Het gaat over niets (2)

Weinig onjuist gebruik van NULL in relationele systemen

Toon Loonen en Nicolette Verdugt

In een vorig artikel (zie DB/M 1) zijn de functionele aspecten en de implementatiedetails van het gebruik van NULL uitgewerkt. In dit tweede deel komt de uitwerking naar query's aan de orde.

Het concept NULL is in alle RDBMS-producten geïmplementeerd. Voor elke kolom in de tabel kan worden aangeven of deze verplicht (NOT NULL) of niet verplicht is. Onze gewoonte is om dit ook altijd expliciet aan te geven, dus bij elke kolom NULL of NOT NULL op te nemen en niet op de default-waarde voor NULL/NOT NULL af te gaan. Indien niets is opgegeven is standaard NULL toegestaan, althans in Oracle, SQL Server en Sybase. Verder is het belangrijk om bij de verplichte kolommen ook expliciet NOT NULL op te nemen, ook al zou de gebruiker deze gegevens toch wel moeten invullen omdat de applicatie het ook afdwingt. Ten eerste omdat het RDBMS eventuele fouten van de applicatie afvangt, maar ook omdat een RDBMS bij elke NOT NULL-kolom een extra byte opneemt dat aangeeft of de inhoud wel of niet 'van toepassing is'. Het onnodig op 'NULL toegestaan' zetten van een kolom, maakt de records in de database dus onnodig groter. Ook kan in een enkel geval de performance lager zijn door een slechter geoptimaliseerde query.

Niet-relationele systemen

Bij niet-relationele systemen, bijvoorbeeld C-ISAM of text files, bestaat de NULL-optie niet. Voor een getal (de hiervoor genoemde temperatuur) zal dan een extra kolom of byte nodig zijn waarmee aangegeven wordt of de waarde in het integer- of float-getal wel van toepassing is. Voor veel getallen en zeker voor tekstattributen zal dit niet strikt nodig zijn. De hiervoor genoemde boete kan dan gewoon 0 (zero) blijven en voor tekst-strings kan de (meestal vaste positie in het record) gevuld worden met spaties. Maar bij bijvoorbeeld de hiervoor genoemde temperaturen kan het wel nodig zijn om zo'n NULL-indicatie mee te nemen.

Null in unieke indexen

Indien een (al of niet unieke) index op een kolom wordt geplaatst die leeg mag blijven (bijvoorbeeld een index op meisjesnaam in een personeelssysteem), dan heeft het weinig zin om de niet gevulde waarden mee te nemen in de index.
Slechts een heel enkele ongebruikelijke query zou daarvan profiteren. Hier zien we verschillen tussen diverse producten:

1. Oracle zal de NULL-waarden hierbij niet opnemen in de index. Een dergelijke index kan bij Oracle dus ook uniek zijn, ook al zijn er veel NULL-waarden. Als de ingevulde waarden maar uniek zijn is het goed;
2. SQL Server en Sybase nemen de NULL-waarden wel mee in de index en deze kan dan niet meer uniek zijn.

Sortering en groepering

Bij oplopend sorteren komen de NULL-waarden:
1. in Oracle achteraan;
2. in SQL Server en Sybase vooraan.

Bij aflopend sorteren is de volgorde andersom. Bij het groeperen met group by, worden alle NULL-waarden bij elkaar in 1 groep geplaatst en standaard ook in de hiervoor aangegeven volgorde gepresenteerd.

Waarde NULL geven aan een kolom

Voor de voorbeelden wordt de tabel en vulling gebruikt, zoals in het kader hieronder te zien is:

```
create table test_NULL (
a char(10)      NULL,
b varchar2(30)  NULL,
c1 number(10)   NULL,
c2 number(10)   NULL,
d date          NULL);


insert into test_null values ('A', 'A', 123,
                                      123, NULL);
insert into test_null (a, b, c1, d) values
                      (NULL, 'A', 123, sysdate);
insert into test_null (a, b, c1, d) values ('A',
                               NULL, 123, sysdate);
insert into test_null (a, b, c1, d) values ('A',
                               'A', NULL, sysdate);
insert into test_null (a, b, c1, d) values ('A',
                                   'A', 123, NULL);
insert into test_null (a, b, c1, d) values
                        (NULL, NULL, NULL, NULL);
```

De inhoud is dan:

Rij nummer	A	B	C1	C2	D
1	'A'	'A'	123	123	NULL
2	NULL	'A'	123	NULL	sysdate
3	'A'	NULL	123	NULL	sysdate
4	'A'	'A'	NULL	NULL	sysdate
5	'A'	'A'	123	NULL	NULL
6	NULL	NULL	NULL	NULL	NULL

Er is te zien dat er twee mogelijkheden zijn om een kolom de waarde NULL te geven:

- Expliciet de waarde NULL opnemen in de value clause van het insert statement;
- De kolom (C2 hiervoor) geheel weglaten uit het insert statement.

Als voor een kolom een default waarde gedefinieerd is als in:

```
create table klant (..
land_code     char(6) default 'NL',
status_code   not NULL char(2) default 'XX');
```

dan zal in het tweede geval (kolom expliciet weglaten in het insert statement) de default-waarde worden toegekend, maar in het eerste geval krijgt de kolom toch de NULL-waarde (indien toegestaan). De default-waarde kan ook worden toegekend met het keyword DEFAULT als in:

```
insert into klant (.., land_code, status_code)
values (.., NULL, DEFAULT);
```

De where clause

De where clause in select, update of delete statements bevat een aantal condities (zie ook [1]). Voor elke conditie geldt dat deze onwaar (feitelijk 'onbekend') is als links of rechts van de operator een NULL staat. Dus de voorwaarde "Where A = B" in de voorgaande tabel is alleen waar voor de rijen 1, 4 en 5.
Dit is logisch: als de waarde van B onbekend is, dan is de uitkomst van de conditie "is A gelijk aan B" ook onbekend en wordt door het RDBMS dus op onwaar gezet. Ook voor rij 6 is deze conditie onwaar/onbekend want NULL is niet gelijk aan NULL!

De condities `where c1 is NULL` of `where c1 is not NULL` testen expliciet op het gevuld of onbekend zijn van de kolom C1.

De conditie `where c1 = NULL`
geeft geen rijen terug, omdat rechts van de operator een onbekende waarde staat (NULL).
Deze conditie is dus een andere dan de conditie: `where c1 is NULL`

De conditie `where c1 > 100 or c1 <= 100`
lijkt altijd waar, maar is alleen waar voor de rijen 1, 2, 3 en 5 en niet voor rij 4 en 6.

De conditie `where c1 > 100 or c1 <= 100 or c1 is NULL` is wel waar voor alle rijen.

De conditie `where a in ('A', 'B', NULL)`
geeft alleen de rijen waar kolom A de waarde 'A' of 'B' heeft en niet de waarde NULL.
De conditie `where a not in ('A', 'B', NULL)`
geeft geen rijen terug!
Dit geldt zowel als de waarden zijn uitgeschreven, maar ook als deze het resultaat van een subselect zijn als in:
`where a not in (subselect..)`
De reden is dat een "IN" wordt omgevormd in (of behandeld als) een serie "OR" condities, de "NOT IN" in een serie "AND NOT" condities. 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.

Outerjoin

In de volgende query willen we de gegevens van de klant met de naam van het bijbehorende land opvragen. Als in de tabel Klant de landcode niet verplicht is zal de volgende query de rijen, waarvan de land_code NULL is echter niet selecteren.

```
select  klant.*, land.naam
from    klant, land
where   land.land_code = klant.land_code;
```

Om deze rijen wel te selecteren is een 'outer join' nodig:
De Oracle syntax:

```
select  klant.*, land.naam
from    klant, land
where   land.land_code(+) = klant.land_code;
```

De Sybase en SQL server syntax:

```
select  klant.*, land.naam
from    klant, land
where   land.land_code =* klant.land_code
```

De nieuwe ANSI standaard-syntax (werkt in alle genoemde producten vanaf een zekere versie):

```
select  klant.*, land.naam
from    klant
left outer join land on (land.land_code =
                                    klant.land_code);
```

Bij deze selectie worden de gevraagde kolommen van de tabel Land op NULL gezet als de land_code van de klant NULL is en er dus geen gerelateerd record in de landentabel staat.

Groepsfuncties op NULL waarden

Groepsfuncties zoals count en sum op een kolom verwerken alleen de niet NULL-waarden:

```
select count(*) from test_null
```

Het resultaat is 6, het aantal rijen in de tabel, ook als alle kolommen NULL zijn.

```
select count(a) from test_null
```

Het resultaat is 4, omdat er vier rijen zijn waarvoor de kolom A een waarde ongelijk aan NULL heeft.

NULL volgens Codd en Date

Het gebruik en de werking van NULL is gebaseerd op de derde regel van Codd: De systematische behandeling van NULL waarden. Zie [1] en [2]. (En zie ook het artikel van Frido van Orden op pagina 32).
Date en Darwen zijn het op dit punt geheel met Codd oneens (zie [3] en [4]). Volgens hen mag een kolom nooit NULL zijn. Dit is onder meer behandeld in het seminar dat Date in Februari gegeven heeft in Amsterdam. Wat is zijn motivering hiervoor en wat zijn de alternatieven?
De motivering voor het niet gebruiken van het begrip en de implementatie van NULL is dat de betekenis van NULL onduidelijk is (is het onbekend, niet van toepassing, gewoon niet ingevuld, etcetera). 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.

Alternatieven

Het eerste alternatief is, zoals in dit artikel ook al aangegeven is, (zie het voorbeeld aan het begin) het maken van subtypes voor bijvoorbeeld vrouwen en gehuwden, waardoor attributen als huwelijksdatum alleen voorkomen indien ze van toepassing zijn. Een verdere uitwerking komt neer op het maken van een separate tabel voor elk optioneel attribuut in combinatie met de primary key van de eerste tabel, of mogelijk zelfs een aparte tabel voor elke reden dat een attribuut kan ontbreken. Dit is uitgewerkt in [3].
Een derde alternatief is het gebruiken van speciale waarden als de waarde onbekend is (9999-12-31 voor een onbekende einddatum, lege string of allemaal spaties voor een onbekende naam en zero of 9999 voor onbekende bedragen, etcetera). In de voorbeelden [zie 3] geeft Darwen ook de tekst "geen salaris" bij een salaris. In een numerieke kolom leidt dit tot zowel numerieke als niet-numerieke waarden met (volgens mij veel) nadelige consequenties voor de controles en bewerkingen op deze kolommen.
Als vierde alternatief kan men denken aan een extra attribuut voor elk optioneel attribuut: een vlag die aangeeft of het andere veld van toepassing is of zelfs waarom het niet ingevuld is (onbekend, niet van toepassing, etcetera).

Conclusie

Naar onze mening zijn de alternatieven 1 en 2 onpraktisch, zowel logisch (een onnodig complex gegevensmodel) als fysiek (een onpraktische implementatie met zeer veel outer joins als de gegevens worden opgehaald). Ook geven deze outerjoins weer NULL's die volgens Date vervangen moeten worden door een andere waarde (zie het derde alternatief hiervoor).
Alternatieven 3 en 4 zijn onpraktisch bij bewerkingen, omdat hier in SQL zelf de uitzonderingen geprogrammeerd moeten worden die juist met het begrip NULL op een praktische wijze zijn geïmplementeerd.

Referenties

1. F. van Orden, Een actueel in memoriam, kader: "Codd's regels", Database Magazine 2003/8.
2. Obituary: Dr. Edgar F. Codd, www.nao.gov.uk/intosai/edp/intoit_articles/18p60top62.pdf.
3. Darwen: How to Handle Missing Information without Using Nulls, www.hughdarwen.freeola.com/TheThirdManifesto.web/Missing-info-without-nulls.pdf
4. C.J. Date, Seminar "Relational Remodeled", georganiseerd door Array Publications B.V. Februari 2004.


```
select sum(c1) from test_null
```

Het resultaat is 492, de NULL waarden doen niet mee in de optelling.

```
select sum(c1), sum(c2), sum(c1)+sum(c2),
                              sum(c1+c2) from test_null
```

Het resultaat is 492, 123, 615 en 246! Let op het verschil tussen de laatste twee waarden.
De laatste som heeft immers alleen een waarde voor de eerste rij.

```
select avg(c1) from test_null
```

Het resultaat is 123, de NULL-waarden doen niet mee in de berekening. Dit is redelijk als we kijken naar bijvoorbeeld de gemiddelde temperatuur. De gemiste metingen worden niet meegenomen in de berekening van het gemiddelde.

Andere functies op NULL waarden

Berekeningen, waarbij een van de componenten NULL is, hebben als resultaat ook altijd NULL. Immers 10 keer "onbekend" is ook onbekend.
Hetzelfde geldt voor numerieke functies zoals sinus- of afrondingsfuncties:

```
select sin(c1), floor(c1/10) from test_null;
```

In condities zoals: `where c1 + c2 > 100`
zullen alleen de rijen geselecteerd worden waarbij beide kolommen C1 en C2 niet NULL (leeg/onbekend) zijn en tevens de som groter is dan 100.
Ook voor berekeningen met datums (waarvan de syntax sterk per product verschilt) geldt dat als een van de variabelen "onbekend" is, het resultaat ook onbekend is:
In Oracle: `select sysdate - factuur_datum`
In Sybase en en SQL server:

```
select datediff(day, getdate(), factuur_datum)
```

Beide geven het verschil in dagen tussen systeemdatum en de factuurdatum of NULL als de factuur_datum onbekend is (zie ook [2]).

De lengte van een character-kolom met de waarde NULL is ook onbekend (dus NULL) en niet 0 (zero).
Oracle:

```
select length(A), length(B) from test_null;
```

SQL server:

```
select len(A), len(B) from test_null
```

Sybase:

```
select char_length(A), char_length(B) from test_null
```

Hier is te zien dat de lengte NULL is als A of B de waarde NULL heeft. Als kolom A (type char(10)) wel is ingevuld, dan is de lengte:
1. In Oracle en Sybase gelijk aan de vaste lengte van de kolom (hier char(10)) dus altijd 10, onafhankelijk van de ingevoerde waarde;
2. In SQL Server gelijk aan de opgegeven waarde, dus hier 1.

De lengte van kolom B (varchar(10)) is in deze producten de werkelijke lengte van de ingevulde string, dus hier altijd 1. Maar ook de behandeling van trailing spaces in de lengte en bij het concateneren, verschilt per product en zelfs per versie van een product.

Soms is het handig om de NULL-waarde te vervangen door een andere waarde, bijvoorbeeld als bij het berekenen van een totaalbedrag van een factuur een aantal componenten bij elkaar wordt opgeteld. Als een van deze componenten (bijvoorbeeld de boete) NULL kan zijn, moet deze voor de optelling door 0 worden vervangen.
In Oracle:

```
select sub_totaal + nvl(boete, 0) from factuur;
```

In Sybase of SQL server:

```
select sub_totaal + isnull(boete, 0) from factuur;
```

Ook bij character-kolommen kan het handig zijn om een lege waarde te vervangen door een duidelijke tekst, bijvoorbeeld:

```
select naam, nvl(email, 'geen mail') from klant;
```

Oracle (9i) heeft nog enkele andere functies die specifiek NULL-waarden vertalen:

Functie	Voorbeeld en toelichting	Resultaat
NVL	Select NVL(boete, 0)	De boete of 0
	Let op, de waarden moeten van hetzelfde type zijn!	
NVL2	Select NVL2(A, B, 'niks') from test_null	B of niks
	Als A is NOT NULL dan B, anders de constante 'niks'	
NULLIF	Select NULLIF(A, B) from test_null	NULL of A
	Als A = B dan NULL, anders A	
COALESCE	Select coalesce(A, B, 'niks') from test_null	A of B of 'niks'
	Selecteert de eerste niet NULL waarde	

De NULLIF- en COALESCE-functies zijn ook beschikbaar in SQL Server maar Sybase ASE kent deze beide functies niet.
De functionaliteit van de coalesce kan dan wel worden verkregen door herhaald de isnull-functie uit te voeren:

```
select isnull(isnull(A, B), 'niks')
```

Verder geeft het CASE statement veel mogelijkheden om NULL-waarden te vertalen of te vergelijken.

Bij het aan elkaar plakken van character-kolommen (Concatenation) wordt NULL in Oracle beschouwd als een string van 0 posities. Het resultaat is weer een string en niet NULL!
Bijvoorbeeld:

```
select voornaam || tussenvoegsel || achternaam from
                                                klant;
```

Ook als het tussenvoegsel leeg (NULL) is, komt hier een resultaat (voornaam plus achternaam). In SQL Server is het resultaat wel NULL als een van de delen NULL is:

```
select voornaam + isnull(tussenvoegsel, '') +
                                  achternaam from klant
```

Hier moeten we dus het tussenvoegsel expliciet door een lege string vervangen om nog een naam over te houden. Ook hier is te zien dat de syntax en werking per product kan verschillen. Bij deze query vergeten we even de spaties tussen de delen van de naam.

Conclusie en samenvatting

In relationele systemen zien we weinig onjuist gebruik van NULL. In niet-relationele systemen (C-ISAM en txt files) komt dit nog regelmatig voor door het niet opnemen van een indicatie of het betreffende veld wel of niet een relevante waarde bevat.
Ondanks alle pogingen tot standaardisatie is op deze punten de syntax tussen de RDBMS-producten nog verschillend. Dit geldt met name voor de functies en andere bewerkingen zoals concatenatie van tekstvelden. 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.

Een en ander is in dit artikel uitgewerkt voor Oracle en Microsoft SQL Server. Sybase (Application Server Enterprice, ASE, niet zijn kleine broertje ASA) is vanwege de gelijke oorsprong meestal gelijk aan SQL Server (maar niet altijd). Bij een ander (R)DBMS (DB2, Informix, Ingres, MySQL, Sybase ASA, etcetera) is het aan te bevelen om, aan de hand van de voorbeelden in dit artikel, de werking voor de eigen situatie te verifiëren.

Literatuur

1. *Loonen, Het schrijven van een nette WHERE clause. Database Magazine 1997/7.*
2. *Loonen, Datum en tijd in het logisch en fysiek gegevensmodel. Database Magazine 2001/6.*

Toon Loonen (toon.loonen@capgemini.com) en **Nicolette Verdugt** (nicolette.verdugt@capgemini.com) zijn werkzaam bij Capgemini.