Non Relational Database

NoSQL is a type of non-relational database that is designed to handle large amounts of unstructured data and scale easily. Unlike relational databases, which use a predefined schema and structured queries, NoSQL databases offer more flexibility and allow for dynamic schemas. This makes them well-suited for storing a wide variety of data types, such as social media posts, video content, and articles.

Features of NoSQL

Multi-Model : Unlike relational databases, where data is stored in relations, different data models in NoSQL databases make them flexible to manage data.

Distributed : NoSQL databases use the shared-nothing architecture, implying that the database has no single control unit or storage. The advantage of using a distributed database is that data is continuously available because data remains distributed between multiple copies. On the contrary, Relational Databases use a centralized application that depends on the location.

Flexible Schema : Unlike relational databases where data is organized in a fixed schema, NoSQL databases are quite flexible while managing data. While relational databases were built typically to manage structured data, NoSQL databases can process structured, semi-structured or unstructured data with the same ease, thereby increasing performance.

Eliminated Downtime : One of the essential features is the eliminated downtime. Since the data is maintained at various nodes owing to its architecture, the failure of one node will not affect the entire system.

High Scalability : One of the reasons for preferring NoSQL databases over relational databases is their high scalability. Since the data is clustered onto a single node in a relational database, scaling up poses a considerable problem. On the other hand, NoSQL databases use horizontal scaling, and thus the data remains accessible even when one or more nodes go down.

Here is an explanation for each type of NoSQL DB.

1. 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.

Key-Value Pairs Database: Features:

Consistency
Transactions
Query Features
Data Structure
Scaling

Pros:

- 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

Cons:

- 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.

2. Document Stores

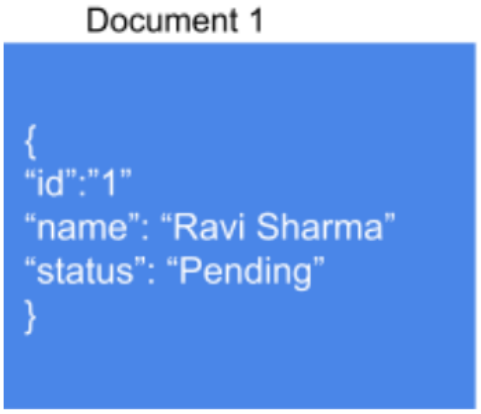
Document Stores are an extension of the simplicity of Key Value stores, where the values are stored in structured documents like XML or JSON. Document stores make it easy to map Objects in the object- oriented software.

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

Pros:

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

Cons:

- 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.

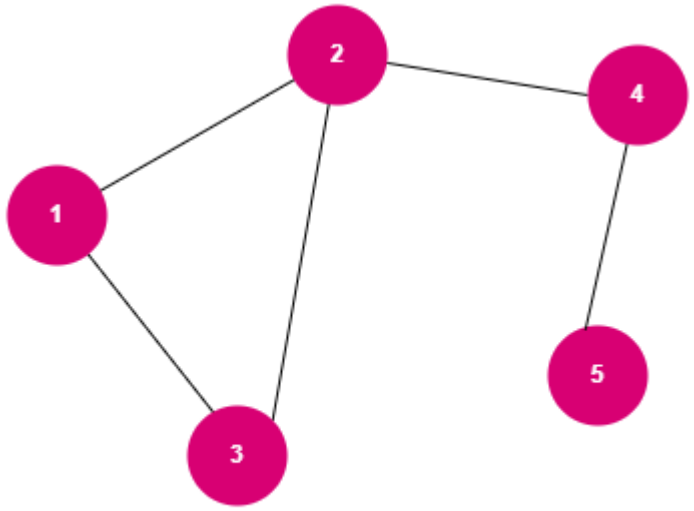
Management of Content: Since it has a flexible schema, collection and storing any data has been made possible. This allows the creation of new types of content, including images, videos, comments, etc. Everyday use is seen in blogging platforms.

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

3. Graph Databases

What is a Graph?

A graph is a collection of nodes that are connected by edges. An edge always connects 2 nodes.



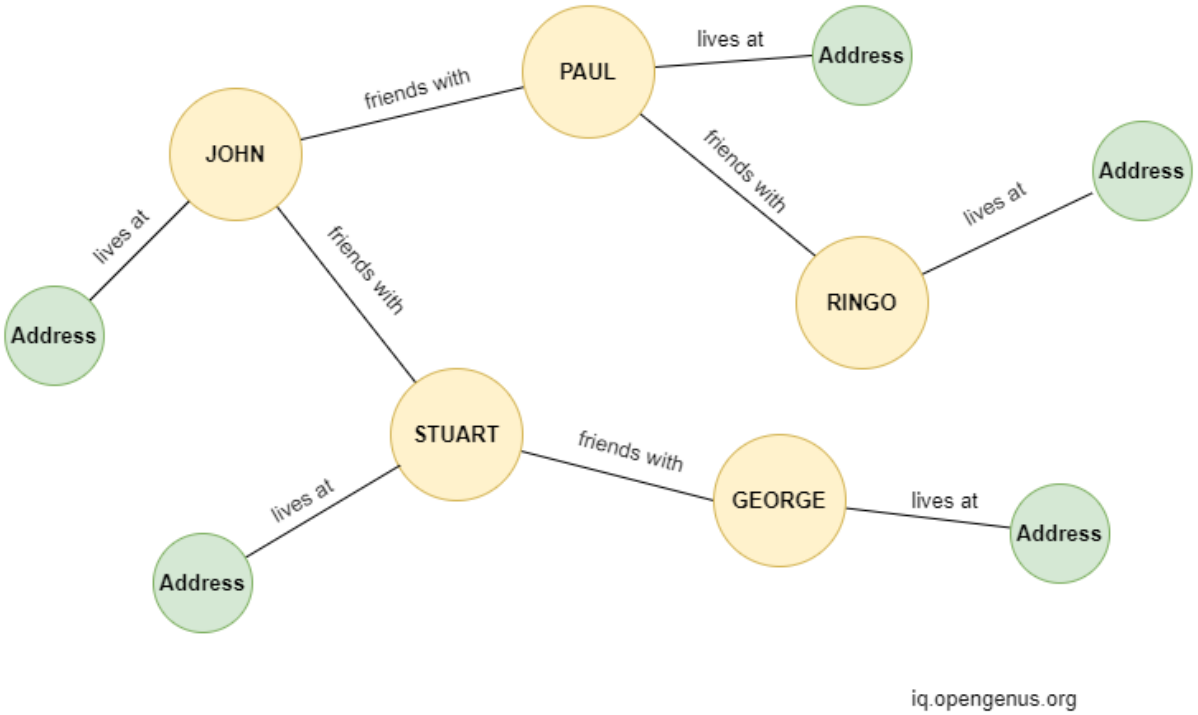
iq.opengenus.org

In the above image, 1,2,3,4 and 5 are called the nodes or vertices of the graph.

They are connected by using edges. The edges can be weighted (have a value) and can be uni-directional depending upon the type of graph.

Graph Databases

A graph database is simply a database that makes use of the graph data model. That is, it stores the data in nodes, and the relationship among the data is given by edges. For example, a graph database may look like this,



This graph database describes a social network, with each person having an address where they live and an edge to those they are friends with.

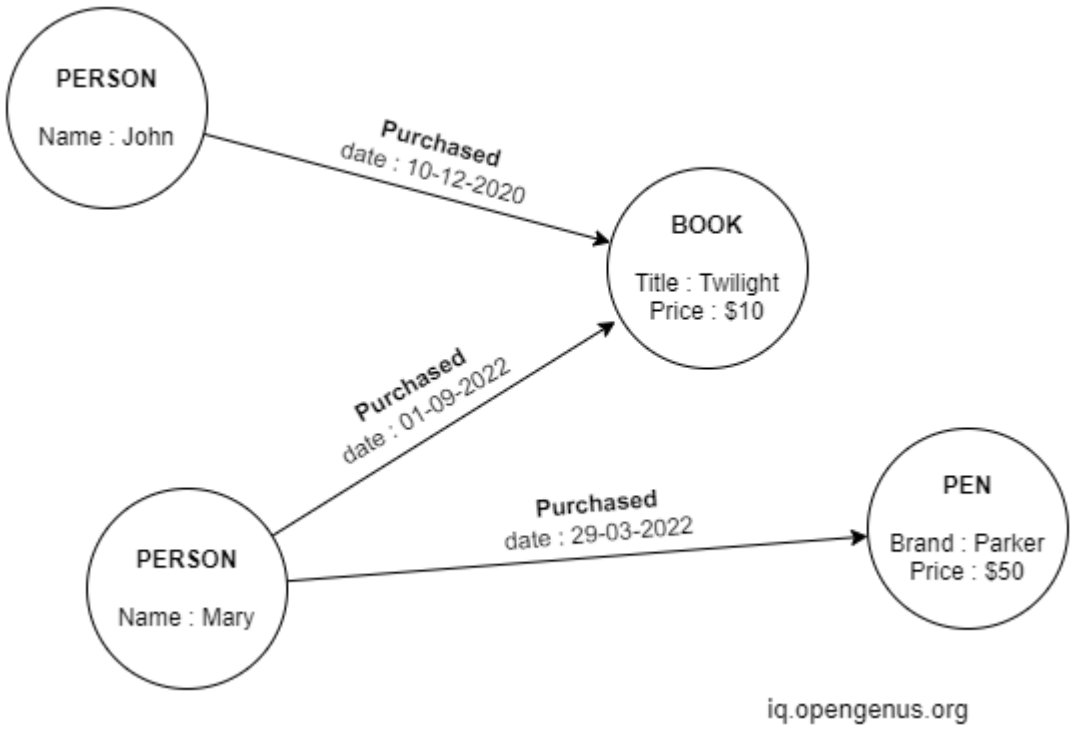
Types of Graph Databases

There are 2 main types of graph databases.

1. Property Graphs
2. Resource Description Framework

Property Graphs:

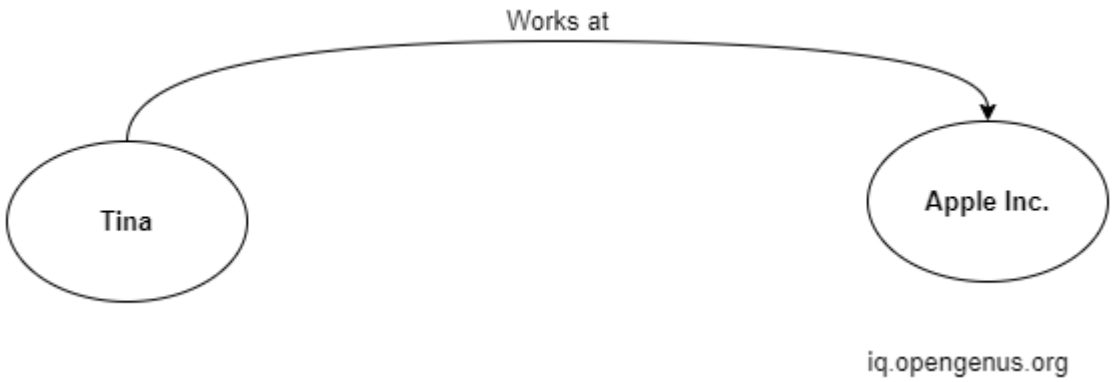
In a property graph, the nodes and vertices may have attributes, called properties. They are commonly used to model relationships among data. Then enable easy querying and analytics of data.



Resource Description Framework:
Resource Description Framework is also known as RDF graph database is a W3C standard originally designed for encoding, exchange, and reuse of structured metadata. It has since become a general method for describing and exchanging graph data.

An RDF consists of 3 components referred to as a triple,

1. The subject is the resource which is a resource that is being described.
2. The predicate is the relationship between the subject and object.
3. The Object is the resource to which the subject is linked.



Relational Databases and the Need for Graph Databases

Oftentimes, the data that we encounter in the real world is highly relational and requires complex traversals. In traditional relational databases, we separate the data into different tables to reduce insertion, update, and deletion anomalies.

But what happens when we need to study the relationships among various data or combine some tables to get a new result? This problem can be solved in relational databases by using deep joins which can be very complex. Further, as the database grows, joins can take a lot of time and slow down the performance of the system.

Enter graph databases. They store our data in a much more logical fashion and in a way that represents the real world. Adding new nodes or adding a new data value to an existing node is as simple as it sounds.

Popular Graph Databases

Let us take a look at some of the most popular graph databases available and their unique features.

Neo4j

Neo4j is one of the most popular graph database management systems. It is ACID compliant and hence can be used for transactional purposes. In Neo4j everything is stored in the form of an edge, node, or attribute. Each node and edge can

have any number of attributes. Both nodes and edges can be labeled. It is written in Java but can be used in other languages through its property graph query language called Cypher.

Dgraph

Dgraph is another popular option for a graph database. Dgraph is currently the most starred graph database in Github. It natively uses GraphQL query language. Dgraph is known for its performance and scalability. It just takes milliseconds to query terabytes of data.

AWS Neptune

AWS Neptune is a fast and fully managed graph database provided by Amazon Web Services. It can perform 100,000 queries per second and is highly reliable and can perform a continuous backup to S3 storage. Neptune supports popular graph models like RDF and property graphs and their query languages like Apache TinkerPop Gremlin and SPARQL.

ArangoDB

ArangoDB is an open-source, multi-model graph database. It combines the power of graphs with JSON documents, a key-value store, and a full-text search engine allowing developers to combine all these data models and benefit from them.

Usecase - Credit Card Fraud Detection

Over the years, there have been many measures taken by financial institutions to prevent credit card fraud such as using an embedded chip in cards. However, In many places which don't use chip readers, card skimming devices card devices can still be stolen by reading the magnetic strips. Once the card details are obtained, they can be used to make duplicate cards and make purchases online or withdraw money.

Detection of card fraud usage involves finding purchase patterns based on the location and other details based on the user. Graph analytics can efficiently be used to establish patterns and identify natural behavior patterns. The graph

nodes are given by the location, transaction, accounts of cardholders, etc.

For example, It is highly unlikely that a user from the UK would suddenly make a car purchase in India through a credit card. This is a flag that the transaction could be potentially fraudulent.

Challenges in Graph Databases

1. The is no widespread standard on the query language used for graph databases. It differs based on the software used.
2. The community is small and hence it is hard to find solutions to a problem when it is encountered.
3. Graph databases need to make trade-offs in ACID properties and hence may not be suitable for mission-critical applications.

Despite these challenges, graph databases have been evolving in popularity over the years with the increase in computing power and used to model complex data relationships.

4. Column Family Data stores or Wide column data stores

How column-oriented databases work

Relational databases have a set schema, and they function as tables of rows and columns. Wide-column databases have a similar, but different schema. They also have rows and columns. However, they are not fixed within a table, but have a

dynamic schema. Each column is stored separately. If there are similar (related) columns, they are joined into column families and then the column families are stored separately from other column families.

The row key is the first column in each column family, and it serves as an identifier of a row. Furthermore, each column after that has a column key (name). It identifies columns within rows and thus enables the querying of the columns. The value and the timestamp come after the column key, leaving a trace of when the data was entered or modified.

The number of columns pertaining to each row, or their name, can vary. In other words, not every column of a column family, and thus a database, has the same number of rows. In fact, even though they might share their name, each column is contained within one row and does not run across all rows.

Column-oriented databases use vertical organization as opposed to the horizontal layout of row databases.

Those who have encountered relational databases know that each column of a relational database has the same number of rows, but it happens that some of the fields have a null value, or they appear to be empty. With wide-column databases, rather than being empty, these rows simply do not exist for a particular column.

The column families are located in a keyspace. Each keyspace holds an entire NoSQL data store and it has a similar role or importance that a schema has for a relational database. However, as NoSQL datastores have no set structure, keyspaces represent a schemaless database containing the design of a data store and its own set of attributes.

One of the most popular columnar databases available is MariaDB. It was created as a fork of MySQL intended to be

robust and scalable, handle many different purposes and a large volume of queries. Apache Cassandra is another example of a columnar database handling heavy data loads across numerous servers, making the data highly available. Some of the other names on this list include Apache HBase, Hypertable and Druid specially designed for analytics. These databases support certain features of platforms such as Outbrain, Spotify and Facebook.

Column family types

·Standard column family. This column family type is similar to a table; it contains a key-value pair where the key is the row key, and the values are stored in columns using their names as their identifiers.

·Super column family. A super column represents an array of columns. Each super column has a name and a value mapping the super column out to several different columns. Related super columns are joined under a single row into super column families. Compared to a relational database, this is like a view of several different tables within a database. Imagine you had the view of the columns and values available for a single row -- that is a single identifier across many different tables -- and were able to store them all in one place: That is the super column family.

Advantages of column-oriented databases

·Scalability. This is a major advantage and one of the main reasons this type of database is used to store big data. With the ability to be spread over hundreds of different machines depending on the scale of the database, it supports massively parallel processing. This means it can employ many processors to work on the same set of computations simultaneously.

·Compression. Not only are they infinitely scalable, but they are also good at compressing data and thus saving storage.

·Very responsive. The load time is minimal, and queries are performed fast, which is expected given that they are designed to hold big data and be practical for analytics.

Disadvantages of column-oriented databases

·Online transactional processing. These databases are not very efficient with online transactional processing as much as they are for online analytical processing. This means they are not very good with updating transactions but are designed to analyze them. This is why they can be found holding data required for business analysis with a relational database storing data in the back end.

·Incremental data loading. As mentioned above, typically column-oriented databases are used for analysis and are quick to retrieve data, even when processing complex queries, as it is kept close together in columns. While incremental data loads are not impossible, columnar databases do not perform them in the most efficient way. The columns first need to be scanned to identify the right rows and scanned further to locate the modified data which requires overwriting.

·Row-specific queries. Like the potential downfalls mentioned above, it all boils down to the same issue, which is using the right type of database for the right purposes. With row-specific queries, you are introducing an extra step of scanning the columns to identify the rows and then locating the data to retrieve. It takes more time to get to individual records scattered in multiple columns, rather than accessing grouped records in a single column. Frequent row-specific queries might cause performance issues by slowing down a column-oriented database, which is particularly designed to

help you get to required pieces of information quickly, thus defeating its purpose.

NoSQL databases are mostly designed to fit specific purposes and are not expected to work as a general type of storage. Wide-column databases are column-oriented rather than row-oriented and are intended to store and query big data. There are many different databases available within the type and it is worth exploring their features while on a hunt for the most suitable data storage solution.

Text search engine — What if we are designing a system like Amazon and need to implement a search feature. The thing about search features is that we need to consider typos as well. Suppose a user wants to search for “shirt” but types “shrt” instead. Now if we don’t show any results, it would be very poor user experience. Our system needs to be smart enough to show results for “shirt” or “shorts”. This is known as fuzzy search and this is where we use text search engines like Elasticsearch.

Caching Solutions — If we are designing a read-heavy system like Twitter or Facebook, we might end up catching a lot of data, even complete timelines, to meet the low latency requirement. Some options here would be Redis or Memcached.

File system storage — If we are designing some sort of asset delivery service, where we might need to store images or audio/video files, we might need to use something called blob storage. An extremely popular example is Amazon S3.

Data Warehouses — I know! We have been discussing data and storage all this time, so how can we not consider Big Data! Sometimes we just need to dump all the data in a single store where we can later perform all sorts of analytics. These systems are used more for offline reporting that usually transactions. This is where we end up using data warehouse solutions like *Hadoop*.