Table of Contents

1. Getting aggregates by department

With each different dbms, you get a different selection of built-in features. Oracle and SQL Server have more built-in techniques to handle some of the tasks we are working on in this section. In MySQL we may have to do a bit more work to build a query to solve the problems. This is not a course in comparative dbms but I did want to show you one example of a technique that is available in ansi sql and how to do the same processing in MySQL.

Demo 01: This is the data we are working with.

```
Select emp_id, emp_name, dept_id, year_hired, salary
From a_testbed.adv_emp
Order by dept_id, emp_id;
+--------+--------------+---------+------------+--------+
| emp_id | emp_name     | dept_id | year_hired | salary |
+--------+--------------+---------+------------+--------+
|    101 | Hilbert      |      10 |       1980 |  45000 |
|    102 | Kovalevskaya |      20 |       1990 |  20000 |
|    104 | Gauss        |      20 |       2000 |  25000 |
|    105 | Hopper       |      20 |       2000 |  25000 |
|    106 | von Neumann  |      20 |       2000 |  28000 |
|    112 | Maxwell      |      20 |       2010 |  30900 |
|    120 | Cantor       |      20 |       2013 |  22000 |
|    103 | Euler        |      30 |       1990 |  28000 |
|    110 | Marcus       |      30 |       2008 |  45000 |
|    115 | Boole        |      30 |       2010 |  24000 |
|    116 | Carroll      |      30 |       2012 |  28000 |
|    117 | Church       |      30 |       2012 |  28000 |
|    107 | Maddy        |      45 |       2000 |  45000 |
|    108 | Pascal       |      45 |       2000 |  28000 |
|    109 | Boole        |      45 |       2008 |  32000 |
|    111 | Turing       |      45 |       2010 |  45000 |
|    113 | Lovelace     |      45 |       2010 |  45000 |
|    114 | Polya        |      45 |       2010 |  32000 |
|    118 | Neumann      |      45 |       2012 |  25000 |
|    119 | Wilkes       |      45 |       2013 |  25000 |
|    121 | Goedel       |      45 |       1995 |  28000 |
+--------+--------------+---------+------------+--------+
21 rows in set (0.00 sec)
```

Demo 02: These are aggregated values for each department.

```
Select dept_id, Sum(salary), Avg(salary)
From a_testbed.adv_emp
group by dept_id
Order by dept_id;
```

```
+---------+-------------+-------------+
| dept_id | Sum(salary) | Avg(salary) |
+---------+-------------+-------------+
|      10 |       45000 |  45000.0000 |
|      20 |      150900 |  25150.0000 |
|      30 |      153000 |  30600.0000 |
|      45 |      305000 |  33888.8889 |
+---------+-------------+-------------+
4 rows in set (0.00 sec)
```

2. Compare an employee to their department- limited to one department

We want to see how each individual employee's salary compares to their department's average salary. We will use just department 30 at first to reduce the output volume. This is the result we want. The average salary for dept 30 is 30600 and employee 110 earns more than that and the other employee salaries are less than the average.

```
    EMP_ID      SALARY    OVER_UNDER_AVG
----------- ----------- -----------------
       110       45000          14400.00
```

This is an ANSI standard query to do that- this does not work in MySQL . It uses the syntax Avg(salary) Over () to get the average salary for the rows.

```
Select emp_id, salary, salary - ( Avg(salary ) Over() ) as Over_under_avg
From adv_emp
Where dept_id = 30
```

These are some approaches to use in MySQL.

2.1. Using a subquery

Demo 03: Using a subquery in the Select clause. This query gets the average salary for department 30.

```
Select Avg(salary)
From a_testbed.adv_emp
Where dept_id = 30;
+-------------+
| Avg(salary) |
+-------------+
|  30600.0000 |
+-------------+
```

Demo 04: Use that query as a subquery for the comparison. Subtract the avg salary for each row's salary value.

```
Select emp_id, salary
, salary - (
    Select Avg(salary)
    From a_testbed.adv_emp
    Where dept_id = 30
    )  as Over_under_avg
From a_testbed.adv_emp
Where dept_id = 30
Order by emp_id;
+--------+--------+----------------+
| emp_id | salary | Over_under_avg |
+--------+--------+----------------+
|    103 |  28000 |     -2600.0000 |
|    110 |  45000 |     14400.0000 |
|    115 |  24000 |     -6600.0000 |
|    116 |  28000 |     -2600.0000 |
|    117 |  28000 |     -2600.0000 |
+--------+--------+----------------+
5 rows in set (0.00 sec)
```

2.2. Using a variable

The following uses a technique we have not used before. The value for avg salary for dept 30 is constant for the life of the query. So we could determine that value once and assign it to a variable and use the variable in the query.

We can assign a value from a table to a variable using a select query; take care that the query returns a single value only since a variable can hold only a single value.

Demo 05: Using a variable

```
set @avgsal = (
    Select Avg(salary)
    From a_testbed.adv_emp
    Where dept_id = 30);

Select
  emp_id, salary
, salary - @avgsal  as Over_under_avg
From a_testbed.adv_emp
Where dept_id = 30
Order by emp_id;
+--------+--------+--------------------------------------+
| emp_id | salary | Over_under_avg                       |
+--------+--------+--------------------------------------+
|    103 |  28000 | -2600.000000000000000000000000000000 |
|    110 |  45000 | 14400.000000000000000000000000000000 |
|    115 |  24000 | -6600.000000000000000000000000000000 |
|    116 |  28000 | -2600.000000000000000000000000000000 |
|    117 |  28000 | -2600.000000000000000000000000000000 |
+--------+--------+--------------------------------------+
```

2.3. Using a cross join

We could also do a cross join between a subquery that calculates the average and the a_testbed.adv_emp view. Be sure you understand why a cross join will work here.

Demo 06: Using a cross join and a subquery

```
Select
  emp_id, salary
, (salary - AvgDept30) as Over_under_avg
From a_testbed.adv_emp
cross join  (
    Select Avg(salary )  as AvgDept30
    From a_testbed.adv_emp
    Where dept_id = 30 ) avgSal
Where dept_id = 30
Order by emp_id;
```

At this point you might wonder which approach to use. I did a comparison of these three queries, and the query using the avg() Over() technique, using SQL Server, and the first three queries were about the same in terms of efficiency and the avg() Over() technique was almost twice as fast. Often when a dbms introduces a new technique they can implement the technique in a way that is efficient but if you need to write sql that is more cross-platform, then being able to build the query from the more common query components is very useful.

2.4. Comparison for all departments

Now we want to expand that query for all the departments, not just dept 30. For comparison this is the technique using the aggregate(col) Over ( ) technique. The code says to partition by the department so we get an average for each department. This does not work in MySQL

```
Select emp_id, dept_id, salary
       , salary - ( Avg(salary) Over( Partition by dept_id) ) as Over_under_avg
From a_testbed.adv_emp
Order by dept_id salary;
```

Demo 07: This demo is incorrect. This simply removed the filter for the department id. Before you read further try to figure out why this is wrong and how the output is not what we want.
Remember that a syntactically correct query will produce output but that does not mean it produces the output we want.

```
Select
  emp_id, dept_id, salary
, salary - (
    Select Avg(salary)
    From a_testbed.adv_emp
    )  as Over_under_avg
From a_testbed.adv_emp
Order by dept_id, emp_id;
+--------+---------+--------+----------------+
| emp_id | dept_id | salary | Over_under_avg |
+--------+---------+--------+----------------+
|    101 |      10 |  45000 |     13861.9048 |
|    102 |      20 |  20000 |    -11138.0952 |
|    104 |      20 |  25000 |     -6138.0952 |
|    105 |      20 |  25000 |     -6138.0952 |
|    106 |      20 |  28000 |     -3138.0952 |
|    112 |      20 |  30900 |      -238.0952 |
|    120 |      20 |  22000 |     -9138.0952 |
|    103 |      30 |  28000 |     -3138.0952 |
|    110 |      30 |  45000 |     13861.9048 |
|    115 |      30 |  24000 |     -7138.0952 |
|    116 |      30 |  28000 |     -3138.0952 |
|    117 |      30 |  28000 |     -3138.0952 |
|    107 |      45 |  45000 |     13861.9048 |
|    108 |      45 |  28000 |     -3138.0952 |
|    109 |      45 |  32000 |       861.9048 |
|    111 |      45 |  45000 |     13861.9048 |
|    113 |      45 |  45000 |     13861.9048 |
|    114 |      45 |  32000 |       861.9048 |
|    118 |      45 |  25000 |     -6138.0952 |
|    119 |      45 |  25000 |     -6138.0952 |
|    121 |      45 |  28000 |     -3138.0952 |
+--------+---------+--------+----------------+
21 rows in set (0.00 sec)
```

Note the results for dept 30- these are not the comparison of dept 30's employees to dept 30's average salary. This compares the employee's salary to the average for all employees- that is not a bad query; it is just not what we are trying to do.

Demo 08: We want to compare the salary to the average for this department. This uses a correlated subquery.

```
Select
  dept_id, emp_id, salary
, salary - (
    Select avg(salary)
    From a_testbed.adv_emp
    Where dept_id = OTR.dept_id
    ) as Over_under_avg
From a_testbed.adv_emp OTR
Order by dept_id, emp_id
;
+---------+--------+--------+----------------+
| dept_id | emp_id | salary | Over_under_avg |
+---------+--------+--------+----------------+
|      10 |    101 |  45000 |         0.0000 |
|      20 |    102 |  20000 |     -5150.0000 |
|      20 |    104 |  25000 |      -150.0000 |
|      20 |    105 |  25000 |      -150.0000 |
|      20 |    106 |  28000 |      2850.0000 |
|      20 |    112 |  30900 |      5750.0000 |
|      20 |    120 |  22000 |     -3150.0000 |
|      30 |    103 |  28000 |     -2600.0000 |
|      30 |    110 |  45000 |     14400.0000 |
|      30 |    115 |  24000 |     -6600.0000 |
|      30 |    116 |  28000 |     -2600.0000 |
|      30 |    117 |  28000 |     -2600.0000 |
|      45 |    107 |  45000 |     11111.1111 |
|      45 |    108 |  28000 |     -5888.8889 |
|      45 |    109 |  32000 |     -1888.8889 |
|      45 |    111 |  45000 |     11111.1111 |
|      45 |    113 |  45000 |     11111.1111 |
|      45 |    114 |  32000 |     -1888.8889 |
|      45 |    118 |  25000 |     -8888.8889 |
|      45 |    119 |  25000 |     -8888.8889 |
|      45 |    121 |  28000 |     -5888.8889 |
+---------+--------+--------+----------------+
21 rows in set (0.00 sec)
```

Demo 09: This uses a join instead of a correlated subquery

```
Select EmpLevel.dept_id, emp_id, salary, salary - avgSalary
From a_testbed.adv_emp as EmpLevel
Join (
    Select dept_id, avg(salary)  as avgSalary
    From a_testbed.adv_emp
    Group by dept_id
    ) as DeptLevel on EmpLevel.dept_id = DeptLevel.dept_Id
Order by dept_id, emp_id;
```

Now we want to know what percentage of the total salary for a department is earned by each employee.

Demo 10: Using the correlated subquery approach, we can calculate the sum(salary) for each department and divide an individual employee's salary by the sum for their department.

```
Select
  dept_id,emp_id,  salary
, salary / (
     Select sum(salary)
     From a_testbed.adv_emp
     Where dept_id = OTR.dept_id
     ) as Over_under_avg
From a_testbed.adv_emp OTR
Order by dept_id, emp_id;
+---------+--------+--------+----------------+
| dept_id | emp_id | salary | Over_under_avg |
+---------+--------+--------+----------------+
|      10 |    101 |  45000 |         1.0000 |
|      20 |    102 |  20000 |         0.1325 |
|      20 |    104 |  25000 |         0.1657 |
|      20 |    105 |  25000 |         0.1657 |
|      20 |    106 |  28000 |         0.1856 |
|      20 |    112 |  30900 |         0.2048 |
|      20 |    120 |  22000 |         0.1458 |
|      30 |    103 |  28000 |         0.1830 |
|      30 |    110 |  45000 |         0.2941 |
|      30 |    115 |  24000 |         0.1569 |
|      30 |    116 |  28000 |         0.1830 |
|      30 |    117 |  28000 |         0.1830 |
|      45 |    107 |  45000 |         0.1475 |
|      45 |    108 |  28000 |         0.0918 |
|      45 |    109 |  32000 |         0.1049 |
|      45 |    111 |  45000 |         0.1475 |
|      45 |    113 |  45000 |         0.1475 |
|      45 |    114 |  32000 |         0.1049 |
|      45 |    118 |  25000 |         0.0820 |
|      45 |    119 |  25000 |         0.0820 |
|      45 |    121 |  28000 |         0.0918 |
+---------+--------+--------+----------------+
21 rows in set (0.00 sec)
```

Demo 11: You can add a round function and multiplication to make the last column display as a percentage

```
Select
   dept_id,emp_id,  salary
,  round(100 * salary / (
      Select sum(salary)
      From a_testbed.adv_emp
      Where dept_id = OTR.dept_id
      ),0
      ) as Percent
From a_testbed.adv_emp OTR
Order by dept_id, emp_id;
+---------+--------+--------+---------+
| dept_id | emp_id | salary | Percent |
+---------+--------+--------+---------+
|      10 |    101 |  45000 |     100 |
|      20 |    102 |  20000 |      13 |
|      20 |    104 |  25000 |      17 |
|      20 |    105 |  25000 |      17 |
|      20 |    106 |  28000 |      19 |
|      20 |    112 |  30900 |      20 |
|      20 |    120 |  22000 |      15 |
|      30 |    103 |  28000 |      18 |
|      30 |    110 |  45000 |      29 |
|      30 |    115 |  24000 |      16 |
|      30 |    116 |  28000 |      18 |
|      30 |    117 |  28000 |      18 |
|      45 |    107 |  45000 |      15 |
|      45 |    108 |  28000 |       9 |
|      45 |    109 |  32000 |      10 |
|      45 |    111 |  45000 |      15 |
|      45 |    113 |  45000 |      15 |
|      45 |    114 |  32000 |      10 |
|      45 |    118 |  25000 |       8 |
|      45 |    119 |  25000 |       8 |
|      45 |    121 |  28000 |       9 |
+---------+--------+--------+---------+
```

Start by looking at the results for dept 10. There is one employee, with a salary of 45000. This row reports as 100% of the department salary total. Then look at the results for dept 30. There are five employees. The total salary for dept 30 is 153000. Employee 103 has a salary of 28000 which is 18% of the department total salary. Employee 110 has a salary of 45000 which is 29% of the department total salary.

Demo 12: This uses the join technique.

```
Select
  EmpLevel.dept_id
, emp_id, salary
, TotDeptSalary
, Round(100 * salary/totDeptSalary, 0) as PercOfDept
From a_testbed.adv_emp as EmpLevel
Join (
   Select dept_id, sum(salary)  as TotDeptSalary
   From a_testbed.adv_emp
   Group by dept_id)
   as DeptLevel
on EmpLevel.dept_id = DeptLevel.dept_Id
Order by EmpLevel.dept_id, EmpLevel.emp_id  ;
```

Although these queries all worked with the employees data and organizing by department, you should be able to see these as applied to other types of analysis.

- what percentage of sales do we get from the different products we sell?
- which customers have a total sales less than the average total sales for a customer in their zip code region?
- how do sales by month compare to sales for the whole year?