Relational vs. No-SQL Databases

Which databases are right for your Use Case?

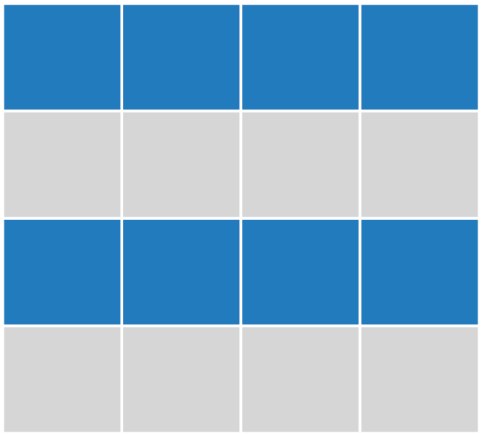
Prabhu R Chennupati | Enterprise Consulting Architect | Mastech InfoTrellis

With the evolution of various types and classifications of databases over the last few decades, it has increasingly become common to discuss and decide which one fits the need and explore the use cases of the application/system being built. This paper lays out various types and classifications of databases, how data is represented in each, and significant characteristics highlighting some key differences.

Contents

Relational Databases

Relational Databases provide a store of related data tables where data is represented in rows. These tables have a fixed schema and use SQL to manage data. Relational Databases support ACID properties which are key for any transactional purposes.



Normalized Representation

Database normalization is an efficient way of organizing data in a database to eliminate redundancy. As part of data normalization, the dependencies of columns and tables of a database are enforced correctly. For the database to be normalized, there are rules known as "normal forms" that are not covered in this paper.

In this example, the customer data is normalized to a certain extent where additional names, phones, emails, etc., can be added without repeating any other customer information, thus avoiding data redundancy.

It can be further normalized but left at this level for the focus of this paper.

Name

PK	PID	Name	Type
N1	1001	John Smith	Legal

Email

PK	PID	Email	Type
E1	1001	John@email.com	Personal

Customer

PK	PID	Cust Type
C1	1001	Individual

Phone

PK	PID	Phone	Type
P1	1001	423-223-7491	Mobile

Address

PK	PID	Address	Type
A1	1001	123 Main St.	Billing

Key

PK	PID	ID	Type
K1	1001	SF001	SF ID

Denormalized Representation

Sometimes, faster data access is more important than data redundancy for specific needs of an application, and in such cases, the data is put together in single rows even though some data is repeated. This kind of data representation is known as a Denormalized Table(s).

In the below example, all the customer attributes are put into a single table. Each row represents one customer in a denormalized fashion.

PK	PID	Name	Name Type	Phone	Phone Type	Email	Email Type	Address	Address Type	Key	Key Type
C1	1001	John Smith	Legal	423-223-7491	Moblie	john@email.com	Personal	123 Main St.	Billing	SF001	SF ID

No-SQL Databases

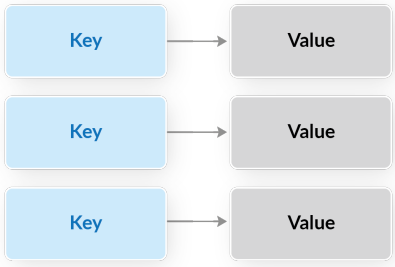
No-SQL databases refer to flexible schema, high-performance, and non-relational data stores and are known for their Scalability, Resilience, and Availability characteristics. They typically don't provide ACID guarantees beyond the scope of a single database partition and does not join and store normalized data tables using SQLs, No-SQL stores, unstructured or semi-structured data.

There are four classifications of No-SQL databases –

- Key-Value Store – data is represented in a collection of key-value pairs.
- Column Family – each attribute is stored as a column, and there are different forms of representations.
- Document Store – data is stored in a document, typically in JSONs or XMLs.
- Graph Store – information is stored as edges and vertices.

Key-Value Store

The data in the Key-Value No-SQL database is stored in a pair of Key and Value. Each attribute is represented in this format - the data is stored as a hash table with a unique key and value in some common formats like String, JSON, BLOB, etc.



A Key-Value store is like a relational database with only two columns, the key or attribute name and the value of the attribute.

For instance, customer information stored in the Key-Value database could be stored in the following format. Key means attribute names like Cust ID, Name, etc., and value column consists of values like SF001 and John Smith.

Key	Value
Cust ID	SF001
Name	John Smith
Phone	423-223-7491
Email	John@email.com
Address	123 Main St., NY, NY, 10011

Column Family

The data in the columnar database is stored by each attribute into a separate column based on its type, unlike in a relational database where the data is stored in rows. The columnar database delivers high performance on aggregation queries as you can read those columns directly without consuming memory with the unwanted data.

For instance, the customer data could be stored in the columnar database by each customer attribute. Its columns like Cust ID, Name, Phone, etc., are separated out into individual column tables.

Typically, the Column Family store is combined with Key-Value data representation. Related data is stored as a set of nested Key/Value pairs within a single column.

ID	Cust ID
1001	SF001

ID	Name
1001	John Smith

ID	Phone
1001	423-223-7491

ID	Email
1001	John@email.com

ID	Address
1001	123 Main St. NY, NY, 10011

Document Store

The data in the document-based No-SQL database is stored in a document format in JSON or XML formats. Each document has a unique key assigned, and the data inside the document can also be queried. When an update is required on a given attribute, the entire document gets updated.

Documents are stored and retrieved in a form closer to the data objects used in applications, which means less translation is needed to use the data in an application.

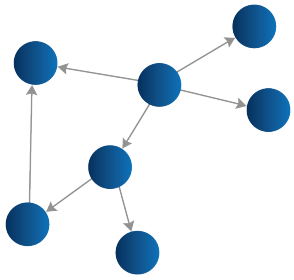
For instance, customer information is stored in the form of a JSON document, as shown here. Each attribute of the customer can be queried and full, or part of the document is retrieved.

Customer Document

```
{
    "Cust ID" : SF001
    "Name": John Smith,
    "Phone": 423-223-7491,
    "Emai": john@email.com
    "Address": 123Main St., NY, NY, 10011
}
```

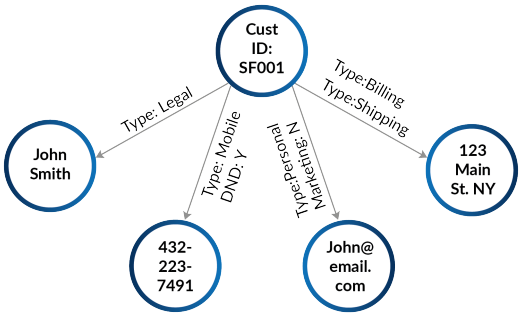
Graph Store

The data in the graph database is stored in vertices (nodes) and edges (relationships/links). The entity is stored as a node with the relationship as edges. An edge gives a relationship between nodes. Traversing relationships are fast as they are already stored as edges, and there is no need to calculate them.



The edges/relationships are first-class elements in graph DB, unlike in the relational database where the relationships are implied.

For instance, the customer data could be stored in a graph database with each attribute value in each of the vertices (nodes). The edges (links) can store the relationship of each vertex and other information, as shown in the below figure.



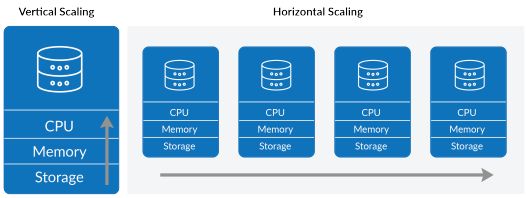
Relational vs. No-SQL

Schema

Relational Databases rely on static schema (or not so much changing schema). No-SQL's greatest strength is that it can handle flexible schema. It allows for attributes to be added or removed; it adds a lot more flexibility when all the data attributes are not known upfront for building applications.

Scalability

Relational Databases are typically provisioned to a single server and scale vertically by adding more resources to the machine. In contrast, No-SQL databases scale out horizontally by adding more nodes to the cluster.

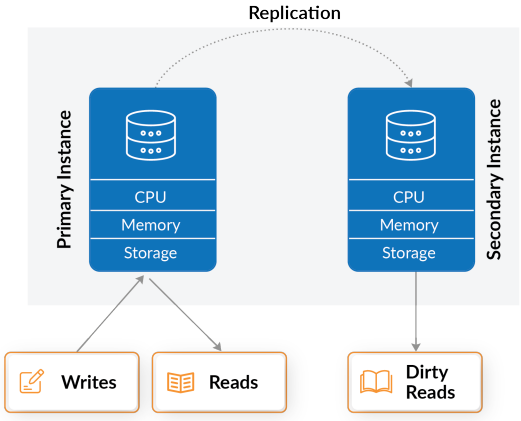


Relational Databases need specialized hardware to handle high resources, unlike No-SQL databases, which can run cost-effectively on commodity hardware.

Consistency

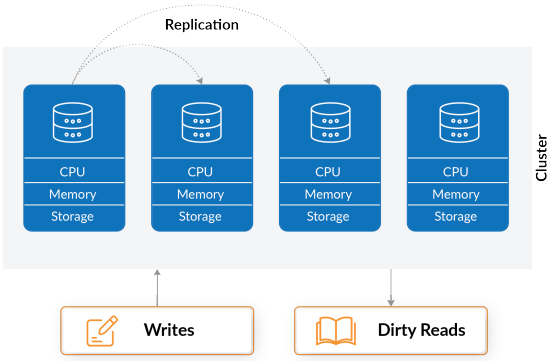
One of the fundamental characteristics of Relational Databases is consistency. Since there is only one primary node where the writes happen, the reads are always consistent if it occurs on the primary node.

Relational Databases offer replication features where copies of the primary database can be made to other secondary nodes. Write operations are made to the primary node and replicated to each of the secondary nodes. Upon a failure, the primary instance can failover to a secondary to provide high availability.



Secondary nodes can be used to distribute read operations. Still, the reads could be Dirty Reads meaning the data might not be the latest or greatest. If the application can tolerate Dirty Read, read operations can be routed to any secondary nodes to reduce system load.

NoSQL Databases mostly go by Eventual Consistency when it comes to writes. This is how No-SQL databases provide superior performance even on high-volume data.

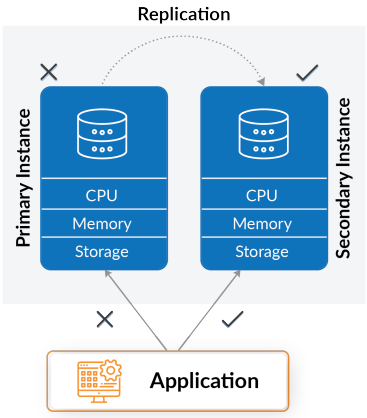


There is a chance that the applications will read old data till the time writes are propagated to all nodes across the cluster – Dirty Reads. Applications that cannot accept Dirty Reads should not use No-SQL databases.

Availability

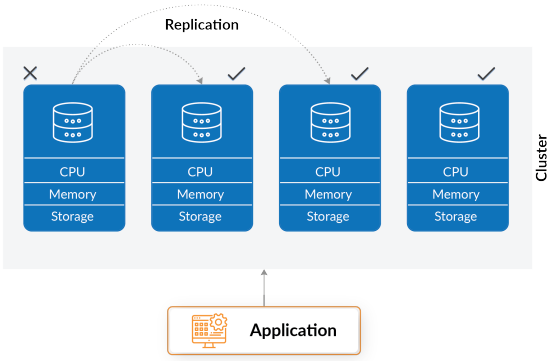
Availability is referred to making sure there is an immediate response or the requester irrespective of recency of data. Both Relational and No-SQL databases support Availability in their ways.

Relational Databases with primary/secondary nodes and a replication feature setup can provide such Availability to its applications.



If the primary goes down due to any reason, the traffic is shifted to the secondary node, even though the secondary might not have up-to-date data. The recency of the data depends on the replication interval setup. Some of the in-flight data written on the primary node at the time of the crash would also be lost.

In a No-SQL database setup, the data is distributed across multiple nodes and replicated across more than one node. Even if one of the nodes goes down, the data residing on the node that went down will still be available on other nodes.



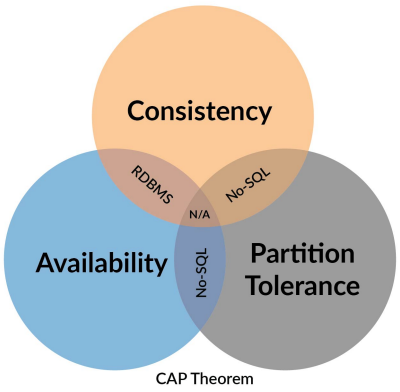
Partition Tolerance

Partition Tolerance means that a given system continues to operate even with data loss or system failure. A single node failure should not cause the entire system to collapse. Unlike Relational Databases, NoSQL databases support Partition Tolerance; they scale horizontally, often across commodity servers. It partitions and replicates data across the nodes, providing redundancy and fault tolerance.

CAP Theorem

According to CAP Theorem, a distributed system can only guarantee two features. If you need Availability and Partition Tolerance, you might have to

let Consistency slip and forget about ACID.



CAP Theorem

Relational Databases support Consistency and Availability; it does not support Partition Tolerance. In comparison, No-SQL databases support Availability and Partition Tolerance along with Eventual Consistency.

Conclusion

Choosing between Relational and No-SQL databases depends on the application's requirements and its use cases to support. Both have their own merits and fit into their use cases; Relational supports ACID properties and No-SQL is available with Eventual Consistency of data.

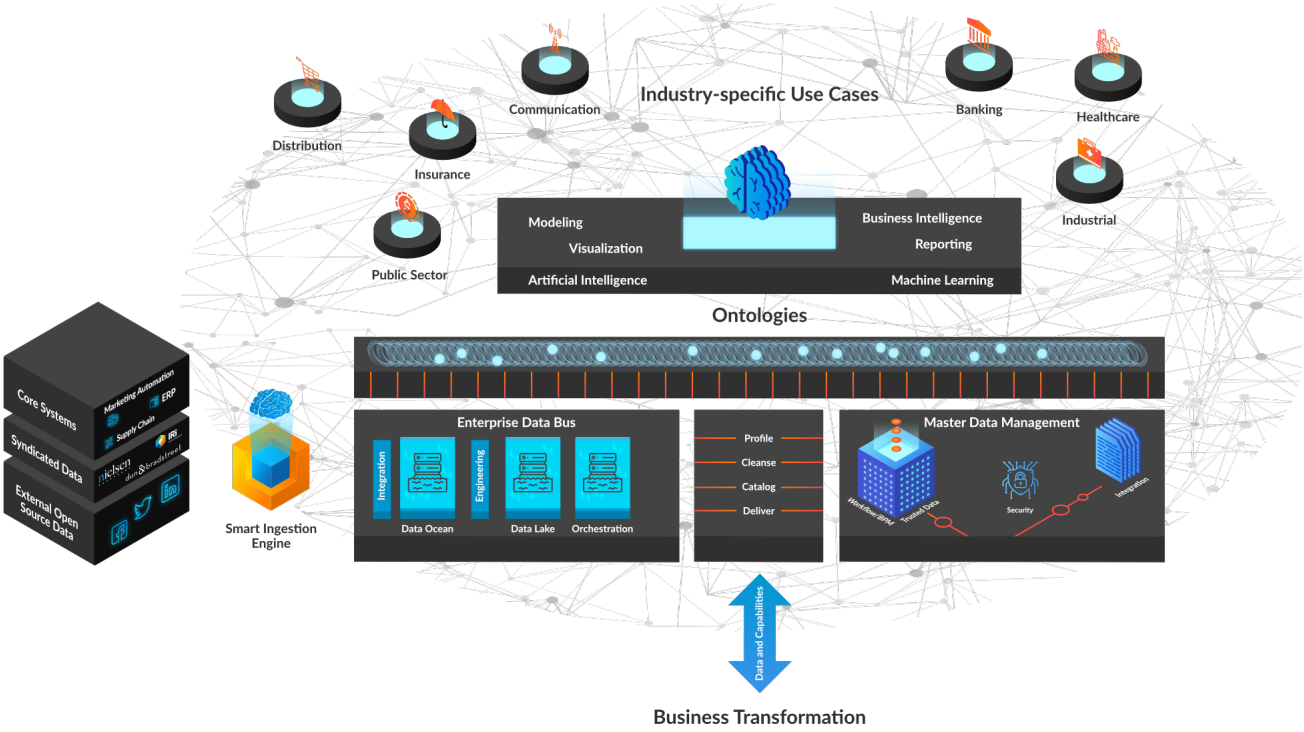
If consistent data is needed for transactional applications, and Eventual Consistency is not acceptable, then it's a clear win for Relational Databases. But if Eventual Consistency is acceptable, then No-SQL databases have a lot to offer.

Feature	Relational	No-SQL
Format	Structured	Structured,Semi-structured, Unstructured
Schema	Static Schema	Flexible Schema
Scalability	Vertical-add more CPU, Memory & Storage	Horizontal-add more nodes
Consistency	Consistent	Eventually Consistent
Availability	Highly Available with Secondary Instance	Highly Available
Pratition Tolerance	Not Supported	Supported

New-SQL is an emerging concept in its nascent state where attempts are being made to combine distributed Scalability of No-SQL with ACID properties of Relational Databases. So eventually, we might see high volumes of data across distributed environments with transactional support and ACID properties.

We Architect Enterprise Intelligence

At Mastech InfoTrellis we work to expose the entire corpus of enterprise data and leverage it with state of the art techniques from Decision & Data Science to accelerate enterprise learning. We would love to talk with you about it.



Author

Prabhu Chennupati is an Enterprise Consulting Architect at Mastech InfoTrellis. He is an avid technologist with extensive experience in enterprise application design and development. He leads the technical teams in Master Data Management (MDM) space spread across multiple continents, delivering high-quality solutions on time.

About

Mastech InfoTrellis partners with enterprises to help them achieve their business objectives by leveraging the power of data to derive deep, analytical insights about their business and its operations. We accelerate business velocity, minimize costs, and drastically improve corporate resiliency through personalized, process-oriented programs, consisting of strategy, data management (including master data management), business intelligence and reporting, data engineering, predictive analytics, and advanced analytics. Part of the NYSE-listed, $193.6M, digital transformation IT services company, Mastech Digital; we drive businesses forward around the world, with offices spread across the US, Canada, India, Singapore, UK, and Ireland.