Keunikan Nilai Null Dalam Database Relasional

Djoni Darmawikarta
djoni_darmawikarta@yahoo.ca

Database relasional (relational database) seperti misalnya Oracle, DB2 (IBM), SQL Server (Microsoft), dan MySQL, memiliki fasilitas untuk menyimpan data bernilai null. Null disini bukan nol (angka nol) dan juga bukan space (untuk jenis data alpha, alphanumeric, dan string), melainkan mewakili nilai data "tidak diketahui" (unknown) atau "tidak menentu" (indeterminate) atau kosong (absence of value)

Meskipun implementasi nilai null didalam database relasional, seperti beberapa yang disebutkan diatas, tidak sama percis satu dengan lainnya, kesemuanya mengacu pada hukum Codd ke-3 (sebagai persyaratan memenuhi kriteria database relasional) yang sudah menjadi standard ANSI SQL-92.

Null memiliki keunikan-keunikan yang perlu dipahami untuk menghindari kesalahan pemakaiannya.

Selanjutnya akan dibahas: Kebutuhan nilai null, keunikan sifat dan fasilitas, diakhiri dengan saran pemakaiannya Agar mudah disimak dan praktis, contoh-contoh akan menyertai pembahasan dan mengacu pada implementasi Oracle.

Kebutuhan Nilai Null

Berikut tiga contoh yang menunjukkan kebutuhan nilai null.

Dalam aplikasi pekerjaan terkadang data *kode pos* tidak (belum) diberikan oleh pelamar (dalam formulir isian masih kosong). Bila kode pos tidak kritis, maka data yang sudah ada dimasukkan kedalam komputer agar aplikasi bersangkutan bisa diproses.

Memasukkan nilai space kurang benar, karena ini berarti (bisa diartikan) pelamar tidak memiliki kode pos (terutama bila diolah komputer, space merupakan nilai data bersangkutan), padahal yang benar adalah bahwa dia memilikinya tetapi belum memberikannya. Dengan kata lain, dari sudut kita (pemroses data) kode pos bernilai belum diketahui (unknown).

Melanjutkan contoh aplikasi pekerjaan diatas. Andaikan data *gaji yang diinginkan* juga kosong, tetapi profil pelamar sangat menarik, dan diputuskan untuk diproses, maka bila diisi angka nol tentu saja sangat salah, karena sudah pasti pelamar tidak mau tidak digaji! Data numerik seharusnya tidak berisi nol memberikan dampak negatif lain. Misalnya kita perlu menghitung angka rata-rata gaji yang dinginkan dari semua pelamar, untuk menentukan besarnya gaji yang akan ditawarkan. Kalau menyertakan yang masih berisi nol, hasil perhitungan rata-rata jelas akan salah.

Kedua masalah diatas dapat diatasi dengan fasilitas null seperti akan diuraikan dibagian berikut (Keunikan dan Fasilitas pemakaian)

Contoh ketiga, misalkan *bonus* merupakan bagian dari data karyawan tetapi hanya berlaku untuk salesman, maka untuk karyawan non-salesman data *bonus*-nya haruslah null, bukan nol, karena nol berarti berhak mendapat bonus (bonusnya nol mungkin karena prestasi penjualannya sangat jelek sehingga besar bonusnya nol)

Ketiga contoh diatas merupakan kasus yang harus ditangani waktu merancang database. Dua contoh berikut semasa pemeliharaan (maintenance, dimana database sudah berisi data)

Setelah terisi, diketahui bahwa sejumlah data yang baru selesai dimasukkan ternyata mengandung banyak kesalahan, dan diputuskan untuk untuk mengulang pengisiannya. Cara terbenar dan termudah adalah mengosongkan sekaligus (di-null-kan) semua yang salah terlebih dahulu, baru diisi kembali dengan data yang betul. Kalau caranya dengan menimpa yang salah (update in place), maka bila keseluruhan data dipakai termasuk yang belum sempat dikoreksi, hasilnya akan salah.

Contoh terakhir sebagai berikut. Diperlukan tambahan data baru, misalnya *jumlah-anggota-keluarga*. Pada saat ditambahkan ke tabel data *jumlah-anggota-keluarga* ini, yang benar, haruslah diisi nilai null; baru setelah itu diisi dengan nilai sesungguhnya.

Dapat disimpulkan, bahwa dalam kenyataan penggunaan database memang ada kebutuhan nilai null.

Keunikan dan Fasilitas pemakaian

Agar data didalam tabel bisa bernilai null jangan dibatasi dengan NOT NULL. Pada contoh pembuatan tabel *pelamar* berikut, kolom *kode_pos* dan *gaji_diinginkan* boleh null.

```
CREATE TABLE pelamar
     (nama_pelamar  VARCHAR(20)  NOT NULL
     , alamat_jalan      VARCHAR(30) NOT NULL
     , kode_pos     VARCHAR(6)
     , gaji_diinginkan   NUMERIC(10));
```

Pada waktu mengisi data, bila diinginkan isi kedua kolom tersebut null, dapat digunakan misalnya SQL statement sebagai berikut:

```
INSERT INTO pelamar (nama_pelamar, alamat_jalan)
     VALUES ('Noah Jambu', '124 Burbank Dr');
```

Atau:

```
INSERT INTO pelamar (nama_pelamar, alamat_jalan, kode_pos)
     VALUES ('Noah Jambu', '124 Burbank Dr', '');
```

Kedua statement ini akan menghasilkan data yang sama didalam tabel *pelamar*, karena didalam statement yang terakhir, nilai *kode_pos* yang berupa string panjangnya nol (bukan space) diberi nilai null.

nama_pelamar	***alamat_jalan***	***kode_pos***	***gaji_diinginkan***
Noah Jambu	124 Burbank Dr		

Andaikan tabel *pelamar* sudah kita isi dengan data sebagai berikut.

nama_pelamar	***alamat_jalan***	***kode_pos***	***gaji_diinginkan***
Noah Jambu	124 Burbank Dr		
Alan Glasgow	69 Aussie Rd	4113	50000
Kian Lee	Pekojan Tengah 17		
Igor Kinosky	Unit 24 Red Square	X0Z8Y9	100000

Untuk membaca semua data yang *kode-pos*nya null, kita gunakan IS NULL, sebagai berikut:

```
SELECT * FROM pelamar WHERE kode_pos IS NULL;
```

Hasilnya:

nama_pelamar	***alamat_jalan***	***kode_pos***	***gaji_diinginkan***
Noah Jambu	**124 Burbank Dr**		
Kian Lee	**17 Pekojan Tengah**		

Bila kita gunakan:

```
SELECT * FROM pelamar WHERE kode_pos = NULL;
```

hasilnya akan tidak seperti kita inginkan – tidak ditemukan apa-apa.

Ini disebabkan null nilainya tidak menentu (indeterminate), karenanya pembandingan null dengan null pun hasilnya tidak menentu (akibatnya, dalam contoh diatas, data yang dicari tidak ditemukan) Sekali lagi, gunakan fasilitas IS NULL untuk mencari (membandingkan dengan) nilai null.

Ketidaktentuan nilai null ini juga ditunjukkan dari hasil program berikut:

```
/* membandingkan a dan b yang keduanya bernilai null */
/* variabel yang dideklarasikan dan tidak diberi nilai akan bernilai null
/* (diberi nilai awal null oleh oracle) */
DECLARE
a INTEGER;
b INTEGER;
BEGIN
     IF a = b THEN
     DBMS_OUTPUT.PUT_LINE('a bernilai = b');
     ELSE
     DBMS_OUTPUT.PUT_LINE('a bernilai <> b');
     END IF;
END;
```

Hasil yang ditampilkan:

```
a bernilai <> b
```

Operasi pembanding yang lain, seperti > (lebih besar), <= (lebih kecil atau sama) dan <> (tidak sama), bila melibatkan null, hasilnya juga null (tidak menentu)

Operasi perhitungan (arithmetic) yang melibatkan null menghasilkan null. Misalnya untuk perkalian sebagai berikut:

```
SELECT nama_pelamar, (gaji_diinginkan * 1.10)
gaji_ditawarkan FROM pelamar;
```

Hasilnya:

nama_pelamar	*gaji_ditawarkan*
Noah Jambu	
Alan Glasgow	**55000**
Kian Lee	
Igor Kinosky	**110000**

Contoh fungsi aritmetik: rata-rata.

```
SELECT AVG(gaji_diinginkan) rata_rata_gaji_diinginkan FROM
pelamar;
```

Hasilnya:

rata_rata_gaji_diinginkan
75000

Hasil 75000 ini diperoleh dari perhitungan (50000 + 100000)/2. Jadi, hanya 2 data diperhitungkan (yang tidak null) , tidak 4 (null tidak diperhitungkan)

Contoh operasi string (alphanumeric): penggabungan.

```
SELECT nama_pelamar, (alamat_jalan ||  ‘ ‘ || kode_pos)
alamat_lengkap FROM pelamar;
```

Hasilnya:

alamat_lengkap
124 Burbank Dr
69 Aussie Rd 4113
Pekojan Tengah 17
Unit 24 Red Square X0Z8Y9

Terlihat dalam penggabungan diatas, bahwa nilai null adalah hampa (string yang panjangnya nol)

Hukum operasi relasional yang berkenaan dengan null ditunjukkan dalam daftar berikut.

Operator relasional	***Operasi***	***Hasil***
AND	true AND null	null
	false AND null	false
	null AND null	null
OR	true OR null	true
	false OR null	null
	null OR null	null
NOT	NOT null	null

Program berikut membuktikan operasi relasional OR diatas.

```
BEGIN

IF (true OR null) = true THEN
  DBMS_OUTPUT.PUT_LINE('true OR null is true'); /*menurut
daftar diatas inilah yang benar – ditampilkan */
ELSE
  IF ((true OR null) = false) THEN
     DBMS_OUTPUT.PUT_LINE('true OR null is not false');
  ELSE
```

```
    IF ((true OR null) IS NULL) THEN
      DBMS_OUTPUT.PUT_LINE('true OR null is not null');
    ELSE
    NULL;
    END IF;
  END IF;
END IF;

IF (false OR null) IS NULL THEN
  DBMS_OUTPUT.PUT_LINE('null'); /*menurut daftar diatas
inilah yang benar – ditampilkan */
ELSE
  IF ((false OR null) = true) THEN
     DBMS_OUTPUT.PUT_LINE('true');
  ELSE
    IF ((false OR null) = false) THEN
      DBMS_OUTPUT.PUT_LINE('false');
    ELSE
    NULL;
    END IF;
  END IF;
END IF;

IF (null OR null) IS NULL THEN
  DBMS_OUTPUT.PUT_LINE('null OR null is null'); /*menurut
daftar diatas inilah yang benar – ditampilkan */
ELSE
  IF ((null OR null) = true) THEN
     DBMS_OUTPUT.PUT_LINE('null OR null is not true');
  ELSE
    IF ((null OR null) = false) THEN
      DBMS_OUTPUT.PUT_LINE('null OR null is not false');
    ELSE
    NULL;
    END IF;
  END IF;
END IF;

END;
```

Hasil yang ditampilkan:

```
true OR null is true
false OR null is null
null OR null is null
```

Oracle menyediakan fasilitas berkenaan dengan null, selain IS NULL yang sudah dibahas diatas, misalnya fungsi NVL(parameter1, parameter2). Fungsi NVL ini

menghasilkan (returning) parameter kedua bila bonus bernilai null, bila tidak null maka nilai bonus yang dihasilkan.

Berikut contohnya.

/* bila nilai *kode_pos* null, tampilkan 'kode pos menyusul' untuk visualisasi nilai null-nya */

```
DECLARE
tampilan_kode_pos VARCHAR(20);
BEGIN
SELECT kode_pos INTO tampilan_kode_pos FROM pelamar
     WHERE nama_pelamar = 'Noah Jambu';
DBMS_OUTPUT.PUT_LINE(NVL(tampilan_kode_pos, 'kode pos
menyusul'));
END;
```

Hasil yang ditampilkan:

```
kode pos menyusul
```

Saran Pemakaian Null

Manfaatkan null pada tempatnya. Karena keunikan null (lain dari nilai data jenis yang normal), bila tidak betul-betul dibutuhkan, lebih baik dihindari. Pahami cara pemakaiannya, termasuk implementasi spesifik di database yang digunakan. Dan, jangan lupa mendidik pemakai data, terutama end-user (business user yang tidak memiliki keakhlian teknis memadai)