**PART 1**

**Summary**

This paper gives a brief introduction about distributed systems. It introduces the concept by talking about how an organization will decide to store its databases when an organization is geographically dispersed. It may either store it on a central database server or decide to distribute them to local servers. This paper talks about distributing data which is spread physically across computers in multiple locations that are connected by a data communications network. It logically belongs to the same system but is spread over the sites of a computer network. The sites of the distributed database may be allocated in the same space and have the same network address but the communication between them is done over a network instead of sharing memory. These systems are mainly concerned with delivery of data to points of query. Then, it talks about how the database distribution is achieved with the help of data fragmentation and describes the process of distribution of data as a collection of fragmentation, replication and allocation.

Moreover, it talks about the fragmentation of data, its types, its importance and advantages as well as disadvantages.

The distributed database design process consists of three phrases – Initial Design, Redesign and materializing that redesign. Distributed database system – makes fragments of classes – in case of object-oriented database or relations – in case of relational database. This paper describes fragmentation as a technique to split a single class or relation of a database into two or more partitions, also; the combination of the partitions should support the original database without any loss of information. Fragmentation helps to improve reliability, performance, balanced storage capacity, communication costs, security.

Fragmentation is carried out on the basis of two parameters

1. Quantitative information: cardinality of relations, frequency of queries, site where query is run, selectivity of the queries, etc.
2. Qualitative information: predicates in queries, types of access of data, read/write, etc.

There are three types of fragmentations.

**Horizontal**

Horizontal fragmentation (HF) allows a relation or class to be partitioned into disjoint tuples which has unique rows but same columns and each fragment is stored at a different node. It stems from the intuition that every site should hold all information that is used to query at the site and the information at the site should be fragmented so the queries of the site run fast.

For example, Human Resources table is having 1000 records can be horizontally fragmented into ten fragments, each fragment having 100 unique records.

**Vertical**

A class or relation in VF will be partitioned into separate sets of columns or attributes except the primary key and these primary key attributes must be included in each set. It’s useful when different sites are responsible for processing different functions involving an entity.

For example, in the Human Resources table; the attributes as EMP\_ID (PRIMARY KEY), F\_NAME, JOB, SALARY, and DEPT\_ID could be Vertically fragmented into two sites such as site1 and site2.

**Mixed**

Mixed fragmentation refers to a combination of horizontal and vertical strategies. A table may be divided into several rows, each one having a subset of the columns. Mixed fragmentation needs two-steps procedures. First of all, horizontal fragmentation is introduced for each site. The horizontal fragmentation yields the subsets of Human Resources horizontal fragments that are located at each site. Vertical fragmentation is used within each horizontal fragment to divide the attributes.

Then, it talks about the correctness rules that should hold true for fragmentation. It talks about three types of correctness rules:

1. Completeness – It ensures no data is lost during the fragmentation process by the correctness property that every data item in the original relation R can be found in at least one of the fragments.
2. Reconstruction – It’s a process of combining the fragments into its original relation. Union is used to combine the fragments created in horizontal fragmentation. Join is used to combine the fragments used in vertical fragmentation. It will need expensive techniques in case of recursive fragmentations.
3. Disjointness - In horizontal fragmentation, where data items are tuples, each tuple should exist in only one fragment to maintain disjointness. In vertical fragmentation, where data items are attributes, each attribute should be present in only one fragment to ensure disjointness.

**PART 2**

**Query.java**

**public String createDB(String query)**

public String createDB(String query) {  
 // Separates the query into Tokens and extracts the desired token  
 query = query.trim();  
 // Create a StringTokenizer object  
 StringTokenizer tokenizer = new StringTokenizer(query);  
 // Skips the CREATE and DATABASE Token  
 tokenizer.nextToken();  
 tokenizer.nextToken();  
 // Extract the nameofthedatabase  
 databaseName = tokenizer.nextToken();  
 // replaces ; at the end to empty string  
 databaseName = databaseName.replace(";", "");  
  
  
 // calls the setDatabaseName and updates the name of the databaseName  
 setDatabaseName(databaseName);  
 // sets the value of the folderpath  
 *folderPath* = new File("C://Users//AVuser//IdeaProjects//Db\_Assignment2//"+databaseName);  
 // calls the setFolderPath and updates the value of the folderPath and databaseName  
 setFolderPath(*folderPath*,databaseName);  
  
  
 // checks if the folderPath already exists, if it doesn't exist then creates the folder  
 // Also returns databaseName  
 if (!*folderPath*.exists()) { // Check if the folder already exists  
 if (*folderPath*.mkdir()) { // Create the folder  
 System.*out*.println("Folder created successfully.");  
 return databaseName;  
 } else {  
 System.*out*.println("Failed to create the folder.");  
 }  
 } else {  
 System.*out*.println("Folder already exists.");  
 }  
 return null;  
}

**Output Screenshot**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Methods** | **Objective** | **Input and Output** | **How ?** | **Code Screenshot** |
| **public String createDB(String query)** | CREATES THE DATABASE FOLDER BASED ON THE DATABASENAME PROVIDED IN THE QUERY |  |  |  |

Overall, this code snippet seems to be handling the creation of a database folder based on a provided query. It extracts the database name from the query, sets the necessary variables, creates the folder, and returns the database name if the folder creation is successful.

The given code appears to be a method called `createDB` within a Java class. Let's break down the code and explain its functionality:

1. The method takes a `String` parameter called `query`, which represents the input query for creating a database. For example, the query could be "CREATE DATABASE db;".

2. The code trims the input query to remove any leading or trailing white spaces.

3. A `StringTokenizer` object named `tokenizer` is created to tokenize the input query string. This tokenizer will split the query into individual tokens based on white spaces.

4. The code calls `tokenizer.nextToken()` twice to skip the first two tokens ("CREATE" and "DATABASE") in the query, as they are not needed for extracting the database name.

5. The next token, representing the database name, is extracted using `tokenizer.nextToken()` and assigned to the variable `databaseName`.

6. The code removes the semicolon at the end of the `databaseName` string using the `replace()` method.

7. The method then calls `setDatabaseName(databaseName)`, which presumably is a setter method to update the name of the database.

8. A `File` object is created with the folder path for the database. The folder path is constructed based on a specific directory structure.

9. The method calls `setFolderPath(folderPath, databaseName)`, which seems to be another setter method that updates the value of the `folderPath` and `databaseName`.

10. The code checks if the `folderPath` already exists using the `exists()` method.

11. If the folder does not exist, it attempts to create the folder using `mkdir()`. If the folder creation is successful, it prints "Folder created successfully." and returns the `databaseName`.

12. If the folder creation fails, it prints "Failed to create the folder."

13. If the `folderPath` already exists, it prints "Folder already exists."

14. If none of the above conditions are met, the method returns `null`.