Contents

Limitations of Relational Databases

- Impedance mismatch
- Application and integrations
- Scale up vs scale out

Emergenece of NoSQL

- First appeared in the 90s are the name of an open-source relational database introduced by Carlo Strozzi
- Two companies played a key role
  - 2004: Google started the project BigTable and published a paper in 2006
  - 2007: Amazon published the research paper on Amazon Dyanamo
- John Oskarsson decides to find out more about the new tendency to alternative data storage
  - 2009: Meetup in San Francisco
  - ## NoSQL is used as simple hashtag to refer to the event

NoSQL - No Definition

- Non Relational
- No SQL as query language
- Schema less
- Usually - not always - Open Source Project
- They are distributed
- RDBMS use AVID transactions, NoSQL don't

Distribution Models

Sharding

- Different data on different nodes
- Each serber acts as a single source for the subset of data it is responsible for
- Ideal setting: one user talks with one server
  - Data accessed together are stored together
  - Example: access based on physical location, place datat to the nearest server
- Many NoSQL databases offer auto sharding
- Scales read and write on the different nodes of the same cluster
- No resilience ifused alone: node failure ⇒ data unavailablity

Replication

- The same data is replicated and copied over multiple nodes
- Matser-slave: one node is the primary responsible for processing the update to data while the other are secondaries used for read operations
  - Scaling by adding slaves
  - Processing incoming data limited by master
  - Read resilience
  - Inconsistency problem
- The same data is replicated and copied over multiple nodes
- Peer-to-peer: all replicas have equal weight and can accept writing
  - Scaling by adding nodes
  - Node failure without losing write capability
  - Inconsistency problem

Combined Approaches

- Master-slave & sharding
  - Multiple master, each data has a single master
- P2P & sharding (common for column-family databases)
  - Replication of the shard

CAP Theorem

- Consistency
  - All nodes see the same data at the same time
- Availability
  - A guarantee that every request receives a response about whether it succeeded or failed
- Partition tolerance
  - The system continues to operate despite arbitrary partitioning due to network failures

Relational bs NoSQL

- Acid
  - **A**tomic
  - **C**onsistent
  - **I**solated
  - **D**urable
- Base
  - **B**asic **A**vailability
  - **S**oft-state
  - **E**ventual consistency

Four Common Types of NoSQL

- Aggregate
  - Key-Value Stores
  - Document Stores
  - Column Stores
- Graph Stores
- Note
  - Lots of hybrids
  - Lots of NewSQL vendors
  - Some niche Graph stores

Key-Value Stores

- Maps keys to values
- Values treated as a blob
  - They can be complex compound objects
- Single index
- Consistency applicable for operations on a single key
- Very fact and scalable
- Inefficient to do aggregate queries, "all the carts work $100 or more" or to represent relationships between data
- Great for shopping carts, user profiles and preferences, storing session information

Document Databases

- A document is like a hash, with one id and many values
- Store Javascript Documents
  - JSON = JavaScript Object Notation
  - An associative array
  - Key value pairs
  - Values can be documents or arrays
  - Arrays can contain documents
- Data is implcitly denormalised

Column Stores

- Store data as columns rather than rows
  - Columns organised in column family
  - Each column belongs to a single column family
  - Column acts as a unit for access
  - Particular column family will be accessed together
- Efficient to do column ordered operations
- Not so great as row based queries
- Adding columns is quite inexpensive and is done on a row-by-row basis
- Each row can have a different set of columns, or none at all, allowing tables to remain sparse without incurring a storage cost for null values

Good for

- Relational
  - Queries hat return small subsets of rows
  - Queries that use a large subset of row data
- Column
  - Queries that require just a column of data
  - Queries that require a small subset of row data

Graph Stores

- Data model composed by nodes connected by edges
  - Nodes represent entities
  - Edges represent the relationships between entities

  - Nodes and edges can have properties
- Querying a graph database means traversing the graph by following the relationships
- Pros
  - Representing objects of the real world that are highly inter-connected
  - Traversing the relationships in these data models is cheap

Graph Stores vs Relational Databases

- Relational databases are not ideally suited to representing relationships
- Relationships implemented through foreign keys
- Expensive joins required to navigate relationships
  - Poor performance for highly connected data models