UNIT- I

Introduction, Overview and History of NoSQL Databases, SQL vs NOSQL, Advantages over RDBMS, Limitations, Different Types of NoSQL Databases, Attack of the Clusters, The Emergence of NoSQL. Aggregate Data Models; Aggregates, Example of Relations and Aggregates, Consequences of Aggregate Orientation.

Introduction:

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.

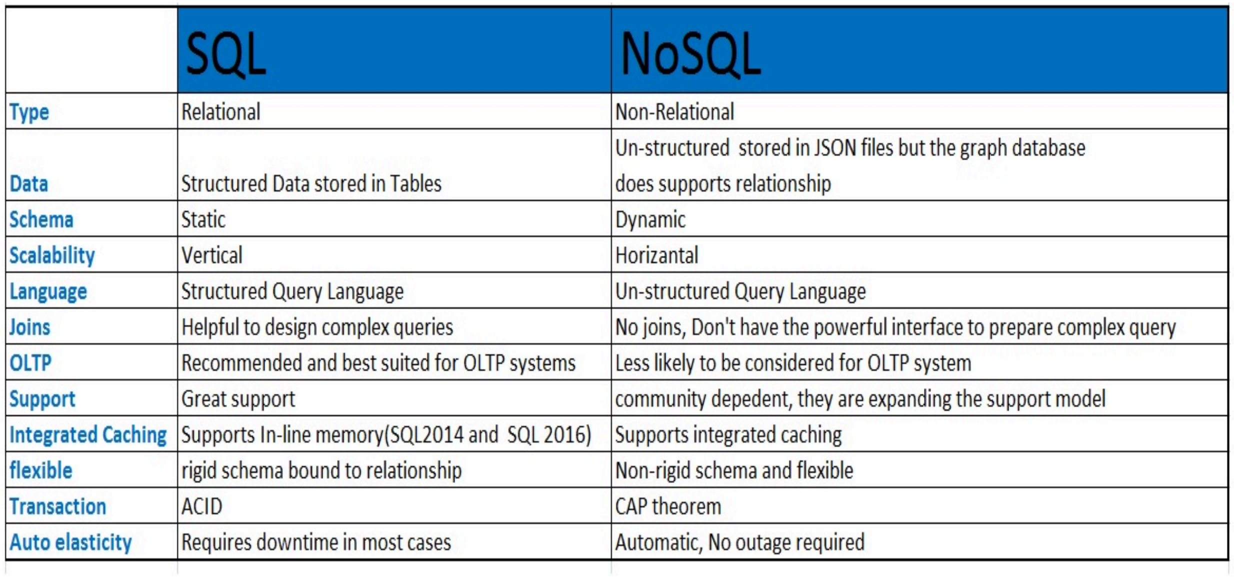
The term NoSQL originally referred to “non-SQL” or “non-relational” databases, but the term has since evolved to mean “not only SQL,” as NoSQL databases have expanded to include a wide range of different database architectures and data models.

NoSQL databases, also known as “not only SQL” databases, are a new type of database management system that have gained popularity in recent years. Unlike traditional relational databases, NoSQL databases are designed to handle large amounts of unstructured or semi-structured data, and they can accommodate dynamic changes to the data model. This makes NoSQL databases a good fit for modern web applications, real-time analytics, and big data processing.

Brief History of NoSQL Databases

- 1998- Carlo Strozzi use the term NoSQL for his lightweight, open-source relational database
- 2000- Graph database Neo4j is launched
- 2004- Google BigTable is launched
- 2005- CouchDB is launched
- 2007- The research paper on Amazon Dynamo is released
- 2008- Facebooks open sources the Cassandra project
- 2009- The term NoSQL was reintroduced

SQL Vs NOSQL



	SQL	NoSQL
Type	Relational	Non-Relational
Data	Structured Data stored in Tables	Un-structured stored in JSON files but the graph database does supports relationship
Schema	Static	Dynamic
Scalability	Vertical	Horizantal
Language	Structured Query Language	Un-structured Query Language
Joins	Helpful to design complex queries	No joins, Don't have the powerful interface to prepare complex query
OLTP	Recommended and best suited for OLTP systems	Less likely to be considered for OLTP system
Support	Great support	community depedent, they are expanding the support model
Integrated Caching	Supports In-line memory(SQL2014 and SQL 2016)	Supports integrated caching
flexible	rigid schema bound to relationship	Non-rigid schema and flexible
Transaction	ACID	CAP theorem
Auto elasticity	Requires downtime in most cases	Automatic, No outage required

Advantages over RDBMS

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

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.
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.

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.

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

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.

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.

Limitations

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.

Different Types of NoSQL Databases

- Document-based databases
- Key-value stores
- Column-oriented databases
- Graph-based databases

Document-Based Database:
The document-based database is a nonrelational database. Instead of storing the data in rows and columns (tables), it uses the documents to store the data in the database. A document database stores data in JSON, BSON, or XML documents.

Documents can be stored and retrieved in a form that is much closer to the data objects used in applications which means less translation is required to use these data in the applications. In the Document database, the particular elements can be accessed by using the index value that is assigned for faster querying.

Key features of documents database:

- Flexible schema: Documents in the database has a flexible schema. It means the documents in the database need not be the same schema.
- Faster creation and maintenance: the creation of documents is easy and minimal maintenance is required once we create the document.
- No foreign keys: There is no dynamic relationship between two documents so documents can be independent of one another. So, there is no requirement for a foreign key in a document database.
- Open formats: To build a document we use XML, JSON, and others.

Key-Value Stores:

A key-value store is a nonrelational database. The simplest form of a NoSQL database is a key-value store. Every data element in the database is stored in key-value pairs. The data can be retrieved by using a unique key allotted to each element in the database. The values can be simple data types like strings and numbers or complex objects.

A key-value store is like a relational database with only two columns which is the key and the value.

Key features of the key-value store:

- Simplicity.
- Scalability.
- Speed.

Column Oriented Databases:
A column-oriented database is a non-relational database that stores the data in columns instead of rows. That means when we want to run analytics on a small number of columns, you can read those columns directly without consuming memory with the unwanted data.

Columnar databases are designed to read data more efficiently and retrieve the data with greater speed. A columnar database is used to store a large amount of data. Key features of columnar oriented database:

- Scalability.
- Compression.
- Very responsive.

Graph-Based databases:
Graph-based databases focus on the relationship between the elements. It stores the data in the form of nodes in the database. The connections between the nodes are called links or relationships.

Key features of graph database:

- In a graph-based database, it is easy to identify the relationship between the data by using the links.
- The Query's output is real-time results.
- The speed depends upon the number of relationships among the database elements.

Attack of the Clusters

- Clustering is establishing connectivity among two or more servers in order to make it work like one. Clustering is a very popular technic among Sys-Engineers that they can cluster servers as a failover system, a load balance system or a parallel processing unit.

- In the systems design world, implementing such a design may be necessary especially in large systems (web or mobile applications), as a single database server would not be capable of handling all of the customers' requests

Why?

- Clustering servers is completely a scalable solution. You can add resources to the cluster afterwards.

- If a server in the cluster needs any maintenance, you can do it by stopping it while handing the load over to other servers

- Among high availability options, clustering takes a special place since it is reliable and easy to configure. In case of a server is having a problem providing the services furthermore, other servers in the cluster can take the load.

Aggregate

Aggregate means a collection of objects that are treated as a unit. In NoSQL Databases, an aggregate is a collection of data that interact as a unit.

Aggregate Data Models:

Aggregate Data Models in NoSQL make it easier for the Databases to manage data storage over the clusters as the aggregate data or unit can now reside on any of the machines. Whenever data is retrieved from the Database all the data comes along with the Aggregate Data Models in NoSQL.

Aggregate Data Models in NoSQL don't support ACID transactions and sacrifice one of the ACID properties.

Types of Aggregate Data Models in NoSQL Databases
.

- Document databases: These databases store data as semi-structured documents, such as JSON or XML, and can be queried using document-oriented query languages.
- Key-value stores: These databases store data as key-value pairs, and are optimized for simple and fast read/write operations.
- 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.
- Graph databases: These databases store data as nodes and edges, and are designed to handle complex relationships between data.

Example of Relations and Aggregates

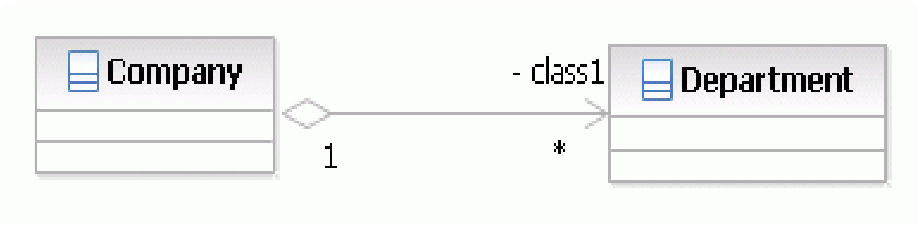
- In UML models, an aggregation relationship shows a classifier as a part of or subordinate to another classifier.

- An aggregation is a special type of association in which objects are assembled or configured together to create a more complex object.

- An aggregation describes a group of objects and how you interact with them. Aggregation protects the integrity of an assembly of objects by defining a single point of control, called the aggregate, in the object that represents the assembly.

- Aggregation also uses the control object to decide how the assembled objects respond to changes or instructions that might affect the collection.

- Data flows from the whole classifier, or aggregate, to the part. A part classifier can belong to more than one aggregate classifier and it can exist independently of the aggregate. For example, a Department class can have an aggregation relationship with a Company class, which indicates that the department is part of the company. Aggregations are closely related to compositions.



Consequences of Aggregate Orientation.

1. Aggregates have an important consequence for transactions.
2. Relational databases allow you to manipulate any combination of rows from any tables in a single (ACID) transaction (i.e., Atomic, Consistent, Isolated, and Durable)
3. It's often said that NoSQL databases don't support ACID transactions and thus sacrifice consistency. This is however not true for graph databases (which are, as relational database, aggregate-agnostic)
4. In general, its true that aggregate-oriented databases don't have ACID transactions that span multiple aggregates. Instead, they support atomic manipulation of a single aggregate at a time: This means that if we need to manipulate multiple aggregates in an atomic way, we have to manage that ourselves in the application code!
5. In practice, we find that most of the time we are able to keep our atomicity needs to within a single aggregate; indeed, that is part of the consideration for deciding how to divide up our data into aggregates