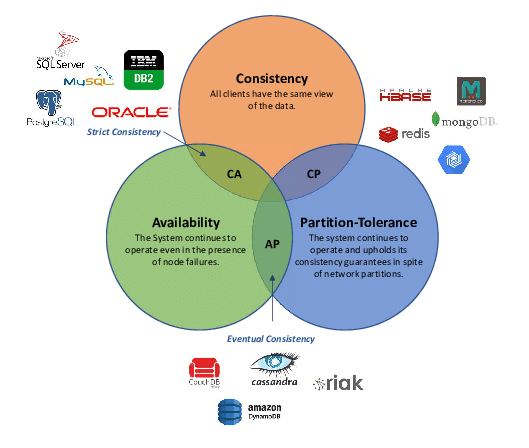
**CAP Theorem**



PACELC

**P**artition a system can favour either **A**vailability or **C**onsistency **E**lse you can favour either **L**atency or **C**onsistency

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Diagram  Description automatically generated | |  |  |  |  |  | | --- | --- | --- | --- | --- | | **DDBS** | **P+A** | **P+C** | **E+L** | **E+C** | | DynamoDB | Yes |  | Yes   |  | | --- | | [[a]](https://en.wikipedia.org/wiki/PACELC_theorem#cite_note-lctradeoff-6) | |  | | Cassandra | Yes |  | Yes   |  | | --- | | [[a]](https://en.wikipedia.org/wiki/PACELC_theorem#cite_note-lctradeoff-6) | |  | | Cosmos DB | Yes |  | Yes   |  | | --- | | [[b]](https://en.wikipedia.org/wiki/PACELC_theorem#cite_note-csmsclvl-8) | |  | | Couchbase |  | Yes | Yes | Yes | | Riak | Yes |  | Yes   |  | | --- | | [[a]](https://en.wikipedia.org/wiki/PACELC_theorem#cite_note-lctradeoff-6) | |  | | VoltDB/H-Store |  | Yes |  | Yes | | Megastore |  | Yes |  | Yes | | BigTable/HBase |  | Yes |  | Yes | | MySQL Cluster |  | Yes |  | Yes | | MongoDB | Yes |  |  | Yes | | PNUTS |  | Yes | Yes |  | | Hazelcast IMDG[7][5] | Yes | Yes | Yes | Yes | | [FaunaDB[8]](https://en.wikipedia.org/wiki/PACELC_theorem#cite_note-10) |  | Yes | Yes | Yes | |

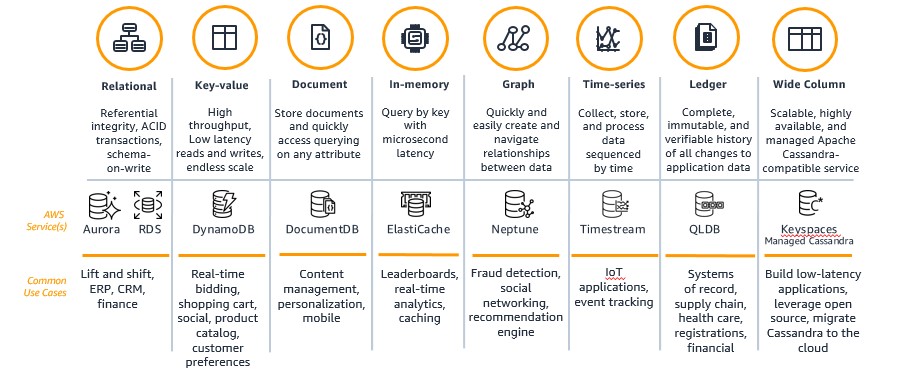
PIE Theorem

* **Pattern Flexibility**
* **Efficiency**
* **Infinite Scale**

**Diagram

Description automatically generated**

Choosing Right Database



Diagram

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Table

Description automatically generated

Logo, company name

Description automatically generated

Table

Description automatically generated

A picture containing calendar

Description automatically generated

**Google Spanner Features**

* Relational database, built for scale
* 99.999% availability
* Automatic sharding
* Fully managed
* Strong transactional consistency
* Regional and multi-regional configurations
* Online schema changes with no downtime
* Built on Google Cloud network
* Enterprise-grade security
* Backup and restore, point-in-time recovery (PITR)
* Multi-language support
* Real-time change data capture and replication

Graphical user interface, application

Description automatically generated

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **RDBMS** | **Key Value Pair** | **Wide Column** | **New SQL** |
| Open Source | MySQL | Mongo DB | Cassandra | YugabyteDB, CockroachDB |
| AWS | Aurora/RDS | Dynamo DB |  |  |
| GCP | Cloud SQL | Datastore | Big Table | Spanner |
| Schema on write | Y | N | N | Y |
| Schema on read | N | Y | Y | N |
| Design Flexibility | Y | N | N | Y |
| Query Pattern Flexibility | Y | N | N | Y |
| ACID Compliance | Y | N | N | Y |
| Joins | Y | N | N | Y |
| Field specific Update | Y | N | Y | Y |
| Where Clause | Y | N | Y | Y |
| Non structured Data | N | Y | Y | N |
| Horizontal Scalability | N | Y | Y | Y |

## Understanding the Features of NoSQL Databases

NoSQL databases offer numerous features over traditional databases. We have listed a few of the most popular features of NoSQL databases.

### 1. Schemaless Tables

NoSQL databases are schema-less and can store heterogeneous data from the same domain easily. Users can quickly load complex schemas and heterogeneous data in the same NoSQL documents or tables.

### 2. Non-Relational Structure

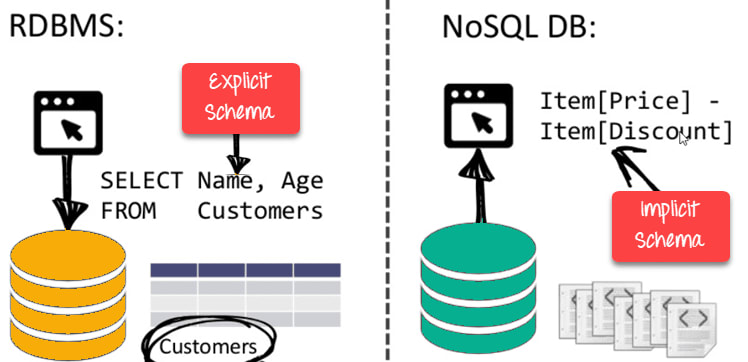
NoSQL databases don’t rely on relational models, and it doesn’t store the data in flat fixed schemas. It also doesn’t support complex features like query language, integrity-joins, ACID operations, etc. The above-listed points make the NoSQL database popular among the BigData and real-time fields.

### 3. Simple API Controls

NoSQL databases offer easy-to-use API interfaces to allow low-level data manipulation. They are very well versed with REST endpoints.

### 4. Distributed Computing

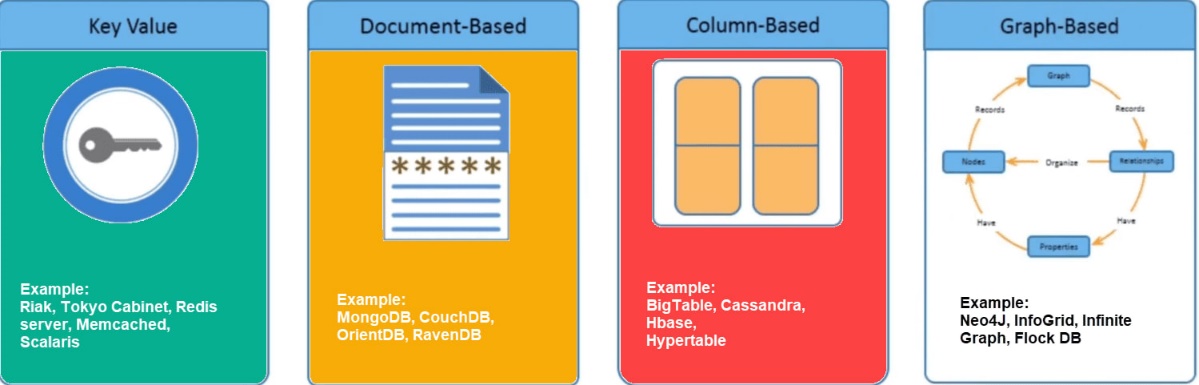
NoSQL databases offer distributed processing of queries along with auto-scaling and failover mechanisms.

Image Source: [Guru99](https://www.guru99.com/images/1/101818_0537_NoSQLTutori3.png)

## Understanding the Types of NoSQL Databases

There are four different categories of NoSQL Databases:

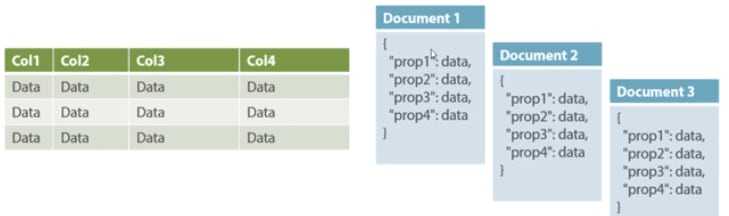
* [Document Databases](https://hevodata.com/learn/nosql-databases-and-its-types-a-guide/#s1)
* [Key-Value Stores](https://hevodata.com/learn/nosql-databases-and-its-types-a-guide/#s2)
* [Column-Oriented Databases](https://hevodata.com/learn/nosql-databases-and-its-types-a-guide/#s3)
* [Graph Databases](https://hevodata.com/learn/nosql-databases-and-its-types-a-guide/#s4)

Image Source: [Guru99](https://www.guru99.com/images/1/101818_0537_NoSQLTutori5.png)

Let’s discuss each of them in detail.

### **1. Document Databases**

Document Databases use key-value pairs to store and retrieve data from the documents. A document is stored in the form of XML and JSON. A typical example of the document database is shown below:

Image Source: [Guru99](https://www.guru99.com/)

The above figure shows that the Document database contains data in JSON (or XML) format and can contain varying schema. The Documents can be nested and indexed for faster querying.

Document databases allow developers to restructure their Documents based on their application requirements which may change over time. In contrast, in the RDBMS world, database administrators are required to restructure the database schemas.

Examples of Document databases are – MongoDB, OrientDB, Apache CouchDB, IBM Cloudant, CrateDB, BaseX, and many more.

### **2. Key-Value Stores**

Key-value Stores are the simplest type of NoSQL database. It uses keys and values to store the data. The attribute name is stored in ‘key’, whereas the values corresponding to that key will be held in ‘value’.

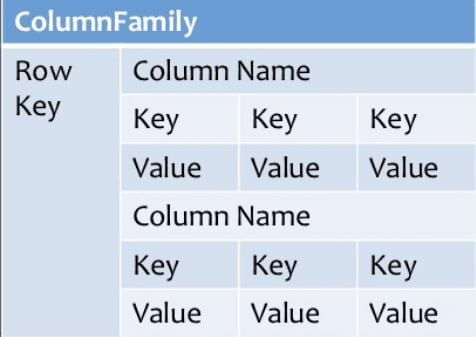
In Key-value store databases, the key can only be string, whereas the value can store string, JSON, XML, Blob, etc. Due to its behavior, it is capable of handling massive data and loads.

The use case of key-value stores mainly stores user preferences, user profiles, shopping carts, etc.

DynamoDB, Riak, Redis are a few famous examples of Key-value store NoSQL databases.

### **3. Column-Oriented Databases**

Column-oriented databases store the data in a set of columns known as column families. That means that whenever a user wants to run queries for a smaller number of columns, they can read those columns directly without consuming memories corresponding to all data. The working of Column-oriented database is based on the concept of the BigTable paper by Google. Below schematics shows how values are stored on Column-oriented databases:

Image Source: [Guru99](https://www.guru99.com/images/1/101818_0537_NoSQLTutori7.png)

HBase, Cassandra, HBase, Hypertable are NoSQL query examples of column-based databases.

### **4. Graph Databases**

Graph databases form and store the relationship of the data. Each element/data is stored in a node, and that node is linked to another data/element. A typical example for Graph database use cases is Facebook. It holds the relationship between each user and their further connections.

Graph databases help search the connections between data elements and link one part to various parts directly or indirectly.

Diagram

Description automatically generatedImage Source: [Guru99](https://www.guru99.com/images/1/101818_0537_NoSQLTutori9.png)

The Graph database can be used in social media, fraud detection, and knowledge graphs. Examples of Graph Databases are – Neo4J, Infinite Graph, OrientDB, FlockDB, etc.

## Understanding the Advantages of NoSQL Databases

* NoSQL database is optimum for processing massive volume data with distributed processing.
* NoSQL database supports failover mechanisms and ensures high availability.
* NoSQL database provides easy replication along with horizontally scalable capability.
* NoSQL database is capable of handling structured, semi-structured, and unstructured data.
* NoSQL databases can be installed on commodity hardware and can form clusters for distributed processing.
* NoSQL database offers flexible schema and can be changed at runtime without service downtime.

## Understanding the Disadvantages of NoSQL Databases

In the above sections, we have discussed a lot about NoSQL databases and their benefits. However, there are certain limitations to the NoSQL database, which we have to look upon. Below are the few listed limitations:

* NoSQL databases have limited query capabilities as compared to RDBMS.
* NoSQL databases don’t offer any RDBMS capabilities like consistency and ACID transactions.
* Most of the NoSQL databases use key-value pairs to store the data. Hence it isn’t easy to maintain as the volume increases.
* NoSQL databases are new to the markets and can be challenging for RDBMS programmers to switch to these technologies.
* Most of the NoSQL databases are open source and are a restricted choice for enterprises.

# [What is the problem with key-value databases and how wide column stores solve it.](https://indexoutofrange.com/What-is-the-problem-with-key-value-databases-and-how-wide-column-databases-solve-it/)

# Problems with key-value databases

The central concept of a key-value database is that the database doesn’t care what the is value. It may have some assumptions, [like Redis](https://indexoutofrange.com/Want-unlimited-scale-and-performanceThis-is-where-to-start#Redis), but the structure of the data is not of its interest. This leads to some limitations that can be problematic in some scenarios.

## 1. Can’t filter on value fields

Quite evident since from the database point of view value is a blob.

## 2. The whole value is returned

This may not seem as a problem, but remember, key-value databases are chosen for speed. When looking at a flow of retrieving data from a database:

Diagram

Description automatically generated

Most of the steps performance is dependent on the **size of the transmitted data, not the data used** (this is why SELECT \* is a sign of someone not giving a f\*\*k about performance).

## 3. The value can be updated only as a whole

It is a problem because we have to:

* get the complete data to the client (see point above)
* operate on the entire data
* send the whole data back to the database

Sounds not that bad, even good. We want to have the whole object when updating it, right?

Think about those cases:

* update users last login date
* append the element to the list (like online checkout basket)
* change the prices on certain products because of a promotion

**So how to solve those problems and not loose all that speed?**

# Wide column databases

The idea behind it is simple:

Let's structure data (that is the value part) again into key-value pairs.

This is hat we have in key-value databases:

Shape, rectangle

Description automatically generated

This is how wide column databases represent data:

A picture containing table

Description automatically generated

Having columns allows defining a subset of data we want to return to the client or subset of data that should be updated.

## How is it different from a regular table in a relational database?

In most wide column databases, columns are defined on the **single item level** meaning there is no database-wide schema, which leads us to some interesting features of wide column databases:

## Benefits from wide column databases

While wide column databases keep most of the [perks of key-value databases mentioned previously](https://indexoutofrange.com/Want-unlimited-scale-and-performanceThis-is-where-to-start/) they have some additional ones (They don’t have to exist in every implementation of a wide-column database, but most of them have it):

* **partial operations** (add column value, update column value).
* **data compression**. When dealing with sparse data we don’t have to store empty/null values. This way we can save space normally reserved because schema defined them.
* **WHERE clause**. This feature is not that common in this type of databases, but filtering on data starts to appear.

**Oracle Vs PostgreSQL**

|  |  |  |
| --- | --- | --- |
|  | Oracle | PostgreSQL |
| Architecture | Shared Disk | Shared Nothing |
| High Availability | RAC | Stand By |
| Read Performance | RAC | Read Replica |
| Write Performance | RAC |  |
| DR | Data Guard | Logical replication using WAL |
| Soft Parsing | Shared Pool |  |
| Security | TDE |  |
| Storage Management | ASM | Relies on OS |
| Encryption at Rest | AES-256 | Relies on OS |
|  |  |  |
|  |  |  |