NoSQL

Carla Teixeira Lopes

Bases de Dados
Mestrado Integrado em Engenharia Informática e Computação, FEUP

Lemahieu, Broucke and Baesens slides

Agenda

The NoSQL movement

Key-Value stores

Tuple and Document stores

Column-oriented databases

Graph based databases

The NoSQL movement

RDBMSs put a lot of emphasis on keeping data consistent.

- Entire database is consistent at all times (ACID)

Focus on consistency may hamper flexibility and scalability

As the data volumes or number of parallel transactions increase, capacity can be increased by

- Vertical scaling: extending storage capacity and/or CPU power of the database server
- Horizontal scaling: multiple DBMS servers being arranged in a cluster

The NoSQL movement

RDBMSs are not good at extensive horizontal scaling

- Coordination overhead because of focus on consistency
- Rigid database schemas

Other types of DBMSs needed for situations with massive volumes, flexible data structures and where scalability and availability are more important -> NoSQL databases

The NoSQL movement

NoSQL databases

- Describes databases that store and manipulate data in other formats than tabular relations, i.e. non-relational databases (NoREL)

NoSQL databases aim at near linear horizontal scalability, by distributing data over a cluster of database nodes for the sake of performance as well as availability

Eventual consistency: the data (and its replicas) will become consistent at some point in time after each transaction

The NoSQL movement

	Relational databases	NoSQL databases
Data paradigm	Relational tables	Key-value (tuple) based Document based Column based Graph based XML, object based Others: time series, probabilistic, etc.
Distribution	Single-node and distributed	Mainly distributed
Scalability	Vertical scaling, harder to scale horizontally	Easy to scale horizontally, easy data replication
Openness	Closed and open source	Mainly open source
Schema role	Schema-driven	Mainly schema-free or flexible schema
Query language	SQL as query language	No or simple querying facilities, or special-purpose languages
Transaction mechanism	ACID: Atomicity, Consistency, Isolation, Durability	BASE: Basically available, Soft state, Eventual consistency
Feature set	Many features (triggers, views, stored procedures, etc.)	Simple API
Data volume	Capable of handling normal-sized data sets	Capable of handling huge amounts of data and/or very high frequencies of read/write requests

Agenda

~~The NoSQL movement~~

Key-Value stores

Tuple and Document stores

Column-oriented databases

Graph based databases

Key-value Stores

Key-value based database stores data as (key, value) pairs

- Keys are unique
- Hash map, or hash table or dictionary

Key-value Stores

```
import java.util.HashMap;
import java.util.Map;
public class KeyValueStoreExample {
         public static void main(String... args) {
                  // Keep track of age based on name
                  Map<String, Integer> age_by_name = new HashMap<>();

                  // Store some entries
                   age_by_name.put("wilfried", 34);
                   age_by_name.put("seppe", 30);
                   age_by_name.put("bart", 46);
                   age_by_name.put("jeanne", 19);

                  // Get an entry
                  int age_of_wilfried = age_by_name.get("wilfried");
                  System.out.println("Wilfried's age: " + age_of_wilfried);

                  // Keys are unique
                  age_by_name.put("seppe", 50); // Overrides previous entry
         }
}
```

Key-value Stores

Keys (e.g., “bart”, “seppe”) are hashed by means of a so-called hash function

- A hash function takes an arbitrary value of arbitrary size and maps it to a key with a fixed size, which is called the hash value.
- Each hash can be mapped to a space in computer memory

Key
wilfried
seppe
bart
jeanne

Hash	Key
01	(wilfried, 34)
03	(seppe, 30)
07	(bart, 46)
08	(jeanne, 19)

Key-value Stores

NoSQL databases are built with horizontal scalability support in mind

Distribute hash table over different locations

Assume we need to spread our hashes over three servers

- Hash every key (“wilfried”, “seppe”) to a server identifier
- index(hash) = mod(hash, nrServers) + 1

Sharding

Key
wilfried
seppe
bart
jeanne

Hash	Key (= mod 3 + 1)
01	2
03	1
07	2
08	3

Server #1	Entries
03	(seppe,30)

Server #3	Entries
08	(jeanne,19)

Server #2	Entries
01	(wilfried,34)
07	(bart, 46)

Key-value Stores

Example: Memcached

- Implements a distributed memory-driven hash table (i.e. a key-value store), which is put in front of a traditional database to speed up queries by caching recently accessed objects in RAM
- Caching solution

Key-value Stores

Request Coordination

Consistent Hashing

Replication and Redundancy

Eventual Consistency

Stabilization

Integrity Constraints and Querying

Request Coordination

In many NoSQL implementations (e.g. Cassandra, Google's BigTable, Amazon's DynamoDB) all nodes implement the same functionality and are all able to perform the role of request coordinator

Need for membership protocol

- Dissemination
  - Based on periodic, pairwise communication
- Failure detection

Consistent Hashing

Consistent hashing schemes are often used, which avoid having to remap each key to a new node when nodes are added or removed

Suppose we have a situation where 10 keys are distributed over 3 servers ($n = 3$) with the following hash function

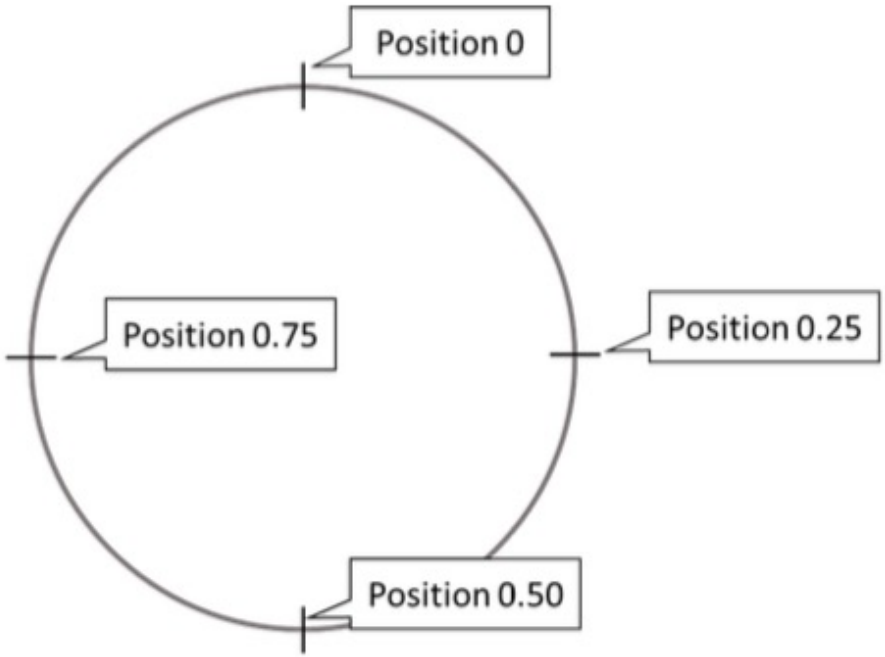
- *h(key)* = *key* modulo *n*

Consistent Hashing

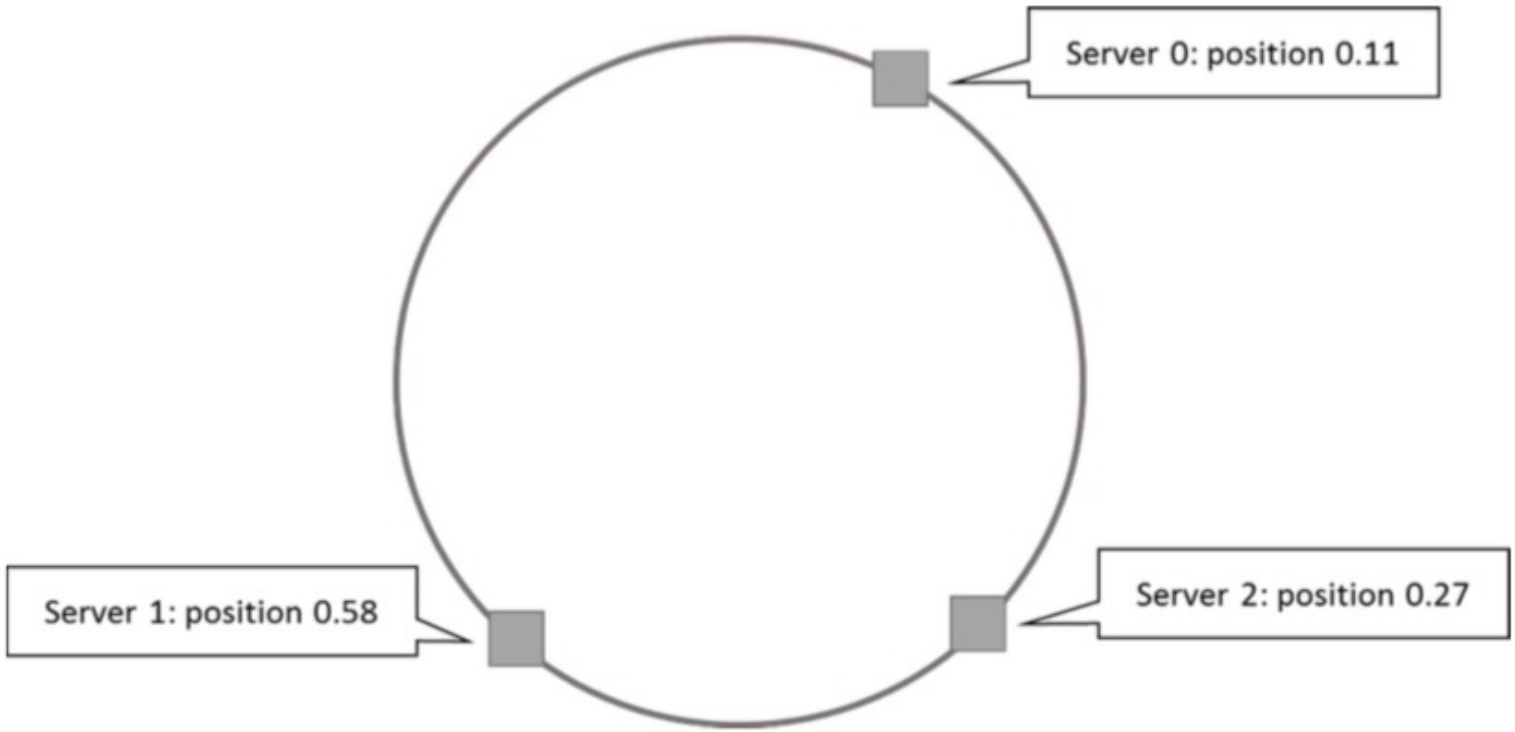
	n		
key	**3**	**2**	**4**
0	0	0	0
1	1	1	1
2	2	0	2
3	0	1	3
4	1	0	0
5	2	1	1
6	0	0	2
7	1	1	3
8	2	0	0
9	0	1	1

Consistent Hashing

At the core of a consistent hashing setup is a so called “ring”-topology, which is basically a representation of the number range [0,1[:

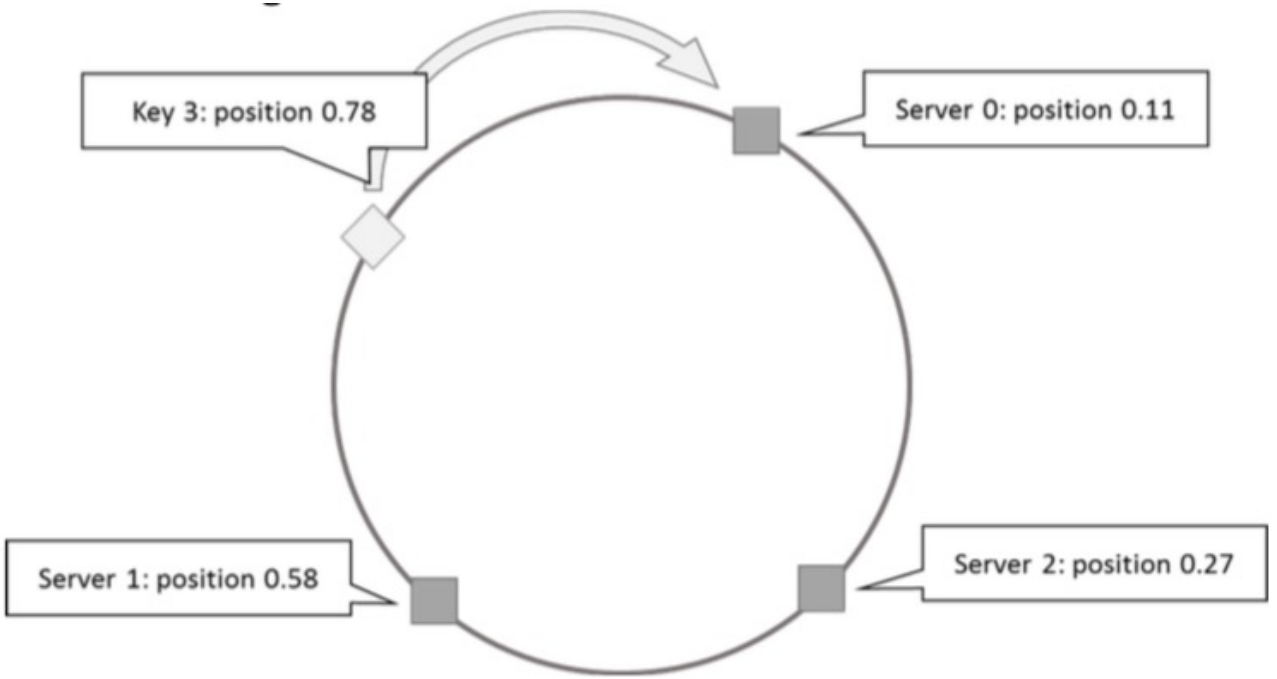


Consistent Hashing



Consistent Hashing

Hash each key to a position on the ring, and store the actual key-value pair on the first server that appears clockwise of the hashed point on the ring

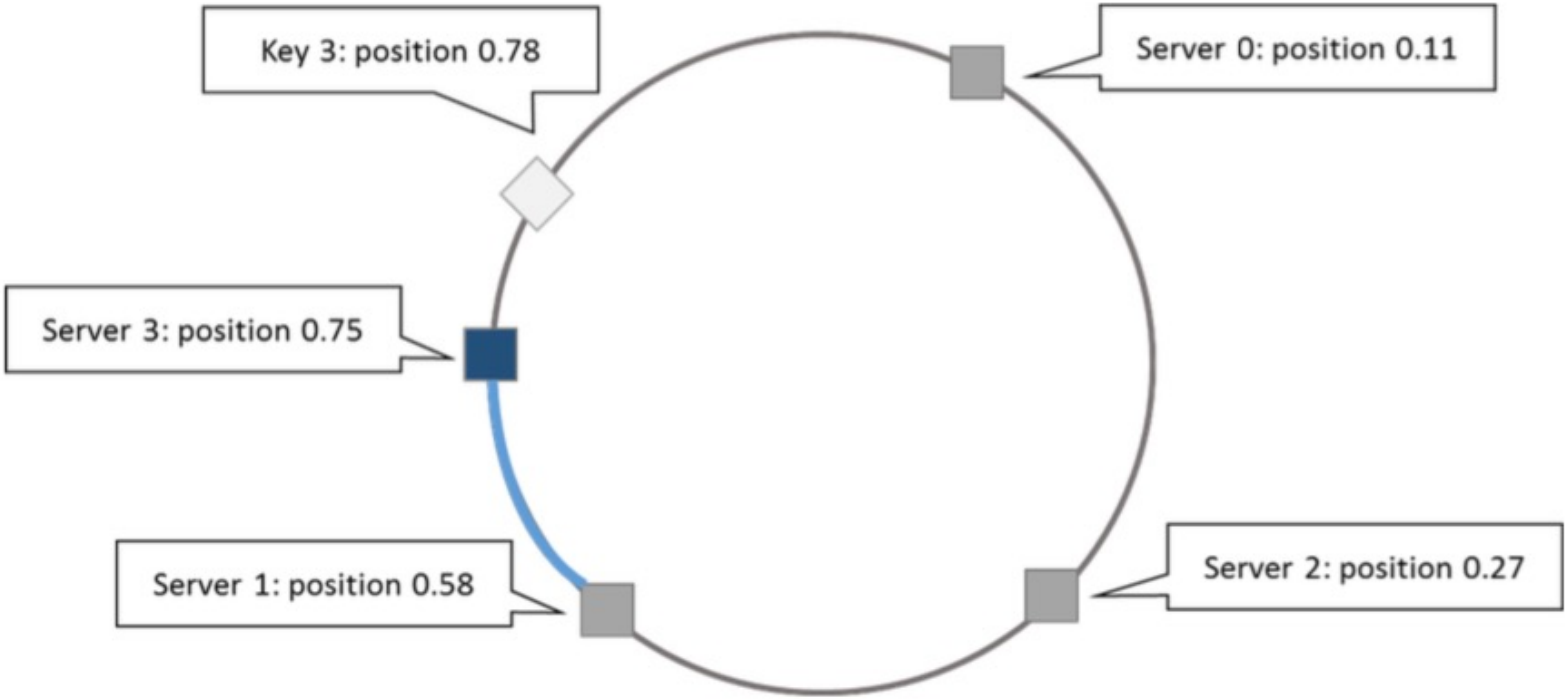


Consistent Hashing

Because of the uniformity property of a “good” hash function, roughly 1/*n* of key-value pairs will end up being stored on each server

Most of the key-value pairs will remain unaffected in case a machine is added or removed

Consistent Hashing



Replication and Redundancy

Problems with consistent hashing:

- If 2 servers end up being mapped close to one another, one of these nodes will end up with few keys to store
- In case a server is added, all of the keys moved to this new node originate from just one other server

Instead of mapping a server *s* to a single point on our ring, we map it multiple positions, called **replicas**

For each physical server *s*, we hence end up with *r* (the number of replicas) points on the ring

Note: each of the replicas still represents the same physical instance (<-> redundancy)

- Virtual nodes

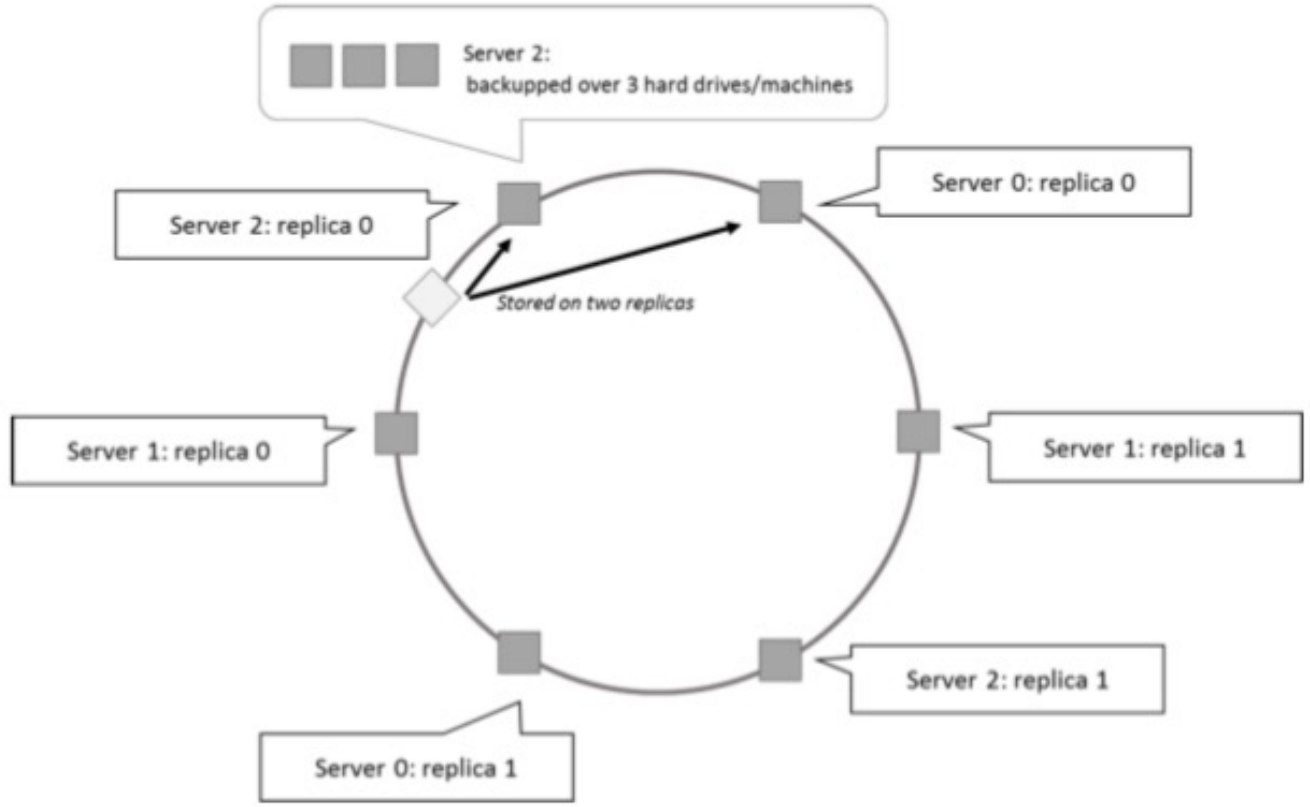
Replication and Redundancy

To handle data replication or redundancy, many vendors extend the consistent hashing mechanism so that key-value pairs are duplicated across multiple nodes

- E.g., by storing the key-value pair on two or more nodes clockwise from the key's position on the ring

Replication and Redundancy

It is also possible to set up a full redundancy scheme where each node itself corresponds to multiple physical machines each storing a fully redundant copy of the data



Eventual Consistency

Membership protocol does not guarantee that every node is aware of every other node *at all times*

- It will reach a consistent state over time

State of the network might not be perfectly consistent at any moment in time, though will become eventually consistent at a future point in time

Many NoSQL databases guarantee so called **eventual consistency**

CAP theorem

A distributed computer system cannot guarantee the following properties at the same time:

- Consistency
  - all nodes see the same data at the same time
- Availability
  - guarantees that every request receives a response indicating a success or failure result
- Partition tolerance
  - the system continues to work even if nodes go down or are added.

BASE

Most NoSQL databases sacrifice the consistency part of CAP in their setup, instead striving for eventual consistency

The full BASE acronym stands for:

- Basically available: NoSQL databases adhere to the availability guarantee of the CAP theorem
- Soft state: the system can change over time, even without receiving input
- Eventual consistency: the system will become consistent over time

Stabilization

The operation which repartitions hashes over nodes in case nodes are added or removed is called **stabilization**

If a consistent hashing scheme being applied, the number of fluctuations in the hash-node mappings will be minimized.

Integrity Constraints and Querying

Key value stores represent a very diverse gamut of systems

Full blown DBMSs versus caches

Only limited query facilities are offered

- E.g. put and set

Limited to no means to enforce structural constraints

- DBMS remains agnostic to the internal structure

No relationships, referential integrity constraints or database schema, can be defined

Agenda

~~The NoSQL movement~~

~~Key-Value stores~~

Tuple and Document stores

Column-oriented databases

Graph based databases

Tuple and Document Stores

A tuple store is similar to a key-value store, with the difference that it does not store pairwise combinations of a key and a value, but instead stores a unique key together with a vector of data

Example:

- marc -> ("Marc", "McLast Name", 25, "Germany")

No requirement to have the same length or semantic ordering (schema-less!)

Tuple and Document Stores

Various NoSQL implementations do, however, permit organizing entries in semantical groups, (aka collections or tables)

Examples:

- Person:marc -> ("Marc", "McLast Name", 25, "Germany")
- Person:harry -> ("Harry", "Smith", 29, "Belgium")

Tuple and Document Stores

Document stores store a collection of attributes that are labeled and unordered, representing items that are semi-structured

Example:

```
{
        Title = “Harry Potter”
        ISBN = “111-11111111”
        Authors = [ “J.K. Rowling” ]
        Price = 32
        Dimensions = "8.5 x 11.0 x 0.5"
        PageCount  = 234
        Genre      = "Fantasy"
}
```

Tuple and Document Stores

Most modern NoSQL databases choose to represent documents using JSON

```
{
    "title": "Harry Potter",
    "authors": ["J.K. Rowling", "R.J. Kowling"],
    "price": 32.00,
    "genres": ["fantasy"],
    "dimensions": {
            "width": 8.5,
            "height": 11.0,
            "depth": 0.5
    },
    "pages": 234,
    "in_publication": true,
    "subtitle": null
}
```

Agenda

~~The NoSQL movement~~

~~Key-Value stores~~

~~Tuple and Document stores~~

Column-oriented databases

Graph based databases

Column-oriented Databases

A column-oriented DBMS is a database management system that stores data as sections of columns of data

Useful if

- aggregates are regularly computed over large numbers of similar data items
- data is sparse, i.e. columns with many null values

Can also be an RDBMS, key-value or document store

Column-oriented Databases

Example

id	Genre	Title	Price	Audiobook price
1	fantasy	My first book	20	30
2	education	Beginners guide	10	null
3	education	SQL strikes back	40	null
4	fantasy	The rise of SQL	10	null

Row based databases are not efficient at performing operations that apply to the entire data set

- Need indexes which add overhead

Column-oriented Databases

In a column-oriented database, all values of a column are placed together on disk

Genre: fantasy:1,4 education:2,3

Title: My first book:1 Beginners guide:2 SQL strikes back:3 The rise of SQL:4

Price: 20:1 10:2,4 40:3

Audiobook price: 30:1

A column matches the structure of a normal index in a row-based system

Operations such as: find all records with price equal to 10 can now be executed directly

Null values do not take up storage space anymore

Column-oriented Databases

Disadvantages

- Retrieving all attributes pertaining to a single entity becomes less efficient
- Join operations will be slowed down

Examples

- Google BigTable, Cassandra, HBase, and Parquet

Agenda

~~The NoSQL movement~~

~~Key-Value stores~~

~~Tuple and Document stores~~

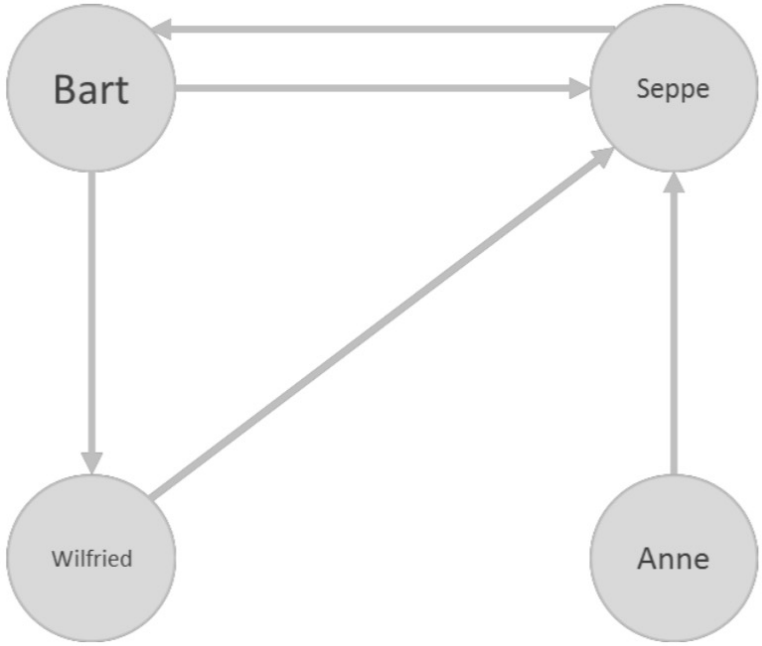
~~Column-oriented databases~~

Graph based databases

Graph based Databases

Graph databases apply graph theory to the storage of information of records

Graphs consist of nodes and edges



Graph based Databases

One-to-one, one-to-many, and many-to-many structures can easily be modeled in a graph

Consider N-M relationship between books and authors

RDBMS needs 3 tables: Book, Author and Books_Authors

SQL query to return all book titles for books written by a particular author would look like follows

```
SELECT title

FROM books, authors, books_authors

WHERE author.id = books_authors.author_id

AND books.id = books_authors.book_id AND author.name = "Bart
Baesens"
```

Graph based Databases

In a graph database (using **Cypher query language** from Neo4j)


```
MATCH (b:Book)<-[:WRITTEN_BY]-(a:Author)

WHERE a.name = "Bart Baesens"

RETURN b.title
```

Graph based Databases

A graph database is a hyper-relational database, where JOIN tables are replaced by more interesting and semantically meaningful relationships that can be navigated and/or queried using graph traversal based on graph pattern matching.

Graph databases

Location-based services

Recommender systems

Social media (e.g. Twitter and FlockDB)

Knowledge based systems

NoSQL DBMS

	RDBMS	**NoSQL**
Relational	Yes	No
SQL	Yes	No
Column stores	No	Yes
Scalability	Limited	Yes
Eventually consistent	Yes	Yes
BASE	No	Yes
Big volumes of data	No	Yes
Schema-less	No	Yes

Kahoot time!

Any doubts?