Table of Contents

1. Partitioning the Table Expression with Group By

When you partition a table expression, you create a series of virtual sub-tables/partitions into which each row from the parent table will be placed; each table row appears in only one of these groups. You can partition the table on the basis of one or more attributes or expressions. When you apply an aggregate function to a partitioned table, you will get a separate summary for each sub-table rather than for the table as a whole.

We start here with using just the Group by clause without any aggregates.

Demo 01: Displaying all departments— one per employee.

```
Select dept_id
From a_emp.employees;
+---------+
| dept_id |
+---------+
|      10 |
|      20 |
|      30 |
|      30 |
|      30 |
|      30 |
|      30 |
|      30 |
|      30 |
|      30 |
|      35 |
|      35 |
|      35 |
|      80 |
|      80 |
|      80 |
|     210 |
|     210 |
|     215 |
|     215 |
|     215 |
|     215 |
+---------+
```

Demo 02: Displaying one of each different department represented in the employee table.
Here we group by the dept_id and return one row for each grouping
We could do this more efficiently by using the keyword Distinct

```
Select dept_id
From a_emp.employees
Group by dept_id;
```

```
+---------+
| dept_id |
+---------+
|      10 |
|      20 |
|      30 |
|      35 |
|      80 |
|     210 |
|     215 |
+---------+
```

Demo 03: If we do the following query- we do get a result, but it is a meaningless result. This groups by the department but MySQL takes one employee from the department and displays that emp_id value. There is no rationale for displaying the emp_id values shown here.

```
Select dept_id, emp_id
From a_emp.employees
Group by dept_id
;
+---------+--------+
| dept_id | emp_id |
+---------+--------+
|      10 |    100 |
|      20 |    201 |
|      30 |    101 |
|      35 |    162 |
|      80 |    145 |
|     210 |    103 |
|     215 |    102 |
+---------+--------+
```

1.1. Multiple-column grouping

In the following example we have two columns in the Group by clause and we get more groups. The more grouping columns we have the finer the distinction we are making between the way that we categorize a row and the more potential groups we will have. If we were to group by the pk columns, we would make a group for each row in the table. This is generally not a good idea.

Note that we get a sub-table only for data combinations that actually exist in our table. We have at least two rows with different job_id values in department 80 so we get two sub-tables for dept 80. This is **not** a Cartesian product of all possible combinations of dept_id and job_id.

Demo 04: Using two grouping attributes gives us more sub-tables.

```
Select dept_id, job_id
From a_emp.employees
Group by dept_id, job_id;
+---------+--------+
| dept_id | job_id |
+---------+--------+
|      10 |      1 |
|      20 |      2 |
|      30 |     16 |
|      30 |     32 |
|      35 |      8 |
|      35 |     16 |
```

```
|      80 |      4 |
|      80 |      8 |
|     210 |     32 |
|     210 |     64 |
|     215 |     16 |
|     215 |     32 |
|     215 |     64 |
+---------+--------+
```

Demo 05: If we group first by the job_id and then by the dept_id, we get the same result set because we are creating the same set of subtables. We get a group for each combination of dept id and job id.

```
Select dept_id, job_id
From a_emp.employees
Group by job_id, dept_id
;+---------+--------+
| dept_id | job_id |
+---------+--------+
|      10 |      1 |
|      20 |      2 |
|      80 |      4 |
|      35 |      8 |
|      80 |      8 |
|      30 |     16 |
|      35 |     16 |
|     215 |     16 |
|      30 |     32 |
|     210 |     32 |
|     215 |     32 |
|     210 |     64 |
|     215 |     64 |
+---------+--------+
```

Demo 06: You can sort the result set after you group it.

```
Select dept_id, job_id
From a_emp.employees
Group by job_id, dept_id
Order by dept_id, emp_id;
+---------+--------+
| dept_id | job_id |
+---------+--------+
|      10 |      1 |
|      20 |      2 |
|      30 |     16 |
|      30 |     32 |
|      35 |     16 |
|      35 |      8 |
|      80 |      4 |
|      80 |      8 |
|     210 |     64 |
|     210 |     32 |
|     215 |     64 |
|     215 |     32 |
|     215 |     16 |
+---------+--------+
```

In these queries we have been displaying the grouping attributes. That is not a requirement, but is generally helpful.

2. Using Groups and Aggregate Functions

We probably wanted to create those sub-tables so that we could get information about those groupings. In this example, we want to know how many employees are in each group and their average salary.

Demo 07: For each department, how many employees are there and what is the average salary for that department?

```
Select dept_id
, COUNT(*)
, avg(salary)
From a_emp.employees
Group by dept_id;
+---------+----------+--------------+
| dept_id | COUNT(*) | AVG(salary)  |
+---------+----------+--------------+
|      10 |        1 | 24000.000000 |
|      20 |        1 | 15000.000000 |
|      30 |        8 | 39838.625000 |
|      35 |        3 | 76000.000000 |
|      80 |        3 | 50500.000000 |
|     210 |        2 | 29500.000000 |
|     215 |        4 | 37313.500000 |
+---------+----------+--------------+
```

Demo 08: What is the average list price and the largest list price for each category of product we sell?

```
Select catg_id
, avg(prod_list_price) as "Avg List Price"
, MAX(prod_list_price) as "Max List Price"
From a_prd.products
Group by catg_id;
+---------+----------------+----------------+
| catg_id | Avg List Price | Max List Price |
+---------+----------------+----------------+
| APL     |     479.986000 |         850.00 |
| GFD     |       8.750000 |          12.50 |
| HD      |      23.875000 |          45.00 |
| HW      |      67.641000 |         149.99 |
| PET     |     135.262000 |         549.99 |
| SPG     |     178.125000 |         349.95 |
+---------+----------------+----------------+
```

The traditional dbms rule for grouping is :If you use an aggregate function and a non-aggregated attribute in the SELECT clause, then you must GROUP BY the non-aggregated attribute(s). If you use a GROUP BY clause, then the SELECT clause can include only aggregate functions and the attributes used for grouping. You are not required to show the grouping column in the output.

However MySQL does not follow this rule; this feature is discussed in the document on MySQLGroupFeatures in this unit's notes.

Demo 09: Using two groups and aggregate functions; this groups by the different departments and job ids.

```
Select dept_id, job_id
, COUNT(*) AS NumEmployee
, AVG(salary) AS AvgSalary
From a_emp.employees
Group by dept_id, job_id
Order by Avg(salary);
+---------+--------+-------------+--------------+
```

```
| dept_id | job_id | NumEmployee | AvgSalary    |
+---------+--------+-------------+--------------+
|     210 |     64 |           1 |  9000.000000 |
|     215 |     32 |           1 | 15000.000000 |
|     215 |     16 |           1 | 15000.000000 |
|      20 |      2 |           1 | 15000.000000 |
|      10 |      1 |           1 | 24000.000000 |
|      30 |     32 |           4 | 37313.500000 |
|      30 |     16 |           4 | 42363.750000 |
|      80 |      8 |           2 | 43250.000000 |
|     210 |     32 |           1 | 50000.000000 |
|     215 |     64 |           2 | 59627.000000 |
|      35 |      8 |           1 | 65000.000000 |
|      80 |      4 |           1 | 65000.000000 |
|      35 |     16 |           2 | 81500.000000 |
+---------+--------+-------------+--------------+
13 rows in set (0.00 sec)
```

Suppose we wanted to show statistics for average salaries by department- but not identify the department by name (for political reasons!) It is up to the user if this output is useful

Demo 10: Using two groups and aggregate functions.

```
Select COUNT(*) AS NumEmployee
, AVG(salary) AS AvgSalary
From a_emp.employees
Group by dept_id, job_id
Order by Avg(salary);
+-------------+--------------+
| NumEmployee | AvgSalary    |
+-------------+--------------+
|           1 |  9000.000000 |
|           1 | 15000.000000 |
|           1 | 15000.000000 |
|           1 | 15000.000000 |
|           1 | 24000.000000 |
|           4 | 37313.500000 |
|           4 | 42363.750000 |
|           2 | 43250.000000 |
|           1 | 50000.000000 |
|           2 | 59627.000000 |
|           1 | 65000.000000 |
|           1 | 65000.000000 |
|           2 | 81500.000000 |
+-------------+--------------+
```

In the previous document we found the most expensive spg item and we found the most pet item. What if we want to find the most expensive item(s) in each category.

Demo 11: We can use this query to find the max price in each category

```
Select catg_id,  MAX(prod_list_price) as MaxPrice
From a_prd.products
Group by catg_id ;
+---------+----------+
| catg_id | MaxPrice |
+---------+----------+
| APL     |   850.00 |
| GFD     |    12.50 |
```

```
| HD       |    45.00 |
| HW       |   149.99 |
| MUS      |    15.95 |
| PET      |   549.99 |
| SPG      |   349.95 |
+----------+----------+
```

Demo 12: We can use a row comparison to get the high priced items in each category

```
Select p.catg_id, p.prod_id, p.prod_list_price, p.prod_desc
From a_prd.products  P
Where  (p.catg_id,p.prod_list_price) IN
       (Select catg_id, MAX(prod_list_price) as MaxPrice
        From  a_prd.products
        Group by catg_id );
+---------+---------+-----------------+------------------------------------------+
| catg_id | prod_id | prod_list_price | prod_desc                                |
+---------+---------+-----------------+------------------------------------------+
| SPG     |    1040 |          349.95 | Super Flyer Treadmill                    |
| HW      |    1090 |          149.99 | Gas grill                                |
| APL     |    1126 |          850.00 | Low Energy washer Dryer combo            |
| HW      |    1160 |          149.99 | Stand Mixer with attachments             |
| MUS     |    2014 |           15.95 | Bix Beiderbecke - Tiger Rag              |
| PET     |    4567 |          549.99 | Our highest end cat tree- you gotta seethis |
| PET     |    4568 |          549.99 | Satin four-poster cat bed                |
| GFD     |    5000 |           12.50 | Cello bag of mixed fingerling potatoes   |
| HD      |    5005 |           45.00 | Steel Shingler hammerhammer              |
+---------+---------+-----------------+------------------------------------------+
```

3. Using the Having Clause

Sometimes you have grouped your rows but you do not want to see all of the groups displayed. Perhaps you have grouped employees by department and only want to see those departments where the average salary is more than 30,000. In this case you want to filter the returns by the aggregate function return values. It is not allowed to put an aggregate function in the Where clause. So we need another clause to do this- the Having clause. The Having clause filters the groups. In most cases you will use aggregate functions in the Having clause.

Demo 13: This groups by department and uses all rows.

```
Select dept_id
, COUNT(*)
, AVG(salary)
From a_emp.employees
Group by dept_id;
+---------+----------+--------------+
| dept_id | COUNT(*) | AVG(salary)  |
+---------+----------+--------------+
|      10 |        1 | 24000.000000 |
|      20 |        1 | 15000.000000 |
|      30 |        8 | 39838.625000 |
|      35 |        3 | 76000.000000 |
|      80 |        3 | 50500.000000 |
|     210 |        2 | 29500.000000 |
|     215 |        4 | 37313.500000 |
+---------+----------+--------------+
```

Demo 14: This groups by department and returns groups where the average salary exceed 30000.

```
Select dept_id
, count(*)
, avg(salary)
From a_emp.employees
Group by dept_id
Having avg(salary) > 30000;
+---------+----------+--------------+
| dept_id | count(*) | avg(salary)  |
+---------+----------+--------------+
|      30 |        8 | 39838.625000 |
|      35 |        3 | 76000.000000 |
|      80 |        3 | 50500.000000 |
|     215 |        4 | 37313.500000 |
+---------+----------+--------------+
```

If you have a choice of filtering in the Where clause or in the Having clause- pick the Where clause. Suppose we want to see the average salary for department where the average is more than 3000 but we only want to consider department 30-40. In that case we should filter on the dept_id in the Where clause and filter on the ave(salary) in the Having clause. There is no sense creating groups for departments that we do not want to consider.

Demo 15: Groups with where clause and a having clause

```
Select dept_id
, count(*)
, avg(salary)
From a_emp.employees
Where  dept_id between 30 and 40
Group by dept_id
Having avg(salary) > 30000;
+---------+----------+--------------+
| dept_id | count(*) | avg(salary)  |
+---------+----------+--------------+
|      30 |        8 | 39838.625000 |
|      35 |        3 | 76000.000000 |
+---------+----------+--------------+
```

There are several different meanings to finding averages. The following query allows only employees who earn more than 30000 to be put into the groups- and we get a different answer. The issue here is not "which is the right query?" but rather "what do you want to know?" Do you want to know the average salary of the more highly paid employees or the department with the higher average salaries?

Demo 16: Using a Where clause— this acts before the grouping. Only employees earning more than 3000 get into the groups.

```
Select dept_id
, COUNT(*)
, AVG(salary)
From a_emp.employees
Where salary > 30000
Group by dept_id;
+---------+----------+--------------+
| dept_id | COUNT(*) | AVG(salary)  |
+---------+----------+--------------+
|      30 |        4 | 65427.250000 |
|      35 |        3 | 76000.000000 |
|      80 |        2 | 72500.000000 |
|     210 |        1 | 50000.000000 |
|     215 |        2 | 59627.000000 |
+---------+----------+--------------+
```

Demo 17: Show statistics only if the department has more than three employees.

```
Select dept_id
, COUNT(*)
, AVG(salary)
From  a_emp.employees
Group by dept_id
Having Count(*) > 3
;
+---------+----------+--------------+
| dept_id | COUNT(*) | AVG(salary)  |
+---------+----------+--------------+
|      30 |        8 | 39838.625000 |
|     215 |        4 | 37313.500000 |
+---------+----------+--------------+
```

Demo 18: What if you want to find out if you have more than one employee with the same last name?

```
Select name_last  As DuplicateName
, COUNT(*)
From a_emp.employees
Group by name_last
Having COUNT(*) > 1;
+---------------+----------+
| DuplicateName | COUNT(*) |
+---------------+----------+
| King          |        2 |
| Russ          |        2 |
+---------------+----------+
```

If you want to determine the number of order lines for each order, you can use the order details table and group on the ord_id.

Demo 19: How many order lines for each order ?

```
Select ord_id
, count(*) AS "NumberLineItems"
From a_oe.order_details
Group by ord_id
Order by ord_id
;
+---------+-----------------+
|  ord_id | NumberLineItems |
+---------+-----------------+
. . .
|    111 |               2 |
|    112 |               1 |
|    113 |               1 |
|    114 |               1 |
|    115 |               4 |
|    117 |               3 |
|    118 |               1 |
|    119 |               1 |
|    120 |               1 |
|    121 |               2 |
|    122 |               4 |
. . .
```

3.1. Grouping on additional columns

But suppose I also wanted to show the customer ID and the shipping mode for each of these orders. You know that each order has a single cust_id and shipping mode, so it would be logical that you could just add them to the Select clause. In MySql you can do that. In a traditional grouping query, you would need to include them in the Group By clause, or wrap them in an aggregate function, such as Max, in order to display them. You need to be certain that the extra grouping attributes are not changing the logic of the query.

Demo 20: MySQL group by.

```
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;
```

Traditional Group By with extra group levels for this query.

```
Select cust_id
, ord_id
, shipping_mode
,  ount(*) 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
;
+---------+--------+---------------+-----------------+
| cust_id | ord_id | shipping_mode | NumberLineItems |
+---------+--------+---------------+-----------------+
|  400300 |    378 | USPS1         |               2 |
|  401250 |    106 | FEDEX1        |               1 |
|  401250 |    113 | FEDEX2        |               1 |
|  401250 |    119 | NULL          |               1 |
|  401250 |    301 | FEDEX2        |               1 |
|  401890 |    112 | USPS1         |               1 |
|  401890 |    519 | USPS1         |               2 |
|  402100 |    114 | USPS1         |               1 |
|  402100 |    115 | USPS1         |               4 |
|  402100 |    117 | NULL          |               3 |
|  403000 |    105 | UPSGR         |               3 |
| 003000  |    109 | UPSGR         |               1 |
. . .  rows omitted
```

Demo 21: What is 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;
```

```
+---------+----------------+--------+------------+---------+
| cust_id | cust_name_last | ord_id | OrderDate  | amntdue |
+---------+----------------+--------+------------+---------+
|  400300 | McGold         |    378 | 2013-06-14 | 4500.00 |
|  401250 | Morse          |    106 | 2012-10-01 |  255.95 |
|  401250 | Morse          |    113 | 2012-11-08 |   22.50 |
|  401250 | Morse          |    119 | 2012-11-28 |  225.00 |
|  401250 | Morse          |    301 | 2013-06-04 |  205.00 |
|  401890 | Northrep       |    112 | 2012-11-08 |   99.98 |
|  401890 | Northrep       |    519 | 2013-03-04 |  114.74 |
|  402100 | Morise         |    114 | 2012-11-08 |  625.00 |
|  402100 | Morise         |    115 | 2012-11-08 | 2305.00 |
|  402100 | Morise         |    117 | 2012-11-28 |  346.96 |
. . .  rows omitted
```

4. Aggregates and Nulls

Demo 22: The product table has a nullable attribute for the warranty period. We can use count(*) to get the number of products, and count(prod_warranty_period) to get the number of products with warranty period values. We have 19 products with no set warranty period

```
Select count(*) as "NumProducts"
, count(prod_warranty_period) as "NumWithWarranty"
From a_prd.products;
+-------------+-----------------+
| NumProducts | NumWithWarranty |
+-------------+-----------------+
|          49 |              30 |
+-------------+-----------------+
```

Demo 23: If we group by the warranty period, we get one group for all of the nulls.

```
Select prod_warranty_period ,  count(*) as "NumProducts"
From a_prd.products
Group by prod_warranty_period
Order by  prod_warranty_period
;
+----------------------+-------------+
| prod_warranty_period | NumProducts |
+----------------------+-------------+
|                 NULL |          19 |
|                    0 |           5 |
|                   12 |           7 |
|                   18 |           2 |
|                   36 |           6 |
|                   60 |          10 |
+----------------------+-------------+
```

Demo 24: What if we want to display a message instead of a null for the null group? MySQL will allow the use of coalesce with different data types.

```
Select coalesce( (prod_warranty_period), 'no warranty') as "Warranty"
, count(*) as "NumProducts"
From a_prd.products
Group by prod_warranty_period
Order by prod_warranty_period
;
```

```
+--------------+-------------+
| Warranty     | NumProducts |
+--------------+-------------+
| no warranty |           19 |
| 0            |            5 |
| 12           |            7 |
| 18           |            2 |
| 36           |            6 |
| 60           |           10 |
+--------------+-------------+
```

Demo 25: Create the following test table

```
use a_testbed;
create table z_aggs (id integer, fee decimal(5,2));
  insert into z_aggs values (1, 45);
  insert into z_aggs values (2, 0);
  insert into z_aggs values (3, null);
  insert into z_aggs values (4, null);
```

If we aggregate across the entire table, we get one row; the row count and the Id count are both 4. We have a count of 2 for fees, since count(fee) does not count the nulls. The sum, avg and max all ignore the nulls.

```
Select count(*), count(id), count(fee), sum(fee), avg(fee), max(fee)
From z_aggs;
+----------+-----------+------------+----------+-----------+----------+
| count(*) | count(id) | count(fee) | sum(fee) | avg(fee)  | max(fee) |
+----------+-----------+------------+----------+-----------+----------+
|        4 |         4 |          2 |    45.00 | 22.500000 |    45.00 |
+----------+-----------+------------+----------+-----------+----------+
1 row in set (0.00 sec)
```

If we group by the fee attribute, we get one group for the value 45.0, one group for the value 0.0 and one group for the two nulls.

```
Select count(*), count(fee) , max(fee)
From z_aggs
Group by fee;
+----------+------------+----------+
| count(*) | count(fee) | max(fee) |
+----------+------------+----------+
|        2 |          0 |     NULL |
|        1 |          1 |     0.00 |
|        1 |          1 |    45.00 |
+----------+------------+----------+
```

5. Summary

Count (*) counts rows.

Count will always return a numeric answer- if the table is empty or there are no matching rows, it will return 0.

Count (prod_warranty_period) counts the values for prod_warranty_period, and does not count Nulls.

Sum, Avg, Max and Min will return nulls when there are no matching rows

Sum, Avg, Max and Min ignore Nulls when doing their calculations