## 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, name_last, mng, dept_id, year_hired, salary
from a_emp.adv_emp
order by dept_id, emp_id;
```

| emp_id | name_last | mng | dept_id | year_hired | salary |
|---|---|---|---|---|---|
| 100 | King | NULL | 10 | 1989 | 24000 |
| 201 | Harts | 100 | 20 | 2004 | 15000 |
| 101 | Koch | 100 | 30 | 2008 | 98005 |
| 108 | Green | 101 | 30 | 1995 | 12000 |
| 109 | Fiet | 108 | 30 | 2012 | 15000 |
| 110 | Chen | 108 | 30 | 2012 | 30300 |
| 203 | Mays | 101 | 30 | 2010 | 44450 |
| 204 | King | 205 | 30 | 2011 | 15000 |
| 205 | Higgs | 101 | 30 | 2008 | 15000 |
| 206 | Geitz | 205 | 30 | 2011 | 88954 |
| 162 | Holme | 101 | 35 | 2011 | 98000 |
| 200 | Whale | 101 | 35 | 2011 | 65000 |
| 207 | Russ | 145 | 35 | 2011 | 65000 |
| 145 | Russ | 100 | 80 | 2008 | 65000 |
| 150 | Tuck | 145 | 80 | 2001 | 6500 |
| 155 | Hiller | 145 | 80 | 2004 | 80000 |
| 103 | Hunol | 102 | 210 | 2010 | 9000 |
| 104 | Ernst | 103 | 210 | 2012 | 50000 |
| 102 | D'Haa | 100 | 215 | 2010 | 30300 |
| 146 | Partne | 100 | 215 | 2012 | 88954 |
| 160 | Dorna | 146 | 215 | 2011 | 15000 |
| 161 | Dewal | 146 | 215 | 2011 | 15000 |

Demo 02: These are aggregated values for each department.

```
Select dept_id, Sum(salary), Avg(salary)
from a_emp.adv_emp
group by dept_id
order by dept_id;
```

```
+---------+-------------+-------------+
| dept_id | Sum(salary) | Avg(salary) |
+---------+-------------+-------------+
|      10 |       24000 |  24000.0000 |
|      20 |       15000 |  15000.0000 |
|      30 |      318709 |  39838.6250 |
|      35 |      228000 |  76000.0000 |
|      80 |      151500 |  50500.0000 |
|     210 |       59000 |  29500.0000 |
|     215 |      149254 |  37313.5000 |
+---------+-------------+-------------+
```

# 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 215 at first to reduce the output volume. This is the result we want. The average salary for dept 215 is 37313 and employee 146 earns a lot more than that and the other employee salaries are less than the average.(I guess it is good to be a manager!)

```
     EMP_ID      SALARY    OVER_UNDER_AVG
---------- ----------- -----------------
        102       30300           -7013.5
        146       88954           51640.5
        160       15000          -22313.5
        161       15000          -22313.5
```

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 = 215
```

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 215.

```
select Avg(salary)
from a_emp.adv_emp
where dept_id = 215;
+-------------+
| Avg(salary) |
+-------------+
|  37313.5000 |
+-------------+
```

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_emp.adv_emp
           where dept_id = 215
         )  as Over_under_avg
from    a_emp.adv_emp
where   dept_id = 215;
```

```
+--------+--------+----------------+
| emp_id | salary | Over_under_avg |
+--------+--------+----------------+
|    102 |  30300 |     -7013.5000 |
|    146 |  88954 |     51640.5000 |
|    160 |  15000 |    -22313.5000 |
|    161 |  15000 |    -22313.5000 |
+--------+--------+----------------+
```

## 2.2. Using a variable

The following uses a technique we have not used before. The value for avg salary for dept 215 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_emp.adv_emp
    where dept_id = 215);

select
   emp_id, salary
,  salary - @avgsal  as Over_under_avg
from a_emp.adv_emp
where dept_id = 215;
+--------+--------+---------------------------------------+
| emp_id | salary | Over_under_avg                        |
+--------+--------+---------------------------------------+
|    102 |  30300 |  -7013.5000000000000000000000000000 |
|    146 |  88954 |  51640.5000000000000000000000000000 |
|    160 |  15000 | -22313.5000000000000000000000000000 |
|    161 |  15000 | -22313.5000000000000000000000000000 |
+--------+--------+---------------------------------------+
```

## 2.3. Using a cross join

We could also do a cross join between a subquery that calculates the average and the a_emp.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 - AvgDept215) as Over_under_avg
from a_emp.adv_emp
cross join  (
    select Avg(salary )  as AvgDept215
    from   a_emp.adv_emp
    where dept_id = 215 ) avgSal
where dept_id = 215;
```

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 215. 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_emp.adv_emp
order by dept_id salary;
```

This is the desired output

| EMP_ID | DEPT_ID | SALARY | OVER_UNDER_AVG |
|---|---|---|---|
| 100 | 10 | 24000 | 0 |
| 201 | 20 | 15000 | 0 |
| 108 | 30 | 12000 | -27838 |
| 109 | 30 | 15000 | -24838 |
| 204 | 30 | 15000 | -24838 |
| 205 | 30 | 15000 | -24838 |
| 110 | 30 | 30300 | -9538 |
| 203 | 30 | 44450 | 4612 |
| 206 | 30 | 88954 | 49116 |
| 101 | 30 | 98005 | 58167 |
| 207 | 35 | 65000 | -11000 |
| 200 | 35 | 65000 | -11000 |
| 162 | 35 | 98000 | 22000 |
| 150 | 80 | 6500 | -44000 |
| 145 | 80 | 65000 | 14500 |
| 155 | 80 | 80000 | 29500 |
| 103 | 210 | 9000 | -20500 |
| 104 | 210 | 50000 | 20500 |
| 160 | 215 | 15000 | -22313 |
| 161 | 215 | 15000 | -22313 |
| 102 | 215 | 30300 | -7013 |
| 146 | 215 | 88954 | 51641 |

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_emp.adv_emp
      )  as Over_under_avg
from  a_emp.adv_emp
order by dept_id, salary;
+--------+---------+--------+----------------+
| emp_id | dept_id | salary | Over_under_avg |
+--------+---------+--------+----------------+
|    100 |      10 |  24000 |    -18975.5909 |
|    201 |      20 |  15000 |    -27975.5909 |
```

```
|    108 |      30 |  12000 |    -30975.5909 |
|    205 |      30 |  15000 |    -27975.5909 |
|    204 |      30 |  15000 |    -27975.5909 |
|    109 |      30 |  15000 |    -27975.5909 |
|    110 |      30 |  30300 |    -12675.5909 |
|    203 |      30 |  44450 |      1474.4091 |
|    206 |      30 |  88954 |     45978.4091 |
|    101 |      30 |  98005 |     55029.4091 |
|    200 |      35 |  65000 |     22024.4091 |
|    207 |      35 |  65000 |     22024.4091 |
|    162 |      35 |  98000 |     55024.4091 |
|    150 |      80 |   6500 |    -36475.5909 |
|    145 |      80 |  65000 |     22024.4091 |
|    155 |      80 |  80000 |     37024.4091 |
|    103 |     210 |   9000 |    -33975.5909 |
|    104 |     210 |  50000 |      7024.4091 |
|    161 |     215 |  15000 |    -27975.5909 |
|    160 |     215 |  15000 |    -27975.5909 |
|    102 |     215 |  30300 |    -12675.5909 |
|    146 |     215 |  88954 |     45978.4091 |
+--------+---------+--------+----------------+
```

Note the results for dept 215- these are not the comparison of dept 215's employees to dept 215'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_emp.adv_emp
       where dept_id = OTR.dept_id
       ) as Over_under_avg
from  a_emp.adv_emp OTR
order by dept_id, salary
;
+---------+--------+--------+----------------+
| dept_id | emp_id | salary | Over_under_avg |
+---------+--------+--------+----------------+
|      10 |    100 |  24000 |         0.0000 |
|      20 |    201 |  15000 |         0.0000 |
|      30 |    108 |  12000 |    -27838.6250 |
|      30 |    205 |  15000 |    -24838.6250 |
|      30 |    204 |  15000 |    -24838.6250 |
|      30 |    109 |  15000 |    -24838.6250 |
|      30 |    110 |  30300 |     -9538.6250 |
|      30 |    203 |  44450 |      4611.3750 |
|      30 |    206 |  88954 |     49115.3750 |
|      30 |    101 |  98005 |     58166.3750 |
|      35 |    200 |  65000 |    -11000.0000 |
|      35 |    207 |  65000 |    -11000.0000 |
|      35 |    162 |  98000 |     22000.0000 |
|      80 |    150 |   6500 |    -44000.0000 |
|      80 |    145 |  65000 |     14500.0000 |
|      80 |    155 |  80000 |     29500.0000 |
|     210 |    103 |   9000 |    -20500.0000 |
```

```
|     210 |    104 |  50000 |     20500.0000 |
|     215 |    161 |  15000 |    -22313.5000 |
|     215 |    160 |  15000 |    -22313.5000 |
|     215 |    102 |  30300 |     -7013.5000 |
|     215 |    146 |  88954 |     51640.5000 |
+---------+--------+--------+----------------+
```

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

```
select EmpLevel.dept_id, emp_id, salary, salary - avgSalary
from a_emp.adv_emp as EmpLevel
join (
    select  dept_id, avg(salary)  as avgSalary
    from  a_emp.adv_emp
    group by dept_id
    ) as DeptLevel on EmpLevel.dept_id = DeptLevel.dept_Id
order by dept_id, salary;
```

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_emp.adv_emp
     where dept_id = OTR.dept_id
     ) as Over_under_avg
from  a_emp.adv_emp OTR
order by dept_id, salary;
+---------+--------+--------+----------------+
| dept_id | emp_id | salary | Over_under_avg |
+---------+--------+--------+----------------+
|      10 |    100 |  24000 |         1.0000 |
|      20 |    201 |  15000 |         1.0000 |
|      30 |    108 |  12000 |         0.0377 |
|      30 |    205 |  15000 |         0.0471 |
|      30 |    204 |  15000 |         0.0471 |
|      30 |    109 |  15000 |         0.0471 |
|      30 |    110 |  30300 |         0.0951 |
|      30 |    203 |  44450 |         0.1395 |
|      30 |    206 |  88954 |         0.2791 |
|      30 |    101 |  98005 |         0.3075 |
|      35 |    200 |  65000 |         0.2851 |
|      35 |    207 |  65000 |         0.2851 |
|      35 |    162 |  98000 |         0.4298 |
|      80 |    150 |   6500 |         0.0429 |
|      80 |    145 |  65000 |         0.4290 |
|      80 |    155 |  80000 |         0.5281 |
|     210 |    103 |   9000 |         0.1525 |
|     210 |    104 |  50000 |         0.8475 |
|     215 |    161 |  15000 |         0.1005 |
|     215 |    160 |  15000 |         0.1005 |
|     215 |    102 |  30300 |         0.2030 |
|     215 |    146 |  88954 |         0.5960 |
+---------+--------+--------+----------------+
```

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_emp.adv_emp
      where dept_id = OTR.dept_id
      ),0
      ) as Percent
from  a_emp.adv_emp OTR
order by dept_id, salary;
+---------+--------+--------+---------+
| dept_id | emp_id | salary | Percent |
+---------+--------+--------+---------+
|      10 |    100 |  24000 |     100 |
|      20 |    201 |  15000 |     100 |
|      30 |    108 |  12000 |       4 |
|      30 |    205 |  15000 |       5 |
|      30 |    204 |  15000 |       5 |
|      30 |    109 |  15000 |       5 |
|      30 |    110 |  30300 |      10 |
|      30 |    203 |  44450 |      14 |
|      30 |    206 |  88954 |      28 |
|      30 |    101 |  98005 |      31 |
|      35 |    200 |  65000 |      29 |
|      35 |    207 |  65000 |      29 |
|      35 |    162 |  98000 |      43 |
|      80 |    150 |   6500 |       4 |
|      80 |    145 |  65000 |      43 |
|      80 |    155 |  80000 |      53 |
|     210 |    103 |   9000 |      15 |
|     210 |    104 |  50000 |      85 |
|     215 |    161 |  15000 |      10 |
|     215 |    160 |  15000 |      10 |
|     215 |    102 |  30300 |      20 |
|     215 |    146 |  88954 |      60 |
+---------+--------+--------+---------+
```

Start by looking at the results for dept 10. There is one employee, with a salary of 24000. This row reports as 100% of the department salary total. Then look at the results for dept 210. There are two employees. The total salary for dept 210 is 5900 (9000+50,000). Employee 103 has a salary of 9000 which is 15% of the department total salary. Employee 104 has a salary of 50000 which is 85% 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_emp.adv_emp as EmpLevel
join  (
   select dept_id, sum(salary)  as TotDeptSalary
   from a_emp.adv_emp
   group by dept_id)
   as DeptLevel
```

```
on EmpLevel.dept_id = DeptLevel.dept_Id
order by EmpLevel.dept_id, EmpLevel.salary  ;
```

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?

If you have not noticed this- you do need to be able to use subqueries and correlated subqueries to do serious work with sql.