Let’s go step-by-step — first **SQL (Structured Query Language)** and then **NoSQL (Non-relational databases)** — with clear definitions, examples, differences, and use-cases.

**🧠 INTRODUCTION TO SQL**

**1️⃣ What is SQL?**

**SQL (Structured Query Language)** is a **standard language** used to **store, manipulate, and retrieve data** in **relational databases (RDBMS)**.

It was developed in the 1970s by IBM and later adopted as an international standard (ANSI SQL).

SQL works with data stored in **tables (rows & columns)** where relationships are maintained using **keys**.

**2️⃣ SQL is used for:**

| **Purpose** | **Example Commands** |
| --- | --- |
| **Data Definition (DDL)** | CREATE, ALTER, DROP, TRUNCATE |
| **Data Manipulation (DML)** | INSERT, UPDATE, DELETE, SELECT |
| **Data Querying** | SELECT ... FROM ... WHERE ... |
| **Transaction Control (TCL)** | COMMIT, ROLLBACK, SAVEPOINT |
| **Data Control (DCL)** | GRANT, REVOKE |

**3️⃣ Example SQL Workflow**

Let’s create a **College Database** step-by-step 👇

CREATE DATABASE CollegeDB;

USE CollegeDB;

-- Table for departments

CREATE TABLE Department (

DeptID INT PRIMARY KEY,

DeptName VARCHAR(50) UNIQUE

);

-- Table for students

CREATE TABLE Student (

StudentID INT PRIMARY KEY,

Name VARCHAR(50),

Gender CHAR(1),

DeptID INT,

FOREIGN KEY (DeptID) REFERENCES Department(DeptID)

);

-- Insert data

INSERT INTO Department VALUES (1, 'Computer Science'), (2, 'Mechanical');

INSERT INTO Student VALUES (101, 'Mayuri', 'F', 1), (102, 'Ravi', 'M', 2);

-- Query data

SELECT s.StudentID, s.Name, d.DeptName

FROM Student s

JOIN Department d ON s.DeptID = d.DeptID;

✅ Output:

| **StudentID** | **Name** | **DeptName** |
| --- | --- | --- |
| 101 | Mayuri | Computer Science |
| 102 | Ravi | Mechanical |

**4️⃣ Features of SQL**

* Simple, declarative syntax (“what to do”, not “how”)
* Portable (works across RDBMS with minor syntax changes)
* Enforces **data integrity** with constraints (PK, FK, CHECK, UNIQUE)
* Allows **joins**, **aggregations**, **subqueries**
* Supports **transactions (ACID)**

**5️⃣ Popular SQL Databases (RDBMS)**

* **MySQL**
* **PostgreSQL**
* **Oracle Database**
* **Microsoft SQL Server**
* **SQLite**

**💡 Now let’s move to NoSQL**

**🚀 INTRODUCTION TO NoSQL**

**1️⃣ What is NoSQL?**

**NoSQL** = “Not Only SQL” — a group of **non-relational databases** designed for **flexibility, scalability, and performance**.

Unlike SQL databases (which use fixed schemas and tables), NoSQL databases use **dynamic schemas** and store data in **different formats**:

* Documents (JSON/XML)
* Key-Value pairs
* Columns
* Graphs

**2️⃣ Why NoSQL?**

* To handle **huge volumes of unstructured or semi-structured data**
* To support **high scalability** (distributed systems, big data)
* To provide **flexible schema** (easy to add new fields)
* To ensure **faster reads/writes** for web-scale applications

**3️⃣ Types of NoSQL Databases**

| **Type** | **Structure** | **Example Databases** | **Best Used For** |
| --- | --- | --- | --- |
| **Document-based** | JSON-like documents | MongoDB, CouchDB | User profiles, content, catalogs |
| **Key-Value store** | Key → Value pairs | Redis, DynamoDB | Caching, sessions, preferences |
| **Column-family** | Columns grouped together | Cassandra, HBase | Analytics, IoT, logs |
| **Graph-based** | Nodes and edges | Neo4j, Amazon Neptune | Social networks, recommendations |

**4️⃣ Example — MongoDB (Document Database)**

MongoDB stores data in **collections** (instead of tables).  
Each record is a **document** in **JSON** format.

// Example document in MongoDB

{

"StudentID": 101,

"Name": "Mayuri",

"Gender": "F",

"Department": {

"DeptID": 1,

"DeptName": "Computer Science"

}

}

**Inserting Data:**

db.students.insertOne({

StudentID: 102,

Name: "Ravi",

Gender: "M",

Department: { DeptID: 2, DeptName: "Mechanical" }

});

**Querying Data:**

db.students.find({ "Department.DeptName": "Computer Science" });

✅ Output (as JSON document):

{ "\_id": 1, "StudentID": 101, "Name": "Mayuri", "Gender": "F", "Department": { "DeptID": 1, "DeptName": "Computer Science" } }

**5️⃣ Differences — SQL vs NoSQL**

| **Feature** | **SQL (RDBMS)** | **NoSQL** |
| --- | --- | --- |
| **Data Model** | Relational (tables, rows, columns) | Non-relational (documents, key-value, graphs, etc.) |
| **Schema** | Fixed | Dynamic / Flexible |
| **Query Language** | SQL | No fixed standard (JSON-based APIs) |
| **Joins** | Supported | Not usually (data embedded instead) |
| **Scalability** | Vertical (add more CPU/RAM) | Horizontal (add more servers) |
| **ACID Transactions** | Strong support | Varies (BASE / eventual consistency) |
| **Examples** | MySQL, PostgreSQL, Oracle | MongoDB, Cassandra, Redis, Neo4j |
| **Best For** | Structured, consistent data | Big Data, flexible schema, high scale apps |

**6️⃣ BASE Model (NoSQL consistency)**

* **B**asically **A**vailable
* **S**oft state (data may change over time)
* **E**ventually consistent (all copies become consistent later)

Contrasts with **ACID** of SQL systems.

**7️⃣ Use Cases — SQL vs NoSQL**

| **Scenario** | **Recommended** |
| --- | --- |
| Banking, Finance | SQL (strong consistency) |
| Social Networks | NoSQL (graph/document) |
| E-commerce Product Catalog | NoSQL (document) |
| Employee/College Management System | SQL |
| IoT, Sensor Data | NoSQL (column or time-series) |
| Data Warehouse/Analytics | SQL (with OLAP extensions) |

**8️⃣ Popular NoSQL Software**

| **Category** | **Example** | **Description** |
| --- | --- | --- |
| Document | **MongoDB**, Couchbase | JSON-like storage |
| Key-Value | **Redis**, Amazon DynamoDB | Fast caching & lookup |
| Column-Family | **Apache Cassandra**, HBase | Distributed large data |
| Graph | **Neo4j**, Amazon Neptune | Relationship modeling |

**9️⃣ Real-world Example**

**SQL (E-commerce Orders)**

SELECT CustomerName, ProductName, Quantity

FROM Orders

JOIN Customers USING(CustomerID)

JOIN Products USING(ProductID);

**NoSQL (MongoDB)**

db.orders.find({

"CustomerName": "Mayuri"

}, {

"ProductName": 1,

"Quantity": 1

});

**🔚 Summary**

| **Concept** | **SQL** | **NoSQL** |
| --- | --- | --- |
| Structure | Tabular | Flexible (JSON, key-value, graph) |
| Schema | Fixed | Dynamic |
| Scalability | Vertical | Horizontal |
| Consistency | Strong (ACID) | Eventual (BASE) |
| Example | MySQL, PostgreSQL | MongoDB, Cassandra, Redis |

**✅ Conclusion**

* **SQL (RDBMS)** → Best for structured, consistent, transactional systems (banking, ERP, college DB).
* **NoSQL** → Best for high-scale, flexible, rapidly changing data (social media, IoT, e-commerce).
* **Modern approach** → Use **both** where needed (polyglot persistence).