Relational databases enforces [ACID](http://en.wikipedia.org/wiki/ACID). So, you will have schema based transaction oriented data stores.

But, there are limitations on speed and scaling when it comes to massive high availability data stores. For example, Google and Amazon have terabytes of data stored in big data centers. Querying and inserting is not performant in these scenarios because of the blocking/schema/transaction nature of the RDBMs. That's the reason they have implemented their own databases (actually, key-value stores) for massive performance gain and scalability.

Why NoSQL appeared?

* SQL (relational) databases was not designed to be horizontally scalable
* NoSQL databases use a flexible schema model (according to the company needs)
* They provide flexible schemas and scale easily with large amounts of big data and high user loads.
* 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 in order to avoid data duplication. NoSQL databases optimized for developer productivity.

The term NoSQL, short for “not only SQL,” refers to non-relational databases that store data in a non-tabular format, rather than in rule-based, relational tables like [relational databases](https://cloud.google.com/learn/what-is-a-relational-database) do. NoSQL databases use a flexible schema model that supports a wide variety of unstructured data such as documents, key-value, wide columns, and graphs.

**5 types of NoSQL databases**

There are five main types of NoSQL databases:

**Document databases**

Document databases, also called document-oriented databases or a document store, are used to store and query semi-structured data. Data is stored in a JSON-like document similar to the data objects that developers use in application code, making it easier to create and update applications without referencing a primary schema. Document databases are most commonly used for blogging platforms, ecommerce, real-time analytics, and content management systems.

**Key-value databases**

Key-value databases, also referred to as key-value stores, are the simplest type of NoSQL databases. Data is stored in a “key-value” structure, where a unique key is paired with a value such as a string, number, boolean, or complex objects. You can use the key to store or retrieve its associated value. Key-value stores are most commonly used for user preferences, shopping carts, and user profiles in web applications.

**Column-oriented databases**

Column-oriented databases, or wide-column stores, store and read data in rows and are organized as a set of columns. While similar to the tabular format of relational databases, column names and formatting in wide-column stores can vary from row to row in a single table. They are optimal for analytics use cases, where you may need to query across specific columns in a database and aggregate the value of a given column quickly. Wide-column stores are most commonly used for catalogs, fraud detection, and recommendation engines.

<https://scaleyourapp.com/wide-column-and-column-oriented-databases/>

**Graph databases**

Graph databases organize data as nodes in a graph, focusing on the relationships between data elements. The connections between nodes (edges) are stored as first-class elements, enabling richer representations of data relationships while offering more simplified storage and navigation. Graph databases are most commonly used in systems that map relationships, including social media platforms, reservation systems, fraud detection systems, and logistics applications.

**In-memory databases**

In-memory databases store data in memory in order to provide ultra-low latency for real-time applications. Redis and Valkey are examples of in-memory NoSQL databases. In-memory databases are most commonly used for caching, messaging, streaming, and real-time analytics.

**Comparison of Data Models**

| **Feature** | **Row-Oriented** | **Column-Oriented** | **Column-Family** |
| --- | --- | --- | --- |
| **Storage** | Rows stored contiguously | Columns stored contiguously | Rows with grouped columns |
| **Optimized For** | OLTP (transactions) | OLAP (analytics) | Mixed workloads |
| **Read Efficiency** | Row-based queries | Column-based aggregations | Flexible column family reads |
| **Write Efficiency** | Fast for inserts/updates | Slow for frequent writes | Scalable writes |
| **Schema Flexibility** | Fixed schema | Fixed schema | Flexible schemas |
| **Examples** | MySQL, PostgreSQL | BigQuery, Redshift | Bigtable, Cassandra |