🔁 Quick Decision Rules:

If Question Says...	Use
"Find employees who earn more than avg salary of their dept"	✅ Correlated Subquery
"Find all customers who placed orders"	✅ EXISTS
"Find employees in departments 10, 20, 30"	✅ Multi-Row Subquery (`IN`)
"Get department name with every employee"	✅ JOIN
"Find employee whose salary = max salary"	✅ Single-Row Subquery
"Compare each row to a group result"	✅ Correlated Subquery

✅ Summary: When to Use What?

Concept	Use When...	Typical Syntax / Use	Returns	Best Used For
🔹 **Single Row Subquery**	Subquery returns **only 1 value**	=, >, <, etc.	One value	Comparing against max/min/avg, filtering
🔹 **Multi Row Subquery**	Subquery returns **one column, multiple rows**	`IN`, `NOT IN`, `ANY`, `ALL`	One column, many rows	Filtering based on list of values
🔹 **Multi Row, Multi Column Subquery**	Subquery returns **multiple columns and rows**	`IN (SELECT col1, col2...)`, used with row constructors	Multiple columns & rows	Row-wise filtering, composite conditions
🔹 **Correlated Subquery**	Subquery **depends on outer row**	Subquery inside `WHERE` or `SELECT`, uses outer table alias	Varies (evaluated per row)	Per-row logic like group-wise comparison
🔹 **INNER JOIN**	You need **matching rows** from 2+ tables	`SELECT ... FROM A JOIN B ON A.x = B.x`	Combined rows	Fetching related data
🔹 **LEFT/RIGHT JOIN**	You need **non-matching rows too**	`LEFT JOIN`, `RIGHT JOIN`	Includes NULLs for missing match	Optional relationships, reporting all even if unmatched
🔹 **EXISTS / NOT EXISTS**	You want to check **presence/absence**	`WHERE EXISTS (SELECT ...)`	Boolean logic	Presence, efficient for large data

📌 Tip for Your Students:

- Use **`IN`** for simple matching
- Use **joins** when you need **columns from both tables**
- Use **correlated subqueries** when logic changes **per row**
- Use **`EXISTS`** when just checking presence is enough (faster than `IN` on big data)

✅ 1. "Find employees who earn more than avg salary of their dept"

Use: Correlated Subquery
🔍 Because average salary changes **per department**, you need a subquery that depends on each employee's `dept_id`.

```
SELECT e.employee_id, e.name, e.salary, e.dept_id
FROM employees e
WHERE e.salary > (
    SELECT AVG(e2.salary)
    FROM employees e2
    WHERE e2.dept_id = e.dept_id
);
```

✅ This compares each employee’s salary with the **average salary of their department**.

> List employees **who earn more than the average salary** of their own department.

🧾 Sample `employees` table:

employee_id	name	salary	dept_id
101	**Raj**	60,000	10
102	**Priya**	50,000	10
103	**Aarav**	40,000	10
104	**Neha**	70,000	20
105	**Rohan**	60,000	20
106	**Kavya**	90,000	30

🔄 Step-by-step Dry Run:

🔹 For Raj (Dept 10, Salary 60,000):

Subquery:

```
SELECT AVG(salary) FROM employees e2 WHERE dept_id = 10
```

→ (60,000 + 50,000 + 40,000) / 3 = 50,000

Check: `60,000 > 50,000` ✅ → Include Raj

🔹 For Priya (Dept 10, Salary 50,000):

Same AVG: 50,000
Check: `50,000 > 50,000` ❌ → Exclude

🔹 For Aarav (Dept 10, Salary 40,000):

Check: `40,000 > 50,000` ❌ → Exclude

🔹 For Neha (Dept 20, Salary 70,000):

Subquery:

```
SELECT AVG(salary) FROM employees WHERE dept_id = 20
```

→ (70,000 + 60,000) / 2 = 65,000

Check: `70,000 > 65,000` ✅ → Include Neha

🔹 For Rohan (Dept 20, Salary 60,000):

Check: `60,000 > 65,000` ❌ → Exclude

🔹 For Kavya (Dept 30, Salary 90,000):

Only one person in Dept 30 → AVG = 90,000
Check: `90,000 > 90,000` ❌ → Exclude

✅ Final Output:

employee_id	name	salary	dept_id
101	Raj	60,000	10
104	Neha	70,000	20

✅ 2. "Find all customers who placed orders"

Use: EXISTS
🔍 You just need to check if a matching order **exists**, not fetch it.

```
SELECT c.customer_id, c.name
FROM customers c
WHERE EXISTS (
    SELECT 1
    FROM orders o
    WHERE o.customer_id = c.customer_id
);
```

✅ `EXISTS` is faster than `IN` on large datasets and ignores duplicates.

🔄 Dry Run (How Oracle Executes It):

Suppose:

🧾 `customers` table:

customer_id	name
101	Alice
102	Bob
103	Carol

🧾 `orders` table:

order_id	customer_id
1	101
2	103

Step-by-step execution:

1. **Pick customer Alice (101)**
   → Run subquery: `SELECT 1 FROM orders o WHERE o.customer_id = 101`
   → Result: **Exists** ✅ → Include Alice
2. **Pick customer Bob (102)**
   → Run subquery: `SELECT 1 FROM orders o WHERE o.customer_id = 102`
   → Result: **No match** ❌ → Exclude Bob
3. **Pick customer Carol (103)**
   → Run subquery: `SELECT 1 FROM orders o WHERE o.customer_id = 103`
   → Result: **Exists** ✅ → Include Carol

✅ Final Output:

customer_id	name
101	Alice
103	Carol

🎯 Query (Correlated Subquery)

```
SELECT e.employee_id, e.name, e.salary, e.dept_id
FROM employees e
WHERE e.salary > (
    SELECT AVG(e2.salary)
    FROM employees e2
    WHERE e2.dept_id = e.dept_id
);
```

🔍 Query Goal:

Select employees who earn **more than the average salary of their own department**.

🧾 Sample `employees` Table

employee_id	name	salary	dept_id
101	Raj	60,000	10
102	Priya	50,000	10
103	Aarav	40,000	10
104	Neha	70,000	20
105	Rohan	60,000	20
106	Kavya	90,000	30

🔄 Dry Run – Step-by-Step Execution:

We will process **each row** and compare its salary with the **average salary of its department**.

✅ Raj (Dept 10, Salary 60,000)

- Subquery:
  `SELECT AVG(salary) FROM employees WHERE dept_id = 10`
  → (60,000 + 50,000 + 40,000) / 3 = **50,000**
- Condition: `60,000 > 50,000` ✅
  → Include Raj

✅ Priya (Dept 10, Salary 50,000)

- Avg = 50,000
- Condition: `50,000 > 50,000` ❌
  → Exclude Priya

✅ Aarav (Dept 10, Salary 40,000)

- Avg = 50,000
- `40,000 > 50,000` ❌
  → Exclude Aarav

✅ Neha (Dept 20, Salary 70,000)

- Avg of Dept 20 = (70,000 + 60,000) / 2 = **65,000**
- `70,000 > 65,000` ✅
  → Include Neha

✅ Rohan (Dept 20, Salary 60,000)

- Avg = 65,000
- `60,000 > 65,000` ❌
  → Exclude Rohan

✅ Kavya (Dept 30, Salary 90,000)

- Only employee in dept 30
- Avg = 90,000
- `90,000 > 90,000` ❌
  → Exclude Kavya

✅ Final Output:

employee_id	name	salary	dept_id
101	Raj	60000	10
104	Neha	70000	20

🧠 Trainer Notes:

- This query is a **classic use of correlated subquery**: every row runs its **own AVG()** query.
- It's very useful to **highlight per-group comparisons** without using JOIN or GROUP BY.

Would you like the same dry-run shown in `.pptx` or `.sql` format with visual callouts for class?

🎯 Goal:

List employees who earn **more than the average salary of their department**, using **`JOIN`** + **`GROUP BY`** instead of a correlated subquery.

✅ Method 1: Using a Derived Table (Subquery + JOIN)

```
sql
CopyEdit
SELECT e.employee_id, e.name, e.salary, e.dept_id
FROM employees e
JOIN (
    SELECT dept_id, AVG(salary) AS avg_salary
    FROM employees
    GROUP BY dept_id
) dept_avg ON e.dept_id = dept_avg.dept_id
WHERE e.salary > dept_avg.avg_salary;
```

🧠 Explanation:

1. **Inner subquery**: Calculates average salary per department.

   ```
   SELECT dept_id, AVG(salary)
   FROM employees
   GROUP BY dept_id
   ```

2. **JOIN**: Matches each employee with their department's average.
3. **WHERE clause**: Filters employees whose salary > average.

🔎 Result

employee_id	name	salary	dept_id
101	Raj	60000	10
104	Neha	70000	20

✅ **Same result** as with the correlated subquery — but may **perform better** on large datasets.

Bring a list of all those employees **who earn more than the average salary of their department**, but **don't use a correlated subquery**.

✅ **Final Query**:

```
sql
```

```
CopyEdit
SELECT e.employee_id, e.name, e.salary, e.dept_id
FROM employees e
JOIN (
    SELECT dept_id, AVG(salary) AS avg_salary
    FROM employees
    GROUP BY dept_id
) dept_avg
ON e.dept_id = dept_avg.dept_id
WHERE e.salary > dept_avg.avg_salary;
```

🔍 Breakdown (Simple Explanation):

🔹 Step 1: Inner Query – Group By Department

```
sql
CopyEdit
SELECT dept_id, AVG(salary) AS avg_salary
FROM employees
GROUP BY dept_id;
```

🟡 This query calculates the **average salary** of each **department**.

Example output:

dept_id	avg_salary
10	50,000
20	65,000
30	90,000

🔹 Step 2: JOIN Outer Table (employees e)

```
sql
CopyEdit
FROM employees e
JOIN (...) dept_avg
ON e.dept_id = dept_avg.dept_id
```

🟢 Now we are **joining each employee** with the **average salary of their department**, using `dept_id` as the matching point.

🔹 Step 3: Filter Employees Whose Salary > Avg Salary

```
sql
CopyEdit
WHERE e.salary > dept_avg.avg_salary;
```

🔴 Here we are **only showing** those employees **whose salary is more than** the **average salary** of their **own department**.

✅ Sample Data:

employee_id	name	salary	dept_id
101	Raj	60,000	10
102	Priya	50,000	10
103	Aarav	40,000	10
104	Neha	70,000	20
105	Rohan	60,000	20
106	Kavya	90,000	30

Avg salaries from subquery:

- Dept 10 → (60+50+40)/3 = 50,000
- Dept 20 → (70+60)/2 = 65,000
- Dept 30 → 90,000

🔚 Final Output:

employee_id	name	salary	dept_id
101	Raj	60,000	10
104	Neha	70,000	20

✅ **`dept_avg` is an alias** — **not** a real table.

🔍 Explanation:

In this part of the query:

```
JOIN (
  SELECT dept_id, AVG(salary) AS avg_salary
  FROM employees
  GROUP BY dept_id
) dept_avg
```

- You are writing a **subquery** (also called a **derived table** or **inline view**).
- `dept_avg` is simply a **nickname** (alias) for that subquery's result.
- Think of it like:
  `JOIN (result_set) AS dept_avg`

🧠 Analogy

🗂️ It's like you made a temporary result with:

```
dept_id | avg_salary
--------|------------
10      | 50000
```

```
20      | 65000
```

And gave that result a **nickname** → `dept_avg`, which you can now use like a table:

```
dept_avg.dept_id
dept_avg.avg_salary
```

✅ 3. "Find employees in departments 10, 20, 30"

Use: Multi-Row Subquery (`IN`)
🔍 You’re filtering based on a list of values (can be static or dynamic).

```
sql
CopyEdit
SELECT *
FROM employees
WHERE dept_id IN (10, 20, 30);
```

Or dynamically:

```
sql
CopyEdit
SELECT *
FROM employees
WHERE dept_id IN (
    SELECT dept_id FROM departments WHERE region = 'North'
);
```

✅ Simple value list filter — **one column, multiple rows**.

✅ 4. "Get department name with every employee"

Use: INNER JOIN
🔍 You need columns from both tables: employee and department.

```
sql
CopyEdit
SELECT e.employee_id, e.name, d.department_name
FROM employees e
JOIN departments d ON e.dept_id = d.dept_id;
```

✅ JOIN is the best when you're **combining columns** from multiple tables.

✅ 5. "Find employee whose salary = max salary"

Use: Single-Row Subquery
🔍 Only one max salary value, so subquery returns **one row**.

```
sql
CopyEdit
SELECT *
FROM employees
WHERE salary = (
    SELECT MAX(salary)
    FROM employees
);
```

✅ Use = with **scalar subquery** — only works if inner query returns exactly **one value**.

✅ 6. "Compare each row to a group result"

Use: Correlated Subquery (again)
🔍 Same logic as #1, e.g., compare to **max salary of their dept**:

```
sql
CopyEdit
SELECT e.employee_id, e.name, e.salary
FROM employees e
WHERE salary = (
    SELECT MAX(salary)
    FROM employees e2
    WHERE e2.dept_id = e.dept_id
);
```

✅ Every row gets its own inner query result.

🔔 Bonus Tip:

If you just want to *check condition*, use:

- ✅ `IN` for simple list matching
- ✅ `EXISTS` for **faster lookup** and avoiding `NULL` problems
- ✅ Correlated subqueries for **row-wise logic**
- ✅ Joins for **data from multiple tables**