

# BAB 11 Menguasai WITH ROLLUP

Pada bab UNION kita telah belajar bagaimana membuat baris total dan subtotal, pada MySQL terdapat fitur khusus untuk menangani hal tersebut yaitu dengan klausa WITH ROLLUP, apakah klausa ini lebih powerful dibanding UNION? Kita bahas tuntas di bab ini.

## 11.1. Memahami Cara Kerja WITH ROLLUP

WITH ROLLUP bekerja dengan menjumlahkan nilai kolom yang terdapat fungsi agregasinya, sehingga jika ingin kolom tersebut muncul baris totalnya, maka kolom tersebut **harus diberi fungsi agregasi**

Selanjutnya, selain fungsi agregasi, untuk dapat menggunakan klausa WITH ROLLUP, kita harus menggunakan klausa GROUP BY, klausa ini harus diletakkan **tepat sebelum** klausa WITH ROLLUP.

Selain itu, ketika menggunakan klausa ini, kita **tidak dapat** menggunakan klausa ORDER BY, namun dapat menggunakan klausa lain seperti HAVING dan LIMIT

Untuk lebih jelasnya, misal kita miliki tabel penjualan dengan data sebagai berikut:

id_trx	id_pelanggan	tgl_trx	total_trx
1	1	2017-03-02	192000
2	1	2017-03-10	186000
3	0	2017-04-10	259000
4	2	2017-04-05	110000
5	2	2016-11-10	256000

Selanjutnya kita tampilkan data penjualan tersebut dengan tambahan baris total di bagian akhir baris, jalankan query berikut:

```
1. SELECT id_trx, id_pelanggan, tgl_trx, SUM(total_trx)
2. FROM penjualan
3. GROUP BY id_trx
4. WITH ROLLUP
```


Hasil:

id_trx	id_pelanggan	tgl_trx	SUM(total_trx)
1	1	2017-03-02	192000
2	1	2017-03-10	186000
3	0	2017-04-10	259000
4	2	2017-04-05	110000
5	2	2016-11-10	256000
NULL	2	2016-11-10	1003000

Perhatikan bahwa pada query diatas, dengan GROUP BY kita kelompokkan data berdasar kolom id\_trx meskipun riilnya tidak ada data yang dikelompokkan, karena tidak ada id\_trx yang sama, hal ini hanya untuk "mengakali" saja agar kita dapat menggunakan WITH ROLLUP

Agak maksa? Ya memang harus seperti itu.....

Selain itu perhatikan bahwa kita juga menggunakan fungsi agregasi SUM pada kolom total\_trx, sehingga pada baris total kita dapatkan jumlah total semua transaksi.



Ciri khas baris hasil WITH ROLLUP ini adalah adanya nilai NULL pada kolom yang ada pada klausa GROUP BY (pada contoh diatas kolom id\_trx pada baris total), sedangkan nilai pada kolom lain (yang tidak ada fungsi agregasinya) biasanya sama dengan nilai pada baris sebelumnya

Selanjutnya mari kita lanjutkan pembahasan dengan membuat baris subtotal, misal kita kelompokkan data berdasarkan bulan transaksi dan id\_pelanggan serta kita tampilkan data nama pelanggan.

Adapun data tabel pelanggan adalah sebagai berikut:

id_pelanggan	nama	alamat	id_staf
1	Alfa	Jakarta	1
2	Beta	Semarang	1
3	Charlie	Surabaya	2
4	Delta	Surakarta	3

Selanjutnya jalankan query berikut:

```

1. SELECT MONTH(tgl_trx) AS bln_trx
2.     , id_pelanggan
3.     , nama
4.     , COUNT(id_trx) AS jml_trx
5.     , SUM(total_trx) AS nilai_trx
6. FROM penjualan
7.   LEFT JOIN pelanggan USING (id_pelanggan)
8. GROUP BY bln_trx, id_pelanggan
9. WITH ROLLUP

```

Hasil yang kita peroleh adalah:

bln_trx	id_pelanggan	nama	jml_trx	nilai_trx
3	1	Alfa	2	378000
3	NULL	Alfa	2	378000
4	0	NULL	1	259000
4	2	Beta	1	110000
4	NULL	Beta	2	369000
11	2	Beta	1	256000
11	NULL	Beta	1	256000
NULL	NULL	Beta	5	1003000

Perhatikan bahwa pada contoh diatas, terdapat beberapa baris tambahan yang dihasilkan dari klausa **WITH ROLLUP** (baris yang mengandung nilai **NULL**), yaitu pada kolom bln\_trx dan id\_pelanggan.

Perhatikan juga bahwa **WITH ROLLUP** mengkalkulasi nilai pada kolom yang terdapat fungsi **COUNT()** dan **SUM()** yaitu kolom jml\_trx dan nilai\_trx

Karena pada klausa **GROUP BY** kita mengelompokkan dataurut berdasarkan bulan, baru kemudian id\_pelanggan, maka ketika MySQL membuat total untuk kelompok tersebut, MySQL akan memberikan nilai **NULL** pada kolom id\_pelanggan (untuk subtotal setiap bulan) dan id\_pelanggan + bulan untuk total semua data, perhatikan gambar berikut:

2

GROUP BY bln\_trx, id\_pelanggan

1

bln_trx	id_pelanggan	nama	jml_trx	nilai_trx
3	1	Alfa	2	378000
3	NULL	Alfa	2	378000
4	0	NULL	1	259000
4	2	Beta	1	110000
4	NULL	Beta	2	369000
11	2	Beta	1	256000
11	NULL	Beta	1	256000
NULL	NULL	Beta	5	1003000

Gambar 11.1 Ilustrsi WITH ROLLUP Pada Beberapa Kolom

Sampai disini sudah paham kan? Bagaimana klausa WITH ROLLUP bekerja? Jika sudah, mari kita lanjutkan...

## 11.2. Mengganti Nilai NULL

Pada bagian sebelumnya terlihat bahwa dengan WITH ROLLUP nilai kolom yang ada pada klausa GROUP BY akan bernilai NULL. Agar lebih memiliki arti, kita perlu menggantinya dengan kata lain, seperti TOTAL atau SUB TOTAL

Melanjutkan contoh sebelumnya, kita kelompokkan data penjualan berdasarkan bulan dan id\_pelanggan dan kita ganti nilai NULL dengan string TOTAL dan SUB TOTAL

Jalankan query berikut:

```
1. SELECT MONTH(tgl_trx) AS bln_trx
2.     , id_pelanggan
3.     , nama
4.     , COUNT(id_trx) AS jml_trx
5.     , SUM(total_trx) AS nilai_trx
6. FROM penjualan
7.   LEFT JOIN pelanggan USING (id_pelanggan)
8. GROUP BY bln_trx, id_pelanggan
9. WITH ROLLUP
```

Hasilnya adalah:

bln_trx	id	nama	jml_trx	nilai_trx
11	2	Beta	1	256000
11	SUB TOTAL	Beta	1	256000
3	1	Alfa	2	378000
3	SUB TOTAL	Alfa	2	378000
4	0	NULL	1	259000
4	2	Beta	1	110000
4	SUB TOTAL	Beta	2	369000
NULL	SUB TOTAL	Beta	5	1003000

Pada contoh diatas, kita berhasil mengubah nilai NULL pada kolom id menjadi SUB TOTAL, namun kenapa nilai pada bln\_trx tetap NULL?

Jawabnya, memang untuk fungsi tertentu, khususnya fungsi terkait date time, kita tidak dapat mengubah NULL secara langsung seketika saat fungsi dijalankan, sebagai solusinya, kita dapat menggunakan subquery sebagai berikut:

```
1. SELECT IFNULL(bln_trx, "TOTAL") AS bulan
2.     , id
3.     , nama
4.     , jml_trx
5.     , nilai_trx
6. FROM
7.     (SELECT MONTH(tgl_trx) AS bln_trx
8.         , IFNULL(id_pelanggan, "TOTAL") AS id, nama
9.         , COUNT(id_trx) AS jml_trx
10.        , SUM(total_trx) AS nilai_trx
11.     FROM penjualan
12.     LEFT JOIN pelanggan USING (id_pelanggan)
13.     GROUP BY bln_trx, id_pelanggan
14.     WITH ROLLUP
15.    ) AS penjualan
```

Hasil:

bulan	id	nama	jml_trx	nilai_trx
3	1	Alfa	2	378000
3	TOTAL	Alfa	2	378000
4	0	NULL	1	259000
4	2	Beta	1	110000
4	TOTAL	Beta	2	369000
11	2	Beta	1	256000
11	TOTAL	Beta	1	256000
TOTAL	TOTAL	Beta	5	1003000

Kenapa subquery? Ingat kembali prinsip bahwa jika kita ingin mengolah tabel hasil query, maka kita **harus** menggunakan subquery, kenapa? karena pada subquery, temporary tabel sudah terbentuk sehingga tabel tersebut dapat diolah layaknya tabel riil.

## 11.3. ORDER BY Pada WITH ROLLUP

Seperti disampaikan diawal, bahwa ketika menggunakan WITH ROLLUP kita **mutlak tidak dapat** menggunakan klausa ORDER BY, sehingga jika kita ingin mengurutkan data (termasuk baris baru hasil WITH ROLLUP) kita harus menggunakan subquery.

Misal melanjutkan contoh sebelumnya, kita urutkan data berdasarkan bulan secara descending, hasil yang kita harapkan adalah sebagai berikut:

bln_trx	id	nama	jml_trx	nilai_trx
4	0	NULL	1	259000
4	2	Beta	1	110000
4	SUB TOTAL	Beta	2	369000
3	1	Alfa	2	378000
3	SUB TOTAL	Alfa	2	378000
NULL	SUB TOTAL	Alfa	4	747000

Bagaimana querynya?

Untuk memperoleh hasil seperti tabel diatas, kita tinggal tambahkan DESC pada ORDER BY sebagai berikut:

```
1. SELECT MONTH(tgl_trx) AS bln_trx
2.     , IFNULL(id_pelanggan, "SUB TOTAL") AS id, nama
3.     , COUNT(id_trx) AS jml_trx
4.     , SUM(total_trx) AS nilai_trx
5. FROM penjualan
6.   LEFT JOIN pelanggan USING (id_pelanggan)
7. WHERE YEAR(tgl_trx) = 2017
8. GROUP BY bln_trx DESC, id_pelanggan
9. WITH ROLLUP
```

Perhatikan bahwa pada baris nomor 8 kita tambahkan **DESC** setelah kolom bln\_trx, simpel kan?

Kenapa bisa seperti itu?

Seperti yang telah kita bahas pada BAB I, ketika MySQL menjalankan GROUP BY, maka otomatis MySQL akan mengurutkan data berdasarkan data kolom yang ada pada klausa GROUP BY, hal ini disebut implisit order.

Secara default, implisit order berbentuk ascending (dari kecil ke besar), namun demikian kita dapat mengubahnya menjadi descending, yaitu dengan menambahkan DESC pada kolom yang ingin kita ubah urutannya.

## Tetapi...

**Tetapi....** sejak MySQL 5.7 cara ini sudah deprecated yang artinya akan dihilangkan pada versi berikutnya (entah kapan), sehingga jika kita menggunakan teknik ini, maka kode yang kita buat tidak “aman” yang artinya tidak akan berjalan pada MySQL versi terbaru.

Saya pribadi tidak nyaman jika menggunakan fitur yang deprecated, karena kedepannya mau tidak mau kita harus beralih ke versi terbaru, entah karena hardware yang sudah tidak kompatibel, kebijakan management yang berubah, dll sehingga sebisa mungkin hindari fitur deprecated.



Ketika menjalankan klausa GROUP BY, dibelakang layar MySQL akan mengurutkan data kolom yang ada pada klausa group by secara ascending (implisit order), kita dapat mengubah pola pengurutan ini dengan menambahkan DESC pada kolom yang ada pada klausa GROUP BY



Sejak MySQL versi 5.7, fitur ini deprecated, yang artinya akan dihilangkan pada MySQL versi berikutnya, sehingga



disarankan untuk menggunakan eksplisit order dengan menuliskan klausa ORDER BY

Nah, melanjutkan case sebelumnya, bagaimana query yang kita gunakan jika menggunakan **eksplisit order**?

Silakan dicoba, sebagai clue, kita akan gunakan subquery

Sudah bisa?

Baiklah, mari kita cocokkan. Query versi saya adalah sebagai berikut:

---

```
1. SELECT *
2. FROM
3.     (SELECT MONTH(tgl_trx) AS bln_trx
4.        , id_pelanggan AS id, nama
5.        , COUNT(id_trx) AS jml_trx
6.        , SUM(total_trx) AS nilai_trx
7.     FROM penjualan
8.     LEFT JOIN pelanggan USING (id_pelanggan)
9.     WHERE YEAR(tgl_trx) = 2017
10.    GROUP BY bln_trx, id_pelanggan
11.    WITH ROLLUP
12.    ) AS penjualan
13. ORDER BY bln_trx DESC, id DESC
```

---

Hasil yang kita peroleh adalah:

bln_trx	id	nama	jml_trx	nilai_trx
4	2	Beta	1	110000
4	0	NULL	1	259000
4	NULL	Beta	2	369000
3	1	Alfa	2	378000
3	NULL	Alfa	2	378000
NULL	NULL	Beta	4	747000

Penjelasan:

- Karena terdapat WITH ROLLUP maka untuk menggunakan ORDER BY, mau tidak mau kita **harus menggunakan subquery**
- Kenapa kembali ke nilai NULL? bukan kata kata TOTAL atau SUB TOTAL? kita menggunakan nilai NULL **agar mudah mengatur posisi baris null**, karena jika kita urutkan secara ascending, nilai NULL **akan selalu diatas** sedangkan untuk descending **akan selalu dibawah**



Untuk memudahkan penempatan baris TOTAL dan SUBTOTAL, selalu perhatikan nilai NULL karena nilai tersebut akan selalu berada dibawah ketika diurutkan secara descending

Selanjutnya sebagai latihan, ubah nilai NULL tersebut dengan nilai string TOTAL dan SUB TOTAL

Silakan dicoba?

Apakah querynya sama seperti saya ini?

---

```

1. SELECT IFNULL(bln_trx, "TOTAL") AS bulan
2.       , IFNULL(id, "SUB TOTAL") AS id
3.       , nama
4.       , jml_trx
5.       , nilai_trx
6. FROM
7. (
8.     SELECT *
9.     FROM
10.    (SELECT MONTH(tgl_trx) AS bln_trx
11.      , id_pelanggan AS id
12.      , IFNULL(nama, "-") AS nama
13.      , COUNT(id_trx) AS jml_trx
14.      , SUM(total_trx) AS nilai_trx
15.     FROM penjualan
16.     LEFT JOIN pelanggan USING (id_pelanggan)
17.     WHERE YEAR(tgl_trx) = 2017
18.     GROUP BY bln_trx, id_pelanggan

```

---

---

```

19.           WITH ROLLUP
20.       ) AS penjualan
21.       ORDER BY bln_trx DESC, id DESC
22. ) AS penjualan

```

---

Hasil:

bulan	id	nama	jml_trx	nilai_trx
4	2	Beta	1	110000
4	0	-	1	259000
4	SUB TOTAL	Beta	2	369000
3	1	Alfa	2	378000
3	SUB TOTAL	Alfa	2	378000
TOTAL	SUB TOTAL	Beta	4	747000

Penjelasan: kita akan mengolah lagi tabel hasil query, sehingga mau tidak mau kita harus menggunakan subquery

## Real World....

Dalam dunia nyata, pengurutan data bisa sangat kompleks, misal pada tabel hasil query diatas, selain terdapat baris total dan subtotal, data diurutkan lagi berdasarkan nilai\_trx tertinggi

Bisakah dilakukan dengan SQL?

Sayang sekali tidak bisa... Kenapa? Karena SQL hanya bahasa deklarasi (seperti HTML dan CSS) yang hanya ditujukan untuk pengambilan data untuk dapat diolah lebih lanjut oleh aplikasi lain bukan bahasa procedural seperti PHP dan ASP yang memang ditujukan untuk menyelesaikan suatu masalah.

So... jika memang tidak bisa dikerjakan di SQL, tidak perlu dipaksakan....

# 11.4. Studi kasus

Sebagai contoh kasus, kita lanjutkan contoh pada bagian sebelumnya sehingga tabel yang dihasilkan lebih informatif dan siap untuk disajikan, tabel hasil query sebelumnya adalah seperti ini:

bulan	id	nama	jml_trx	nilai_trx
4	2	Beta	1	110000
4	0	-	1	259000
4	SUB TOTAL	Beta	2	369000
3	1	Alfa	2	378000
3	SUB TOTAL	Alfa	2	378000
TOTAL	SUB TOTAL	Beta	4	747000

Nah, kita akan mengubahnya menjadi seperti ini:

nama	bulan	jml_trx	nilai_trx
Beta	4	1	110000
-	4	1	259000
SUB TOTAL	4	2	369000
Alfa	3	2	378000
SUB TOTAL	3	2	378000
TOTAL		4	747000

Bagaimana querynya?

Silakan dicoba, cara yang digunakan sudah dibahas pada bab bab sebelumnya....

Sebagai clue, kita gunakan ekspresi logika CASE

Sudah? Baik, mari kita cocokkan. Query versi saya adalah sebagai berikut:

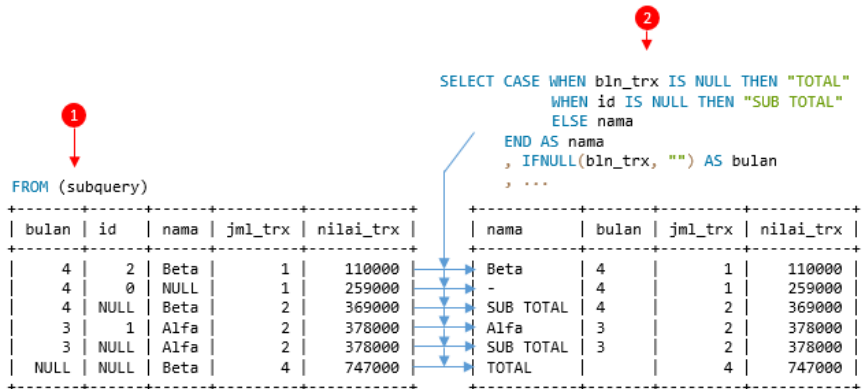
1. `SELECT CASE WHEN bln_trx IS NULL THEN "TOTAL"`
2. `WHEN id IS NULL THEN "SUB TOTAL"`

```

3.         ELSE nama
4.     END AS nama
5.     , IFNULL(bln_trx, "") AS bulan
6.     , jml_trx
7.     , nilai_trx
8. FROM
9. (
10.     SELECT *
11.     FROM
12.     (
13.         SELECT MONTH(tgl_trx) AS bln_trx
14.         , id_pelanggan AS id
15.         , IFNULL(nama, "-") AS nama
16.         , COUNT(id_trx) AS jml_trx
17.         , SUM(total_trx) AS nilai_trx
18.         FROM penjualan
19.         LEFT JOIN pelanggan USING (id_pelanggan)
20.         WHERE YEAR(tgl_trx) = 2017
21.         GROUP BY bln_trx, id_pelanggan
22.         WITH ROLLUP
23.     ) AS penjualan
24. ) AS penjualan

```

Ingat kembali prinsip eksekusi sql, bahwa pada statemen SELECT, setiap baris pada tabel hasil klausa FROM akan di evaluasi sesuai dengan yang ada pada klausa select, perhatikan ilustrasi berikut:



Gambar 11.2 Ilustrasi Eksekusi Klausa SELECT

Pada contoh diatas, karena baris ketiga kolom id bernilai NULL, maka memenuhi kondisi kedua pada ekspresi CASE, sehingga kolom nama bernilai SUB TOTAL, demikian juga dengan kolom lain.