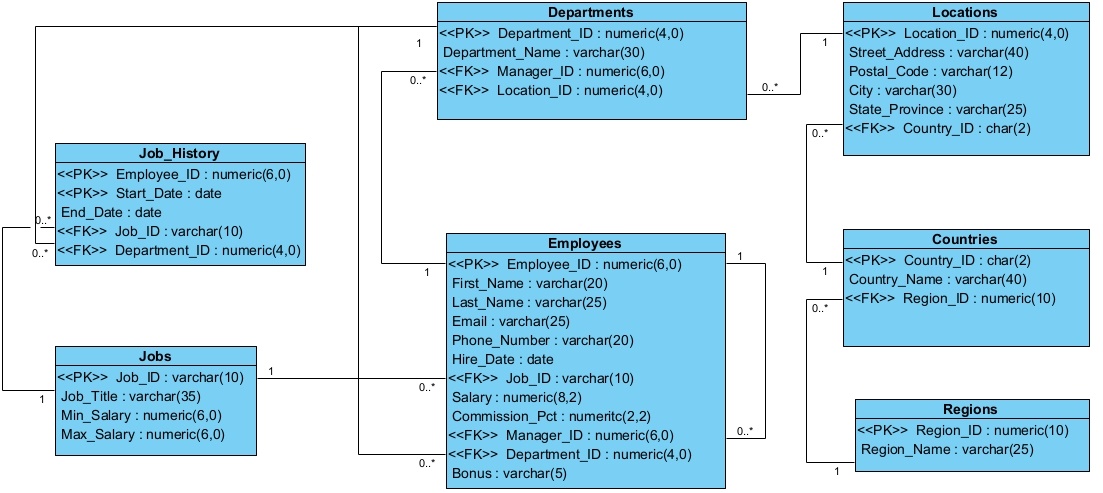
Databases 6G4Z0016 Labsheet

# Topic 5 – Subqueries



# Worked Examples

### Q1. List the employees earning more than the average for the company

The question asks us to return information about employees but only about some of them. When we want to only return some of the rows from a table we will use a WHERE clause. The difficulty in this case is that the value we want to compare each row to is a function of all the rows. We therefore have to use a subquery which can calculate the average salary and record that for use with the WHERE clause.

To design our query we first use a placeholder for the average salary and imagine it was a known constant. If we had that, then the query is a simple SELECT, FROM, WHERE query:

SELECT first\_name, last\_name

FROM Employees

WHERE salary > “average salary”;

Next, we design the query that would return the average salary. That is a simple group function, using the AVG function:

SELECT AVG(salary)

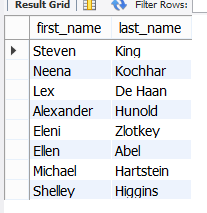
FROM Employees;

Finally, to complete the query we insert the subquery inside brackets, replacing the placeholder:

SELECT first\_name, last\_name

FROM Employees

WHERE salary > (SELECT AVG(salary) FROM Employees);

The result should be:  


### Q2. List the employees earning more than the average for their department

Once again, the question asks us to filter the results using a value that must be calculated from many rows. We therefore have to use a subquery to calculate the average salary for a given department.

As before, we design our query in two steps. First, we design the outer query, using a placeholder for the WHERE clause:

SELECT first\_name, last\_name

FROM Employees

WHERE salary > “average department salary”;

Next, we need to design the inner query, which, in this case, is the average salary of the employee’s department. The complicating factor is that we need to use information from the outer query inside the inner query. Specifically, we need to use the department\_id from the outer query to limit the inner query to the average of employees in the same department. Using information from the outer query as part of the inner query is called a *correlated subquery*. Since the table in the inner query is the same as the outer query we will have to use aliases.

Assume we called the Employees table in the outer query e1 then we can call the Employees table in the inner query e2 and now we can write:

SELECT AVG(salary)

FROM Employees AS e2

WHERE e1.department\_id = e2.department\_id;

Finally, we insert the inner query into the outer query in place of the placeholder and get:

SELECT first\_name, last\_name

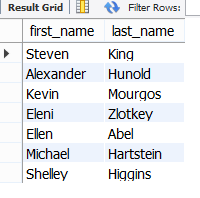
FROM Employees AS e1

WHERE salary > (SELECT AVG(salary)

FROM Employees AS e2

WHERE e1.department\_id = e2.department\_id);

The result should be:



### Q3. List the employees earning more than the total salary of any department

This question asks us to compare each employee’s salary to a collection of other values and return the row if the salary is greater than any of the values in the list provided. Again, we follow the same three-step process.

First, we design the outer query which is a little different in this case because of the list of values we expect back from the inner query. We want to return a row if salary is greater than any of the values so we use the ANY keyword:

SELECT first\_name, last\_name

FROM Employees

WHERE salary > ANY (“totals for departments”);

Next, we design a query to return the total salaries in each department. Since we are reducing many rows down to a single row (the total) we know we need a group function. In this case, it is the SUM function to add up all the values. We also know that we need one row per department so we need to use GROUP BY.

We add a WHERE clause to remove any employees who are not assigned to a department, because we’re interested in the total salary for each department. As we’re not including the department number itself, only the total salaries, without removing NULL departments, we would have a total for all those not in a department and we need to avoid that.

Therefore, the inner query is:

SELECT SUM(salary)

FROM Employees  
 WHERE department\_id IS NOT NULL

GROUP BY department\_id;

In previous weeks, we have been careful to join the Employees table with the Departments table so that we could use department\_name not department\_id. But, in this case we don’t need to worry about that. The only reason we wanted to return department\_name is because it has meaning to the user whereas IDs do not. But since the results from the inner query are not returned to the end user we do not need to be concerned with that.

Finally, we replace the placeholder in the outer query with our inner query and we get the final SQL:

SELECT first\_name, last\_name

FROM Employees

WHERE salary > ANY (SELECT SUM(salary)

FROM Employees  
 WHERE department\_ID IS NOT NULL

GROUP BY department\_id);

The results should be:



### Q4. List the employees whose salaries are within 10% of the average of the IT department

The question asks us to return some of the rows from the Employees table, but the criterion for returning a row depends on information from many other rows. This is another case when we need a subquery. We follow our usual steps.

First, create the outer query with a placeholder. The placeholder will stand for the average salary of the IT department, but that still leaves us a bit of work to decide whether an employee’s salary is within 10% of that value or not. A straightforward way of making the comparison is to provide two checks: that the salary is less than 10% higher and less than 10% lower. We would therefore have:

SELECT first\_name, last\_name

FROM Employees

WHERE salary <= “average it department” \* 1.1

AND salary >= “average it department” \* 0.9;

Next, we design the inner query, which is simply the average salary of employees in the IT department. We know this requires a group function – AVG. But we need to limit the rows that are included in the group function to only employees in the IT department. To do this a simple WHERE clause is sufficient:

SELECT AVG(salary)

FROM Employees

INNER JOIN Departments USING (department\_id)

WHERE department\_name = ‘IT’;

Finally, we need to insert the inner query into the outer query where the placeholders are. This is a straightforward process but in this case we would need to write out the subquery twice because we need the result of the subquery twice in the outer query.

To avoid this repetition, we can use a Common Table Expression using the WITH clause:

WITH average\_it\_department AS (

SELECT AVG(salary)

FROM Employees

INNER JOIN Departments USING (department\_id)

WHERE department\_name = ‘IT’)

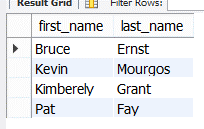
SELECT first\_name, last\_name

FROM Employees

WHERE salary <= (SELECT \* FROM average\_it\_department) \* 1.1

AND salary >= (SELECT \* FROM average\_it\_department) \* 0.9;

The result should be:



### Q5. List the employees whose salaries are above average but have worked at the company for less than the average

Once more, we know we will need a subquery because we want to filter rows based on information from multiple rows. The outer query requires two WHERE conditions, one for salary and one for the time worked at the company. Recall that to work out how much time has passed we use the TIMESTAMPDIFF function with CURDATE(). Therefore, the outer query would be:

SELECT first\_name, last\_name, salary,

TIMESTAMPDIFF(MONTH, hire\_date, CURDATE()) AS ‘Months Worked’

FROM Employees

WHERE salary > “average salary”

AND TIMESTAMPDIFF(MONTH, hire\_date, CURDATE()) < “average time worked”;

Next, we need to design the subquery. We have two different placeholders in the outer query which might make us think we need to use two different subqueries. But in fact, if we think about it, we realise that both the average salary and average time worked can be calculated in the same subquery. Namely:

SELECT AVG(salary) AS avg\_salary,

AVG(TIMESTAMPDIFF(MONTH, hire\_date, CURDATE()) AS avg\_time\_worked

FROM Employees;

Notice that we gave aliases to the two calculated columns in the subquery. What we need to do now is insert the subquery into the outer query.

In this case, we need to use the subquery twice because we have two placeholders to replace. Therefore, like in the previous question, we can use the WITH clause to avoid writing it out twice:

WITH avg\_sal\_time AS (

SELECT AVG(salary) AS avg\_salary,

AVG(TIMESTAMPDIFF(MONTH, hire\_date, CURDATE())) AS avg\_time\_worked

FROM Employees)

SELECT first\_name, last\_name, salary,

TIMESTAMPDIFF(MONTH, hire\_date, CURDATE()) AS ‘Months Worked’

FROM Employees

WHERE salary > “average salary”

AND TIMESTAMPDIFF(MONTH, hire\_date, CURDATE()) < “average time worked”;

We haven’t quite finished because we need to replace the placeholders with the columns that are inside the subquery.

The trick here is that when we “access” the subquery, we do not need to take every column. In all previous examples there has been only one column in the subquery so we simply said SELECT \* FROM subquery. In this case we have two columns and so we can specify which one we want on each line. This is why we provided convenient aliases for the columns.

The resultant SQL is:

WITH avg\_sal\_time AS (

SELECT AVG(salary) AS avg\_salary,

AVG(TIMESTAMPDIFF(MONTH, hire\_date, CURDATE())) AS avg\_time\_worked

FROM Employees)

SELECT first\_name, last\_name, salary,

TIMESTAMPDIFF(MONTH, hire\_date, CURDATE()) AS ‘Months Worked’

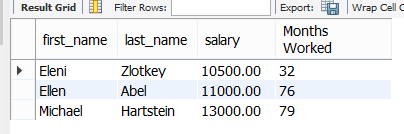
FROM Employees

WHERE salary > (SELECT avg\_salary FROM avg\_sal\_time)

AND TIMESTAMPDIFF(MONTH, hire\_date, CURDATE()) <

(SELECT avg\_time\_worked FROM avg\_sal\_time);

The result should be:



### Q6. List the employees who have the same manager and work in the same department as Curtis Davies (employee ID 142)

The question is asking us to filter rows based on information from the same table but on different rows, hence we need a subquery. Adopting the three-step process we first design the outer query. This is a straightforward SELECT and WHERE:

SELECT first\_name, last\_name, manager\_ID, department\_ID  
FROM Employees  
WHERE manager\_ID = “same as Curtis Davies”  
 AND department\_ID = “same as Curtis Davies”;

Next, we design the subquery. In this case we have two to design, but they are simple queries:

SELECT manager\_ID   
FROM Employees  
WHERE employee\_ID = 142;

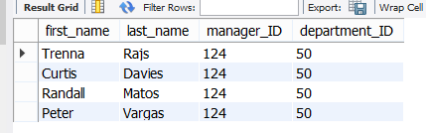
and

SELECT department\_ID  
FROM Employees  
WHERE employee\_ID = 142

Finally, we insert the subquery into the outer query in place of the placeholders:

SELECT first\_name, last\_name, manager\_ID, department\_ID  
FROM Employees  
WHERE manager\_ID = (SELECT manager\_ID   
 FROM Employees  
 WHERE employee\_ID = 142)  
 AND department\_ID = (SELECT department\_ID  
 FROM Employees  
 WHERE employee\_ID = 142);

The result is:



In this example, we have two subqueries and both use the same condition (“WHERE employee\_ID = 142”). Not only that, both are used with the same condition outside of them in the outer query (“=”). In this kind of situation we can combine them into a single subquery which will return a list of values. In this case the list would be manager\_ID and department\_ID. We can then combine the two values in the outer query into a list and compare one list to the other.

To make the manager\_ID and department\_ID into a list, we place them in brackets as a comma-separated list. The reworked query is:

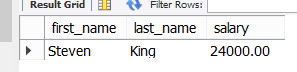
SELECT first\_name, last\_name, manager\_ID, department\_ID  
FROM Employees  
WHERE (manager\_ID, department\_ID) =   
 (SELECT manager\_ID, department\_ID   
 FROM Employees  
 WHERE employee\_ID = 142);

We can use this whenever we have multiple subqueries that are essentially the same (i.e. retrieve the same set of rows) and when the comparison to the subqueries are the same (i.e. both =, or IN etc).

# Completion Problems

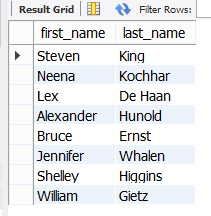
### Q1. List the employees who earn more than the average salary in every department.

SELECT first\_name, last\_name, salary   
FROM Employees   
WHERE salary > ALL (SELECT )



### Q2. List the employees who have worked at the company for longer than the average.

SELECT first\_name, last\_name   
FROM Employees  
WHERE DATEDIFF(CURDATE(), hire\_date)



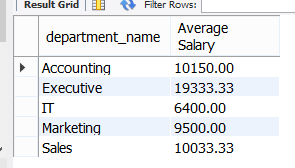
### Q3. List the employees where the length of their department name is below average.

SELECT first\_name, last\_name, department\_name   
FROM Employees   
INNER JOIN Departments USING (department\_ID)   
 (SELECT AVG(length(department\_name)) FROM Departments);



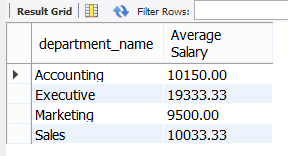
### Q4. List the departments where the average salary in that department is greater than 5000.

SELECT department\_name, ROUND(AVG(salary),2) AS ‘Average Salary’   
FROM Employees   
INNER JOIN Departments   
GROUP BY department\_name   
HAVING AVG(salary) > 5000;



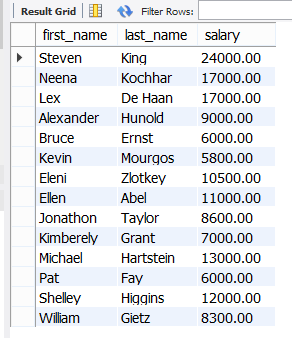
### Q5. List the departments whose average salary is above the average of all employees.

SELECT department\_name, ROUND(AVG(salary),2) AS ‘Average Salary’   
FROM Employees   
INNER JOIN Departments USING(department\_id)   
GROUP BY department\_name



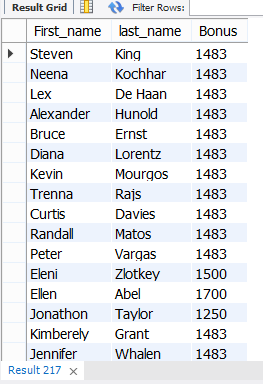
### Q6. List the employees whose salary is greater than the average salary among employees whose surnames end in the letter ‘s’.

SELECT first\_name, last\_name, salary   
FROM Employees   
WHERE salary >



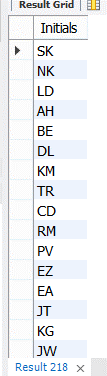
### Q7. List all the employees and their bonuses. For those without a bonus, show the average bonus instead.

SELECT First\_name, last\_name,   
 IFNULL( ) AS Bonus  
FROM Employees;



Q8. Generate a list of every employees’ initials

SELECT as Initials  
FROM Employees;



### Q9. List the countries with at least three employees working in it.

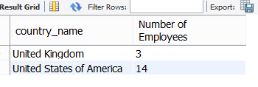
SELECT country\_name, COUNT(\*) AS ‘Number of Employees’

FROM Countries

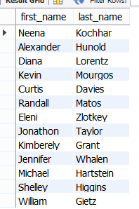
INNER JOIN Locations USING(country\_id)

INNER JOIN Departments USING(location\_id)

INNER JOIN Employees USING(department\_id)



### Q10. List the employees whose names (first name and surname combined) are longer than the average.

SELECT first\_name, last\_name  
FROM Employees  


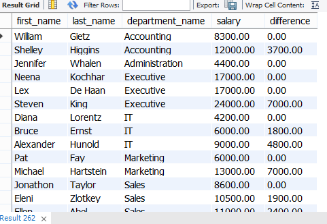
### Q11. List every employee, their department name, salary and the difference between their salary and the lowest in their department.

SELECT first\_name, last\_name, department\_name, salary,   
 salary - AS difference

FROM Employees AS e1

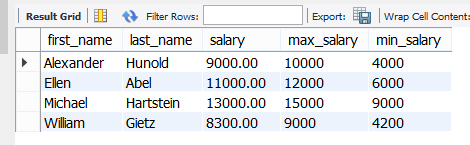
INNER JOIN Departments on e1.department\_id = Departments.department\_id

ORDER BY department\_name, salary;

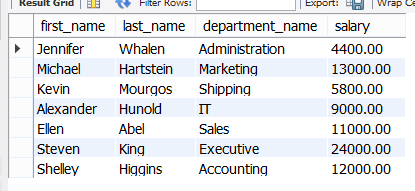


# Deliberate Practice: Write the SQL

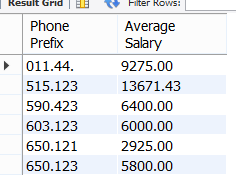
### Q1. List the employees earning more than the mid-range for their job.



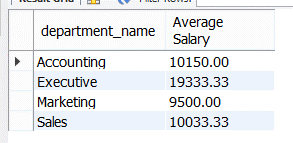
### Q2. List the employees earning the most in each department.



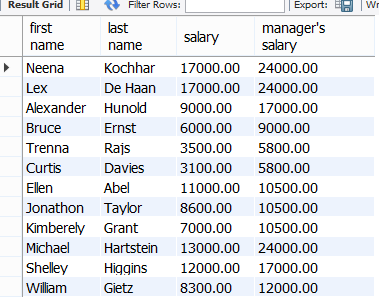
### Q3. For each telephone group (i.e. they share the same first six digits), give the average salary of the group.



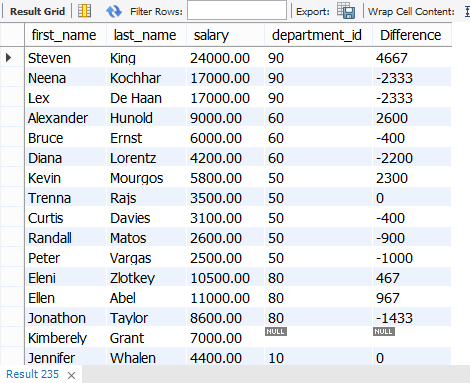
### Q4. List the average salary of departments whose average is above the company-wide average



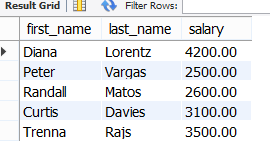
### Q5. List the employees earning more than half of their manager’s salary.



### Q6. List all employees, their salaries and the difference between their salary and the average salary for their department.



### Q7. List the employees who earn less than half of the average salary. Order them by how much their salary is below the maximum salary for their job.



### Q8. List the top three departments with the highest spread of actual wages

Graphical user interface, table

Description automatically generated

### Q9. CHALLENGE! List, for each employee, what their annual salary would be if they received an end of year bonus equal to the average of the bonuses paid out to those who receive them.



### Q10. CHALLENGE! List the employees earning the most of those hired in the same year as they were.

