**What Do You Mean By Allocation Fragmentation and Replication In Distributed Database?**

In this section on fragmentation and replication in distributed database guide, we help you in understanding replication in databases and distributed systems to a greater depth.

**Data Allocation**

Data Allocation is an intelligent distribution of your data pieces, (called data fragments) to improve database performance and Data Availability for end-users. It aims to reduce overall costs of transaction processing while also providing accurate data rapidly in your DDBMS systems.

Data Allocation is one of the key steps in building your Distributed Database Systems. There are two common strategies used in optimal Data Allocation: Data Fragmentation and Data Replication. In the following sections on fragmentation and replication in distributed databases, we discuss both of these techniques in greater detail.

**Fragmentation and Replication In Distributed Database**

**Data Fragmentation**

Fragmentation is a process of disintegrating relations or tables into several partitions in multiple sites. It divides a database into various subtables and sub relations so that data can be distributed and stored efficiently.

Database Fragmentation can be of two types: horizontal or vertical. In a horizontal fragmentation, each tuple of a relation *r* is assigned to one or more fragments. In vertical fragmentation, the schema for a relation *r* is split into numerous smaller schemas with a common candidate key and a special attribute. More details on horizontal and vertical fragmentation will be discussed in the next section.

**Data Replication**

Distributed Database Replication is the process of creating and maintaining multiple copies (redundancy) of data in different sites. The main benefit it brings to the table is that duplication of data ensures faster retrieval. This eliminates single points of failure and data loss issues if one site fails to deliver user requests, and hence provides you and your teams with a fault-tolerant system.

However, Distributed Database Replication also has some disadvantages. To ensure accurate and correct responses to user queries, data must be constantly updated and synchronized at all times. Failure to do so will create inconsistencies in data, which can hamper business goals and decisions for other teams.

**Methods of Data Fragmentation of a Table**

In this section of our fragmentation and replication in distributed database guide, we discuss the two fundamental fragmentation strategies: horizontal and vertical. In addition to these, Distributed Database Management Systems also allow the nesting of fragments in a hybrid fashion, called Hybrid Fragmentation. This will be discussed separately in our third fragmentation strategy.

* [Horizontal Fragmentation](https://hevodata.com/learn/fragmentation-and-replication-in-distributed-database/#horizontal)
* [Vertical Fragmentation](https://hevodata.com/learn/fragmentation-and-replication-in-distributed-database/#vertical)
* [Hybrid Fragmentation](https://hevodata.com/learn/fragmentation-and-replication-in-distributed-database/#hybrid)

**Horizontal Fragmentation (or Sharding)**

Image Source: [IJICT (*Parul Tomar, 2014*)](https://www.ripublication.com/irph/ijict_spl/ijictv4n2spl_15.pdf)

A Horizontal Fragmentation strategy divides a table horizontally by selecting a subset of rows in accordance with values of one or more fields. After partition, these data fragments are assigned to different sites of a Distributed Database System. When a user makes a complete table request, these fragments are then combined using a ***union*** operation.

There are two versions of Horizontal Fragmentation: **Primary Horizontal Fragmentation**, which uses predicates of relation to perform fragmentation, and **Derived Horizontal Fragmentation**, which uses predicates defined on another relation to partition a relation.

Horizontal Fragmentation allows parallel processing of a relation. You can also split a global table into tuples and allocate them to places where they are most frequently accessed for efficient data storage and better access.

**Vertical Fragmentation**

Image Source: [IJICT (*Parul Tomar, 2014)*](https://www.ripublication.com/irph/ijict_spl/ijictv4n2spl_15.pdf)

Vertical Fragmentation splits a table vertically by attributes or columns. In this case, data fragments keep only certain attributes of the original table. They are then assigned to different sites of a DDBMS.

Every data fragment gets a primary key that is required while restoring the original table. The fragmentation is done in such a way that reconstructing a table from fragments only requires a normal *JOIN* operation. To do so, a specific property called *Tuple-id* is added to the schema.

Vertical Fragmentation is highly useful for cases when you want to enforce data privacy.

**Hybrid Fragmentation**

Out of the two discussed in fragmentation and replication in distributed databases, Hybrid Fragmentation takes a different approach. It comprises a combination of both Horizontal and Vertical Fragmentation.

Here the tables are initially fragmented in any form (horizontal or vertical) and then these fragments are partially replicated across different sites according to the frequency of accessing the database fragments. In this case, the original table can be reconstructed by applying *union* and natural *JOIN* operations in the appropriate order.

**Advantages and Disadvantages of Fragmentation**

Database Fragmentation improves Data Accessibility and provides faster transaction processing to user queries. Using fragmentation, you can decompose a relation into multiple independent units so that your users can perform a number of transactions, and retrieve data concurrently without any noticeable lag.

However, Data Fragmentation raises some difficulties as well. Fragmentation and replication in distributed databases must ensure fault tolerance and zero data loss while reconstructing your original table from its fragments. This must happen correctly and at all times whenever your users pass a request.

Moreover, your database fragments must be split up “*sensibly*” so that users with a high demand volume can request and receive data from fragmented tables quickly. In other words, your **Database** **Fragmentation should ensure high query performance** and **concurrent user processing**. Additionally, you must be mindful of the need to reduce dispersed joins throughout the process, which can inevitably add to your costs.

Here, in this section on fragmentation and replication in distributed database guide, we discuss the pros and cons of Database Fragmentation. Let’s have a closer look at those.

**Advantages**

Using Database Fragmentation, you and your teams can:

* Concurrently execute a number of transactions.
* Capitalize on parallel processing of a single query.
* Take advantage of increased system throughput.
* Store data efficiently, by saving frequently used data close to the site of usage.
* Use local query optimization.
* Preserve the security and privacy of your database systems.
* Benefit from fault-tolerance architecture with better disaster recovery mechanisms.

**Disadvantages**

Database Fragmentation falls short in the following scenarios:

* When application views are defined on more than one fragment, they can develop conflicting requirements.
* When doing recurrent fragmentation, the reconstruction task might become rather large.
* In simple operations like checking for dependencies, which might result in chasing data across several sites.
* When data from several fragments is required, access times can be extremely fast.

**Data Replication In DBMS**

Data Replication duplicates your database and creates multiple copies at two or more sites. This is useful for providing continuous system availability, performance, and scalability to fulfill your application requirements. It creates a system of distributed databases, which can in turn help your users access data relevant to their tasks without interfering with the work of other users.

**Advantages of Data Replication**

* **Data Reliability**: Your databases continue to work even in case of a site failure. Using Distributed Database Replication, you can request and receive the same copy from a different site.
* **Scalability:**As your systems grow geographically and in terms of the number of locations (and hence the number of access requests), replication provides a seamless way to handle this expansion without compromising on response times.
* **Quicker Response**: Data Replication enables copies of data to be available close to their access sites. This method of localization delivers quick query processing and consequently fast response times.
* **Simpler Transactions**: With Data Replication, user transactions become simple since they require fewer table joins and minimal coordination across the network.

**Disadvantages of Data Replication**

* **High Storage Requirements**: If your databases are of a gigantic scale, creating and maintaining copies of those databases will demand a high storage capacity.
* **Increased Costs and Complexity**: More copies mean more storage costs. And with every update, your DDBMS system must ensure that new changes are reflected in all the copies of the data at all sites.
* **Undesirable Application – Database Coupling**: Inherent to data update mechanisms are possibilities of Data Inconsistency. Eliminating those requires complex coordination and careful investigation at the application level.

**Distributed Database Replication Schemas**

You can perform Data Replication in DDBMS in either full or partial forms. A **Full Replication** is when you create full copy clones of your existing database and store them on multiple sites. A **Partial Replication** is when you copy and store only the frequently used fragments of your database, leaving the rest. A **No Replication** is when you exclusively hold fragments at one site, without any duplication.

* [Full Replication](https://hevodata.com/learn/fragmentation-and-replication-in-distributed-database/#full)
* [Partial Replication](https://hevodata.com/learn/fragmentation-and-replication-in-distributed-database/#partial)
* [No Replication](https://hevodata.com/learn/fragmentation-and-replication-in-distributed-database/#no)

In this section of fragmentation and replication in distributed databases guide, we discuss these replication schemes in detail:

**Full Replication**

Image Source: [ManageEngine](https://www.manageengine.com/device-control/images/full-replication1.png)

Full Replication is a process of cloning your database entirely. In Full Replication, you would create full copies of your database at every site of the distributed system. Although this improves system availability, since you have multiple backups of the same database, this process takes a long time to take effect.

Full Replication is appropriate for worldwide enterprises with teams that require complete data access. Using Full Replication, for example, North American counterparts can take up the remaining work from their European counterparts if their servers fail.

**Advantages of Full Replication**

* Guaranteed High Availability.
* Rapid retrieval of global queries.
* Faster query execution.
* Negligible communication costs.

**Disadvantages of Full Replication**

* Processing concurrent queries is difficult.
* Slower update process due to bulk duplication.

**Partial Replication**

Image Source: [ManageEngine](https://www.manageengine.com/device-control/images/partial-replication1.png)

Partial Replication is a process of replicating ***only*** frequently used fragments of your database. Here, each fragment is replicated based on its importance and user demand. The number of copies can range from one to many, depending upon the total number of available nodes in the Distributed Database Management System.

Partial Replication is a very helpful Distributed Database Replication Schema for mobilized teams. They can use it to carry relevant and important pieces of data on their laptops and mobiles and periodically synchronize it with the main server once they are back online.

More so than this, even for some businesses, keeping confidential information inside their private network is a necessity. Teams who want to use such datasets can duplicate only the necessary pieces and sync them back once they are complete. Alternatively, they can continue working on their own datasets, leaving the secret data on the private network, and then returning to make the necessary changes, all using Partial Replication.

**Advantages of Partial Replication**

* Better database performance.
* Duplication of only relevant data fragments ensures better resource utilization and fewer costs.
* Increased Data Availability.

**Disadvantages of Partial Replication**

* Keeping all data current and relevant could be a challenge.
* Performance may suffer when a user requests less used data fragments.

**No Replication**

Image Source: [ManageEngine](https://www.manageengine.com/device-control/webp/no-replication3.webp)

When your database fragments are partitioned and exist in singularity, you have a No Replication situation. In this case, each fragment resides at only one site, and therefore failure of one site might result in users losing access to data.

Of all the distributed database replication schemas, No Replication is the fastest to perform, but provides lower data availability and is slow to execute user queries when multiple users request at the same time.

**Advantages of No Replication**

* Easy recovery of data.

**Disadvantages of No Replication**

* Slow execution time in case of concurrent user requests.
* High probability of data loss; no replicas.