Table of Contents

My SQL has a RollUp phrase that you can use with the Group By clause. The purpose of the roll up is to get grand totals of the result set. Compare the output of the first two queries below. The first groups by the department id and has one row per department. The second adds the roll up and gets an additional row that is the total values for all of the departments. This is called the grand total super aggregate row.

1. Group by WITH ROLLUP

The Group by With RollUp clause is part of MySQL. We will start with a regular group by query.

Demo 01: Do a group by each department with aggregates on salary and emp_id and we get one row for each department.

```
select dept_id, sum(salary), count(salary), count(emp_id)
from   a_emp.adv_emp
group by dept_id
;
+---------+-------------+---------------+---------------+
| dept_id | sum(salary) | count(salary) | count(emp_id) |
+---------+-------------+---------------+---------------+
|      10 |       24000 |             1 |             1 |
|      20 |       15000 |             1 |             1 |
|      30 |      318709 |             8 |             8 |
|      35 |      228000 |             3 |             3 |
|      80 |      151500 |             3 |             3 |
|     210 |       59000 |             2 |             2 |
|     215 |      149254 |             4 |             4 |
+---------+-------------+---------------+---------------+
```

Demo 02: Adding the `WITH ROLLUP` phrase. This gives us one additional row which totals the various groups. The first column has a null to signify that this is a rollup row; there is no valid entry for a department id when the row refers to all the departments.

```
select dept_id, sum(salary), count(salary), count(emp_id)
from   a_emp.adv_emp
group by dept_id WITH ROLLUP
;
+---------+-------------+---------------+---------------+
| dept_id | sum(salary) | count(salary) | count(emp_id) |
+---------+-------------+---------------+---------------+
|      10 |       24000 |             1 |             1 |
|      20 |       15000 |             1 |             1 |
|      30 |      318709 |             8 |             8 |
|      35 |      228000 |             3 |             3 |
|      80 |      151500 |             3 |             3 |
|     210 |       59000 |             2 |             2 |
|     215 |      149254 |             4 |             4 |
|    NULL |      945463 |            22 |            22 |
+---------+-------------+---------------+---------------+
```

Demo 03: We can use coalesce to improve the output. But we do need to be certain that the only null we can have in that column is the null that was added for the Rollup

```
select concat("dept: " , coalesce(dept_id, 'All' ) ) as Dept
 ,     sum(salary)
 ,     count(salary)
 ,     count(emp_id)
from   a_emp.adv_emp
group by dept_id WITH ROLLUP
;
+-----------+-------------+---------------+---------------+
| Dept      | sum(salary) | count(salary) | count(emp_id) |
+-----------+-------------+---------------+---------------+
| dept: 10  |       24000 |             1 |             1 |
| dept: 20  |       15000 |             1 |             1 |
| dept: 30  |      318709 |             8 |             8 |
| dept: 35  |      228000 |             3 |             3 |
| dept: 80  |      151500 |             3 |             3 |
| dept: 210 |       59000 |             2 |             2 |
| dept: 215 |      149254 |             4 |             4 |
| dept: All |      945463 |            22 |            22 |
+-----------+-------------+---------------+---------------+
8 rows in set (0.00 sec)
```

Demo 04: We could have gotten this result with a Union query that did the totals for the table in the second query.

```
select concat("dept: " ,dept_id) as dept
 ,    sum(salary), count(salary), count(emp_id)
from  a_emp.adv_emp
group by dept_id
union
select "dept: All"
 ,     sum(salary), count(salary), count(emp_id)
from a_emp.adv_emp
;
```

It is not unusual to have more than one way to perform a task. Sometimes there is a difference in efficiency of execution; sometime there is a difference in efficiency of maintenance; sometimes it is just a matter of personal preference.

Demo 05: Doing a rollup on the salary attribute- how many employees do we have at each salary level?

```
select salary,  count(emp_id)
from   a_emp.adv_emp
group by salary WITH ROLLUP
;
+--------+---------------+
| salary | count(emp_id) |
+--------+---------------+
|   6500 |             1 |
|   9000 |             1 |
|  12000 |             1 |
|  15000 |             6 |
|  24000 |             1 |
|  30300 |             2 |
|  44450 |             1 |
```

```
|  50000 |             1 |
|  65000 |             3 |
|  80000 |             1 |
|  88954 |             2 |
|  98000 |             1 |
|  98005 |             1 |
|   NULL |            22 |
+--------+---------------+
14 rows in set (0.00 sec)
```

We want a label for that last column- otherwise it looks like we have 22 employees with no salary.

```
select coalesce(salary, 'all') as salaryCol,  count(emp_id)
from   a_emp.adv_emp
group by salary WITH ROLLUP
;
+-----------+---------------+
| salaryCol | count(emp_id) |
+-----------+---------------+
| 6500      |             1 |
| 9000      |             1 |
| 12000     |             1 |
| 15000     |             6 |
| 24000     |             1 |
```

But that made the salary values into strings so they left align. We can use functions to get the alignment we want. Generally we prefer to leave the formatting of the output - such as alignment- to the application level.

```
select coalesce(lpad(salary,12,' '), 'All Salaries') as salaryCol
,  count(emp_id)
from   a_emp.adv_emp
group by salary WITH ROLLUP
;
+--------------+---------------+
| salaryCol    | count(emp_id) |
+--------------+---------------+
|         6500 |             1 |
|         9000 |             1 |
|        12000 |             1 |
|        15000 |             6 |
|        24000 |             1 |
|        30300 |             2 |
|        44450 |             1 |
|        50000 |             1 |
|        65000 |             3 |
|        80000 |             1 |
|        88954 |             2 |
|        98000 |             1 |
|        98005 |             1 |
| All Salaries |            22 |
+--------------+---------------+
```

Now we decide that we really do not need that much detail- after all there is no that much difference between a salary of 98000 and 98005. We decide we would rather have these grouped by 10K salary levels.

Demo 06: Truncate the salary to the 10K level

```
select salary , truncate(salary, -4),  emp_id
from   a_emp.adv_emp;
```

```
+--------+----------------------+--------+
| salary | truncate(salary, -4) | emp_id |
+--------+----------------------+--------+
|  24000 |                20000 |    100 |
|  98005 |                90000 |    101 |
|  30300 |                30000 |    102 |
|   9000 |                    0 |    103 |
|  50000 |                50000 |    104 |
|  12000 |                10000 |    108 |
|  15000 |                10000 |    109 |
|  30300 |                30000 |    110 |
|  65000 |                60000 |    145 |
|  88954 |                80000 |    146 |
|   6500 |                    0 |    150 |
|  80000 |                80000 |    155 |
|  15000 |                10000 |    160 |
|  15000 |                10000 |    161 |
|  98000 |                90000 |    162 |
|  65000 |                60000 |    200 |
|  15000 |                10000 |    201 |
|  44450 |                40000 |    203 |
|  15000 |                10000 |    204 |
|  15000 |                10000 |    205 |
|  88954 |                80000 |    206 |
|  65000 |                60000 |    207 |
+--------+----------------------+--------+
22 rows in set (0.00 sec)
```

Demo 07: Use that query with the truncated salary and the emp id as a subquery and do the rollup on that

```
select salary_10K, count(emp_id) as NumbrEmp
from (
   select  truncate(salary, -4)as salary_10K, emp_id
   from   a_emp.adv_emp) tbl_trunc
group by salary_10K with rollup;
+------------+----------+
| salary_10K | NumbrEmp |
+------------+----------+
|          0 |        2 |
|      10000 |        7 |
|      20000 |        1 |
|      30000 |        2 |
|      40000 |        1 |
|      50000 |        1 |
|      60000 |        3 |
|      80000 |        3 |
|      90000 |        2 |
|       NULL |       22 |
+------------+----------+
10 rows in set (0.00 sec)
```

We want to improve the display in the first column- we don't like it to see like we have 2 employees who are not paid anything- actually they are under 10K. So we want to display that message instead of 0.

It can help to use another level of subquery which will be used for the formatting; that way we avoid the formatting getting in the way of the group and rollup.

Demo 08: A three level query; there is no change in the display-yet

```
select salary_10K, NumbrEmp
from (
   select salary_10K, count(emp_id) as NumbrEmp
   from (
      select  truncate(salary, -4) as salary_10K, emp_id
      from   a_emp.adv_emp) tbl_trunc
      group by salary_10K with rollup) rolled;
+------------+----------+
| salary_10K | NumbrEmp |
+------------+----------+
|          0 |        2 |
|      10000 |        7 |
|      20000 |        1 |
|      30000 |        2 |
|      40000 |        1 |
|      50000 |        1 |
|      60000 |        3 |
|      80000 |        3 |
|      90000 |        2 |
|       NULL |       22 |
+------------+----------+
10 rows in set (0.00 sec)
```

Demo 09: I have three things that could be displayed in the first column- (1)the salary value, or(2) "under 10K' for the first row with a salary of 0 and (3) 'Total if the first column is null. Do that with a Case.

```
select case
    when  salary_10K is null then '    Total'
    when  salary_10K =0  then 'Under 10K'
    else  lpad(format(salary_10K,0), 9, ' ') end as salary_10K , NumbrEmp
from (
   select salary_10K, count(emp_id) as NumbrEmp
   from (
      select  truncate(salary, -4) as salary_10K, emp_id
      from   a_emp.adv_emp) tbl_trunc
      group by salary_10K with rollup) rolled;
+------------+----------+
| salary_10K | NumbrEmp |
+------------+----------+
| Under 10K  |        2 |
|    10,000  |        7 |
|    20,000  |        1 |
|    30,000  |        2 |
|    40,000  |        1 |
|    50,000  |        1 |
|    60,000  |        3 |
|    80,000  |        3 |
|    90,000  |        2 |
|     Total  |       22 |
+------------+----------+
10 rows in set (0.00 sec)
```

Demo 10: Having more than one group by with Rollup. We get a rollup by year hired as well as for the departments and the whole table. This would not have been practical with a Union query. I highlighted the total rows

```
Select dept_id, year_hired, sum(salary), count(salary), count(emp_id)
from   a_emp.adv_emp
group by dept_id, year_hired WITH ROLLUP
;
```

dept_id	year_hired	sum(salary)	count(salary)	count(emp_id)
10	1989	24000	1	1
10	NULL	24000	1	1
20	2004	15000	1	1
20	NULL	15000	1	1
30	1995	12000	1	1
30	2008	113005	2	2
30	2010	44450	1	1
30	2011	103954	2	2
30	2012	45300	2	2
30	NULL	318709	8	8
35	2011	228000	3	3
35	NULL	228000	3	3
80	2001	6500	1	1
80	2004	80000	1	1
80	2008	65000	1	1
80	NULL	151500	3	3
210	2010	9000	1	1
210	2012	50000	1	1
210	NULL	59000	2	2
215	2010	30300	1	1
215	2011	30000	2	2
215	2012	88954	1	1
215	NULL	149254	4	4
NULL	NULL	945463	22	22

These are the rows for department 30.

```
select emp_id, name_last, year_hired, salary
from a_emp.adv_emp
where dept_id = 30
order by year_hired;
```

emp_id	name_last	year_hired	salary	
108	Green	1995	12000	
101	Koch	2008	98005	<< 2 emp, salary total is 113005
205	Higgs	2008	15000	
203	Mays	2010	44450	
204	King	2011	15000	<< 2 emp, salary total is 103954
206	Geitz	2011	88954	
109	Fiet	2012	15000	<< 2 emp, salary total is 45300
110	Chen	2012	30300	

```
8 rows in set (0.00 sec)
```

Let's look at some more examples. This is from the order entry tables in the a_oe database. I am going to use only orders in February 2013 to reduce the output.

Demo 11: Without RollUp: Group by customer and order and get the amount due and number of items per order. We have 8 orders in that time span.

```
select   cust_id, ord_id
,        sum( quantity_ordered * quoted_price) as AmntDue
,        sum( quantity_ordered) as NumberItems
from     a_oe.order_headers
join     a_oe.order_details using (ord_id)
where    ord_date between '2013-02-01' and '2013-02-28'
group by cust_id, ord_id
;
+---------+--------+---------+-------------+
| cust_id | ord_id | AmntDue | NumberItems |
+---------+--------+---------+-------------+
|  403000 |    508 |  383.90 |          10 |
|  403000 |    509 | 1049.93 |           7 |
|  403000 |    511 |  405.94 |           2 |
|  403050 |    507 |  145.99 |           1 |
|  404950 |    510 |   74.25 |           3 |
|  409150 |    515 |   49.99 |           1 |
|  409150 |    518 |  759.43 |           6 |
|  409150 |    716 |   12.95 |           1 |
+---------+--------+---------+-------------+
8 rows in set (0.00 sec)
```

Adding RollUp we get a summary for each grouping components- so we have totals by customer and also a grand total. This gives us 6 customer total lines and one grand total line.

Demo 12: With RollUp; I filtered for only a few rows to be returned

```
select   cust_id, ord_id
,        sum( quantity_ordered * quoted_price) as AmntDue
,        sum( quantity_ordered) as NumberItems
from     a_oe.order_headers
join     a_oe.order_details using (ord_id)
where    ord_date between '2013-02-01' and '2013-02-28'
group by cust_id, ord_id with rollup;
+---------+--------+---------+-------------+
| cust_id | ord_id | AmntDue | NumberItems |
+---------+--------+---------+-------------+
|  403000 |    508 |  383.90 |          10 |
|  403000 |    509 | 1049.93 |           7 |
|  403000 |    511 |  405.94 |           2 |
|  403000 |   NULL | 1839.77 |          19 |
|  403050 |    507 |  145.99 |           1 |
|  403050 |   NULL |  145.99 |           1 |
|  404950 |    510 |   74.25 |           3 |
|  404950 |   NULL |   74.25 |           3 |
|  409150 |    515 |   49.99 |           1 |
|  409150 |    518 |  759.43 |           6 |
|  409150 |    716 |   12.95 |           1 |
|  409150 |   NULL |  822.37 |           8 |
|    NULL |   NULL | 2882.38 |          31 |
+---------+--------+---------+-------------+
13 rows in set (0.00 sec)
```

The rows with Null for just the order id are the customer totals-the total for all orders for that customer. There is no order id that would make sense here since this is an aggregate. The row with null for the customer id is the total for all customers.

You need to pay attention to the MySQL conventions for the group by clause. In demo 11, we would get the same results if we did Group by ord_id since for each order id there is only one customer id. But for demo 12, if you use group by ord_id with rollup and do not include the cust_id, then you are asking for a roll up only by order id and you do not get the customer totals. You do get the grand total line at the bottom of the result set but it will show a cust id value which can be confusing.

```
+---------+--------+---------+-------------+
| cust_id | ord_id | AmntDue | NumberItems |
+---------+--------+---------+-------------+
|  403050 |    507 |  145.99 |           1 |
|  403000 |    508 |  383.90 |          10 |
|  403000 |    509 | 1049.93 |           7 |
|  404950 |    510 |   74.25 |           3 |
|  403000 |    511 |  405.94 |           2 |
|  409150 |    515 |   49.99 |           1 |
|  409150 |    518 |  759.43 |           6 |
|  409150 |    716 |   12.95 |           1 |
|  409150 |   NULL | 2882.38 |          31 |
+---------+--------+---------+-------------+
9 rows in set (0.00 sec)
```

Try working with this to get the following result

```
+----------+----------+---------+-------------+
| Customer | order_id | AmntDue | NumberItems |
+----------+----------+---------+-------------+
| 403050   | 507      |  145.99 |           1 |
| 403000   | 508      |  383.90 |          10 |
| 403000   | 509      | 1049.93 |           7 |
| 404950   | 510      |   74.25 |           3 |
| 403000   | 511      |  405.94 |           2 |
| 409150   | 515      |   49.99 |           1 |
| 409150   | 518      |  759.43 |           6 |
| 409150   | 716      |   12.95 |           1 |
| Total    | Total    | 2882.38 |          31 |
+----------+----------+---------+-------------+
9 rows in set (0.00 sec)
```

If you are in the habit of using the MySQL Group by convention- take more care with these queries.

Demo 13: You can work with functions to adjust the output; this is often easier if you use another level of subquery,

```
select coalesce(cust_id, 'Grand Total') as CustID
, case when cust_id is null then ' ' else coalesce(ord_id, 'Cust Total') end as
OrderID
, AmntDue, NumberItems
from (
   select   cust_id, ord_id
   ,        sum( quantity_ordered * quoted_price) as AmntDue
   ,        sum( quantity_ordered) as NumberItems
   from     a_oe.order_headers
```

```
   join     a_oe.order_details using (ord_id)
   where    ord_date between '2013-02-01' and '2013-02-28'
   group by cust_id, ord_id with rollup
) tbl;
+------------+-----------+---------+-------------+
| CustID     | OrderID   | AmntDue | NumberItems |
+------------+-----------+---------+-------------+
| 403000     | 508       |  383.90 |          10 |
| 403000     | 509       | 1049.93 |           7 |
| 403000     | 511       |  405.94 |           2 |
| 403000     | Cust Total| 1839.77 |          19 |
| 403050     | 507       |  145.99 |           1 |
| 403050     | Cust Total|  145.99 |           1 |
| 404950     | 510       |   74.25 |           3 |
| 404950     | Cust Total|   74.25 |           3 |
| 409150     | 515       |   49.99 |           1 |
| 409150     | 518       |  759.43 |           6 |
| 409150     | 716       |   12.95 |           1 |
| 409150     | Cust Total|  822.37 |           8 |
| Grand Total|           | 2882.38 |          31 |
+------------+-----------+---------+-------------+
13 rows in set (0.00 sec)
```

Demo 14: Changing the order of the grouping items. Take demo 12 and reverse the order of the attributes in the Group by clause. If you were doing a plain Group By clause this would not make any difference; with the roll up it does. Since the ord_id was a finer level of grouping we do not get any customer id totals. These are order totals.

```
select   cust_id, ord_id
,        sum( quantity_ordered * quoted_price) as AmntDue
,        sum( quantity_ordered) as NumberItems
from     a_oe.order_headers
join     a_oe.order_details using (ord_id)
where    ord_date between '2013-02-01' and '2013-02-28'
group by ord_id, cust_id  with rollup;
+---------+--------+---------+-------------+
| cust_id | ord_id | AmntDue | NumberItems |
+---------+--------+---------+-------------+
|  403050 |    507 |  145.99 |           1 |
|    NULL |    507 |  145.99 |           1 |
|  403000 |    508 |  383.90 |          10 |
|    NULL |    508 |  383.90 |          10 |
|  403000 |    509 | 1049.93 |           7 |
|    NULL |    509 | 1049.93 |           7 |
|  404950 |    510 |   74.25 |           3 |
|    NULL |    510 |   74.25 |           3 |
|  403000 |    511 |  405.94 |           2 |
|    NULL |    511 |  405.94 |           2 |
|  409150 |    515 |   49.99 |           1 |
|    NULL |    515 |   49.99 |           1 |
|  409150 |    518 |  759.43 |           6 |
|    NULL |    518 |  759.43 |           6 |
|  409150 |    716 |   12.95 |           1 |
|    NULL |    716 |   12.95 |           1 |
|    NULL |   NULL | 2882.38 |          31 |
+---------+--------+---------+-------------+
17 rows in set (0.00 sec)
```

Demo 15: Here we have three levels of groupings and you can see the same pattern of nulls.
We have a rollup by each month in the year (highlighted in green) and a rollup by year (highlighted in orange) and a grand total

```
select   year(ord_date), month(ord_date)
, ord_id
, sum( quantity_ordered * quoted_price) as amntdue
, sum( quantity_ordered) as NumberItems
from     a_oe.order_headers
join     a_oe.order_details using (ord_id)
group by year(ord_date), month(ord_date), ord_id with rollup;
+----------------+-----------------+--------+----------+-------------+
| year(ord_date) | month(ord_date) | ord_id | amntdue  | NumberItems |
+----------------+-----------------+--------+----------+-------------+
|           2012 |               2 |    516 |    12.95 |           1 |
|           2012 |               2 |   NULL |    12.95 |           1 |<< month total
|           2012 |              10 |    105 |  1205.40 |          29 |
|           2012 |              10 |    106 |   255.95 |           1 |
|           2012 |              10 |    107 |    49.99 |           1 |
|           2012 |              10 |    108 |    22.50 |           1 |
|           2012 |              10 |    109 |   149.99 |           1 |
|           2012 |              10 |    110 |   299.98 |           2 |
|           2012 |              10 |    400 |   465.00 |          20 |
|           2012 |              10 |    401 |   158.85 |           6 |
|           2012 |              10 |    402 |   158.85 |           6 |
|           2012 |              10 |   NULL |  2766.51 |          67 |<< month total
|           2012 |              11 |    111 |   324.50 |          51 |
|           2012 |              11 |    112 |    99.98 |           2 |
|           2012 |              11 |    113 |    22.50 |           1 |
|           2012 |              11 |    114 |   625.00 |           5 |
|           2012 |              11 |    115 |  2305.00 |          11 |
|           2012 |              11 |    117 |   346.96 |           9 |
|           2012 |              11 |    118 |  1900.00 |           4 |
|           2012 |              11 |    119 |   225.00 |          10 |
|           2012 |              11 |    405 |  1068.75 |           6 |
|           2012 |              11 |    407 |    92.65 |           3 |
|           2012 |              11 |    408 |    15.00 |           1 |
|           2012 |              11 |   NULL |  7025.34 |         103 |<< month total
|           2012 |              12 |    124 |    14.99 |           1 |
|           2012 |              12 |    125 |    55.00 |           1 |
|           2012 |              12 |    126 |    49.99 |           1 |
|           2012 |              12 |    127 |   124.98 |           3 |
|           2012 |              12 |    128 |   511.90 |           2 |
|           2012 |              12 |    129 |   299.97 |           3 |
|           2012 |              12 |    130 |  1145.00 |           3 |
|           2012 |              12 |   NULL |  2201.83 |          14 |<< month total
|           2012 |            NULL |   NULL | 12006.63 |         185 |<< year total
|           2013 |               1 |    120 |  1750.00 |          10 |
|           2013 |               1 |    121 |  3800.25 |          21 |
|           2013 |               1 |    122 |   105.00 |           5 |
|           2013 |               1 |    503 |   149.99 |           1 |
|           2013 |               1 |    504 |   149.99 |           1 |
|           2013 |               1 |    505 |   350.94 |           4 |
|           2013 |               1 |   NULL |  6306.17 |          42 |<< month total

. . . rows skipped here for display
|           2013 |               8 |    550 |  2050.25 |          11 |
|           2013 |               8 |    551 |  3500.00 |          20 |
|           2013 |               8 |    552 |   157.30 |          10 |
|           2013 |               8 |   NULL |  5707.55 |          41 |<< month total
|           2013 |            NULL |   NULL | 50590.75 |         427 |<< year total
|           NULL |            NULL |   NULL | 62597.38 |         612 |<< Grand Total
+----------------+-----------------+--------+----------+-------------+
90 rows in set (0.00 sec)
```

1.1. Some Issues with RollUp

- With MySQL, you cannot use an Order by clause if you use With Rollup

- You can do a Descending sort of the Group By attributes; the totals are still at the bottom of each group.

In the previous demo, I can use the following Group By clause and the rows appear with year 2012 first followed by year 2011

```
group by year(ord_date) desc, month(ord_date), ord_id with rollup;
```

You may have trouble trying to show only the total lines by simply adding a Having clause testing for nulls. This has to do with the way that MySQL implements this clause and the time at which it generates and adds the "null" rows. If you try to get just the totals by using the having clause here, you get no rows returned.

Demo 16:

```
select   year(ord_date), month(ord_date), ord_id
,        sum( quantity_ordered * quoted_price) as amntdue
,        sum( quantity_ordered) as NumberItems
from     a_oe.order_headers
join     a_oe.order_details using (ord_id)
group by year(ord_date), month(ord_date), ord_id with rollup
having ord_id is null;
```

Manual: Because the NULL values in the super-aggregate rows are placed into the result set at such a late stage in query processing, you cannot test them as NULL values within the query itself. For example, you cannot add HAVING product IS NULL to the query to eliminate from the output all but the super-aggregate rows.

On the other hand, the NULL values do appear as NULL on the client side and can be tested as such using any MySQL client programming interface.