**UNIT-V**

**Distributed Databases**

* **A distributed database** is a database that is not limited to one computer system. It is like a database that consists of two or more files located in different computers or sites either on the same network or on an entirely different network.
* These sites do not share any physical component. Distributed databases are needed when a particular data in the database needs to be accessed by various users globally.
* A **distributed database** is a collection of multiple interconnected databases, which are spread physically across various locations that communicate via a computer network.
* Though there are many distributed databases to choose from, some examples of distributed databases include *Apache Ignite*, *Apache Cassandra*, *Apache HBas*e, *Amazon SimpleDB*, *Clusterpoint*, and *FoundationDB*.

**Transparency**

Distribution transparency is the property of distributed databases by the virtue of which the internal details of the distribution are hidden from the users. The DDBMS designer may choose to fragment tables, replicate the fragments and store them at different sites. However, since users are oblivious of these details, they find the distributed database easy to use like any centralized database.

**The three dimensions of distribution transparency are −**

* Location transparency
* Fragmentation transparency
* Replication transparency

Location Transparency

Location transparency ensures that the user can query on any table(s) or fragment(s) of a table as if they were stored locally in the user’s site. The fact that the table or its fragments are stored at remote site in the distributed database system, should be completely oblivious to the end user. The address of the remote site(s) and the access mechanisms are completely hidden.

In order to incorporate location transparency, DDBMS should have access to updated and accurate data dictionary and DDBMS directory which contains the details of locations of data.

Fragmentation Transparency

Fragmentation transparency enables users to query upon any table as if it were unfragmented. Thus, it hides the fact that the table the user is querying on is actually a fragment or union of some fragments. It also conceals the fact that the fragments are located at diverse sites.

This is somewhat similar to users of SQL views, where the user may not know that they are using a view of a table instead of the table itself.

Replication Transparency

Replication transparency ensures that replication of databases are hidden from the users. It enables users to query upon a table as if only a single copy of the table exists.

Replication transparency is associated with concurrency transparency and failure transparency. Whenever a user updates a data item, the update is reflected in all the copies of the table. However, this operation should not be known to the user. This is concurrency transparency. Also, in case of failure of a site, the user can still proceed with his queries using replicated copies without any knowledge of failure. This is failure transparency.

Combination of Transparencies

In any distributed database system, the designer should ensure that all the stated transparencies are maintained to a considerable extent. The designer may choose to fragment tables, replicate them and store them at different sites; all oblivious to the end user. However, complete distribution transparency is a tough task and requires considerable design efforts.

**Features of Distributed Databases**

In general, distributed databases include the following features:

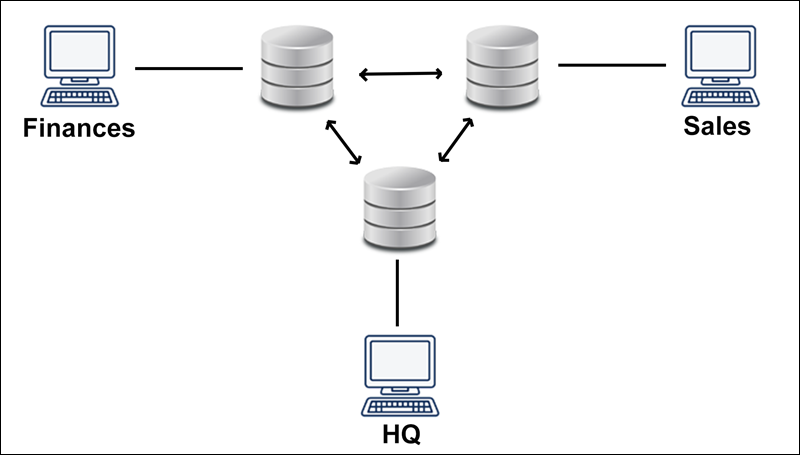
1. **Location independency**: Data is independently stored at multiple sites and managed by independent *Distributed database management systems* **(DDBMS)**.
2. **Network linking**: All distributed databases in a collection are *linked by a network* and communicate with each other.
3. **Distributed query processing**: Distributed query processing is the procedure of answering queries (which means mainly read operations on large data sets) in a distributed environment.
   * Query processing involves the transformation of a **high-level query** (e.g., *formulated in SQL*) into a **query execution plan** (*consisting of lower-level query operators in some variation of relational algebra*) as well as the execution of this plan.
4. **Hardware independent**: The different sites where data is stored are *hardware-independent*. There is no physical contact between these distributed databases which is accomplished often through virtualization.
5. **Distributed transaction management**: Distributed database provides a consistent distribution through *commit protocols*, *distributed recovery methods*, and *distributed concurrency control techniques* in case of many transaction failures.

**Types of Distributed Database**

There are two types of distributed databases:

* **Homogenous** distributed database.
* **Heterogeneous** distributed database.

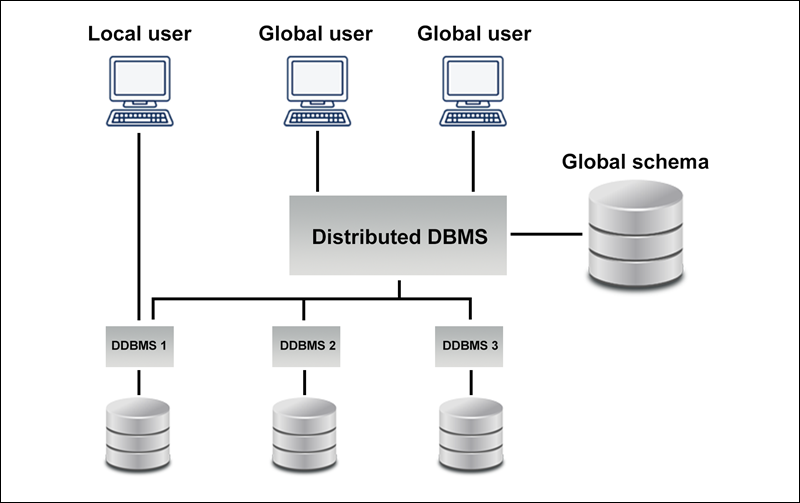
**Homogenous Distributed Database**



* A **Homogenous** distributed database is a network of **identical databases** stored on multiple sites. All databases stores data identically, the *operating system*, *DDBMS* and the *data structures used* – all are same at all sites, making them **easy to manage**.
* Below is a diagram for the same,

**Heterogeneous Distributed Database**

* It is the opposite of a Homogenous distributed database. It uses *different schemas*, *operating systems*, *DDBMS*, and *different data models* causing it **difficult to manage**.
* In the case of a Heterogeneous distributed database, a particular site can be **completely unaware** of other sites. This causes limited cooperation in processing *user requests*, this is why translations are required to establish communication between sites.
* Below is a diagram for the same,



**Fragmentation**

Fragmentation is the task of dividing a table into a set of smaller tables. The subsets of the table are called **fragments**. Fragmentation can be of three types: horizontal, vertical, and hybrid (combination of horizontal and vertical). Horizontal fragmentation can further be classified into two techniques: primary horizontal fragmentation and derived horizontal fragmentation.

Fragmentation should be done in a way so that the original table can be reconstructed from the fragments. This is needed so that the original table can be reconstructed from the fragments whenever required. This requirement is called “reconstructiveness.”

Advantages of Fragmentation

* Since data is stored close to the site of usage, efficiency of the database system is increased.
* Local query optimization techniques are sufficient for most queries since data is locally available.
* Since irrelevant data is not available at the sites, security and privacy of the database system can be maintained.

Disadvantages of Fragmentation

* When data from different fragments are required, the access speeds may be very low.
* In case of recursive fragmentations, the job of reconstruction will need expensive techniques.
* Lack of back-up copies of data in different sites may render the database ineffective in case of failure of a site.

Vertical Fragmentation

In vertical fragmentation, the fields or columns of a table are grouped into fragments. In order to maintain reconstructiveness, each fragment should contain the primary key field(s) of the table. Vertical fragmentation can be used to enforce privacy of data.

For example, let us consider that a University database keeps records of all registered students in a Student table having the following schema.

STUDENT

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Regd\_No | Name | Course | Address | Semester | Fees | Marks |

Now, the fees details are maintained in the accounts section. In this case, the designer will fragment the database as follows −

CREATE TABLE STD\_FEES AS

SELECT Regd\_No, Fees

FROM STUDENT;

Horizontal Fragmentation

Horizontal fragmentation groups the tuples of a table in accordance to values of one or more fields. Horizontal fragmentation should also confirm to the rule of reconstructiveness. Each horizontal fragment must have all columns of the original base table.

For example, in the student schema, if the details of all students of Computer Science Course needs to be maintained at the School of Computer Science, then the designer will horizontally fragment the database as follows −

CREATE COMP\_STD AS

SELECT \* FROM STUDENT

WHERE COURSE = "Computer Science";

Hybrid Fragmentation

In hybrid fragmentation, a combination of horizontal and vertical fragmentation techniques are used. This is the most flexible fragmentation technique since it generates fragments with minimal extraneous information. However, reconstruction of the original table is often an expensive task.

Hybrid fragmentation can be done in two alternative ways −

* At first, generate a set of horizontal fragments; then generate vertical fragments from one or more of the horizontal fragments.
* At first, generate a set of vertical fragments; then generate horizontal fragments from one or more of the vertical fragments.

# DATA REPLICATION

**Data replication is the process of making multiple copies of data and storing them at different locations for backup purposes, fault tolerance and to improve their overall accessibility across a network**. Similar to [data mirroring](https://www.manageengine.com/device-control/file-shadowing.html), data replication can be applied to both individual computers and servers. The data replicates can be stored within the same system, on-site and off-site hosts, and cloud-based hosts.

Common database technologies today either have built-in capabilities, or use third-party tools to accomplish data replication. While Oracle Database and Microsoft SQL actively support data replication, some traditional technologies may not include this feature out of the box.

Data replication can either be synchronous, meaning that any changes made to the original data will be replicated, or asynchronous, meaning replication is initiated only when the Commit statement is passed to the database.

## *Data replication in DBMS*

Data replication in DBMS (distribution servers) can be carried out using a suitable replication scheme. The widely-adopted replication schemes are as follows:

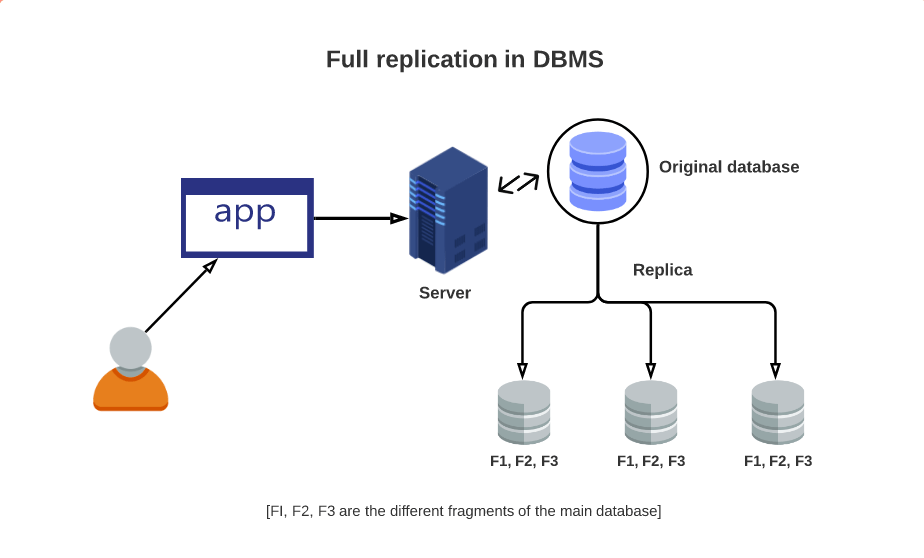
1. [Full data replication](https://www.manageengine.com/device-control/data-replication.html#link9)
2. [Partial data replication](https://www.manageengine.com/device-control/data-replication.html#link10)
3. [No replication](https://www.manageengine.com/device-control/data-replication.html#link11)

### Full data replication

Full replication means that the complete database is replicated at every site of the distributed system. This scheme maximizes data availability and redundancy across a wide area network.

For example, users in a cross-country network have access to the complete database from an Asia based server if the European or North American server experiences a technical difficulty.

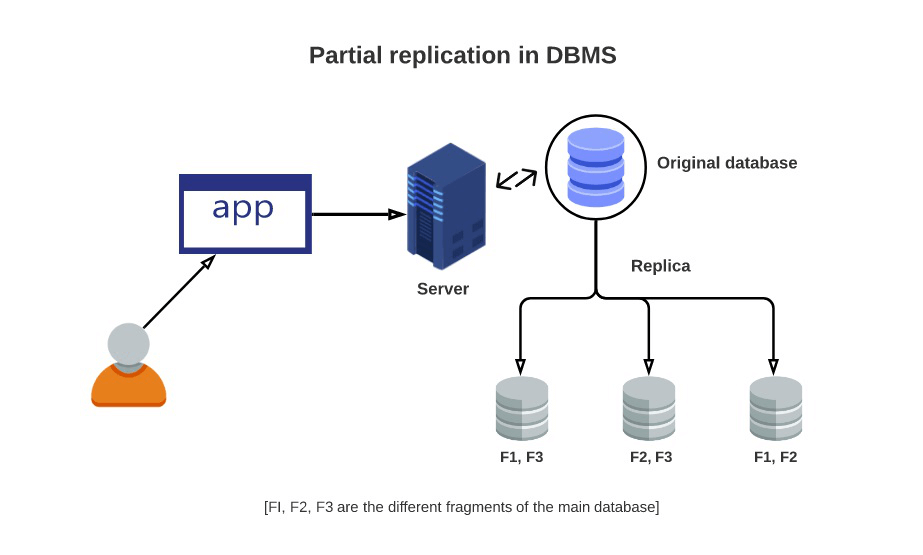
Full replication also contributes to faster execution of global queries as the results can be obtained from any local server.The disadvantage of full replication is that the update process tends to be on the slower side. This makes keeping up-to-date copies of data at every location quite challenging.



### Partial data replication

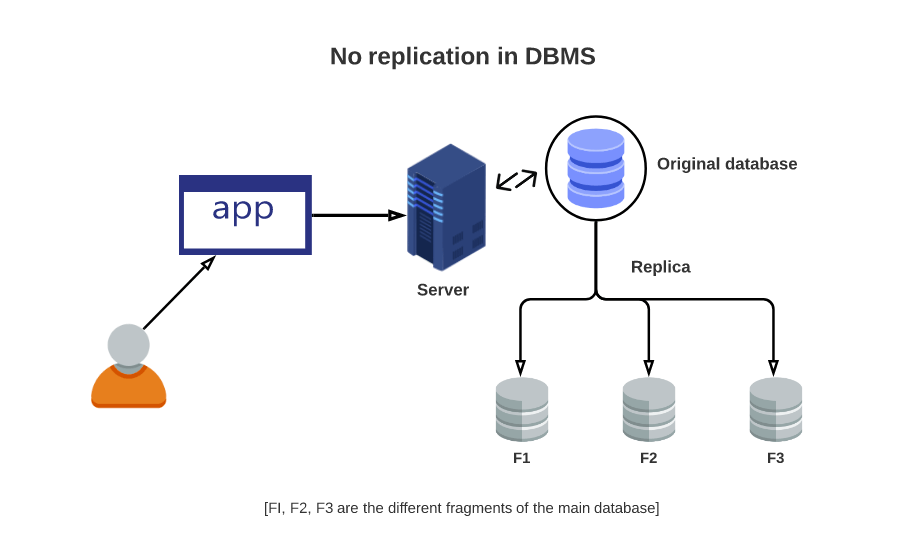
Partial replication occurs when only certain fragments of the database are replicated based on the importance of data at each location. Here, the number of copies can range from one to the total number of nodes in the distributed system.

In an enterprise environment, this mode of replication can be useful for members of sales and marketing teams where a partial database is stored on personal computers and regularly synced with the main server.



### No replication

In this mode of replication, only one fragment exists on each site of the distributed system. While no replication can be attributed to the ease of data recovery, it can have an adverse effect on the speed of execution of queries since multiple users access the same server. Compared to other replication schemes, no data replication in DBMS provides poor availability of data.



**Data Replication and Allocation**

Replication is useful in improving the availability of data. The most extreme case is replication of the *whole database* at every site in the distributed system, thus creating a **fully replicated distributed database**. This can improve availability remarkablybecause the system can continue to operate as long as at least one site is up. It also improves performance of retrieval for global queries because the results of such queries can be obtained locally from any one site; hence, a retrieval query can be processed at the local site where it is submitted, if that site includes a server module. The disadvantage of full replication is that it can slow down update operations drastically, since a single logical update must be performed on every copy of the database to keep the copies consistent. This is especially true if many copies of the database exist. Full replication makes the concurrency control and recovery techniques more expensive than they would be if there was no replication, as we will see in Section 25.7.

The other extreme from full replication involves having **no replication**—that is, each fragment is stored at exactly one site. In this case, all fragments *must be* dis-joint, except for the repetition of primary keys among vertical (or mixed) fragments. This is also called **nonredundant allocation**.

Between these two extremes, we have a wide spectrum of **partial replication** of the data—that is, some fragments of the database may be replicated whereas others may not. The number of copies of each fragment can range from one up to the total number of sites in the distributed system. A special case of partial replication is occurring heavily in applications where mobile workers—such as sales forces, financial plan-ners, and claims adjustors—carry partially replicated databases with them on laptops and PDAs and synchronize them periodically with the server database.7 A descrip-tion of the replication of fragments is sometimes called a **replication schema**.

Each fragment—or each copy of a fragment—must be assigned to a particular site in the distributed system. This process is called **data distribution** (or **data allocation**). The choice of sites and the degree of replication depend on the performanceand availability goals of the system and on the types and frequencies of transactions submitted at each site. For example, if high availability is required, transactions can be submitted at any site, and most transactions are retrieval only, a fully replicated database is a good choice. However, if certain transactions that access particular parts of the database are mostly submitted at a particular site, the corresponding set of fragments can be allocated at that site only. Data that is accessed at multiple sites can be replicated at those sites. If many updates are performed, it may be useful to limit replication. Finding an optimal or even a good solution to distributed data allocation is a complex optimization problem.

# XML Database

XML database is a data persistence software system used for storing the huge amount of information in XML format. It provides a secure place to store XML documents.

You can query your stored data by using XQuery, export and serialize into desired format. XML databases are usually associated with document-oriented databases.

## Types of XML databases

There are two types of XML databases.

1. XML-enabled database
2. Native XML database (NXD)

## XML-enable Database

XML-enable database works just like a relational database. It is like an extension provided for the conversion of XML documents. In this database, data is stored in table, in the form of rows and columns.

## Native XML Database

Native XML database is used to store large amount of data. Instead of table format, Native XML database is based on container format. You can query data by XPath expressions.

Native XML database is preferred over XML-enable database because it is highly capable to store, maintain and query XML documents.

Let's take an example of XML database:

1. **<?xml** version="1.0"**?>**
2. **<contact-info>**
3. **<contact1>**
4. **<name>**Vimal Jaiswal**</name>**
5. **<company>**SSSIT.org**</company>**
6. **<phone>**(0120) 4256464**</phone>**
7. **</contact1>**
8. **<contact2>**
9. **<name>**Mahesh Sharma **</name>**
10. **<company>**SSSIT.org**</company>**
11. **<phone>**09990449935**</phone>**
12. **</contact2>**
13. **</contact-info>**

In the above example, a table named contacts is created and holds the contacts (contact1 and contact2). Each one contains 3 entities name, company and phone.

**XML Schema:**

XML Schema is commonly known as **XML Schema Definition (XSD)**. It is used to describe and validate the structure and the content of XML data. XML schema defines the elements, attributes and data types. Schema element supports Namespaces. It is similar to a database schema that describes the data in a database.

## Syntax

You need to declare a schema in your XML document as follows −

### Example

The following example shows how to use schema −

<?xml version = "1.0" encoding = "UTF-8"?>

<xs:schema xmlns:xs = "http://www.w3.org/2001/XMLSchema">

<xs:element name = "contact">

<xs:complexType>

<xs:sequence>

<xs:element name = "name" type = "xs:string" />

<xs:element name = "company" type = "xs:string" />

<xs:element name = "phone" type = "xs:int" />

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:schema>

The basic idea behind XML Schemas is that they describe the legitimate format that an XML document can take.

## Elements

As we saw in the [XML - Elements](https://www.tutorialspoint.com/xml/xml_elements.htm) chapter, elements are the building blocks of XML document. An element can be defined within an XSD as follows −

<xs:element name = "x" type = "y"/>

## Definition Types

You can define XML schema elements in the following ways −

### Simple Type

Simple type element is used only in the context of the text. Some of the predefined simple types are: xs:integer, xs:boolean, xs:string, xs:date. For example −

<xs:element name = "phone\_number" type = "xs:int" />

### Complex Type

A complex type is a container for other element definitions. This allows you to specify which child elements an element can contain and to provide some structure within your XML documents. For example −

<xs:element name = "Address">

<xs:complexType>

<xs:sequence>

<xs:element name = "name" type = "xs:string" />

<xs:element name = "company" type = "xs:string" />

<xs:element name = "phone" type = "xs:int" />

</xs:sequence>

</xs:complexType>

</xs:element>

In the above example, *Address* element consists of child elements. This is a container for other **<xs:element>** definitions, that allows to build a simple hierarchy of elements in the XML document.

### Global Types

With the global type, you can define a single type in your document, which can be used by all other references. For example, suppose you want to generalize the *person* and *company* for different addresses of the company. In such case, you can define a general type as follows −

<xs:element name = "AddressType">

<xs:complexType>

<xs:sequence>

<xs:element name = "name" type = "xs:string" />

<xs:element name = "company" type = "xs:string" />

</xs:sequence>

</xs:complexType>

</xs:element>

Now let us use this type in our example as follows −

<xs:element name = "Address1">

<xs:complexType>

<xs:sequence>

<xs:element name = "address" type = "AddressType" />

<xs:element name = "phone1" type = "xs:int" />

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name = "Address2">

<xs:complexType>

<xs:sequence>

<xs:element name = "address" type = "AddressType" />

<xs:element name = "phone2" type = "xs:int" />

</xs:sequence>

</xs:complexType>

</xs:element>

Instead of having to define the name and the company twice (once for *Address1* and once for *Address2*), we now have a single definition. This makes maintenance simpler, i.e., if you decide to add "Postcode" elements to the address, you need to add them at just one place.

## Attributes

Attributes in XSD provide extra information within an element. Attributes have *name* and *type* property as shown below −

<xs:attribute name = "x" type = "y"/>

## NoSQL Database

NoSQL Database is used to refer a non-SQL or non relational database.

It provides a mechanism for storage and retrieval of data other than tabular relations model used in relational databases. NoSQL database doesn't use tables for storing data. It is generally used to store big data and real-time web applications.

## ****Characteristics of NoSQL Database****

Although there are different ways that can be incorporated to understand how NoSQL databases work, we will now look at some of the most common features that define a basic NoSQL database.

### ****Complex-free working****

Unlike SQL databases, NoSQL databases are not complicated. They store data in an unstructured or a semi-structured form that requires no relational or tabular arrangement. Perhaps they are easier to use and can be accomplished by all.

### ****Independent of Schema****

Secondly, NoSQL databases are independent of schemas which implies that they can be run over without any predetermined schemas.

That said, they are far more efficient to work with and perhaps this particular feature works well for young programmers and organizations handling large amounts of heterogeneous data that requires no schemas to structure it.

### ****Better Scalability****

One of the most prominent features of such a database is that it has high scalability that makes it suitable for large amounts of data.

Needless to mention that the contemporary data scientists often prefer to work with NoSQL databases due to this feature since it allows them to accommodate humongous data without rupturing its efficacy.

### ****Flexible to accommodate****

 Since such databases can accommodate heterogeneous data that requires no structuring, they are claimed to be flexible in terms of their usage and reliability.

For beginners intending to try their hands in the field, NoSQL databases are easy to handle yet very useful.

### ****Durable****

 If durability is not one of its most striking features, then what is? NoSQL databases are highly durable as they can accommodate data ranging from heterogeneous to homogeneous.

Not only can they accommodate structured data, but they can also incorporate unstructured data that requires no query language. Undoubtedly, these databases are durable and efficient.

**Types of NoSQL databases**

Here are the four main types of NoSQL databases:

* [Document databases](https://www.mongodb.com/scale/types-of-nosql-databases#document-databases)
* [Key-value stores](https://www.mongodb.com/scale/types-of-nosql-databases#key-value-stores)
* [Column-oriented databases](https://www.mongodb.com/scale/types-of-nosql-databases#column-orientated-databases)
* [Graph databases](https://www.mongodb.com/scale/types-of-nosql-databases#graph-databases)

Document databases

A [document database](https://www.mongodb.com/document-databases) stores data in [JSON, BSON](https://www.mongodb.com/json-and-bson), or XML documents (not Word documents or Google Docs, of course). In a document database, documents can be nested. Particular elements can be indexed for faster querying.

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 the data in an application. SQL data must often be assembled and disassembled when moving back and forth between applications and storage.

Document databases are popular with developers because they have the flexibility to rework their document structures as needed to suit their application, shaping their data structures as their application requirements change over time. This flexibility speeds development because, in effect, data becomes like code and is under the control of developers. In SQL databases, intervention by database administrators may be required to change the structure of a database.

The most widely adopted document databases are usually implemented with a scale-out architecture, providing a clear path to scalability of both data volumes and traffic.

Use cases include ecommerce platforms, trading platforms, and mobile app development across industries.

[Comparing MongoDB vs. PostgreSQL](https://www.mongodb.com/compare/mongodb-postgresql) offers a detailed analysis of MongoDB, the leading NoSQL database, and PostgreSQL, one of the most popular SQL databases.

**Key-value stores**

The simplest type of NoSQL database is a [key-value store](https://www.mongodb.com/key-value-database). Every data element in the database is stored as a key value pair consisting of an attribute name (or "key") and a value. In a sense, a key-value store is like a relational database with only two columns: the key or attribute name (such as "state") and the value (such as "Alaska").

Use cases include shopping carts, user preferences, and user profiles.

**Column-oriented databases**

While a relational database stores data in rows and reads data row by row, a column store is organized as a set of columns. This means that when you want to run analytics on a small number of columns, you can read those columns directly without consuming memory with the unwanted data. Columns are often of the same type and benefit from more efficient compression, making reads even faster. Columnar databases can quickly aggregate the value of a given column (adding up the total sales for the year, for example). Use cases include analytics.

Unfortunately, there is no free lunch, which means that while columnar databases are great for analytics, the way in which they write data makes it very difficult for them to be strongly consistent as writes of all the columns require multiple write events on disk. Relational databases don't suffer from this problem as row data is written contiguously to disk.

**Graph databases**

A graph database focuses on the relationship between data elements. Each element is stored as a node (such as a person in a social media graph). The connections between elements are called links or relationships. In a graph database, connections are first-class elements of the database, stored directly. In relational databases, links are implied, using data to express the relationships.

A graph database is optimized to capture and search the connections between data elements, overcoming the overhead associated with JOINing multiple tables in SQL.

Very few real-world business systems can survive solely on graph queries. As a result graph databases are usually run alongside other more traditional databases.

Use cases include fraud detection, social networks, and knowledge graphs.

As you can see, despite a common umbrella, NoSQL databases are diverse in their data structures and their applications.

## ****Applications of NoSQL Databases****

### ****Data Mining****

When it comes to data mining, NoSQL databases are useful in retrieving information for data mining uses. Particularly when it’s about large amounts of data, NoSQL databases store data points in both structured and unstructured formats leading to efficient storage of big data.

Perhaps when a user wishes to mine a particular dataset from large amounts of data, one can make use of NoSQL databases, to begin with. Data is the building block of technology that has led mankind to such great heights.

Therefore, one of the most essential fields where NoSQL databases can be put to use is data mining and data storage.

### ****Social Media Networking Sites****

Social media is full of data, both structured and unstructured. A field that is loaded with tons of data to be discovered, social media is one of the most effective applications of NoSQL databases.

From comments to posts, user-related information to advertising, [social media marketing](https://www.analyticssteps.com/blogs/social-media-marketing-working-and-advantages) requires NoSQL databases to be implemented in certain ways to retrieve useful information that can be helpful in certain ways.

Social media sites like Facebook and Instagram often approach open-source NoSQL databases to extract data that helps them keep track of their users and the activities going on around their platforms.

### ****Software Development****

The third application that we will be looking at is [software development](https://www.analyticssteps.com/blogs/7-top-trends-software-development). Software development requires extensive research on users and the needs of the masses that are met through software development.

However, a developer must be able to scan through data that is available.

Perhaps NoSQL databases are always useful in helping software developers keep a tab on their users, their details, and other user-related data that is important to be noted. That said, NoSQL databases are surely helpful in software development.