**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%';

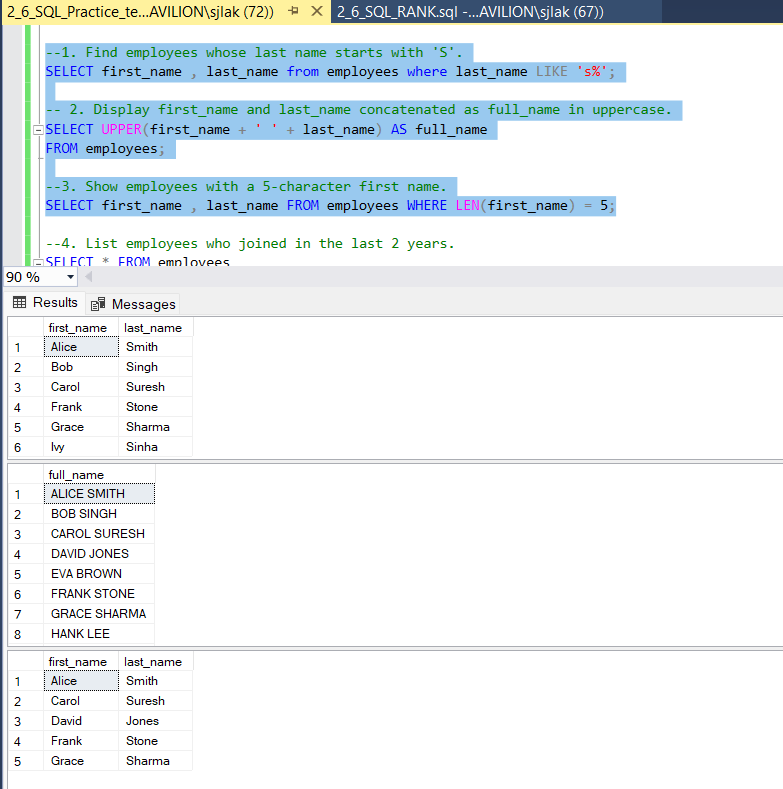
-- 2. Display first\_name and last\_name concatenated as full\_name in uppercase.

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.

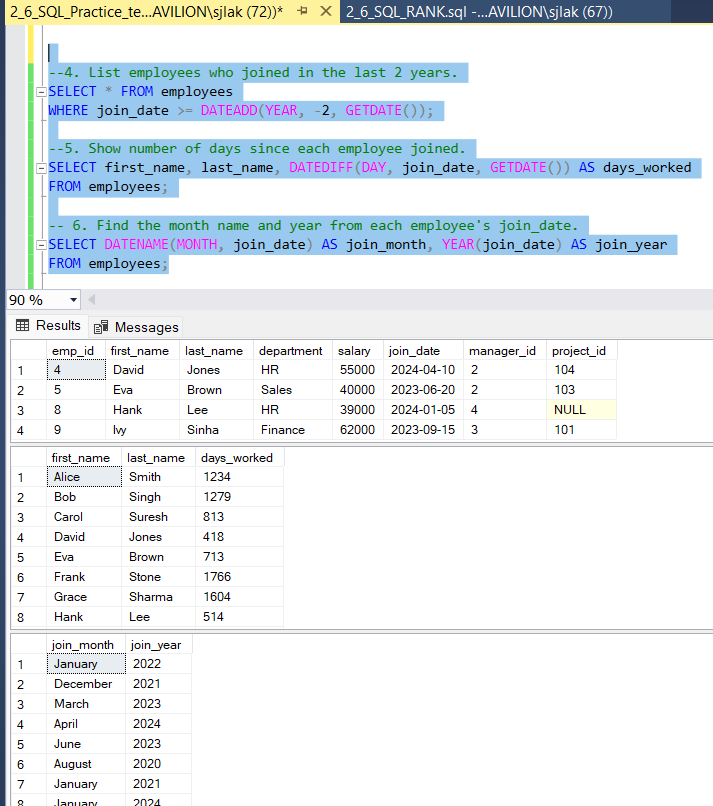
SELECT first\_name, last\_name, DATEDIFF(DAY, join\_date, GETDATE()) AS days\_worked

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.

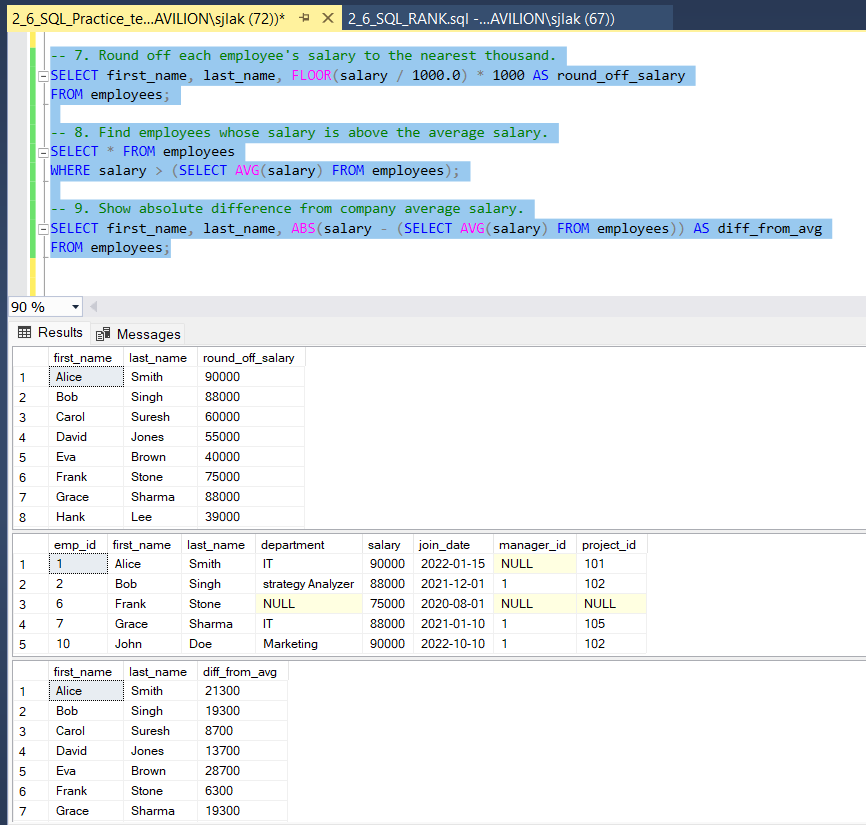
SELECT \* FROM employees

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

);

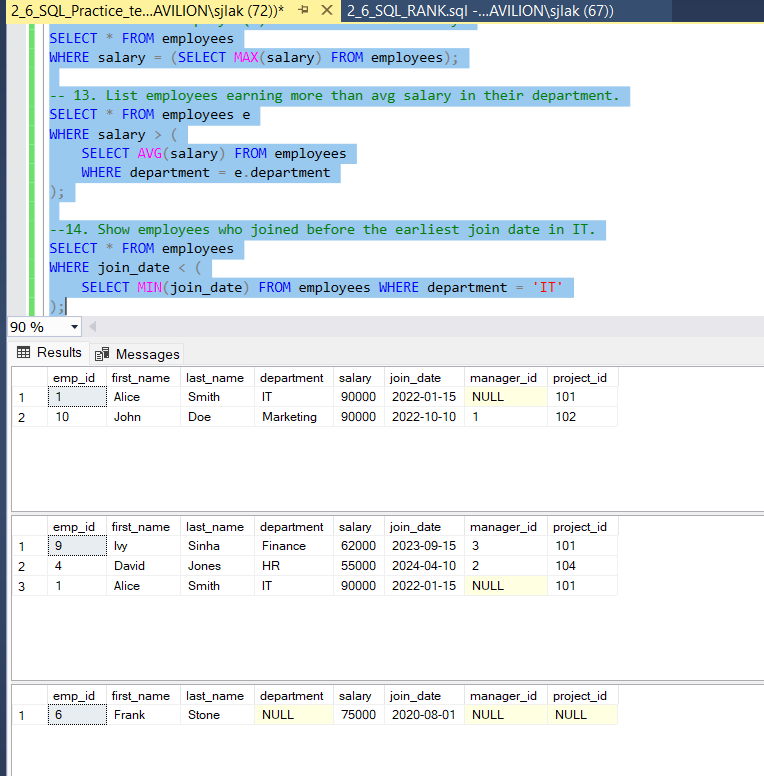
--14. Show employees who joined before the earliest join date in IT.

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.

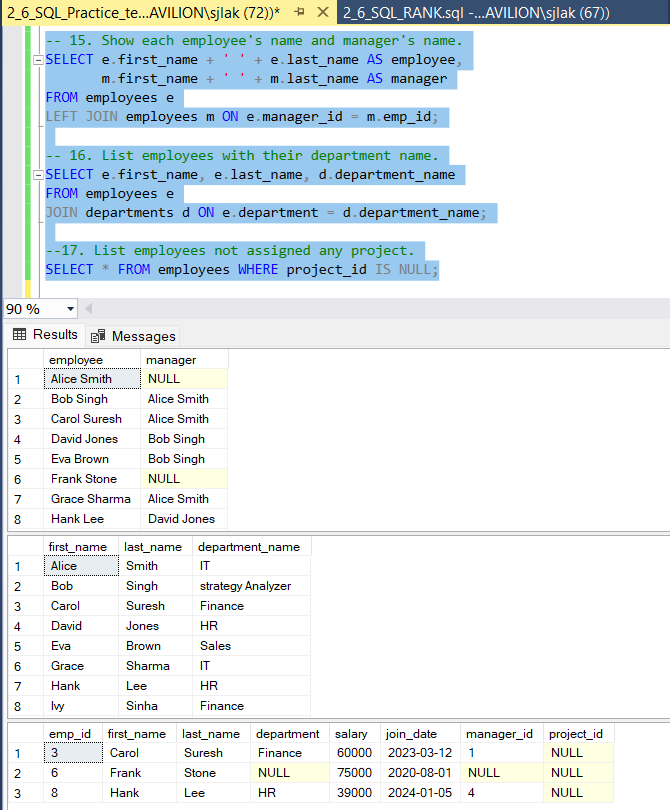
SELECT e.first\_name, e.last\_name, d.department\_name

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

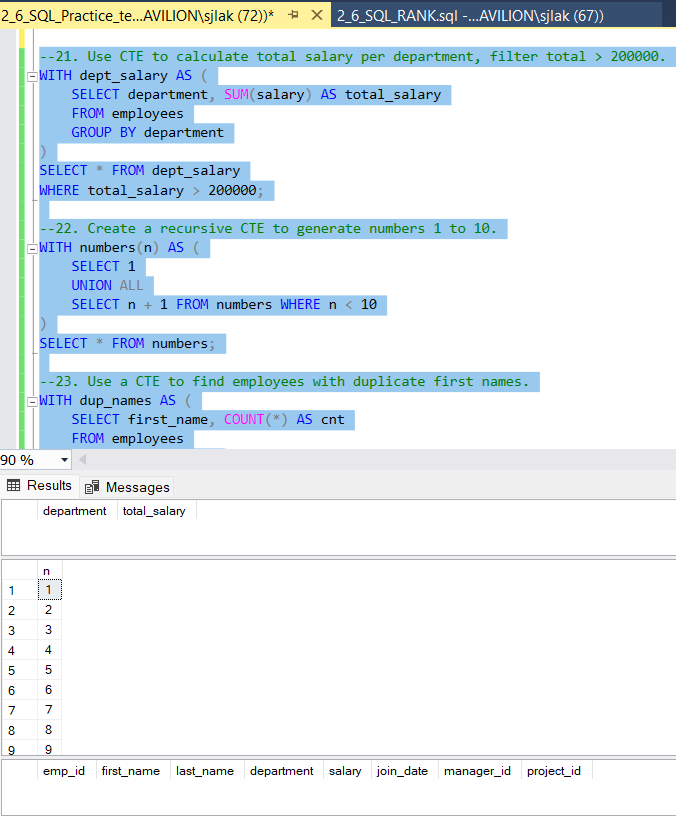
HAVING COUNT(\*) > 1

)

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;

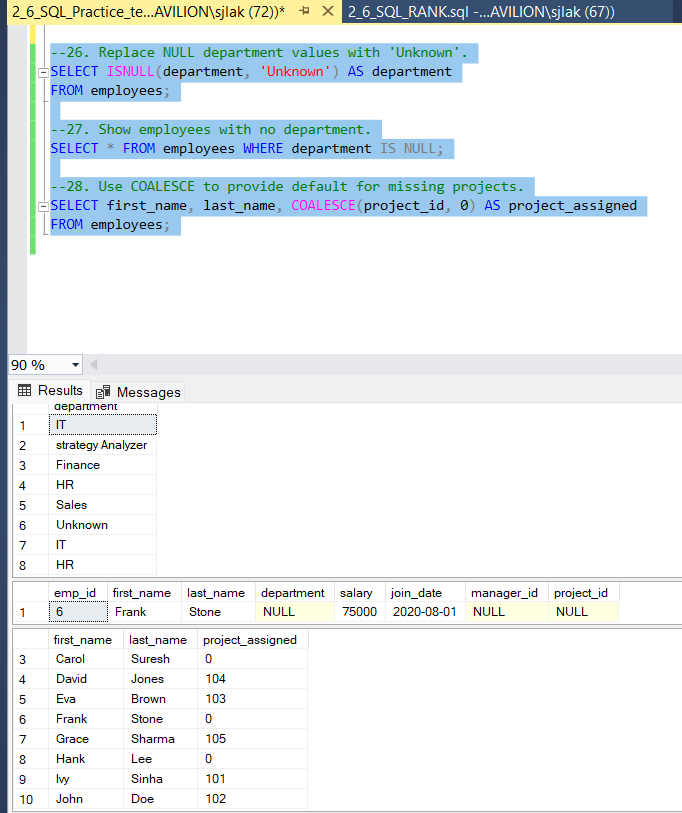
--27. Show employees with no department.

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.

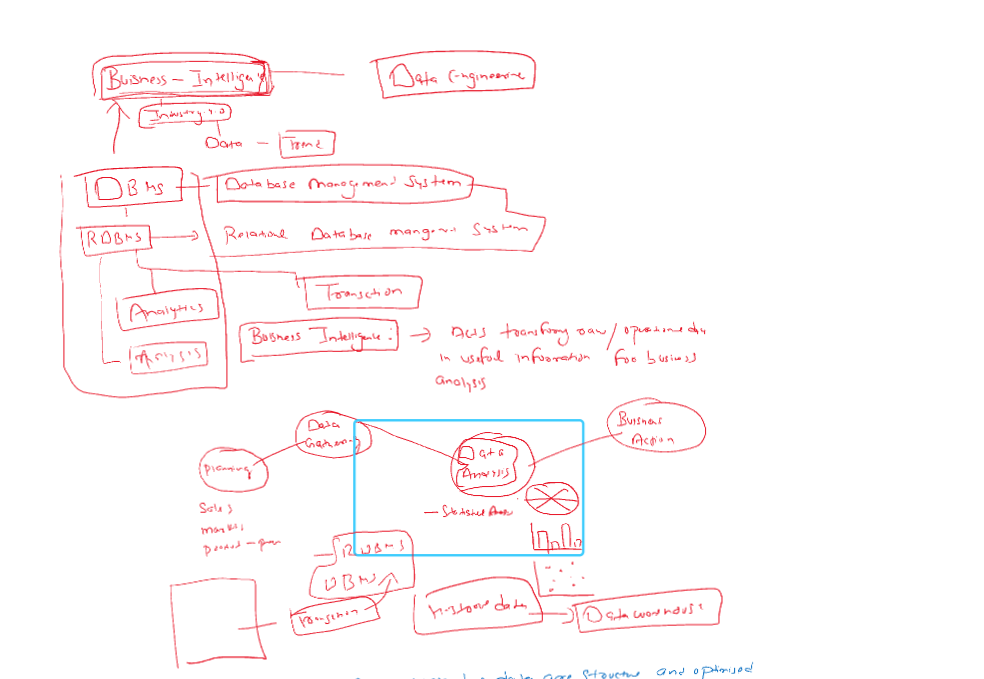
Day 3 (03-06-2025)

[Data Warehousing](https://www.analyticsvidhya.com/blog/2022/06/what-are-schemas-in-data-warehouse-modeling/)

What is Business Intelligence?

**Business Intelligence (BI)** is a **technology-driven process** that collects, integrates, analyzes, and presents **business data** to help organizations **make informed decisions**.

It turns **raw data** into **meaningful insights** using tools like dashboards, reports, visualizations, and analytics.



What is Database?

A **database** is an organized collection of data that is stored and accessed electronically. It allows users to create, read, update, and delete (CRUD) data efficiently.

* **Examples**: MySQL, Oracle, PostgreSQL, MongoDB

What is RDBMS?

Abbreviation - Relational Database Management System

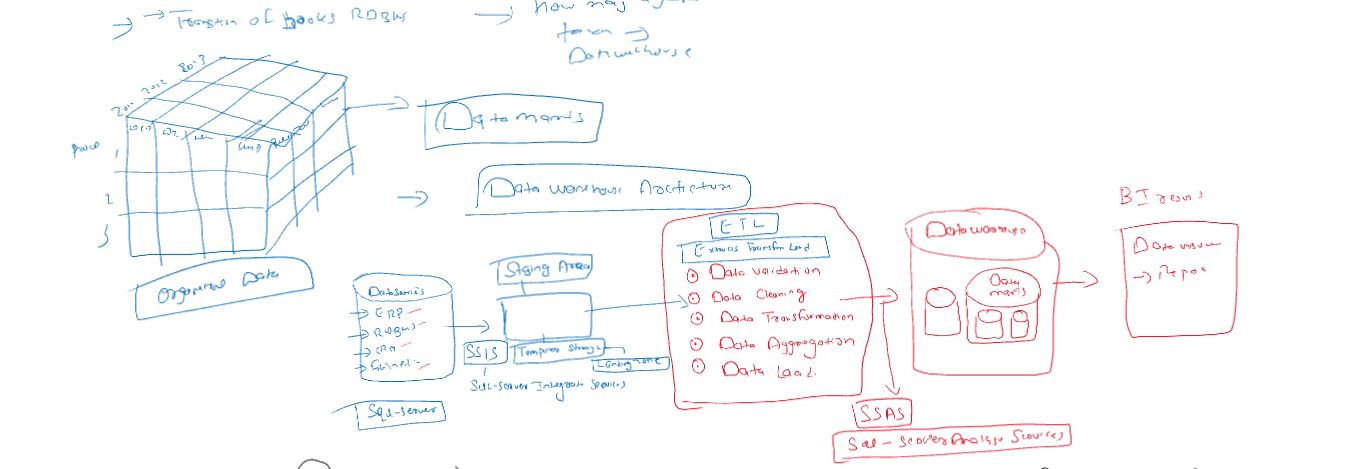
An **RDBMS** is a DBMS based on the **relational model**, where data is stored in **tables** with relationships between them.

* **Examples**: MySQL, Oracle, SQL Server, PostgreSQL

What is Data Warehousing?

A **data warehouse** is a centralized system used for storing large volumes of historical data from multiple sources, primarily for **analytics and reporting**.

* **Purpose**: Optimized for reading, analysis, and business intelligence (not frequent updates).
* **Examples**: Amazon Redshift, Snowflake, Google BigQuery
* **Related Terms**:
  + **ETL (Extract, Transform, Load)**: Process of moving and preparing data for warehousing.
  + **OLAP (Online Analytical Processing)**: Multidimensional analysis of business data.
  + **Star Schema / Snowflake Schema**: Common designs for organizing data in warehouses.



What is Datamart?

A **data mart** is a **subset** of a data warehouse, focused on a specific business area (like sales, finance, or HR).

* **Purpose**: Speeds up access for specific users or departments.
* **Example**: A sales data mart within a larger company-wide data warehouse.
* **Types**:
  + **Dependent Data Mart**: Extracted from an existing data warehouse.
  + **Independent Data Mart**: Created directly from operational systems.

What is Data Lake?

A **data lake** is a centralized repository that stores **raw, unstructured, semi-structured, and structured data** at any scale.

* **Examples**: AWS S3, Azure Data Lake, Hadoop HDFS
* **Purpose**: Allows storage of data without defining structure first (schema-on-read).
* **Related Terms**:
  + **Big Data**: Large and complex datasets that traditional databases can't handle efficiently.
  + **Schema-on-Read**: The data structure is applied when data is read, not when stored.
  + **Unstructured Data**: Data like images, logs, video, and text.

Difference between RDBMS and Data Warehousing:

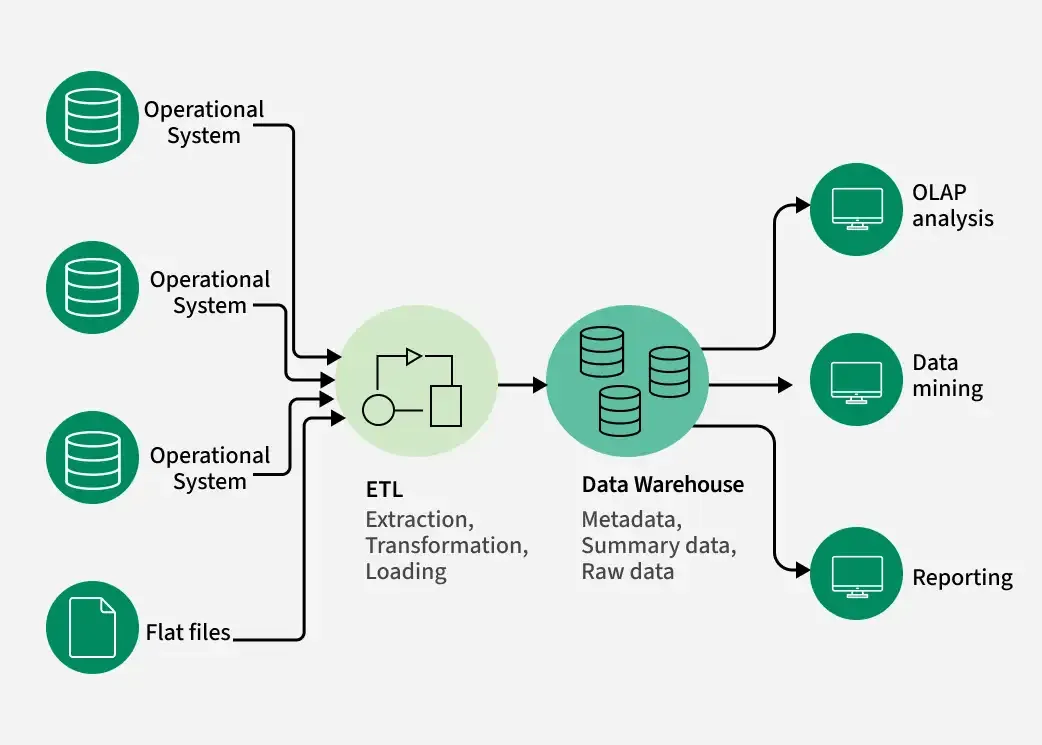
| **Aspect** | **RDBMS (Relational Database Management System)** | **Data Warehouse** |
| --- | --- | --- |
| **Primary Purpose** | Handles **day-to-day operations (OLTP)** | Handles **analytical queries & reporting (OLAP)** |
| **Data Type** | **Current, transactional data** | **Historical, aggregated data** |
| **Data Format** | **Normalized** tables for efficiency and integrity | **Denormalized** for faster querying and reporting |
| **Use Case** | CRUD operations for applications (insert, update, delete) | Complex queries for business insights |
| **Query Type** | **Simple, fast transactions** | **Complex queries and aggregations** |
| **Users** | Application developers, operational staff | Business analysts, data scientists, management |
| **Read/Write Ratio** | **Balanced read/write** operations | **High read**, low write (write-once, read-many) |
| **Latency** | Low latency, real-time access | Higher latency, not designed for real-time |
| **Data Volume** | Handles **smaller datasets** (GB to TB) | Handles **very large datasets** (TB to PB) |
| **Storage Type** | Row-based storage | Columnar or hybrid storage models |
| **Schema Type** | **Schema-on-write** (predefined schema) | **Schema-on-write** (but supports star/snowflake schemas) |
| **Data Sources** | Typically one application | Multiple heterogeneous sources (ERP, CRM, logs, etc.) |
| **Examples** | MySQL, PostgreSQL, Oracle, SQL Server | Amazon Redshift, Google BigQuery, Snowflake, Teradata |
| **Performance Optimization** | Optimized for fast **transactional processing** | Optimized for fast **read-intensive analytical queries** |
| **Backup/Restore** | Frequent, due to data changes | Periodic backups (data is mostly read-only) |
| **Indexing** | Highly indexed for quick transaction lookups | Indexed selectively for analytical efficiency |
| **Data Integrity** | Strong integrity constraints (PK, FK, ACID) | Integrity maintained during ETL, but less rigid |
| **Scalability** | Vertical scaling (limited horizontal) | Designed for **horizontal scaling** (distributed systems) |
| **Data History** | Usually does **not** maintain history | Maintains **historical snapshots** and audit trails |
| **ETL (Extract, Transform, Load)** | Rarely used inside an RDBMS itself | Central to data warehouse architecture |
| **Real-Time Use** | Yes, supports real-time transactions | Not ideal for real-time (used for periodic loads) |
| **Examples of Queries** | “Update customer address”, “Insert order” | “What were the monthly sales trends in 2023?” |
| **Data Redundancy** | Avoided using normalization | Allowed/used intentionally for query speed |
| **Data Modeling** | ER Modeling (Entity-Relationship) | Dimensional modeling (Star, Snowflake schema) |

**Types of Data Warehouses**

The different [types of Data Warehouses](https://www.geeksforgeeks.org/types-of-data-warehouses/)are:

1. **Enterprise Data Warehouse (EDW)**: A centralized warehouse that stores data from across the organization for analysis and reporting.
2. **Operational Data Store (ODS)**: Stores real-time operational data used for day-to-day operations, not for deep analytics.
3. **Data Mart**: A [subset](https://www.geeksforgeeks.org/data-marts-storage-component-of-hdfs/) of a data warehouse, focusing on a specific business area or department.
4. **Cloud Data Warehouse**: A data warehouse hosted in the cloud, offering scalability and flexibility.
5. **Big Data Warehouse**: Designed to store vast amounts of unstructured and structured data for big data analysis.
6. **Virtual Data Warehouse**: Provides access to data from multiple sources without physically storing it.
7. **Hybrid Data Warehouse**: Combines on-premises and cloud-based storage to offer flexibility.
8. **Real-time Data Warehouse**: Designed to handle real-time data streaming and analysis for immediate insights.

Date warehouse Architecture:



RDBMS architecture:

A diagram of a software system

AI-generated content may be incorrect.

What is Data Mart?

A **data mart** is a subject-oriented database that contains a **focused portion of the data warehouse**, designed to meet the specific requirements of a **particular department or business unit**.

Difference between Datamart and data warehouse:

| **Feature** | **Data Mart** | **Data Warehouse** |
| --- | --- | --- |
| **Scope** | Department-specific | Enterprise-wide |
| **Data Volume** | Smaller | Large |
| **Complexity** | Less complex | More complex |
| **Implementation** | Faster and cheaper | Requires more resources and planning |
| **Users** | Business users of a specific department | Company-wide data consumers |
| **Maintenance** | Easier to manage | More complex and centralized |

Types of Datamart:

| **Type** | **Description** |
| --- | --- |
| **Dependent** | Created by extracting data from a central **data warehouse**. Fully consistent with enterprise data. |
| **Independent** | Built **directly from operational systems**, without a data warehouse. Used in small organizations. |
| **Hybrid** | Combines features of both dependent and independent data marts. |

**Dependent Data Mart -**

A diagram of data storage

AI-generated content may be incorrect.

**2. Independent Data Mart -**

A diagram of data processing

AI-generated content may be incorrect.

**3. Hybrid Data Mart -**

A diagram of a data storage system

AI-generated content may be incorrect.

What is OLAP?

Abbreviation - Online Analytical Processing

It is a category of technologies that enables users to **interactively analyze multidimensional data** from multiple perspectives, typically for **business intelligence (BI)**, **reporting**, and **decision-making**.

Key Features:

| **Feature** | **Description** |
| --- | --- |
| **Multidimensional View** | Data is structured into **dimensions** and **measures** (e.g., sales by region, time, product) |
| **High-Speed Querying** | Optimized for **fast read-only access** on large volumes of data |
| **Complex Calculations** | Supports aggregations (SUM, AVG), trend analysis, rankings, etc. |
| **Drill-Down/Up** | Navigate data hierarchies (e.g., year → quarter → month) |
| **Slice & Dice** | Select and filter data along any dimension |
| **Pivoting** | Rearranging dimensions for different views |

Important Terms in OLAP:

| **Term** | **Explanation** |
| --- | --- |
| **Cube** | A multidimensional data structure used in OLAP |
| **Measure** | Numeric data values (e.g., sales amount, profit) |
| **Dimension** | Categories by which data is grouped (e.g., time, product, location) |
| **Hierarchy** | Levels within dimensions (e.g., Year → Quarter → Month → Day) |
| **Drill-down** | Going from summary to detailed data |
| **Roll-up** | Aggregating data up to a higher level |
| **Slice** | Filtering a cube by a single dimension value (e.g., all sales in 2023) |
| **Dice** | Filtering the cube on multiple dimension values (e.g., sales in Q1 2023 for Product A in Europe) |

Types of OLAP:

| **Type** | **Description** | **Storage** | **Performance** |
| --- | --- | --- | --- |
| **MOLAP** (Multidimensional OLAP) | Uses **precomputed cubes** stored in multidimensional databases | Fastest for fixed queries | Very high |
| **ROLAP** (Relational OLAP) | Uses **relational databases** with SQL for analysis | No pre-aggregation | Slower |
| **HOLAP** (Hybrid OLAP) | Combines both MOLAP and ROLAP approaches | Mixed | Balanced |

**Benefits of OLAP:**

* Fast performance for large datasets
* Multidimensional analysis
* Better support for business decision-making
* Easy to explore and visualize data

How does OLAP work?

An online analytical processing (OLAP) system works by collecting, organizing, aggregating, and analyzing data using the following steps:

1. The OLAP server collects data from multiple data sources, including relational databases and data warehouses.
2. Then, the extract, transform, and load (ETL) tools clean, aggregate, precalculate, and store data in an OLAP cube according to the number of dimensions specified.
3. Business analysts use OLAP tools to query and generate reports from the multidimensional data in the OLAP cube.

OLAP uses Multidimensional Expressions (MDX) to query the OLAP cube. MDX is a query, like SQL, that provides a set of instructions for manipulating databases.

OLAP Storage Approaches:

| **Type** | **Data Storage** | **Speed** | **Flexibility** | **Example Use Case** |
| --- | --- | --- | --- | --- |
| **MOLAP** | Multidimensional cubes | Very Fast | Less flexible | Financial dashboards |
| **ROLAP** | Relational DB | Slower | Highly flexible | Dynamic ad-hoc analysis |
| **HOLAP** | Mixed | Balanced | Balanced | Scalable enterprise reporting |

OLAP Architecture:

┌──────────────────────────────┐

│ 1. Front-End Tools │ ← User Interface Layer

│ (BI Tools: Tableau, Power BI)│

└─────────────▲────────────────┘

│

┌─────────────┴──────────────┐

│ 2. OLAP Server (Engine) │ ← Analytical Processing Layer

│(MOLAP / ROLAP / HOLAP engine)│

└─────────────▲──────────────┘

│

┌─────────────┴──────────────┐

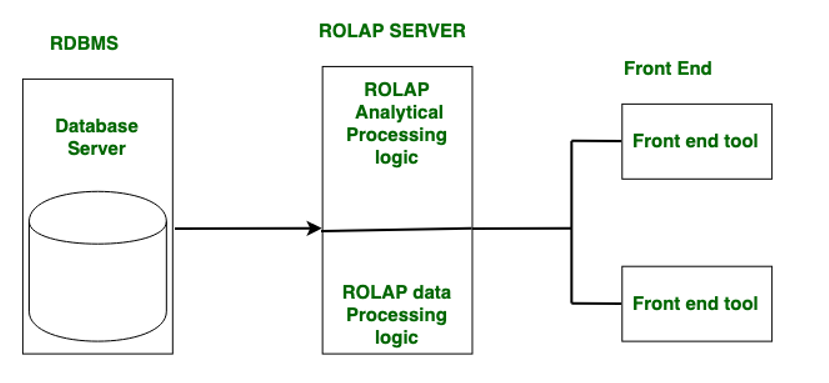
│ 3. Data Warehouse / DBMS │ ← Data Storage Layer

│(Star/Snowflake Schema) │

└────────────────────────────┘

**Relational OLAP (ROLAP):**

Relational On-Line Analytical Processing (ROLAP) is primarily used for data stored in a relational database, where both the base data and dimension tables are stored as relational tables. ROLAP servers are used to bridge the gap between the relational back-end server and the client's front-end tools. ROLAP servers store and manage warehouse data using RDBMS, and OLAP middleware fills in the gaps.



**Multidimensional OLAP (MOLAP):**

**MOLAP (Multidimensional OLAP)** stores data in **pre-aggregated multidimensional cubes**, allowing for **very fast querying**, especially for **fixed, structured analysis**. It offers **high performance** but less flexibility and scalability compared to other OLAP types.

A diagram of a computer process

AI-generated content may be incorrect.

**Hybrid OLAP (HOLAP):**

ROLAP and MOLAP are combined in Hybrid On-Line Analytical Processing (HOLAP). HOLAP offers greater scalability than ROLAP and faster computation than MOLAP.HOLAP is a hybrid of ROLAP and MOLAP. HOLAP servers are capable of storing large amounts of detailed data. On the one hand, HOLAP benefits from ROLAP's greater scalability. HOLAP, on the other hand, makes use of cube technology for faster performance and summary-type information. Because detailed data is stored in a relational database, cubes are smaller than MOLAP.



**Facts and Dimensions in Data Warehousing**

In a data warehouse, **facts and dimensions** are the core components used to organize data into a **multidimensional model** (like OLAP cubes). This structure supports efficient analysis and reporting.

Fact Table

A **Fact Table** stores **quantitative, measurable data** (facts) related to business processes — such as **sales amount, revenue, units sold**, etc.

**Characteristics:**

* Contains **numeric values** (measures)
* **Foreign keys** referencing dimension tables
* Typically **large in size**
* **Additive or semi-additive**

Dimension Table

A **Dimension Table** contains **descriptive attributes** (context) related to the facts — such as **product name, category, region, customer name**, etc.

**Characteristics:**

* Used to **filter, group, label** facts
* Typically **textual or categorical**
* **Smaller** in size than fact tables
* Contains **hierarchies** (e.g., Year → Quarter → Month)

Difference Between Facts and Dimensions:

| **Component** | **Fact Table** | **Dimension Table** |
| --- | --- | --- |
| Contains | Measures / Metrics | Descriptive Attributes |
| Type of Data | Quantitative | Qualitative / Categorical |
| Size | Large | Smaller |
| Keys | Foreign keys to dimensions | Primary keys used in fact table |
| Examples | Sales amount, Profit, Quantity sold | Product name, Date, Region, Customer name |
| Usage | Aggregation, Calculations | Filtering, Grouping, Labeling |



**Schema in Data Warehousing:**

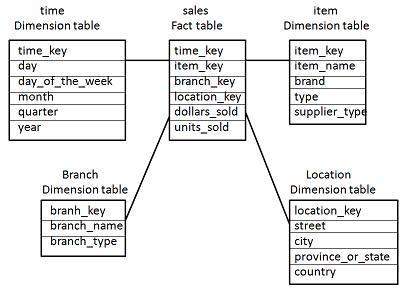
In data warehousing, a **schema** defines the **structure and organization** of data — how fact and dimension tables are arranged and how they relate to each other. It’s essentially the **blueprint** for designing a data warehouse.

Three Types of Schemas are there in Data warehousing:

* Star Schema
* Snowflakes Schema
* Fact Constellation Schema (Galaxy Schema)

Star Schema:

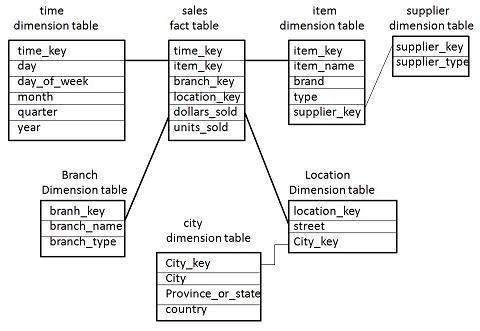
* Each dimension in a star schema is represented with only one-dimension table.
* This dimension table contains the set of attributes.
* The following diagram shows the sales data of a company with respect to the four dimensions, namely time, item, branch, and location.



* There is a fact table at the center. It contains the keys to each of four dimensions.
* The fact table also contains the attributes, namely dollars sold and units sold.

Snowflake Schema

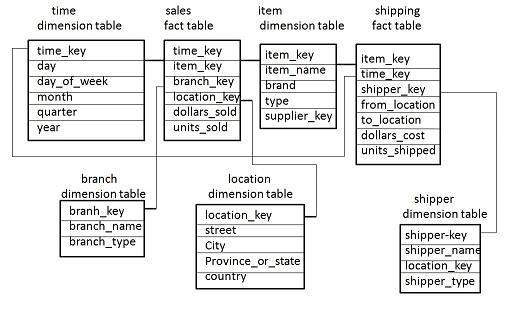
* Some dimension tables in the Snowflake schema are normalized.
* The normalization splits up the data into additional tables.
* Unlike Star schema, the dimensions table in a snowflake schema are normalized. For example, the item dimension table in star schema is normalized and split into two dimension tables, namely item and supplier table.



* Now the item dimension table contains the attributes item\_key, item\_name, type, brand, and supplier-key.
* The supplier key is linked to the supplier dimension table. The supplier dimension table contains the attributes supplier\_key and supplier\_type.

Fact Constellation Schema

* A fact constellation has multiple fact tables. It is also known as galaxy schema.
* The following diagram shows two fact tables, namely sales and shipping.



* The sales fact table is same as that in the star schema.
* The shipping fact table has the five dimensions, namely item\_key, time\_key, shipper\_key, from\_location, to\_location.
* The shipping fact table also contains two measures, namely dollars sold and units sold.
* It is also possible to share dimension tables between fact tables. For example, time, item, and location dimension tables are shared between the sales and shipping fact table.

**MONGODB**

* Mongo db is nosql document database
* stores data in json like documents (internal uses BSON)
* Flexible schema, ideal for changing data

How to choose database?

Databases can be chosen according to the requirement,

| **Feature** | **SQL (Relational DB)** | **NoSQL (Non-Relational DB)** |
| --- | --- | --- |
| **Data Structure** | Tables (rows & columns) | Documents, Key-Value, Graph, or Column-based |
| **Schema** | Fixed schema (predefined structure) | Dynamic or flexible schema |
| **Scalability** | Vertical (scale-up: bigger machine) | Horizontal (scale-out: more machines) |
| **Transactions** | ACID-compliant (reliable, strong consistency) | Often BASE (eventual consistency) |
| **Query Language** | Structured Query Language (SQL) | Various: MongoDB uses JSON-like query syntax |
| **Best Use Cases** | Complex queries, transactions, joins | Big data, real-time apps, flexible data |
| **Examples** | MySQL, PostgreSQL, SQLite, SQL Server, Oracle | MongoDB, Cassandra, Redis, DynamoDB, CouchDB |

**Model Coding Challenge (9-6-25)**

**MongoDB**

Creating Database Students,

A screenshot of a computer

AI-generated content may be incorrect.

The `students` collection includes:

{

"name": "Amit Kumar",

"age": 17,

"gender": "Male",

"grade": "11th",

"subjects": ["Math", "Physics", "English"],

"marks": { "Math": 82, "Physics": 91, "English": 75 },

"address": {

"city": "Delhi",

"pincode": "110001"

}

}

Q1. Insert 5 student records

db.students.insertMany([

{

name: "Amit Kumar",

age: 17,

gender: "Male",

grade: "11th",

subjects: ["Math", "Physics", "English"],

marks: { Math: 82, Physics: 91, English: 75 },

address: { city: "Delhi", pincode: "110001" }

},

{

name: "Sneha Verma",

age: 16,

gender: "Female",

grade: "10th",

subjects: ["Biology", "Chemistry", "English"],

marks: { Biology: 78, Chemistry: 88, English: 92 },

address: { city: "Mumbai", pincode: "400001" }

},

{

name: "Rahul Singh",

age: 18,

gender: "Male",

grade: "12th",

subjects: ["Math", "Computer", "Physics"],

marks: { Math: 95, Computer: 89, Physics: 93 },

address: { city: "Delhi", pincode: "110002" }

},

{

name: "Anjali Mehra",

age: 17,

gender: "Female",

grade: "11th",

subjects: ["Math", "English", "History"],

marks: { Math: 39, English: 42, History: 78 },

address: { city: "Bangalore", pincode: "560001" }

},

{

name: "Karan Patel",

age: 15,

gender: "Male",

grade: "9th",

subjects: ["Science", "English", "Math"],

marks: { Science: 65, English: 38, Math: 41 },

address: { city: "Ahmedabad", pincode: "380001" }

}

])

A screenshot of a computer

AI-generated content may be incorrect.

Q2. Find all students who take “Math” as a subject

db.students.find(

{ subjects: "Math" },

{ \_id: 0, name: 1, grade: 1, subjects: 1 }

)

A black screen with a black background

AI-generated content may be incorrect.

Q3. Find students from city "Delhi" scoring more than 85 in Physics

db.students.find(

{

"address.city": "Delhi",

"marks.Physics": { $gt: 85 }

},

{

\_id: 0,

name: 1,

marks: 1,

"address.city": 1

}

)

A computer screen shot of a black screen

AI-generated content may be incorrect.

Q4. Sort all students by English marks (descending) and show top 3

db.students.find(

{ "marks.English": { $exists: true } },

{ \_id: 0, name: 1, subjects: 1, marks: 1 }

).sort({ "marks.English": -1 }).limit(3)

A screenshot of a computer program

AI-generated content may be incorrect.

Q5. Group students by city and count how many are from each

db.students.aggregate([

{

$group: {

\_id: "$address.city",

total\_students: { $sum: 1 }

}

}

])

A screen shot of a computer program

AI-generated content may be incorrect.

Q6. Update all students who scored less than 40 in any subject

db.students.updateMany(

{

$or: [

{ "marks.Math": { $lt: 40 } },

{ "marks.English": { $lt: 40 } },

{ "marks.Physics": { $lt: 40 } },

{ "marks.Computer": { $lt: 40 } },

{ "marks.Chemistry": { $lt: 40 } },

{ "marks.Biology": { $lt: 40 } },

{ "marks.Science": { $lt: 40 } },

{ "marks.History": { $lt: 40 } }

]

},

{ $set: { status: "Needs Improvement" } }

)

A screen shot of a computer program

AI-generated content may be incorrect.

Q7. Calculate average marks of all students in each subject

db.students.aggregate([

{

$group: {

\_id: null,

avgMath: { $avg: "$marks.Math" },

avgPhysics: { $avg: "$marks.Physics" },

avgEnglish: { $avg: "$marks.English" },

avgComputer: { $avg: "$marks.Computer" },

avgBiology: { $avg: "$marks.Biology" },

avgChemistry: { $avg: "$marks.Chemistry" },

avgScience: { $avg: "$marks.Science" },

avgHistory: { $avg: "$marks.History" }

}

}

])

A screenshot of a computer screen

AI-generated content may be incorrect.

**Project – 2**

**Coding Challenge – 2 (09-06-2025)**

**Online Store Order Management - MongoDB**

Creating new database and named it as OnlineStore,

A screenshot of a computer program

AI-generated content may be incorrect.

Q1. Insert 5 different orders

db.orders.insertMany([

{

order\_id: "ORD1001",

order\_date: "2024-06-01",

customer: { name: "Deepa Roy", email: "deepa@example.com", phone: "9876543210" },

shipping\_address: { street: "123 Green Park", city: "Mumbai", state: "MH", pincode: "400001" },

items: [

{ product: "Wireless Mouse", price: 599, quantity: 2 },

{ product: "Keyboard", price: 899, quantity: 1 }

],

total\_amount: 2097,

payment\_status: "Paid",

delivery\_status: "Delivered"

},

{

order\_id: "ORD1002",

order\_date: "2024-06-03",

customer: { name: "Arjun Mehta", email: "arjun@example.com", phone: "9998887777" },

shipping\_address: { street: "45 Sector 9", city: "Delhi", state: "DL", pincode: "110001" },

items: [

{ product: "Laptop Bag", price: 1299, quantity: 1 }

],

total\_amount: 1299,

payment\_status: "Paid",

delivery\_status: "Shipped"

},

{

order\_id: "ORD1003",

order\_date: "2024-06-04",

customer: { name: "Neha Sharma", email: "neha@example.com", phone: "8887776666" },

shipping\_address: { street: "78 Hill View", city: "Chennai", state: "TN", pincode: "600001" },

items: [

{ product: "Monitor", price: 10999, quantity: 1 }

],

total\_amount: 10999,

payment\_status: "Pending",

delivery\_status: "Not Shipped"

},

{

order\_id: "ORD1004",

order\_date: "2024-06-05",

customer: { name: "Suresh Babu", email: "suresh@example.com", phone: "7776665555" },

shipping\_address: { street: "Plot 22", city: "Hyderabad", state: "TS", pincode: "500001" },

items: [

{ product: "Pen Drive", price: 399, quantity: 3 },

{ product: "Notebook", price: 99, quantity: 5 }

],

total\_amount: 1392,

payment\_status: "Paid",

delivery\_status: "Processing"

},

{

order\_id: "ORD1005",

order\_date: "2024-06-06",

customer: { name: "Meena Das", email: "meena@example.com", phone: "6665554444" },

shipping\_address: { street: "Lake Road", city: "Delhi", state: "DL", pincode: "110002" },

items: [

{ product: "Desk Lamp", price: 799, quantity: 1 },

{ product: "Chair", price: 2499, quantity: 1 },

{ product: "Table", price: 3499, quantity: 1 }

],

total\_amount: 6797,

payment\_status: "Paid",

delivery\_status: "Delivered"

}

])

A screen shot of a computer program

AI-generated content may be incorrect.

Q2. Find "Paid" but not "Delivered" orders

db.orders.find({

payment\_status: "Paid",

delivery\_status: { $ne: "Delivered" }

})

A screen shot of a computer program

AI-generated content may be incorrect.

Q3. Orders with at least one product priced > 1000

db.orders.find({

"items.price": { $gt: 1000 }

})

A screen shot of a computer program

AI-generated content may be incorrect.

Q4. Orders shipped to Mumbai or Delhi, sorted by date

db.orders.find(

{ "shipping\_address.city": { $in: ["Mumbai", "Delhi"] } },

{ \_id: 0, order\_id: 1, "shipping\_address.city": 1, order\_date: 1 }

).sort({ order\_date: -1 })

A computer screen with text and numbers

AI-generated content may be incorrect.

Q5. Group by payment\_status

db.orders.aggregate([

{ $group: { \_id: "$payment\_status", count: { $sum: 1 } } }

])

A screen shot of a computer program

AI-generated content may be incorrect.

Q6. Total revenue from delivered orders

db.orders.aggregate([

{ $match: { delivery\_status: "Delivered" } },

{ $group: { \_id: null, total\_revenue: { $sum: "$total\_amount" } } }

])

A screen shot of a computer

AI-generated content may be incorrect.

Q7. Add "priority: High" where items > 3

db.orders.updateMany(

{ $expr: { $gt: [{ $size: "$items" }, 3] } },

{ $set: { priority: "High" } }

)

A screen shot of a computer

AI-generated content may be incorrect.

Q8. Count how many times each product was ordered

db.orders.aggregate([

{ $unwind: "$items" },

{ $group: {

\_id: "$items.product",

total\_quantity: { $sum: "$items.quantity" }

}}

])

A screen shot of a computer program

AI-generated content may be incorrect.

Q9. Add "USB Cable" to order\_id "ORD1001"

db.orders.updateOne(

{ order\_id: "ORD1001" },

{

$push: {

items: { product: "USB Cable", price: 199, quantity: 1 }

}

}

)

A screen shot of a computer program

AI-generated content may be incorrect.

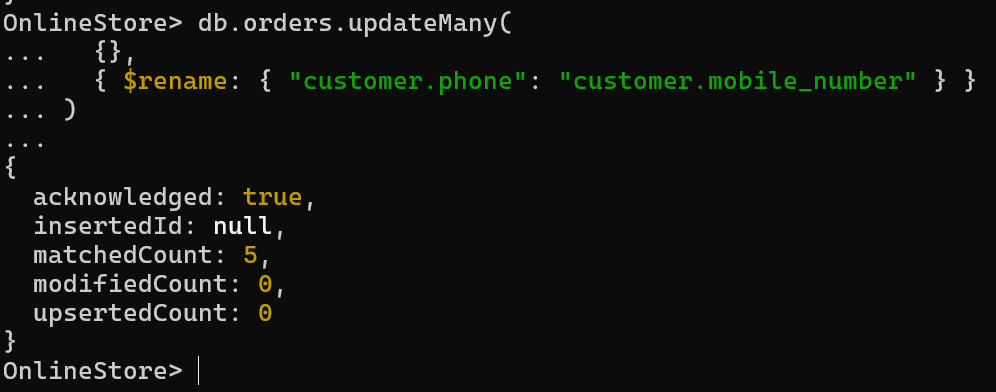
Q10. Rename "phone" to "mobile\_number" in customer

db.orders.updateMany(

{},

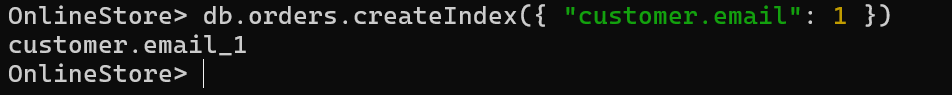
{ $rename: { "customer.phone": "customer.mobile\_number" } }

)



Q11. Create index on customer email

db.orders.createIndex({ "customer.email": 1 })



Q12. Export the data using `mongoexport` for reporting purposes.

mongoexport --db=OnlineStore --collection=orders --out=orders.json

--(for specific path)

mongoexport --db=OnlineStore --collection=orders --out=C:\Users\sjlak\Downloads\orders.json  
  
A computer screen with white text

AI-generated content may be incorrect.

**Day 5 – (10-06-2025)**

**Python Refresher & Advanced Python for Data Engineering**

* **Core Python Refresher**

**Python Basics**

* **Features:** Easy syntax, interpreted, dynamic typing, huge libraries.
* **First Program:**
* print("Hello, Data Engineers!")
* **Variables & Data Types**
* name = "Laks" # String
* age = 21 # Integer
* score = 94.5 # Float
* passed = True # Boolean

**Control Structures**

for i in range(3):

print("Looping", i)

if score > 90:

print("Excellent!")

elif score >= 75:

print("Good job")

else:

print("Keep learning")

**Collections Overview**

| **Type** | **Example** |
| --- | --- |
| List | marks = [85, 78, 92] |
| Dictionary | {"name": "Sakshi", "score": 95} |
| Set | {1, 2, 3} |
| Tuple | ("Alice", [85, 78, 92]) |

**Functions & Lambda**

def greet(name="Friend"):

return f"Hello, {name}!"

add = lambda x, y: x + y

print(add(10, 5)) # Output: 15

**OOP in Python**

class Student:

def \_\_init\_\_(self, name, marks):

self.name = name

self.marks = marks

def average(self):

return sum(self.marks) / len(self.marks)

s = Student("Sakshi", [85, 90, 95])

print(s.average()) # Output: 90.0

**File Handling & Exception**

try:

with open("data.csv", "r") as f:

content = f.read()

except FileNotFoundError:

print("File not found!")

**Modules and Packages**

# mymodule.py

def add(a, b):

return a + b

# main.py

import mymodule

print(mymodule.add(2, 3)) # Output: 5

**Data Engineering Essentials**

**CSV / JSON Handling with Pandas**

import pandas as pd

df = pd.read\_csv("students.csv")

print(df.head())

df.to\_json("students.json")

**MySQL with Python**

import mysql.connector

conn = mysql.connector.connect(

host="localhost", user="root", password="1234", database="school"

)

cursor = conn.cursor()

cursor.execute("SELECT \* FROM students")

for row in cursor.fetchall():

print(row)

**Data Cleaning with NumPy & Pandas**

import numpy as np

import pandas as pd

arr = np.array([[1, np.nan, 3], [4, 5, np.nan]])

df = pd.DataFrame(arr)

df.fillna(0, inplace=True)

**Task 1 – Students Marks Management System**

students = [

("lakshan",[88,89,90]),

("haley",[89,88,90]),

("harish",[90,91,85])

]

print("Average Marks of Each Student: ")

topper\_name = ""

highest\_avg = 0

for name, marks in students:

avg = sum(marks)/len(marks)

print(f"{name} - Average: {avg:.2f}")

if avg > highest\_avg:

highest\_avg = avg

topper\_name = name

print("\nTopper:",topper\_name)

students.append(("Rock", [80,89,95]))

for i in range(len(students)):

if students[i][0] == "Jennifer":

students[i][1][1] = 89

break

print("\n Updated Student's List: ")

for name, marks in students:

print(f"{name}: {marks}")

student\_stats = []

for name, marks in students:

total = sum(marks)

avg = total/len(marks)

student\_stats.append((name, total, round(avg, 2)))

student\_stats.sort(key=lambda x: x[2], reverse=True)

print("\n Students sorted by Average Marks: ")

for name, total, avg in student\_stats:

print(f"{name} - Total: {total}, Average: {avg}")

A screenshot of a computer program

AI-generated content may be incorrect.

**Task 2 – Smart City Project**

import numpy as np

temps = np.array([

[28, 30, 29, 31, 32, 33, 34],

[35, 34, 36, 37, 38, 36, 35],

[22, 23, 24, 25, 26, 27, 28],

[31, 30, 29, 28, 27, 26, 25],

[39, 38, 40, 41, 42, 40, 39]

])

days = ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"]

#1. Function: Average temperature of a city for the week

def average\_temperature(city\_index):

return np.mean(temps[city\_index])

# 2. Function: Hottest day index for a city

def hottest\_day(city\_index):

return np.argmax(temps[city\_index])

# 3. Loop through all cities and print average temperature and hottest day

print("City-wise Average Temperature and Hottest Day:")

for i in range(5):

avg\_temp = average\_temperature(i)

hot\_day\_index = hottest\_day(i)

hot\_day\_name = days[hot\_day\_index]

print(f"City {i+1}: Avg Temp = {avg\_temp:.2f}°C, Hottest Day = {hot\_day\_name} (Index: {hot\_day\_index})")

print()

#4. Function: Day-wise average temperature across all cities

def day\_wise\_avg():

return np.mean(temps, axis=0)

# Print day-wise average

day\_avg = day\_wise\_avg()

print("Day-wise Average Temperatures Across All Cities:")

for i in range(7):

print(f"{days[i]}: {day\_avg[i]:.2f}°C")

print()

#5. Loop to find:

# City with highest weekly average

city\_avgs = [average\_temperature(i) for i in range(5)]

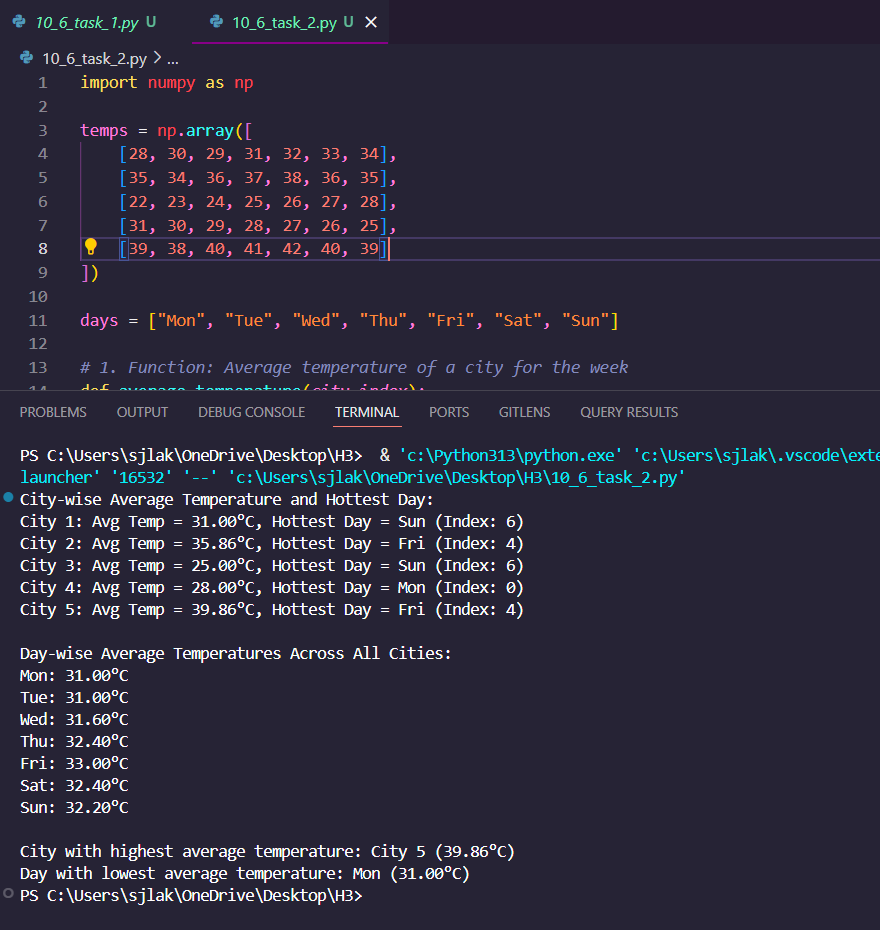
highest\_city\_index = np.argmax(city\_avgs)

# Day with lowest average temperature

lowest\_day\_index = np.argmin(day\_avg)

print(f"City with highest average temperature: City {highest\_city\_index + 1} ({city\_avgs[highest\_city\_index]:.2f}°C)")

print(f” Day with lowest average temperature: {days[lowest\_day\_index]} ({day\_avg[lowest\_day\_index]:.2f}°C)")



**Python Lists**

In Python, **lists** are one of the most used data structures. A list stores a collection of items in a specific order. You can think of it like a container that holds different types of values.

**📦 What is a List?**

A list is a collection of items separated by commas and enclosed in square brackets [ ].

fruits = ['apple', 'banana', 'cherry']

numbers = [10, 20, 30, 40]

mixed = ['hello', 123, True]

* Lists can store different data types.
* Indexing starts from **0** (just like string indexing).

**🔍 Accessing List Elements**

Use square brackets with the index number to get an item from the list.

colors = ['red', 'green', 'blue', 'yellow']

print(colors[0]) # red

print(colors[2]) # blue

print(colors[-1]) # yellow (last item)

You can also access a **range** of items using **slicing**:

print(colors[1:3]) # ['green', 'blue']

**✏️ Updating List Elements**

You can change any item in the list by assigning a new value at a specific index.

animals = ['cat', 'dog', 'rabbit']

animals[1] = 'parrot'

print(animals) # ['cat', 'parrot', 'rabbit']

**➕ Adding Items to a List**

Use the append() method to add an item to the end:

languages = ['Python', 'Java']

languages.append('C++')

print(languages) # ['Python', 'Java', 'C++']

You can also use insert(index, value) to add an item at a specific position:

languages.insert(1, 'JavaScript')

print(languages) # ['Python', 'JavaScript', 'Java', 'C++']

**❌ Removing Items from a List**

**Using del:**

numbers = [10, 20, 30, 40]

del numbers[2]

print(numbers) # [10, 20, 40]

**Using remove():**

names = ['Alice', 'Bob', 'Charlie']

names.remove('Bob')

print(names) # ['Alice', 'Charlie']

**Using pop():**

scores = [100, 90, 80]

scores.pop() # removes last item

scores.pop(0) # removes item at index 0

print(scores) # [90]

**📏 Useful List Functions**

**1. len(): Count of items**

print(len(['a', 'b', 'c'])) # 3

**2. max() / min(): Highest / Lowest value**

marks = [55, 89, 73]

print(max(marks)) # 89

print(min(marks)) # 55

**3. list(): Convert other types to list**

text = "hello"

print(list(text)) # ['h', 'e', 'l', 'l', 'o']

**🔁 List Operations**

| **Operation** | **Example** | **Result** |
| --- | --- | --- |
| Length | len([1, 2, 3]) | 3 |
| Concatenation | [1, 2] + [3, 4] | [1, 2, 3, 4] |
| Repetition | ['hi'] \* 3 | ['hi', 'hi', 'hi'] |
| Membership | 'apple' in ['apple', 'mango'] | True |
| Iteration | for x in [1, 2, 3]: print(x) | prints 1 2 3 |

**🧰 More List Methods**

| **Method** | **Description** | **Example** |
| --- | --- | --- |
| append(x) | Adds x to end | mylist.append(5) |
| extend(seq) | Adds all items from seq | mylist.extend([6, 7]) |
| count(x) | Counts how many times x appears | mylist.count(2) |
| index(x) | Finds the first index of x | mylist.index('apple') |
| reverse() | Reverses the list in place | mylist.reverse() |
| sort() | Sorts the list | mylist.sort() |

**✅ Examples to Try**

shopping = ['milk', 'bread', 'eggs']

shopping.append('butter')

shopping.remove('bread')

shopping.insert(1, 'juice')

print(shopping)

**Output:**

['milk', 'juice', 'eggs', 'butter']