**Guided Lab - 304.7.1: Subqueries**

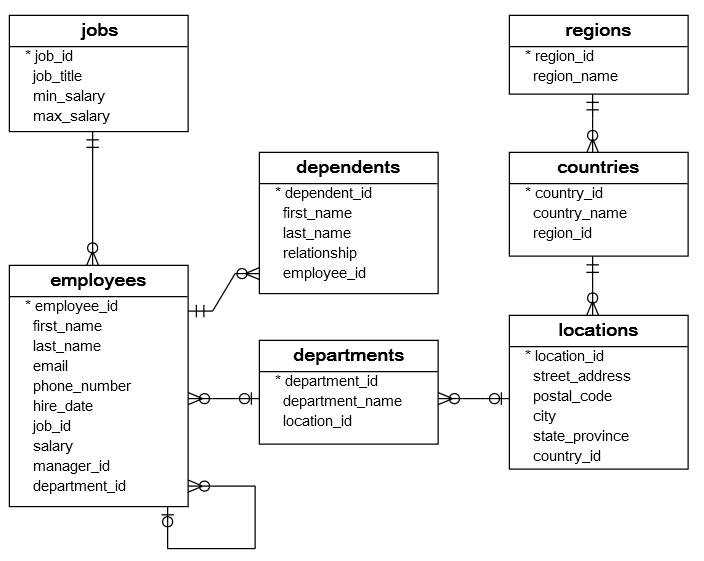
# **Lab Overview:**

In this lab, we will demonstrate the SQL "***Subquery***," and how to make our query more dynamic. By the end of this lab, you will be able to use the SQL Subquery.

**Prerequisites:**

For this lab, you must have an **HR sample database**. **Click to download the** [**HR sample**](https://drive.google.com/file/d/1KRA_hxMEYJQwgxd8dtUV1s6aMRCR2HbY/view?usp=sharing) **database.**

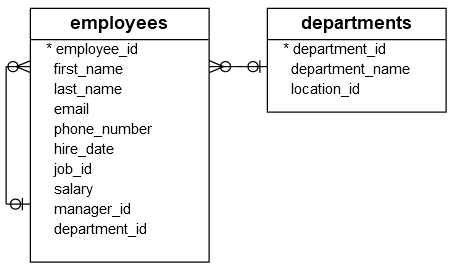
The following ER diagram illustrates the HR database Schema.



The HR sample database has seven tables:

1. The**employees** table stores the data of employees.
2. The **jobs** table stores the job data, including job titles and salary ranges.
3. The **departments** table stores departmental data.
4. The **dependents** table stores the employee’s dependents.
5. The **locations** table stores the locations of the departments of the company.
6. The **countries** table stores the data of countries where the company is doing business.
7. The **regions** table stores the data of regions (e.g., Asia, Europe, America, the Middle East, and Africa). The countries are grouped into regions.

Consider the following **employees** and **departments** tables.

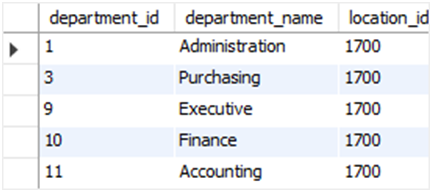


Suppose you have to find all employees who are located at the location with the ID 1700. You might come up with the following solution.

First, find all departments located at the **location\_id is 1700**:

**SELECT** \* **FROM** departments **WHERE** location\_id = 1700;

**Result:**



According to the result, the departments of 1, 3, 9, 10, and 11 are located at 1700 ID.

Secondly, to find employees that belong to the location 1700, we can use the ***department id*** list from the previous query, as depicted in the query below.

**SELECT employee\_id, first\_name, last\_name, department\_id**

**FROM employees**

**WHERE department\_id IN (1, 3, 9, 10, 11)**

**ORDER BY first\_name, last\_name;**

**Result:**

The above solution has two problems. To start with, you have looked at the ***department’s***table to check which department belongs to location 1700. However, the original question was not referring to any specific departments — it referred to location 1700. Because of the small data volume, you can easily get a list of departments. However, in a real system with high volumes of data, it might be problematic.

Another problem was that you had to revise the queries whenever you wanted to find employees who were located in a different location. A much better solution to this problem is to use a "subquery."

# 

# 

# **Example: Subquery with Where Clause**

## **Let’s utilize the IN or NOT IN operator.**

1. You can rewrite and combine the queries above as follows:

**SELECT employee\_id, first\_name, last\_name, department\_id**

**FROM employees**

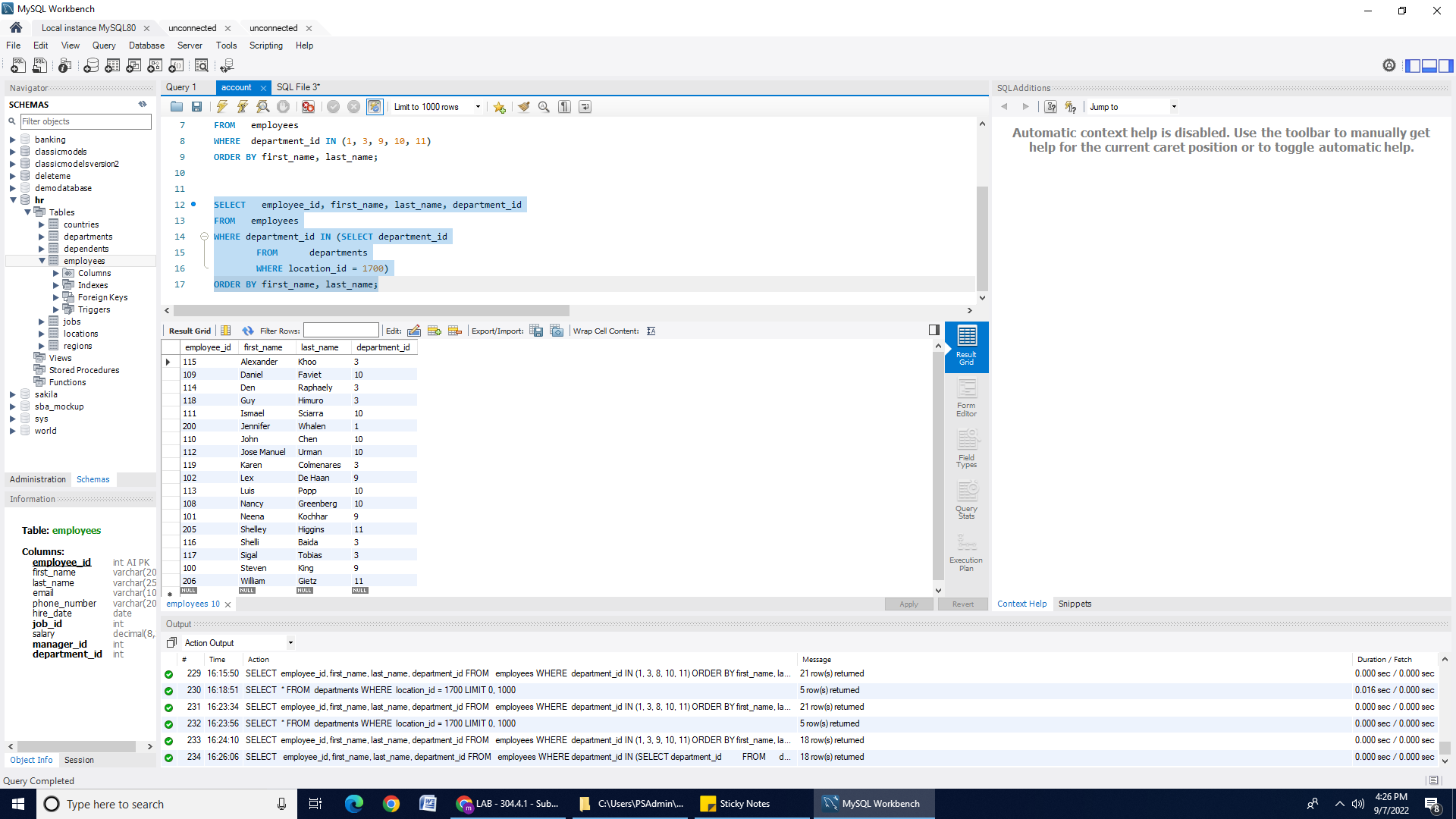
**WHERE department\_id IN (SELECT department\_id**

**FROM departments**

**WHERE location\_id = 1700)**

**ORDER BY first\_name, last\_name;**

**Result:**



1. The following example uses a subquery with the **NOT IN**operator to find all employees' information who *do not* locate at location 1700.

**SELECT** employee\_id, first\_name, last\_name

**FROM** employees

**WHERE** department\_id **NOT** **IN** (**SELECT** department\_id

**FROM** departments

**WHERE** location\_id = 1700)

**ORDER** **BY** first\_name , last\_name;

**Result:**



## 

## **Let's utilize the comparison operator (=, >, =, <= , !=)**

* 1. The following example finds the employees who have the highest salary:

**SELECT** employee\_id, first\_name, last\_name, salary

**FROM** employees

**WHERE** salary = (**SELECT** **MAX**(salary) **FROM** employees)

**ORDER** **BY** first\_name, last\_name;

**Result:**

SQL Subquery with the Equal operator

In the above subquery, the subquery returns the highest salary of all employees, and the outer query finds the employees whose salary is equal to the highest one.

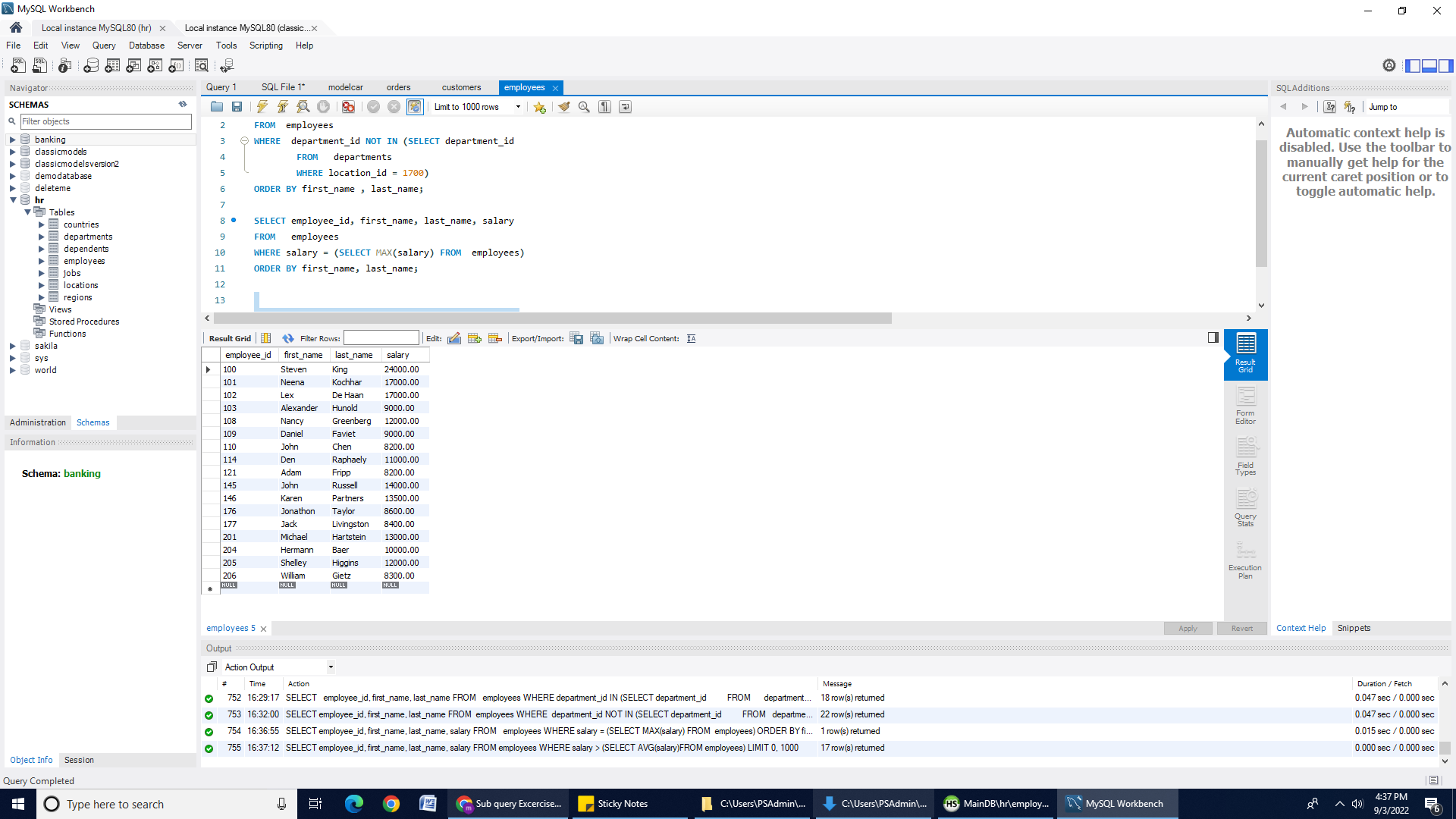
* 1. The following query finds the **employee number, name,** and **salary** for the employees whose salary is higher than the average salary throughout the company:

**SELECT** employee\_id, first\_name, last\_name, salary

**FROM** employees

**WHERE** salary > (**SELECT** **AVG**(salary)**FROM** employees);

**Result:**



In the above example, first, the subquery returns the average salary of all employees. Then the outer query uses the greater than operator ( > ) to find all employees whose salaries are greater than the average.

## **Let's utilize the EXISTS or NOT EXISTS operators.**

The **EXISTS**operator checks for the existence of rows returned from the subquery. It returns true if the subquery contains any rows. Otherwise, it returns false.

[Click here for more information](https://dev.mysql.com/doc/refman/8.0/en/exists-and-not-exists-subqueries.html).

* 1. The following example finds that all ***departments*** have at least one employee with a salary greater than 10,000. We will use the ***Where*** clause and the ***EXISTS*** operator.

**SELECT** department\_name

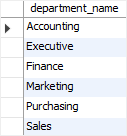
**FROM** departments d

**WHERE** **EXISTS** ( **SELECT** \* **FROM** employees e

**WHERE** salary > 10000 **AND** e.department\_id = d.department\_id)

**ORDER** **BY** department\_name;

**Result:**



1. Similarly, the following query finds all departments that do not have an employee with a salary greater than 10,000, We will use the ***Where clause,*** and the ***NOT EXISTS*** operator.

**SELECT** department\_name

**FROM** departments d

**WHERE NOT** **EXISTS** ( **SELECT** \* **FROM** employees e

**WHERE** salary > 10000 **AND** e.department\_id = d.department\_id) **ORDER** **BY** department\_name;

**Result:**

### 

# **SQL subquery in the FROM clause**

For this section, you must have a “**classicmodels”** database. If you do not have a ‘**classicmodels ‘** database setup, [**click here to download the database script file**.](https://drive.google.com/file/d/1JoT6N-kNhJ048ahXvvSgWNE0737NAdbb/view?usp=sharing)

**Syntax**: You can use a subquery in the FROM clause of the **SELECT** statement as follows:

**SELECT \* FROM (subquery) AS table\_name\_alias**

In the syntax above, the ***table alias***is mandatory because all tables in the FROM clause must have a name.

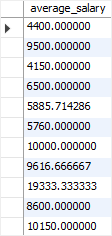
## **Example 1:**

The following query returns the average salary of every department:

**SELECT** **AVG**(salary) average\_salary

**FROM** employees **GROUP** **BY** department\_id;

**Result:**



You can use the above query as a subquery in the FROM clause to calculate the average salary of departments as follows:

**SELECT** **ROUND**( **AVG**(average\_salary), 0)

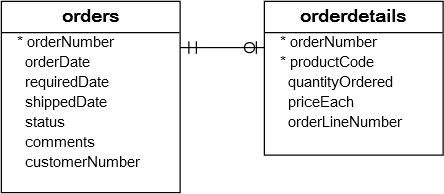
**FROM** ( **SELECT** **AVG**(salary) as average\_salary **FROM** employees **GROUP** **BY** department\_id) *department\_salary*;

**Result:**

SQL Subquery - average of average salary by department

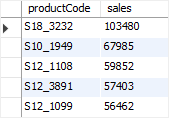
## **Example 2:**

The following query gets the top five products by sales revenue in 2003 from the ***orders*** and ***orderdetails*** tables.

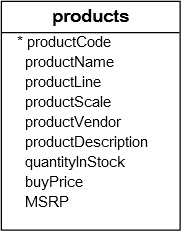


| SELECT productCode, ROUND(SUM(quantityOrdered \* priceEach)) AS sales  FROM orderdetails  INNER JOIN orders USING (orderNumber)  WHERE YEAR(shippedDate) = 2003  GROUP BY productCode  ORDER BY sales DESC  LIMIT 5; |
| --- |

**Result:**

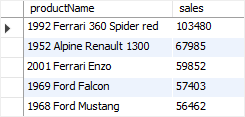


You can use the result of this query as a derived table and join it with the products table as follows:



| SELECT productName, sales  FROM (SELECT productCode, ROUND(SUM(quantityOrdered \* priceEach)) AS sales  FROM orderdetails INNER JOIN orders USING (orderNumber)  WHERE YEAR(shippedDate) = 2003  GROUP BY productCode  ORDER BY sales DESC  LIMIT 5) as top5products2003  INNER JOIN products USING (productCode); |
| --- |

The following shows the output of the query above:



In this example:

1. First, the subquery is executed to create a result set or derived table.
2. Then the outer query is executed that joins the ***top5product2003*** derived table with the products table using the ***productCode*** column.

**Canvas submission Instructions:** Please include the following deliverables in your submission -

* + All queries should be written and submitted in a single SQL script file.
    - Example :**<your\_name\_labname>.sql**.
  + Submit your SQL script file using the **Start** **Assignment** button in the top-right corner of the assignment page in Canvas.