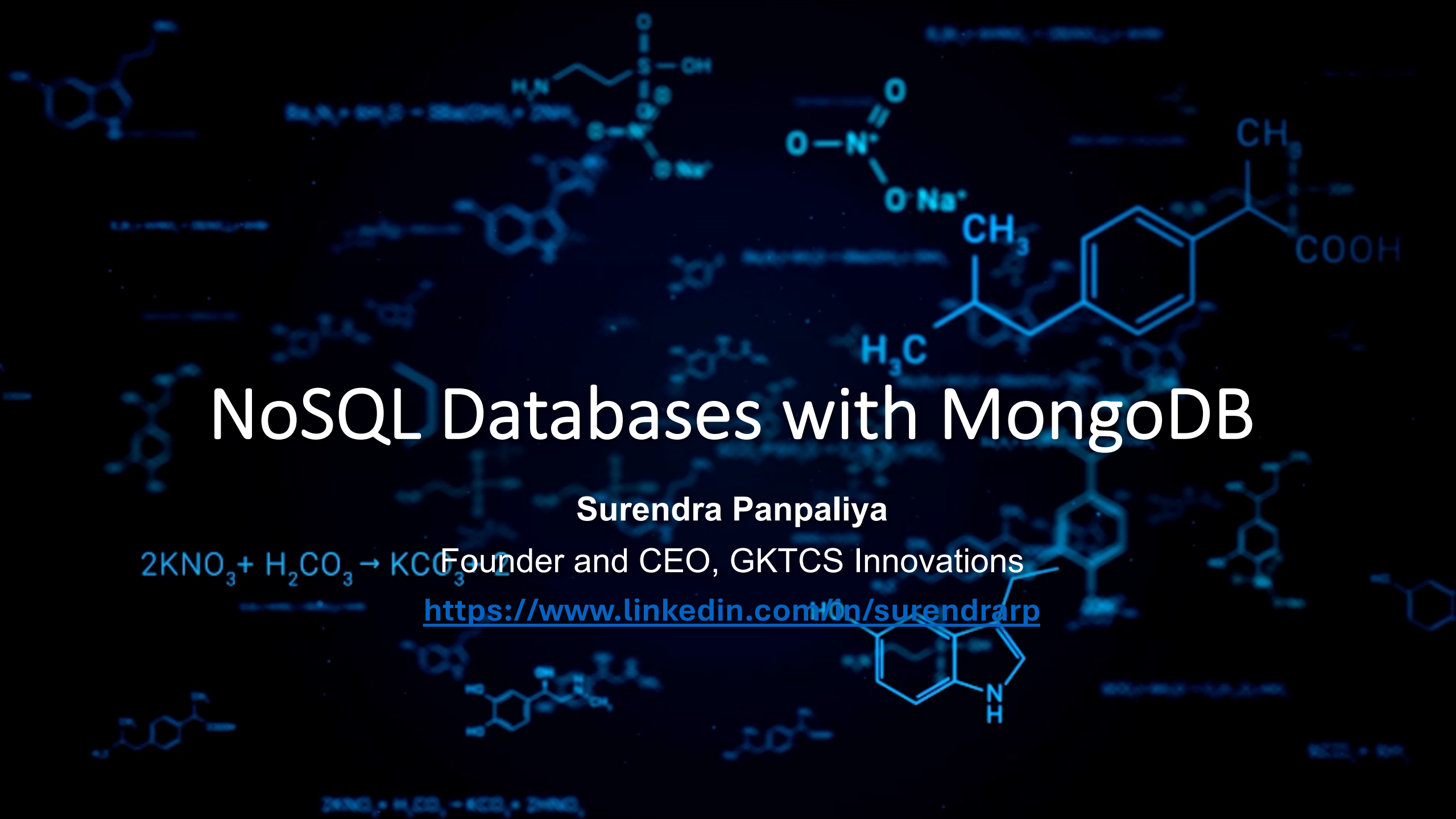


NoSQL Databases with MongoDB

Surendra Panpaliya

Founder and CEO, GKTCS Innovations

https://www.linkedin.com/in/surendrarp

Surendra Panpaliya, DTM

Founder & CEO, GKTCS Innovations – building future-ready enterprises.

Empowered 35,000+ IT professionals through training, mentoring, and consulting.

Partnered with 300+ multinational corporations to accelerate business growth.

🎯 Learning Objectives

- Understand NoSQL fundamentals and
- how they differ from RDBMS.
- Model data efficiently for NoSQL systems.
- Perform CRUD operations, aggregations, and
- indexing in MongoDB.

🎯 Learning Objectives

Apply NoSQL patterns to real-world banking scenarios

e.g., Customer360, Transaction Logging, Fraud Detection

Build and query NoSQL databases

using MongoDB tools.

Prerequisites

- Basic understanding of **Databases & SQL** concepts
- tables, joins, normalization
- Familiarity with **Python / JavaScript** syntax
- for data access (optional but helpful).
- Prior exposure to **JSON** and **REST APIs** recommended.

Lab Setup Requirements

System Requirements:

Laptop Configuration:

8 GB RAM minimum, 20 GB free disk space

Operating System:

Windows 10+, macOS, or Ubuntu

Lab Setup Requirements

Software Installation:

MongoDB Community Edition (latest version)

MongoDB Compass – GUI for query visualization

VS Code or **Jupyter Notebook** for scripting

Python 3.11+ with pymongo library installed

Agenda

Day1

Introduction to NoSQL & MongoDB Fundamentals

Day2

Data Modeling, Aggregation, and Use Cases

Module 1: NoSQL Overview

Evolution of Databases:

RDBMS → NoSQL

Key Characteristics:

Scalability, Flexibility,

Schema-less Design

Module 1: NoSQL Overview

Types of NoSQL Databases:

Key-Value Stores (Redis)

Document Stores (MongoDB)

Column Stores (Cassandra)

Graph Databases (Neo4j)

Module 1: NoSQL Overview

Relational vs. NoSQL

Comparison table

Architecture diagram

Module 2: MongoDB Essentials

MongoDB Architecture:

Database → Collection → Document

BSON & JSON structure

CRUD Operations:

insertOne(), find(),

updateOne(), deleteOne()

Module 2: MongoDB Essentials

Indexing & Query Optimization

Data Import/Export using

mongoimport, mongoexport

Module 3: Hands-On Lab

Lab 1: Create and Query a MongoDB Customer Database

Create a customers collection

Insert 10 customer profiles (JSON documents)

Query customers using filters (find(), $and, $or)

Update address & contact info using update operators

What is NoSQL Database

High Scalability

High Availability

NoSQL Database

Big Data Capability

Fast Performance

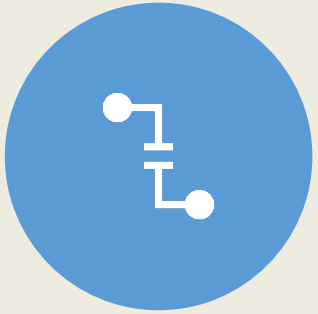
Easy Replication

What is NoSQL?

A non-relational Data Management System

Does not require a fixed schema.

Avoid join, & easy to scale.

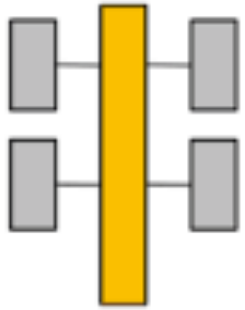


Stands for "Not Only SQL" or "Not SQL."

SQL Database

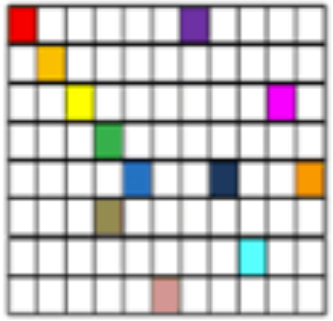
Relational

Analytical (OLAP)



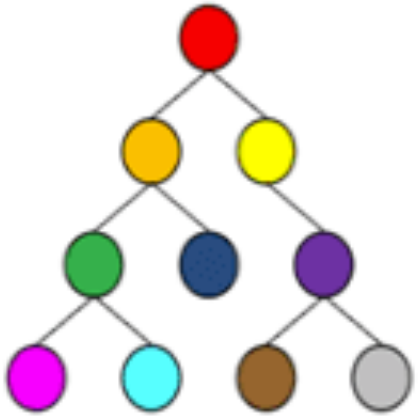
NoSQL Database

Column-Family

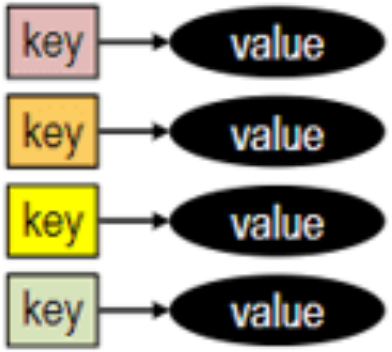


Graph

Document

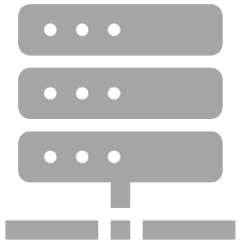


Key-Value



What is NoSQL?

Purpose is for distributed data stores



with humongous (big) data storage needs.

Used for Big data and real-time web apps.

Example

COMPANIES LIKE TWITTER,

FACEBOOK AND GOOGLE

COLLECT TERABYTES OF USER DATA

EVERY SINGLE DAY


AEROSPIKE
cassandra
Clustrix
Couchbase
amazon
DynamoDB
APACHE
HBASE
IBM
Cloudant
MarkLogic
memsql
Speed. Scale. Simplicity.
mongoDB
NUODB
ORACLE
NOSQL
DATABASE
riak
splice
MACHINE
TRANSLATTICE
VOLTDB

Module 1: NoSQL Overview

Evolution of Databases:

RDBMS → NoSQL

Key Characteristics:

Scalability, Flexibility,

Schema-less Design

Evolution of Databases: RDBMS → NoSQL

Surendra Panpaliya
Global IT Trainer | Consultant |
Founder & CEO, GKTCS Innovations

Brief History of NoSQL Databases

1998

Carlo Strozzi use the term NoSQL for his lightweight, open-source relational database

2000

Graph database Neo4j is launched

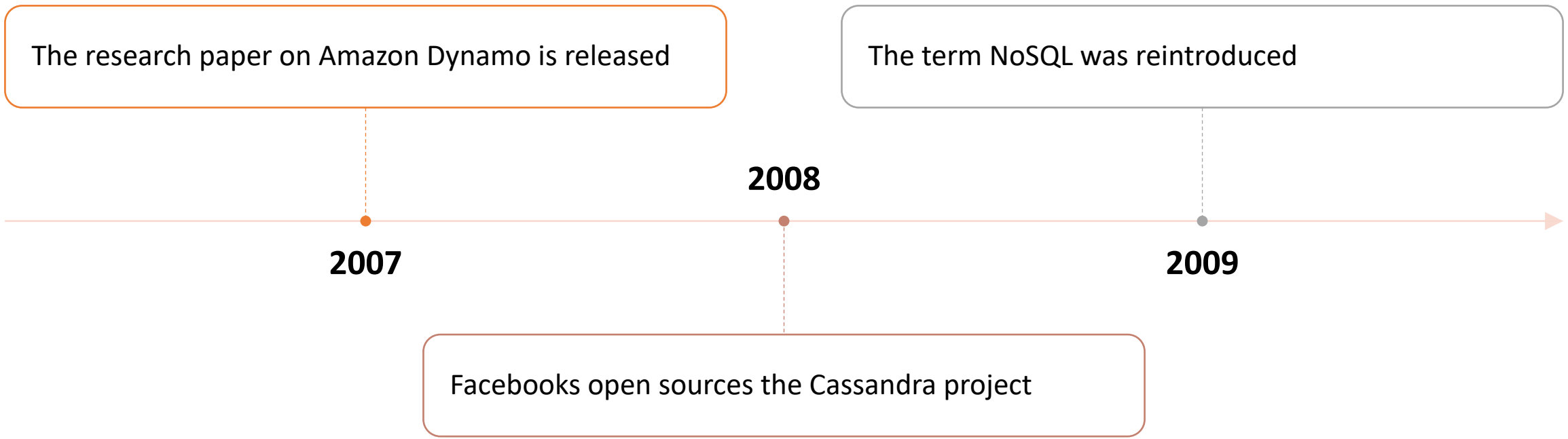
2004

Google BigTable is launched

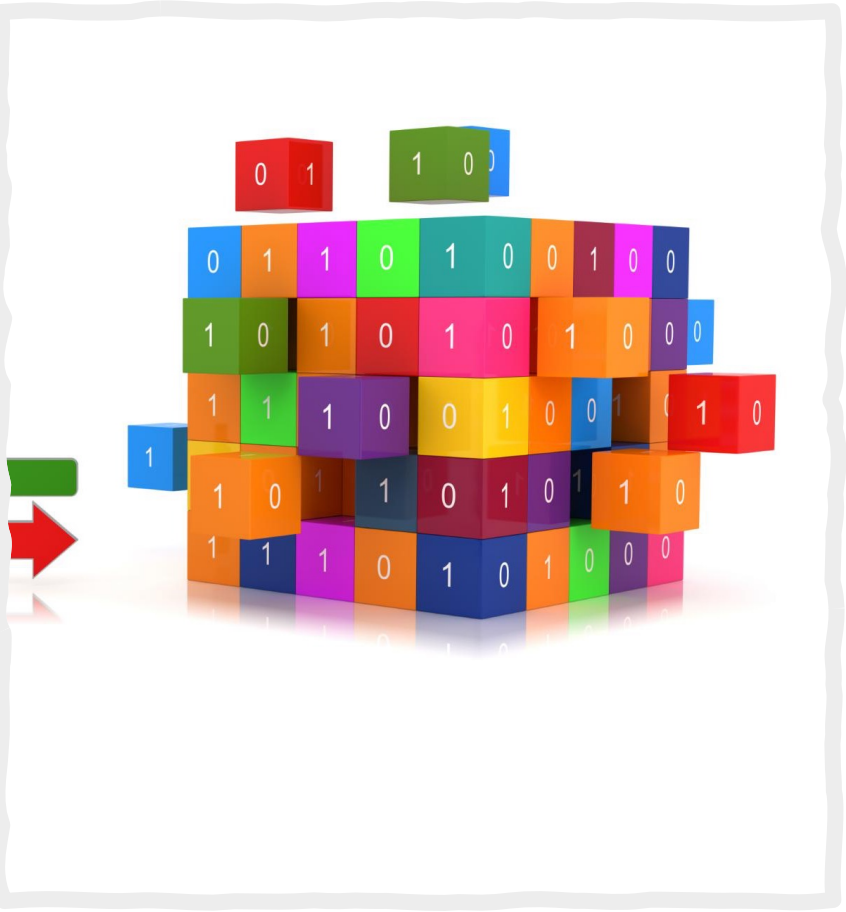
2005

CouchDB is launched

Brief History of NoSQL Databases



From RDBMS to NoSQL



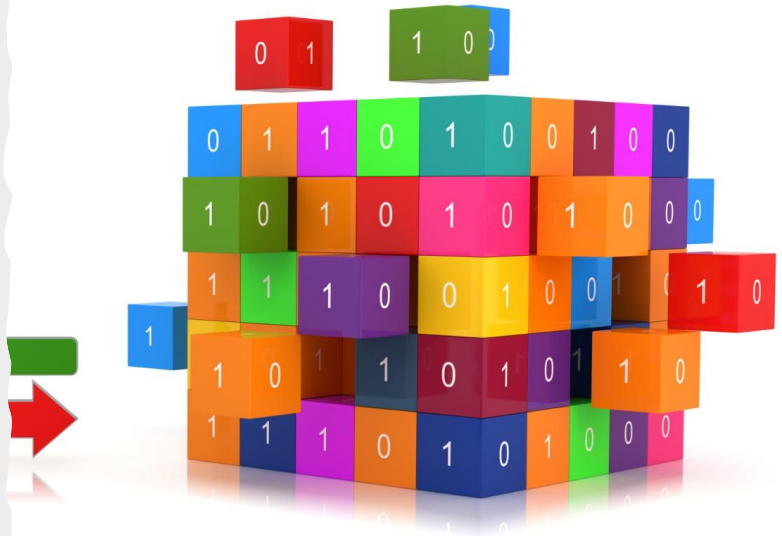
RDBMS Characteristics

RDBMS provides structured data storage with strong ACID properties ideal for high data integrity applications.

Challenges of RDBMS

RDBMS struggles with horizontal scaling and handling massive unstructured data from modern digital platforms.

From RDBMS to NoSQL



NoSQL Advantages

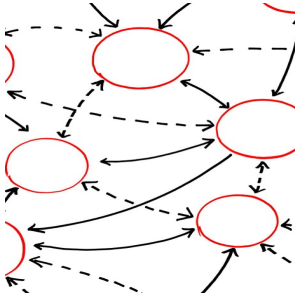
NoSQL databases enable distributed storage, horizontal scaling, and flexible schema for diverse data types.

Modern Applications

NoSQL supports IoT, big data, social media, mobile apps, and real-time analytics in today's ecosystem.

Traditional RDBMS Era

Features and Use Cases of RDBMS



Core Features of RDBMS

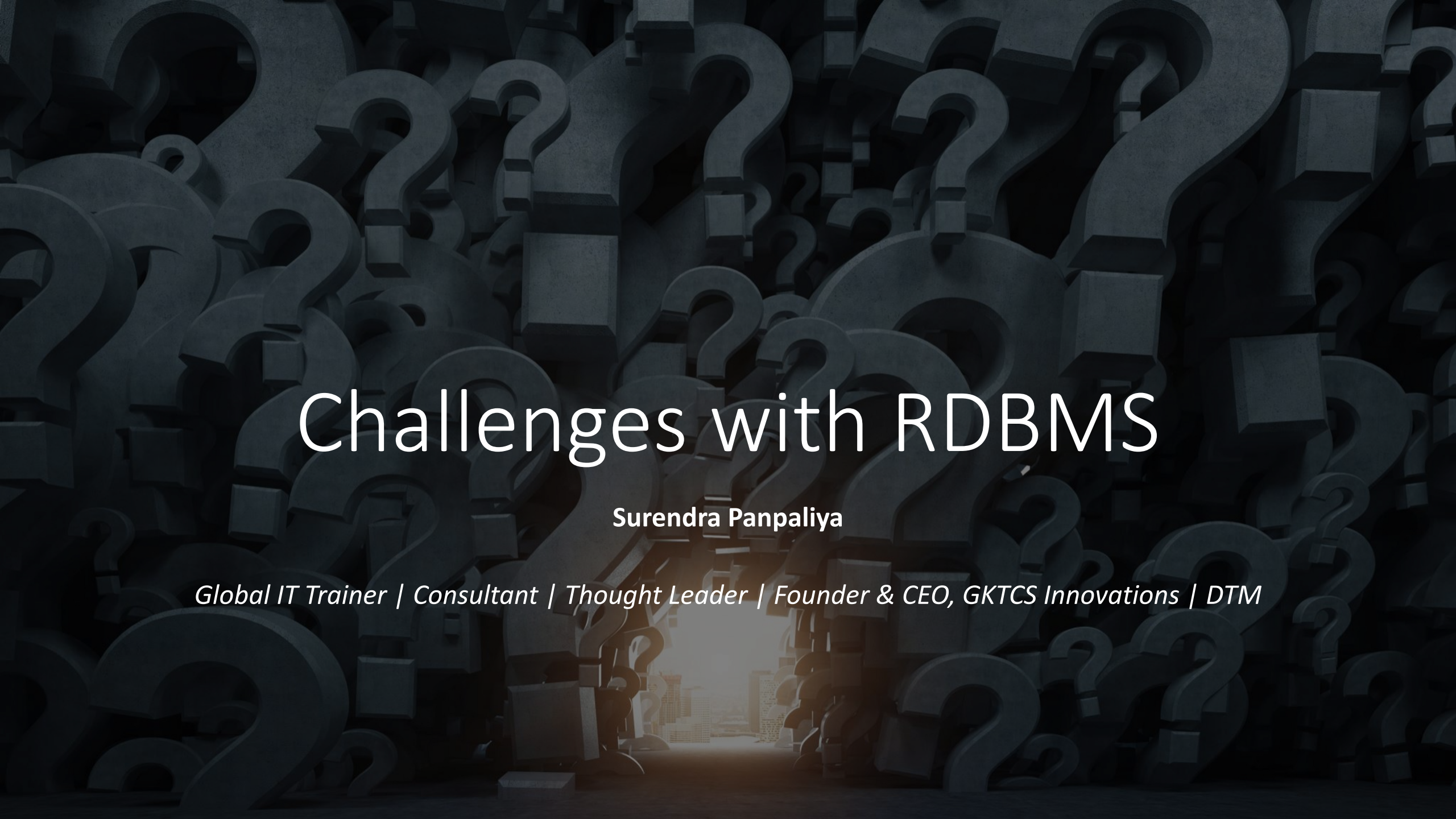
RDBMS use structured tables with rows and columns and support ACID transactions ensuring reliability and consistency.

Common Use Cases

Used widely in banking, ERP, CRM, billing, accounting, and inventory systems for managing complex and critical data.

Limitations

Designed for centralized systems, RDBMS lack inherent distributed computing and horizontal scaling capabilities.

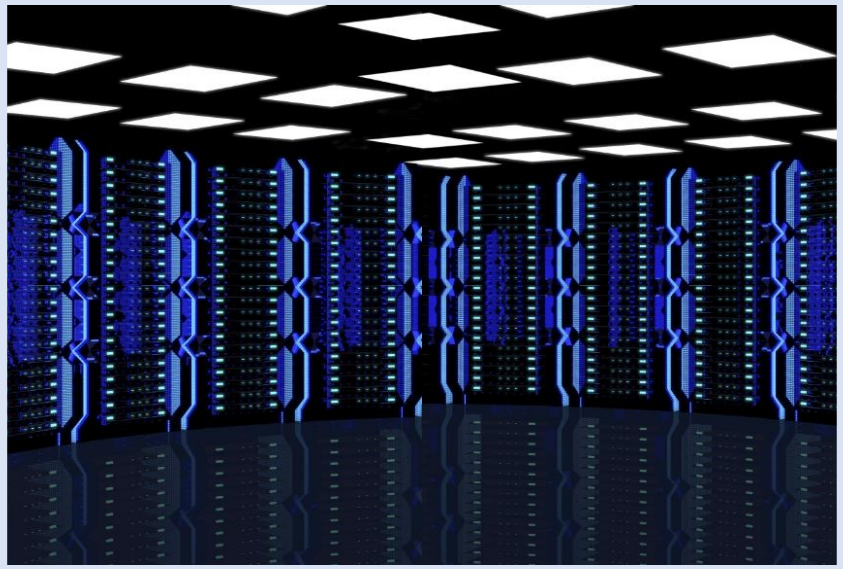


Challenges with RDBMS

Surendra Panpaliya

Global IT Trainer | Consultant | Thought Leader | Founder & CEO, GKTCS Innovations | DTM

Limitations in the Face of Digital Transformation



Challenges with RDBMS Scaling

Traditional RDBMS struggled with horizontal scaling, limiting support for millions of concurrent users.

Handling Unstructured Data

RDBMS rigid schemas made it difficult to manage unstructured and semi-structured data from Web 2.0 sources.

Limitations in the Face of Digital Transformation

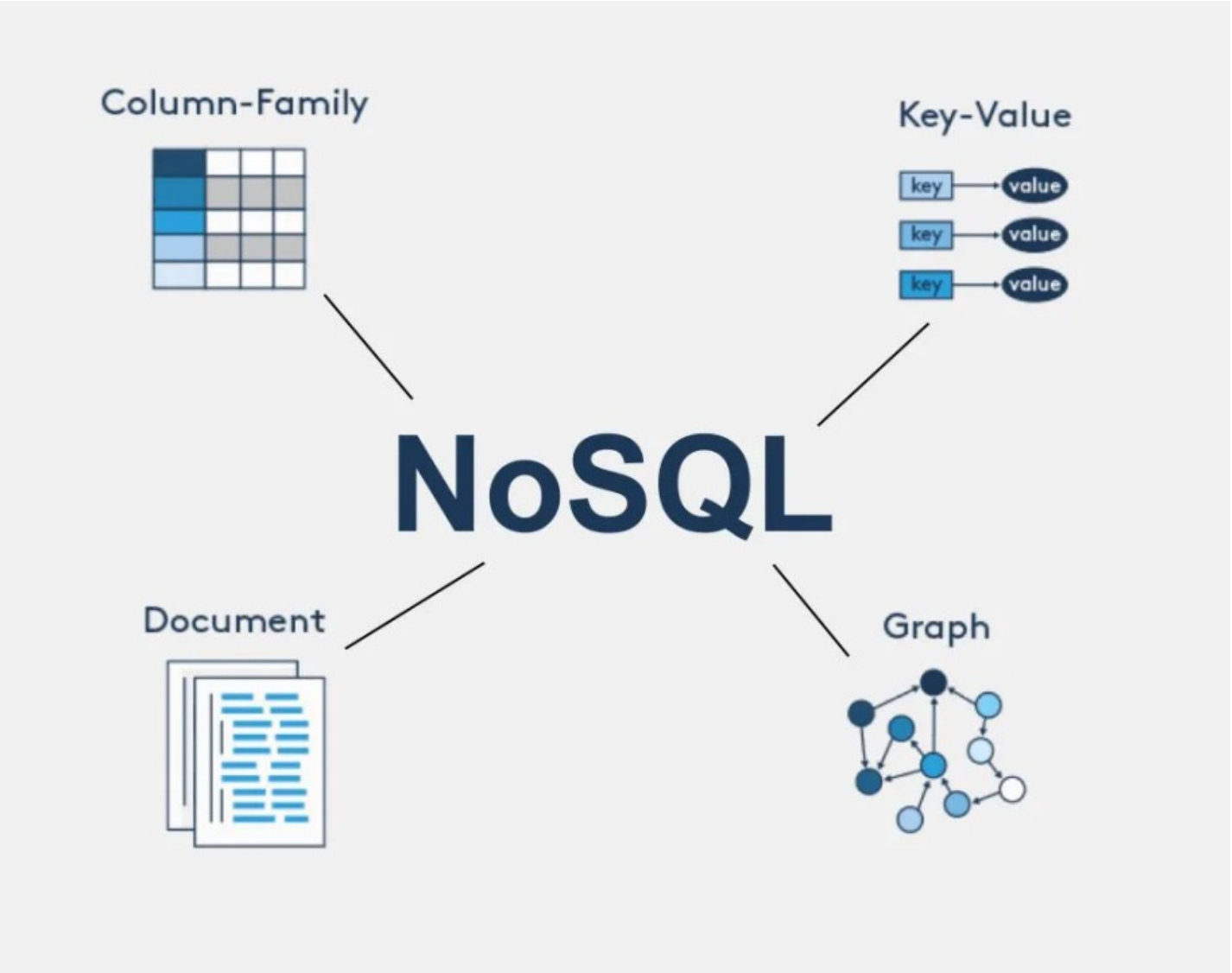
Real-Time Analytics Limitations

RDBMS lacked the performance and flexibility required for real-time analytics in dynamic environments.

Distributed Architecture Constraints

RDBMS were not designed for efficient operation in distributed, modern application architectures.

Birth of NoSQL



Emergence and Advantages of NoSQL

Speed and Low Latency

NoSQL databases prioritize speed and low latency, essential for real-time data processing and applications.

Distributed Storage and Fault Tolerance

NoSQL supports distributed data storage for high availability and fault tolerance across multiple servers.

Emergence and Advantages of NoSQL

Flexible Data Models

These databases handle semi-structured and unstructured data such as JSON, graphs, key-value pairs, and wide-columns.

Scalable and Adaptable Applications

NoSQL enables horizontal scaling and flexible schema, ideal for IoT, big data, social media, and mobile apps.

Key Characteristics of NoSQL Databases

Scalability (Horizontal Scaling)

Horizontal Scaling Concept

NoSQL databases expand across multiple servers to handle large data volumes with high throughput and low latency.

Performance and Capacity

Adding nodes increases data processing capacity without performance loss, supporting massive concurrent data access.

Scalability (Horizontal Scaling)

Distributed Architecture Support

Horizontal scaling enables distributed systems to maintain availability and performance as data volumes grow exponentially.

Real-World Applications

Ideal for platforms like social media timelines and video comments where large data volumes are accessed concurrently.

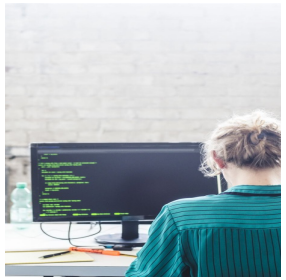
Flexibility

Schema Flexibility

NoSQL databases allow fields to vary between records, avoiding rigid predefined schemas.

Support for Multiple Data Formats

Supports JSON documents, key-value pairs, and graph data for diverse application needs.



Benefits for Agile Development

Enables rapid prototyping and adaptation to changing requirements without extensive schema changes.

Schema-less / Dynamic Schema

Schema-less Data Model

NoSQL databases use a schema-less model, allowing flexible and dynamic data structures without predefined tables.

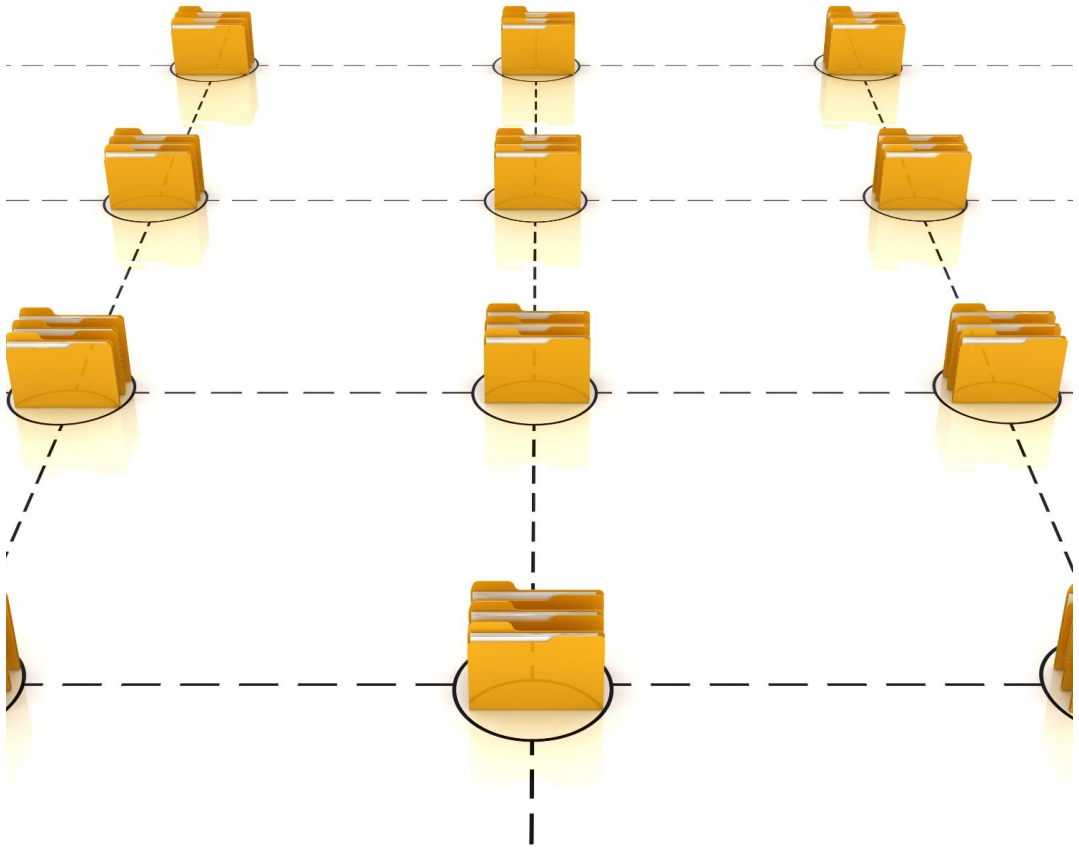
Supports Agile Development

Dynamic schema supports rapid changes to data models, enabling agile methodologies without downtime or migrations.

Facilitates Rapid Iteration

Developers can quickly introduce new fields or data types, speeding up development and response to user feedback.

High Performance (Low Latency)



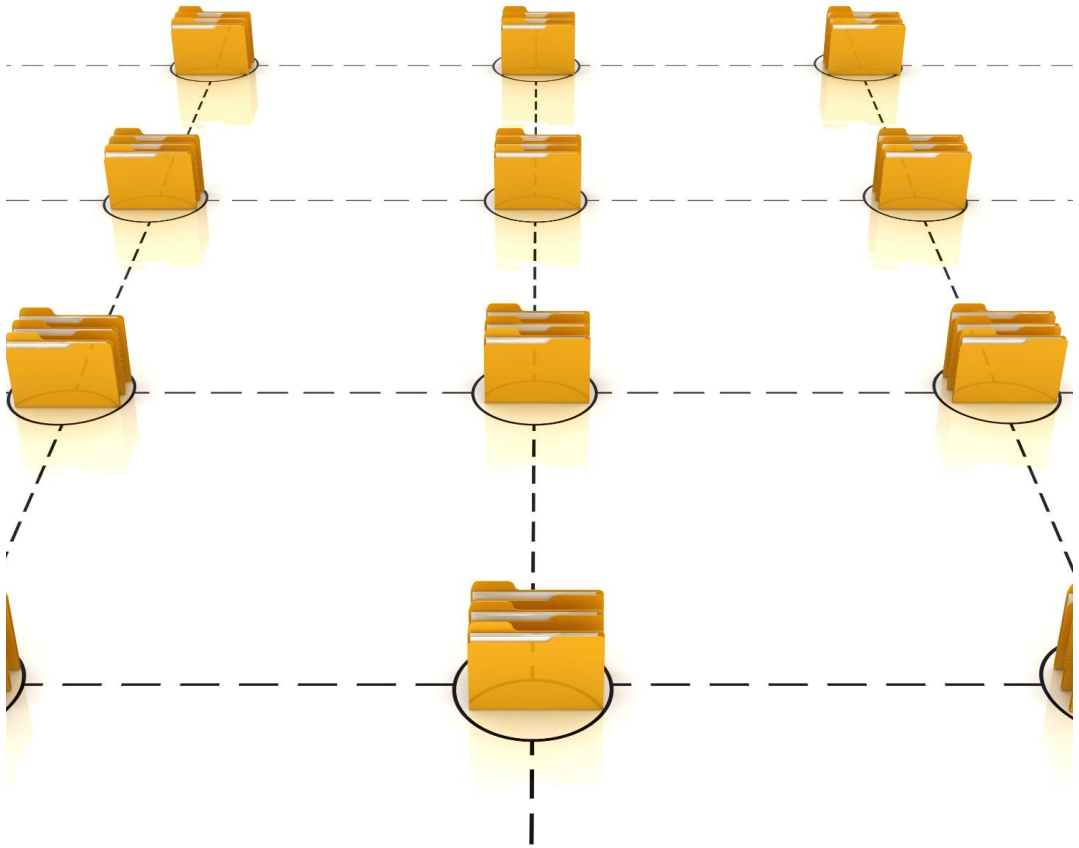
In-Memory Caching

In-memory caching reduces latency by storing frequently accessed data for quick retrieval.

Distributed Write Capability

Distributed writes allow data to be recorded across multiple nodes simultaneously, enhancing performance.

High Performance (Low Latency)



Suitability for Real-Time Applications

NoSQL databases' low latency and high throughput support critical real-time applications like gaming and finance.

High Availability & Fault Tolerance

- **Distributed Systems Architecture**
- **Fault Tolerance and Failover**
- **Continuous Uptime for Critical Apps**

Distributed Systems Architecture

- NoSQL databases are distributed systems
- with replication to duplicate data
- across several nodes.

Fault Tolerance and Failover

- On node failure,
- other nodes take over automatically
- to maintain system operation
- without downtime.

Continuous Uptime for Critical Apps

- High availability supports applications
- like e-commerce and cloud services
- requiring uninterrupted access.

Designed for Big Data

Handling Massive Data Volumes

NoSQL databases efficiently manage workloads from terabytes to petabytes, supporting vast big data operations.

Real-Time Analytics Support

They support real-time event stream processing and analytics for immediate data insights.

Designed for Big Data

Flexible Data Management

Capable of storing and processing semi-structured and unstructured data for diverse applications.

Applications in Modern Ecosystems

Widely used across IoT, log analysis, and machine learning pipelines for scalable performance.

RDBMS vs NoSQL Comparison

Feature / Aspect	RDBMS (SQL Databases)	NoSQL Databases
Data Model	Structured (Tables, Rows, Columns)	Semi-structured or unstructured (JSON, Key-value, Graph, Column-family)
Schema	Fixed schema; strict	Schema-less / dynamic schema
Query Language	SQL	No standard language; uses APIs or specific query formats (e.g., MongoDB Query Language)

RDBMS vs NoSQL Comparison

Feature / Aspect	RDBMS (SQL Databases)	NoSQL Databases
Scalability	Vertical scaling (bigger server)	Horizontal scaling (add more servers)
Transactions	Strong ACID	Often BASE (eventual consistency), but some support ACID (MongoDB now supports ACID)
Joins	Supports complex joins	No joins (except graph databases); denormalization is common

RDBMS vs NoSQL Comparison

Feature / Aspect	RDBMS (SQL Databases)	NoSQL Databases
Performance	Slower for massive scale	High performance for large distributed workloads
Use Cases	Banking, ERP, accounting, billing, inventory	Social media, IoT, mobile apps, real-time analytics, product catalogs
Examples	Oracle, MySQL, PostgreSQL, SQL Server	MongoDB, Cassandra, Redis, DynamoDB, Neo4j

RDBMS vs NoSQL Comparison

Feature / Aspect	RDBMS (SQL Databases)	NoSQL Databases
Flexibility	Low (strict structure)	High (schema evolves with business needs)
Handling Big Data	Not optimized for big data	Designed for massive and rapidly growing datasets
Relationships	Strong relationships with referential integrity	Weak relationships; relationships handled at application level (except graph DBs)

Summary

RDBMS

BEST FOR STRUCTURED DATA

STRONG CONSISTENCY

IDEAL FOR FINANCE, BANKING, ERP

Summary

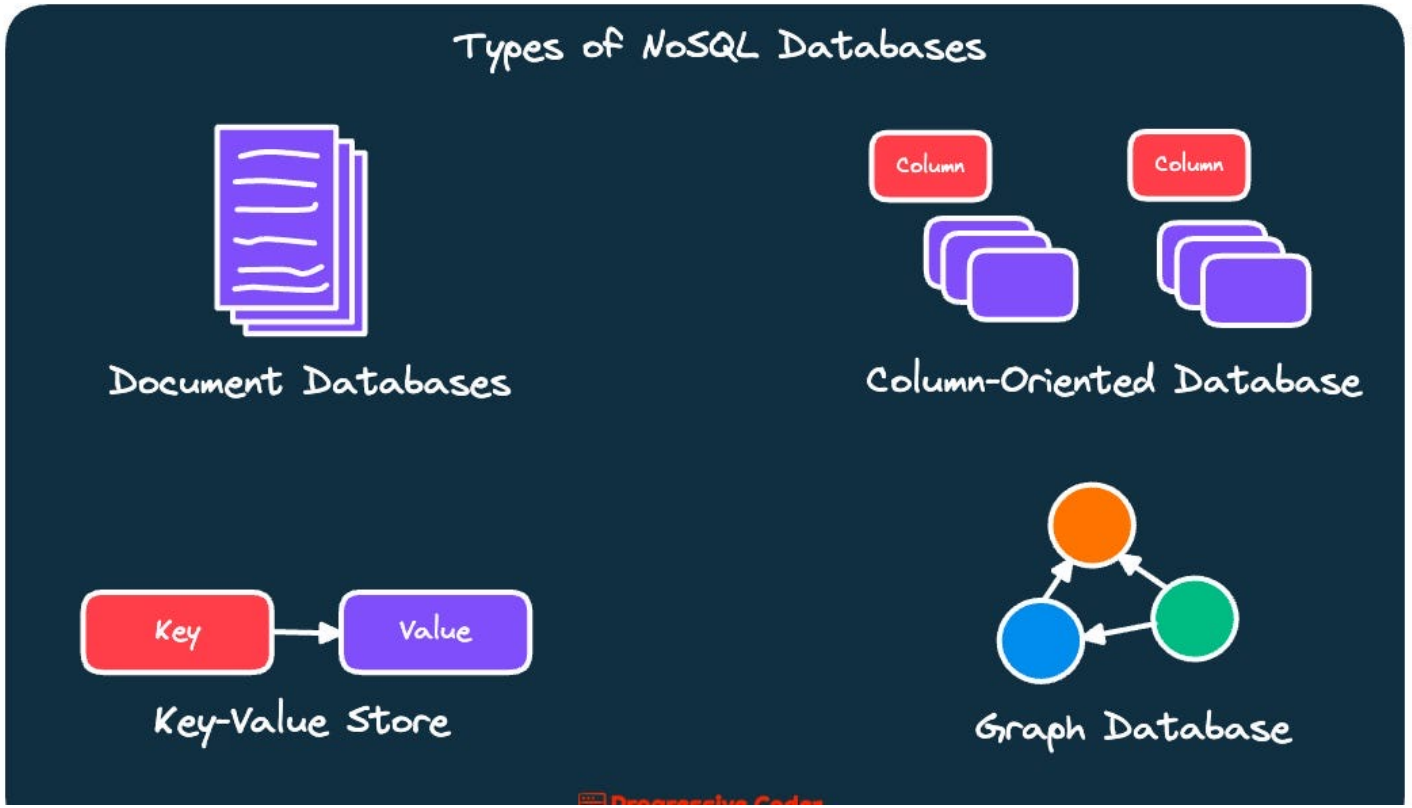
NoSQL

Best for rapidly changing, large-scale apps

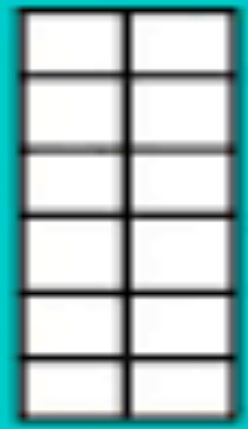
Handles massive data & real-time workloads

Perfect for modern applications (IoT, social media, ecommerce)

NoSQL Types and Examples



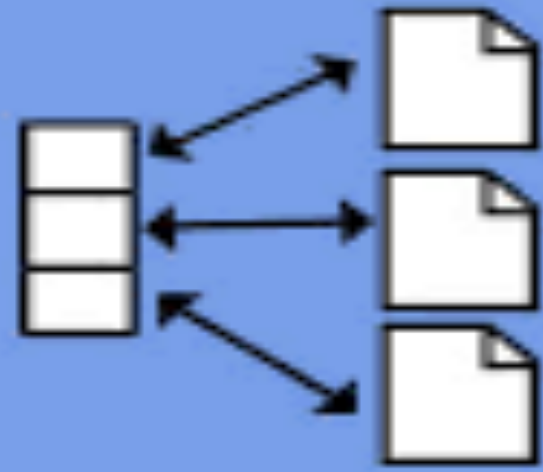
Key Value



Graph DB

Column Family

Document



NoSQL Database Types

Key Value

- In a key-value NoSQL Database, all of the data within consists of an indexed key and a value
- *Examples include :*
  - *DynamoDB*
  - *Cassandra*

Column Based

- In Column Based NoSQL Database, DB is designed for storing data tables as sections of columns of data, rather than as rows of data
- *Examples include :*
- *HBase*
- *SAP HANA*

Document Database

- This NoSQL Database expands the key-value stores where "documents" contain more complex in that they contain data and each document is assigned a unique key, which is used to retrieve the document
- *Examples include :*
- *MongoDB*
- *CouchDB*

Graph Database

- This No SQL database IS designed for data whose relations are well represented as a graph and has elements which are interconnected, with an undetermined number of relations between them
- *Examples include :*
- *Polyglot*
- *Neo4J*

Classification and Use Cases of NoSQL Databases

Document Databases

Store data as JSON-like documents, ideal for flexible schemas and content management systems.

Wide-Column Stores

Optimized for large distributed data volumes, commonly used in analytics and time-series data.

Classification and Use Cases of NoSQL Databases

Key-Value Stores

Provide rapid data access, suitable for caching and session management applications.

Graph Databases

Designed to map relationships between data, used in social networks and fraud detection.

Module 1: NoSQL Overview

Types of NoSQL Databases:

Key-Value Stores (Redis)

Document Stores (MongoDB)

Column Stores (Cassandra)

Graph Databases (Neo4j)

Key–Value Databases

STORES DATA AS SIMPLE

KEY → VALUE PAIRS.

SIMILAR TO A DICTIONARY OR

A HASH MAP.

Examples

Redis

Amazon DynamoDB

Riak

Best for

CACHING

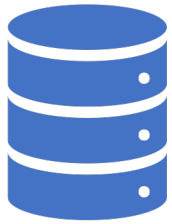
SESSION STORAGE

HIGH-SPEED LOOKUPS

SHOPPING CARTS (E-COMMERCE)

Pros

Extremely fast



Simple data model

Distributed by design

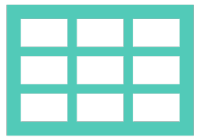
Cons

No querying on value fields

Not suitable for complex queries

Document Databases

Data is stored as **JSON-like documents**



(BSON in MongoDB).

Each document can have different fields

schema flexibility.

Examples

MongoDB

Couchbase

Firestore

Best for

User profiles

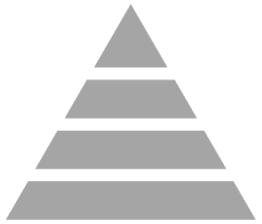
Product catalogs

Customer360 systems

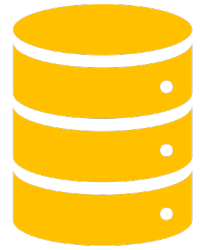
Real-time analytics

Pros

Schema-less



Nested structures



Rich queries and indexing

Cons

No multi-document JOINs

Requires denormalization

Columnar (Wide Column) Databases

DATA STORED IN **COLUMN FAMILIES**,

OPTIMIZED FOR LARGE-SCALE

DISTRIBUTED WORKLOADS.

Examples

Apache Cassandra

HBase

ScyllaDB

Best for

Time-series data

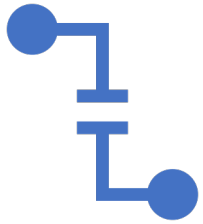
IoT sensor data

Banking transaction logs

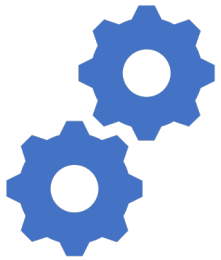
Massive write throughput

Pros

High write scalability



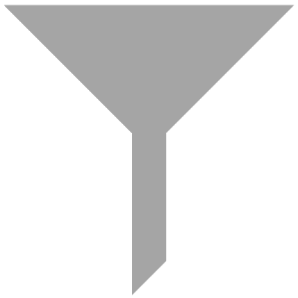
Distributed and fault-tolerant



Linear horizontal scaling

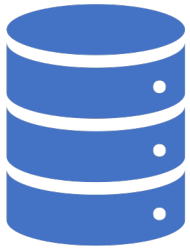
Cons

Complex data modeling

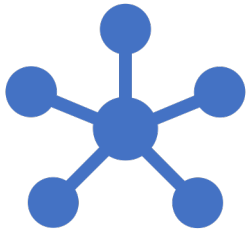


No ACID multi-row transactions

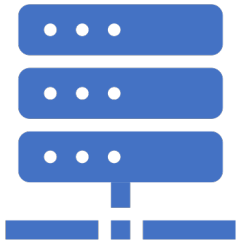
Graph Databases



Data stored as



nodes and relationships.



Best for highly connected data.

Examples

Neo4j

JanusGraph

Amazon Neptune

Best for

Fraud detection

Recommendation engines

Social networks

Network topologies

Pros

Relationship-first

Fast traversal with Cypher queries

Cons

Not suitable for heavy writes

Scaling graph DBs is hard

NoSQL Types Comparison Table

Feature	Key–Value	Document	Columnar (Wide Column)	Graph
Data Model	Key → Value	JSON / BSON document	Column families	Nodes & Edges
Schema	None	Flexible	Semi-structured	Schema-free
Best For	Caching, sessions	CMS, Customer360	Time-series, IoT	Social graphs, fraud

NoSQL Types Comparison Table

Feature	Key–Value	Document	Columnar (Wide Column)	Graph
Querying	By key only	Rich queries	Limited SQL-like	Graph traversal
Scaling	Horizontal	Horizontal	Massive horizontal	Harder to scale
Examples	Redis, DynamoDB	MongoDB, Couchbase	Cassandra, HBase	Neo4j, Neptune

Module 1: NoSQL Overview

Relational vs. NoSQL

Comparison table

Architecture diagram

Relational vs NoSQL — Comparison Table

Dimension	Relational (RDBMS)	NoSQL (Key-Value / Document / Column / Graph)	Banking Example
Data model	Tables, rows, fixed schema	Keys, JSON docs, wide rows, nodes/edges	Flexible Customer360 profile as JSON
Schema	Strict (DDL)	Schema-less / schema-light	Add riskScore to customer without migration
Queries	SQL, JOINs, strong ACID	API/Cypher/CQL/MQL; denormalize; limited cross-entity joins	Aggregate transactions by account (Cassandra)

Relational vs NoSQL — Comparison Table

Dimension	Relational (RDBMS)	NoSQL (Key-Value / Document / Column / Graph)	Banking Example
Scaling	Vertical; read replicas	Horizontal (sharding/partitioning)	Scale transaction ingestion across shards
Transactions	Strong ACID, multi-row	Varies (often single-item ACID); tunable consistency	Posting ledger in RDBMS, event log in NoSQL
Use cases	Core ledger, settlement	Profiles, logs, events, caching, fraud graphs	Fraud ring detection (Neo4j), OTP cache (Redis)

Module 2: MongoDB Essentials

MongoDB Architecture:

Database → Collection → Document

BSON & JSON structure

CRUD Operations:

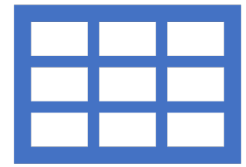
insertOne(), find(),

updateOne(), deleteOne()

What is MongoDB?

An **open-source NoSQL database**

designed to handle **large volumes of**



unstructured or semi-structured data.

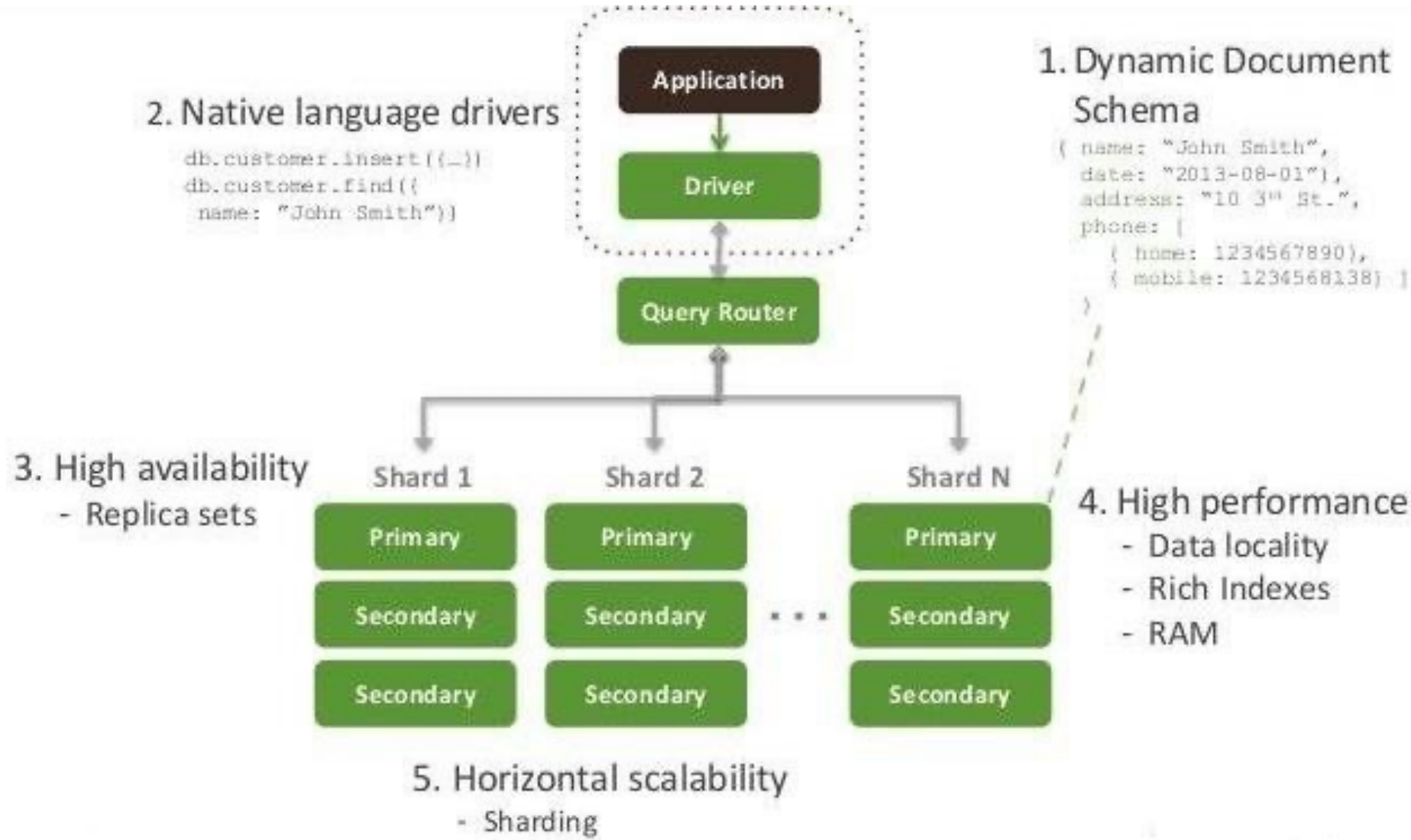
What is MongoDB?

Instead of storing data in tables and rows

like relational databases

MongoDB stores data as

documents in **collections**.


2. Native language drivers
db.customer.insert({…})
db.customer.find({
name: "John Smith"})
Application
Driver
Query Router
1. Dynamic Document Schema
{ name: "John Smith",
date: "2013-08-01",
address: "10 3rd St.",
phone: [
{ home: 1234567890},
{ mobile: 1234568138}]
}
3. High availability
- Replica sets
Shard 1
Shard 2
Shard N
Primary
Secondary
Secondary
Primary
Secondary
Secondary
Primary
Secondary
Secondary
4. High performance
- Data locality
- Rich Indexes
- RAM
5. Horizontal scalability
- Sharding

What is MongoDB?

Good at handling different kinds of data.

Well-suited for handling both structured and

unstructured data, making it

a versatile database solution.

What is MongoDB?

JSON is a simple way to organize data

in a text format.

It uses key-value pairs,

kind of like a dictionary.

What is MongoDB?

if you wanted to store information about a person,

you might use JSON like this,

```
{
"name": ”Surendra",
"age": 35,
"city": “Pune"
}
```

Key Characteristics

Document-Oriented

Schema-less

Scalable

High Performance

Cross-platform

Key Characteristics

DOCUMENT-ORIENTED

DATA IS STORED IN

JSON-LIKE (BSON) DOCUMENTS

FLEXIBLE AND READABLE.

Key Characteristics

Schema-less

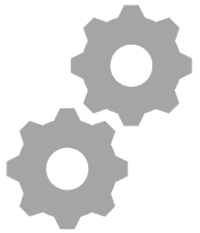
No fixed schema

each document can have

a different structure.

Key Characteristics

Scalable:



Built for horizontal scaling

using **sharding** across servers.

Key Characteristics

HIGH PERFORMANCE:

PROVIDES FAST READ/WRITE

OPERATIONS WITH INDEXES AND

IN-MEMORY CACHING.

Key Characteristics

CROSS-PLATFORM:

RUNS ON WINDOWS, LINUX,

MACOS, AND CLOUD

MONGODB ATLAS

Why MongoDB?

Reason	Description	Example
Flexible Schema	No need to predefine table columns; each document can store different fields	Store customer details with dynamic preferences
High Scalability	Supports horizontal scaling via sharding (distributing data across servers)	Handle millions of banking transactions efficiently

Why MongoDB?

Reason	Description	Example
High Availability	Replica sets ensure automatic failover and data redundancy	Keeps DBS Tech Bank apps online 24×7
JSON/BSON Storage	Natural fit for modern applications using APIs or microservices	Easier to integrate with Node.js / Python apps

Why MongoDB?

Reason	Description	Example
Aggregation Framework	Advanced analytics and data transformation directly inside MongoDB	Calculate total transactions, fraud detection, etc.
Easy Integration with Cloud	Fully managed service on MongoDB Atlas	Quick deployment without server setup
Developer Friendly	Works seamlessly with most programming languages and frameworks	Python (pymongo), JavaScript, Java, Go, etc.

MongoDB vs. Relational Databases

Feature	MongoDB (NoSQL)	Relational DB (SQL)
Data Model	Document-based (JSON/BSON)	Table-based (rows/columns)
Schema	Dynamic / Flexible	Fixed schema
Relationships	Embedded or referenced	Foreign keys

MongoDB vs. Relational Databases

Feature	MongoDB (NoSQL)	Relational DB (SQL)
Scalability	Horizontal (Sharding)	Vertical (Hardware upgrade)
Joins	Application-level / $lookup	SQL joins
Ideal For	Big Data, IoT, Microservices	Traditional transactional systems

MongoDB vs. MySQL

Parameters	MongoDB	MySQL
Data Model	NoSQL database	Relational database
Data Type	Ideal for applications with rapidly changing data structures or unstructured data	Ideal for applications with structured data and complex relationships between entities
Query language	MongoDB query language	Standard Query Language
Data Storage	It uses a schema-less format called JSON-like documents	It uses structured tables with fixed schemas
Scalability	Horizontal scaling	Vertical scaling

MongoDB vs. MySQL

Parameters	MongoDB	MySQL
ACID Compliance	Limited	Full
Licensing	Server Side Public License (SSPL)	GNU General Public License (GPL)
Cost	Open-source with commercial support options	Open-source with commercial support options
Data Integrity	Flexible	Rigid
Indexing	Dynamic and automatic	Manual

Example Use Cases (DBS Tech Bank Context)

Customer360 View:

Combine customer, account, and transaction data in one document.

Transaction Logs:

Store high-frequency logs with timestamps and devices.

Example Use Cases (DBS Tech Bank Context)

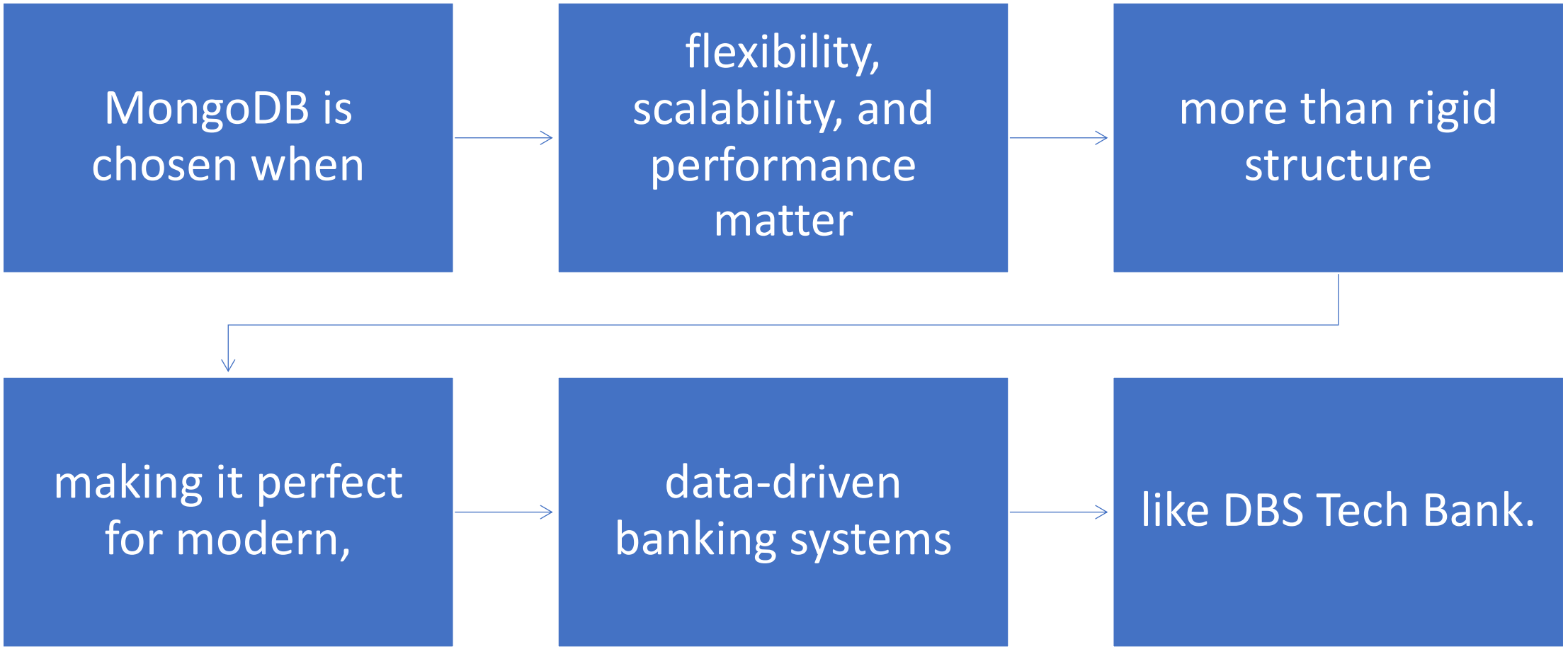
Fraud Detection:

Analyze transaction patterns using MongoDB aggregations.

Real-time Dashboards:

Build analytics dashboards using MongoDB + PowerBI or Tableau.

Summary


MongoDB is chosen when
flexibility, scalability, and performance matter
more than rigid structure
making it perfect for modern,
data-driven banking systems
like DBS Tech Bank.

Module 2: MongoDB Essentials

Indexing & Query Optimization

Data Import/Export using

mongoimport, mongoexport

MongoDB Architecture

Database → Collection → Document

Quick mental model

- **Database**: logical namespace (e.g., dbs_bank).
- **Collection**: group of similar documents (e.g., customers, accounts, transactions).
- **Document**: JSON-like object stored as BSON (binary JSON).

DBS lens

Keep **operational data**

profiles, accounts, txns separated

from **analytics** (downstream warehouse).

DBS lens

Use **replica sets** for

high availability;

consider **sharding** on

high-volume collections (e.g., transactions).

BSON & JSON Structure

Concept	JSON	BSON (Mongo internal)	Why it matters
Types	string, number, bool, array, object	+ Date, ObjectId, Decimal128, Binary, Regex	Accurate money (Decimal128), timestamps, IDs
Order	Unordered by spec	Maintains field order	Slightly impacts storage; not query logic
Size	Text	Compact binary	Faster IO, rich types

Indexing & Query Optimization

Why indexes?

Speed up selective queries.

Trade-off: extra storage + write overhead.

Aim for **high-cardinality** and **query prefixes**.

Module 3: Hands-On Lab

Lab 1: Create and Query a MongoDB Customer Database

Create a customers collection

Insert 10 customer profiles (JSON documents)

Query customers using filters (find(), $and, $or)

Update address & contact info using update operators

Lab 1: Create and Query a MongoDB Customer Database

- Create a dbs_bank database with a customers collection.
- Insert **10 customer profiles** (JSON docs).
- Practice queries using find(), **filters**, $and, $or, **projection**, **sorting**, **pagination**.
- Update **addresses** and **contact** with update operators.

Data Model & Sample Schema

Field	Type	Notes / Example
_id	ObjectId	Auto-assigned
customerId	string	Business ID, e.g., "C1001"
name	object	{ first, last }
mobile	string	Unique, Indian format
email	string	Optional

Data Model & Sample Schema

Field	Type	Notes / Example
kyc	object	{ pan, aadhaar_verified }
addresses	array	{ type, line1, city, state, pin }
products	array	e.g., ["savings","credit_card"]
createdAt	Date	new Date()

NoSQL Fundamental For Reference

Agenda

What is NoSQL?

Why NoSQL?

Brief History of NoSQL Databases

Difference Between RDBMS and NoSQL Databases

Benefits of NoSQL

Agenda

- Types of NoSQL
- CAP Theorem
- Advantages of NoSQL
- Disadvantages of NoSQL

Why NoSQL?

The concept of NoSQL databases became popular with Internet giants like Google, Facebook, Amazon, etc. who deal with huge volumes of data.

The system response time becomes slow when you use RDBMS for massive volumes of data.

To resolve this problem, we could "scale up" our systems by upgrading our existing hardware. This process is expensive.

Why NoSQL?

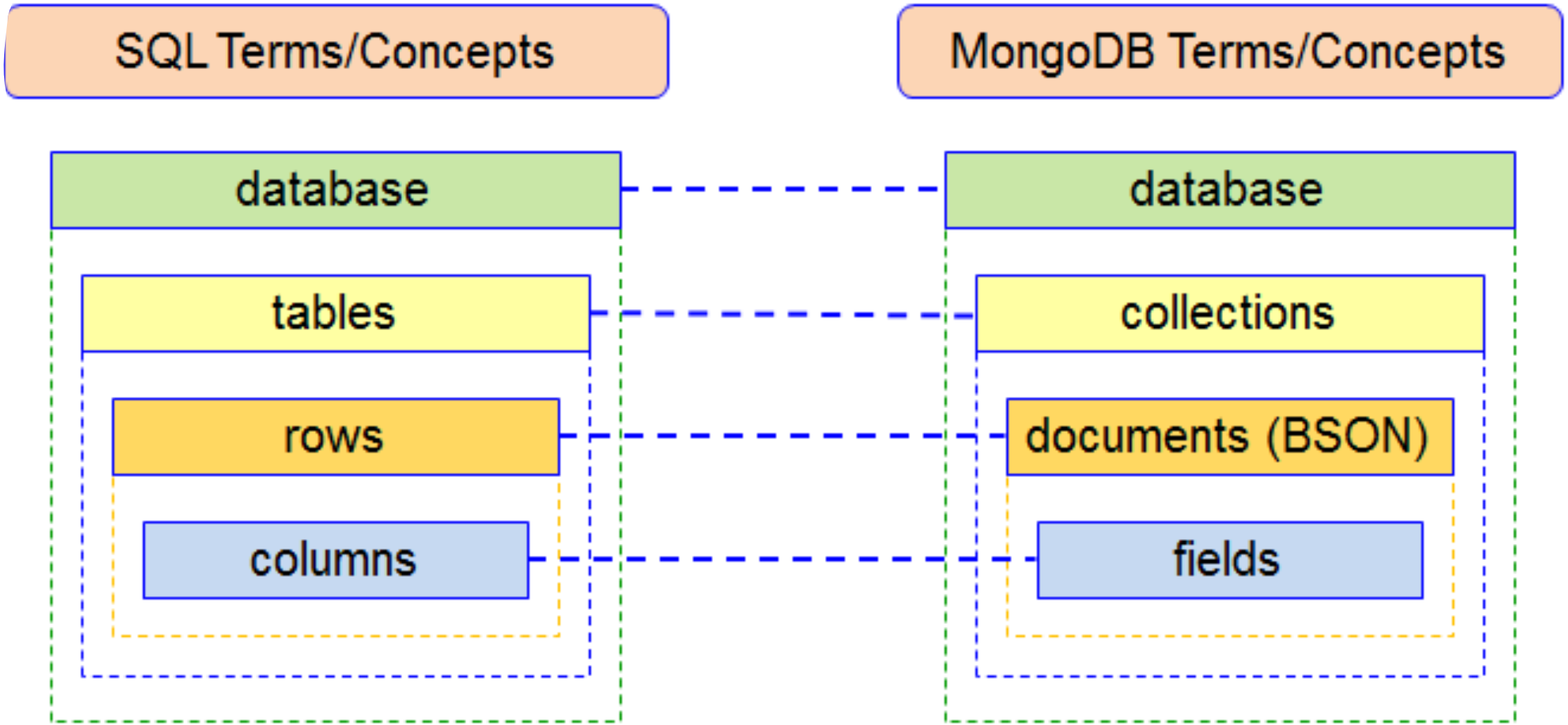
The alternative for this issue is to distribute database load on multiple hosts whenever the load increases. This method is known as "scaling out."

NoSQL database is non-relational, so it scales out better than relational databases as they are designed with web applications in mind.

SQL	NoSQL
SQL is a Relational Database.	NoSQL is a Non-Relational Database.
SQL is Table based.	NoSQL is Document based.
It has predefined schema for structured data.	It has dynamic schema for unstructured data.
SQL are vertically scalable.	NoSQL is horizontally scalable.
SQL is not fit for hierarchical work	NoSQL is best fit for hierarchical work as it follows key-value pair way of storing data.

Difference Between RDBMS and NoSQL Databases

Relational Database	NoSQL Database
1. Handles data coming in low velocity	1. Handles data coming in high velocity
2. Data arrive from one or few locations	2. Data arrive from many locations
3. Manages structured data	3. Manages structured unstructured and semi-structured data.
4. Supports complex transactions (with joins)	4. Supports simple transactions
5. Single point of failure with failover	5. No single point of failure
6. Handles data in the moderate volume	6. Handles data in very high volume
7. Centralized deployments	7. Decentralized deployments
8. Transactions written in one location	8. Transaction written in many locations
9. Gives read scalability	9. Gives both read and write scalability
10. Deployed in vertical fashion	10. Deployed in Horizontal fashion


SQL Terms/Concepts
MongoDB Terms/Concepts
database
tables
rows
columns
database
collections
documents (BSON)
fields

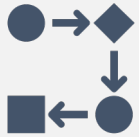
Benefits of NoSQL

It allows developers to create large volumes of structured, semi-structured as well as unstructured data for making the application diverse

Not restricting its use because of the type of data being used within the application.

It also allows agile development.

Benefits of NoSQL



Rapid iteration along with frequent code pushes, which makes it more popular.

Used with object-oriented programming (OOP), which makes it easy to use with flexibility.

Data can be stored more efficiently, making it less expensive, providing massive architecture.

Types of NoSQL

Types of NoSQL databases and the name of the databases system that falls in that category are:

MongoDB falls in the category of NoSQL document based database.

Key value store: Memcached, Redis, Coherence

Tabular: Hbase, Big Table, Accumulo

Document based: MongoDB, CouchDB, Cloudant

1. Key-Value Database

Data is stored in key/value pairs.

It is designed in such a way to handle lots of data and heavy load.

Key-value pair storage databases store data as a hash table where each key is unique, and the value can be a JSON, BLOB(Binary Large Objects), string, etc.

For example, a key-value pair may contain a key like "Website" associated with a value like "gktcs".

Key-Value Database

Key	Value
Name	Joe Bloggs
Age	42
Occupation	Stunt Double
Height	175cm
Weight	77kg

1. Key-Value Database

It is one of the most basic NoSQL database example.

This kind of NoSQL database is used as a collection, dictionaries, associative arrays, etc.

Key value stores help the developer to store schema-less data. They work best for shopping cart contents.

Redis, Dynamo, Riak are some NoSQL examples of key-value store DataBases. They are all based on Amazon's Dynamo paper.

2. Column-based

Column-oriented databases work on columns and are based on BigTable paper by Google.

Every column is treated separately. Values of single column databases are stored contiguously.

Deliver high performance on aggregation queries like SUM, COUNT, AVG, MIN etc. as the data is readily available in a column.

2. Column-based

Column-based NoSQL databases are widely used to manage data warehouses, business intelligence, CRM, Library card catalogs.

NoSQL query examples of column based database - HBase, Cassandra, HBase, Hypertable.

Column based NoSQL database

ColumnFamily			
Row Key	Column Name		
	Key	Key	Key
	Value	Value	Value
	Column Name		
	Key	Key	Key
	Value	Value	Value

3. Document-Oriented

Document-Oriented NoSQL DB stores and retrieves data as a key value pair but the value part is stored as a document.

The document is stored in JSON or XML formats.

The value is understood by the DB and can be queried.

Relational Vs. Document

Col1	Col2	Col3	Col4
Data	Data	Data	Data
Data	Data	Data	Data
Data	Data	Data	Data

Document 1

```
{
  "prop1": data,
  "prop2": data,
  "prop3": data,
  "prop4": data
}
```

Document 2

```
{
  "prop1": data,
  "prop2": data,
  "prop3": data,
  "prop4": data
}
```

Document 3

```
{
  "prop1": data,
  "prop2": data,
  "prop3": data,
  "prop4": data
}
```

Relational Vs. Document

In diagram on left we can see we have rows and columns, and in the right, we have a document database which has a similar structure to JSON.

For the relational database, we have to know what columns we have and so on.

However, for a document database, we have data store like JSON object.

we do not require to define which make it flexible.

3. Document-Oriented

Used for CMS systems, blogging platforms, real-time analytics & e-commerce applications.

Not use for complex transactions which require multiple operations or queries against varying aggregate structures.

Popular Document originated DBMS systems - Amazon SimpleDB, CouchDB, MongoDB, Riak, Lotus Notes, MongoDB.

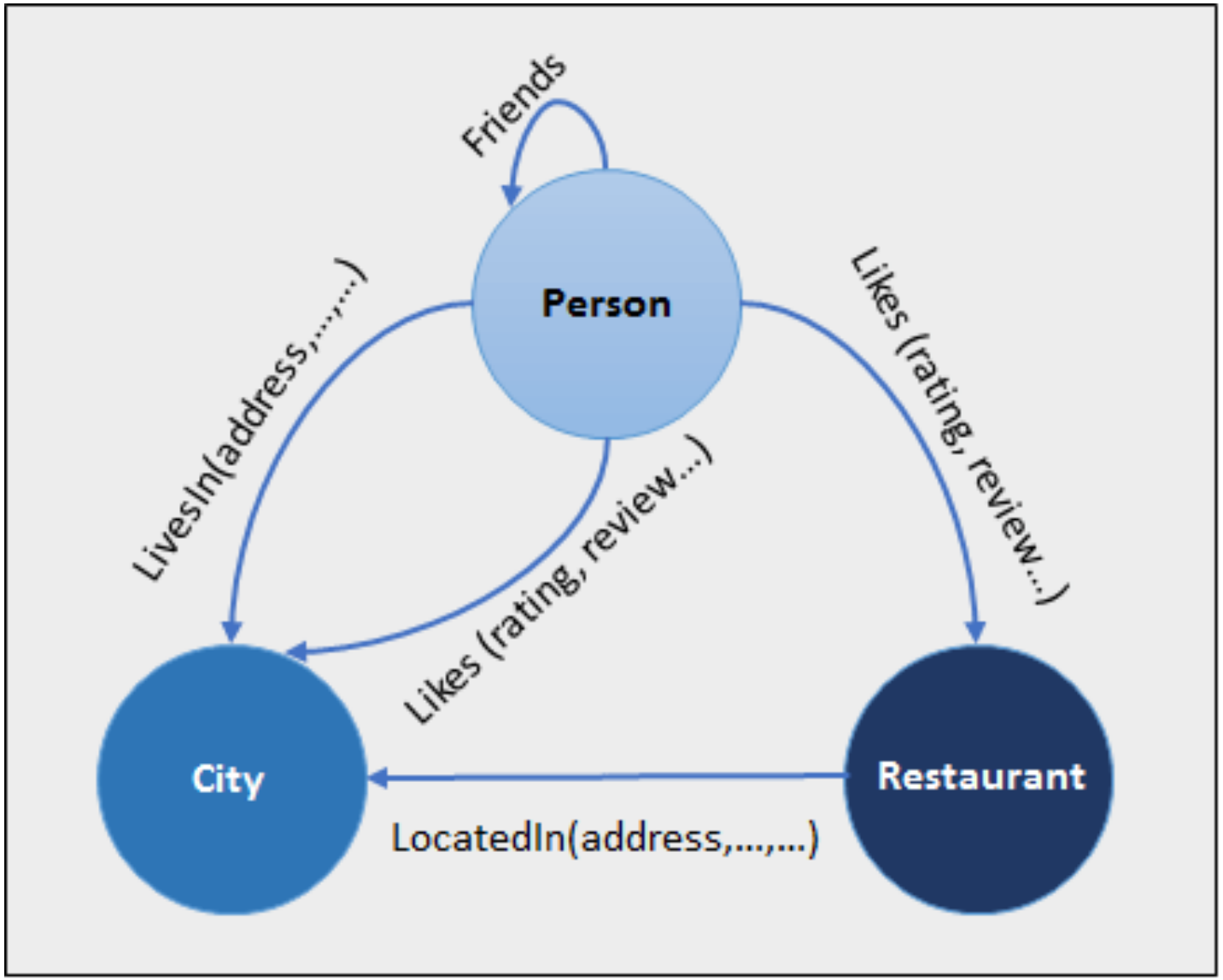
4. Graph-Based

Stores entities as well the relations amongst those entities.

The entity is stored as a node with the relationship as edges.

An edge gives a relationship between nodes. Every node and edge has a unique identifier.

4.
Graph-Based



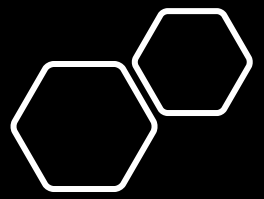
4. Graph-Based

Compared to a relational database where tables are loosely connected, a Graph database is a multi-relational in nature.

Traversing relationship is fast as they are already captured into the DB, and there is no need to calculate them.

Graph base database mostly used for social networks, logistics, spatial data.

Popular graph-based databases - Neo4J, Infinite Graph, OrientDB, FlockDB

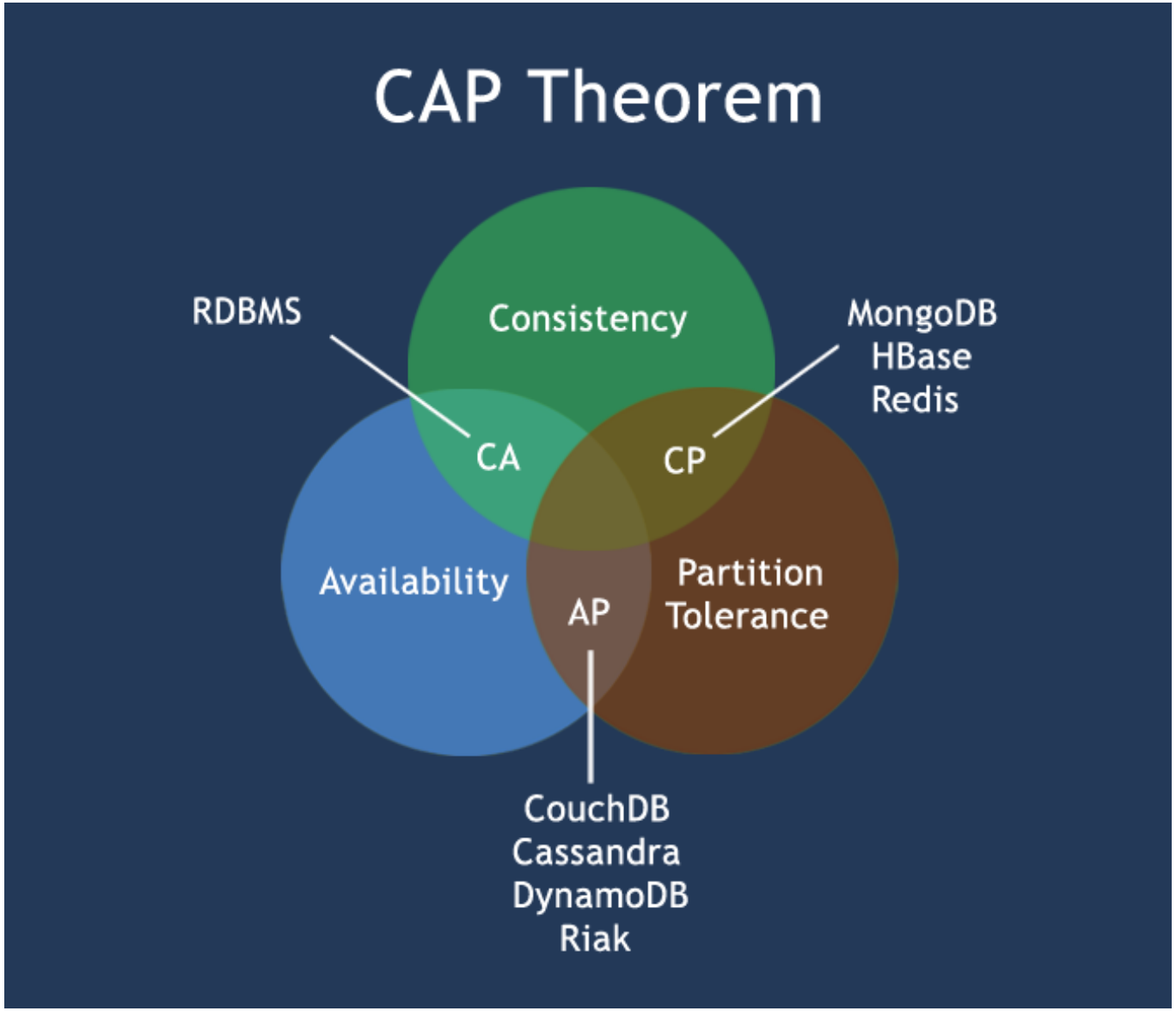


CAP Theorem

Also called brewer's theorem.

States that it is impossible for a distributed data store to offer more than two out of three guarantees

- Consistency
- Availability
- Partition Tolerance


CAP Theorem
RDBMS
Consistency
MongoDB
HBase
Redis
CA
CP
Availability
Partition
Tolerance
AP
CouchDB
Cassandra
DynamoDB
Riak

1. Consistency

The data should remain consistent even after the execution of an operation.

Means once data is written, any future read request should contain that data.

For example, after updating the order status, all the clients should be able to see the same data.

2. Availability

Database should always be available and responsive.

Should not have any downtime.

3. Partition Tolerance

Means that the system should continue to function even if the communication among the servers is not stable.

For example, the servers can be partitioned into multiple groups which may not communicate with each other.

Here, if part of the database is unavailable, other parts are always unaffected.

Advantages of NoSQL

- Can be used as Primary or Analytic Data Source
- Big Data Capability
- No Single Point of Failure
- Easy Replication

Advantages of NoSQL

- No Need for Separate Caching Layer
- It provides fast performance and horizontal scalability.
- Can handle structured, semi-structured, and unstructured data with equal effect
- Object-oriented programming which is easy to use and flexible

Advantages of NoSQL

NoSQL databases don't need a dedicated high-performance server

Support Key Developer Languages and Platforms

Simple to implement than using RDBMS

It can serve as the primary data source for online applications.

Advantages of NoSQL

- Handles big data which manages data velocity, variety, volume, and complexity
- Excels at distributed database and multi-data center operations
- Eliminates the need for a specific caching layer to store data
- Offers a flexible schema design which can easily be altered without downtime or service disruption

Disadvantages of NoSQL

- No standardization rules
- Limited query capabilities
- RDBMS databases and tools are comparatively mature
- It does not offer any traditional database capabilities, like consistency when multiple transactions are performed simultaneously.

Disadvantages of NoSQL

- When the volume of data increases it is difficult to maintain unique values as keys become difficult
- Doesn't work as well with relational data
- The learning curve is stiff for new developers
- Open source options so not so popular for enterprises.

Let’s Connect

surendra@gktcs.com

https://www.linkedin.com/in/surendrarp

https://www.gktcs.com

Happy Learning !!
Thanks for Your Patience ☺

Surendra Panpaliya

GKTCS Innovations