknown, in het bijzonder:
  NULL = NULL ➔ Unknown
- NOT(NULL) ➔ Unknown
- NULL OR anything ➔ Unknown
- NULL || 'any string' ➔ NULL (deze klopte, als enige, wel!)

Een paar eenvoudige SQL query's ter illustratie van het effect:

```
Select * from tabel where veld is null: 23 rows
Select * from tabel where veld = null: 0 rows
Select * from tabel where not(veld = null): 0 rows
```

Ergo:

```
Select * from tabel where (veld = null) or
                          not(veld = null): 0 rows !!
```

Relationeel: all or nothing?

Terug naar het eigenlijke onderwerp van deze keer: de relationele catalogus. Een handige faciliteit, zult u zeggen, en met u vele leveranciers van relationele DBMS-producten. Maar waarom eist Codd van een relationeel DBMS dat het over een relationele catalogus beschikt? Een dergelijke eis lijkt toch van een geheel andere orde dan de eerder behandelde eisen zoals 'alle informatie wordt op één manier gerepresenteerd', 'primaire sleutels' en systematische behandeling van NULL-waarden. Een relationele catalogus is toch niet fundamenteel, maar op z'n best een handigheidje?
In zekere zin is dat waar. Waar de regels 0 tot en met 3 funda-

Relational Rules (6)

De vorig jaar overleden dr. E.F. Codd werd wereldberoemd met zijn serie publicaties over een gegevens(meta)model dat later bekend zou worden onder de naam 'Relationeel Model'. De eerste uit die serie publicaties was een intern IBM Research Report dat uitkwam in 1969. 2004 zal dus gelden als een lustrum – 35 jaar Relationeel Model.
Frido van Orden schrijft in Database Magazine een serie artikelen over de betekenis en de erfenis van het gedachtengoed van Codd. Aflevering zes gaat over de vierde regel: Actieve online catalogus gebaseerd op het relationele model. Het systeem dient een online, inline, relationele catalogus te ondersteunen die toegankelijk is voor geautoriseerde gebruikers door middel van hun reguliere vraagtaal.

mentele waarden vertegenwoordigen, neigen de regels 4 tot en met 12 toch meer naar, oneerbiedig gezegd, implementatie-details. Toch heeft Codd niet voor niets alle regels op gelijke hoogte gesteld en geen van de regels als optionele eis opgenomen. Het verschil tussen regels 0 tot en met 3 enerzijds en 4 tot en met 12 anderzijds, is eigenlijk het verschil tussen 'u kunt niet zonder' en 'u zou niet zonder moeten willen'. Met gevoel voor de manier waarop zaken werken in de IT-wereld heeft Codd willen voor komen dat er vele, basale, 'hakken over de sloot' relationele implementaties zouden kunnen ontstaan, die zich allemaal met Codd in de hand 'relationeel' zouden mogen noemen. In plaats van een standaard met veel ontsnappingsmogelijkheden (of 'compliance levels' als u het netjes wilt formuleren) legde hij de lat simpelweg hoog. Een systeem is pas relationeel als het alle mogelijkheden van de relationele filosofie uitbuit.

Catalogus

Wat is dan toch de bijzonderheid van de relationele catalogus die Codd ertoe heeft gebracht haar verplicht te stellen voor relationele DBMS'en? Het antwoord op die vraag is even simpel als verrassend: er is totaal NIETS bijzonders aan een relationele catalogus. Een relationele catalogus is een database, net zoals uw privé-adressenbestand, de loonadministratie van uw werkgever of welke andere geordende gegevensverzameling men zich ook maar kan voorstellen.
Vele databases, zoals uw adressenbestand, zijn puur ontwikkeld voor registratieve doeleinden. Hun enige doel is het vastleggen van informatie. Andere databases maken deel uit van een of meer informatiesystemen, die de gegevens in de database gebruiken voor het uitvoeren van berekeningen mee uit te voeren, of andere activiteiten zoals het doen uitgaan van uw salarisbetaling.
Ook de relationele catalogus maakt deel uit van een informatie-systeem, namelijk het RDBMS zelf. Het RDBMS gebruikt de informatie in de catalogus om zijn werk te kunnen doen, bijvoorbeeld het zo snel en efficiënt mogelijk ophalen van de adressen van uw kennissen of de looncomponenten van uw salaris. Zoals de wereld voor uw adressensysteempje bestaat uit personen, adressen en telefoonnummers, en voor de loonadministratie van uw werkgever uit medewerkers, salariscomponenten en betalingen, zo bestaat de wereld van uw RDBMS uit tabellen, indexen en table spaces.

Dat is mooi, zult u zeggen, maar wat heb ik daaraan? Ik kan, door met mijn favoriete query tool de catalogus van mijn RDBMS uit te lezen, kijken wat de structuur van mijn database is. Maar dat doen gespecialiseerde tools als de Database Explorer, TOAD of vul uw favoriet maar in, toch veel mooier en makkelijker? Natuurlijk, net zoals u liever met Exact uw salarisadministratie raadpleegt dan met SQL*Plus. Een op maat ontwikkelde applicatie werkt altijd plezieriger dan een all-purpose query tool, hoe fraai op zich ook. Uw favoriete database tool is echter op precies dezelfde wijze ontwikkeld als uw salarispakket of het maatwerksysteem dat u misschien momenteel wel aan het ontwikkelen bent. Het leest gegevens uit de relationele catalogus en toont die op een prettige manier op het scherm.
En schrijven dan? Ik kan toch geen 'insert into systables' doen net zoals dat kan met 'insert into medewerker'? Jawel, dat kan wel, maar helaas niet zomaar met Oracle, DB2 of welk relationeel product van vandaag dan ook. Het is zelfs wetenschappelijk bewezen. We moeten het echter niet simpeler maken dan het is. Bij een serieuze salarisadministratie voert u echt geen medewerker op met alleen 'insert into medewerker'. Een insert van tientallen records in vele verschillende tabellen ligt dichter bij de waarheid. Evenzo voert u geen nieuwe tabel in uw relationele catalogus op met alleen 'insert into medewerker'. Wat dacht u van kolommen, sleutels, sleutelkolommen, indexen en de meer technische zaken als 'extents' en 'table spaces'. Daar komt nog bij dat het aanmaken van een tabel in uw RDBMS niet alleen een kwestie is van het vastleggen van informatie in de catalogus. Zo zal er ook opslagruimte moeten worden geclaimd en eventueel gealloceerd. Net zoals bij een salarisadministratie trouwens, al vinden activiteiten als het inrichten van een werkplek daar buiten het informatiesysteem plaats.

> Een systeem is pas relationeel als het alle mogelijkheden van de relationele filosofie uitbuit

Om een en ander handig voor u (en programmeurs) samen te vatten is er een batch-programma (of misschien wel stored procedure) create_employee en zo praat de rest van de salarisadministratie met de database. Het overeenkomstig 'catalogus-programma' luistert in SQL naar de naam 'CREATE TABLE'.

Orthogonaliteit

Dit is leuk allemaal, en wetenschappelijk zeer boeiend, maar u trappelt vast nog niet van enthousiasme. Codd moet toch echt met iets beters komen om deze regel te kunnen verantwoorden. Welnu, dat heeft hij reeds gedaan maar vermoedelijk zonder dat u het in de gaten had.
Zoals al eerder gesteld gebruikt het DBMS de informatie in de catalogus om zijn werk te doen. Soms wordt die informatie gebruikt om de functionaliteit van het DBMS te verbeteren, denk aan informatie over indexen die het voor de optimizer mogelijk maken om een query efficiënter en sneller te beantwoorden. Andere informatie wordt gebruikt om extra functionaliteit te bieden, zoals de mogelijkheid om gebruikers te autoriseren voor bepaalde gegevens of de mogelijkheid om programma's uit te voeren indien een bepaalde operatie op de database wordt uitgevoerd (triggers).
En weet u wat het mooie is? Al die functionaliteit werkt niet alleen op uw eigen database, maar ook op de catalogus zelf! De functionaliteit om constraints te specificeren die voorkomen

dat er onbestaanbare informatie in de database komt, zorgt er ook voor dat de catalogus zelf integer blijft. Indien de optimizer verbetert draaien niet alleen uw query's sneller maar wordt het DBMS zelf (dat vele query's op zijn eigen catalogus uitvoert) ook sneller. U kunt gebruikers autoriseren om de catalogus al dan niet te mogen raadplegen. U kunt zelfs triggers maken die afgaan bij het aanmaken van een nieuwe tabel of het wijzigen van een kolomdefinitie!

Mist u sommige van deze functionaliteiten in uw favoriete RDBMS, dan is dat meestal niet zozeer omdat het niet zou kunnen maar omdat veiligheidshalve bepaalde zaken dichtgetimmerd zijn, zelfs voor gebruikers met de meest uitgebreide autorisaties. In vroege versies van Oracle kon men iets zeggen à la 'delete from systables' en daarna was Oracle dood en hielp alleen een herinstallatie. Een verkeerd geprogrammeerde trigger op de catalogus kan er theoretisch voor zorgen dat het DBMS het spoor in zijn eigen catalogus bijster raakt. U mag het betutteling noemen, of een uitwas van de Amerikaanse claim-cultuur. En vindt u het echt ondraaglijk, ga dan aan de slag met een open source RDBMS en laat alles werken wat God verboden heeft. Helaas is het populaire MySQL een van de weinige RDBMS 'en (pardon: zich relationeel noemende DBMS'en) zonder relationele catalogus, maar probeert u het gerust eens met een ander.

> De repository is passief en wordt alleen gebruikt om programmatuur mee te genereren

Voorwaar een mooi verhaal, niet? De reden achter deze onverwachte kracht van de relationele catalogus is dat deze *zelfbeschrijvend* is. Dat betekent dat de catalogus niet alleen de beschrijving bevat over uw eigen database, maar ook de beschrijving van zichzelf. In tabellen als SYSTABLES en SYSCOLUMNS vindt u ook de definities van deze tabellen zelf terug! Bij het uitvoeren van query's op de catalogus maakt het DBMS gebruik van gegevens uit diezelfde catalogus; een Droste-effect. U kunt dat zelf controleren door de sql trace-faciliteit van uw DBMS aan te zetten. En als u toch bezig bent: huiver van de schendingen van elementaire datamodelleerregels in de catalogus van uw favoriete DBMS!

Goed voorbeeld doet slecht volgen

Het zal u niet verrassen dat het idee van een catalogus niet alleen in relationele databases navolging heeft gekregen. Case tools, workflow management-omgevingen en zelfs complete ontwikkelomgevingen hebben het goede voorbeeld gevolgd. De catalogus is in dergelijke omgevingen beter bekend onder de naam *repository*. Jammer genoeg heeft men gelijk met de term 'catalogus' ook het concept van zelfbeschrijvendheid overboord gegooid. Als u Oracle Designer installeert, of welk ander repository gebaseerd product dan ook, dan is de repository initieel leeg. Oracle Designer is dus niet ontwikkeld met Oracle Designer zelf, in ieder geval niet op *runtime*. In het beste geval is de repository gebruikt om programmacode uit te genereren.
Vrijwel alle repository-gebaseerde tools werken tegenwoordig zo: de repository is passief en wordt alleen gebruikt om programmatuur mee te genereren. Een eenvoudige aanpassing van de gegevens in de repository leidt dan ook niet zelden tot uren werk om programmatuur te genereren, bouwen en distribueren. Hoe ver zijn we afgedreven van het RDBMS, waar iedere wijziging van de catalogusgegevens direct wordt doorgevoerd! En een van de hoofdargumenten die voorstanders van code-generatie telkens weer aandragen, is natuurlijk dat het iedere keer moeten raadplegen van de repository leidt tot een slechte performance. Inderdaad, dit is hetzelfde argument wat in de beginjaren tegen het relationele DBMS werd gehanteerd. L'histoire se répéte.
De volgende keer blijven we nog even in de wereld van SQL en CREATE TABLE. We pakken dan twee regels in een keer: over de taal waarin met het RDBMS wordt gecommuniceerd en over views.

Frido van Orden (frido.van.orden@faapartners.com) is partner bij FAA Partners.