# NoSQL databases

Dmitry A. Duev

# NoSQL databases

**NoSQL: not only SQL / non SQL**

- What drove the development of NoSQL?
  - Pros and cons of traditional SQL systems
- What are the different flavors of NoSQL?
  - Wide column-based
  - Key-value
  - Document-based
  - Graph-based

# Pros and cons of traditional RDBMS

**+**

- Have been around for quite a while
- SQL: mature and powerful
- Transactions + ACID-compliance built-in
- Data normalization + joins
- Schema: good when data can be represented in appropriate way
- Many open-source solutions

**-**

- Vertical scale-out | scaling issues
- Joins + transactions across multiple dbs quickly become costly for complicated objects and big data -> performance and availability are affected
- Object-relational impedance mismatch
- Schema: bad for unstructured/evolving data

# NoSQL: motivation and promise

The design and development of NoSQL databases has been largely driven by the RDBMS cons from the previous slide.

- Deliver performance for big, potentially unstructured or evolving data that may come in in real time
    - Simple design
    - Horizontal scaling
- NoSQL ~= not only SQL, i.e. some systems support SQL-like query languages

# NoSQL: fulfilling the promise

Always comes with a cost!

- Most such systems lack ACID transactions and offer BASE (Basic Availability, Soft state and Eventual consistency) instead
  - See ACID vs BASE
  - Some systems exhibit potential lost writes and other forms of data loss
  - A notable exception is MongoDB
- No schema means data integrity might become an issue
  - Some systems do allow defining schemas and perform validation/enforcement (e.g. MongoDB)
- Many query languages vs single SQL (albeit with different flavors)

# NoSQL: fulfilling the promise

In practice, you almost always still need to deal with relational data! There are three main techniques to do that:

- Nesting/embedding data
  - Store all data needed for a specific task in one place (e.g. in a single document)
- Linking + Multiple queries
  - Store a foreign key and fetch data in multiple queries. Since single queries in NoSQL are often more performant than in SQL, may be ok
- Caching and replication
  - Instead of storing a foreign key, store the actual values

Data schema modelling must be done very differently from RDBMS. A simple translation would often not work.

# Main types of NoSQL databases

- Key-value store
  - Uses maps/dictionaries/associated lists/hash tables with corresponding operation complexities
  - Examples: Redis, ArangoDB, ZooKeeper, Couchbase, Cassandra, Amazon DynamoDB
- Wide column store
  - Essentially, a two-dimensional semi-structured key-value store
  - Examples: Cassandra, HBase
- Document store
  - Semi-structured; data are encapsulated in some standard form (XML, JSON, BSON) in “documents” with unique keys/identifiers
  - Often uses B-trees with corresponding operation complexities
  - Examples: Couchbase, MongoDB, Amazon DynamoDB
- Graph store
  - Uses graphs to represent data + relationship between them (the latter can be queried, too)
  - Examples: Neo4J, ArangoDB

# Performance

| Data model | Performance | Scalability | Flexibility | Complexity | Functionality |
|---|---|---|---|---|---|
| Key–value store | high | high | high | none | variable |
| Column-oriented store | high | high | moderate | low | minimal |
| Document-oriented store | high | variable (high) | high | low | variable |
| Graph database | variable | variable | high | high | graph theory |
| Relational database | variable | variable | low | moderate | relational algebra |

Ben Scofield