Database II Lab/ 3rd Grade

[Second Semester] and [2026]

[Lab 3]

[17/2/ 2026] [8.30 + 10.30 am]

Instructor Information

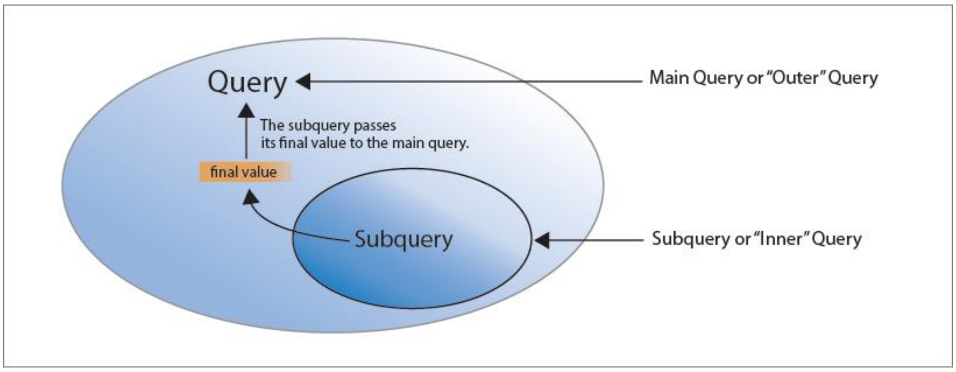
Instructor	Email	Hours
Dr. Rasool Hisham Dr. Zainab Namh Dr. Azhar Flaih Assist. Prof. Zahraa Jaaz	[rasool.hisham@nahrainuniv.edu.iq]	[2 Hrs]

Queries within Queries (Subquery)

The purpose of this lab is to understand and apply **subqueries** (queries within queries) in SQL. Students will learn how to use subqueries to retrieve specific data from a database by nesting one query inside another. By the end of this lab, students should be able to:

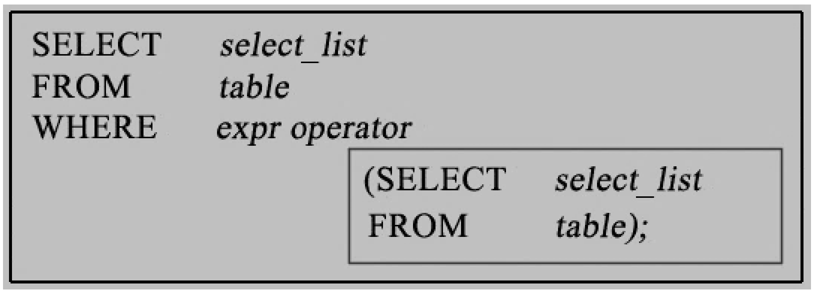
1. **Understand how subqueries pass values to the outer query.**
2. **Apply subqueries in SELECT, FROM and WHERE clauses.**
3. **Use subqueries to filter, aggregate, and compare data effectively**

A **subquery** (also known as a **nested query**) is a query that is placed inside another SQL query. It is used to retrieve data that will be used by the main (outer) query. Subqueries can appear in **SELECT**, and **WHERE** clauses.


Query
Main Query or "Outer" Query
The subquery passes its final value to the main query.
final value
Subquery
Subquery or "Inner" Query

1. Basic Subquery (Using WHERE Clause)

SQL usually use subqueries inside the WHERE clause to narrow down the result of the main query (Filtering the results)



```
SELECT    select_list
FROM      table
WHERE     expr operator
                        (SELECT   select_list
                         FROM     table);
```

Database II Lab/ 3rd Grade

[Second Semester] and [2026]

[Lab 3]

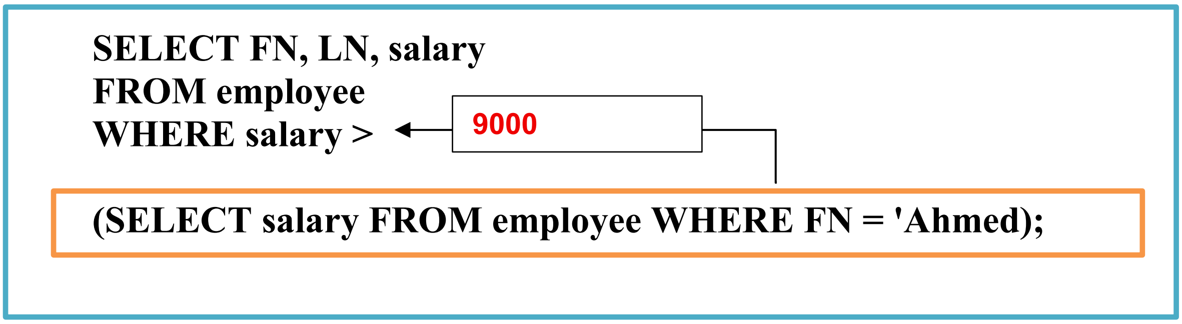
[17/2/ 2026] [8.30 + 10.30 am]

Instructor Information

Instructor	Email	Hours
Dr. Rasool Hisham Dr. Zainab Namh Dr. Azhar Flaih Assist. Prof. Zahraa Jaaz	[rasool.hisham@nahrainuniv.edu.iq]	[2 Hrs]

A. Basic Filtering (Single Value)

Find employees earning more than a specific person (e.g., "Ahmed").



```
SELECT FN, LN, salary
FROM employee
WHERE salary >
(SELECT salary FROM employee WHERE FN = 'Ahmed);
```

B. Using Aggregate Functions

Find customers information who are the youngest.

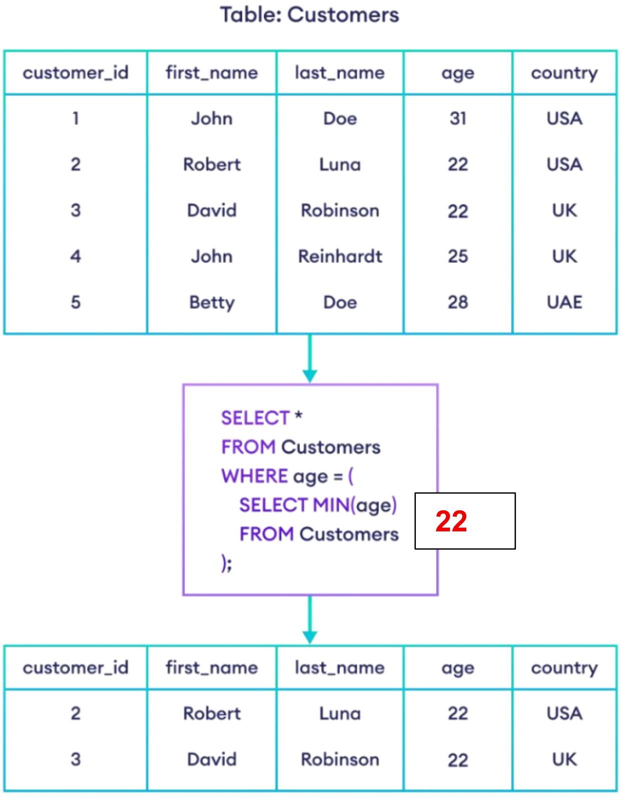


Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT *
FROM Customers
WHERE age = (
  SELECT MIN(age)
  FROM Customers
);
```

customer_id	first_name	last_name	age	country
2	Robert	Luna	22	USA
3	David	Robinson	22	UK

C. The IN Operator (Multiple Values)

Use IN when the subquery returns more than one result.

Find all products that have been ordered at least once. This filters the Products table based on IDs found in the Order_Items table.

```
SELECT product_name
FROM products
WHERE product_id IN (SELECT DISTINCT product_id FROM order_items);
```

Filters orders belonging to any customer located in UAE

```
SELECT  order_id, customer_id
FROM   orders
WHERE    customer_id IN
(SELECT   customer_id  FROM customer
WHERE  city = 'UAE' )
ORDER BY total DESC;
```

D. Using the ANY / SOME/ ALL/ EXISTS Operators

Operator	**Usage**	**Example Description**
EXISTS	Returns TRUE if the subquery returns any rows.	Find Departments that currently have at least one employee assigned to them.
ALL	True only if the comparison is true for **every** value in the list.	Find Employees who earn more than every single person working in Department 5.
ANY / SOME	True if the comparison is true for **at least one** value in the list.	Find Products with a price that is less than at least one product in the 'Electronics' category.

NOTE: A JOIN is not required whenever you use a subquery operator like:

- **EXISTS**
- **IN**
- **ANY / SOME**
- **ALL**

Instead, the comparison happens in the subquery's WHERE.

➢ Using the ANY Operator

This returns true if the value meets the criteria for any single row in the list.

Find employees who earn more than <u>at least one employee</u> in Department 3.

```
SELECT FN, salary
FROM employee
WHERE salary > ANY (SELECT salary FROM employee WHERE dno = 3);
```

➢ Using the ALL Operator

Used to compare a value to every value in a list.

Find employees who earn more than all employees in Department 1.

```
SELECT FN, salary
FROM employees
WHERE salary > ALL (SELECT salary FROM employee WHERE dno= 1);
```

➢ Using the EXISTS Operator

The EXISTS operator is used to test for the existence of any record in a subquery. It returns TRUE if the subquery returns one or more rows.

Tip: We usually use SELECT 1 inside EXISTS because the actual data doesn't matter—only the fact that a row exists.

Retrieve names of employees who are currently assigned to a project.

```
SELECT FN, LN
FROM employee e
WHERE EXISTS (SELECT 1 FROM works_on w WHERE w.empid = e.empid);
```

List the customer ID and first name of all customers who have made any orders.

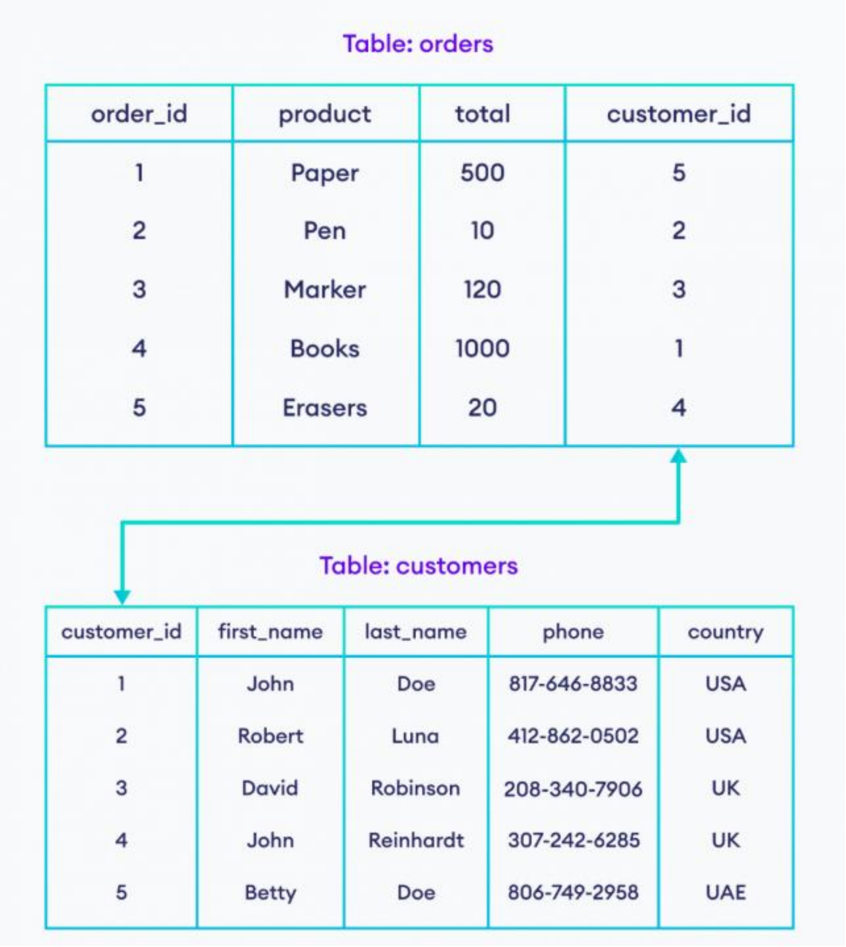


Table: orders

order_id	product	total	customer_id
1	Paper	500	5
2	Pen	10	2
3	Marker	120	3
4	Books	1000	1
5	Erasers	20	4

Table: customers

customer_id	first_name	last_name	phone	country
1	John	Doe	817-646-8833	USA
2	Robert	Luna	412-862-0502	USA
3	David	Robinson	208-340-7906	UK
4	John	Reinhardt	307-242-6285	UK
5	Betty	Doe	806-749-2958	UAE

2. Subquery in SELECT Clause

A subquery in the SELECT clause is used to compute a single value for each row of the main query. This is useful when you need to retrieve an additional piece of information related to each row without using a JOIN.

SYNTAX:

```
SELECT
   column1,
   (SELECT single_value FROM another_table WHERE condition) AS
alias_name
FROM
   main_table;
```

Retrieve an employee's salary alongside the company-wide average.

```
SELECT FN, salary,
(SELECT AVG(salary) FROM employee) AS average_pay
FROM employee;
```

3. Subquery in FROM Clause

A **subquery in the FROM clause**, often called a **derived table** whose results is treated as a temporary table by the outer query.

When you put a query inside the FROM clause, SQL executes that inner query first, stores the result in temporary memory, and then treats it exactly like a real table (like employees or orders) for the rest of the operation.

The Syntax Requirement: The Alias

In almost every SQL database (like PostgreSQL, MySQL), you must give a subquery in the FROM clause a name using **AS**. If you don't, the database won't know how to reference the columns.

SYNTAX:

```
SELECT t.column_name
FROM (
   SELECT ... -- This is the subquery
) AS t; -- 't' is the Alias (the name of your virtual table)
```

Find the average salary of only the 10 highest-paid people in the company.

- **Inner Layer: Get the top 10 salaries.**
- **Outer Layer: Calculate the average of those specific 10 rows.**

```
SELECT AVG(top_ten.salary) AS top_avg
FROM (
   SELECT salary
   FROM employee
   ORDER BY salary DESC
   LIMIT 10
) AS top_ten;
```

Find the average of the department's salary

The Scenario: The "Fair" Company Average

Imagine your company has two departments:

- Engineering: 100 employees (Average Salary: $150,000)
- HR: 2 employees (Average Salary: $50,000)

```
SELECT AVG(dept_table.avg_sal) AS department_unit_average
FROM (
   SELECT dno, AVG(salary) AS avg_sal
   FROM employee
   GROUP BY dno) AS dept_table;
```

Nested Subquery (Multi-Level)

You can nest a subquery inside another subquery to perform complex filtering.

Find employees who work in departments located in 'London'. Note suppose you have location table where each location has a specific id.

1. **Inner:** Get the location_id for 'London'.
2. **Middle:** Get department_ids associated with those location_ids.
3. **Outer:** Get employees in those departments.

```
SELECT FN, LN
FROM employee
WHERE dno IN (
   SELECT dno
   FROM department
   WHERE location_id = (
      SELECT location_id
      FROM locations
      WHERE city = 'London'
   )  );
```