An Introduction to
NoSQL

NoSQL

NotOnlySQL Database

What is NoSQL

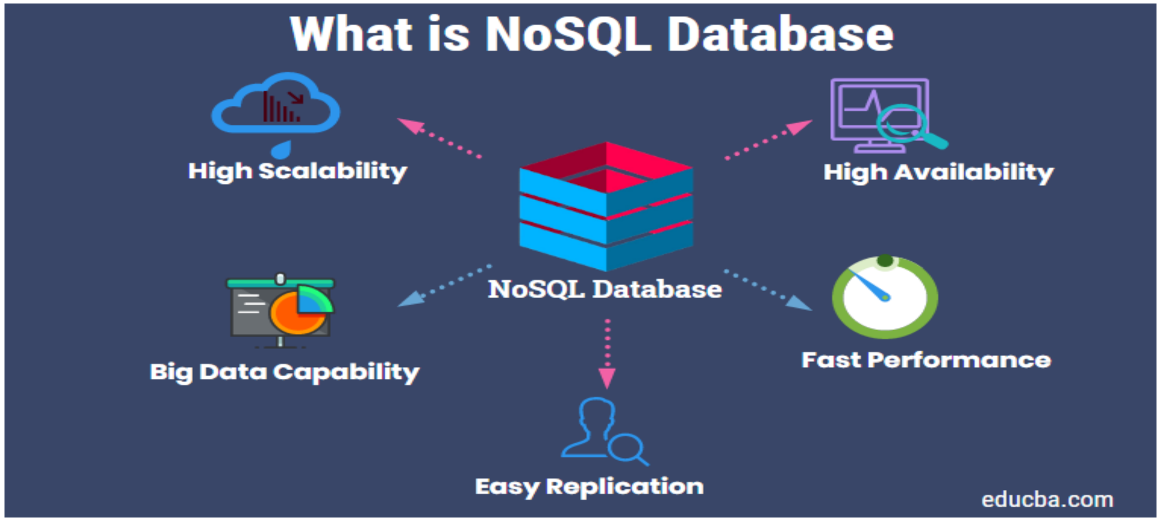
- When people use the term “NoSQL database”, they typically use it to refer to any non-relational database. Some say the term “NoSQL” stands for “non SQL” while others say it stands for “not only SQL.” Either way, most agree that NoSQL databases are databases that store data in a format other than relational tables.

- A common misconception is that NoSQL databases or non-relational databases don’t store relationship data well. NoSQL databases can store relationship data—they just store it differently than relational databases do. In fact, when compared with SQL databases, many find modeling relationship data in NoSQL databases to be easier than in SQL databases, because related data doesn’t have to be split between tables.

- NoSQL data models allow related data to be nested within a single data structure.

- NoSQL databases emerged in the late 2000s as the cost of storage dramatically decreased. Gone were the days of needing to create a complex, difficult-to-manage data model simply for the purposes of reducing data duplication. Developers (rather than storage) were becoming the primary cost of software development, so NoSQL databases optimized for developer productivity.

Why NoSQL



- To gain a competitive edge, businesses need to innovate and adopt agile methodologies. NoSQL is the perfect fit for such needs in the database sphere. They allow you to store data in a flexible and fluid data model, provide scaling up capabilities to your database tier, and provide 100% uptime through replication.

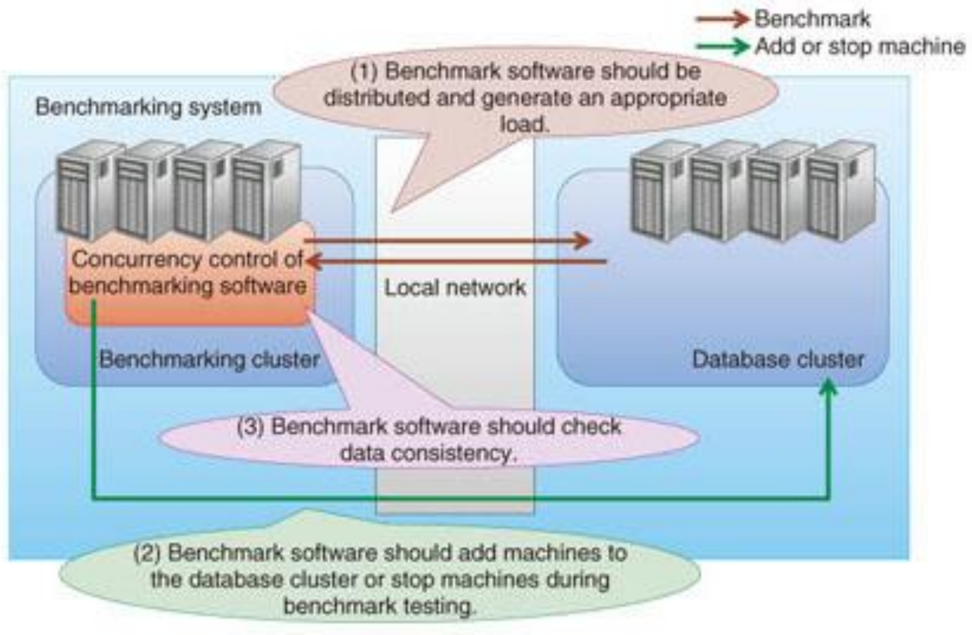
- To top it off, NosDB is a NoSQL Document Database written 100% in .NET. and it comes with algorithms like MapReduce baked right into the database. To assist in migration from an RDBMS, NosDB provides an official ADO.NET wrapper. So if you're a .NET developer and want to try a .NET native NoSQL database, give NosDB a try.

- With a NoSQL database offering so much, the question "Why NoSQL?" should be "Why not NoSQL?"

NoSQL characteristics

- Because NoSQL databases feature scale-out and replication, a NoSQL benchmark should take scalability, elasticity, consistency, and availability into account as well as performance. We explain each characteristic and describe the benchmarking software design points concerning these characteristics.

- Scalability indicates how the performance of a NoSQL database cluster scales with the number of physical machines. If performance improves as machines are added to a NoSQL database cluster, we can say that the NoSQL product has high scalability. In scalability benchmarking with many physical machines, the load generator, which benchmarks the NoSQL database cluster, is often a bottleneck. An effective approach for preventing this is to design benchmark software as a distributed system running on the cluster.


Benchmark
Add or stop machine
(1) Benchmark software should be distributed and generate an appropriate load.
Benchmarking system
Concurrency control of benchmarking software
Local network
Benchmarking cluster
Database cluster
(3) Benchmark software should check data consistency.
(2) Benchmark software should add machines to the database cluster or stop machines during benchmark testing.

NoSQL characteristics

- **No Structured Query Language (SQL) :**

You can use a similar language, like Cassandra Query Language (CQL), or you can use a radically different API, such as one using JSON. But if you can compliantly accept all SQL statements, verbatim, you are not "NoSQL," but "NewSQL." Because you literally are a SQL database.

- **No table joins**:

Remember the 'relational' part of RDBMS? One of the other big differentiators was that NoSQL used to mean absolutely no table JOINs; the tables do not relate to one another. If you can permit table JOINs, you are again treading into the world of "NewSQL" rather than "NoSQL." e.g., MongoDB $lookup. Using JOINs in MongoDB NoSQL Databases — SitePoint

- **Schema-optional or schemaless:**

SQL requires all tables have pre-defined schemas. NoSQL permits you to have a schema (schema optional) or may be schemaless. These are anathema to the SQL RDBMS world, where you have to pre-define your schema, have strong typing, and you don't want to hear about sparse data models. Schema-optional or schemaless permit dealing with a wider variety of data, and with rapidly evolving data models.

- **Horizontal scalability**:

The NoSQL world was designed (ideally) to scale to the web, or the cloud, or Internet of Things (IoT). Whereas SQL was designed (ideally) with the enterprise in mind — a Fortune 500 company, for instance. Summarily, SQL was designed to scale vertically for an enterprise ("one very big box" like a mainframe), whereas NoSQL was designed to scale horizontally ("many little boxes, all alike" on commodity hardware). However, while this was generally true in the past — a rule of thumb — there are a few NoSQL systems ported to mainframes, and now some SQL systems designed to scale horizontally. Ideally a database can be architected to scale horizontally and vertically (e.g., Scylla).

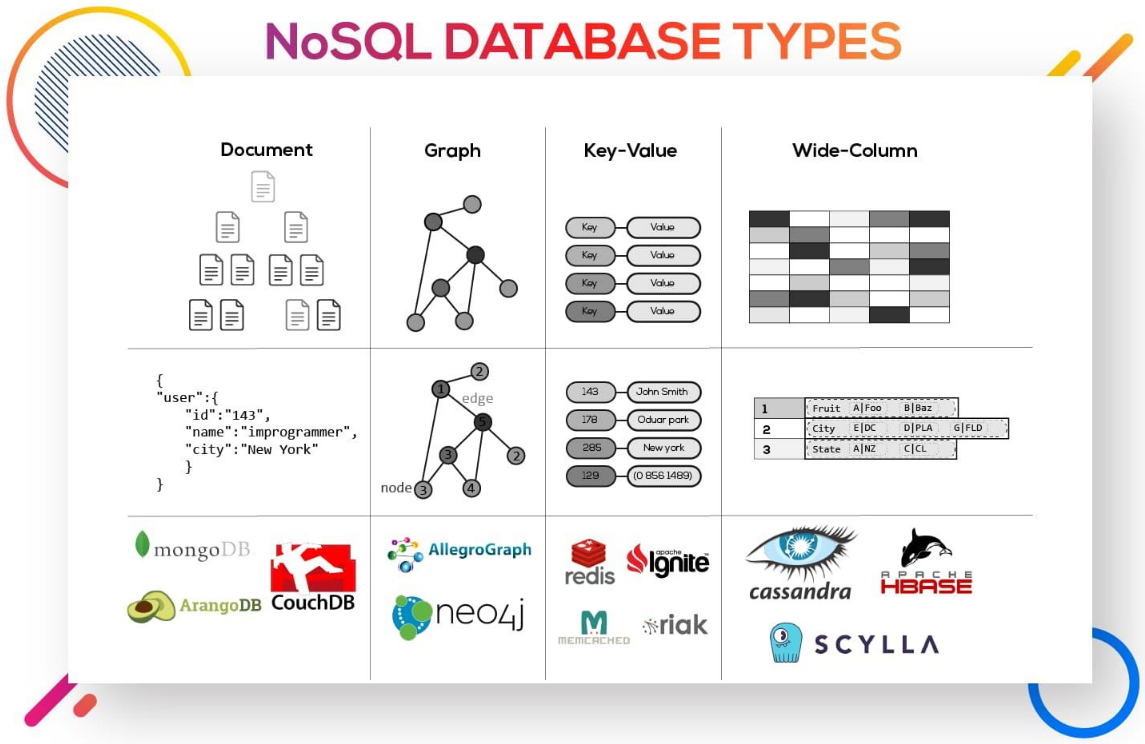
- **Availability-focused (vs. Consistency-focused):**

SQL RDBMS's grew up in the world of big banking and other commercial use cases that required consistency in transaction processing. The money was either in your account, or it wasn't. And if you checked your balance, it needed to be precise. Reloading it should not change what's in there. But that means the database needs to take its own sweet time to make sure your balance is right. Whereas for the NoSQL world, the database needed to be available. No blocking transactions. Which means that data may be eventually consistent. i.e., reload and you'll eventually see the right answer. That was fine with use cases that were less mission-critical like social media posting or caching ephemeral data like browser cookies. However, again, this is a rule of thumb from the early days. Now, many NoSQL systems support ACID, two-phase commits, strong consistency, and so on. Still, the prevalence is for many NoSQL systems to be more aligned with the "AP" (Availability/Partition Tolerant) side than the "CP" (Consistency/Partition Tolerant) side of the CAP theorem.

NoSQL databases types

- Here are the four main types of NoSQL databases:

  Document databases
  Key-value stores
  Column-oriented databases
  Graph databases



```
{
"user":{
    "id":"143",
    "name":"improgrammer",
    "city":"New York"
    }
}
```

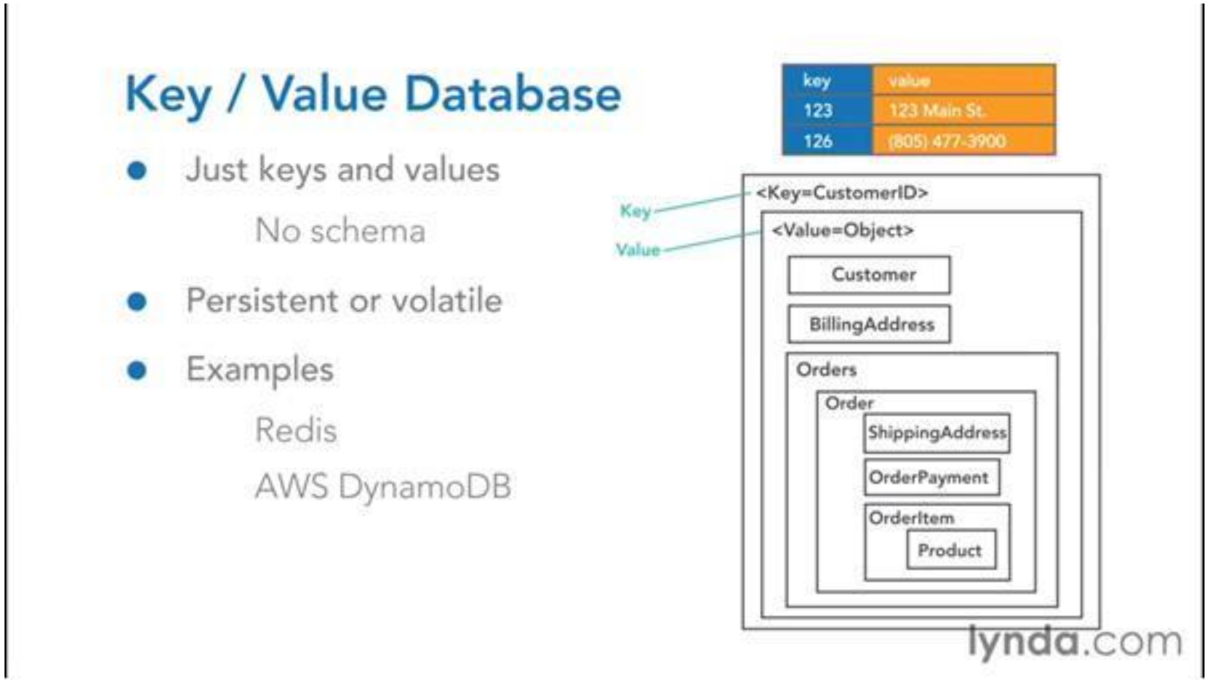
Document Databases

- A document database stores data in JSON, BSON , or XML documents (not Word documents or Google docs, of course). In a document database, documents can be nested. Particular elements can be indexed for faster querying.
- Documents can be stored and retrieved in a form that is much closer to the data objects used in applications, which means less translation is required to use the data in an application. SQL data must often be assembled and disassembled when moving back and forth between applications and storage.
- Document databases are popular with developers because they have the flexibility to rework their document structures as needed to suit their application, shaping their data structures as their application requirements change over time. This flexibility speeds development because in effect data becomes like code and is under the control of developers. In SQL databases, intervention by database administrators may be required to change the structure of a database.
- The most widely adopted document databases are usually implemented with a scale-out architecture, providing a clear path to scalability of both data volumes and traffic.
- Use cases include ecommerce platforms, trading platforms, and mobile app development across industries.
- Comparing MongoDB vs PostgreSQL offers a detailed analysis of MongoDB, the leading NoSQL database, and PostgreSQL, one of the most popular SQL databases.

Key-value stores

- The simplest type of NoSQL database is a key-value store . Every data element in the database is stored as a key value pair consisting of an attribute name (or "key") and a value. In a sense, a key-value store is like a relational database with only two columns: the key or attribute name (such as state) and the value (such as Alaska).

- Use cases include shopping carts, user preferences, and user profiles.



Key / Value Database

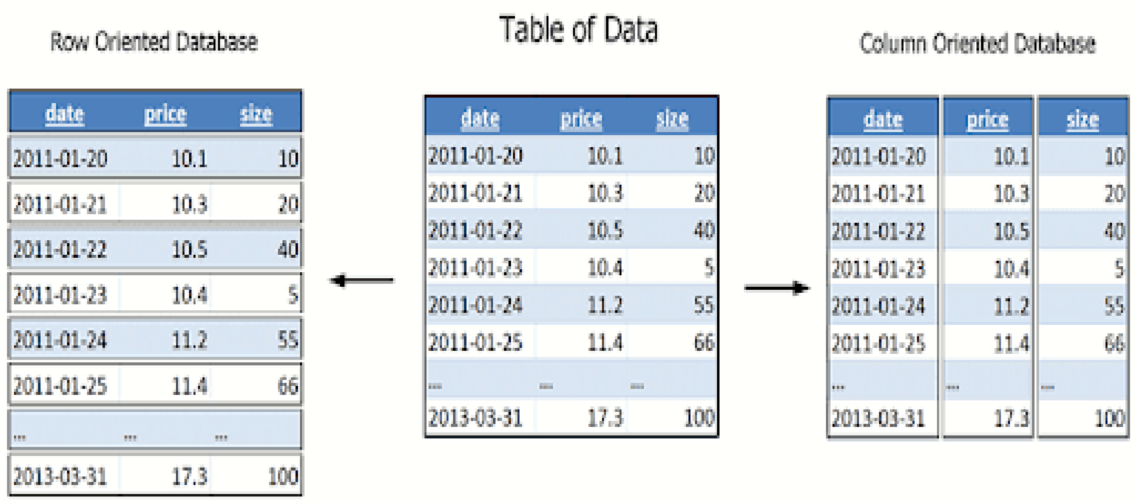
- Just keys and values
  - No schema
- Persistent or volatile
- Examples
  - Redis
  - AWS DynamoDB

key	value
123	123 Main St.
126	(805) 477-3900

Column-oriented databases

- While a relational database stores data in rows and reads data row by row, a column store is organized as a set of columns. This means that when you want to run analytics on a small number of columns, you can read those columns directly without consuming memory with the unwanted data. Columns are often of the same type and benefit from more efficient compression, making reads even faster. Columnar databases can quickly aggregate the value of a given column (adding up the total sales for the year, for example). Use cases include analytics.

- Unfortunately there is no free lunch, which means that while columnar databases are great for analytics, the way in which they write data makes it very difficult for them to be strongly consistent as writes of all the columns require multiple write events on disk. Relational databases don't suffer from this problem as row data is written contiguously to disk.



Row Oriented Database

date	price	size
2011-01-20	10.1	10
2011-01-21	10.3	20
2011-01-22	10.5	40
2011-01-23	10.4	5
2011-01-24	11.2	55
2011-01-25	11.4	66
...	...	...
2013-03-31	17.3	100

Table of Data

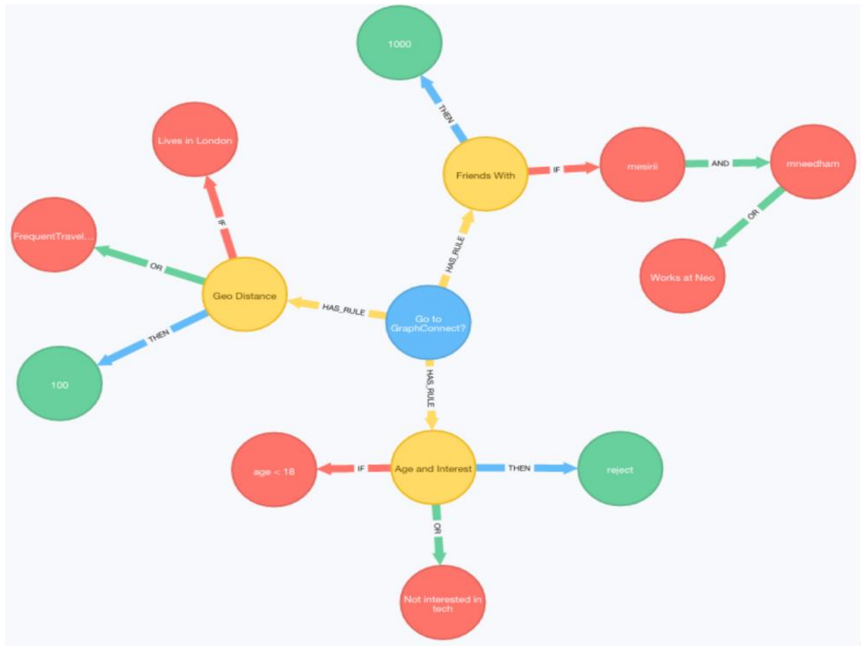
date	price	size
2011-01-20	10.1	10
2011-01-21	10.3	20
2011-01-22	10.5	40
2011-01-23	10.4	5
2011-01-24	11.2	55
2011-01-25	11.4	66
...	...	...
2013-03-31	17.3	100

Column Oriented Database

date	price	size
2011-01-20	10.1	10
2011-01-21	10.3	20
2011-01-22	10.5	40
2011-01-23	10.4	5
2011-01-24	11.2	55
2011-01-25	11.4	66
...	...	...
2013-03-31	17.3	100

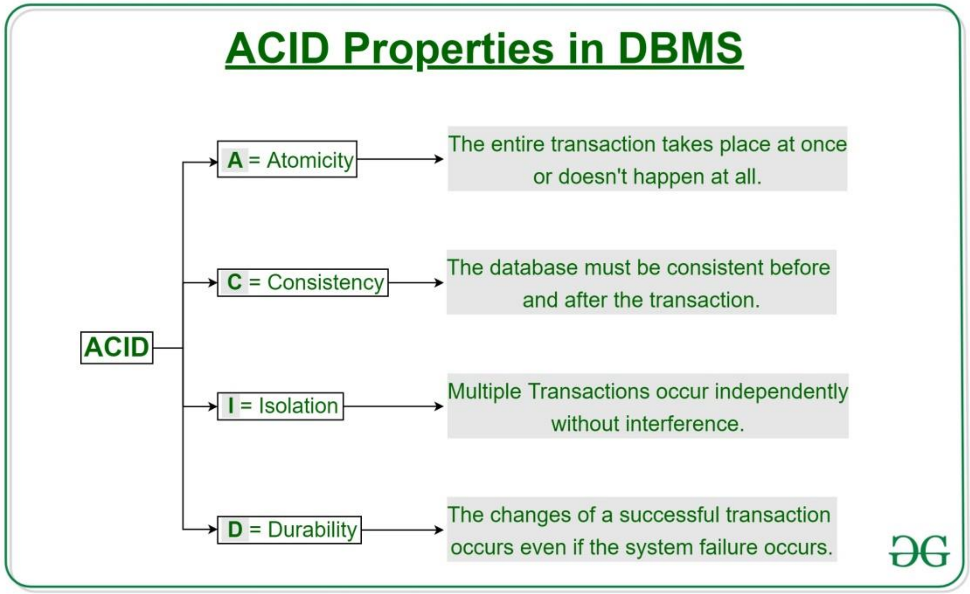
Graph Databases

- A graph database focuses on the relationship between data elements. Each element is stored as a node (such as a person in a social media graph). The connections between elements are called links or relationships. In a graph database, connections are first-class elements of the database, stored directly. In relational databases, links are implied, using data to express the relationships.
- A graph database is optimized to capture and search the connections between data elements, overcoming the overhead associated with JOINing multiple tables in SQL.
- Very few real-world business systems can survive solely on graph queries. As a result graph databases are usually run alongside other more traditional databases.
- Use cases include fraud detection, social networks, and knowledge graphs.
- As you can see, despite a common umbrella, NoSQL databases are diverse in their data structures and their applications.



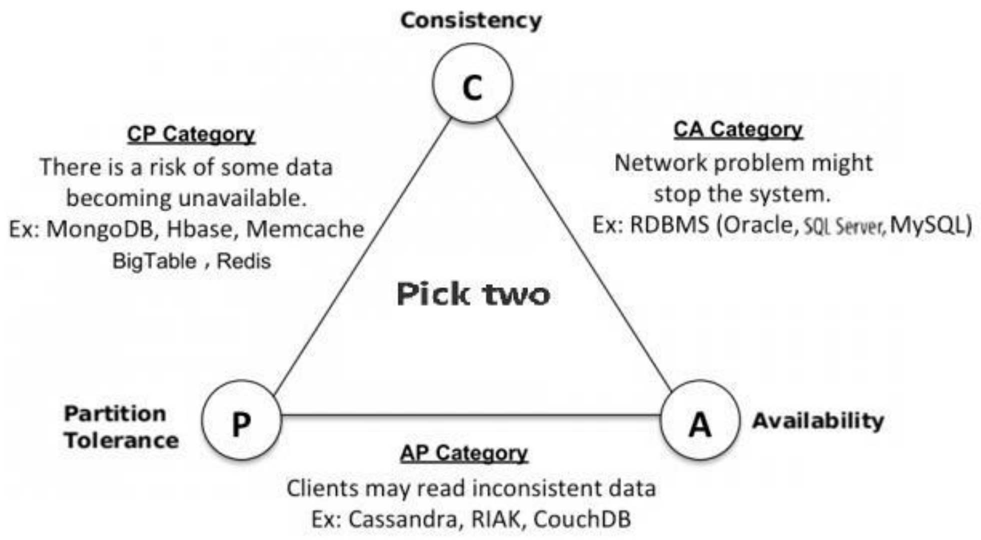
ACID theorem

- In computer science, ACID (atomicity, consistency, isolation, durability) is a set of properties of database transactions intended to guarantee data validity despite errors, power failures, and other mishaps. In the context of databases, a sequence of database operations that satisfies the ACID properties (which can be perceived as a single logical operation on the data) is called a transaction. For example, a transfer of funds from one bank account to another, even involving multiple changes such as debiting one account and crediting another, is a single transaction.
- In 1983, Andreas Reuter and Theo Härder coined the acronym ACID, building on earlier work by Jim Gray who named atomicity, consistency, and durability, but not isolation, when characterizing the transaction concept. These four properties are the major guarantees of the transaction paradigm, which has influenced many aspects of development in database systems.
- According to Gray and Reuter, the IBM Information Management System supported ACID transactions as early as 1973 (although the acronym was created later).


ACID Properties in DBMS
ACID
A = Atomicity
The entire transaction takes place at once or doesn't happen at all.
C = Consistency
The database must be consistent before and after the transaction.
I = Isolation
Multiple Transactions occur independently without interference.
D = Durability
The changes of a successful transaction occurs even if the system failure occurs.

CAP theorem

- Have you ever seen an advertisement for a landscaper, house painter, or some other tradesperson that starts with the headline, "Cheap, Fast, and Good: Pick Two"?
- The CAP theorem applies a similar type of logic to distributed systems—namely, that a distributed system can deliver only two of three desired characteristics: consistency, availability, and partition tolerance (the 'C,' 'A' and 'P' in CAP).
- A distributed system is a network that stores data on more than one node (physical or virtual machines) at the same time. Because all cloud applications are distributed systems, it's essential to understand the CAP theorem when designing a cloud app so that you can choose a data management system that delivers the characteristics your application needs most.
- The CAP theorem is also called Brewer's Theorem, because it was first advanced by Professor Eric A. Brewer during a talk he gave on distributed computing in 2000. Two years later, MIT professors Seth Gilbert and Nancy Lynch published a proof of "Brewer's Conjecture."


Consistency
C
CP Category
There is a risk of some data becoming unavailable.
Ex: MongoDB, Hbase, Memcache BigTable , Redis
CA Category
Network problem might stop the system.
Ex: RDBMS (Oracle, SQL Server, MySQL)
Pick two
Partition Tolerance
P
A
Availability
AP Category
Clients may read inconsistent data
Ex: Cassandra, RIAK, CouchDB

NoSQL Advantages

- **1. Elastic Scalability**

In the past, the best DBA services still had to depend on scaling up whenever there was a need for expansion. This meant purchasing larger servers to deal with the increasing data load. NoSQL databases offer the much easier option of scaling out – the databases are distributed across multiple pre-existing hosts. With an increase in the availability requirements and transaction rates, scaling out onto virtual environments offers a more economical alternative to hardware scaling.

- **2. Useful for big data**

The last decade has witnessed a rapid growth in the transaction rates, as have the volumes of data that need to be stored. This is what led to the creation of the term 'big data', and has been affectionately referred to as the "industrial revolution of data" in certain circles.

- **3. Reduced reliance on in-house DBAs**

A major disadvantage of implementing these powerful high-end RDBMSs is that maintenance is only possible by employing trained DBAs, which certainly don't come cheap. They are intricately involved in the design, installation and performance tuning of these RDBMSs, which makes them virtually indispensable.

- **4. It's cheaper**

NoSQL databases are designed to utilize cheap commodity server clusters for the management of ever-growing transaction and data volumes. RDBMSs, on the other hand, require expensive storage systems and patented servers, which means that the latter has a greater cost per volumes of data stored. This means that for a much lower price, you can store and process a higher volume of data.

- **5. Agile data models**

RDBMSs give colossal headaches when it comes to change management, especially for the large production ones. The minor change must be carefully monitored, and may still involve some downtime or reduction in service levels. NoSQL does not have such restrictions on their data models, and even the more rigid NoSQL databases based on BigTable structure still allow for relative flexibility like an addition of new columns with no major breakdowns.