UNIT 8 NoSQL DATABASE

Structure **Page Nos.**

8.0	Introduction	88
8.1	Objectives	46
8.2	Introduction to NoSQL	46
	8.2.1 What is NoSQL	56
	8.2.2 Brief history of NoSQL Databases	89
	8.2.3 NoSQL database features	90
	8.2.4 Differentiate between RDBMS and NoSQL	98
8.3	Types of NoSQL Databases	66
	8.3.1 Column based	77
	8.3.2 Graph based	77
	8.3.3 Key-value pair based	55
	8.3.4 Document based	76
8.4	Summary	65
8.5	Solutions/Answers	65
8.6	Further Readings	66

8.0 INTORDUCTION

In the previous Units of this Block, you have gone through various large data architectural frameworks, such as Hadoop, SPARK and other similar technologies. However, these technologies are not a replacement for large-scale database systems. NoSQL databases arose because databases at the time were not able to support the rapid development of scalable web-based applications.

NoSQL databases have changed the manner in which data is stored and used, despite the fact that relational databases are still commonly employed. Most applications come with features like Google-style search, for instance. The growth of data, online surfing, mobile use, and analytics have drastically altered the requirements of contemporary databases. These additional requirements have spurred the expansion of NoSQL databases, which now include a range of types such as key-value, document, column, and graph.

In this Unit, we will discuss the many kinds of NoSQL databases, including those that are built on columns, graphs, key-value pairs, and documents respectively.

8.1 OBJECTIVES

After going through this unit, you should be able to:

- define what is NoSQL;
- differentiate between NoSQL and SQL;
- explain the basic features of Column based NoSQL Database;
- explain the Graph-based NoSQL Database;
- explain the Key-value pair based NoSQL Database and
- explain the Document based NoSQL Database.

8.2 INTRODUCTION TO NoSQL

Databases are a crucial part of many technological and practical systems. The phrase "NoSQL database" is frequently used to describe any non-relational database. NoSQL is sometimes referred to as "non SQL," but it is also referred to as "not only SQL." In either case, the majority of people agree that a NoSQL database is a type of database that stores data in a format that is different from relational tables.

Whenever you want to use the data, it must first be saved in a particular structure and then converted into a usable format. On the other hand, there are some circumstances in which the data are not always presented in a structured style, which means that their schemas are not always rigorous. This unit provides an in-depth look into NoSQL and the features that make it unique.

8.2.1 What is NoSQL?

NoSQL is a way to build databases that can accommodate many different kinds of information, such as key-value pairs, multimedia files, documents, columnar data, graphs, external files, and more. In order to facilitate the development of cutting-edge applications, NoSQL was designed to work with a variety of different data models and schemas.

The great functionality, ease of development, and performance at scale offered by NoSQL have helped make it a popular name. NoSQL is sometimes referred to as a non-relational database due to the numerous data handling features it offers. Due to the fact that it does not adhere to the guidelines established by Relational Database Management Systems (RDBMS), you cannot query your data using conventional SQL commands. We can think of such well-known examples as MongoDB, Neo4J, HyperGraphDB, etc.

8.2.2 Brief history of NoSQL Databases

In the late 2000s, as the price of storage began to plummet, No-SQL databases began to gain popularity. No longer is it necessary to develop a sophisticated, difficult-to-manage data model to prevent data duplication. Because developers' time was quickly surpassing the cost of data storage, NoSQL databases were designed with efficiency in mind.

Table 1: History of Databases

Year	Database Solutions	Company / Database Technology
1970-2000	Mainly RDBMS related	Oracle, IBM DB2, SQL Server, MySQL
2000-2005	DotCom boom – new scale solutions, start of NoSQL dev, whitepapers	Google, Facebook, IBM, amazon
2005-2010	New open source & mainstream databases	Cassandra, Riak, Apache Hbase, neo4j, MongoDB, CouchDB, Redis

2010 onwards	Adoption of Cloud	DBaaS (Database as a Service)

As storage costs reduced significantly, the quantity of data that applications were required to store and query grew. This data came in all forms— structured, semi-structured, and unstructured — and sizes making it practically difficult to define the schema in advance. NoSQL databases give programmers a great deal of freedom by enabling them to store enormous amounts of unstructured data.

In addition, the Agile Manifesto was gaining momentum, and software developers were reconsidering their approach to software development. They were beginning to understand the need of being able to quickly adjust to ever-evolving requirements. They required the flexibility to make rapid iterations and adjustments to all parts of their software stack, including the underlying database. They were able to achieve this flexibility because of NoSQL databases.

The use of the public cloud as a platform for storing and serving up data and applications was another trend that arose, as cloud computing gained popularity. To make their applications more robust, to expand out rather than up, and to strategically position their data across geographies, they needed the option to store data across various servers and locations. These features are offered by some NoSQL databases like MongoDB.

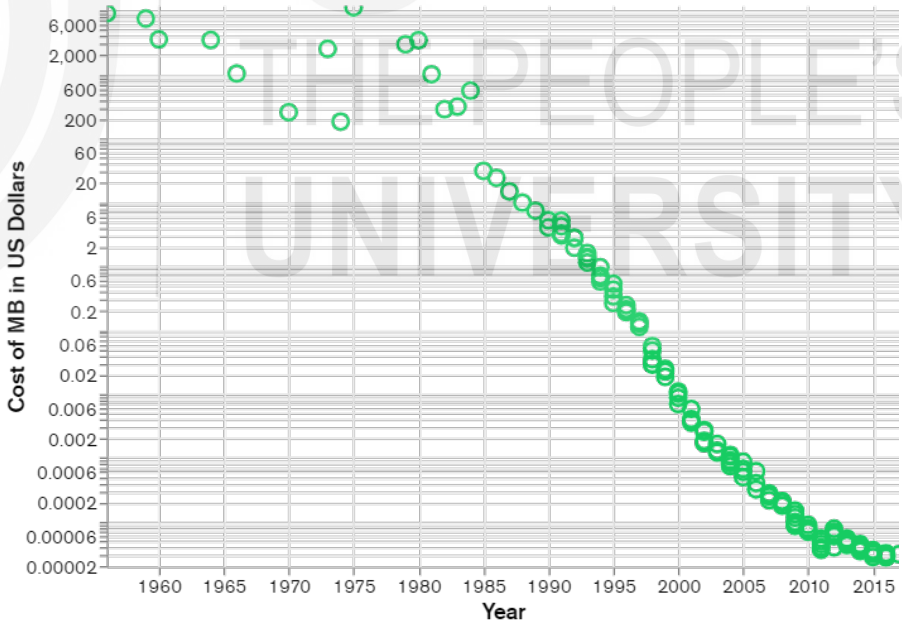

Figure 1: Cost per MB of Data over Time (Log Scale)
(Adapted from https://www.mongodb.com)

8.2.3 NoSQL database features

Every NoSQL database comes with its own set of one-of-a-kind capabilities. The following are general characteristics shared by several NoSQL databases:

- Schema flexibility

- Horizontal scaling
- Quick responses to queries as a result of the data model
- Ease of use for software developers

8.2.4 Difference between RDBMS and NoSQL

The differences and similarities between the two DBMSs are as follows:

- For the most part, NoSQL databases fall under the category of non-relational or distributed databases, while SQL databases are classified as Relational Database Management Systems (RDBMS).
- Databases that use the Structured Query Language (SQL) are table-oriented, while NoSQL databases use either document-oriented or key-value pairs or wide-column stores, or graph databases.
- Unlike NoSQL databases, which have dynamic or flexible schema to manage unstructured data, SQL databases have a strict or static schema.
- Structured data is stored using SQL, whereas both structured and unstructured data can be stored using NoSQL.
- SQL databases are thought to be scalable in a vertical direction, whereas NoSQL databases are thought to be scalable in a horizontal direction.
- Increasing the computing capability of your hardware is the first step in the scaling process for SQL databases. In contrast, NoSQL databases scale by distributing the load over multiple servers.
- MySQL, Oracle, PostgreSQL, and Microsoft SQL Server are all examples of SQL databases. BigTable, MongoDB, Redis, Cassandra, RavenDb, Hbase, CouchDB, and Neo4j are a few examples of NoSQL databases.

Vertical scalability is required for SQL databases. This means that an excessive amount of load must be able to be managed by increasing the amount of CPU, SSD, RAM, GPU, etc. on your server. When it comes to NoSQL databases, the ability to scale horizontally is one of their defining characteristics. This means that the addition of additional servers will make the task of managing demand more manageable.

☞ **Check Your Progress 1**

1) What is NoSQL?

……

……

……

2) What are the features of NoSQL databases?

……

……

……

3) Differentiate between the NoSQL and SQL.

……

……

……

8.3 TYPES OF NOSQL DATABASES

In this section, we will discuss the many classifications of NoSQL databases. There are typically four types of NoSQL databases:

1) Column-based: Instead of accumulating data in rows, this method organizes it all together into columns, which makes it easier to query large datasets.
2) Graph-based: These are systems that are utilized for the storage of information regarding networks, such as social relationships.
3) Key-value pair based: This is the simplest sort of database, in which each item of your database is saved in the form of an attribute name (also known as a "key") coupled with the value.
4) Document-based: Made up of sets of key-value pairs that are kept in documents.

8.3.1 Column based

A column store, in contrast to a relational database, is arranged as a set of columns, rather than rows. This allows you to read only the columns you need for analysis, saving memory space that would otherwise be taken up by irrelevant information. Because columns are frequently of the same kind, they are able to take advantage of more efficient compression, which makes data reading even quicker. The value of a specific column can be quickly aggregated using columnar databases.

Although columnar databases are excellent for analytics, because of the way they publish data, it is challenging for them to remain consistent because writes to all the columns need several write events on the disk. However, this problem never arises with relational databases because row data is continuously written to disk.

How Does a Column Database Work?

A columnar database is a type of database management system (DBMS) that allows data to be stored in columns rather than rows. It is accountable for reducing the amount of time needed to return a certain query. Additionally, it is accountable for the significant enhancement of the disk I/O performance. Both data analytics and data warehousing benefit from it. Additionally, the primary goal of a Columnar Database is to read and write data in an efficient manner. Column-store databases include Casandra, CosmoDB, Bigtable, and HBase, to name a few.

Columnar Database Vs Row Database:
When processing big data analytics and data warehousing, there are a number of different techniques that can be used, including columnar databases and row databases. But they each take a different method.

For instance:
• Row Database: “Customer 1: Name, Address, Location". (The fields for each new record are stored in a long row).
• Columnar Database: “Customer 1: Name, Address, Location”. (Each field has its own set of columns). Refer Table 2 for relational database example.

Table 2: Relational database: an example

ID Number	First Name	Last Name	Amount
A01234	Sima	Kaur	4000
B03249	Tapan	Rao	5000
C02345	Srikant	Peter	1000

In a Columnar DBMS, the data will be stored in the following format:
A01234, B03249, C02345; Sima, Tapan, Srikant; Kaur, Rao, Peter; 4000, 5000, 1000.

In a Row-oriented DBMS, the data will be stored in the following format:
A01234, Sima, Kaur, 4000; B03249, Tapan, Rao, 5000; C02345, Srikant, Peter, 1000.

Columnar databases: advantages

The use of columnar databases has various advantages:

- Column stores are highly effective in compression, making them storage efficient. This implies that you can conserve disk space while storing enormous amounts of data in a single column.
- Aggregation queries are fairly quick with column-store databases because the majority of the data is kept in a column, which is beneficial for projects that need to execute a lot of queries quickly.
- Load times are also quite good; a table with a billion rows can be loaded in a matter of seconds. This suggests that you can load and query practically instantly.
- A great deal of versatility because columns do not have to resemble one another. The database would not be affected if you add new or different columns, however, updating all tables is necessary to input whole new record queries.
- Overall, column-store databases are excellent for analytics and reporting due to their quick query response times and capacity to store massive volumes of data without incurring significant costs.

Column databases: Disadvantages

While there are many benefits to adopting column-oriented databases, there are also a few drawbacks to keep in mind.

- It takes a lot of time and effort to create an efficient indexing schema.
- Incremental data loading is undesirable and is to be avoided, if at all possible, even though this might not be a problem for some users.
- This applies to all forms of NoSQL databases, not just those with columns. Web applications frequently have security flaws, and the absence of security features in NoSQL databases does not help. If security is your top goal, you should either consider using relational databases or, if it's possible, use a clearly specified schema.
- Due to the way data is stored, Online Transaction Processing (OLTP) applications are incompatible with columnar databases.

Are columns databases always NoSQL?

Before we conclude, we should note that column-store databases are not always NoSQL-only. It is frequently argued that column-store belongs firmly in the NoSQL camp because it differs so much from relational database approaches. The debate between NoSQL and SQL is generally quite nuanced, therefore this is not usually the case. They are essentially the same as SQL techniques when it comes to column-store databases. For instance, keyspaces function as schema, so schema management is still necessary. A NoSQL data store's keyspace contains all column families. The concept is comparable to relational database management systems' schema. There is typically only one keyspace per program. Another illustration is the fact that the metadata occasionally resembles a conventional relational DBMS perfectly. Ironically, column-store databases frequently adhere to ACID and SQL standards. However, NoSQL databases are often either document-store or key-store, neither of which are column-store. Therefore, it is difficult to claim that column-store is a pure NoSQL system.

8.3.2 Graph based

The initial hardware hurdles that made it feasible for SQL to handle vast quantities of data are no longer there, despite the fact that SQL is an excellent superb RDBMS and has been used for many years to manage massive amounts of data. As a result, NoSQL has rapidly emerged as the dominant form of contemporary database management and many of the largest websites, we rely on today, are powered by NoSQL, like Twitter's use of FlockDB and Amazon's DynamoDB.

A database that stores data using graph structures is known as a graph database. It represents and stores data using nodes, edges, and attributes rather than tables or documents. Relationships between the nodes are represented by the edges. This makes data retrieval simpler and, in many circumstances, only requires one action. Additionally, it works fantastically as a database for fast, threaded data structures like those used on Twitter

How does a Graph Database Work?

Graphs, which are not relational databases, rely heavily on the idea of multi-

relational data "pathways" for their functionality. However, the structure of graph databases is typically simple. They are largely made up of two elements:

- The Node: This represents the actual data itself. It may be the number of people who watched a video on YouTube; it could be the number of people who read a tweet; or it could even be fundamental information like people's names, addresses, and other such details.
- The Edge: This clarifies the real connection between the two nodes. It is interesting to note that edges can also have their own data, such as the type of connection between two nodes. Similar to edges, mentioned directions may also describe the direction in which the data is flowing.

Graph databases are mostly utilized for studying relationships. For instance, businesses might extract client information from social media using a graph database. For example, some organization might use a graph database to extract data about relationships between Person, Restaurant, and City, as shown in Figure 2.

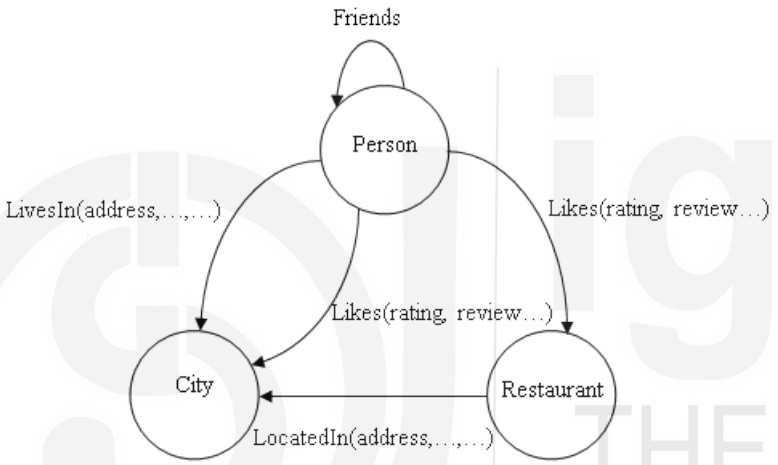

Figure 2. Different Nodes and Edges in Graph Database.
(Adapted from https://www.kdnuggets.com/)

When do we need Graph Database?

1) It resolves issues with many-to-many relationships. For example, many-to-many relationships include friends of friends.
2) When connections among data pieces are more significant. For example, there is a profile with some unique information, but the main selling point is the relationship between these different profiles, which is how you get connected inside a network.
3) Low latency with big amounts of data. The relational database's data sets will grow significantly as you add more relationships, and when you query it, its complexity will increase and it will take longer than usual. However, graph databases are specifically created for this purpose, and one can easily query relationships.

Now, let’s look at a more specific illustration to explain a group of people's complicated relationships. For example, five friends share a social network. These friends are Binny, Bhawna, Chaitaya, Manish, and Mohit. Their personal data may be kept in a graph database that resembles this, as shown in Figure 3 and Table 3:

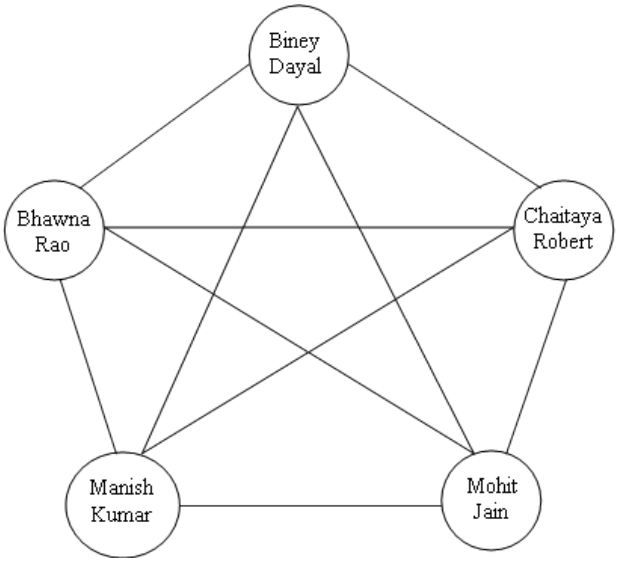

Figure 3. Example-Five friends sharing social network.

Table 3: Relational database: an example

Id	Firstname	Lastname	Email	Mobile
1001	Biney	Dayal	binnya@example.com	8645212321
1002	Bhawna	Rao	bhawanrao@example.com	9645212323
1003	Chaitaya	Robert	chaitayarob@example.com	7645212356
1004	Manish	Kumar	mkumar@example.com	9955212320
1005	Mohit	Jain	mjain@example.com	9945212329

This means we will need yet another table to keep track of user relationships. Our friendship table (refer Table 4) will resemble the following:

Table 4: Friendship Table

user_id	friend_id
1001	1002
1001	1003
1001	1004
1001	1005
1002	1001
1002	1003
1002	1004
1002	1005
1003	1001
1003	1002
1003	1004
1003	1005
1004	1001
1004	1002
1004	1003
1004	1005
1005	1001
1005	1002
1005	1003
1005	1004

We won't go too deeply into the theory of the database's main key and foreign key. Instead, presume that the friendship table uses both friends' ids. Let's say that every member on our social network gets access to a feature that lets them view the personal information of their other users who are friends with them. This means that if Chaitaya were to ask for information, it would be regarding Biney, Bhawna, Manish and Mohit. We shall address this issue in a conventional

(relational database) manner. First, we need to locate Chaitaya's user id in the database's Users table (refer Table 5).

Table 5: Chaitaya's Record

Id	Firstname	Lastname	Email	Mobile
1003	Chaitaya	Robert	chaitanyarob@example.net	7645212356

We would now search the friendship table (refer Table 6) for all tuples with the user id of 3. The resulting relationship would look like this:

Table 6: Friendship Table for user id 3

user_id	friend_id
1003	1001
1003	1002
1003	1004
1003	1005

Let us now examine the time required for this Relational database strategy. This will be close to log (N) times, where N is the number of tuples in the friendship table. In this case, the database continues to keep the entries in sequential order based on their ids. So, in general, the time complexity for 'M' number of queries is M*log (N). Only, if we had used a graph database strategy the overall time complexity have been O (N). For the simple reason that once Chaitaya has been located in the database, all the rest of her friends may be found with a single click, as shown in Figure 4.

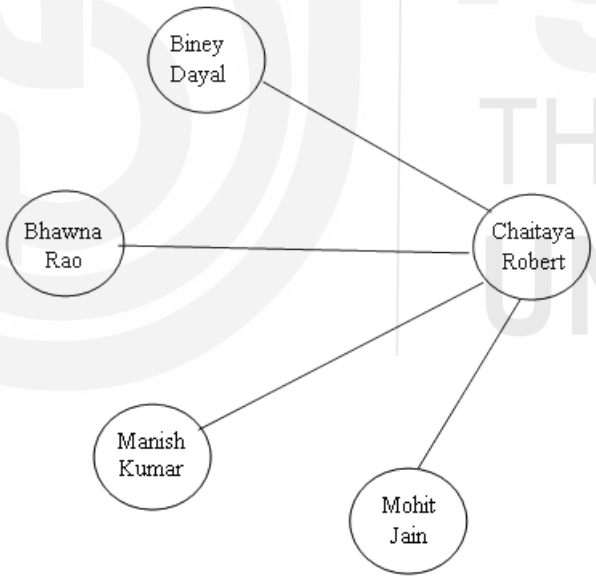

Figure 4. Accessing others data with a single click.

Graph Database Examples

Although graph databases are not as widely used as other NoSQL databases, there are a handful that have become de facto standards when discussing NoSQL:

Neo4j is both an open-source and an interestingly developed on Java graph database. It is considered to be one of the best graph databases. In addition to that, it comes with its own language known as Cypher, which is comparable to the declarative SQL language but is designed to work with graphs. In addition to Java, it supports a number of other popular programming languages, including Python, .NET, JavaScript, and a few others. Neo4j excels in applications such as

the administration of data centers and the identification of fraudulent activity.

RedisGraph is a graph module that is integrated into Redis, which is a key-value NoSQL database. RedisGraph was developed to have its data saved in RAM for the same reason that Redis itself is constructed on in-memory data structures. As a result, a graph database with excellent speed and quick searching and indexing is created. RedisGraph also makes use of Cypher, which is ideal if you're a programmer or data scientist looking for greater database flexibility. Applications that require blazing-fast performance are the main uses.

OrientDB It is interesting to note that OrientDB supports graph, document store, key-value store, and object-based data formats. Having stated that, the graph model, which uses direct links between databases, is used to hold all of the relationships. Although it does not use Cypher, OrientDB is open-source and developed in Java, just like Neo4j and the two prior graph databases. OrientDB is designed to be used in situations when many data models are necessary, and as a result, it is optimized for data consistency as well as minimizing data complexity.

8.3.3 Key-value pair based

Key-value stores are perhaps the most widely used of the four major NoSQL database formats because of their simplicity and quick performance. Let us examine key-value stores' operation and application in more detail. With some of the most well-known platforms and services depending on them to deliver material to users with lightning speed, NoSQL has grown in significance in our daily lives. Of course, NoSQL includes a range of database types, but key-value store is unquestionably the most used.

Because of its extreme simplicity, this kind of data model is built to execute incredibly quickly when compared to relational databases. Furthermore, because key-value stores adhere to the scalable NoSQL design philosophy, they are flexible and simple to set up.

How Does a Key-Value Work?

In reality, key-value storage is quite simple. A value is saved with a key that specifies its location, and a value can be pretty much any piece of data or information. In reality, this design idea may be found in almost every programming language as an array or map object, refer Figure 5. The fact that it is persistently kept in a database management system makes a difference in this case.

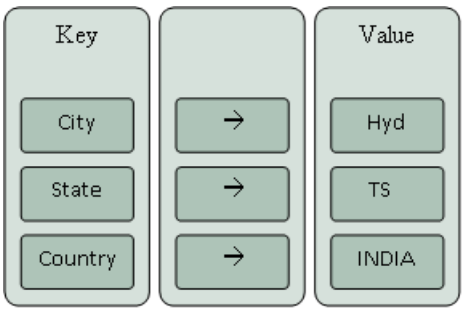

Figure 5. Example Key-Value database.

Popularity of key-value stores is due to the fact that information is stored as a single large piece of data instead of as discrete data. As a result, indexing the database is not really necessary to improve its performance. Instead, because of

the way it is set up, it operates more quickly on its own. Similar to that, it mostly uses the get, put, and delete commands rather than having a language of its own.

Of course, this has the drawback that the data you receive in response to a request is not screened. Under certain conditions, this lack of data management may be problematic, but generally speaking, the trade-off is worthwhile. Because key-value stores are both quick and reliable, the vast majority of programmers find ways to get around any filtering or control problems that may arise.

Benefits of Key-Value

Key-value data models, one of the more well-liked types of NoSQL data models, provide many advantages when it comes to creating a database:

Scalability: Key-value stores, like NoSQL in general, are infinitely scalable in a horizontal fashion, which is one of its main advantages over relational databases. This can be a huge advantage for sophisticated and larger databases compared to relational databases, where expansion is vertical and finite, as shown in Figure 6.

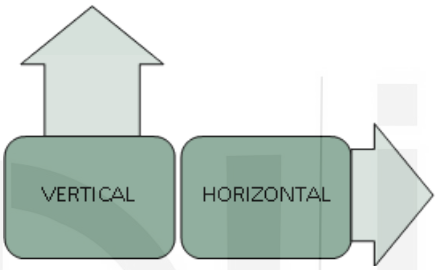

Figure 6. Horizontal and Vertical Scalability.

More specifically, partitioning and replication are used to manage this. Additionally, by avoiding things like low-overhead server calls, it decreases the ACID guarantees.

No/Simpler Querying: With key-value stores, querying is really not possible except in very particular circumstances when it comes to querying keys, and even then, it is not always practicable. Because there is just one request to read and one request to write, key-value makes it easier to manage situations like sessions, user profiles, shopping carts, and so on (due to the blob-like nature of how the data is stored). Similar to this, concurrency problems are simpler to manage because only one key needs to be resolved.

Mobility: Because key-value stores lack a query language, it is simple to move them from one system to another without modifying the architecture or the code. Thus, switching operating systems is less disruptive than switching relational databases.

When to Use Key-Value

Key-value stores excel in this area because traditional relational databases are not actually designed to manage a large number of read/write operations. Key-value can readily scale to thousands of users per second due to its scalability. Additionally, it can easily withstand lost storage or data because of the built-in redundancy.

As a result, key-value excels in the following instances:

- Profiles and user preferences
- Large-scale user session management
- Product suggestions (such as in eCommerce platforms)

- Delivery of personalized ads to users based on their data profiles
- Cache data for infrequently updated data

There are numerous other circumstances where key-value works nicely. For instance, because of its scalability, it frequently finds usage in big data research. Similar to how it works for web applications, key-value is effective for organizing player sessions in MMOG (massively multiplayer online game) and other online games.

Key-Value Database Examples

Some key-value database models, for instance, save information to a solid-state drive (SSD), while others use random-access memory (RAM). We depend on key-value stores on a daily basis in our lives since they are some of the most popular and frequently used databases. The fact is that some of the most popular and commonly used databases are key-value stores.

Amazon DynamoDB is most likely the database that is used the most often for key-value storage. In point of fact, study into Amazon DynamoDB was the impetus for the rise in popularity of NoSQL.

Aerospike is a free and open-source database that was designed specifically for use with in-memory data storage.

Berkeley DB: Another free and open-source database, Berkeley DB is a high-performance framework for storing databases, despite the fact that it has a very simple interface.

Couchbase: Text searches and querying in a SQL-like format are both possible with Couchbase, which is an interesting feature.

Memcached not only saves cached data in RAM, which helps websites load more quickly, but it is also free and open source.

Riak was designed specifically for use in the app development process, and it plays well with other databases and app platforms.

Redis: A database that serves as both a memory cache and a message broker.

8.3.4 Document based

A non-relational database that stores data as structured documents is known as a document database (also known as a NoSQL document store). Instead of using standard rows and columns, JSON format is a more recent technique to store data. An XML or JSON file, or a PDF, are all examples of documents. NoSQL is everywhere nowadays; just look at Twitter and its use of FlockDB or Amazon and their use of DynamoDB. Figure 7 shows the difference between the Relational and Document Store model.

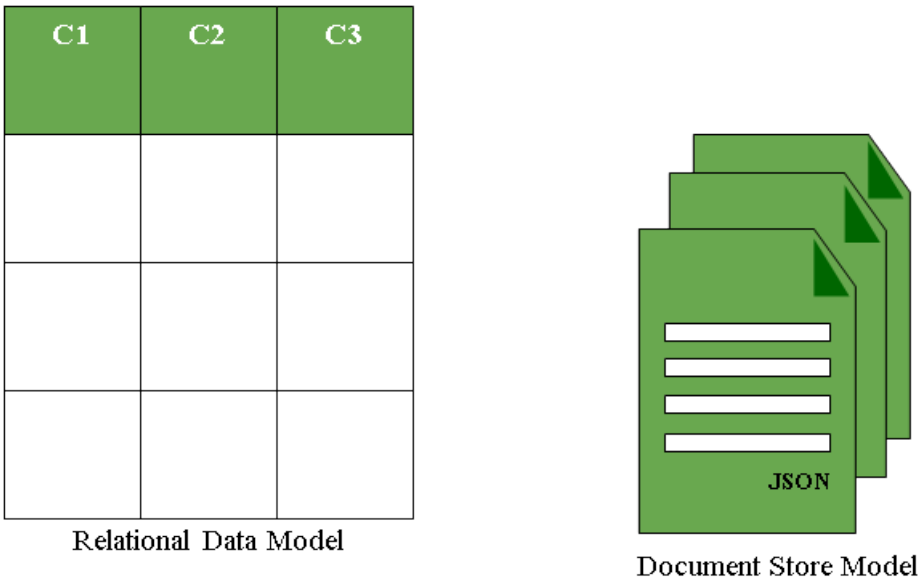

Figure 7: Relational Vs Document Store Model.

In spite of the fact that there are a great deal of data models, each of which contains hundreds of databases, the one we are going to investigate today is called Document-store. One of the most common database models now in use, document-store functions in a manner that is somewhat similar to that of the key-value model in the sense that documents are saved together with particular keys that access the information. Figure 8 (a) shows the document that holds information about a book. This file is a JSON representation of a book's metadata, which includes the book's BookID, Title, Author, and Year and Figure 8 (b) shows the same metadata for Key value database.

A Document

{
"BookID": "978-1449396091",
"Title": "DBMS",
"Author": "Raghu Ramakrishnan",
"Year": "2022",
}

(a)

Key	**Value**
BookID	**978-1449396091**
Title	**DBMS**
Author	**Raghu Ramakrishnan**
Year	**2022**

(b)

Figure 8: Example of Document and Key value database

When to use a document database?

- When your application requires data that is not structured in a table format.
- When your application requires a large number of modest continuous reads and writes and all you require is quick in-memory access.
- When your application requires CRUD (Create, Read, Update, Delete) functionality.
- These are often adaptable and perform well when your application has to run across a broad range of access patterns and data kinds.

How does a Document Database Work?

It appears that document databases work under the assumption that any kind of information can be stored in a document. This suggests that you shouldn't have to worry about the database being unable to interpret any combination of data types. Naturally, in practice, most document databases continue to use some sort of schema with a predetermined structure and file format.

Document stores do not have the same foibles and limitations as SQL databases, which are both tubular and relational. This implies that using the information at hand is significantly simpler and running queries may also be much simpler. Ironically, you can execute the same types of operations in a document storage that you can in a SQL database, including removing, adding, and querying.

Each document requires a key of some kind, as was previously mentioned, and this key is given to it through a unique ID. This unique ID processed the document directly instead of being obtained column by column.

Document databases often have a lower level of security than SQL databases. As a result, you really need to think about database security, and utilizing Static Application Security Testing (SAST) is one approach to do so. SAST, examines the source code directly to hunt for flaws. Another option is to use DAST, a dynamic version that can aid in preventing NoSQL injections.

Document database advantages

One major benefit of document-store is that all of the data is stored in a single location, rather than being spread out over many interconnected databases. As a result, if you do not employ relational processes, you perform better than a SQL database.

- **Schema-less:** Because there are no constraints on the format and structure of data storage, they are particularly effective at keeping huge quantities of existing data.
- **Faster creation of document and maintenance:** The creation of a document is a fairly straightforward process, and apart from that, the upkeep requirements are virtually nonexistent.
- **Open formats**: It offers a relatively easy construction process that makes use of XML, JSON, and other formats.
- **Built-in versioning:** Because it contains built-in versioning, it means that when the documents expand in size, there is a possibility that they will also expand in complexity. Versioning makes conflicts less likely.

More precisely, document stores are excellent for the following applications because schema can be changed without any downtime or because you could not know future user needs:

- eCommerce giants (Like Amazon)
- Blogging platforms (such as Blogger, Tumblr)
- CMS (Content management systems) (Like WordPress, windows registry)
- Analytical platforms (such as Tableau, Oracle server)

Document databases' drawbacks

- **Weak Atomicity:** Multi-document ACID transactions are not supported. We will need to perform two different queries, one for each collection, in order to handle a change in the document data model involving two collections. This is where the atomicity criteria are violated.
- **Consistency Check Limitations:** A database performance issue may arise from searching for documents and collections that aren't linked to an author collection.
- **Security:** In today's world, many online apps do not have enough security, which in turn leads to the disclosure of critical data. Thus, web app vulnerabilities become a cause for concern.

Document databases examples

- One of the best NoSQL database engines is **MongoDB**, which is not only well-known but also uses JSON like format. It has its own query language.
- A search engine built on the document-store data architecture is **Elasticsearch**. Database searching and indexing may be accomplished using this straightforward and easy-to-learn tool.
- **CouchDB:** In addition to Ubuntu, it also works with the social networking site Facebook. It utilizes Javascript and is developed in the Erlang programming language.
- **BaseX** is a simple, open-source, XML-based DBM that makes use of Java.

☞ **Check Your Progress 2**

1) How Does a Column Database Work? Discuss.

……

……

……

2) What are the different Graph Database Examples?

……

……

……

3) Explain document based NoSQL database.

……

……

……

8.4 SUMMARY

This unit covered the fundamentals of NoSQL as well as the many kinds of NoSQL databases, such as those based on columns, graphs, key-value pairs, and

documents. Numerous businesses now use NoSQL. It is difficult to pick the best database platform. NoSQL databases are used by many businesses because of their ability to handle mission-critical applications while decreasing risk, data spread, and total cost of ownership.

Despite their incredible capability, column-store databases do have their own set of problems. Due to the fact that columns require numerous writes to the disk, for instance, the way the data is written results in a certain lack of consistency. Graph databases can be used to offer content in high-performance scenarios while producing threads that are simple to comprehend for the typical user, beyond merely expressive information in a graphical and effective way (such as in the case of Twitter). The simplicity of a key-value store is what makes it so brilliant. Although this has potential drawbacks, particularly when dealing with more complicated issues like financial transactions, it was designed specifically to fill in relational databases' inadequacies. We may create a pipeline that is even more effective by combining relational and non-relational technologies, whether we are working with users or data analysis. Document-store data models are quite popular and regularly used due to their versatility. It helps analytics by making it easy for firms to store multiple sorts of data for later use.

8.5 SOLUTIONS/ANSWERS

Check Your Progress 1

1) NoSQL is a way to build databases that can accommodate many different kinds of information, such as key-value pairs, multimedia files, documents, columnar data, graphs, external files, and more. In order to facilitate the development of cutting-edge applications, NoSQL was designed to work with a variety of different data models and schemas.

2)
   - Schema flexibility
   - Horizontal scaling
   - Quick responses to queries as a result of the data model
   - Ease of use for software developers

3) It is different in the following ways:
   - For the most part, NoSQL databases fall under the category of non-relational or distributed databases, while SQL databases are classified as Relational Database Management Systems (RDBMS).
   - Databases that use the Structured Query Language (SQL) are table-oriented, while NoSQL databases use either document-oriented or key-value pairs or wide-column stores, or graph databases.
   - Unlike NoSQL databases, which have dynamic or flexible schema to manage unstructured data, SQL databases have a strict, preset or static schema.
   - Structured data is stored using SQL, whereas both structured and unstructured data can be stored using NoSQL.
   - SQL databases are thought to be scalable in a vertical direction, whereas NoSQL databases are thought to be scalable in a horizontal direction.

- Increasing the computing capability of your hardware is the first step in the scaling process for SQL databases. In contrast, NoSQL databases scale by distributing the load over multiple servers.
- MySQL, SQLite, Oracle SQL, PostgreSQL, and Microsoft SQL Server are all examples of SQL databases. BigTable, MongoDB, Redis, Cassandra, RavenDb, Hbase, CouchDB, and Neo4j are a few examples of NoSQL databases.

Check Your Progress 2

1) A columnar database is a type of database management system (DBMS) that allows data to be stored in columns rather than rows. It is accountable for reducing the amount of time needed to return a certain query. Additionally, it is accountable for the significant enhancement of the disk I/O performance. Both data analytics and data warehousing benefit from it. Additionally, the primary goal of a Columnar Database is to read and write data in an efficient manner. Column-store databases include Casandra, CosmoDB, Bigtable, and HBase, to name a few. Also, refer 8.3.1.

2) Graph Database Examples:

- **Neo4j** is both an open-source and an interestingly developed on Java graph database. It is considered to be one of the best graph databases in the world. In addition to that, it comes with its own language known as Cypher, which is comparable to the declarative SQL language but is designed to work with graphs. In addition to Java, it supports a number of other popular programming languages, including Python, .NET, JavaScript, and a few others. Neo4j excels in applications such as the administration of data centres and the identification of fraudulent activity.
- **RedisGraph** is a graph module that is integrated into Redis, which is a key-value NoSQL database. RedisGraph was developed to have its data saved in RAM for the same reason that Redis itself is constructed on in-memory data structures. As a result, a graph database with excellent speed and quick searching and indexing is created. RedisGraph also makes use of Cypher, which is ideal if you're a programmer or data scientist looking for greater database flexibility. Applications that require blazing-fast performance are the main uses.
- **OrientDB:** It's interesting to note that OrientDB supports graph, document store, key-value store, and object-based data formats. Having stated that, the graph model, which uses direct links between databases, is used to hold all of the relationships.

3) It is generally agreed that document stores, which are a sort of NoSQL database, are the most advanced of the available options. They use JSON as their data storage format, which is different from the more traditional rows and columns layout. Most of the day-to-day activities that we carry out on the internet are supported by NoSQL databases. NoSQL is everywhere nowadays; just look at Twitter and its use of FlockDB or Amazon and their use of DynamoDB. Also, refer 8.3.4.

8.6 FURTHER READINGS

1) Next Generation Databases: NoSQL and Big Data 1st ed. Edition, G. Harrison, Apress, December 26, 2015.
2) Shashank Tiwari, Professional NoSQL, 1st Edition, Wrox, September 2011.
3) https://www.kdnuggets.com/