Day 2 (2-6-25)

Advanced SQL Concepts

What are Window Functions?

Window functions perform **calculations across a set of rows** that are related to the current row — without grouping or losing individual row data.

**Common examples:**

* ROW\_NUMBER()
* RANK()
* DENSE\_RANK()
* LAG(), LEAD()
* SUM(), AVG() OVER a window

**Use cases:**

* Ranking employees by salary
* Getting running totals or moving averages
* Comparing values from previous or next rows (e.g., sales growth)
* Detecting duplicates (ROW\_NUMBER() for de-duplication)

**When to use:**

Use window functions when you need **row-level calculations** that depend on other rows — but still want to **see every row**.

What is CTE?

Abbreviation – Common Table Expression

A CTE is a **temporary result set** defined with WITH, used to simplify complex queries by giving a name to a subquery.

**Use cases:**

* Breaking a big query into smaller parts
* Reusing the same subquery multiple times
* Making recursive queries (e.g., employee hierarchies)
* Improving readability

**When to use:**

Use CTEs when your SQL is getting messy with nested subqueries, or when you want to make your logic easier to follow step-by-step.

**SQL\_Practice\_Coding\_challenge:**

create database dataset;

use dataset;

-- Create Departments Table

CREATE TABLE departments (

department\_id INT PRIMARY KEY,

department\_name VARCHAR(50)

);

-- Create Projects Table

CREATE TABLE projects (

project\_id INT PRIMARY KEY,

project\_name VARCHAR(50),

start\_date DATE,

end\_date DATE

);

-- Create Employees Table with Foreign Keys

CREATE TABLE employees (

emp\_id INT PRIMARY KEY,

first\_name VARCHAR(50),

last\_name VARCHAR(50),

department VARCHAR(50),

salary INT,

join\_date DATE,

manager\_id INT,

project\_id INT NULL,

CONSTRAINT fk\_project FOREIGN KEY (project\_id) REFERENCES projects(project\_id)

);

-- Departments

INSERT INTO departments VALUES

(1, 'IT'), (2, 'HR'), (3, 'Finance'), (4, 'Sales'), (5, 'Marketing'), (6, 'strategy Analyzer');

-- Projects

INSERT INTO projects VALUES

(101, 'Apollo', '2023-01-01', '2023-12-31'),

(102, 'Zeus', '2022-06-01', '2023-05-31'),

(103, 'Hermes', '2024-01-15', '2024-12-15'),

(104, 'Hendry', '2025-02-10','2025-05-12'),

(105, 'Ronald', '2024-02-10','2025-04-12');

-- Employees

INSERT INTO employees VALUES

(1, 'Alice', 'Smith', 'IT', 90000, '2022-01-15', NULL, 101),

(2, 'Bob', 'Singh', 'strategy Analyzer', 88000, '2021-12-01', 1, 102),

(3, 'Carol', 'Suresh', 'Finance', 60000, '2023-03-12', 1, NULL),

(4, 'David', 'Jones', 'HR', 55000, '2024-04-10', 2, 104),

(5, 'Eva', 'Brown', 'Sales', 40000, '2023-06-20', 2, 103),

(6, 'Frank', 'Stone', NULL, 75000, '2020-08-01', NULL, NULL),

(7, 'Grace', 'Sharma', 'IT', 88000, '2021-01-10', 1, 105),

(8, 'Hank', 'Lee', 'HR', 39000, '2024-01-05', 4, NULL),

(9, 'Ivy', 'Sinha', 'Finance', 62000, '2023-09-15', 3, 101),

(10, 'John', 'Doe', 'Marketing', 90000, '2022-10-10', 1, 102);

**String Functions:**

--1. Find employees whose last name starts with 'S'.

SELECT first\_name , last\_name from employees where last\_name LIKE 's%';

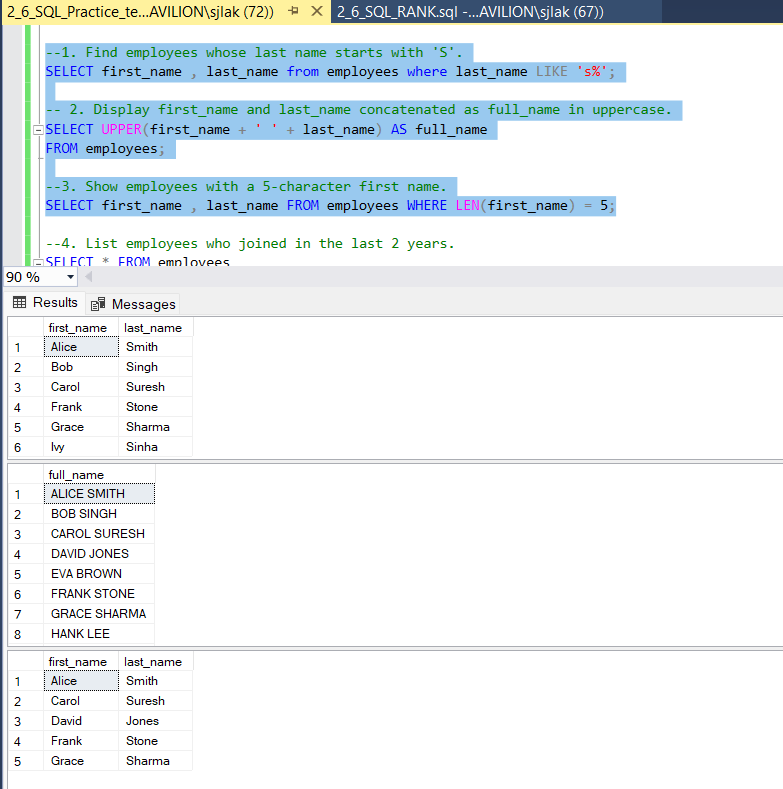
-- 2. Display first\_name and last\_name concatenated as full\_name in uppercase.

SELECT UPPER(first\_name + ' ' + last\_name) AS full\_name

FROM employees;

--3. Show employees with a 5-character first name.

SELECT first\_name , last\_name FROM employees WHERE LEN(first\_name) = 5;



**Date Functions:**

--4. List employees who joined in the last 2 years.

SELECT \* FROM employees

WHERE join\_date >= DATEADD(YEAR, -2, GETDATE());

--5. Show number of days since each employee joined.

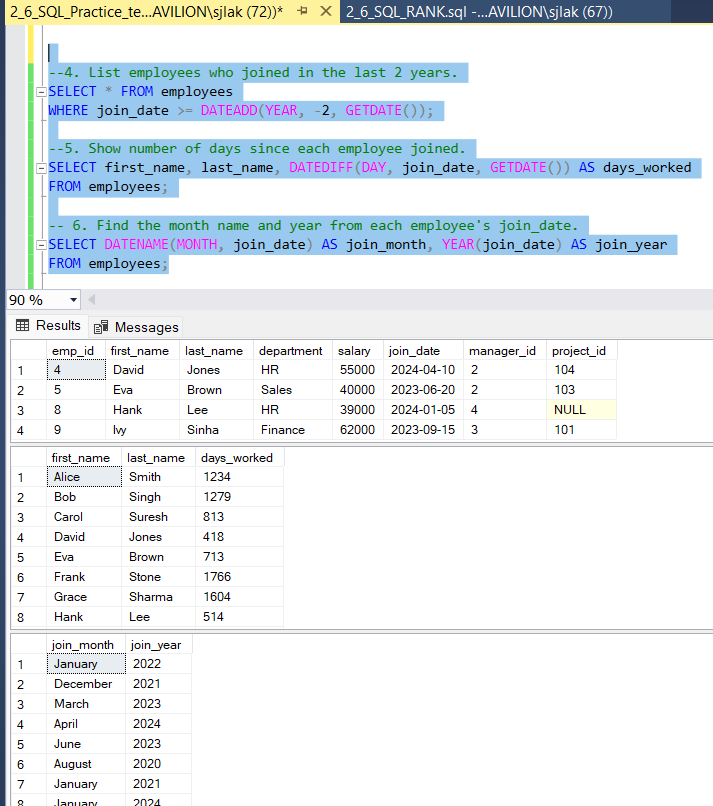
SELECT first\_name, last\_name, DATEDIFF(DAY, join\_date, GETDATE()) AS days\_worked

FROM employees;

-- 6. Find the month name and year from each employee's join\_date.

SELECT DATENAME(MONTH, join\_date) AS join\_month, YEAR(join\_date) AS join\_year

FROM employees;



**Math Functions:**

-- 7. Round off each employee's salary to the nearest thousand.

SELECT first\_name, last\_name, FLOOR(salary / 1000.0) \* 1000 AS round\_off\_salary

FROM employees;

-- 8. Find employees whose salary is above the average salary.

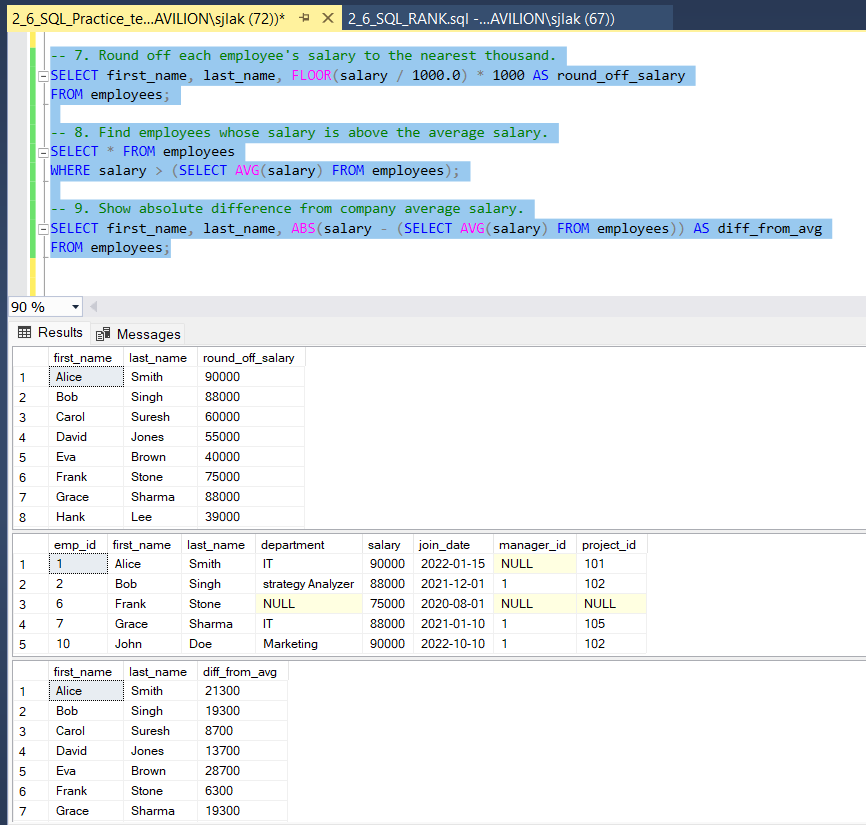
SELECT \* FROM employees

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

-- 9. Show absolute difference from company average salary.

SELECT first\_name, last\_name, ABS(salary - (SELECT AVG(salary) FROM employees)) AS diff\_from\_avg

FROM employees;



**Aggregate Functions with HAVING:**

-- 10. Find departments with more than 3 employees.

SELECT department, emp\_count FROM (

SELECT department, COUNT(\*) AS emp\_count

FROM employees

GROUP BY department) AS temp

WHERE emp\_count > 3;

--11. Show total and average salary per department with avg salary > 60000.

SELECT department, SUM(salary) AS total\_salary, AVG(salary) AS avg\_salary

FROM employees

GROUP BY department

HAVING AVG(salary) > 60000;

A screenshot of a computer

AI-generated content may be incorrect.

**Subqueries:**

-- 12. Find the employee(s) with the maximum salary.

SELECT \* FROM employees

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

-- 13. List employees earning more than avg salary in their department.

SELECT \* FROM employees e

WHERE salary > (

SELECT AVG(salary) FROM employees

WHERE department = e.department

);

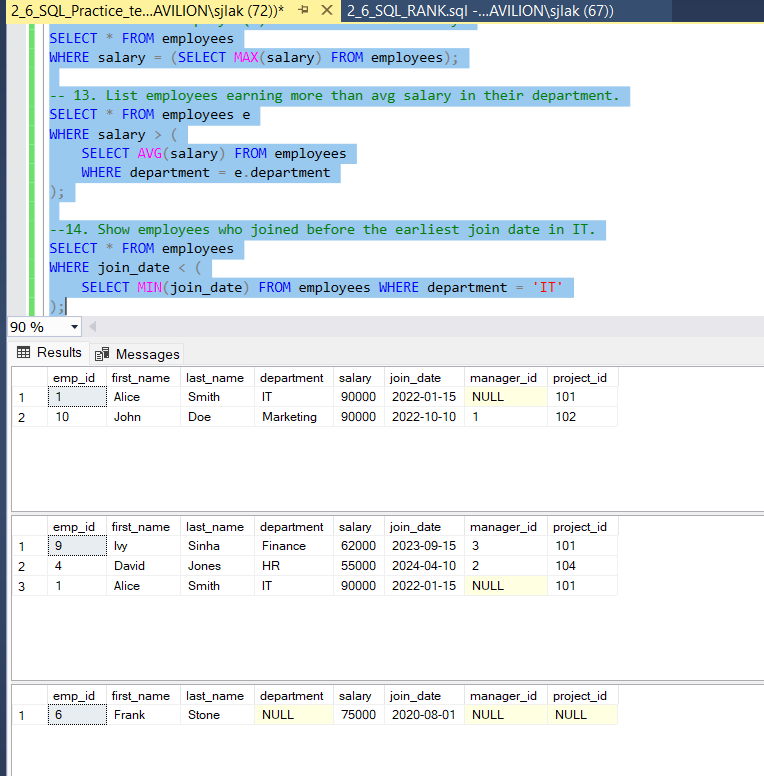
--14. Show employees who joined before the earliest join date in IT.

SELECT \* FROM employees

WHERE join\_date < (

SELECT MIN(join\_date) FROM employees WHERE department = 'IT'

);



**JOINS:**

-- 15. Show each employee's name and manager's name.

SELECT e.first\_name + ' ' + e.last\_name AS employee,

m.first\_name + ' ' + m.last\_name AS manager

FROM employees e

LEFT JOIN employees m ON e.manager\_id = m.emp\_id;

-- 16. List employees with their department name.

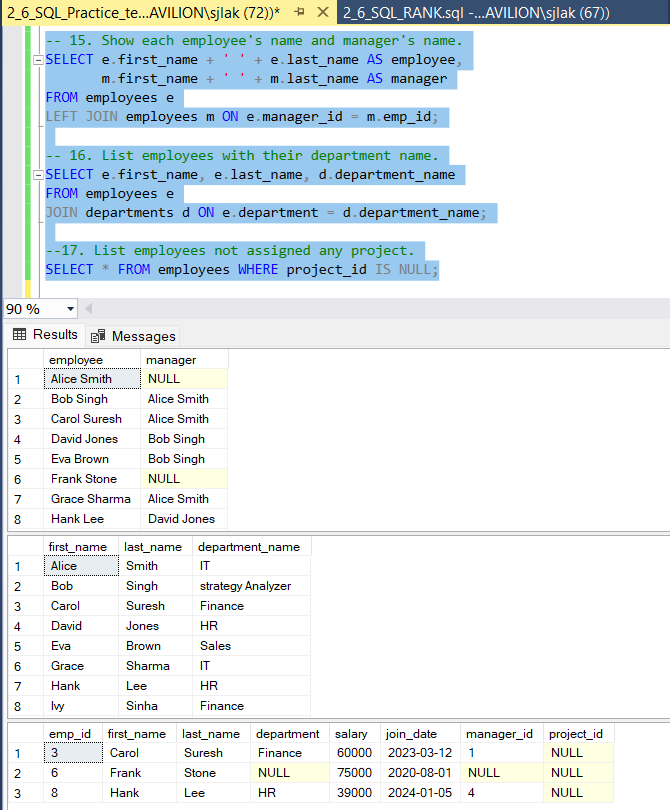
SELECT e.first\_name, e.last\_name, d.department\_name

FROM employees e

JOIN departments d ON e.department = d.department\_name;

--17. List employees not assigned any project.

SELECT \* FROM employees WHERE project\_id IS NULL;



**Window Functions:**

-- 18. Assign a row number to employees in each department based on salary.

SELECT \*, ROW\_NUMBER() OVER (PARTITION BY department ORDER BY salary DESC) AS row\_num

FROM employees;

-- 19. Show running total salary within each department.

SELECT \*, SUM(salary) OVER (PARTITION BY department ORDER BY join\_date) AS running\_total

FROM employees;

--20. Show difference in salary between employee and previous by join date.

SELECT \*, salary - LAG(salary) OVER (ORDER BY join\_date) AS salary\_diff\_prev

FROM employees;

A screenshot of a computer

AI-generated content may be incorrect.

**CTE (Common Table Expressions):**

--21. Use CTE to calculate total salary per department, filter total > 200000.

WITH dept\_salary AS (

SELECT department, SUM(salary) AS total\_salary

FROM employees

GROUP BY department

)

SELECT \* FROM dept\_salary

WHERE total\_salary > 200000;

--22. Create a recursive CTE to generate numbers 1 to 10.

WITH numbers(n) AS (

SELECT 1

UNION ALL

SELECT n + 1 FROM numbers WHERE n < 10

)

SELECT \* FROM numbers;

--23. Use a CTE to find employees with duplicate first names.

WITH dup\_names AS (

SELECT first\_name, COUNT(\*) AS cnt

FROM employees

GROUP BY first\_name

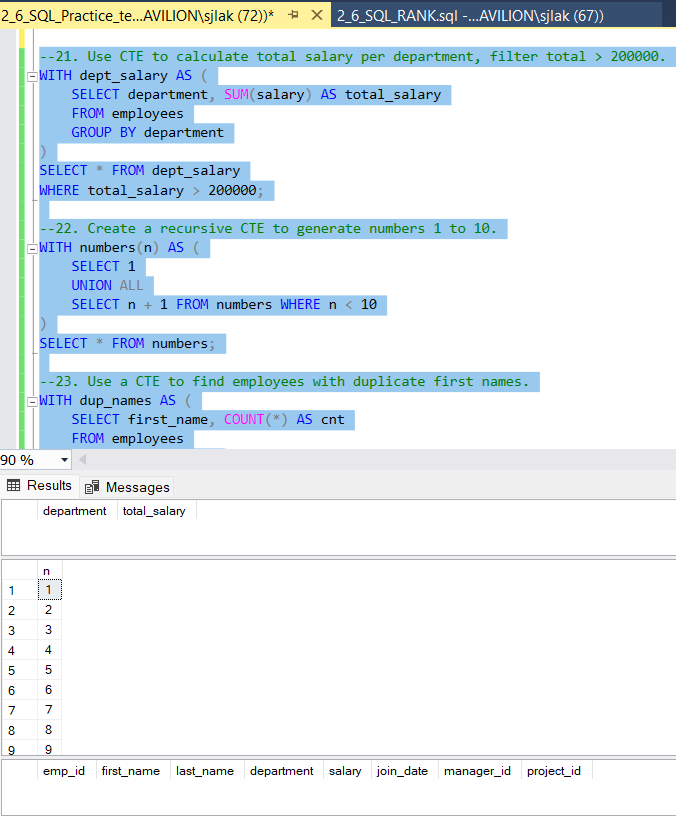
HAVING COUNT(\*) > 1

)

SELECT e.\*

FROM employees e

JOIN dup\_names d ON e.first\_name = d.first\_name;



**Case Statements:**

--24. Label employees as 'Junior', 'Mid', or 'Senior' based on salary.

SELECT first\_name, last\_name, salary,

CASE

WHEN salary < 40000 THEN 'Junior'

WHEN salary BETWEEN 40000 AND 80000 THEN 'Mid'

ELSE 'Senior'

END AS level

FROM employees;

--25. Count employees in salary categories using CASE.

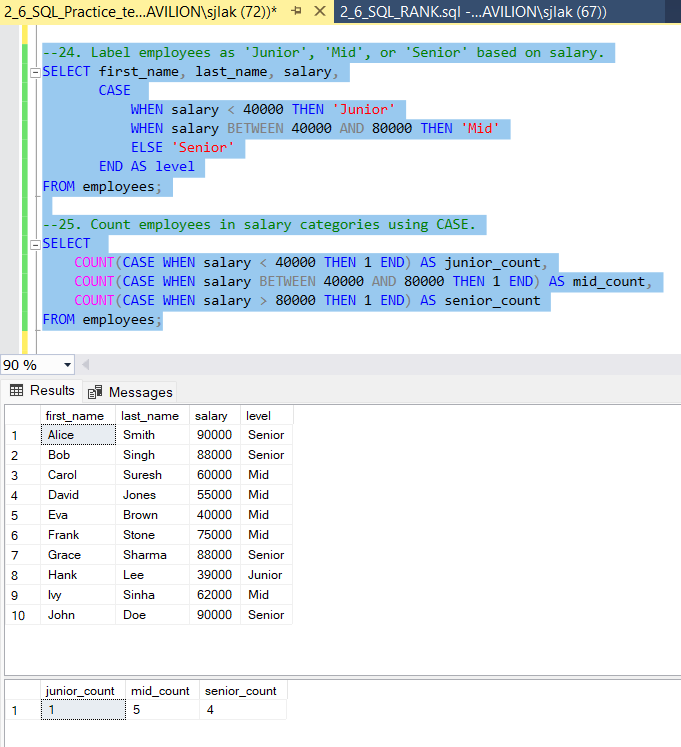
SELECT

COUNT(CASE WHEN salary < 40000 THEN 1 END) AS junior\_count,

COUNT(CASE WHEN salary BETWEEN 40000 AND 80000 THEN 1 END) AS mid\_count,

COUNT(CASE WHEN salary > 80000 THEN 1 END) AS senior\_count

FROM employees;



**NULL Functions:**

--26. Replace NULL department values with 'Unknown'.

SELECT ISNULL(department, 'Unknown') AS department

FROM employees;

--27. Show employees with no department.

SELECT \* FROM employees WHERE department IS NULL;

--28. Use COALESCE to provide default for missing projects.

SELECT first\_name, last\_name, COALESCE(project\_id, 0) AS project\_assigned

FROM employees;

