THE UNIVERSITY of EDINBURGH

Advanced Database Systems

Spring 2026

Lecture #27:

NoSQL

NoSQL Motivation

Driven by Web 2.0 Applications

- Emergence of massive-scale applications (e.g., Facebook, Amazon, Instagram)
- Need for handling high-volume, real-time data operations (OLTP)
- Load can increase rapidly with web traffic and unpredictably

Core problems with distributed RDBMS

Scaling transactions across multiple nodes is hard

- Traditional protocols like 2PC are too slow
- Require all nodes to agree before committing – too much coordination overhead at scale

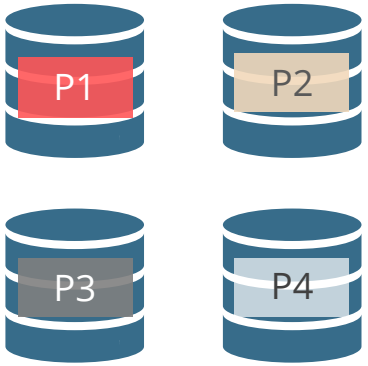
Consistency is hard to enforce when data is partitioned & replicated

- Keeping every copy in sync is expensive – and sometimes impossible w/o sacrificing availability

Scaling through Partitioning

Partition (shard) data across multiple machines

- Enables data to fit into main memory for faster access
- User queries / transactions are spread across multiple machines



Advantages

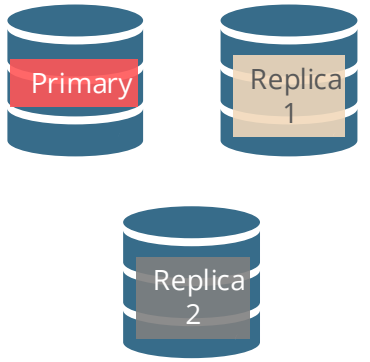
- **Higher throughput:** can handle more clients simultaneously
- **Efficient writes:** updates impact only a single data copy

Disadvantages

- **Expensive reads:** retrieving data may need accessing many machines, increasing latency
- **Concurrency challenges:** reads need locks on each machine to handle concurrent writes

Scaling through Replication

Create multiple copies (replicas) across machines



Each database partition is replicated across multiple nodes

Queries can be distributed among replicas for load balancing

Advantages

Better throughput & latency: clients can query different replicas, reducing latency

Improved fault tolerance: if a machine fails, another replica can serve the request

Efficient reads: multiple replicas make read operations faster and more scalable

Disadvantages

Expensive writes: every write must update all replicas to maintain consistency

Potentially stale reads: If updates aren't synced properly, replicas may serve outdated data

NoSQL: "Not Only SQL"

A paradigm shift

- Focus on scalability and performance
- Complements, rather than replaces, RDBMS

Trade-off

- Scalability and performance through horizontal scaling (sharding and replication)
- Sacrifice consistency and complex analytics (OLAP)

NoSQL: Core Principles

Flexible schema

- No upfront schema definition
- Documents can have different fields
- Adapts to evolving data needs

Simplified data models

- Designed for speed and horizontal scalability
- Restricted operations are easier to distribute

Horizontal scaling

- Scale out by adding commodity machines rather than scaling up expensive hardware

NoSQL Popularity

https://trends.google.com/trends/explore?date=all&q=NoSQL

Recap: ACID in Relational DBMS

Atomicity: *All* actions in the txn happen, or *none* happen

"all or nothing"

Consistency: If each txn is consistent and the DB *starts* consistent, then it *ends* up consistent

"it looks correct to me"

Isolation: Execution of one txn is isolated from that of other txns

"as if alone"

Durability: If a txn commits, its effects persist

"survive failures"

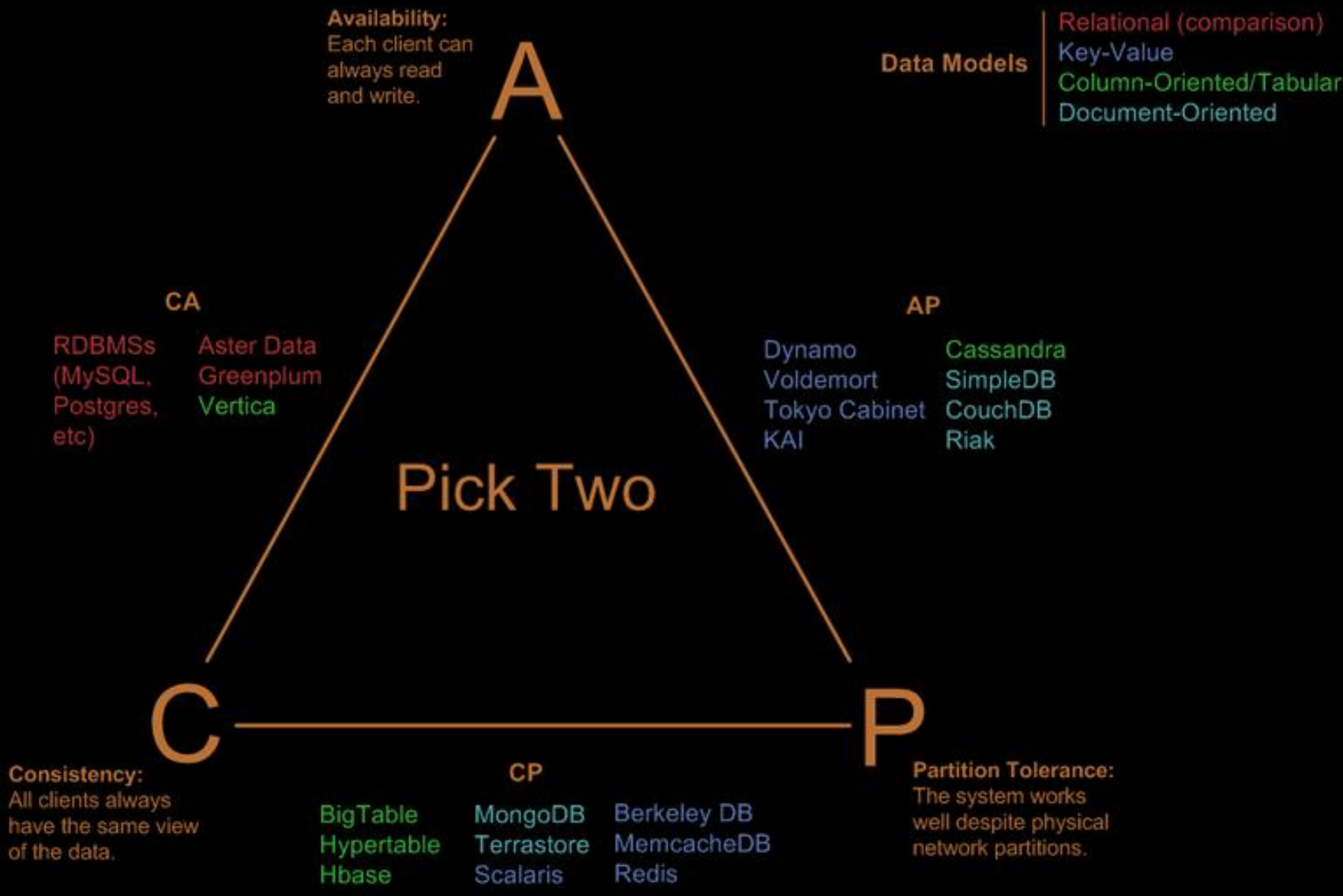
CAP Theorem

> *"Of three properties of shared-data systems –*
> *data Consistency, system Availability, and tolerance to network Partitions –*
> *only two can be achieved at any given moment in time" – Brewer, 1999*

Consistency	All nodes see the same data at the same time. Every read returns the most recent write (or an error)
Availability	Every request receives a response – success or failure. The system never just refuses to answer
Partition tolerance	The system keeps operating despite arbitrary message loss or failure of part of the system

"Consistency" in ACID means integrity constraints are satisfied – a different meaning from "Consistency" in the CAP theorem, which means all nodes see the same data at the same time

Visual Guide to NoSQL Systems



https://blog.nahurst.com/visual-guide-to-nosql-systems

NoSQL Paradigm: BASE

Basically Available

Guarantees availability even during failures

The system always responds, though the response may not have the freshest data

Soft State

The system state can change over time, even without updates as replicas converge

Replicas may temporarily have different data until synchronised

Eventually Consistent

Data will eventually become consistent across all nodes

No guarantee of immediate consistency, but eventual convergence

Taxonomy of NoSQL Systems

Key-Value Stores

Data is stored as key-value pairs. Value is opaque

Systems: Redis, DynamoDB, Memcached

Use cases: Caching, session storage, simple data storage

Document Stores

Value is structured data (JSON/BSON). Can query inside value. Schema-free

Systems: MongoDB, CouchDB

Use cases: User profiles, content management, e-commerce applications

Taxonomy of NoSQL Systems (Cont.)

Column-Family Stores

Data organised into columns and column families. Excellent for wide, sparse tables

Systems: Cassandra, HBase (open-source implementation of Google's BigTable)

Use cases: Log processing, time-series data, large-scale analytics

Graph Databases

Data is stored as nodes and edges. Optimised for traversal queries

Systems: Neo4j, ArangoDB, Amazon Neptune

Use cases: Social networks, fraud detection, recommendation engines

Key-Value Stores

Data model: `(key,value)` pairs

Key = string/integer, unique for the entire data

Value = can be anything (very complex object)

Key	Value
`user:42`	`{ name:"Ali", age:24, ... }`
`session:xyz`	`"logged_in"`
`counter:views`	`1048576`

Operations: `get(key)`, `put(key,value)`, `del(key)`

Operations on value not supported

Partitioning & Replication: using hashing

Partitioning: key *k* is stored at server *h(k)*

Multiway replication: e.g., key *k* stored at *h1(k), h2(k), h3(k)*

On update, propagate changes to the other servers (**eventual consistency**)

Problem: stale reads

A read after a write may return old data if it hits an un-synced replica. Apps must tolerate this!

Document Stores

Motivation

In key-value stores, the *value* is often a very complex object

Example: `key = '18/05/2024', value = [ all flights that date ]`

Better approach: store the *value* as structure data

Formats like JSON, Protobuf, or XML are commonly used

Query any nested field

Example: "*Find all documents in the **users** collection where the **age** field is greater than 25*"

MongoDB syntax: `db.users.find({age: {$gt: 25}})`

"Document" is simply structured data

A document database is a collection of documents

Each document can represent a complex data model

JSON: Semi-Structured Data Model

Human-readable data interchange

Text-based, open standard for exchanging data between systems

Core structures

Object: A collection of key-value pairs

Array: An ordered list of values

Data types in JSON

Atomic values: e.g., strings, numbers

Objects: Nested JSON objects

Arrays: A list of values, can include objects or other arrays

```
{
    "firstName": "John",
    "lastName": "Smith",
    "age": 25,
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": "10021"
    },
    "phoneNumber": [
        { "type": "home", "number": "212 555-1234" },
        { "type": "fax", "number": "646 555-4567" }
    ]
}
```

Relational Data Model

Rigid, Flat Structure:
Data is stored in tables

Fixed Schema:
Schema must be defined in advance

Binary Representation:
Good for performance, bad for exchange

Query Language:
Based on Relational Algebra,
joins are first-class citizens

Semi-Structured Data Model (JSON)

Flexible, Nested Structure:
Data is organised in trees (objects and arrays)

No Predefined Schema:
JSON is "self-describing", allowing flexibility

Text Representation:
Good for exchange, bad for performance

Query Language:
NoSQL use their own QL and avoid joins,
RDBMS support JSON via SQL extensions

Graph Databases

Represent data as a graph structure

- Nodes: entities (e.g., people, products)
- Edges: relationships (e.g., “friends_with”, “bought”)
- Properties: attributes on nodes/edges

Core strengths

- Efficient traversals & pattern matching
- Flexible / schema-optional
- Handles highly connected data at scale

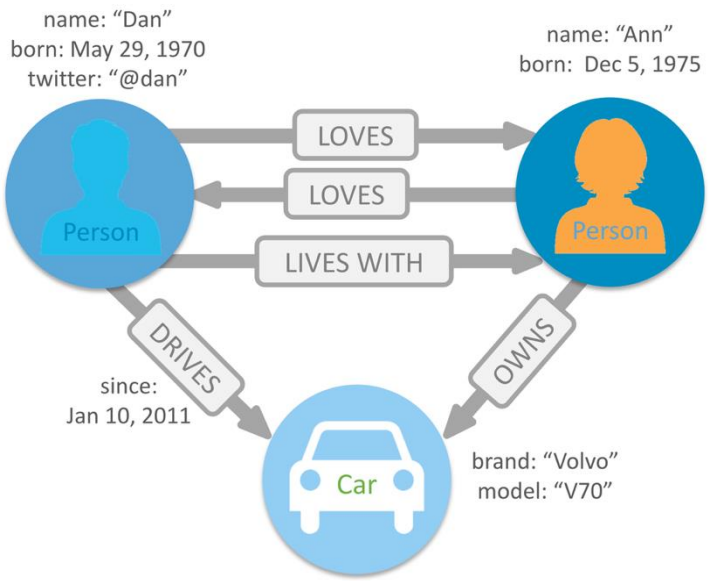


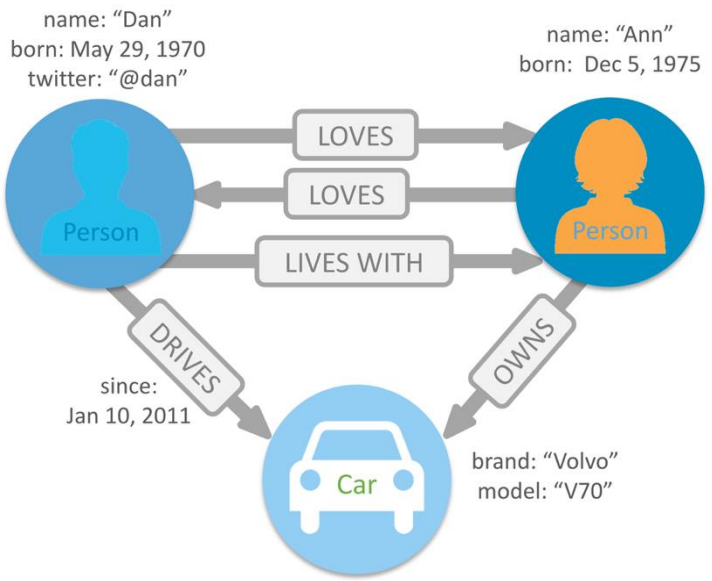
Image credit: neo4j.com

Graph Query Languages

Designed to express patterns of nodes and relationships

Common graph queries

- Pattern matching: find subgraphs (e.g., “friends of friends”)
- Traversals: follow paths across relationships
- Path queries: shortest path, connectivity, variable-length paths



Common languages

- GQL (Graph Query Language): emerging ISO standard
- SQL/PGQ: Property Graph Queries extension for SQL
- Cypher (used by Neo4j)
- SPARQL (for RDF graphs / semantic web)

```
// Example query (GQL)
SELECT car
FROM myGraph
MATCH (p:Person WHERE p.name = 'Dan')-[:DRIVES]->(car);

// Example query (Cypher)
MATCH (:Person{name:"Dan"})-[:DRIVES]->(car)
RETURN car;
```

Ranking of DBMS Technologies 2026

429 systems in ranking, March 2026

Rank			DBMS	Database Model	Score		
Mar 2026	Feb 2026	Mar 2025			Mar 2026	Feb 2026	Mar 2025
1.	1.	1.	Oracle	Relational, Multi-model	1182.46	-21.05	-70.62
2.	2.	2.	MySQL	Relational, Multi-model	858.34	-9.88	-129.79
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	711.47	+3.33	-76.67
4.	4.	4.	PostgreSQL	Relational, Multi-model	680.08	+8.05	+16.66
5.	5.	5.	MongoDB	Document, Multi-model	383.58	+4.85	-12.85
6.	6.	6.	Snowflake	Relational	211.24	+3.10	+49.46
7.	↑ 8.	↑ 13.	Databricks	Multi-model	145.81	+1.29	+49.80
8.	↓ 7.	↓ 7.	Redis	Key-value, Multi-model	145.19	-1.85	-10.17
9.	9.	9.	IBM Db2	Relational, Multi-model	111.38	+0.16	-15.19
10.	10.	↓ 8.	Elasticsearch	Multi-model	103.58	-2.88	-27.80
11.	11.	11.	Apache Cassandra	Wide column, Multi-model	101.88	+0.28	-4.78
12.	12.	↓ 10.	SQLite	Relational	95.97	-3.22	-17.11
13.	13.	↑ 14.	MariaDB	Relational, Multi-model	87.00	+0.91	-7.23

Based on #mentions (e.g., stack overflow), google trends, job postings, profile data on LinkedIn, tweets…

http://db-engines.com/en/ranking

SQL vs NoSQL

Choose SQL when…

- Strong ACID guarantees needed
- Complex ad-hoc queries & joins
- Schema is stable and well-defined
- Financial systems, healthcare, inventory

Choose NoSQL when…

- Horizontal scale is essential
- High write throughput required
- Flexible / evolving schema
- Social media, catalogues, real-time feeds

The boundary is blurring

- Modern RDBMSs now support storing and querying JSON data (e.g., PostgreSQL)
- NoSQL systems can support ACID txns (e.g., MongoDB)
- Distributed SQL systems provide scalability & strong consistency (e.g., CockroachDB)

NoSQL and SQL coexist – the skill is understanding the trade-offs well enough to choose correctly

Right tool depends on workload, not hype