NoSQL Database

NoSQL is a type of database management system (DBMS) that is designed to handle and store large volumes of unstructured and semi-structured data. Unlike traditional relational databases that use tables with pre-defined schemas to store data, NoSQL databases use flexible data models that can adapt to changes in data structures and are capable of scaling horizontally to handle growing amounts of data.

The term NoSQL originally referred to "non-SQL" or "non-relational" databases, but the term has since evolved to mean "not only SQL," as NoSQL databases have expanded to include a wide range of different database architectures and data models.

NoSQL databases are generally classified into four main categories:

1. **Document databases:** These databases store data as semi-structured documents, such as JSON or XML, and can be queried using document-oriented query languages.
2. **Key-value stores:** These databases store data as key-value pairs, and are optimized for simple and fast read/write operations.
3. **Column-family stores:** These databases store data as column families, which are sets of columns that are treated as a single entity. They are optimized for fast and efficient querying of large amounts of data.
4. **Graph databases:** These databases store data as nodes and edges, and are designed to handle complex relationships between data.

NoSQL databases are often used in applications where there is a high volume of data that needs to be processed and analyzed in real-time, such as social media analytics, e-commerce, and gaming. They can also be used for other applications, such as content management systems, document management, and customer relationship management.

However, NoSQL databases may not be suitable for all applications, as they may not provide the same level of data consistency and transactional guarantees as

traditional relational databases. It is important to carefully evaluate the specific needs of an application when choosing a database management system.

Key Features of NoSQL :

1. **Dynamic schema:** NoSQL databases do not have a fixed schema and can accommodate changing data structures without the need for migrations or schema alterations.
2. **Horizontal scalability:** NoSQL databases are designed to scale out by adding more nodes to a database cluster, making them well-suited for handling large amounts of data and high levels of traffic.
3. **Document-based:** Some NoSQL databases, such as MongoDB, use a document-based data model, where data is stored in semi-structured format, such as JSON or BSON.
4. **Key-value-based:** Other NoSQL databases, such as Redis, use a key-value data model, where data is stored as a collection of key-value pairs.
5. **Column-based:** Some NoSQL databases, such as Cassandra, use a column-based data model, where data is organized into columns instead of rows.
6. **Distributed and high availability:** NoSQL databases are often designed to be highly available and to automatically handle node failures and data replication across multiple nodes in a database cluster.
7. **Flexibility:** NoSQL databases allow developers to store and retrieve data in a flexible and dynamic manner, with support for multiple data types and changing data structures.
8. **Performance:** NoSQL databases are optimized for high performance and can handle a high volume of reads and writes, making them suitable for big data and real-time applications.

Advantages of NoSQL: There are many advantages of working with NoSQL databases such as MongoDB and Cassandra. The main advantages are high scalability and high availability.

1. **High scalability :** NoSQL databases use sharding for horizontal scaling. Partitioning of data and placing it on multiple machines in such a way that the order of the data is preserved is sharding. Vertical scaling means adding more

resources to the existing machine whereas horizontal scaling means adding more machines to handle the data. Vertical scaling is not that easy to implement but horizontal scaling is easy to implement. Examples of horizontal scaling databases are MongoDB, Cassandra, etc. NoSQL can handle a huge amount of data because of scalability, as the data grows NoSQL scale itself to handle that data in an efficient manner.

2. **Flexibility:** NoSQL databases are designed to handle unstructured or semi-structured data, which means that they can accommodate dynamic changes to the data model. This makes NoSQL databases a good fit for applications that need to handle changing data requirements.
3. **High availability :** Auto replication feature in NoSQL databases makes it highly available because in case of any failure data replicates itself to the previous consistent state.
4. **Scalability:** NoSQL databases are highly scalable, which means that they can handle large amounts of data and traffic with ease. This makes them a good fit for applications that need to handle large amounts of data or traffic
5. **Performance:** NoSQL databases are designed to handle large amounts of data and traffic, which means that they can offer improved performance compared to traditional relational databases.
6. **Cost-effectiveness:** NoSQL databases are often more cost-effective than traditional relational databases, as they are typically less complex and do not require expensive hardware or software.
7. **Agility:** Ideal for agile development.

Disadvantages of NoSQL:

NoSQL has the following disadvantages.

1. **Lack of standardization** : There are many different types of NoSQL databases, each with its own unique strengths and weaknesses. This lack of standardization can make it difficult to choose the right database for a specific application
2. **Lack of ACID compliance :** NoSQL databases are not fully ACID-compliant, which means that they do not guarantee the consistency, integrity, and durability

of data. This can be a drawback for applications that require strong data consistency guarantees.

3. **Narrow focus :** NoSQL databases have a very narrow focus as it is mainly designed for storage but it provides very little functionality. Relational databases are a better choice in the field of Transaction Management than NoSQL.
4. **Open-source :** NoSQL is open-source database. There is no reliable standard for NoSQL yet. In other words, two database systems are likely to be unequal.
5. **Lack of support for complex queries :** NoSQL databases are not designed to handle complex queries, which means that they are not a good fit for applications that require complex data analysis or reporting.
6. **Lack of maturity :** NoSQL databases are relatively new and lack the maturity of traditional relational databases. This can make them less reliable and less secure than traditional databases.
7. **Management challenge :** The purpose of big data tools is to make the management of a large amount of data as simple as possible. But it is not so easy. Data management in NoSQL is much more complex than in a relational database. NoSQL, in particular, has a reputation for being challenging to install and even more hectic to manage on a daily basis.
8. **GUI is not available :** GUI mode tools to access the database are not flexibly available in the market.
9. **Backup :** Backup is a great weak point for some NoSQL databases like MongoDB. MongoDB has no approach for the backup of data in a consistent manner.
10. **Large document size :** Some database systems like MongoDB and CouchDB store data in JSON format. This means that documents are quite large (BigData, network bandwidth, speed), and having descriptive key names actually hurts since they increase the document size.

Types of NoSQL database: Types of NoSQL databases and the name of the databases system that falls in that category are:

1. **Graph Databases**: Examples – Amazon Neptune, Neo4j
2. **Key value store:** Examples – Memcached, Redis, Coherence
3. **Tabular:** Examples – Hbase, Big Table, Accumulo

4. **Document-based:** Examples – MongoDB, CouchDB, Cloudant

When should NoSQL be used:

1. When a huge amount of data needs to be stored and retrieved.
2. The relationship between the data you store is not that important
3. The data changes over time and is not structured.
4. Support of Constraints and Joins is not required at the database level
5. The data is growing continuously and you need to scale the database regularly to handle the data.

Key-Value (KV) Stores

This is the simplest type of NoSQL database. Under this type, the data is stored in the form of key/value pairs. For each Key, there is a value assigned to it. Each Key is unique and accepts only strings, whereas the value corresponding to the particular Key can accept String, JSON, XML, etc. Owing to this behavior, it is capable of dealing with massive loads of data.

Key Value Stores maintain data as pair consisting of an index key and a value. KV stores query Values using the index Key. Every item in the database is stored in the pairs of Keys (Indexes) and Values. KV stores resemble a relational database but with each table having only two columns.Some KV stores may even allow basic joins to help you scan through if there are composite joins, they may not be a suitable options.

There are multiple KV Stores available, each differing mainly in their adaption of the CAP theorem and their configurations of memory v/s storage usage. KV stores have fast query performance and are best suited for applications that require content caching, e.g. a gaming website that constantly updates the top 10 scores & players. NoSQL key value stores example

Key	Value
Name	Raman Sharma
Account Number	21657243
Amount	567543

Key-Value Pairs Database:

Features:

- Consistency
- Transactions
- Query Features
- Data Structure
- Scaling

Advantages:

- Simple Data model
- Scalable
- Value can include JSON, XML, flexible schemas
- Extremely Fast Owing to it's simplicity
- Best fit for cases where data is not highly related

Disadvantages:

- No relationships, create your own foreign keys
- Not suitable for complex data
- Lacks Scanning Capabilities
- Not ideal for operations rather than CRUD (create, read, update Delete)

Key-Value Pairs Database: Use Case:

These kinds of databases are best suited for the following cases:

Storing session information: offers to save and restore sessions.

User preferences: Specific Data for a particular user

Shopping carts: easily handle the loss of storage nodes and quickly scale Big data during a holiday/sale on an e-commerce application.

Product recommendations: offering recommendations based on the person's data. Popular KV Stores would include Dynamo DB, Redis, BerkleyDB.

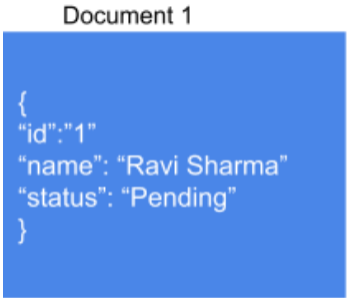
Document Stores

Document Stores are an extension of the simplicity of Key Value stores, where the values are stored in structured documents like XML or JSON. Document stores make it easy to map Objects in the object- oriented software.

A document database is schema free, you don't have to define a schema beforehand and adhere to it. It allows us to store complex data in document formats (JSON, XML etc.). Document databases do not support relations. Each document in the document store is independent and there is no relational integrity.

Document stores can be used for all use cases of a KV store database, but it also has additional advantages like there is no limitation of querying just by the key but even querying attributes within a document, also data in each document can be in a different format. E.g. A product review website where zero or many users can review each product and each review can be commented on by other users and can be liked or disliked by zero to many users.

E.g, A product review website where zero or many users can review each product, and each review can be commented on by other users and can be liked or disliked by zero to many users.



Document Stores:

Features:

- Features of document databases
- Faster Querying
- A large amount of data can be easily handled owing to its structure
- Flexible Indexing

Advantages:

- Simple & Powerful Data model
- Scalable
- Open Formats
- No foreign Keys

Disadvantages:

- Not suitable for relational data
- Querying limited to keys & indexes
- Map Reduce for more significant queries

Document Stores: Use Case:

User Profiles: Since they have a flexible Schema, the document can store different attributes and values. This enables the users to store different types of information.

Management of Content: Since it has a flexible schema, collection and storing any data has been made possible. This allows the creation of new types of content, including images, videos, comments, etc. Everyday use is seen in blogging platforms.

Some popular Document stores are MongoDB, CouchDB, Lotus Notes.

Column Family Data stores or Wide column data stores

Wide column data stores take a hybrid approach mixing the declarative characteristics game of relational databases with the key-value pair based and totally variables schema of key-value stores. Wide Column databases stores data tables as sections of columns of data rather than as rows of data.

Columnar Family databases have their origins in Google's Bigtable. According to Google's paper on Bigtable, "A Bigtable is a sparse, distributed, persistent multidimensional sorted map." This definition might leave you confused, just as I was, it was all greek to my RDBMS oriented mind.

Here is a more simplified explanation, a column family data store is a multi-dimensional key value store (map or associative array) which is persistent (values persist after creation or access), distributed (data is distributed across multiple computing & storage nodes), sorted (sorted keys) and sparse (values for certain dimensions may not be populated, similar to sparsely populated rows in RDBMS).

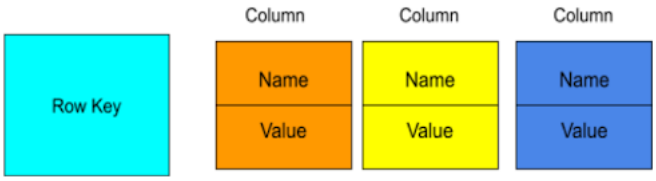
The multi-dimensional aspect of column stores brings in another concept of column families.

Column-family databases store data in column families as rows that have many columns associated with a row key. Column families are groups of related data that is often accessed together.

There are two types of column families:

Standard Column family: Standard Column family consists of a key-value pair, where the key is mapped to a value that is a set of columns. In analogy with relational databases, a standard column family is as a “table”, each key-value pair being a “row”.

Super Column family: Super Column family consists of a key-value pair, where the key is mapped to a value that are column families. In analogy with relational databases, a super column family is something like a “view” on a number of tables. It can also be seen as a map of tables.



Column Oriented Database:

Features:

- Features of Wide column stores:
- Multidimensional key store
- Persistent in nature
- Distributed
- High flexibility

Advantages:

- Supports semi-structured data
- Naturally indexed
- Scalable

Disadvnatges:

- Not suitable for relational data

Column Based Databases: Use Case:

User Preferences

Business Intelligence

Managing data warehouses

Reporting Systems

Some of the popular Wide column data stores include Google's Bigtable, Cassandra, HBase.

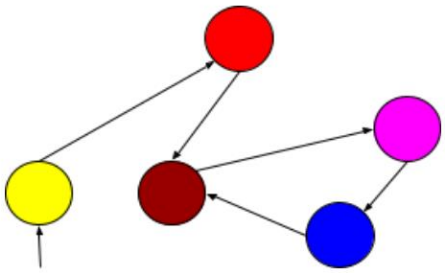
Note, wide column data stores are not to be confused with column-oriented databases, I'll may be cover this in a separate post.

Graph Databases

Graph Databases specific purpose is the storage of graph-oriented data structures. A graph database is any storage system that provides index-free adjacency. This means that every node contains a direct pointer to its adjacent element and no index lookups are necessary. As the number of nodes increases, the cost of a hop remains the same.

Graph databases are optimized for traversing through connected data, e.g. traversing through a list of contacts on your social network to find out the degree of connections.

Graph databases usually come with a flexible data model, which means there is no need to define the types of edges and vertices.



Graph Databases:

Features:

- Features of graph databases
- Flexibility
- Agility
- Improved performance, even with huge volumes of data.

Advantages:

- Extremely powerful
- Connected data is locally indexed
- Can provide ACID
- Results in real-time
- Agile Structure

Disadvantages:

- Difficult to scale out, though can scale up

Graph Databases: Use Case:

Typical use cases for graph databases would include

Social networking site

Recommendation engine

Logistics

Risk assessment

Fraud detection

Some of the popular graph databases are Neo4j, OrientDB, Allegrograph.