What is NoSQL Database?

NoSQL stands for **"Not Only SQL"**.
It is a type of database designed to **store, retrieve, and manage large volumes of unstructured, semi-structured, or rapidly changing data** that traditional **RDBMS (Relational Database Management Systems)** cannot efficiently handle.

Unlike relational databases, NoSQL databases **do not require fixed table schemas** and **can scale horizontally** across many servers.

Why NoSQL is Required

Traditional **RDBMS** (like MySQL, Oracle, PostgreSQL) work well for structured data and fixed schema.
However, with the rise of **Big Data, IoT, social media, and cloud computing**, new challenges emerged:

Problem in RDBMS	**Solution in NoSQL**
Rigid schema (tables, columns)	Schema-less or dynamic schema
Difficult horizontal scaling	Easily scalable across servers
Performance issues with huge data	High-speed read/write operations
Not suitable for unstructured data (e.g. images, JSON, videos)	Supports unstructured/semi-structured data
Complex joins slow down performance	Data is often stored in a denormalized way for faster access

Types of NoSQL Databases

1. Document-oriented Databases

- Store data as **JSON, BSON, or XML documents**.
- Each document contains all the information for a record.
- No need for predefined schema.

Examples: MongoDB, CouchDB, Firebase Firestore

Use Case: Content management systems, e-commerce product catalogs.

Example Document:

```
{
  "student_id": 101,
  "name": "Amit Kumar",
  "courses": ["DBMS", "AI", "Python"],
  "marks": {"DBMS": 85, "AI": 90, "Python": 92}
}
```

2. Key-Value Stores

Simplest type of NoSQL.

Data stored as key-value pairs (like a dictionary or hash map).

Extremely fast for lookups.

Examples: Redis, DynamoDB, Riak
Use Case: Caching, session management, real-time recommendations.

Example:
"user101" : "{name:'Amit', age:22, city:'Pune'}"

3. Column-Family Stores

Data stored in columns instead of rows.

Designed for large analytical workloads and high read/write throughput.

Examples: Apache Cassandra, HBase, ScyllaDB
Use Case: Time-series data, IoT data storage, analytics systems.

Structure Example:
Row Key: Student101 Columns: {Name: Amit, Course: DBMS, Marks: 85}

4. Graph Databases

Store data as nodes (entities) and edges (relationships).

Ideal for representing complex relationships and networked data.

```
Examples: Neo4j, Amazon Neptune, ArangoDB
Use Case: Social networks, fraud detection, recommendation engines.

Example:
(Amit) - [FRIENDS_WITH] -> (Rohit)
```

Use Cases of NoSQL Databases

Domain	Example Use Case	NoSQL Type
Social Media	User profiles, connections	Graph DB
E-commerce	Product catalog, reviews	Document DB
Banking	Fraud detection, transaction graph	Graph DB
IoT Systems	Sensor data storage	Column-family DB
Gaming	Real-time leaderboard, session data	Key-value DB
Big Data Analytics	High-speed data ingestion	Column-family DB

Key Benefits of NoSQL Databases

- **Scalability:** Easily scaled horizontally across servers.
- **Flexibility:** Schema-less design supports dynamic data.
- **Performance:** Faster reads/writes for large datasets.
- **High Availability:** Distributed architecture ensures uptime.
- **Supports Modern Data:** Handles JSON, images, videos, logs, etc.

Difference between RDBMS and NoSQL

Feature	RDBMS	NoSQL
Data Model	Structured (tables)	Unstructured/Semi-structured
Schema	Fixed	Dynamic
Scalability	Vertical	Horizontal
Query Language	SQL	Non-SQL or API-based
Relationships	Supported (joins)	Limited or via graph
Examples	MySQL, Oracle	MongoDB, Cassandra, Redis, Neo4j

Key Points on NoSQL

1. Do NoSQL Databases Have Primary Keys?
Yes, but not in the same way as RDBMS.

Every NoSQL database has a **way to uniquely identify each record** — though it might not be called a "primary key" formally.

NoSQL Type	Equivalent of Primary Key	Example
Key-Value Store	The **key** itself is the primary key.	"user101" → {name:'Amit'}
Document Store	Each document has a unique **_id** field.	{ "_id": 1, "name": "Amit" }
Column-Family Store	Each row has a **row key** (acts like primary key).	RowKey: student101
Graph Database	Each **node** has a unique identifier (node ID).	(user:101)

2. Do NoSQL Databases Have Foreign Keys?

No, most NoSQL databases do NOT use foreign keys.

- There are **no foreign key constraints** between collections or tables.
- Relationships are usually **embedded** or **referenced manually** inside documents.
- The database **does not enforce** referential integrity like an RDBMS.

Example in MongoDB (Document DB):
Instead of two tables with a foreign key relation, we might embed data:

```
{
  "_id": 101,
  "name": "Amit",
  "courses": [
    {"course_id": "DBMS", "marks": 85},
    {"course_id": "AI", "marks": 90}
  ]
}
```

3. Do NoSQL Databases Support Joins?
Generally, NoSQL databases do not support joins like SQL does.
Why?

- Joins slow down performance in large distributed systems.
- NoSQL prefers **denormalized data**, meaning related information is often **stored together** in one document or collection.

However, there are **exceptions**:

- **MongoDB** supports a limited $lookup operator that works like a left outer join.

- **Cassandra** and **Redis** generally avoid joins altogether.
- **Graph Databases (like Neo4j)** do not need joins — relationships are built-in and directly traversed.

```
db.orders.aggregate([
  {
    $lookup: {
      from: "customers",
      localField: "customer_id",
      foreignField: "_id",
      as: "customer_info"
    }
  }
]);
```

Summary of Features

Feature	RDBMS	NoSQL
Primary Key	Yes	Yes (in all types, but implemented differently)
Foreign Key	Yes	Rarely used / Not enforced
Joins	Supported	Not supported or limited
Data Relation Handling	Through normalization and joins	Through embedding or linking

```
Q1.Show a small practical example in MongoDB comparing how the same data
(Students & Courses) is stored with joins in SQL vs embedded in NoSQL ?
```

1. **RDBMS (MySQL)** – using **primary key, foreign key, and joins**
2. **NoSQL (MongoDB)** – using **embedded documents (no joins)**

Example Scenario
We have two entities:

- **Students** – each student has an ID, name, and department.
- **Courses** – each student can take multiple courses with marks.

1. **In RDBMS (MySQL)**

➤ **Tables and Relationships**
We use **two tables** with a **foreign key**.

```sql
-- Students Table
CREATE TABLE Students (
    student_id INT PRIMARY KEY,
    name VARCHAR(50),
    department VARCHAR(50)
);

-- Courses Table
CREATE TABLE Courses (
    course_id INT AUTO_INCREMENT PRIMARY KEY,
    student_id INT,
    course_name VARCHAR(50),
    marks INT,
    FOREIGN KEY (student_id) REFERENCES Students (student_id)
);

INSERT INTO Students VALUES ( 101, 'Amit Kumar', 'Computer Science');

INSERT INTO
    Students
VALUES (
        102,
        'Rohit Sharma',
        'Information Tech'
    );

INSERT INTO
    Courses (
        student_id,
        course_name,
        marks
    )
VALUES (101, 'DBMS', 85),
    (101, 'AI', 90),
    (102, 'Networking', 88),
    (102, 'Python', 92);
```

2.In NoSQL (MongoDB)

In MongoDB, we typically embed the courses inside each student document — no foreign key, no join.

Collection: students

```
db.students.insertMany([
  {
    _id: 101,
    name: "Amit Kumar",
    department: "Computer Science",
    courses: [
      { course_name: "DBMS", marks: 85 },
      { course_name: "AI", marks: 90 }
    ]
  },
  {
    _id: 102,
    name: "Rohit Sharma",
    department: "Information Tech",
    courses: [
      { course_name: "Networking", marks: 88 },
      { course_name: "Python", marks: 92 }
    ]
  }
]);
```

➤ Query to fetch all students with their courses:

```
db.students.find({}, { name: 1, courses: 1, _id: 0 }).pretty();
```

Result (directly from single collection):


```
{
  "name": "Amit Kumar",
  "courses": [
    {"course_name": "DBMS", "marks": 85},
    {"course_name": "AI", "marks": 90}
  ]
}
{
  "name": "Rohit Sharma",
  "courses": [
    {"course_name": "Networking", "marks": 88},
    {"course_name": "Python", "marks": 92}
  ]
}
```

```
No JOIN needed — all related data is inside one document.
```

Comparison Summary

Feature	RDBMS (MySQL)	NoSQL (MongoDB)
Data Model	Tables (rows & columns)	Documents (JSON-like)
Relationship	Separate tables linked by foreign key	Embedded sub-documents
Joins	Required for related data	Not needed
Schema	Fixed (predefined columns)	Flexible (dynamic)
Performance (large scale)	Slower due to joins	Faster (data retrieved in one read)

How data is stored in JSON in NoSQL
In **NoSQL databases** (especially **Document-oriented databases** like MongoDB, CouchDB, and Firebase), data is stored as **documents** in a **JSON-like format**.

What is JSON?
JSON (JavaScript Object Notation) is a **text-based format** used to represent structured data. It uses **key–value pairs** and **nested objects/arrays**, for example:

In NoSQL, each document like this is **stored in a collection** (similar to a table in RDBMS).
So in MongoDB:

- The **database** contains **collections**.
- Each **collection** contains **documents**.
- Each **document** is stored internally in a **binary form of JSON** (called **BSON**).

Do we type JSON data manually?
Sometimes yes, but mostly no.
Here's what that means:
➤ **Option 1: Manually (for small data or examples)**
You can directly insert JSON manually in MongoDB (for testing, labs, or examples):

```
db.students.insertOne({
  student_id: 101,
  name: "Amit Kumar",
  department: "Computer Science",
  courses: [
    { course_name: "DBMS", marks: 85 },
    { course_name: "AI", marks: 90 }
```

```
  ]
});
```

MongoDB automatically stores this JSON document internally as BSON (binary JSON).

➤ Option 2: Automatically (in real applications)

In real-world systems, data is not typed manually.
It is automatically generated or sent to the database by an application (like a website, Python program, or API).

Example using Python (PyMongo):

```
import pymongo

# connect to MongoDB
client = pymongo.MongoClient("mongodb://localhost:27017/")
db = client["college"]
students = db["students"]

# insert data from your application
data = {
  "student_id": 102,
  "name": "Rohit Sharma",
  "department": "Information Tech",
  "courses": [
    {"course_name": "Networking", "marks": 88},
    {"course_name": "Python", "marks": 92}
  ]
}
students.insert_one(data)
```

Here, the program sends a Python dictionary, which is automatically converted to JSON format when stored in MongoDB.

How JSON data is stored internally
Steps:
1 You insert JSON (or similar data) called JSON text
2 Database stores it internally BSON (Binary JSON) faster, supports extra data types
3 When you retrieve it Returned back as JSON

Why JSON format is used in NoSQL

Advantage	Explanation
Flexible Schema	You can add or remove fields easily without altering the structure.
Nested Data	You can store arrays and objects (like courses inside students).
Human-Readable	Easy to understand and debug.
Compatible	Works well with APIs, web apps, and programming languages.

Give me a Real-World Example: Data Flow in NoSQL

Example: Ride-Sharing App (like Uber or Ola)

Every second, thousands of events happen:

- New rides requested
- Driver location updates (GPS)
- Payment logs
- Ratings and feedback

All this data is:

- Generated rapidly by mobile devices.
- Sent to a backend server (API).
- Stored in MongoDB almost instantly as JSON-like documents.

Advantage

- Flexible Schema You can add or remove fields easily without altering the structure.
- Nested Data You can store arrays and objects (like courses inside students).
- Human-Readable, Easy to understand and debug.
- Compatible, Works well with APIs, web apps, and programming languages.

```
{
  "ride_id": "R456",
  "user_id": "U101",
  "driver_id": "D55",
```

```
  "pickup": "Pune Station",
  "drop": "Airport",
  "start_time": "2025-11-07T10:15:00",
  "status": "completed",
  "fare": 320.50,
  "rating": 5
}
```

Where MongoDB Excels (Use Cases)

Application Type	Why MongoDB Fits
E-commerce websites	Product details vary (different fields, reviews, images). JSON allows flexible storage.
Social media apps	Posts, comments, likes — all dynamic and fast-changing data.
IoT systems	Billions of sensor readings per second — stored rapidly.
Gaming apps	Real-time player stats, leaderboards, and game events.
Analytics dashboards	Continuous log and event data ingestion.

Example Architecture

[Mobile / Web App]
↓
(Sends JSON data)
↓
[Backend API Server]
↓
(Inserts into MongoDB)
↓
[MongoDB Database]
├— Collection: users
├— Collection: rides
└— Collection: payments

MongoDB is ideal when:

- Data comes rapidly or in huge volumes.
- The data format changes often (dynamic fields).
- You need fast writes and reads.
- You want to scale easily across multiple servers.