**DAY 1 (29-5-25)**

Intro:

What is the Industrial Revolution?

1. The **Industrial Revolutions** are distinct phases in history that radically changed how humans **produce goods**, **work**, and **live** — each driven by **technological innovation**.
2. In Short it is Known as Revolution which happen based upon the latest tech in that period.

INDUSTRY 1.0 (Late 1700s to Mid-1800s):

* **Steam Engine**
* **Water Wheels**
* **Mechanical Looms**
* **Textile Machines**
* **Iron Production (Blast Furnace)**
* **Canals and Railways**
* **Hand Tools to Machines**

**INDUSTRY 2.0** (Late 1800s to Early 1900s):

* Electricity
* Assembly Line (e.g., Ford Model T)
* Telegraph & Telephone
* Steel Manufacturing
* Internal Combustion Engine
* Light Bulb
* Railroads and Shipping Expansion
* Oil and Chemical Industries

INDUSTRY 3.0 (1970s–2000s):

* Computers & Microprocessors
* Programmable Logic Controllers (PLCs)
* Robotics
* Information Technology (IT)
* Enterprise Resource Planning (ERP) systems
* Internet (early stages)
* Digital Communication (e.g., email)
* Basic Data Storage & Retrieval Systems

INDUSTRY 4.0 (2010s–Now):

* GEN AI
* Metaverse
* Cyber security and Cyber Physical Systems
* Block Chain
* IOT
* BIG DATA
* Quantum Computing
* Cloud Computing
* Prompt Engineering

What is Data Engineering?

Data Engineering focuses on the design, building, and maintenance of systems that collect, store, and process data at scale.

Processes in Data Engineering:

Data Engineering involves the following core processes:

1. **Data Collection** – Getting data from different sources.
2. **Data Ingestion** – Moving raw data into a system (cloud, lake, warehouse).
3. **Data Cleaning & Transformation (ETL/ELT)** – Ensuring data quality, consistency.
4. **Data Storage** – Storing data efficiently using databases or data lakes/warehouses.
5. **Data Pipeline Orchestration** – Automating workflows using tools like Apache Airflow.
6. **Serving Data** – Making it accessible for analysts and data scientists (Power BI).

What is Data Science?

Data Science focuses on analysing and interpreting complex data to extract insights, build models, and drive decision-making.

A whiteboard with red text

AI-generated content may be incorrect.

DATA PIPELINE WORKFLOW:

A whiteboard with writing on it

AI-generated content may be incorrect.

Step 1:

Collect Data from multiple sources like,

* Flat Files (CSV, Excel, etc.)
* Sensors (IoT devices)
* APIs
* Cloud-based data
* Social Media
* Streaming data (real-time sources)
* CRM systems (like Salesforce)

Step 2:

Store the collected Raw Data,

* All Structured and Un-structured Data are stored.
* Stored in Amazon or Azure clouds.

Step 3:

Convert the data into usable Format and need to structure according to it and this process is known as ETL.

* Extract - Pull data from the storage space.
* Transform – clean the data, validate data and prepare for Loading.
* Load - Send the transformed data to the next stage

Step 4:

Storing the cleaned data,

* Transformed data is loaded here.
* Uses schema-based storage for High-speed queries retrieval and Analytical workloads.

Step 5:

Provide insights, dashboards, and reports from the stored data.

Tools which can be used for this process are,

* Power BI
* Tableau
* Excel

A whiteboard with red writing

AI-generated content may be incorrect.

**Data Ingestion**: Importing raw data from multiple sources into a system.

* + *Example*: Bringing data from sensors or websites into a cloud platform.

**Data Lake**: A large storage repository for raw, unstructured, semi-structured, and structured data.

* + *Tools*: AWS S3, Azure Data Lake

**Machine Learning:**

**Machine Learning (ML)** is a subfield of **Artificial Intelligence (AI)** that enables computers to **learn from data** and **make decisions or predictions** without being explicitly programmed for every task.

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%';

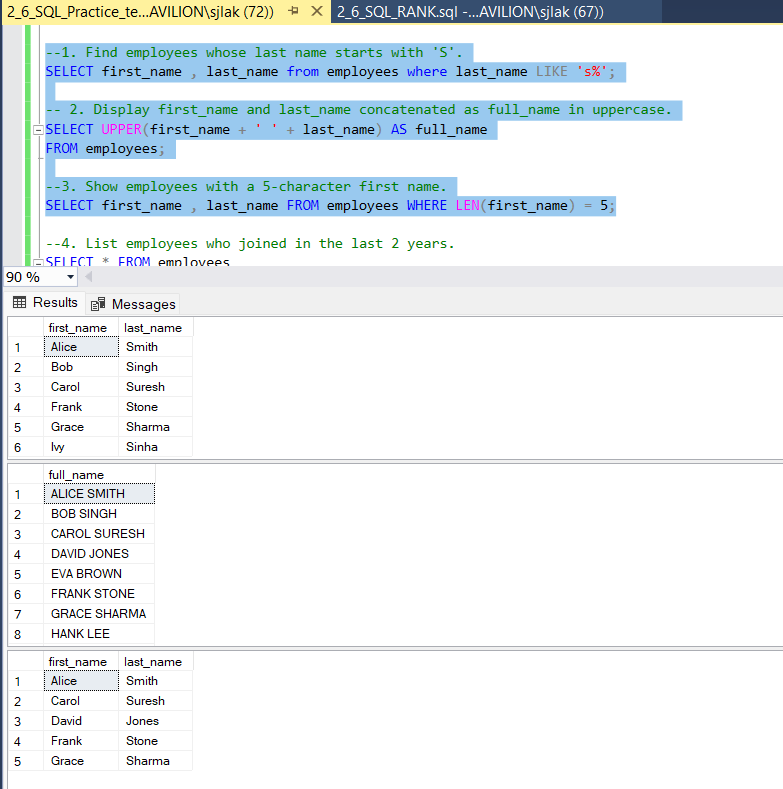
-- 2. Display first\_name and last\_name concatenated as full\_name in uppercase.

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.

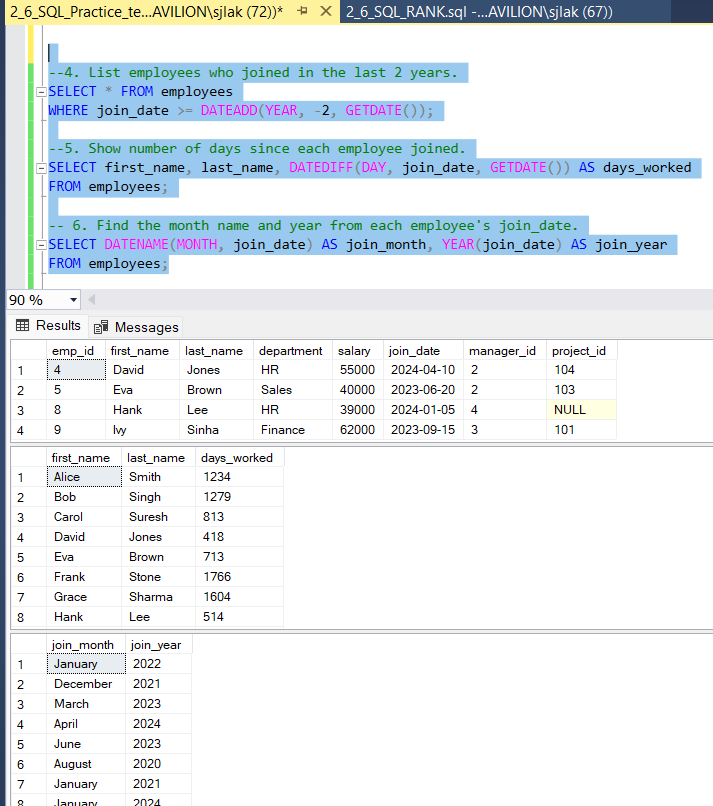
SELECT first\_name, last\_name, DATEDIFF(DAY, join\_date, GETDATE()) AS days\_worked

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.

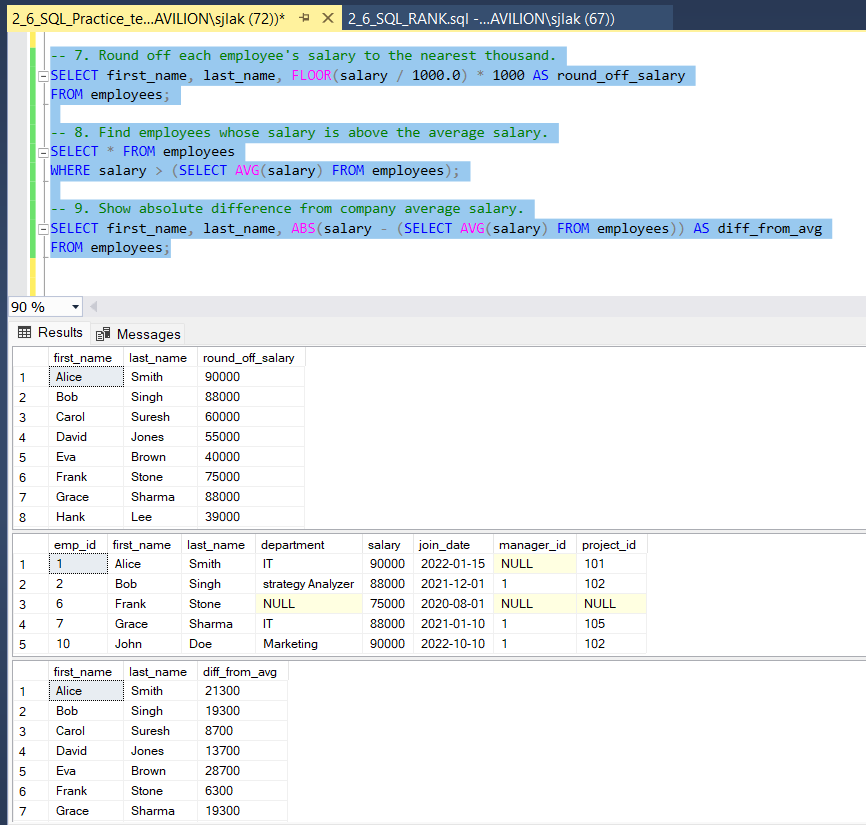
SELECT \* FROM employees

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

);

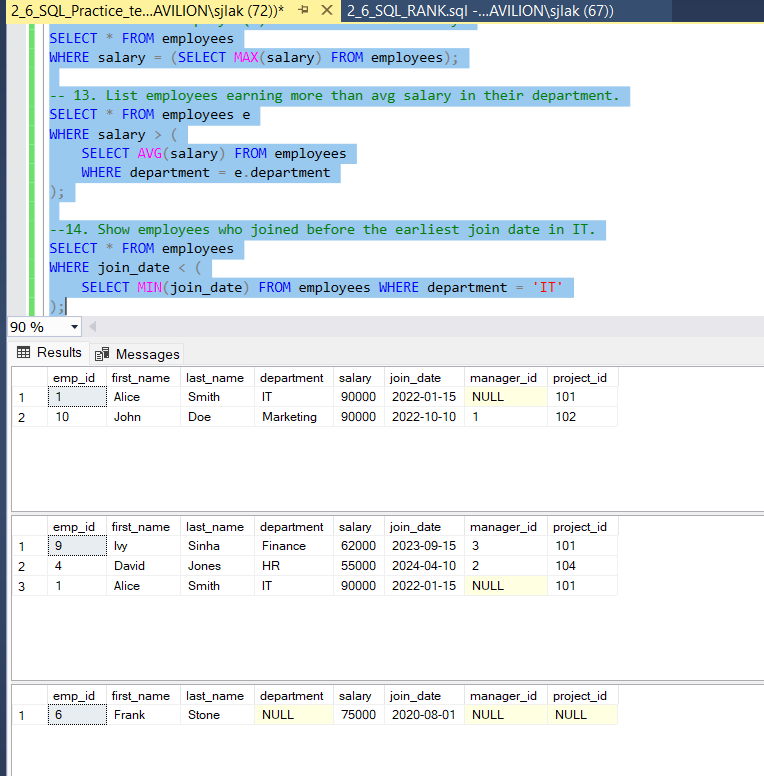
--14. Show employees who joined before the earliest join date in IT.

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.

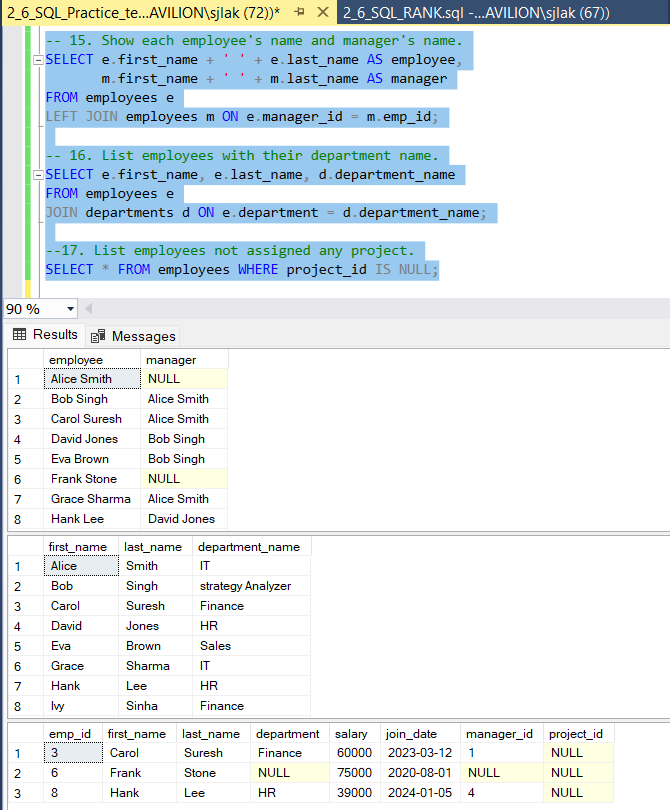
SELECT e.first\_name, e.last\_name, d.department\_name

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

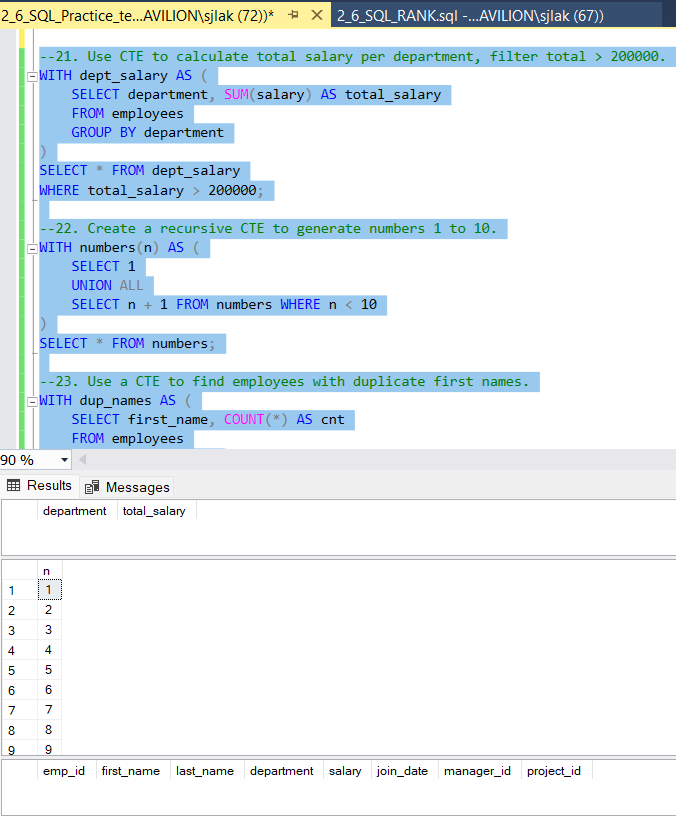
HAVING COUNT(\*) > 1

)

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;

A screenshot of a computer

AI-generated content may be incorrect.

**NULL Functions:**

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

SELECT ISNULL(department, 'Unknown') AS department

FROM employees;

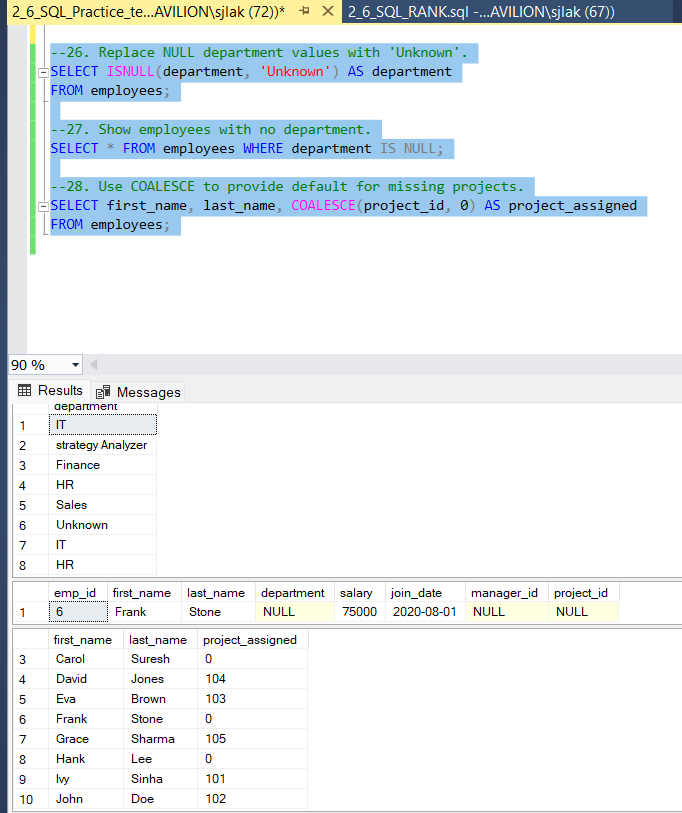
--27. Show employees with no department.

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;



**PROJECT 1 (02-06-2025)**

**ShopFast**

**STEP 1: Creating a new Database,**

create database ShopFast;

**STEP 2: using the new database,**

use ShopFast;

**STEP 3: Inserting the values through given csv files,**

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**STEP 4: Check weather all the data’s are inserted properly,**

select \* from CUSTOMERS;

select \* from ORDER\_ITEMS;

select \* from ORDERS;

select \* from PRODUCTS;

A screenshot of a computer

AI-generated content may be incorrect.

**STEP 5: Answers for all Problem Set**

--1. Customer Sign-up Trend: New customers per month (last 12 months)

SELECT customer\_id, name, signup\_date FROM CUSTOMERS

WHERE signup\_date >= DATEADD(MONTH, -12, GETDATE());

A screenshot of a computer

AI-generated content may be incorrect.

--2. Top 5 Customers by Revenue: Total orders, revenue, and avg order value

SELECT TOP 5 o.customer\_id, c.name, o.total\_amount FROM ORDERS o

JOIN CUSTOMERS c ON o.customer\_id = c.customer\_id

ORDER BY o.total\_amount DESC;

A screenshot of a computer

AI-generated content may be incorrect.

--3. Order Status Distribution: Count of each status

SELECT status, COUNT(\*) AS order\_count FROM ORDERS

GROUP BY status;

A screenshot of a computer

AI-generated content may be incorrect.

--4. Revenue by Category: Total revenue by category

SELECT o.product\_id, p.product\_name, o.quantity, o.price\_per\_unit

FROM ORDER\_ITEMS o

JOIN PRODUCTS p ON o.product\_id = p.product\_id;

A screenshot of a computer

AI-generated content may be incorrect.

--5. Best-Selling Products: Top 5 by quantity sold

SELECT o.product\_id, p.product\_name, o.quantity

FROM ORDER\_ITEMS o

JOIN PRODUCTS p ON o.product\_id = p.product\_id

ORDER BY o.quantity DESC;

A screenshot of a computer

AI-generated content may be incorrect.

--6. Low-Stock Products: Products with <10% stock using CASE

SELECT product\_name, stock\_quantity, CASE WHEN stock\_quantity < 10 THEN 'Low Stock' ELSE 'OK' END AS stock\_status

FROM PRODUCTS;

A screenshot of a computer

AI-generated content may be incorrect.

--7. Avg Delivery Time per Month

SELECT FORMAT(order\_date, 'yyyy-MM') AS order\_month, AVG(DATEDIFF(DAY, order\_date, delivery\_date)) AS avg\_delivery\_days

FROM ORDERS

WHERE delivery\_date IS NOT NULL

GROUP BY FORMAT(order\_date, 'yyyy-MM')

ORDER BY order\_month;

A screenshot of a computer

AI-generated content may be incorrect.

--8. Orders with Delivery >7 days

SELECT order\_id, order\_date, delivery\_date

FROM ORDERS

WHERE DATEDIFF(DAY, order\_date, delivery\_date) > 7;

A screenshot of a computer program

AI-generated content may be incorrect.

--9. Repeat Customers: More than 1 order

SELECT customer\_id FROM ORDERS GROUP BY customer\_id

HAVING COUNT(\*) > 1;

A screenshot of a computer

AI-generated content may be incorrect.

--10. Monthly Revenue Growth with LAG()

SELECT

FORMAT(order\_date, 'yyyy-MM') AS order\_month,

SUM(total\_amount) AS monthly\_revenue,

SUM(total\_amount) - LAG(SUM(total\_amount)) OVER (ORDER BY FORMAT(order\_date, 'yyyy-MM')) AS revenue\_growth

FROM ORDERS

WHERE total\_amount IS NOT NULL

GROUP BY FORMAT(order\_date, 'yyyy-MM')

ORDER BY order\_month;

A screenshot of a computer

AI-generated content may be incorrect.

--11. Cohort Analysis using CTE (signup year)

SELECT customer\_id, name, YEAR(signup\_date) AS signup\_year

FROM CUSTOMERS;

A screenshot of a computer

AI-generated content may be incorrect.

--12. Cancelled/Returned Product Revenue Loss

SELECT order\_id, status, total\_amount FROM ORDERS

WHERE status = 'Cancelled' OR status = 'Returned';

A screenshot of a computer

AI-generated content may be incorrect.

--13. Customer City Heatmap

SELECT customer\_id, city FROM CUSTOMERS;

A screenshot of a computer

AI-generated content may be incorrect.

--14. First & Last Order per Customer with ROW\_NUMBER()

WITH RankedOrders AS (

SELECT

customer\_id,

order\_id,

order\_date,

ROW\_NUMBER() OVER (PARTITION BY customer\_id ORDER BY order\_date ASC) AS rn\_asc,

ROW\_NUMBER() OVER (PARTITION BY customer\_id ORDER BY order\_date DESC) AS rn\_desc

FROM ORDERS

)

SELECT customer\_id, order\_id, order\_date, 'First Order' AS order\_type

FROM RankedOrders WHERE rn\_asc = 1

UNION

SELECT customer\_id, order\_id, order\_date, 'Last Order'

FROM RankedOrders WHERE rn\_desc = 1;

A screenshot of a computer

AI-generated content may be incorrect.

--15. NULL Handling: Orders with missing delivery/amount

SELECT order\_id, delivery\_date, total\_amount FROM ORDERS

WHERE delivery\_date IS NULL OR total\_amount IS NULL;

A screenshot of a computer

AI-generated content may be incorrect.