CMSC 206: Database Management Systems

Database Trends

Thomas Laurent
Hilmi Egemen Ciritoglu

28/11/2020

University of the Philippines Open University

Table of contents

Foreword

Foreword

This week, I let my colleague Egemen guide you through some of the trends in the world of databases. He is an expert of distributed data management and big data working with Hadoop.

Introduction

Introduction

RDBMS (e.g., Oracle Database, MySQL, PostgreSQL) are convenient to store data as they guaranty the ACID properties

- Atomicity: a transaction either entirely passes or entirely fails
- Consistency: rejects wrong transactions (or transactions that lead to an invalid state)
- Isolation: handles concurrency
- Durability: once a modification is committed, it persists even after a power loss, a system crash.

SQL is so powerful and easy to use

But Big Data ?

- There are things that are so big that they have implications for everyone, whether we want it or not.
- Big Data is one of those things that is completely transforming the way we do business and is impacting most other parts of our live.

A World of Data

- New York Stock Exchange generates 1Tb transaction data per day
- Internet Archive stores around 15Pb of data and grows 20Tb per month
- The LHC (Large Hadron Collider) produces approximately 15Pb of data per year

More of what we do is leaving a digital footprint (data) which can be recorded. We are generating so much data that we can not store it on a single machine. Furthermore, data is not always fit for RDBMS. The choice of the technologies to use depends on the type of data and on the problem to solve.

Even more data

- Facebook manages approximately 10 billion photos or more than 1Pb storage.
- Instagram users have shared over 40 billion photos.
- 300 hours of video are uploaded to YouTube every minute!
- 4.9 million CCTV cameras in the UK in 2013.

What about unstructured data ?

Companies discover hidden treasures in their documents and unstructured data, which grow at an exponential rate.

However...
How to store so much information? (Pb, Eb)
How to access such information quickly?
How to treat information in such heterogeneous formats?
It used to be much easier when everything was in SQL tables
How to make it scalable, fault-tolerant and flexible?

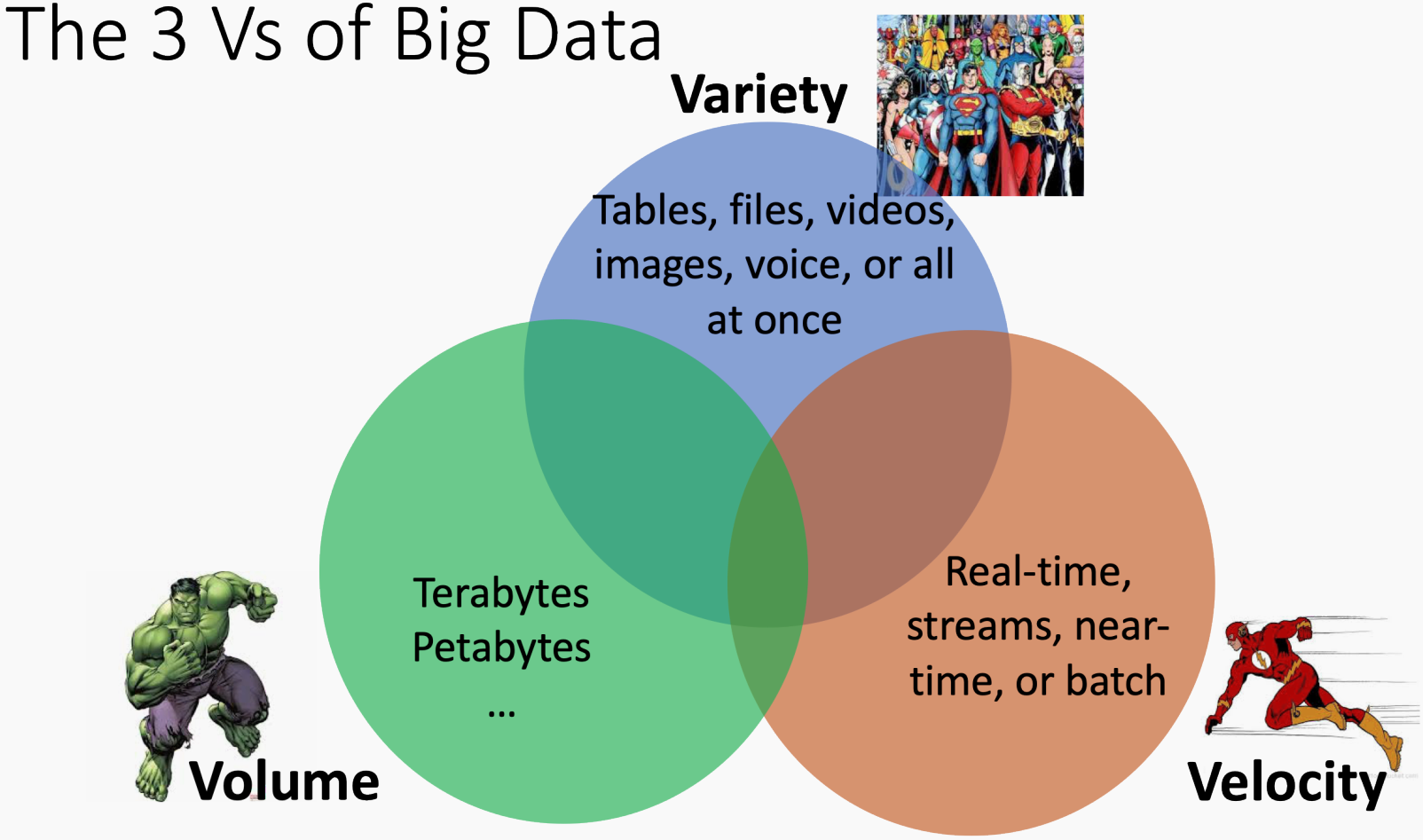


Figure 1: Volumes/Variety/Velocity

Severity of 3Vs ?

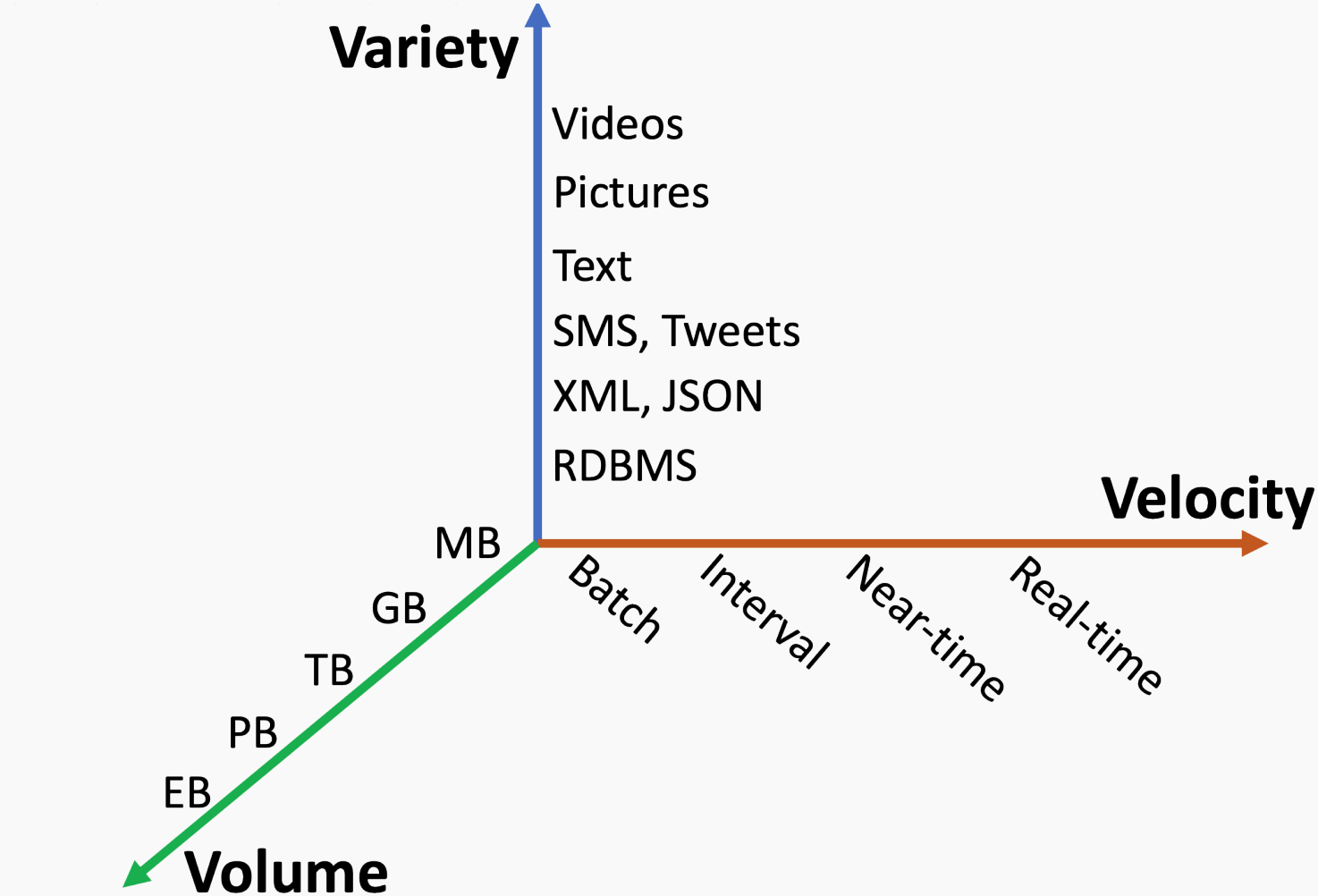


Figure 2: Severity

What is downside of using Relational DB

- Does not scale out horizontally (concurrency and data size) only vertically through sharding.
- Data is normalised, meaning lots of joining, affecting speed.
- Schema-on-write (requires the schema to be well-defined)

So, difficult to handle our rapidly enlarging schema-less data in RDBMS.

NoSQL DB ?

Non relational databases no SQL, not only SQL, not yet SQL, ... NoSQL is a movement promoting a loosely defined class of non-relational data.
Easy on replication and clusters.
However, some restrictions on concurrency and transactions.
Many of them developed for the “web world”. So can deal with high traffic.

BASE in NoSQL?

NoSQL brings BASE rather than ACID

- Basically Available: Guaranteed Availability
- Soft-state: The state of the system may change, even without a query
- Eventually Consistent: The system will become consistent over time.

Important to note, NoSQL systems can impose full ACID guarantees by adding a supplementary middleware layer.

Different Types of NoSQL DBs

- Key-Value Pair: A very simple database that uses an associative array (think of it as a map or a dictionary)
- Columnar: Stores data tables by column rather than row.
- Document: Oriented to the way data is saved and retrieved.
- Graph: Node-Relationship Attributes on nodes and edges

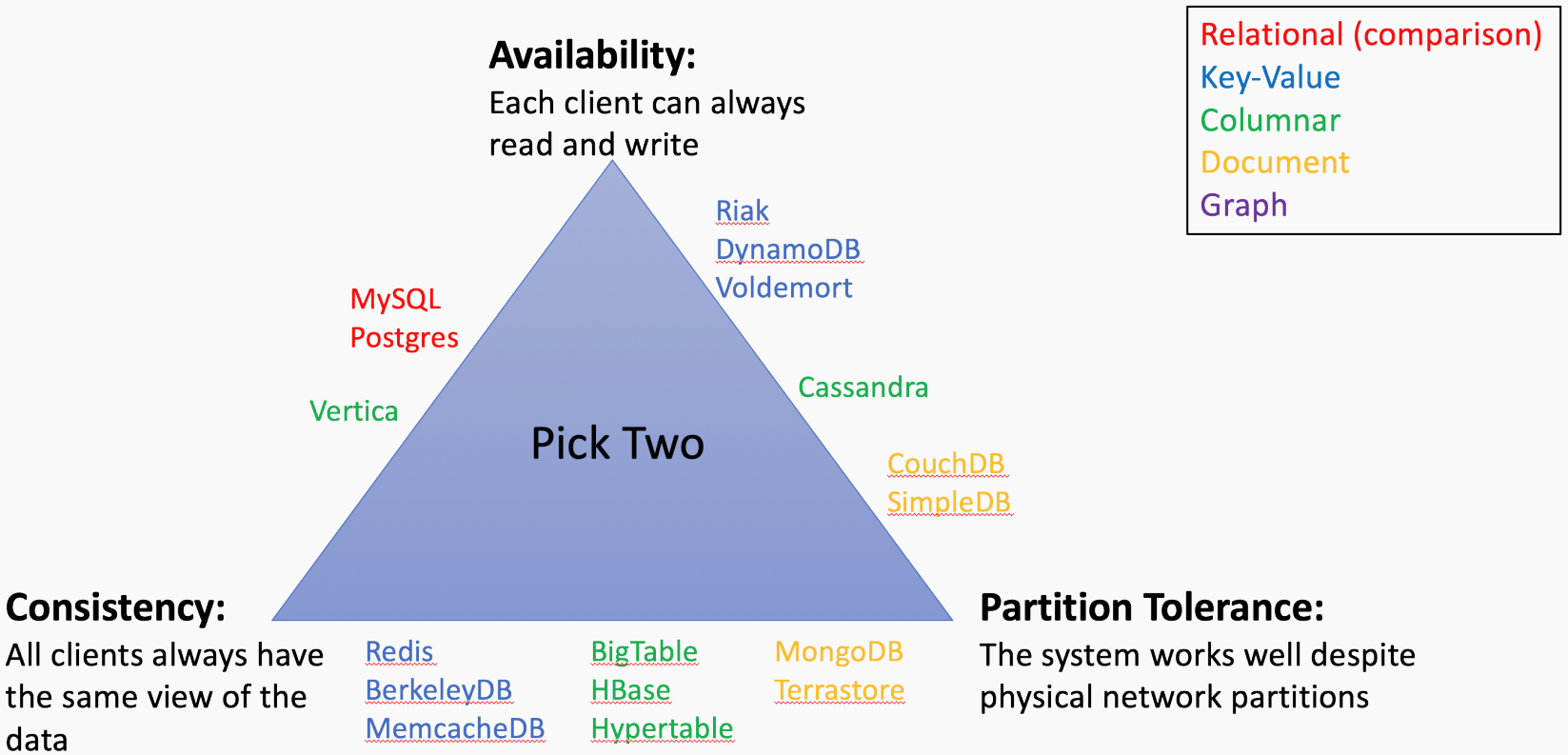
Different Types of Non-NoSQL DBs ?

Figure 3: Types of Non-Relational DBs

CAP Theorem?

Impossible for any shared data system to guarantee simultaneously all of the following three properties: Availability, Consistency, Partitioning.

What Will You Get With Each NoSQL Systems?



Request Handling

We focused on the problem of data volume and scalability for now. But sometimes the number of requests can be enormous and become a problem as well. For now, take 2 minutes and think of how you would design a system that handles the distribution of all incoming requests to multiple servers?

Distributed request handling (Round robin approach)

We have N servers (e.g. webserver), we should distribute all requests among these servers while making sure that all servers receive a similar amount of requests. One naive approach could be round robin. The idea is to assign each request to each server sequentially. After you reached the number of total servers, reset the counter, and start the process from scratch. Round robin is an easy solution and good for balancing the workload of each machine.

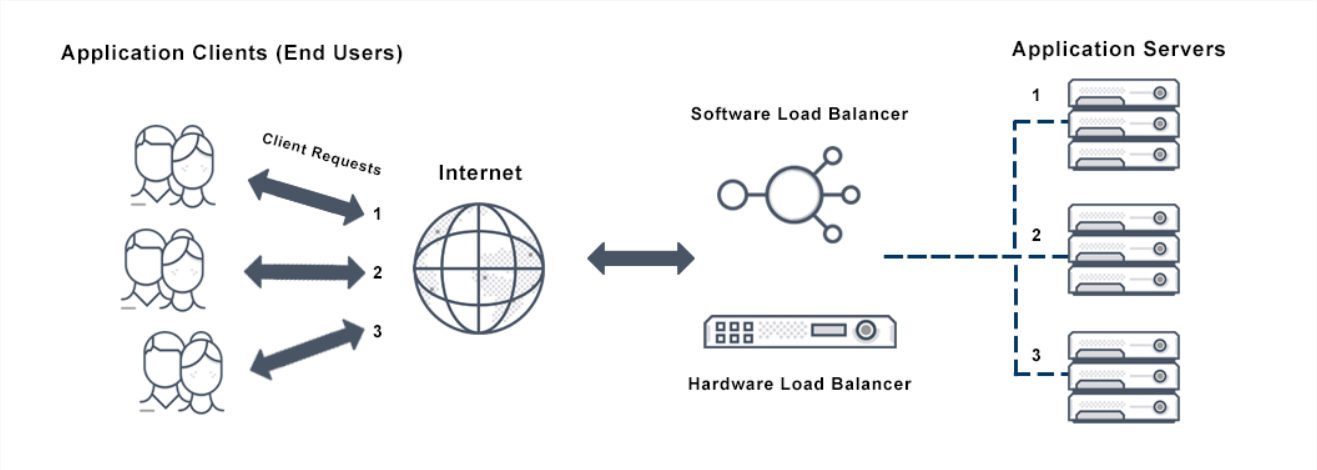


Figure 5: Load Balancing

Distributed request handling (Hash-based approach)

A problem occurs if the same user's request comes another time, the request can end up in a different web server every time .

One concrete example to this problem: let's assume the web server stores data of people located in Asia. When the new request of a user from Europe arrives, the web server needs to ask data from a DB, because it does not store data for that user. Consequently, the strategy creates overhead, increases the system's latency and lower the throughput of the system.

If we use some kind of hashing approach, we can guarantee a specific user's request goes to the same server every time without creating unbalanced distribution.

$$server := serverList[hash(key)\%N] \tag{1}$$

Distributed request handling

Hashing-based technique solve our problem for both good distribution as well as guarantee of the request always going to to the same node. However, how can we handle the failure of node ?

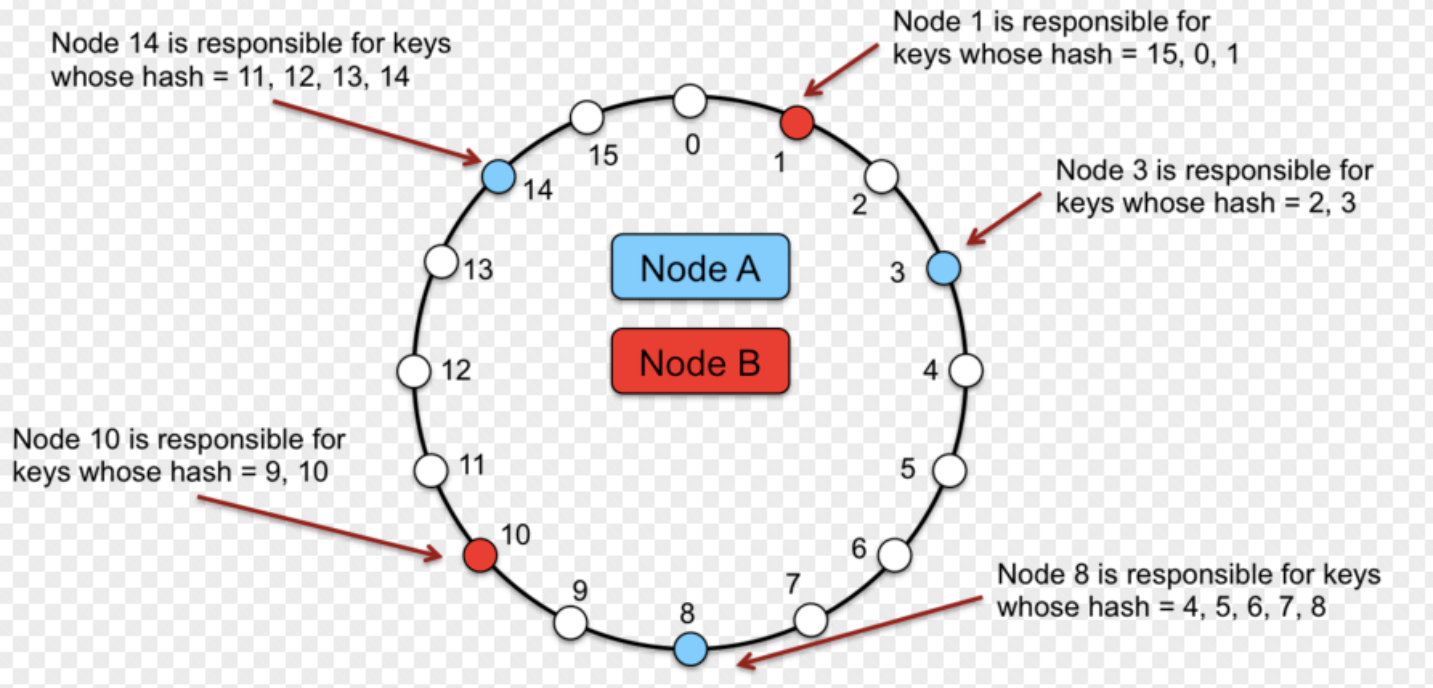
One of the most powerful approach in (big) distributed systems for this problem is consistent hashing. First introduced by Karger et. al. at MIT in 1996[1]. The basic idea is that each server is mapped to a point on a circle with a hash function. To lookup the server for a given key, you hash the key and find that point on the circle. Then you scan forward until you find the first hash value for any server.

---

[1] Karger, D.; Lehman, E.; Leighton, T.; Panigrahy, R.; Levine, M.; Lewin, D. (1997). Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing. ACM Press New York, NY, USA. pp. 654–663. doi:10.1145/258533.258660.

Distributed request handling (Consistent Hashing)

The benefit of this new complexity is any server from the circle can be added or removed. For example, in the case of Node 8 failure, Node 10 will be responsible for keys of Node 8(i.e., 4, 5, 6, 7, 8). We can cover the case of node failure by using consistent hashing. However, it might cause uneven distribution of requests. We can overcome this issue by using multiple hash function, therefore the node appears different times on the circle, requests would be well-balanced.



Closing Remarks

We covered main aspects (i.e., concept of big data, NoSQL, load balancing) of distributed data-intensive computing. Do you think relational databases are going to die soon?

Will Relational Databases Die ?

Nope! Growth of NoSQL databases does not mean the end of RDBMS.

Relational model is still relevant

- perfectly fit for many applications (reasonable amount of structured data)
- ACID
- A lot of SQL tools
- Many professionals are trained to use them

Won't die in the near future! Different problems, different solutions
We need more than a hammer to not see everything as a nail!