Table of Contents

1. Non-traditional Group By

MySQL extends the Group By clause in a non-traditional way that you should carefully consider. There is nothing wrong with this extension, but you need to take the responsibility to avid logic errors.

Demo 01: Create the following table in a_testbed.

```
Create table a_testbed.z_group ( an_id integer primary key
, an_name varchar(15) not null
, an_type varchar(15) not null
, an_price integer not null);

Insert into a_testbed.z_group values
(1, 'Max',  'dog', 500),    (2, 'Max',  'cat', 25),
(3, 'Max',  'dog', 350),    (4, 'Spot', 'dog', 400),
(5, 'Spot', 'snake', 125),  (6, 'Spot', 'dog', 400),
(7, 'Max',  'elephant', 75000);

Select *
From a_testbed.z_group;
+-------+---------+----------+----------+
| an_id | an_name | an_type  | an_price |
+-------+---------+----------+----------+
|     1 | Max     | dog      |      500 |
|     2 | Max     | cat      |       25 |
|     3 | Max     | dog      |      350 |
|     4 | Spot    | dog      |      400 |
|     5 | Spot    | snake    |      125 |
|     6 | Spot    | dog      |      400 |
|     7 | Max     | elephant |    75000 |
+-------+---------+----------+----------+
```

We could group these rows by the animal name. This output makes sense- we have three animals named Spot and 4 named Max.

Demo 02: Group by an_name, display grouping column and aggregate

```
Select an_name, count(*)
From   a_testbed.z_group
Group by an_name
;
+---------+----------+
| an_name | count(*) |
+---------+----------+
| Max     |        4 |
| Spot    |        3 |
+---------+----------+
```

We could do the following query. This is a non-traditional extension of the Group By that MySQL allows.

Demo 03: Group by an_name, display grouping column and aggregate and non-aggregate columns

```
Select an_name, an_type, an_price, count(*)
From a_testbed.z_group
```

```
Group by an_name;
+---------+---------+----------+----------+
| an_name | an_type | an_price | count(*) |
+---------+---------+----------+----------+
| Max     | dog     |      500 |        4 |
| Spot    | dog     |      400 |        3 |
+---------+---------+----------+----------+
```

But what does this output mean? We have a group for animals named Max- but why does this query return an_type dog and an_price 500. Why not an_type elephant and an_price 75000?
So how did MySQL decide which type and price to return for the group of animals named Max? The manual says that it can return any one of the values from that group that it feels like. Kinda a bad attitude!

But you can write traditional SQL and decide on one of these two Select versions:

Demo 04: Here you group by all of the non-aggregated attributes. We have one group that has all of the dogs named Max that cost 400. The other groups are size 1.

```
Select an_name, an_type, an_price, count(*)
From a_testbed.z_group
Group by an_name, an_type, an_price
;
+---------+----------+----------+----------+
| an_name | an_type  | an_price | count(*) |
+---------+----------+----------+----------+
| Max     | cat      |       25 |        1 |
| Max     | dog      |      350 |        1 |
| Max     | dog      |      500 |        1 |
| Max     | elephant |    75000 |        1 |
| Spot    | dog      |      400 |        2 |
| Spot    | snake    |      125 |        1 |
+---------+----------+----------+----------+
```

Demo 05: Here we display the an_name which is the grouping attribute and use an aggregate for price- this finds the most expensive price for animals of each name.

```
Select an_name,  max(an_price), count(*)
From a_testbed.z_group
Group by an_name
;
+---------+---------------+----------+
| an_name | max(an_price) | count(*) |
+---------+---------------+----------+
| Max     |         75000 |        4 |
| Spot    |           400 |        3 |
+---------+---------------+----------+
```

If you are grouping by an attribute and you know from the design of the query that all of the rows in that group will have the same values for another attribute, then you can display that other attribute in the select without an aggregate function.

The following two queries will return the same record set. The group by in the first query is traditional SQL and the Group By in the second query uses the MySQL extension. The reason the second query works properly is that we know that if we group by the ord_id, all of the rows in each group will have the same cust_id and the same shipping mode.

Demo 06:

```
Select cust_id
, ord_id
, shipping_mode
, count(*) AS "NumberLineItems"
From a_oe.order_headers
Join a_oe.order_details using (ord_id)
Group by ord_id, cust_id, shipping_mode
Order by cust_id, ord_id
;
```

Demo 07:

```
Select cust_id
, ord_id
, shipping_mode
, count(*) AS "NumberLineItems"
From a_oe.order_headers
Join a_oe.order_details using (ord_id)
Group by ord_id
Order by cust_id, ord_id
;
```

We are grouping by the order id; we know from the design of the tables that all of the rows in a order id group will have the same values for the customer id and for the shipping mode. These attributes come from the orders table and there is only one customer id per order and only one shipping mode per order and the order id uniquely identifies the order. The query has added in additional rows as it joined to the order details table but those extra rows are reflected in the count. So it makes sense that we could display the customer ID and shipping mode.

Demo 08: This is a query we did earlier to find the amount due for each order

```
Select cust_id, cust_name_last, ord_id
, cast(ord_date as date) as OrderDate
,  sum( quantity_ordered * quoted_price) as amntdue
From a_oe.customers
Join a_oe.order_headers  using (cust_id)
Join a_oe.order_details using (ord_id)
Group by ord_id, cust_id, cust_name_last, ord_date
Order by cust_id, ord_id
;
```

Demo 09: We could do this by grouping on just the order id and get the same results. These each returned 67 rows with my data set; I have 67 orders with detail rows.

```
Select cust_id, cust_name_last, ord_id
, cast(ord_date as date) as OrderDate
, sum( quantity_ordered * quoted_price) as amntdue
From  a_oe.customers
Join a_oe.order_headers  using (cust_id)
Join a_oe.order_details using (ord_id)
Group by ord_id
Order by cust_id, ord_id
;
```

But suppose we decided to group on the order date; that is also a column in the order table and each order has a single order date. When I run this query I get 41 rows even though I have 67 orders.

Demo 10: Incorrect grouping to find the amount due per order

```
Select cust_id, cust_name_last, ord_id
, cast(ord_date as date) as OrderDate
, sum( quantity_ordered * quoted_price) as amntdue
From a_oe.customers
Join a_oe.order_headers  using (cust_id)
Join a_oe.order_details using (ord_id)
Group by ord_date
Order by cust_id, ord_id
;
```

It will be easier to see what happened if we filter on only one order date.

Demo 11: This is grouping by the order id and returns six rows- one for each order for '2012-06-04'

```
Select cust_id, cust_name_last, ord_id
, cast(ord_date as date) as OrderDate
, sum( quantity_ordered * quoted_price) as amntdue
From  a_oe.customers
Join a_oe.order_headers  using (cust_id)
Join a_oe.order_details using (ord_id)
Where ord_date = '2012-06-04'
Group by ord_id
Order by cust_id, ord_id;
+---------+---------------+--------+------------+---------+
| cust_id | cust_name_last | ord_id | OrderDate  | amntdue |
+---------+---------------+--------+------------+---------+
|  401250 | Morse          |    301 | 2012-06-04 |  205.00 |
|  403000 | Williams       |    390 | 2012-06-04 | 1400.00 |
|  403000 | Williams       |    395 | 2012-06-04 | 2925.00 |
|  404000 | Olmsted        |    302 | 2012-06-04 |  469.95 |
|  900300 | McGold         |    307 | 2012-06-04 | 4500.00 |
|  903000 | McGold         |    306 | 2012-06-04 | 1000.00 |
+---------+---------------+--------+------------+---------+
```

Demo 12: This is grouping by the order date and returns one row for '2012-06-04'. The amount due is the sum of the amount due for all orders for that date. Customer 401250 is not happy being charged for all of those orders.

```
Select cust_id, cust_name_last, ord_id
, cast(ord_date as date) as OrderDate
, sum( quantity_ordered * quoted_price) as amntdue
From a_oe.customers
Join a_oe.order_headers  using (cust_id)
Join a_oe.order_details using (ord_id)
Where ord_date = '2012-06-04'
Group by  ord_date
Order by cust_id, ord_id
;
+---------+---------------+--------+------------+----------+
| cust_id | cust_name_last | ord_id | OrderDate  | amntdue  |
+---------+---------------+--------+------------+----------+
|  401250 | Morse          |    301 | 2012-06-04 | 10499.95 |
+---------+---------------+--------+------------+----------+
```

The order date does not identify orders so it is not a good candidate for the group by if you want an amount due per order. If you want to see the amount of the orders on each date, then a group by order date makes sense.

Demo 13: Order total by Order date

```
   Select cast(ord_date as date) as OrderDate
   , count(*) as NumberOfOrders
   , sum( quantity_ordered * quoted_price) as amntdue
   From  a_oe.order_headers
   Join a_oe.order_details using (ord_id)
   Group by OrderDate
   Order by OrderDate;
+------------+----------------+----------+
| OrderDate  | NumberOfOrders | amntdue  |
+------------+----------------+----------+
| 2011-02-05 |              1 |    45.00 |
| 2011-12-07 |              1 |    14.99 |
| 2011-12-09 |              1 |    55.00 |
| 2011-12-15 |              6 |   986.84 |
| 2011-12-30 |              3 |  1145.00 |
| 2012-01-04 |              1 |   149.99 |
| 2012-01-11 |              1 |   149.99 |
| 2012-01-12 |              3 |   350.94 |
| 2012-02-01 |              7 |  1010.08 |
| 2012-02-02 |              2 |    62.94 |
| 2012-02-03 |              4 |   759.43 |
| 2012-02-04 |              4 |   192.44 |
| 2012-02-12 |              2 |  1049.93 |
. . . rows ommitted
| 2013-01-23 |              4 |   105.00 |
| 2013-02-05 |              3 |    81.45 |
| 2013-02-06 |              1 |    12.95 |
| 2013-02-08 |              2 |   727.97 |
+------------+----------------+----------+
41 rows in set (0.00 sec)
```

The morale here is- as always- if you write bad code you are apt to get bad results. One of the problems with small data sets as we often use for testing is that you might not have noticed that problem; if you had picked order date 2011-02-05, the error would not have been evident as there is only one order for that date.

2. Group_Concat function

MySQL has a function Group_Concat which you can use to get a comma separated list of the values in a group. This function skips any null values in the list. The list is limited to a default length of 1024.

Demo 14: Here we group by the an_type and get a list of all of the animals names in that group.

```
   Select an_type, count(*), group_concat(an_name)
   From a_testbed.z_group
   Group by an_type;
   +----------+----------+----------------------+
   | an_type  | count(*) | group_concat(an_name) |
   +----------+----------+----------------------+
   | cat      |        1 | Max                  |
   | dog      |        4 | Max,Max,Spot,Spot    |
   | elephant |        1 | Max                  |
   | snake    |        1 | Spot                 |
   +----------+----------+----------------------+
```

Demo 15: You can add distinct to get distinct animal names.

```
Select an_type, count(*), group_concat(distinct an_name)
From a_testbed.z_group
Group by an_type;
+----------+----------+--------------------------------+
| an_type  | count(*) | group_concat(distinct an_name) |
+----------+----------+--------------------------------+
| cat      |        1 | Max                            |
| dog      |        4 | Max,Spot                       |
| elephant |        1 | Max                            |
| snake    |        1 | Spot                           |
+----------+----------+--------------------------------+
```

Demo 16: You can add distinct to get distinct prices

```
Select an_type, count(*), group_concat(an_price order by an_price )
From a_testbed.z_group
Group by an_type;
+----------+----------+-------------------------------------------+
| an_type  | count(*) | group_concat(an_price order by an_price ) |
+----------+----------+-------------------------------------------+
| cat      |        1 | 25                                        |
| dog      |        4 | 350,400,400,500                           |
| elephant |        1 | 75000                                     |
| snake    |        1 | 125                                       |
+----------+----------+-------------------------------------------+
```

Demo 17:

```
Select an_type, count(*), group_concat(concat(an_name, ' at $', an_price) )
From a_testbed.z_group
Group by an_type;
+----------+----------+-----------------------------------------------------+
| an_type  | count(*) | group_concat(concat(an_name, ' at $', an_price) )   |
+----------+----------+-----------------------------------------------------+
| cat      |        1 | Max at $25                                          |
| dog      |        4 | Max at $500,Max at $350,Spot at $400,Spot at $400   |
| elephant |        1 | Max at $75000                                       |
| snake    |        1 | Spot at $125                                        |
+----------+----------+-----------------------------------------------------+
```

Demo 18: Group_concat with and without Distinct

```
Select left(ord_date, 10) as OrdDate
, group_concat(prod_id order by prod_id) as ProductsSold
, group_concat(distinct prod_id order by prod_id ) as DistinctProductsSold
From a_oe.customers
Join a_oe.order_headers  using (cust_id)
Join a_oe.order_details using (ord_id)
Where year(ord_date) = 2012
Group by  ord_date
Order by ord_date
;
```

Selected rows

```
+------------+----------------------------------+------------------------------+
| OrdDate    | ProductsSold                     | DistinctProductsSold         |
+------------+----------------------------------+------------------------------+
```

```
| 2012-01-04 | 1130                                | 1130                          |
| 2012-01-11 | 1130                                | 1130                          |
| 2012-01-12 | 1060,1080,1110                      | 1060,1080,1110                |
| 2012-02-01 | 1060,1071,1072,1090,1130,1152,1152 | 1060,1071,1072,1090,1130,1152 |
| 2012-02-02 | 1020,1110                           | 1020,1110                     |
| 2012-02-03 | 1125,1141,2014,2984                 | 1125,1141,2014,2984           |
| 2012-02-04 | 1020,1020,1110,2747                 | 1020,1110,2747                |
| 2012-02-12 | 1090,1130                           | 1090,1130                     |
| 2012-05-01 | 1010,1030,1070,1141,1150            | 1010,1030,1070,1141,1150      |
| 2012-05-05 | 1130                                | 1130                          |
```