universität wien

Data Engineering

Chapter 1-4

VU Information Management & Systems Engineering

Peter Kalchgruber

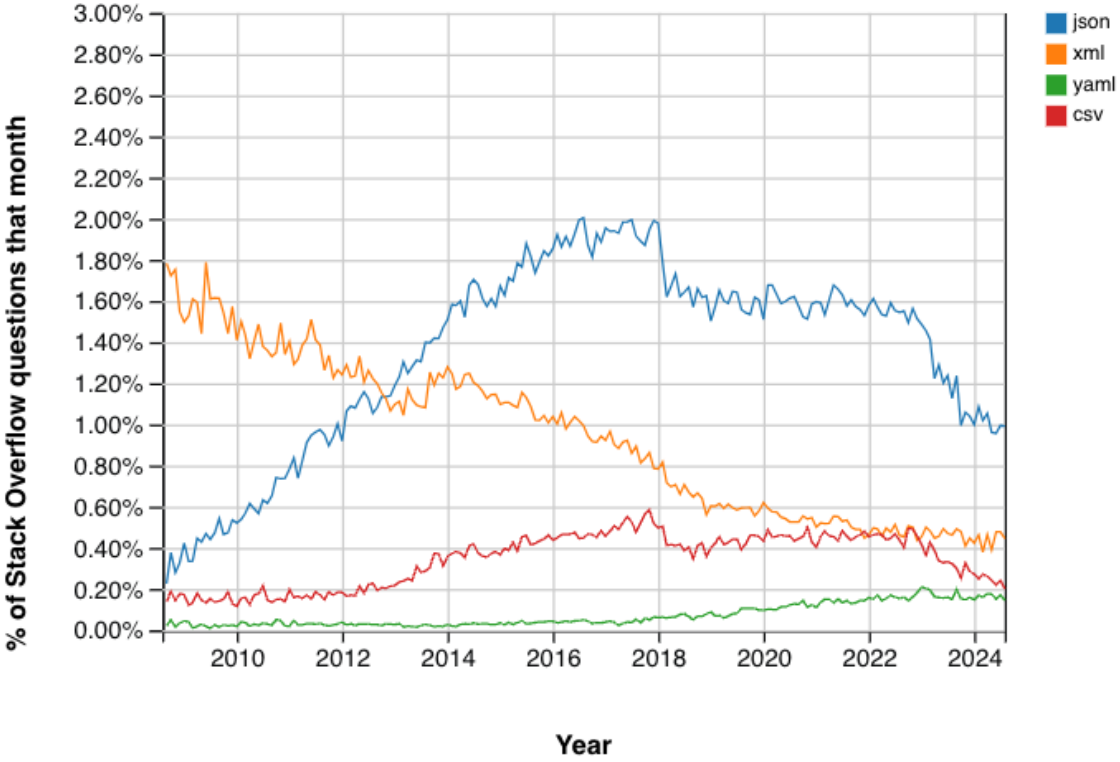
JSON

JavaScript Object Notation

JavaScript Object Notation (JSON)

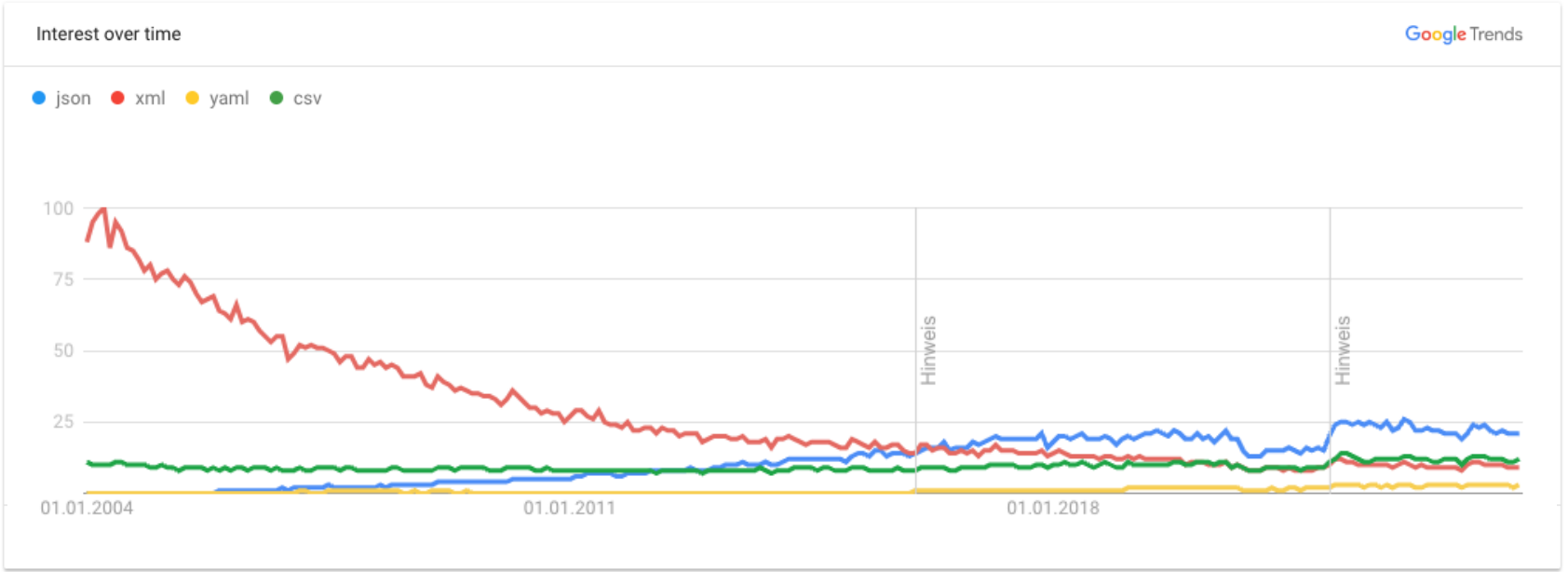
- Lightweight, text-based, human-readable data format for structured data
- Key Features
  - Simplicity
  - Language-Independent
  - Universal
- Syntactically Similar to JavaScript but with stricter rules
- Independent Standard (ECMA-404, RFC 8259)

Popularity Overview



Source: Stackoverflow

Popularity Overview



Source: Google Trends

JSON Elements: Building Blocks of Data

- Key-Value-Pairs: e.g. "name": "John Doe"

Data Type	Description	Example
Strings	Double quotes	"text"
Numbers	Integers or Decimals	123, 1.23
Booleans	True or False	true
Null	Absence of a value	"website": null
Arrays	Ordered list of values	["apple", "banana"]
Objects	Unordered collection of key-value pairs	"Book": { … }

JSON Example


```json
{
  "name": "Jane Smith",
  "title": "Marketing Manager",
  "address": {
    "street": "123 Main Street",
    "city": "Los Angeles",
    "state": "CA",
    "zipcode": 90012
  },
  "phone": "+1 (555) 555-1212",
  "email": "jane.smith@example.com",
  "established": 2005,
  "leave": true,
  "skills": [
    "Marketing Strategy",
    "Social Media",
    "SEO"
  ],
  "logo": "https://www.techinnovations.com/images/logo.png",
  "notes": null
}
```

XML vs. JSON: Mapping Challenges

XML

JSON

```
{
  "person": {
    "name": "Alice",
    "favorite_numbers": [ 5, 12, 23 ]
  }
}
```

XML vs. JSON: Mapping Challenges

XML

```xml
<book author="J.K. Rowling" genre="Fantasy">
  <title>Harry Potter and the
    Sorcerer's Stone</title>
  <year>1997</year>
</book>
```

JSON

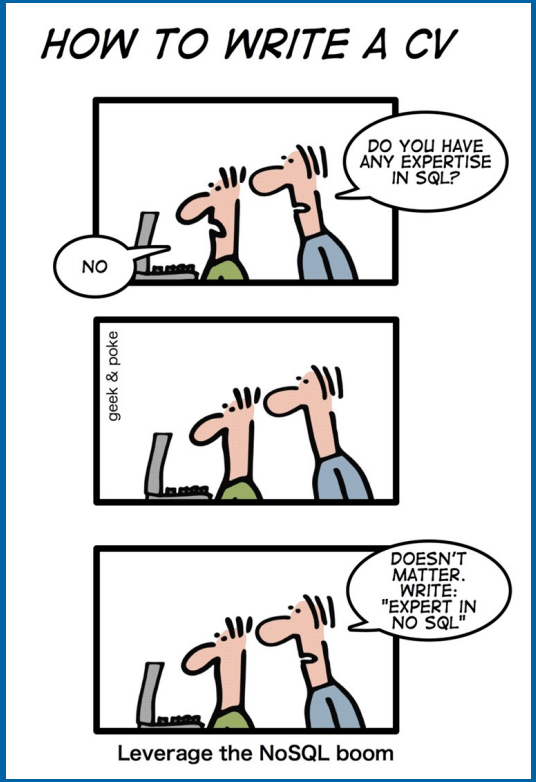
Comparison XML with JSON

Feature	XML (Extensible Markup Language)	JSON (JavaScript Object Notation)
Purpose	General data representation and markup. Focus on documents and structuring content.	Lightweight data exchange. Focus on ease of use and readability.
Syntax	Tags to define elements and attributes. Stricter syntax with opening and closing tags. Supports namespaces.	Key-value pairs enclosed in curly braces, arrays in square brackets. Simpler syntax.
Data Types	Rich variety, within schema (e.g., dates and times).	Limited set of core data types
Hierarchy	Nested elements, order significant	Nested objects, order in object is unordered
Schema Support	Supports schemas for defining strict data structures and validation (XSD, DTD, Relax NG)	JSON Schema exists, but it's a separate standard. Validation isn't built-in.
Verbosity	More verbose due to tags and attributes.	Less verbose, generally smaller files.
Readability	Readable, less so with complexity.	Generally easier to read.
Best Uses	Configuration files, rich text formatting, complex data exchange formats.	APIs, web apps, lightweight data.

NoSQL

Next Generation Database Management Systems mostly addressing some of the points: being **non-relational, distributed, open-source** and **horizontally scalable**. The original intention has been **modern web-scale database management systems**. The movement began early 2009 and is growing rapidly. Often more characteristics apply such as: **schema-free, easy replication support, simple API, eventually consistent / BASE** (not ACID), a huge amount of data and more. So the misleading term "nosql" (the community now translates it mostly with "**not only sql**") should be seen as an alias to something like the definition above.

Definition from https://hostingdata.co.uk/nosql-database/



universität wien

NoSQL Database Systems

- Reasons for the popularity of NoSQL:
  - Development Speed: Often faster development than SQL databases.
  - Data Versatility: Better suited for managing and easily evolving various data structures.
  - Cost-Effectiveness: Handling large data volumes can be more economical than with SQL.
  - Scalability and Uptime: Can better manage high traffic and maintain continuous uptime, unlike SQL.
  - Innovation Support: Supports new application paradigms more effectively.

NoSQL Characteristics

- Schema Flexibility: Evolution over time
- Scalability
- High Performance & Low Latency
- Specialized Data Models
- Large Data Volumes
- BASE vs. ACID

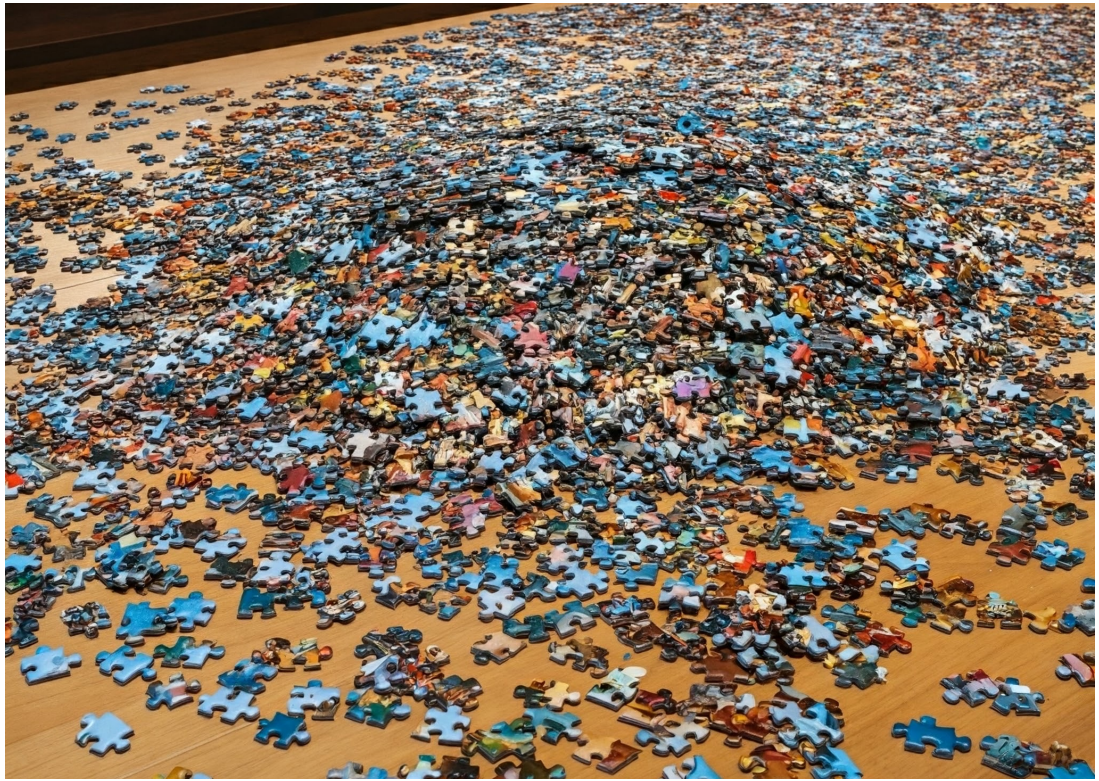
Core NoSQL Systems I

- Key-value Stores:
  - Simplest type: Data is stored as key-value pairs.
  - Ideal for: Caching, session management, storing user preferences
  - Examples: Redis, Memcached, Riak
- Document Databases:
  - Like JSON: Data is stored in document-like structures
  - Ideal for: Content management, semi-structured data, flexible schemas
  - Examples: MongoDB, Couchbase, Amazon DocumentDB, BaseX

Core NoSQL Systems II

- Wide-Column Stores
  - Table-like but flexible: Data is organized into rows and dynamic columns (columns can vary by row).
  - Ideal for: Large-scale analytics, time-series data, event logging
  - Examples: Cassandra, HBase
- Graph Databases
  - Nodes and relationships: Focus on representing relationships between data entities (nodes) and connections between them (edges).
  - Ideal for: Social networks, recommendation engines, fraud detection
  - Examples: Neo4j, JanusGraph
- Multimodel Databases
  - Support multiple data models within a single system
  - Examples: ArangoDB, OrientDB, Cosmos DB

Sharding



Sharding

Horizontally partitioning a large database into smaller, independent pieces called "shards"

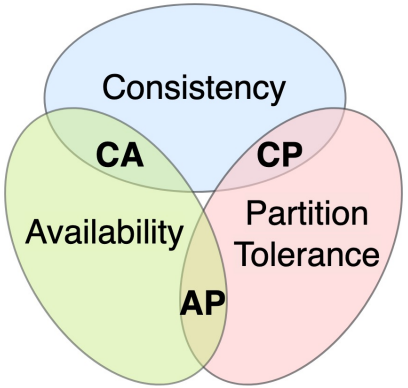
Why Shard?

- Scalability: Handle more data and requests.
- Availability: Improve system resilience.
- Performance: Faster query responses.

Key Elements:

- Sharding Key: Decides data placement.
- Sharding Function: Maps data to shards.
- Query Router: Directs queries to the correct shard(s).

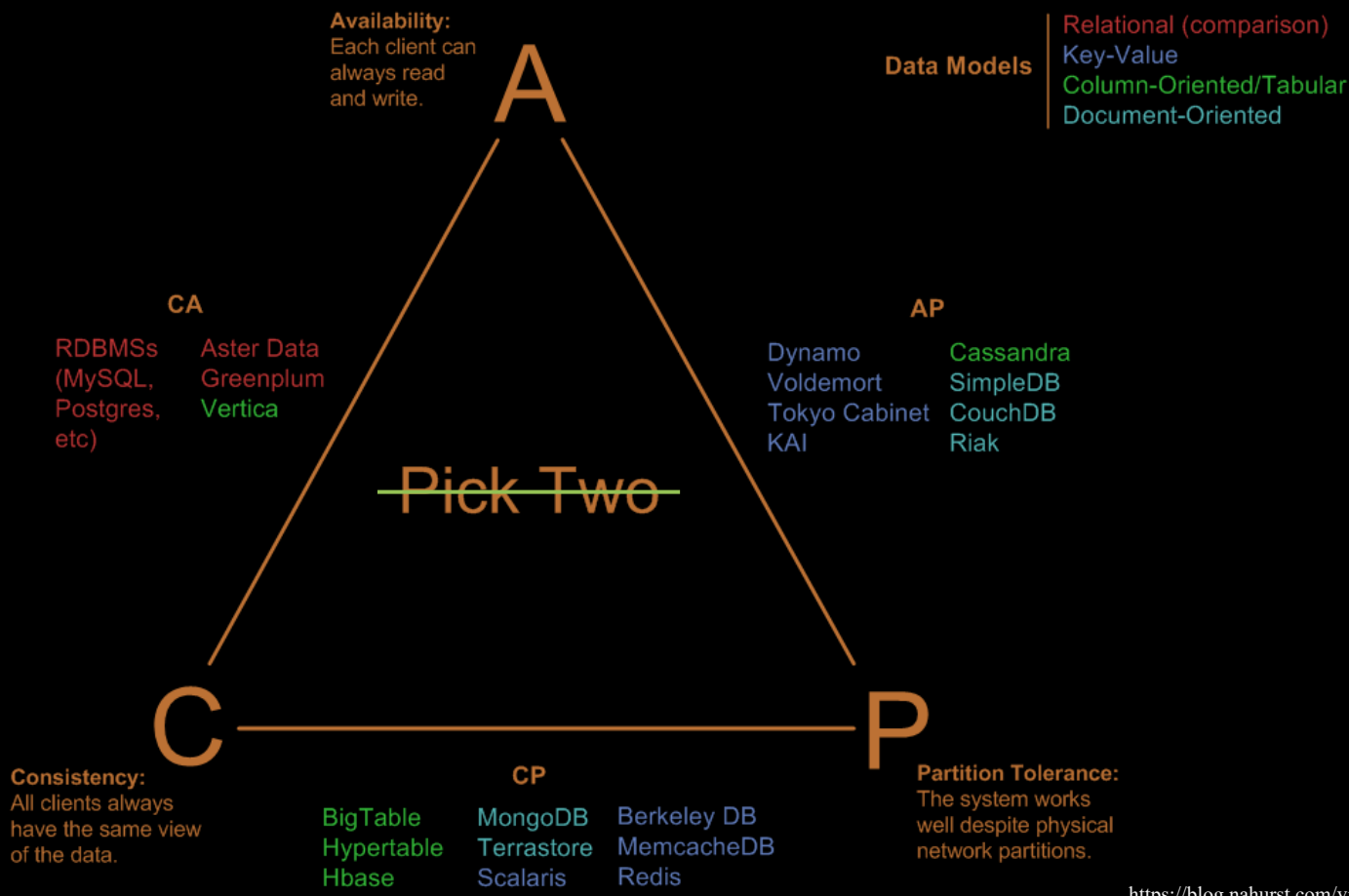
CAP Theorem



CAP theorem. (2024, January 1). In *Wikipedia*. https://en.wikipedia.org/wiki/CAP_theorem

- A distributed system can have at most "two of the three" properties:
  - Consistency
    - Every read receives the most recent write or an error.
  - Availability
    - Every request receives a (non-error) response without guaranteeing it contains the most recent write.
  - Partition
    - The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes.

Visual Guide to NoSQL Systems



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

Basically Available, Soft State, Eventual Consistency

- Promotes availability over consistency - “Optimistic approach”
- Contrary concept to ACID - “Pessimistic” approach
- Abstinence of strong consistency
  - „Soft-state” (State of system may change over time)
  - Database changes between consistent and inconsistent state
  - User has no guarantee to see only one version of data
    - During inconsistency windows, different versions of data possible
- Simplifies the redundancy management of data
  - Less synchronization between replicates necessary
  - Higher availability due to more replicated copies

Advantages NoSQL System to Relational DBMS

- **Flexible Data Models**: Accommodate evolving data structures without complex schema changes. Applications can be designed to work well with less rigid schemas.
- **Horizontal Scalability**: Scale-out cost-effectively by adding commodity hardware.
- **High Performance for Specific Workloads**: Optimized for fast reads and writes, particularly in key-value or denormalized data models.
- **Developer-Friendly**: Many NoSQL systems align with modern application development practices and data formats, reducing reliance on specialized DBAs.
- **Big Data Ready**: Designed to handle massive data volumes
- **Lower Costs**: Often leverages clusters of commodity servers, reducing hardware expenses

Drawbacks of NoSQL System to Relational DBMS

- **Support & Maturity**: Often open-source with varying support levels, still maturing.
- **Administration**: Designed for simpler management, yet skilled oversight is beneficial.
- **Expertise**: Growing developer community, but expertise less widespread than RDBMS.
- **Analytics & BI Focus**: Optimized for web-scale operational needs, analytics features evolving.
- **Standardization & Transactions**: Lacks a single standard, inconsistent support for complex transactions.


redis


cassandra
APACHE HBASE


riak
BASE

JanusGraph

mongoDB®
neo4j

OrientDB

Amazon DocumentDB

Couchbase

Microsoft Azure

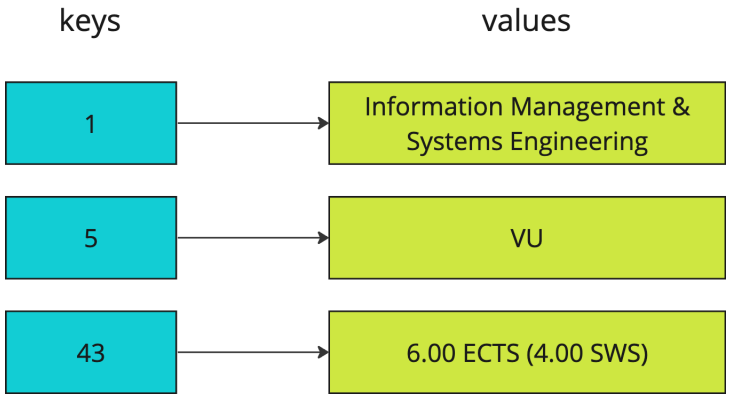
ArangoDB

Key-Value Store

A key-value database is a data storage system that resembles a dictionary or hash table. It stores data as a collection of key-value pairs, where a unique key is used to quickly retrieve the associated data record.

Key-Value Store

- Simple key-value access
- Flexible schema-less design
- Queries are restricted to keys (focused queries)
- Operations usually: put, get, delete
- Advantages of decreased complexity
  - High Scalability
  - Efficient Distribution
  - Fault tolerance
- Foundation of MapReduce

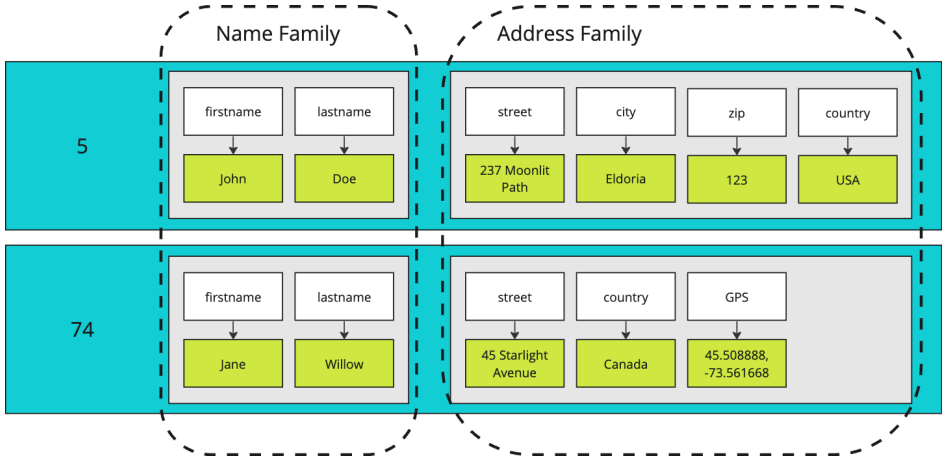


Key Considerations for Key-Value Stores

- Suitable
  - Simple data model
  - High performance for simple retrievals
  - Scalability
- Unsuitable
  - Complex queries
  - Relational Data Management
  - Applications requiring ACID transactions
- Advantages
  - Extremely fast reads/writes
  - Simple data model
  - Highly scalable
- Disadvantages
  - Limited data modeling
  - No native support for complex queries
  - No relationships

Wide-Column Store

- Two-dimensional Key-Value Store.
- Columns not predefined (may vary from row to row)
- Column families



Key Considerations for Wide-Column Stores

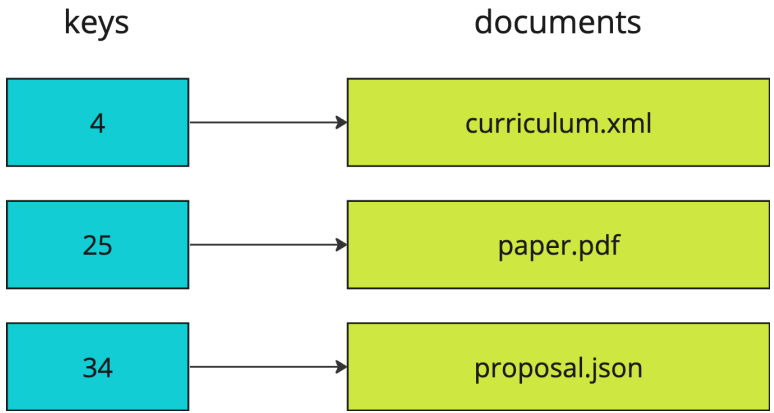
- Suitable
  - Large volumes of data with variable schema
  - Fast reads and writes
  - Scalability
- Unsuitable
  - Complex transactions
  - Strong consistency across multiple operations
  - Complex data relationships
- Advantages
  - Highly flexible handling varied column sets
  - Efficient for analytics
- Disadvantages
  - Complexity in schema managing
  - Less intuitive for relational data model users

Document Stores

Document Stores are specifically designed to handle semi-structured data. They are a popular type of NoSQL database, with XML databases being a specialized subclass for XML document management.

Document Stores

- Collection of documents (eq. rows in RDBMS)
- Documents Formats: JSON, XML, YAML, …
  - Structured set of key/value pairs
- Addressed via a unique key
- Documents are treated as whole (schema-free)
- Access via API or Query Language
- Support: MapReduce
- Not directly Supported: Joins



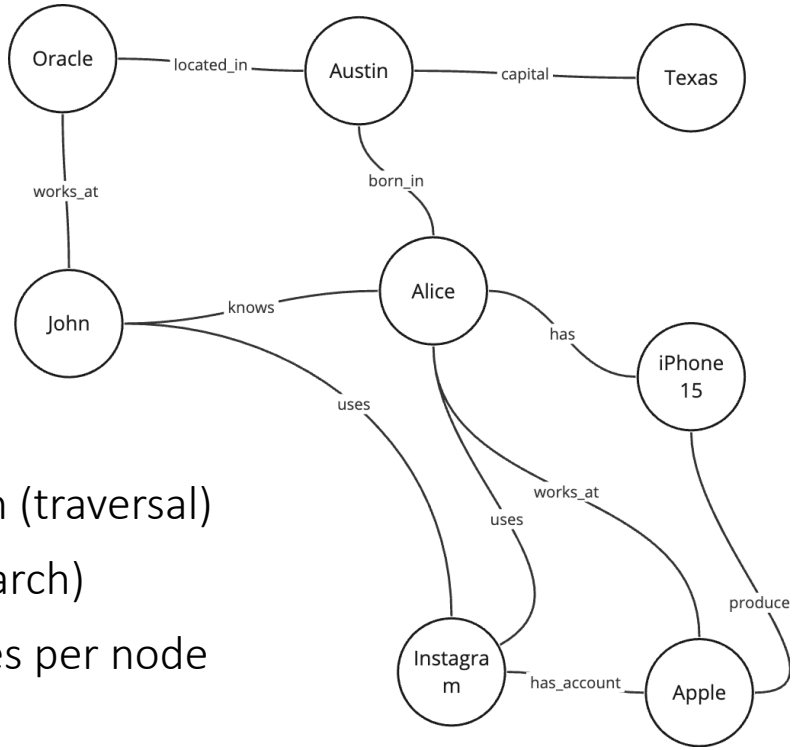
Key Considerations for Document Stores

- Suitable
  - Flexible schema
  - Document encapsulation of data
  - Moderate relationship management
- Unsuitable
  - Highly relational data
  - Applications requiring complex joins and multi-level transactions
- Advantages
  - Flexible data model
  - Rich query capabilities
- Disadvantages
  - Less efficient for complex queries involving multiple document relationships

Graph Databases

A graph database stores data using nodes (data points), edges (relationships between the data), and properties (attributes). This structure prioritizes relationships, allowing for fast queries and intuitive visualization of complex, interconnected data.

Graph Databases



- Graph-oriented
- Entity types are edges, nodes or attributes
- No global key, no joins are necessary
- Data are identified by relative position in the graph (traversal)
- Nodes and edges can be labeled (used later for search)
- No limitation in the number of edges and attributes per node

Key Considerations for Graph Databases

- Suitable
  - Complex relationships
  - Data interconnectivity
  - Deep queries involving multiple hops
  - Dynamic, evolving data
- Unsuitable
  - Simple, non-connected data
  - High throughput operations on massive datasets
  - If applying same operation to multitude of elements
- Advantages
  - Highly optimized for relationships
  - Intuitive modeling
  - Visual representations
- Disadvantages
  - Less performant for non-graph queries
  - Specialized query languages
  - Higher learning curve

Multimodel Database Management Systems

A multi-model database is a database management system that supports multiple data models on a single backend.

Multimodel Database Management System

- Support different data models in the same database
- Different data models can easily be combined in queries and even transactions.
- Common features typically include
  - Data storage, backup, and recovery
  - Querying and indexing mechanisms by a unified query language
  - ACID transactions (mostly in stand-alone mode only)
  - Integration by the support of multiple data models depending on the application
  - Advanced security features

Key Considerations for Multimodel Database Management Systems

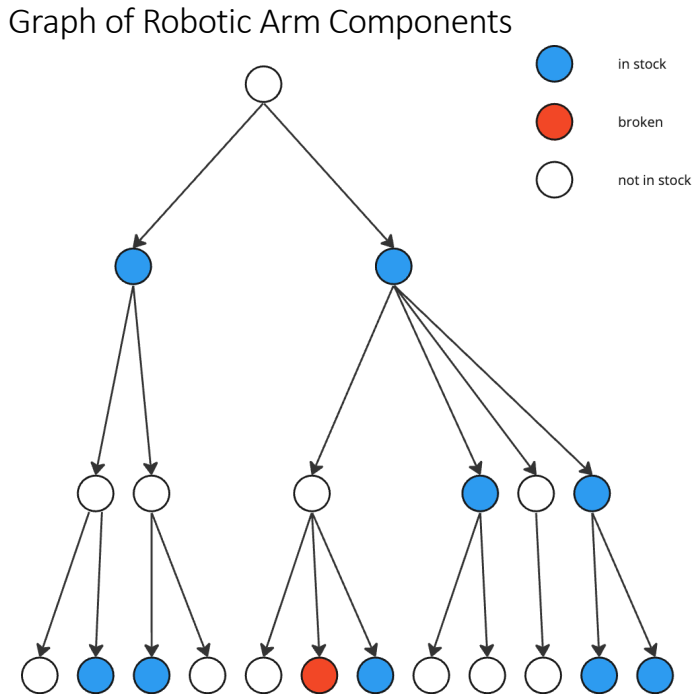
- Suitable
  - Support of multiple data models within a single backend
  - Diverse data types
- Unsuitable
  - Simple applications with a single data model
  - Low complexity environments
- Advantages
  - Master and administer with a single technology
  - Less locked to specific data models and limitations
  - More flexible in requirement changes
- Disadvantages
  - Potentially complex to manage
  - Overhead from support of multiple models can impact performance

Example for Multi-Model Databases

Scenario

- Production Line Robots: Each robot is carefully tracked, with individual parts and maintenance history stored as JSON documents.
- Part Relationships: All robot components are interconnected in a detailed graph. This graph maps everything from tiny screws to complete robotic arms.
- Problem: A critical component on a robot arm breaks.
- **Task:** Identify a compatible replacement component that's in stock.

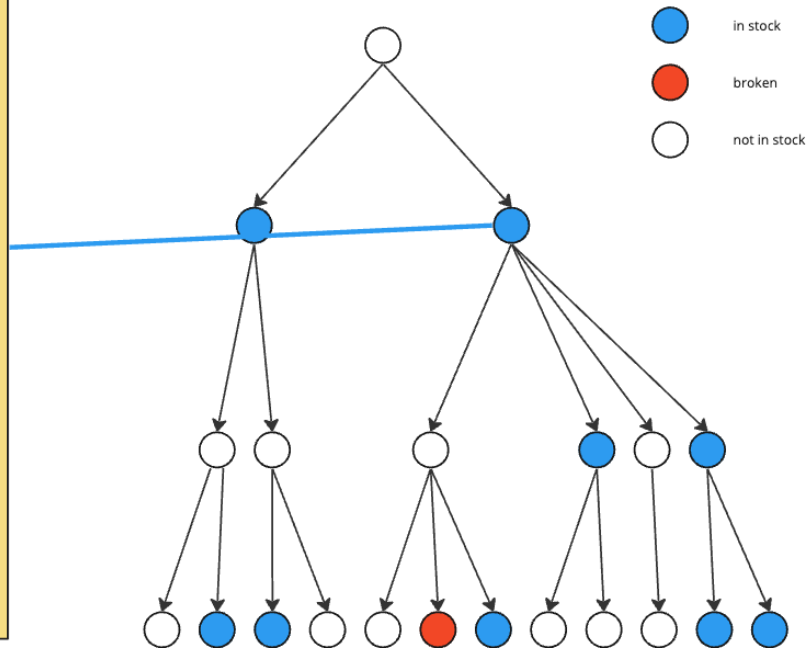
Solution



Solution

```
{
  "component": "Servo Motor",
  "id": "SM-3498",
  "description": "High-torque servo motor",
  "specifications": {
    "torque": "150 Nm",
    "speed": "0.12 sec/60°",
    "voltage": "24V DC",
    "type": "Brushless",
    "weight": "3.5 kg",
    "dimensions": {
      "length": "130 mm",
      "width": "80 mm",
      "height": "70 mm"
    }
  },
  "manufacturer": "RoboTech Dynamics",
  "model": "RTD-24B150",
  "operationalTemperatureRange": {
    "minimum": "-10°C",
    "maximum": "60°C"
  },
}
```

Graph of Robotic Arm Components



MongoDB

MongoDB is a flexible, document-oriented NoSQL database that uses JSON-like structures for data storage. It's known for its scalability and ability to handle diverse data types.

MongoDB

- Schema-less, documented-oriented Open-Source-Database.
- Highly scaleable, highly flexible
- Manages collections of JSON-based documents (+BSON: Binary JSON)
- Editions: Community Server, Enterprise Server, Atlas
- Written in C++
- Developed 2007 by 10gen, now MongoDB Inc.
- Consistency over Availability
- API-support by many programming languages

Drivers

- C
- C++
- C#
- Go
- Java
- Kotlin
- Node.js
- PHP
- Python
- Ruby
- Rust
- Scala
- Swift
- TypeScript
- Elixir
- Mongoose
- Prisma
- R

Advantages

- Schema-less
- Sharding (automatically)
- MapReduce Support
- Simple Replication with automated failover
- Serverless access
- GridFS (Load balancing, data replication features)
- Simple Query Language

Schema-less design

- No predefined structure for documents.
- Flexible data model adapts to changing needs.
- Easy to add new fields or data types.
- Handles unstructured and semi-structured data effectively.
- Faster development cycles with less upfront design.
- Enables agile iterations and rapid prototyping. Reduces schema migration overhead.

Terminology

RDBMS	MongoDB
database	database
table, view	collection
row, tuple, record	document
column, attribute	field
index	index
join	embedded document or reference
primary key	default key _id (delivered by MongoDB)

MongoDB in DB Ranking

#	Name
1	Oracle
2	MySQL
3	Microsoft SQL Server
4	PostgreSQL
5	MongoDB
6	Redis
7	Elasticsearch
8	IBM Db2
9	Snowflake
10	SQLite

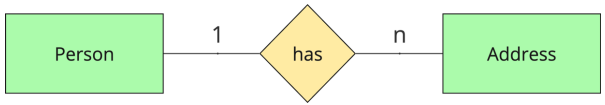
Source: https://db-engines.com/en/ranking
October 2024

Disadvantages

- Only limited transactions
- No joins, but *$lookup* operator
- No referential integrity
- Eventual consistency

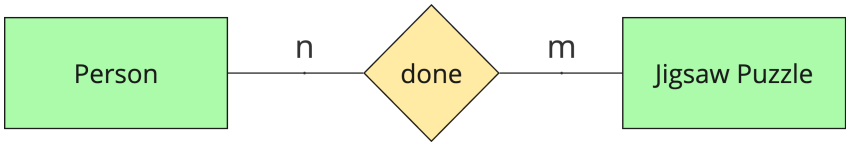
Relationship modeling

- Nesting, Embedding (Denormalization)
  - Possible with 1:1 and 1:n relationships
  - Consider 16 MB document limit




```json
{
    "name": "Alice",
    "level": 30,
    "addresses": [
        {
            "type": "home",
            "street": "123 Main St",
            "city": "Anytown",
            "zip": "12345"
        },
        {
            "type": "work",
            "street": "456 Oak Ave",
            "city": "Businessville",
            "zip": "67890"
        }
    ]
}
```

Relationship modeling



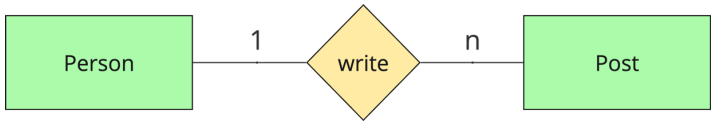
- Referencing
  - Used for n:m relationships and hierarchical relationships
  - Reference _id field to related document
  - Needs an additional query for each reference (aggregate, $lookup)

```
[
    {
        "name": "Alice",
        "level": 30,
        "completed_jigsaw_puzzles": [
            "Van Gogh's Starry Night (1000 pieces)",
            "World Map (500 pieces)"
        ]
    },
    {
        "name": "Bob",
        "level": 24,
        "completed_jigsaw_puzzles": [
            "Mona Lisa (1500 pieces)",
            "Tropical Rainforest (2000 pieces)",
            "Classic Cars (500 pieces)"
        ]
    },
    {
        "name": "Charlie",
        "level": 42,
        "completed_jigsaw_puzzles": [
            "Neuschwanstein Castle (1000 pieces)"
        ]
    }
]
```

```
[
    {
        "_id": ObjectId("6536d7439a7cf70c3d69781a"),
        "name": "Alice",
        "age": 30,
        "completed_puzzle_ids": [
        ObjectId("6536d7439a7cf70c3d69781b"),
        ObjectId("6536d7439a7cf70c3d69781c")
        ]
    },
    {
        "_id": ObjectId("6536d7439a7cf70c3d69781d"),
        "name": "Bob",
        "age": 24,
        "completed_puzzle_ids": [
        ObjectId("6536d7439a7cf70c3d69781e"),
        ObjectId("6536d7439a7cf70c3d69781f"),
        ObjectId("6536d7439a7cf70c3d697820")
        ]
    },
    {
        "_id": ObjectId("6536d7439a7cf70c3d697821"),
        "name": "Charlie",
        "age": 42,
        "completed_puzzle_ids": [
        ObjectId("6536d7439a7cf70c3d697822")
        ]
    }
]
```

Relationship modeling - Discussion

- Social Media Platform
- How to store a post?
- Should you embed the person's name in a post?



Denormalization

- Combination of two relations into one new relation
- Increases data redundancy to avoid expensive lookups (controlled redundancy)
- Improves read performance by reducing the need for joins.
- Reduces query complexity by consolidating related data.
- Simplifies data retrieval for frequently accessed information.
- Trades storage space for faster query execution.
- Requires careful management to maintain data consistency.
- Suitable for read-heavy workloads with infrequent updates.

Embed vs. Reference

Considerations for Embedding Information

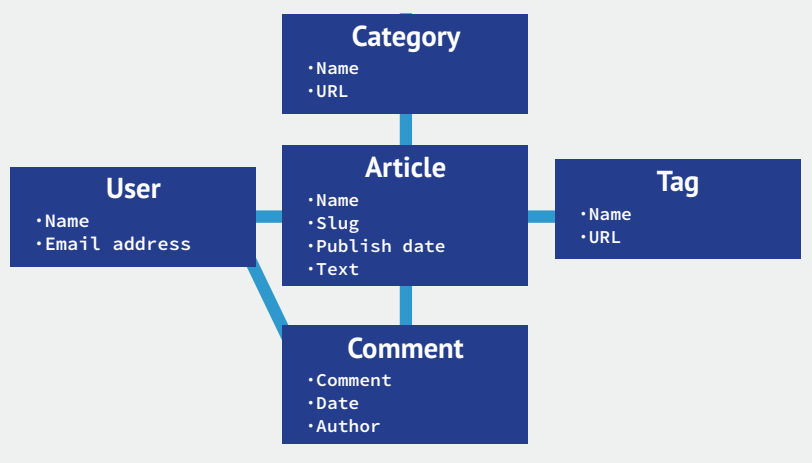
- Data with tight relationships
- One-to-few relationships
- Data does not change frequently
- Atomic updates

Considerations for Using References

- One-to-many relationships
- Data with high update frequency
- Data needs to be accessed independently
- Normalization

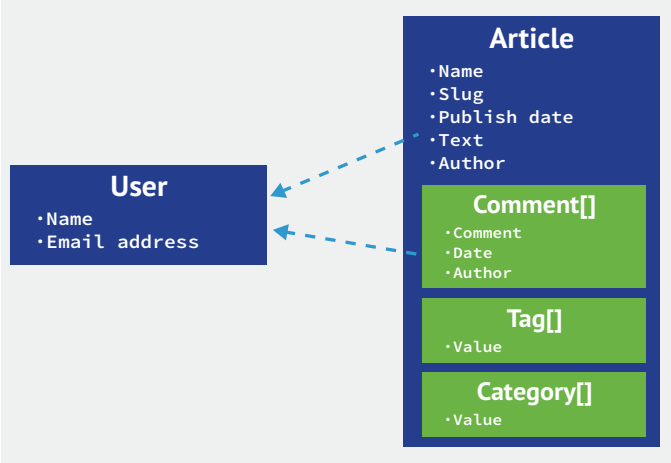
Migration Example

RDBMS: JOIN 5 tables



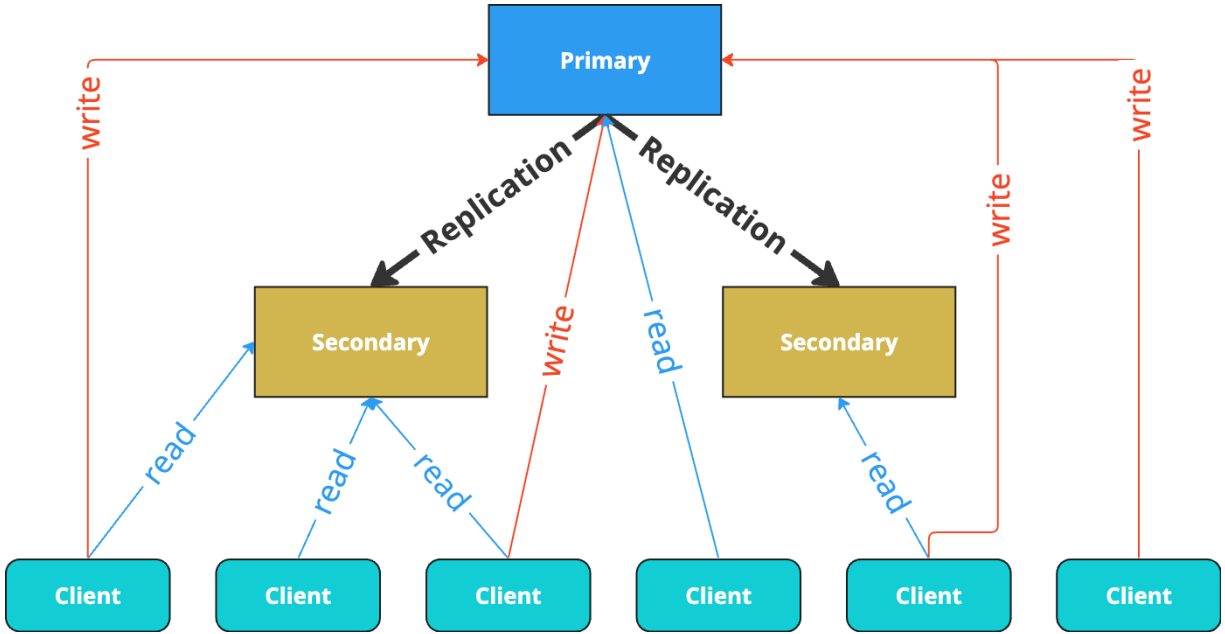
Five tables, one Blog entry is built by a Join of 5 tables

MongoDB: Nested documents

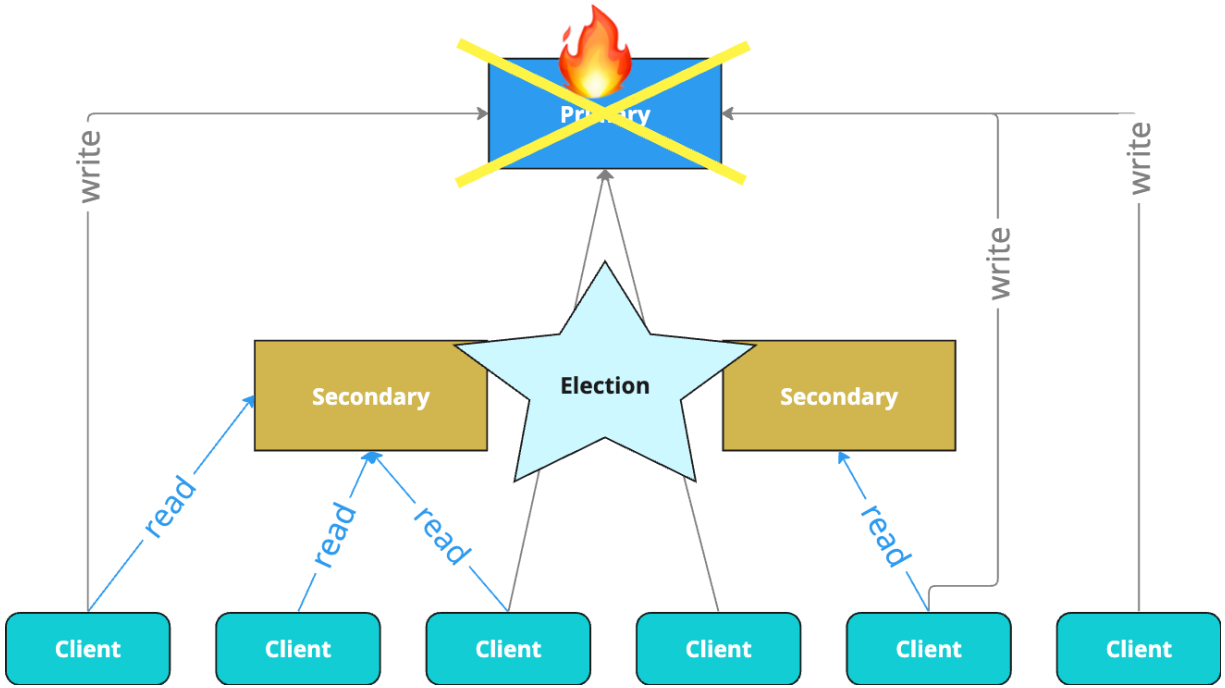


- Single document with a link to a user document, which contains the Blog entry and the comment of user
- Thus: Pre-JOINing to merge 5 RDBMS tables to 2 BSON documents

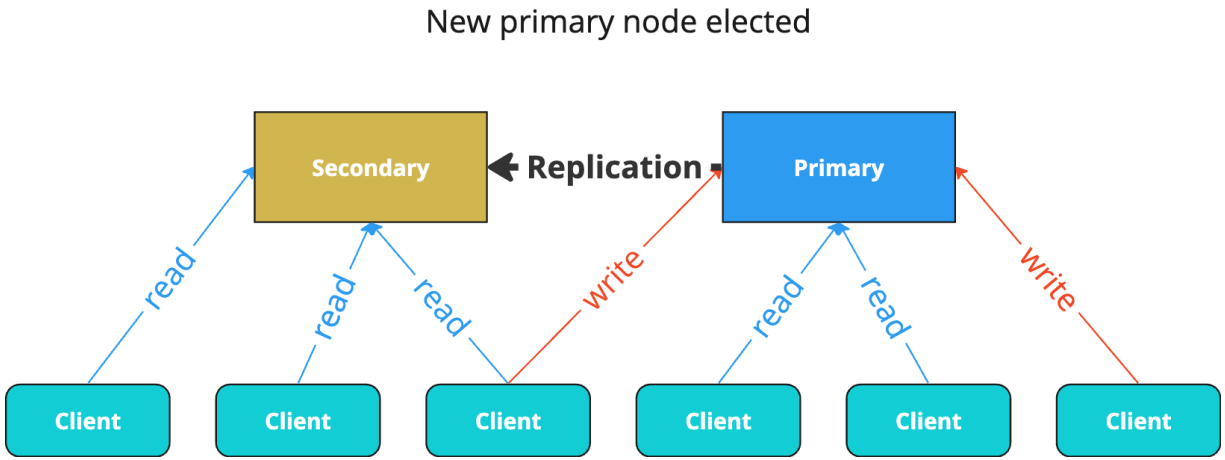
Primary-Secondary Replication



Primary-Secondary Replication – Primary down



Primary-Secondary Replication – New Primary



Literature

- ECMA-404 (JSON): https://ecma-international.org/publications-and-standards/standards/ecma-404/
- Gilbert, Seth, and Nancy Lynch. "Perspectives on the CAP Theorem." Computer 45.2 (2012): 30-36. Link
- Brewer, Eric. "CAP twelve years later: How the" rules" have changed." Computer 45.2 (2012): 23-29. Link
- Ploetz, Aaron, et al. Seven NoSQL Databases in a Week: Get up and running with the fundamentals and functionalities of seven of the most popular NoSQL databases. Packt Publishing, 2018. Link
- Giamas, Alex. Mastering MongoDB 6. x: Expert techniques to run high-volume and fault-tolerant database solutions using MongoDB 6. x. Packt Publishing Ltd, 2022. Link