# **Explain difference between scaling horizontally and vertically for databases**

Answer

* Horizontal scaling means that you scale by adding more machines into your pool of resources whereas
* Vertical scaling means that you scale by adding more power (CPU, RAM) to an existing machine.

In a database world horizontal-scaling is often based on the partitioning of the data i.e. each node contains only part of the data, in vertical-scaling the data resides on a single node and scaling is done through multi-core i.e. spreading the load between the CPU and RAM resources of that machine.

Good examples of horizontal scaling are Cassandra, MongoDB, Google Cloud Spanner. and a good example of vertical scaling is MySQL - Amazon RDS (The cloud version of MySQL).

# **What are the advantages of NoSQL over traditional RDBMS?**

**NoSQL is better** than RDBMS because of the following reasons/properities of NoSQL:

* It supports semi-structured data and volatile data
* It does not have schema
* Read/Write throughput is very high
* Horizontal **scalability** can be achieved easily
* Will support Bigdata in volumes of Terra Bytes & Peta Bytes
* Provides good support for Analytic tools on top of Bigdata
* Can be hosted in cheaper hardware machines
* In-memory caching option is available to increase the performance of queries
* Faster development life cycles for developers

Still, **RDBMS is better** than NoSQL for the following reasons/properties of RDBMS:

* Transactions with **ACID** properties - Atomicity, Consistency, Isolation & Durability
* Adherence to **Strong Schema** of data being written/read
* Real time query management ( in case of data size < 10 Tera bytes )
* Execution of complex queries involving **join** & **group by** clauses

# **Define ACID Properties**

* **Atomicity**: It ensures all-or-none rule for database modifications.
* **Consistency**: Data values are consistent across the database.
* **Isolation**: Two transactions are said to be independent of one another.
* **Durability**: Data is not lost even at the time of server failure.

# **What are the different types of NoSQL Database.**

There are 4 most common types of NoSQL databases as below.

* **Key-value databases**
* **Document databases**
* **Wide-column or Column-family databases**
* **Graph databases stores**

# **What is Key-Value store or Key-Value database?**

**Key/Value** database use the key-value pairs to store the data in a hash table. Key uniquely identifies the data. This is used to store data using appropriate hash function. Key-Value is good choice when you want to lookup some data based on key rather than performing joins over multiple tables of key values.

# **What is DocumentDB?**

**DocumentDB** or **Document database** is a completely NoSQL database service that stores the data as schema-free JSON (JavaScript Object Notation) document.  
When you are working on some application that needs to handle data with changing schema or you are not sure about the data which you needs to work with and how much data application needs to handle. You are also not sure about the structure of data. You also need scalability, low cost and fast deployment for your data. In all these scenarios we consider DocumentDB. There are many DocumentDB services as below.

* Microsoft Azure [Cosmos DB](https://www.qfles.com/interview-question/cosmos-db-interview-questions)
* Amazon DocumentDB
* [Mongo DB](https://www.qfles.com/interview-question/mongodb-interview-questions)

# **What is NoSQL - CAP Theorem?**

SQL follows ACID properties- Atomicity, Consistency, Isolation and Durability while NoSQL follows CAP theorem- Consistency, Availability and Partition tolerance(scalable).

**The CAP Theorem**

Published by Eric Brewer in 2000, the theorem is a set of basic requirements that describe any distributed system. If you imagine a distributed database system with multiple servers, here's how the CAP theorem applies:

* **Consistency** - All the servers in the system will have the same data so users will get the same copy regardless of which server answers their request.
* **Availability** - The system will always respond to a request (even if it's not the latest data or consistent across the system or just a message saying the system isn't working).
* **Partition Tolerance** - The system continues to operate as a whole even if individual servers fail or can't be reached.

It's theoretically impossible to have all 3 requirements met, so a combination of 2 must be chosen and this is usually the deciding factor in what technology is used.

When it comes to distributed databases, the two choices are only AP or CP because if it's not partition tolerant, it's not really a reliable distributed database. So the choice is simpler: if a network split happens, do you want the database to keep answering but with possibly old/bad data (AP)? Or should it just stop responding unless you can get the absolute latest copy (CP)?

**ACID**

This describes a set of properties that apply to data transactions, defined as follows:

* **Atomicity** - Everything in a transaction must happen successfully or none of the changes are committed. This avoids a transaction that changes multiple pieces of data from failing halfway and only making a few changes.
* **Consistency** - The data will only be committed if it passes all the rules in place in the database (ie: data types, triggers, constraints, etc).
* **Isolation** - Transactions won't affect other transactions by changing data that another operation is counting on; and other users won't see partial results of a transaction in progress (depending on isolation mode).
* **Durability** - Once data is committed, it is durably stored and safe against errors, crashes or any other (software) malfunctions within the database.

CAP provides the basic requirements that a distributed system *must follow* and ACID is a set of rules that a database *can choose to follow* that guarantees how it handles transactions and keeps data safe.

There are lots of options other than relational databases for storing more or different kinds of data and they often use a distributed set of servers working together and are designed either for AP or CP under the CAP theorem. When it comes to how safe the committed data is, any ACID compliant system can be considered reliable.

Final note: There really is no such thing as "NoSQL" - it's just a meaningless term that caught on and it's far better to just reference the type of database itself:

* Relational (mysql, oracle, sql server, postgres)
* Document Store (mongodb, riak, couchbase, rethinkdb)
* Key/Value (redis, aerospike, leveldb)
* Wide-Column (different from relational db with columnar storage and really more like nested-key/value: hbase, cassandra)
* Graph (neo4j, titan)
* Search (optimized for storing and searching against text, elasticsearch, solr, lucene)

# [Explanation of BASE terminology](https://stackoverflow.com/questions/3342497/explanation-of-base-terminology)

The BASE acronym was defined by [Eric Brewer](http://en.wikipedia.org/wiki/Eric_Brewer_%28scientist%29), who is also known for formulating the [CAP theorem](http://en.wikipedia.org/wiki/CAP_theorem).  
This leaves me with some questions about **the definition**:   
**B**asically **A**vailable, **S**oft state, **E**ventual consistency

The CAP theorem states that a distributed computer system cannot guarantee all of the following three properties at the same time:

* Consistency
* Availability
* Partition tolerance

A BASE system gives up on consistency.

* **Basically available** indicates that the system does guarantee availability, in terms of the CAP theorem.  
  rather than enforcing immediate consistency, BASE-modelled NoSQL databases will ensure availability of data by spreading and replicating it across the nodes of the database cluster.
* **Soft state** indicates that the state of the system may change over time, even without input. This is because of the eventual consistency model. Due to the lack of immediate consistency, data values may change over time. The BASE model breaks off with the concept of a database which enforces its own consistency, delegating that responsibility to developers.
* **Eventual consistency** indicates that the system will become consistent over time, given that the system doesn't receive input during that time. The fact that BASE does not enforce immediate consistency does not mean that it never achieves it. However, until it does, data reads are still possible (even though they might not reflect the reality).

**[What are the pros and cons of a graph database under NoSQL databases?](https://www.onlineinterviewquestions.com/nosql-interview-questions/" \l "collapseUnfiled7)**

Following are the pros and cons of a graph database which is a type of NoSQL databases: –

# Pros of using graph database:

* These are tailor-made for the networking applications. A social network is a good example of this.
* They can also be perfect for an object-oriented programming system.

# Cons of using graph database:

* Since the degree of interconnection between nodes is high in graph database, so it is not suitable for network partitioning.
* Also, graph databases don’t scale out well in NoSQL databases.

# **[What do you mean by eventual consistency in NoSQL stores?](https://www.onlineinterviewquestions.com/nosql-interview-questions/" \l "collapseUnfiled10)**

# Eventual consistency in NoSQL means that when all the service logics have been executed, the system is left in a consistent state. For achieving high availability, this concept is used in the distributed systems. It gives a guarantee that, if new updates are not made to a given data item, then eventually all accesses to that item will return the last updated value. In NoSQL, it is provided in terms of BASE and RDMS also known as ACID properties. Present NoSQL databases provide client applications with guarantee of eventual consistency. Some NoSQL databases like- MongoDB and Cassandra are eventually consistent in some of the configurations.

Differences between SQL and NoSQL databases

|  |  |
| --- | --- |
| **SQL** | **NoSQL** |
| Works with the relational database system | Works with the distributed database system |
| Vertically scalable | Horizontally scalable |
| Systems are fixed and not flexible | Systems are dynamic and flexible |
| Failure in hierarchical data storage | Perfect for hierarchical data storage |
| Any complex queries can be used | Not suitable for complex queries |

### **[15) What is sharding in nosql?](https://www.onlineinterviewquestions.com/nosql-interview-questions/" \l "collapseUnfiled15)**

**Sharding in NOSQL** is the process of storing data records across multiple machines.It is a partitioning pattern in which each shard is held on a separate database server instance, to spread load and provide fast data access.

### **[11) What do you understand by Polyglot Persistence in NoSQL?](https://www.onlineinterviewquestions.com/nosql-interview-questions/" \l "collapseUnfiled11)**

While storing data, it is advisable to choose multiple data storage systems so that the system allows us to store various data in our future. This is a safer type of data storage system because we do not want to risk on single data storage system. This type of storage is called as **polyglot persistence in NoSQL**.

### **[12) What do you understand by " Polyglot Persistence " in NoSQL?](https://www.onlineinterviewquestions.com/nosql-interview-questions/" \l "collapseUnfiled12)**

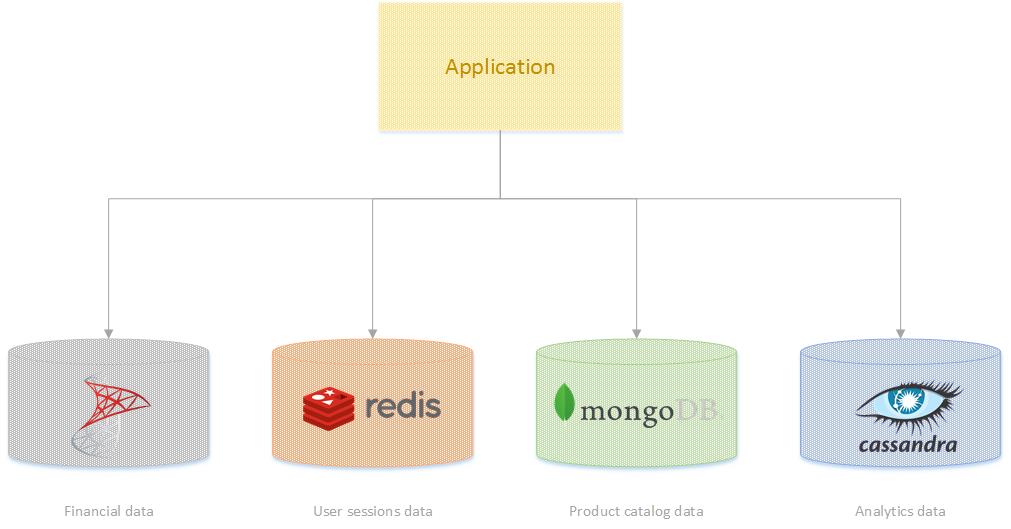
The term Polyglot Persistence was coined by Neal Ford in 2006 to express the idea that applications should be written in mixed languages. As we all know that different problems arise in all the applications. So, when an application is written using different languages, then those languages can be used to solve or tackle with different kind of problems. This is known as polyglot persistence. Picking the right language for a particular problem can be more productive rather than trying to fit all the aspects of that problem into a single language. Hence, polyglot persistence is the term which is used to define this hybrid approach to persistence.

## Polyglot Persistence

Basically, there are two main reasons why engineers choose NoSQL databases for their problems:

* **Minimizing the impedance mismatch** – This effectively entails an increase in developers’ productivity. A lot of effort is spent on mapping data between in-memory data structures and a relational database. Sometimes, a NoSQL database has a data model that fits better in the needs of our application, thus simplifying interaction of application code with the database. This way, we have less code to develop and maintain. For example, in MEAN stack (M is for Mongo DB), the whole stack uses JSON objects, and interaction of application code and database is minimal.
* **Embracing large-scale data** – Today, it is expensive to store a large amount of data in the relational databases. Businesses today have a need for capturing and processing a lot of data more quickly. Because many NoSQL databases are designed to run on clusters, they are a better fit for this kind of problem. The large-scale clusters give us the possibility to store larger data sets and to process large amounts of analytic data. Also, NoSQL databases have different data models that may be better for processing that huge amount of data.

Does this mean that relational databases are dead? No, not at all. The relational data model is still the best choice for a great number of problems out there. Apart from that, relational databases have also been here for decades, meaning there are a bunch of tools for them and people are familiar with them, in comparison to the fairly new concept of NoSQL databases.



The only difference is in the way that we should perceive relational databases. They are no longer the only option for data storage. Now we need to understand the nature of the data and use different data stores in different situations. Martin Flower calls this point of view as Polyglot Persistence – [NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence](https://www.amazon.com/NoSQL-Distilled-Emerging-Polyglot-Persistence/dp/0321826620). This way of looking at data storage will lead us to solutions that will have multiple databases, and each database will be used for a different purpose. For example, we can use SQL for a financial part of the application but use MongoDB for products catalog and Cassandra for large-scale analytics.