Unit 5: Object Oriented Database Concepts (6 Hrs.)

Object-oriented programming (OOP) is a way of designing and writing software programs that uses objects as the basic building blocks. Objects are entities that have attributes (data) and behaviors (methods) that can be defined by classes (blueprints) and instantiated (created) as individual instances. OOP allows for creating modular, reusable, and maintainable code that can model real-world entities and problems.
Some of the main concepts of OOP are:

- **Encapsulation**: Encapsulation of operations in DDBMS is the concept of binding data and the operations that can be performed on that data together into a single unit. This allows the data to be hidden from the outside world, and only the operations that are needed to manipulate the data can be accessed.

- **Abstraction**: Abstraction, on the other hand, is the act of representing essential features without including the details. This allows the user to focus on the important aspects of an object without having to worry about the implementation details.

- **Inheritance**: This means creating new classes from existing ones by inheriting their attributes and methods. This helps to avoid code duplication and promote code reuse. For example, a truck class may inherit from a car class and add some new attributes and methods specific to trucks, such as load capacity and unload cargo. The truck class can use all the attributes and methods of the car class, as well as its own.

- **Polymorphism**:
  Polymorphism is the ability of an object to take on different forms or behave in different ways depending on the context in which it is used. In object-oriented programming (OOP), polymorphism is achieved through the use of inheritance, interfaces, and method overriding.

  There are two main types of polymorphism in OOP:

  - Static polymorphism, also known as compile-time polymorphism, occurs when a method with the same name is defined in different classes. The method that is actually called is determined at compile time, based on the type of the object that is being used.
  - Dynamic polymorphism, also known as runtime polymorphism, occurs when a method with the same name is defined in a parent class and overridden in a child class. The method that is actually called is determined at runtime, based on the type of the object that is being used.

•

Object Identity

1. Object identity is a concept that applies to object-oriented data models, where each object has a unique and immutable identifier that distinguishes it from other objects.

2. Object identity allows for defining relationships and associations among objects based on their identifiers rather than their attribute values.

3. Object identity also enables comparing objects based on their identifiers or their states.

4. In a distributed database management system (DDBMS), object identity becomes more challenging to implement and maintain, as objects may be stored and accessed across multiple sites. **A DDBMS must ensure that each object has a globally unique identifier that does not conflict with other objects in the system, and that the identifier remains consistent even if the object is moved or replicated to another site.**

5. There are different approaches for implementing object identity in a DDBMS, such as using system-generated identifiers, user-defined identifiers, or hybrid identifiers. Each approach has its own advantages and disadvantages in terms of performance, scalability, flexibility, and complexity.

Object Strcutrue:

1. Object structure is an imp concept of object-oriented databases, which is widely used in distributed database management system (DDBMS) because provides a way to model real-world entities as an object in the database and enables more natural representation and manipulation of data rather than as tables or records.

2. Object structure refers to the way an object is composed of attributes and methods that define its state and behavior. An object can have simple or complex attributes, which can be atomic values or references to other objects. An object can also have methods, which are functions or procedures that operate on the object's attributes or invoke other methods. For example, a car object may have attributes such as color, model, and owner, and methods such as start, stop, and drive.

3. In a distributed context, each object might be physically stored on one or more nodes (computers) within the distributed system.

Type Constructors:

Type constructors are mechanisms that allow creating new types of objects from existing ones by specifying their structure and behavior. They provide a way to create more complex data structures by combining or extending existing data types. Type constructors can also be used to define complex types, which are composed of other types using certain operators. Some common type constructors are:

1. **Atom**: This is the simplest type constructor that creates atomic types, such as integer, string, or boolean. Atomic types cannot be further decomposed into smaller types.

2. **Tuple**: This is a type constructor that creates tuple types, which are ordered collections of other types. A tuple type can be seen as a record or a row in a table. For example, a person type can be defined as a tuple of name, age, and address.

3. **Set**: This is a type constructor that creates set types, which are unordered collections of other types. A set type can be seen as a table or a relation in a relational database. For example, a department type can be defined as a set of employees.

4. **List**: This is a type constructor that creates list types, which are ordered collections of other types that allow duplicates. A list type can be seen as an array or a vector in a programming language. For example, a shopping list type can be defined as a list of items.

5. **Bag**: This is a type constructor that creates bag types, which are unordered collections of other types that allow duplicates. A bag type can be seen as a multiset or a histogram in mathematics. For example, a word count type can be defined as a bag of word

6. **Array**: This is a type constructor that creates array types, which are ordered collections of other types that have fixed size and allow random access by index. An array type can be seen as a matrix. For example, an image type can be defined as an array of pixels.

<u>Encapsulation of operations</u>:

Encapsulation of operations in Object-Oriented Programming (OOP) is closely related to the concept of encapsulating data. Just as data encapsulation involves binding data and methods together into a single unit. Encapsulation of operations involves binding related methods or functions that perform specific tasks or operations into the same class or object.

Encapsulation of operations is a powerful concept that can be used to improve the security and maintainability of a DDBMS. By hiding the data and the operations that can be performed on that data, encapsulation can help to prevent unauthorized access and errors.

Encapsulation of Operations in DDBMS:

In a DDBMS, encapsulation of operations can be used to define and manage various functionalities related to distributed data. Here's how encapsulation of operations can be applied:

1. **Query Processing and Optimization:** Encapsulating query processing and optimization operations within the DDBMS allows for efficient distribution and execution of queries across the distributed nodes. This includes operations such as query parsing, query optimization, and query execution plan generation.
2. **Data Replication:** Operations related to data replication, synchronization, and consistency can be encapsulated within the DDBMS. This ensures that data is accurately replicated across distributed nodes and that changes are propagated efficiently.
3. **Transaction Management:** Encapsulation of transaction-related operations like commit, rollback, and two-phase commit protocols helps maintain data integrity and consistency across distributed transactions.
4. **Security and Access Control:** Operations for enforcing security measures, authentication, authorization, and access control can be encapsulated within the DDBMS. This ensures that data is accessed and manipulated only by authorized users.
5. **Concurrency Control:** Encapsulating operations related to managing concurrent access to distributed data helps prevent conflicts and maintain data consistency in a multi-user environment.
6. **Data Distribution and Fragmentation:** Operations that involve dividing and distributing data across different nodes in the distributed system can be encapsulated. This includes strategies for data partitioning and replication.
7. **Failure Handling and Recovery:** Operations for detecting failures, initiating recovery processes, and maintaining data availability can be encapsulated to ensure system resilience.

2. Persistence:

Persistence refers to the ability of a software system to store and retrieve data over time. In the context of databases and software applications, data persistence involves saving data to a storage medium (such as a file, database, or memory) so that it can be accessed even after the application's execution has ended.

Encapsulation of Operations, Methods, and Persistence:

Combining encapsulation of operations and methods with data persistence leads to well-structured and maintainable software systems, especially when dealing with data storage and retrieval. Here's how these concepts interact:

Type hierarchy and inheritance:

Type hierarchy and inheritance are related concepts in the field of object-oriented programming (OOP), but they have distinct meanings and implications:

Type Hierarchy:

Type hierarchy refers to the organization of classes or data types in a hierarchical structure based on their relationships and common characteristics. It defines a classification system where classes are grouped into more general (base) types and more specific (derived) types. Type hierarchy involves arranging classes in a way that captures their similarities and differences, allowing for classification and organization of objects.

It focuses on dealing with the broader categorization of classes into a structured hierarchy.

For example, in a type hierarchy for animals, you might have a base type "Animal," and derived types like "Mammal," "Bird," and "Reptile," each of which further has their own subtypes.

Inheritance:
Inheritance is a mechanism provided by object-oriented programming languages that allows a class (subclass or derived class) to inherit attributes and methods from another class (superclass or base class). Inheritance enables the creation of a new class that is based on an existing class, inheriting its properties and behaviors while also having the ability to add or override them as needed.are related concepts in the field of object-oriented programming (OOP), but they have distinct meanings and implications:
It focuses on the mechanism through which a subclass can inherit the properties of a superclass, promoting code reuse and modularity.

Complex objects

Complex objects in the context of a distributed Object-Oriented Database (OODB) refer to data structures that can encapsulate multiple types of data, including both primitive data types (such as integers and strings) and other complex objects(i.e instances of classes that have their own set of attributes and relationships.)

Let's take an example to illustrate this concept:

Consider a scenario where you are building a distributed OODB to manage an e-commerce platform. You want to represent both products and customers in your system, and you want to capture the relationship between orders and products, as well as orders and customers.

```
class Customer:
    def __init__(self, customer_id, name, email):
        self.customer_id = customer_id
        self.name = name
        self.email = email

class Product:
    def __init__(self, product_id, name, price):
        self.product_id = product_id
        self.name = name
        self.price = price


class Order:
    def __init__(self, order_id, customer, products):
        self.order_id = order_id
        self.customer = customer
        self.products = products

# Creating complex objects
customer1 = Customer(1, "Alice", "alice@example.com")
customer2 = Customer(2, "Bob", "bob@example.com")

product1 = Product(101, "Phone", 599.99)
product2 = Product(102, "Laptop", 999.99)
product3 = Product(103, "Tablet", 299.99)

order1 = Order(1001, customer1, [product1, product3])
order2 = Order(1002, customer2, [product2])

# Now you have interconnected complex objects representing customers, products, and orders.
```

Unit 4(Distributed Concurrency Control)

Serializability Theory:

Serializability theory is a theroy that deals with the correctness and consistency of concurrent transactions in a distributed database system. Serializability theory defines the conditions under which a set of concurrent transactions can produce the same result as if they were executed serially, that is, one after another.

Serializability theory is important for distributed concurrency control because it ensures that the concurrent transactions do not interfere with each other and do not violate the integrity and consistency of the distributed database. Serializability theory also helps to improve the performance, availability, and reliability of the distributed database system by allowing more concurrency and parallelism among transactions

Taxonomy of Concurrency Control Mechanisms

Concurrency: The fact of two or more events or circumstances happening or existing at the same time. **Taxonomy:** concerned with classification,

A taxonomy of concurrency control mechanisms is a classification of the different methods and techniques that are used to ensure the correctness and consistency of concurrent transactions in a distributed database system. Concurrency control mechanisms can be categorized based on various criteria, such as the following:

- **The type of synchronization:** Concurrency control mechanisms can be either pessimistic or optimistic. Pessimistic methods prevent conflicts from occurring by using locks, timestamps, or other techniques to coordinate the access and update of shared data. Optimistic methods allow conflicts to occur, but detect and resolve them before committing the transactions.

- **The granularity of locking:** Concurrency control mechanisms can use different levels of locking, such as record-level, page-level, file-level, or database-level. The granularity of locking affects the trade-off between concurrency and overhead. Finer-grained locking allows more concurrency, but incurs more overhead. Coarser-grained locking reduces overhead, but limits concurrency.

- **The location of control:** Concurrency control mechanisms can be either centralized or decentralized. Centralized methods use a single site or a coordinator to manage the concurrency control of all transactions. Decentralized methods distribute the responsibility of concurrency control among multiple sites or participants.

- **The degree of replication:** Concurrency control mechanisms can be either non-replicated or replicated. Non-replicated methods assume that each data item is stored at only one site. Replicated methods assume that each data item can be copied and maintained at multiple sites. Replication can improve the availability and performance of the system, but also increase the complexity and cost of maintaining consistency.

Unit 2 (Distributed Database Design and Access Control)

<u>Distribution Design Issues,</u>

The two fundamental design issues in distributed database are:

- **Fragmentation:** This refers to the process of dividing the database into smaller pieces, called fragments. The fragments can be stored at different sites in the distributed system.
- **Distribution:** This refers to the placement of the fragments at different sites. The objective of fragmentation and distribution is to optimize certain objective functions, such as minimizing the response time of queries, minimizing the communication cost, or maximizing the availability of data.

Here are some other design issues in distributed databases:

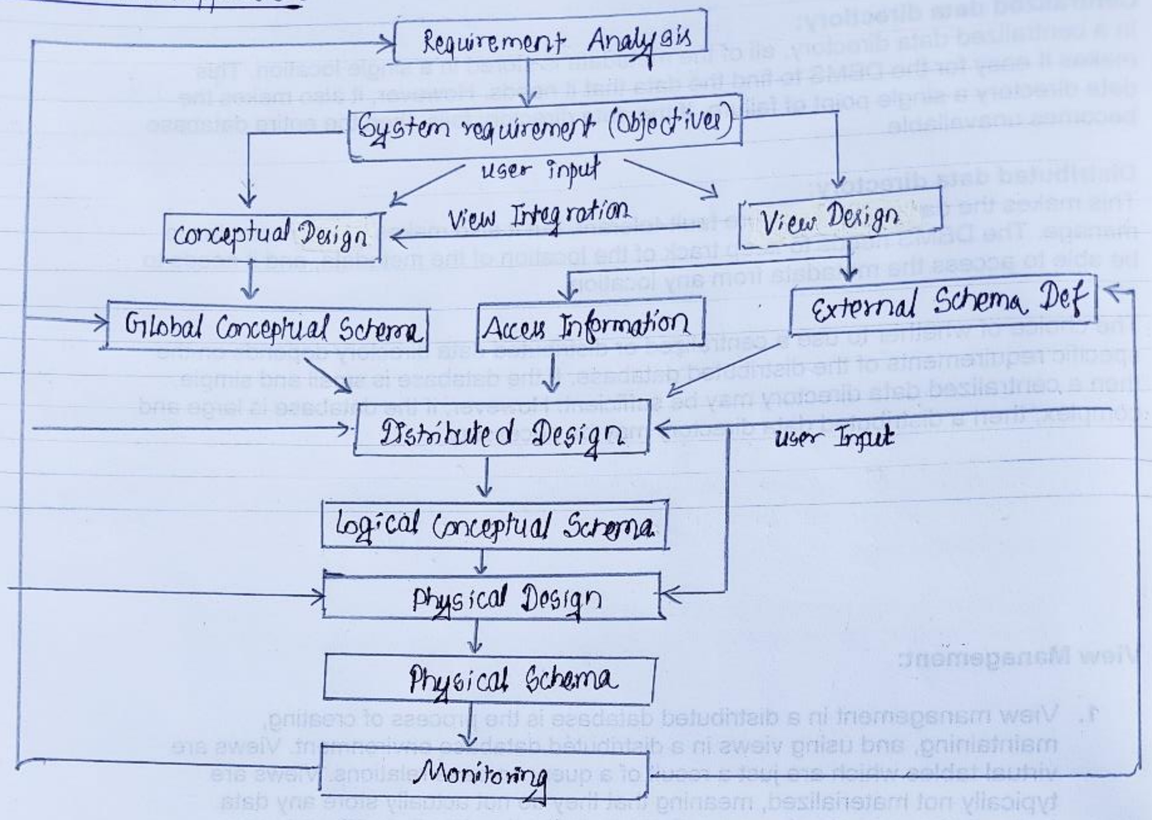
- **Load balancing:**

  This refers to the problem of distributing the workload evenly among the different sites in the distributed system. This is important to ensure that no single site is overloaded.
- **Failure recovery:**
  This refers to the problem of recovering from failures in the distributed system. This includes failures of individual sites, failures of communication links, and failures of the database itself.

- **Heterogeneity:**

  This refers to the fact that the different sites in the distributed system may use different hardware, software, and operating systems. This can make it difficult

Top Down Approach



Top Down Approach :

1) Top Down approach in a distributed Database is suitable for such application where the database is need to be build from scratch.

2) It is mostly found or used homogeneous system. And in the context of distributed database it refers to the approach of designing the overall architecture + structure of distributed database from high level perspective /description.

1) Requirement Analysis : (First step)

a) Requirement Analysis is the First + curical phase in the design of distributed database using top-down approach. In this phase at First we start by gathering different types of useful information such as user needs, system goals, how data should be distributed, communication protocol etc And after analysing these information we can identify system goals and how system should perform

b) This phase involves identyfying + understanding functional + non functional requirements of system.

1) System Requirements : This phase focus on understanding the high level objectives, functionalities, constraints that will guide the design process of entire system
like system should be scalable on increase demands, should be reliabile & purpose of system should be clear, communicatn its cost of data flow should be smooth and less, should clearly perform data handling like how to store data, reterive & share And security, performance, cost effective, userfriendly

6) Distributed Design : Distribute Design is defined by the GCS, Access information and External Schema. All the phase that comes upto this phases will be involved like in centralize manner But phases below Distribution Design will be different / specific to each site.
In this phase decision like how the data & program must be allocated, fragmented, is taken & in some cases design of network is also need to be perform to satisfy the necesity of Distributed database. It is also one of most cirical phase

3) Conceptual Design : (Conceptual Project) This process involves identifying the entities & relationship
a) defining the attribute of each entity, defining the relationship betn entities & normalizing the data model.
b) It provides a high level overview of the data that will be stored in database and helps to ensure that design database meets the user needs.

4) View Design (Logical Project)
It deals with defining the interface for end user
And it have to sub phase/steps/part
a) External schema Defination : which decides that which subset of overall database is relevant to which specific user or application,
means it specify the logical view of data that each user group requires.
b) Access Information : like connection details, authentication creden, protols.
It provides the necessary info for users & applications to interact with distributed database

5) Global Conceptual Schema (GCS)
It is the first schema which gives the high level description of logical structure & organization of entire database. It gives the common view of data that is shared by all users & application without being specific to not any site.

7) Logical Conceptual Schema :
It defines the logical structure & organization of the data at specific site
It gives high level description of data independent to hardware & software.
& provides a way to map from Distributed Design to Physical Design.

8) Physical Design: Comes after LCS. It involves making decision about how the logical design will be implemented ~~implemented~~ at the physical level.
It focuses on determining how data will be stored, organized & distributed in distributed nodes to optimize performance, availibility.

9) Physical Schema:
It defines how logical schema & external schema are mapped onto the physical storages. It specifies the actual storage structuctures, file organization, indexing technique used in distributed database.

10) Monitoring:
This is the final phase which an imp aspects because it ensure weather the designed system is properly functioning as per requirement or not. It involves countinuously observing & measuring various aspect of system behaviour to identify issue, optimize performance, resources utilizatn security and if any issue is found then it will be again send as feedback to above diff phase. feedback

Data Directory:

A data directory in a distributed database design is a repository of metadata about the data that is stored in the database. The metadata includes information such as the location of the data, the schema of the data, and the access permissions for the data.

The data directory can be

a. Centralized
b. Distributed.

Centralized data directlory:
In a centralized data directory, all of the metadata is stored in a single location. This makes it easy for the DBMS to find the data that it needs. However, it also makes the data directory a single point of failure. If the data directory fails, then the entire database becomes unavailable.

Distributed data directory:
This makes the data directory more fault-tolerant, but it also makes it more complex to manage. The DBMS needs to keep track of the location of the metadata, and it needs to be able to access the metadata from any location.

The choice of whether to use a centralized or distributed data directory depends on the specific requirements of the distributed database. If the database is small and simple, then a centralized data directory may be sufficient. However, if the database is large and complex, then a distributed data directory may be necessary.

View Management:

1. View management in a distributed database is the process of creating, maintaining, and using views in a distributed database environment. Views are virtual tables which are just a result of a query on base relations. Views are typically not materialized, meaning that they do not actually store any data. Instead, they are simply a way of representing the data in a different way.

2. View management in a distributed database is more complex than view management in a centralized database. This is because the data that is used to define a view may be stored in multiple locations.

There are two main approaches to view management in a distributed database:

- Local view management: In this approach, each view is managed independently at each site where the data is stored. This approach is simple to implement, but it can lead to inconsistencies between views.
- Centralized view management: In this approach, all views are managed by a central server. This approach ensures that the views are consistent, but it can be more complex to implement.

Unit 3 (Query Processing, Decomposition & Localization)

Query Processing Query:

The main problem in query processing in a distributed database is to find an efficient way to access the data. This involves determining which sites contain the data that is needed to answer the query, and then transferring the data to the site where the query is being evaluated.

There are a number of different approaches to query processing in a distributed database. One approach is to use a centralized query processor. In this approach, the query is sent to a central server, which then retrieves the data from the different sites and evaluates the query. This approach is simple to implement, but it can be a bottleneck if the database is large and complex.

Another approach is to use a distributed query processor. In this approach, the query is divided into multiple subqueries, each of which is sent to a different site. The subqueries are then evaluated independently, and the results are then combined to answer the original query. This approach is more complex to implement, but it can be more efficient for large and complex queries.

Here are some of the other problems in query processing in a distributed database:

- **Data fragmentation:**

  The data that is needed to answer a query may be fragmented across multiple sites. This can make it difficult to access the data and to evaluate the query.

- **Concurrency control:**

  Multiple users may be accessing the data at the same time. This can lead to conflicts if the query is not properly processed.

- **Security:**

  The data needs to be protected from unauthorized access.

Query decomposition

Query decomposition in a distributed database is the process of breaking down a query into subqueries that can be evaluated independently. This is done by analyzing the query and identifying the fragments that are involved.

The query decomposition process can be divided into the following steps:

1. **Scanning the query:**

   The query is scanned to identify the tables and columns that are involved.
2. **Identifying the fragments:**
   The fragments are identified by consulting the data dictionary, which stores information about the location of the data.
3. **Decomposing the query:**
   The query is decomposed into subqueries by assigning each fragment to a subquery.

The query decomposition process can be performed in a centralized or distributed manner. In a centralized implementation, the query decomposition process is performed at a single site. In a distributed implementation, the query decomposition process can be performed at different sites.

The choice of whether to use a centralized or distributed implementation depends on the specific requirements of the distributed database. If the database is small and simple, then a centralized implementation may be sufficient. However, if the database is large and complex, then a distributed implementation may be necessary.

Advantages of query decomposition:

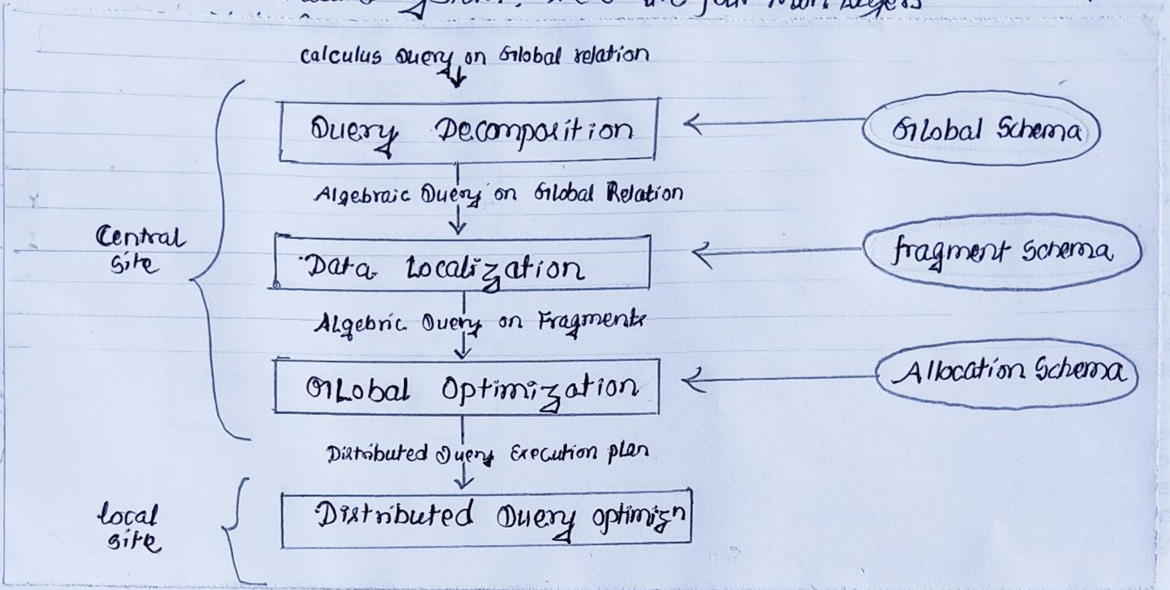
- It can improve the performance of queries by reducing the amount of data that needs to be transferred.
- It can improve the availability of queries by making it possible to execute the subqueries at different sites.

Disadvantages of query decomposition:

- It can add overhead to the query processing process.
- It can make the query processing process more complex.
- It can be difficult to ensure that the different subqueries are properly coordinated.

Layers of Query processing

The layers of Query processing are the set of steps/phases through which each query need to go. for its processing in a distributed Database system. There are four main layers


calculus Query on Global relation
Query Decomposition
Global Schema
Algebraic Query on Global Relation
Central site
Data Localization
fragment Schema
Algebric Query on Fragments
Global Optimization
Allocation Schema
Distributed Query Execution plan
local site
Distributed Query optimizn

1> Query Decomposition This layer breaks down the query into smaller parts, such as the relation involved, the operations to be performed & the constraints on the result. It can be further divided into 3 sub layers

a> Parsing: This sublayer breaks down the query into syntactic components.

b> Semantic Analysis: " Checks that the query makes sense or not

c> Query rewriting: " rewrites the query in the form that becomes more easy and efficient to process.

2> DataLocalization: "

This second layer determines that physical or logical Location of data that are involved in query within the distributed database system.

Divided into two sublayers

a> Fragmentation: This sublayers divides the relations into multiple fragments and store them at different site.

b> Directory maintenance: This sublayers keeps track of the Location of fragments

3> Global Optimization:

The goal of Global Query optimization is to find out the best query plan that minizies the overall cost of executing the query based upon the cost of data distribution & the cost of Executing the query at different site

a> cost Estimation : This sub estimate the cost of executing query plan.

b> Heuristic Search : This sublayer searches for query plan with minimum cost

c> Constraint Satisfaction : " ensure that the selected query plan satisfies constraints.

4) <u>Distributed Query Execution :</u>

Finally this [redacted] furth layer [redacted] executes the selected Query plan. This layer executes each sub query with the required data [redacted] to respective sites & finally collect the result of each sub query & return the combined form of result to the application which has initiated that particular query.

<u>Have below sub layer</u>

a) Data Transfer : This sublayer transfer the necessary data beth sites.

b) Operation Execution :" Executes query/subquery to the appropriate sites.

c) Result Aggregation : " aggregates the result from different sites.

involved in the query

#Localization of Distributed Data:

Localization of the distributed data is the process of determing the physical or Logical location, of data with in the distributed system or network. In the distributed system, data is often spread across multiple nodes, servers, or locations and the ability to locate and access this data efficiently is imp for various application

There are two main approaches to localization of distributed data:

- Data partitioning: This approach divides the data into fragments and stores the fragments at different sites. The fragments are typically stored at sites that are close to the users who are likely to access them.
- Data replication: This approach creates multiple copies of the data and stores the copies at different sites. This can improve the availability of the data by making it possible to access the data even if one of the sites is unavailable.

The choice of which approach to use depends on the specific requirements of the distributed database. If the data is frequently accessed by a small number of users, then data partitioning may be a good approach. If the data is frequently accessed by a large number of users, then data replication may be a good approach.

Here are some of the benefits of localization of distributed data:

- It can improve the performance of queries by reducing the amount of data that needs to be transferred.
- It can improve the availability of data by making it possible to access the data even if one of the sites is unavailable.
- It can improve the scalability of the database by making it possible to add new sites without having to redesign the database.

Here are some of the drawbacks of localization of distributed data:

- It can add complexity to the database design and management.
- It can increase the cost of the database.
- It can make it more difficult to maintain the consistency of the data.

Overall, localization of distributed data can be a beneficial technique for improving the performance, availability, and scalability of distributed databases.