Introduction to Big Data
IN411

Course Contents

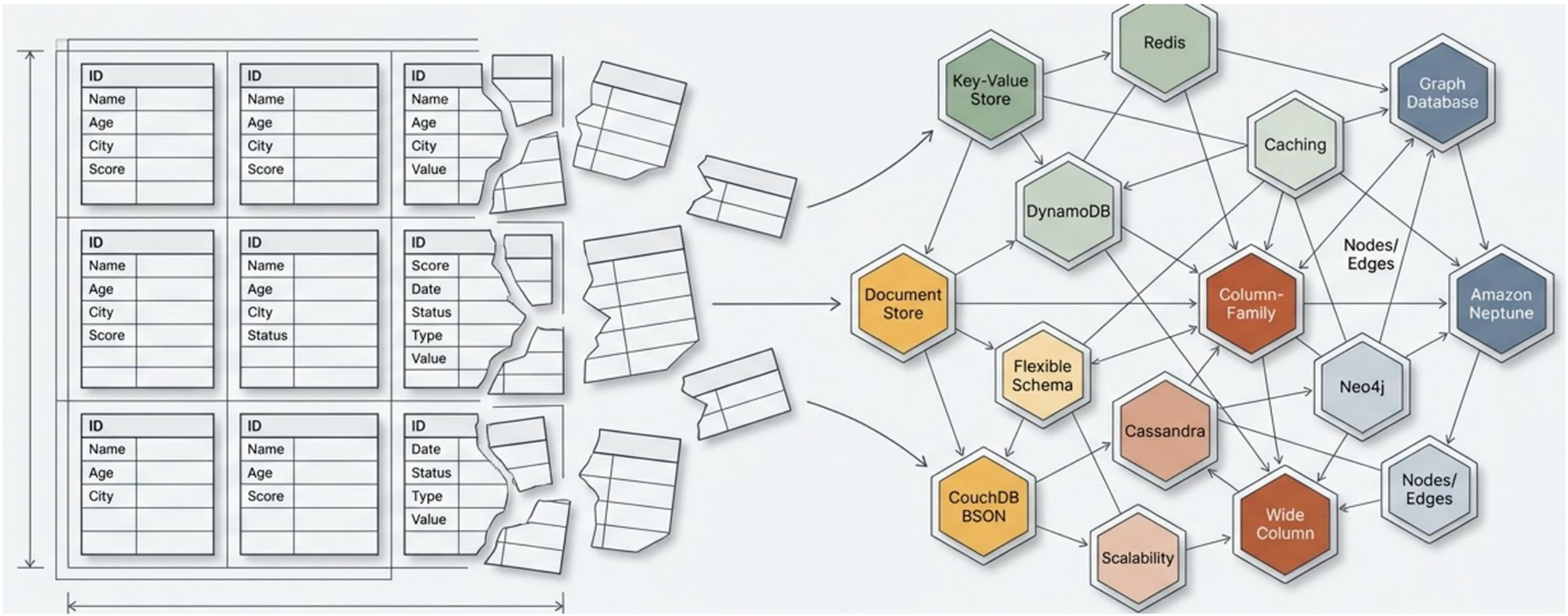
- Chapter 1: Overview of Big Data
- Chapter 2: Hadoop Ecosystem
- Chapter 3: Apache Spark Basics
- **Chapter 4: Introduction to NoSQL Databases**



Plan

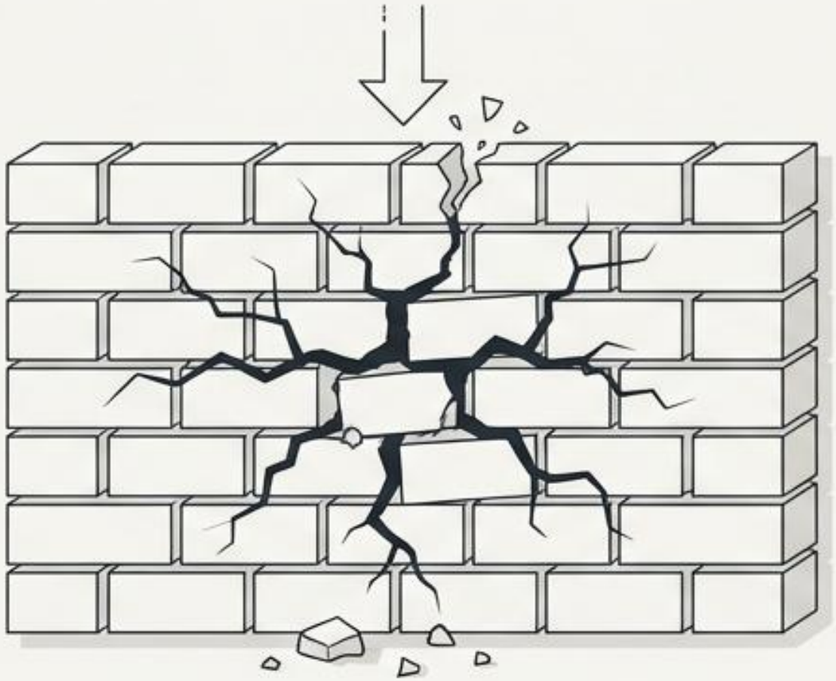
Introduction to NoSQL Databases

- NoSQL Databases
- CAP Theorem
- Four Core Data Models



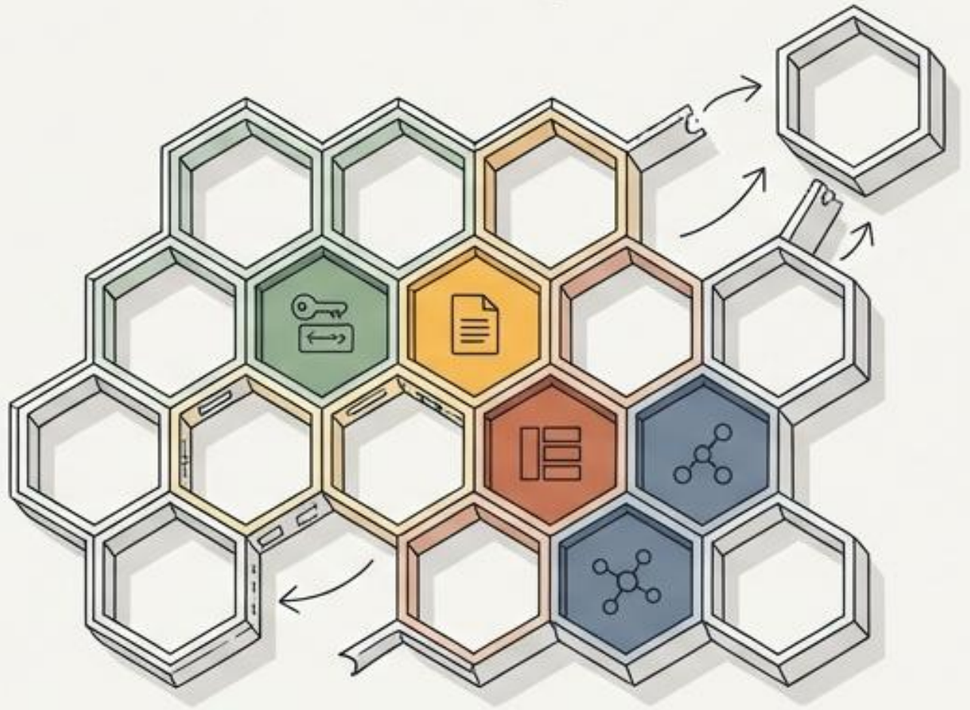
The Big Data Catalyst

The Schema Evolution Problem



Example: Modifying a rigid SQL schema to add new features (like mobile photo uploads) to millions of existing e-commerce reviews requires hours of downtime.

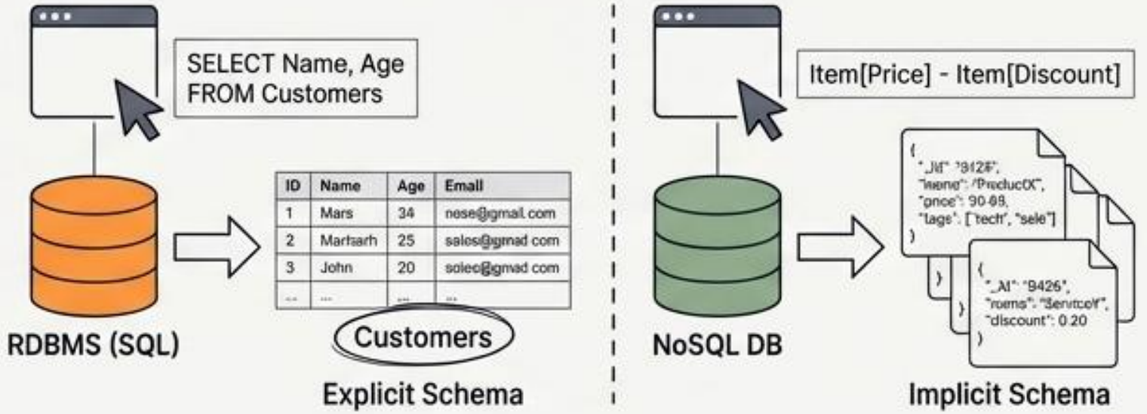
The Scale-Up Wall



RDBMS relies on Vertical Scaling (buying a bigger server). Big Data requires Horizontal Scaling (distributing load across many cheaper servers).

Key Takeaway: RDBMS is all-or-nothing. Modern web applications need flexibility, high availability, and massive scalability over strict, instantaneous consistency.

SQL vs. NoSQL: The Paradigm Shift

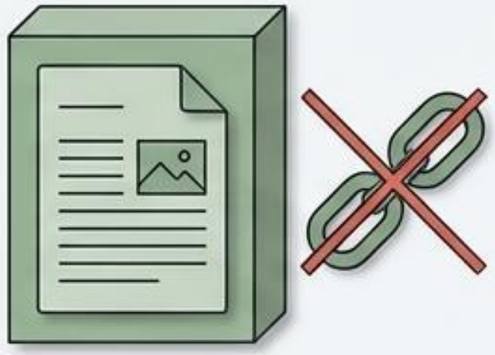


	SQL	NoSQL
Model	Relational (Tables/Rows)	Non-Relational (Collections/Documents/Graphs)
Schema	Rigid, predefined schema	Schema-free (Dynamic/Implicit)
Scaling	Vertical Scaling (Scale-Up)	Horizontal Scaling (Scale-Out)
Transaction Rules	Strict ACID compliance	BASE (Basically Available, Soft state, Eventual consistency)
Complexity	Complex JOINs	Denormalized, self-contained aggregates

Core Anatomy of NoSQL Systems

Non-Relational

Works with self-contained aggregates or BLOBs; no complex referential integrity joins or object-relational mapping.



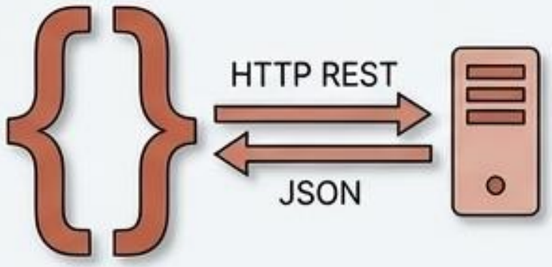
Schema-Free

Offers heterogeneous data structures in the same domain.
Schema-on-read rather than schema-on-write.



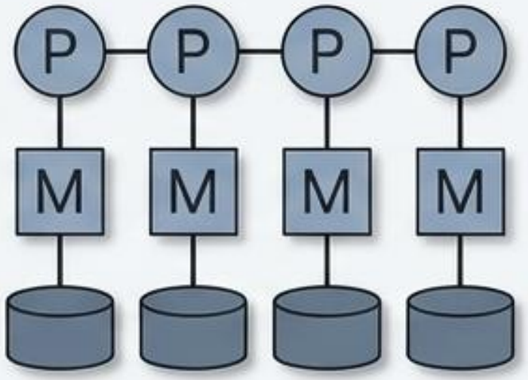
Simple API

Text-based protocols (mostly HTTP REST with JSON) allowing low-level data manipulation instead of a complex standard query language.



Distributed & Shared Nothing

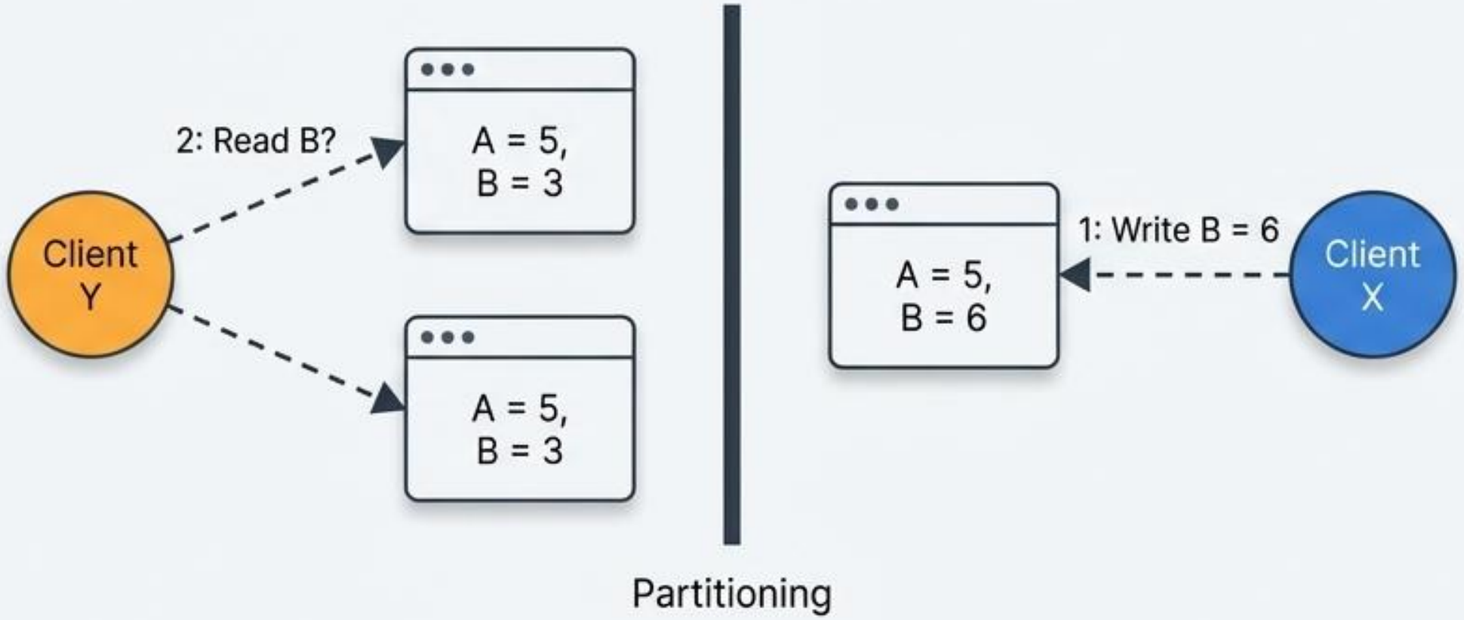
Auto-scaling, peer-to-peer multi-master replication.
Enables less coordination and higher distribution, providing eventual consistency.



Shared Nothing
e.g. “NoSQL”

The Law of Distributed Data: CAP Theorem

In a distributed system, network partitions (communication failures between nodes) are inevitable.
When a partition occurs, a database must choose between two outcomes.



Definition 1 (Consistency)

Every user has an identical view of the data at any given instant (Linearizability).

Definition 2 (Availability)

Every request receives a response, ensuring the system remains operational even if nodes fail.

Definition 3 (Partition Tolerance)

The database maintains operations despite network failures dropping messages between nodes.

Navigating CAP Trade-Offs

Because network partitions (P) are unavoidable in distributed systems, databases must trade off between Consistency (C) and Availability (A).

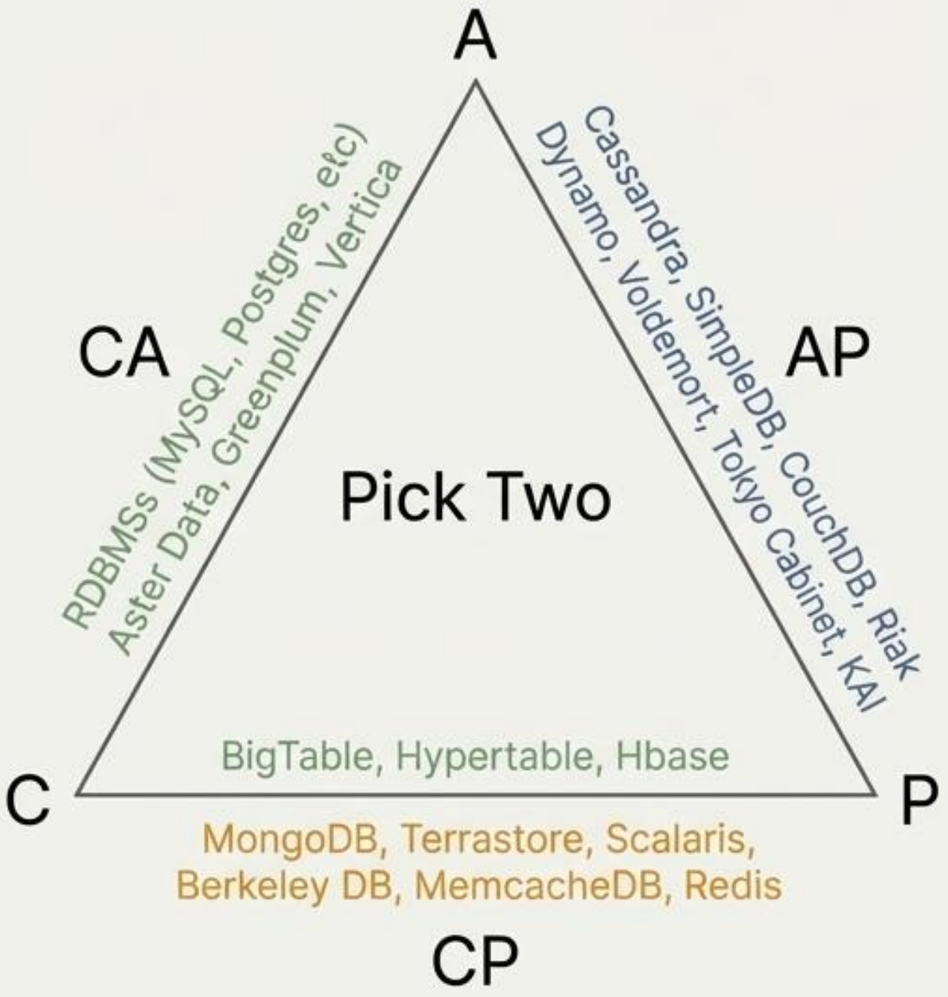
CA Edge (Consistency + Availability)

Traditional RDBMS (MySQL, Postgres). Works well when there are no physical network partitions (single node or tightly coupled).

CP Edge (Consistency + Partition Tolerance)

MongoDB, HBase, Redis

If a partition happens, the system will shut down access to out-of-sync nodes to prevent stale reads.



AP Edge (Availability + Partition Tolerance)

Cassandra, CouchDB, Dynamo

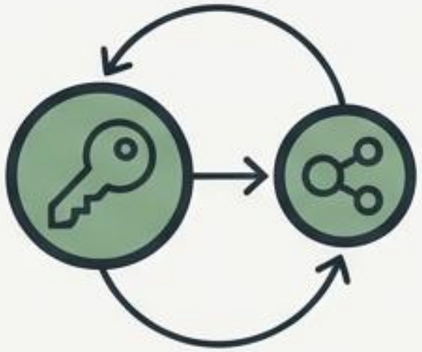
If a partition happens, the system stays online and returns the most recent local data, accepting Eventual Consistency.

The NoSQL Family Tree

Four distinct data models, each designed to solve specific scalability and structural challenges.

Category 1: Key-Value Stores

(Dictionary format)



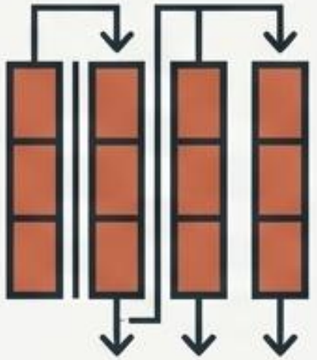
Category 2: Document-Oriented

(JSON/XML format)



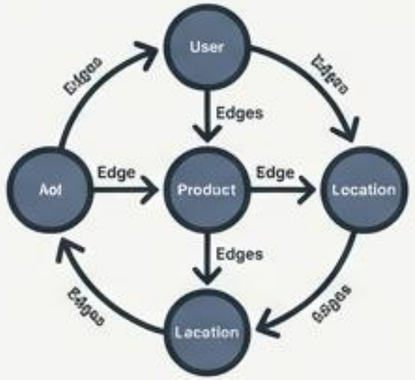
Category 3: Column-Family Stores

(Contiguous column format)



Category 4: Graph Databases

(Nodes and Edges)



Type 1: Key-Value Stores

Mechanism

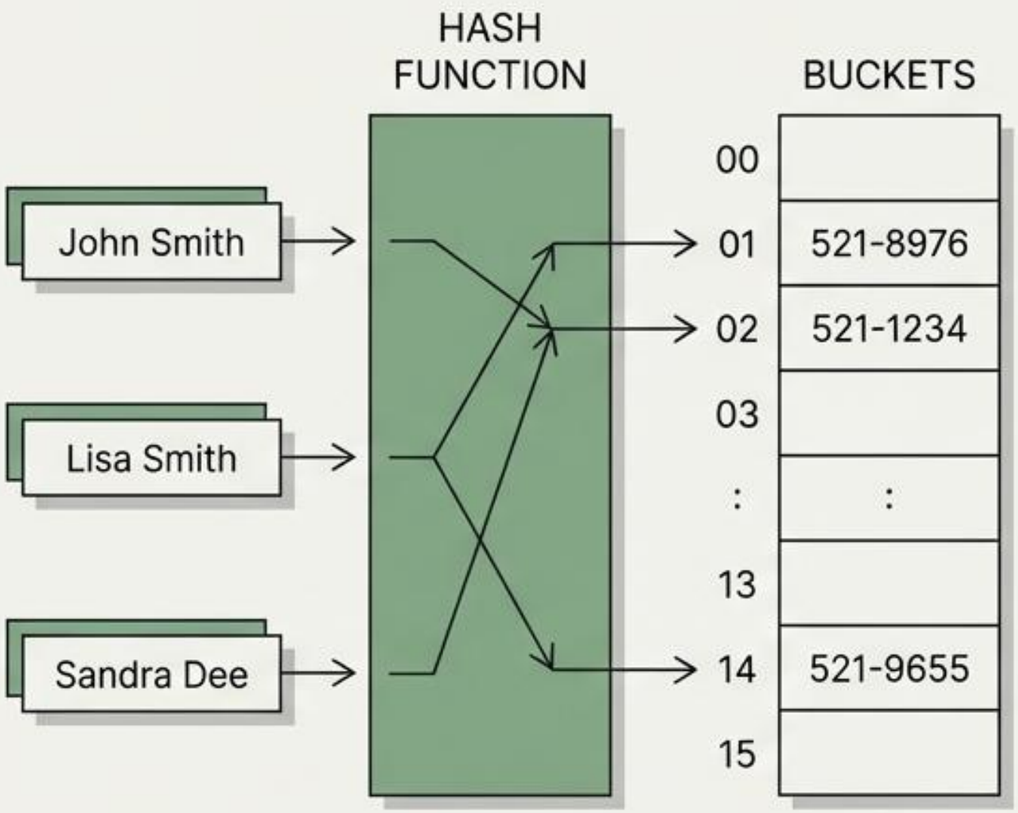
The simplest and fastest NoSQL model. Stores data as a massive hash table. Each unique key points to a specific value (which can be a string, JSON, or BLOB).

Strength

Incredibly fast read/write operations; highly partitionable.

Limitation

Cannot query the value itself (only the key).



Key	Value
Name	Joe Bloggs
Age	42

Cheat Sheet

Best For: Shopping cart contents, session management, in-memory caching.

Examples: Redis, Amazon DynamoDB, Memcached.

Type 2: Document-Oriented

Mechanism

Stores data as documents (typically JSON, BSON, or XML). Unlike Key-Value, the database understands the internal structure of the document, allowing for deep querying.

Strength

Ultimate schema flexibility. A single document can contain lists, nested data, and missing fields without breaking the database.

Limitation

Not ideal for complex transactions requiring multi-document operations.

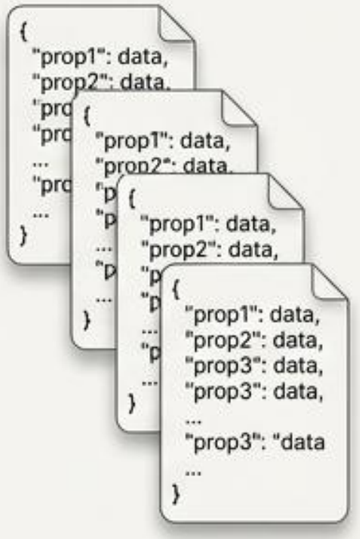
```
{
  "id": "Rongo",
  "name": "Jon",
  "capital": true,
  "friends": 198,
  "favoriteCountries": [
     {
       "name": "France",
       "capital": "Paris"
     },
     {
       "name": "Japan"
     }
   ]
  ],
   "friends": []
  },
  "friends": 354,
  "favorite Country": {
    "name": "Italy",
    "capital": "Rome"
  }
  ...
}
```

← Embedded document

← Embedded document

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

Rigid Table Structure



Flexible JSON Documents

Cheat Sheet

- **Best For:** Content Management Systems (CMS), user profiles, e-commerce product catalogs.
- **Examples:** MongoDB, CouchDB.

Type 3: Column-Family Stores

Mechanism:

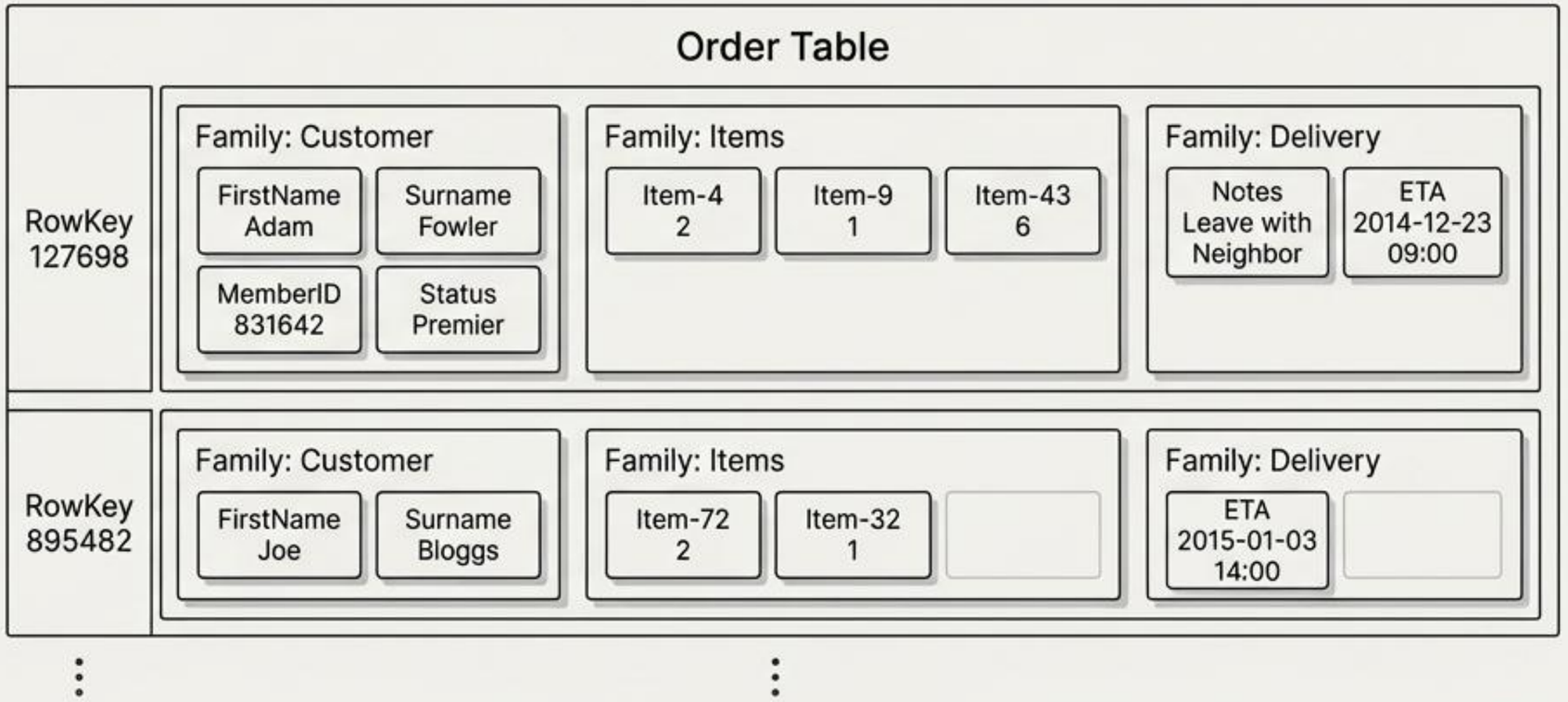
Organizes data by columns rather than rows. Related columns are grouped into Column Families. A row does not need to have an entry for all columns.

Strength:

Delivers extreme performance on aggregation queries (SUM, COUNT, AVG) because values of a single column are stored contiguously on disk.

Limitation:

Updates to individual records can be slower; poor for heavy transactional workloads.



Row Key	Column Name	
	Key	Value
	...	...
	Key	Value
	...	...

Cheat Sheet

Best For: Data warehousing, time-series data, heavy analytics, loggIng.

Examples: Apache Cassandra, HBase, Google BigTable.

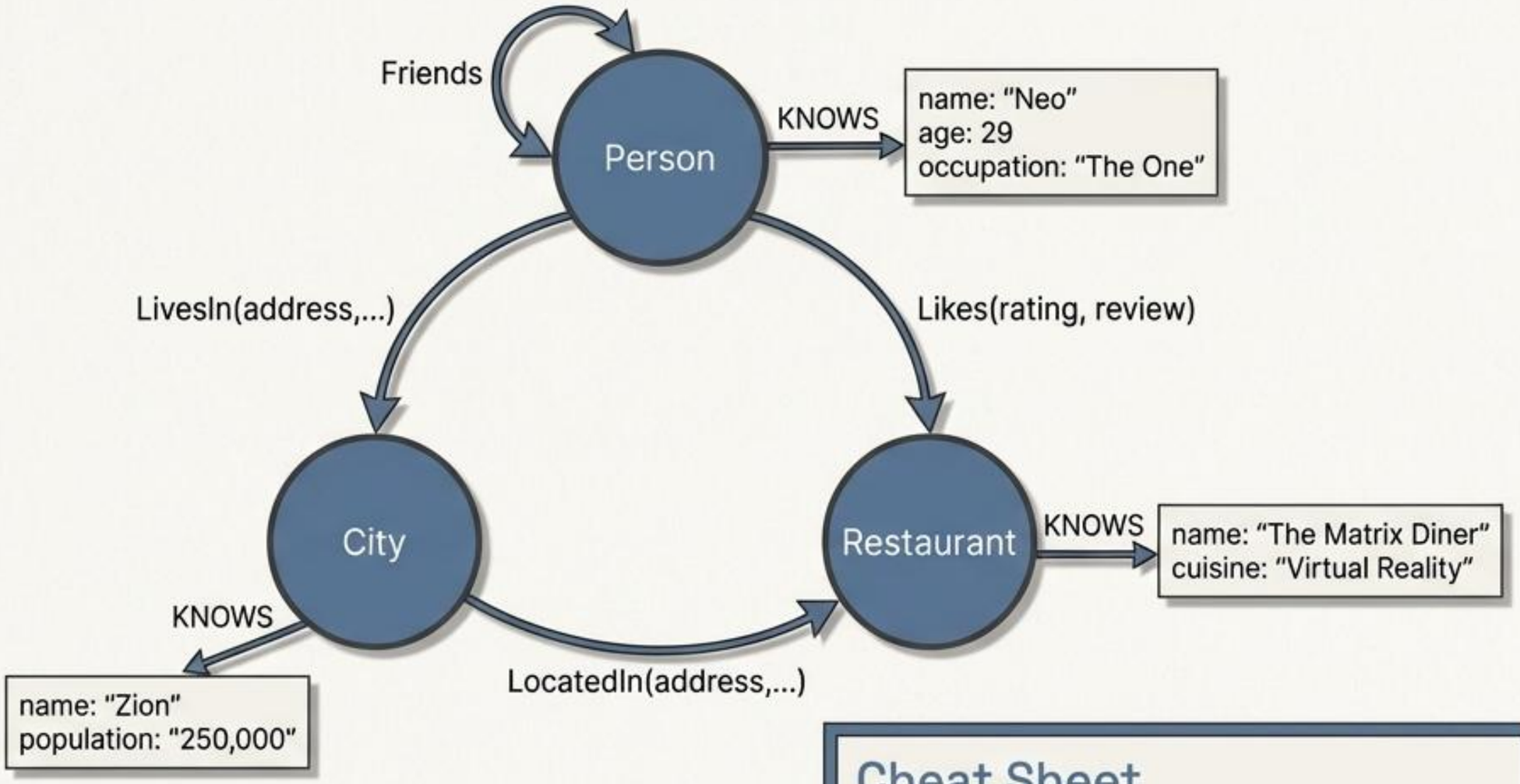
Type 4: Graph Databases

Mechanism

Multi-relational by nature. Treats the relationships between data points as equally important as the data itself. Entities are Nodes and relationships are Edges.

Strength

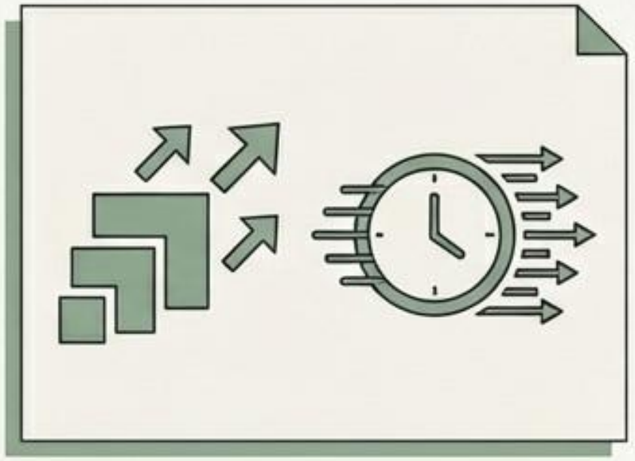
Traversing relationships is lightning fast because connections are physically captured in the database structure, eliminating the need for expensive JOIN calculations.



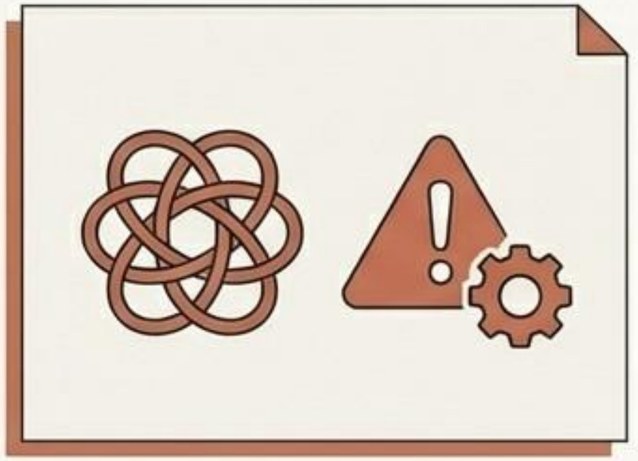
Cheat Sheet

- Best For: Social networks, recommendation engines, spatial data, fraud detection.
- Examples: Neo4j, OrientDB, Infinite Graph.

The Trade-offs: Evaluating NoSQL







Advantages (Pros):

- Big Data Capability: Handles massive volume, velocity, and variety.
- High Availability: Easy replication with no single point of failure.
- Agility: Flexible schema design allows alteration without downtime.

Disadvantages (Cons):

- No Standardization: Every database has its own API and learning curve.
- Weaker Guarantees: Sacrifices ACID properties (Consistency) for eventual consistency.
- Analytics Gap: Limited capability for complex, ad-hoc JOIN queries natively.

NoSQL in the Wild: Real-World Applications



Panel 1: Social Media Networking

Use Case: Managing unstructured posts, comments, and relationship networks.

Tech Match: **Graph DBs** (for friend connections) and **Document DBs** (for varied post formats).



Panel 2: E-Commerce & Retail

Use Case: High-speed shopping carts and dynamic product catalogs with varying attributes.

Tech Match: **Key-Value stores** (for session data) and **Document DBs** (for product catalogs).



Panel 3: Data Mining & Analytics

Use Case: Efficiently storing and analyzing massive sets of unstructured data points.

Tech Match: **Column-Family stores** (for rapid aggregation queries).

The Ultimate Study Cheat Sheet

Database Type	Data Model	Primary CAP Focus	Best Use Case	Popular Examples
Key-Value	Hash Table / Dictionary	AP or CP	Caching, Carts, Sessions	Redis, DynamoDB
Document	JSON / XML / BSON	CP or AP	CMS, Catalogs, Profiles	MongoDB, CouchDB
Column-Family	Tabular (By Column)	AP or CP	Analytics, Data Warehousing	Cassandra, HBase
Graph	Nodes & Edges	CA or CP	Social Networks, Fraud Detection	Neo4j, OrientDB