ASSIGNMENT-2

Reg NO:- Y20AIT467

Explain in detail about the types of fragmentation with examples and correctness of fragments?

Types of Fragmentation :-

There are two main types of fragmentation : horizontal and vertical. Horizontal fragments are subsets of tuples and vertical fragments are subsets of attributes, as illustrated. There are also two other types of fragmentation: mixed; illustrated and derived, a type of horizontal fragmentation.

Horizontal Fragment

consists of a subset of the tuples of a relation.

Horizontal fragmentation groups together the tuples in a relation that are collectively used by the important transactions. A horizo -ntal fragment is produced by specifying a predicate that performs a restriction on the tuples in the relation. It is defined using the Selection operation of the relational algebra. the Selection operation groups together tuples that have some common property; for example, the tuples are all used by the same application or at the same site. Given a relation R, a horizontal fragment is defined as:

$$\sigma_P(R)$$

where P is a predicate based on one or more attributes of the relation.

Vertical Fragment :-

consists of a subset of the attributes of a relation.

Vertical fragmentation groups together the attributes in a relation that are used jointly by the import transactions. A vertical fragment is defined using the projection operation of the relational algebra. Given a relation R, a vertical fragment is defined as:

$$\Pi_{a1,\ldots\ldots,an}(R)$$

where $a_1, \ldots\ldots a_n$ are attributes of the relation R.

Mixed Fragment :-

Consists of a horizontal fragment that is subsequently vertically fragmented, or a vertical fragment that is then horizontally fragmented.

A mixed fragment is defined using the Selection and Projection operations of the relational algebra. Given a relation R, a mixed fragment is defined as:

$$\sigma_p(\Pi_{a1\ldots\ldots an}(R))$$

or

$$\Pi_{a1\ldots,an}(\sigma_p(R))$$

where P is a predicate based on one or more attributes of R and $a_1, \ldots\ldots a_n$ are attributes of R.

Derived Mixed Fragmentation :-

A horizontal fragment that is based on the horizontal fragmentation of a parent relation.

Some applications may involve a join of two or more relations. If the relations are stored at different locations, there may be a significant overhead in processing the join. In such cases, it may be more appropriate to ensure that the relations, or fragments of relations, are at the same location. We can achieve this using derived horizontal fragmentation.

We use the term child to refer to the relation that contains the foreign key and parent to the relation containing the targeted primary key. Derived fragmentation is defined using the semijoin operation of the relational algebra. Given a child relation R and parent S, the derived fragmentation of R is defined as:

$$R_i = R \triangleright_f S_i \qquad 1 \leq i \leq w$$

where w is the number of horizontal fragments defined on S and f is the join attribute.

Examples:
Horizontal fragmentation

Assuming that there are only two property types, Flat and House, the horizontal fragmentation of PropertyforRent by property type can be obtained as follows:

$P_1 : \sigma_{type = 'House'}(PropertyforRent)$

$P_2 : \sigma_{type = 'Flat'}(PropertyforRent)$

- Completeness :- Each tuple in the relation appears in either fragment P_1 or P_2.

- Reconstruction :- The PropertyforRent relation can be reconstructed from the fragments using the union Operation.

$$P_1 \cup P_2 = \text{PropertyforRent}$$

- Disjointness :- the fragments are disjoint; there can be no property type that is both 'House' and 'Flat'.

vertical fragmentation :-

S_1 : $\Pi_{staffNO, position, sex, DOB, salary}(Staff)$

S_2 : $\Pi_{staffNO, fName, lName, branchNo}(Staff)$

- Completeness :- Each attribute in the Staff relation appears in either fragment S_1 or S_2.

- Reconstruction :- The staff relation can be reconstructed from the fragments using the Natural join operation,

$$S_1 \bowtie S_2 = Staff$$

- Disjointness :- The fragments are disjoint except for the primary key, which is necessary for reconstruction.

Mixed fragmentation :-

Vertically fragmented Staff for the payroll and personnel departments into:

S_1 : $\Pi_{staffNO, position, sex, DOB, salary}(Staff)$

S_2 : $\Pi_{staffNO, fName, lName, branchNO}(staff)$

'S_2' should now horizontally fragment S_2 according to branch number.

S_{21} : $\sigma_{branchNo = 'B003'}(S_2)$

S_{22} : $\sigma_{branchNO = 'B005'}(S_2)$

S_{23} : $\sigma_{branchNO = 'B007'}(S_2)$

completeness - Each attribute in the staff relation appears in in either fragments S_1 or S_2; each (part) tuple appears in fragment S_1 and either fragment S_{21}, S_{22}, or S_{23}

- Reconstruction - the staff relation can be reconstructed from the fragments using the Union and Natural join operations

$$S_1 \bowtie (S_{21} \cup S_{22} \cup S_{23}) = Staff$$

- Disjointness - the fragments are disjoint; there can be no staff member who works in more than one branch and S_1 and S_2 are disjoint except for the necessary duplication of primary key.

Derived Horizontal Fragmentation:-

$$S_3 = \sigma_{branchNO = 'B003'}(Staff)$$

$$S_4 = \sigma_{branchNO = 'B005'}(Staff)$$

$$S_5 = \sigma_{branchNO = 'B007'}(Staff)$$

Correctness of fragmentation:-

Fragmentation cannot be carried out haphazardly. There are three rules that must be followed during fragmentation:

(1) completeness - If a relation instance R is decomposed into fragments $R_1, R_2, \ldots, R_n$, each data item that can be found in R must appear in at least one fragment. This rule is necessary to ensure that there is no loss of data during fragmentation.

(2) Reconstruction - It must be possible to define a relational operation that will reconstruct the relation R from the

fragments. This rule ensures that functional dependencies are preserved.

(3) Disjointness – If a data item di appears in fragment Ri, then it should not appear in any other fragment. Vertical fragmentation is the exception to this rule, where primary key attributes must be repeated to allow reconstruction. this rule ensures minimal data redundancy.

2. Discuss why the weekness of relational DBMS may make them unsuitable for advanced database applications?

Poor representation of 'real world' entities :-

The process of normalization generally leads to the creation of relations that do not correspond to entities in the 'real world'. the fragmentation of a 'real world' entity into many relations, with a physical representation that reflects this structure, is inefficient leading to many joins during query processing.

Semantic overloading :-

The relational model has only one construct for representing data and relationships between data, namely the relation. For example, to represent a many-to-many (*.*) relationship between two entities A and B, we create three relations, one to represent each of the entities A and B, and one to represent the relationship.

- Poor support for integrity and general constraints

Integrity refers to the validity and consistency of stored data. Integrity is usually expressed in terms of constraints, which are consistency rules that the database is not permitted to violate.

- Homogeneous data structure

the relational model assumes both horizontal and vertical homogeneity. Horizontal homogeneity means that each tuple of a relation must be composed of the same attributes. Vertical homogeneity means that the values in a particular column of a relation must all come from the same domain.

- Limited operations:

the relational model has only a fixed set of operations, such as set and tuple-oriented operations, operations that are provided in the SQL specification. However, SQL does not allow new operations to be specified. Again this is too restrictive to model the behavior of many 'real world' objects.

- Difficulty handling recursive queries

Atomicity of data means that repeating groups are not allowed in the relational model. As a result, it is extremely difficult to handle recursive queries, that is, queries about relationships that a relation has with itself.

- Impedance mismatch:-
  this approach produces an impedance mismatch because we are mixing different programming paradigms:
- SQL is a declarative language that handles rows of data, whereas a high-level language such as 'C' is a procedural language that can handle only one row of data at a time.
- SQL and 3GLs use different models to represent data. For example, SQL provides the built-in data types Date and Interval, which are not available in traditional programming languages.

- Other problems with RDBMSs:-
- Transactions in business processing are generally short-lived and the concurrency control primitives and protocols such as two-phase locking are not particularly suited for long-duration transactions, which are more common for complex design objects.
- Schema changes are difficult. Database administrators must intervene to change database structures and typically, programs that access these structures must be modified to adjust to the new structures.
- RDBMSs were designed to use content-based associative access (that is, declarative statements with selection based on one or more predicates) and are poor at navigational access (that is, access based on movement between individual records).

xplain the functions and architectures of DDBMS with diagram?

Functions of a DDBMS :-

DDBMS to have the following functionality;

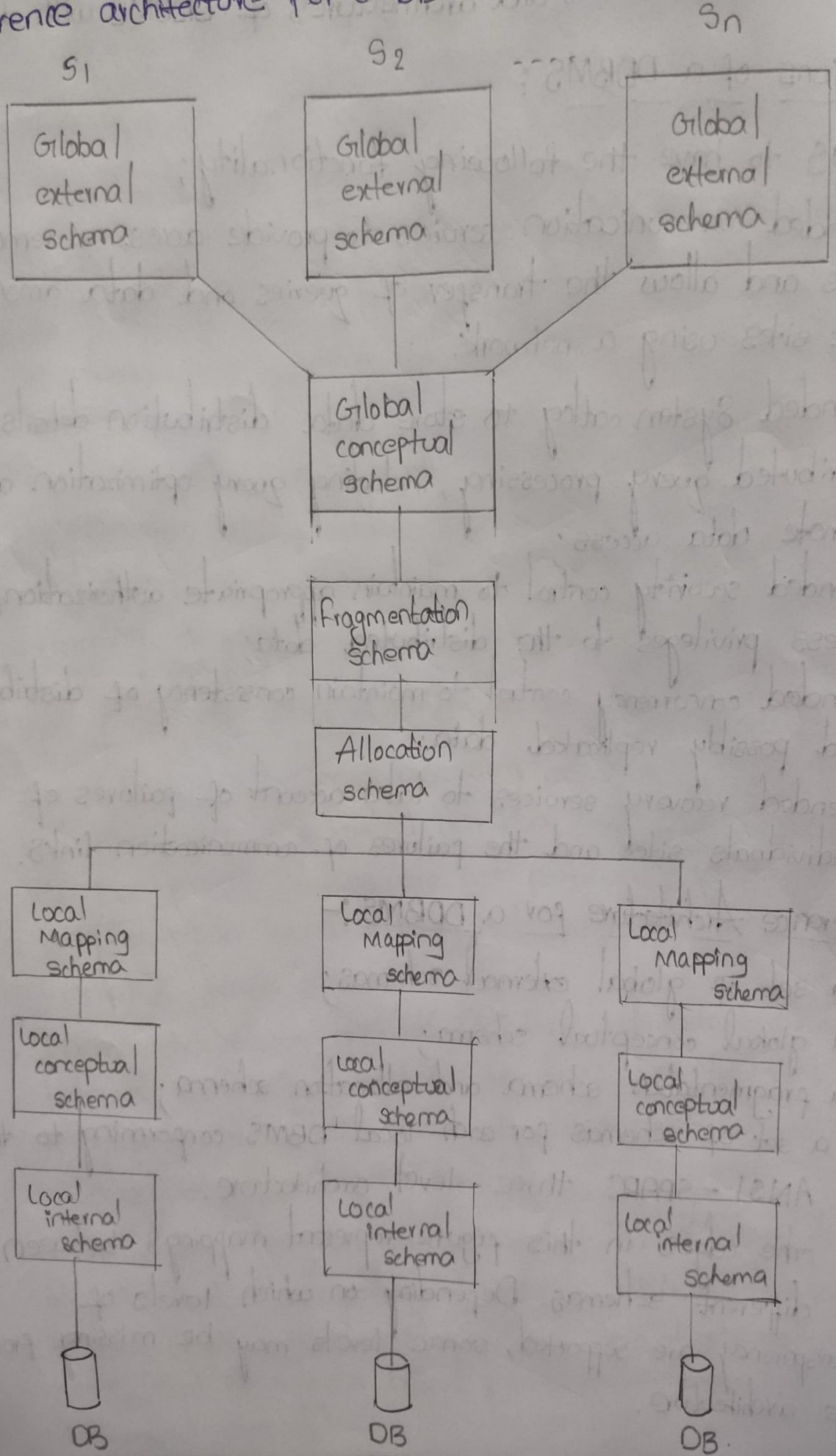
- extended communication services to provide access to remote sites and allow the transfer of queries and data among the sites using a network.
- extended system catlog to store data distribution details;
- distributed query processing, including query optimization and remote data access;
- extended security control to maintain appropriate authorization/ access privileges to the distributed data;
- extended concurrency control to maintain consistency of distributed and possibly replicated data;
- extended recovery services to take account of failures of individuals sites and the failures of communication links.

Reference Architecture for a DDBMS :-

- a set of global external schemas;
- a global conceptual schema;
- a fragmentation schema and allocation schema;
- a set of schemas for each local DBMS conforming to the ANSI - SPARC three - level architecture.

the edges in this figure represent mappings between the different schemas. Depending on which levels of transparency are supported, some levels may be missing from the architecture.

Reference architecture for a DDBMS.



Global conceptual schema :-

The global conceptual schema is a logical description of the whole database, as if it were not distributed. this level corresp -onds to the conceptual level of the ANSI-SPARC architecture and contains definitions of entities, relationships, constraints, security, and integrity information.

Fragmentation and allocation schemas :-

The fragmentation schema is a description of how the data is to be logically partitioned. the allocation schema is a descri -ption of where the data is to be located, taking account of any replication.

Local schemas :-

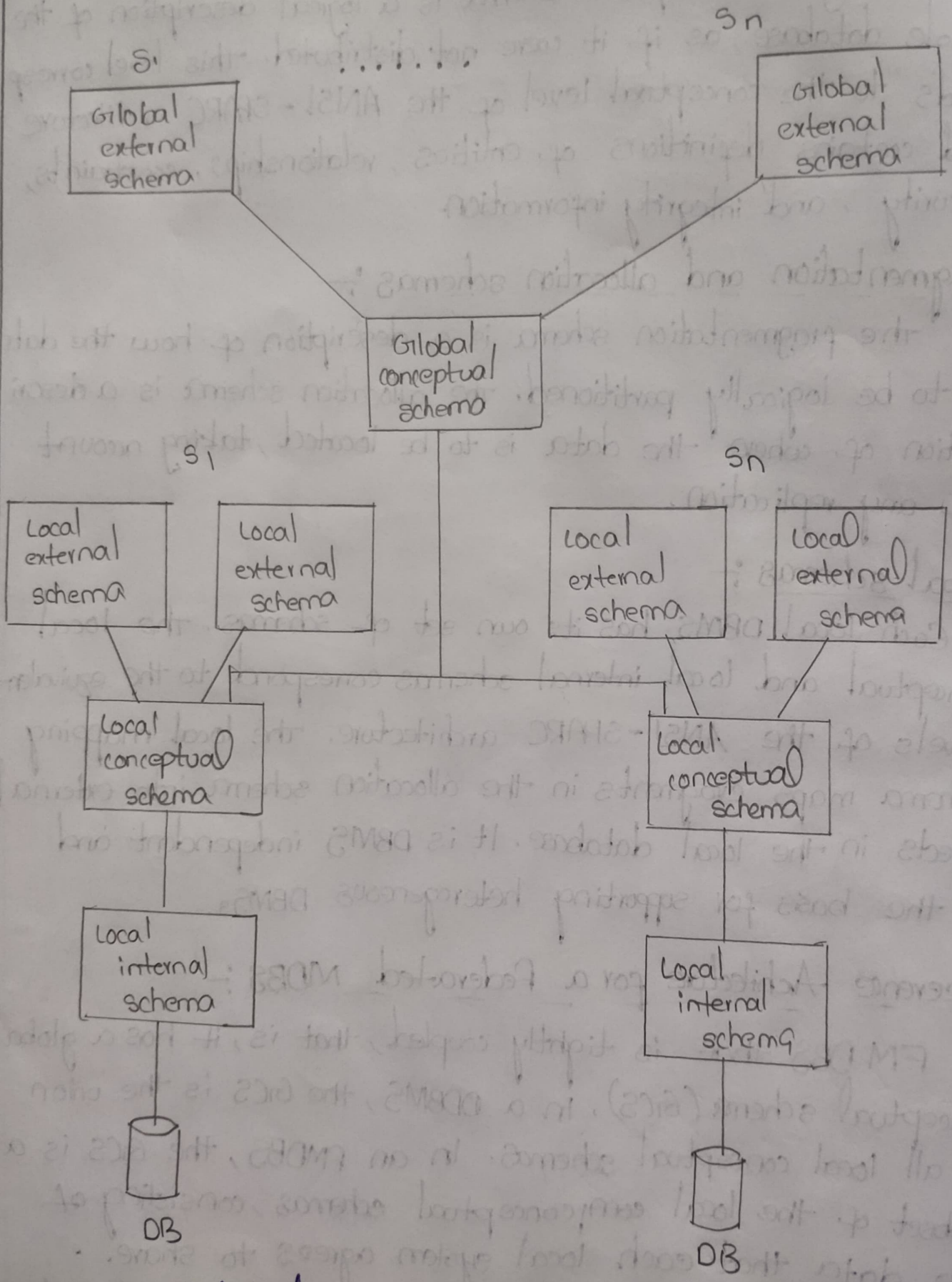
Each local DBMS has its own set of schemas. the local conceptual and local internal schemas correspond to the equivalent levels of the ANSI-SPARC architecture. the local mapping schema maps fragments in the allocation schema into external objects in the local database. It is DBMS independent and is the basis for supporting heterogeneous DBMSs.

Reference Architecture for a Federated MDBS :-

FMDBS that is tightly coupled, that is, it has a global conceptual schema (GCS). In a DDBMS, the GCS is the union of all local conceptual schemas. In an FMDBS, the GCS is a subset of the local ~~com~~ conceptual schemas, consisting of the data that each local system agrees to share.

It has been argued that an FMDBS should not have a GCS (Litwin, 1988), in which case the system is

referred to as loosely coupled.

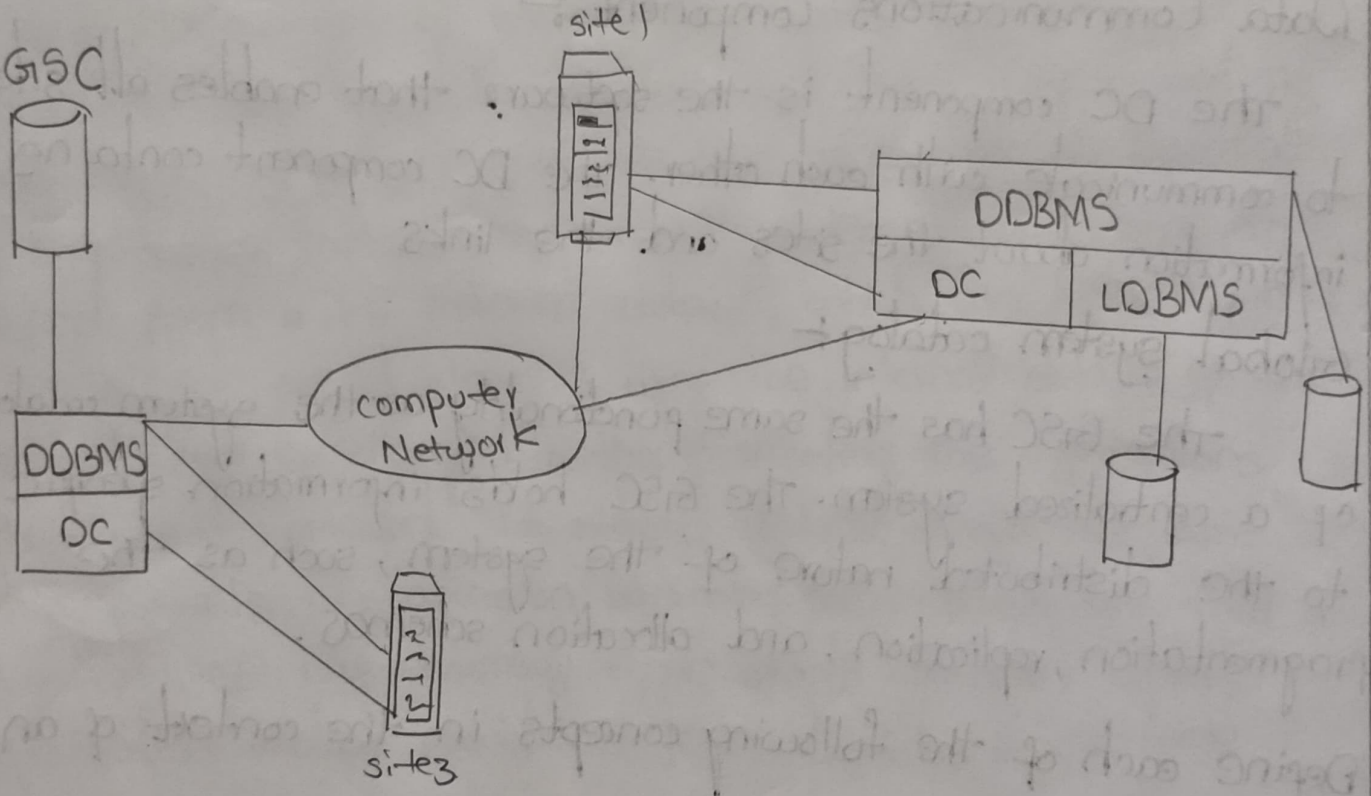


Reference ~~Architecture~~ for a tightly coupled FMDBS.

Component Architecture for a DDBMS :-

Independent of the reference architecture, we can identify a component architecture for a DDBMS consisting of four major components:

- local DBMS (LDBMS) component;
- data communications (DC) component;
- global system catalog (GSC);
- distributed DBMS (DDBMS) component.



component of a DDBMS.

The component architecture for a DDBMS based on figure is illustrated in figure. We have omitted site 2 from the diagram as it has the same structure as site 1

Local DBMS component

The LDBMS component is a standard DBMS, responsible for controlling the local data at each site that has a database. It has its own local system catalog that stores information about the data held at that site. In a homogeneous system, the LDBMS component is the same product, replicated at each site. In a heterogeneous system, there would be at least two sites with different DBMS product and/or platforms.

Data Communications Component :-

The DC component is the software that enables all sites to communicate with each other. The DC component contains information about the sites and the links.

Global system catalog :-

The GSC has the same functionality as the system catalog of a centralized system. The GSC holds information specific to the distributed nature of the system, such as the fragmentation, replication, and allocation schemas.

1. Define each of the following concepts in the context of an OODM? a) Abstraction
   b) Encapsulation
   c) Object Identity
   d) Inheritance.

a) Abstraction :-

Abstraction is the process of identifying the essential aspects of an entity and ignoring the unimportant properties.

software engineering this means that we concentrate on what an object is and what it does before we decide how it should be implemented. In this way we delay implementation details for as long as possible, thereby avoiding commitments that we may find restrictive at a later stage. There are two fundamental aspects of abstraction: encapsulation and information hiding.

b) Encapsulation:-

The concept of encapsulation means that an object contains both the data structure and the set of operations that can be used to manipulate it. The concept of information hiding means that we separate the external aspects of an object from its internal details, which are hidden from the outside world. In this way the internal details of an object can be changed without affecting the applications that use it, provided the external details remain the same. this prevents an aplicaton becoming so interdependent that a small change has enormous ripple effects. In other words information hiding provides a form of data independence.

c) Object Identity:-

In an object-oriented system, each object is assigned an Object Identifier (OID) when it is created that is:

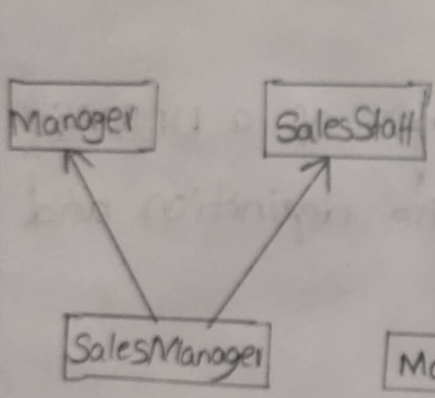
- System-generated;
- unique to that object
- invariant, in the sense that it cannot be altered during its lifetime. Once the object is created, this OID will

will not be reused for any other object, even after the object has been deleted;
- independent of the values of its attributes (that is, its state). Two objects could have the same same state but would have different identities;
- Invisible to the user (ideally).

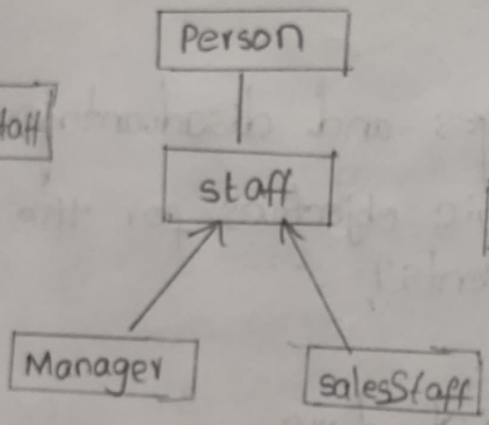
there are several advantages to using OIDs as the mechanism for object identity;
- they are efficient - OIDs require minimal storage within a complex object. Typically, they are smaller than textual names, foreign keys, or other semantic-based references
- They are fast - OIDs point to an actual address or to a location within a table that gives the address of the referenced object. this means that objects can be located quickly whether they are currently stored in local memory or on disk.
- they cannot be modified by the user - if the OIDs are system-generated and kept invisible, or at least read-only, the system can ensure entity and referential integrity more easily. Further, this avoids the user having to maintain integrity.
- they are independent of content - OIDs do not depend upon the data contained in the object in any way. this allows the values of every attribute of an object to change, but for the object to remain the same object with the same OID.

Inheritance:

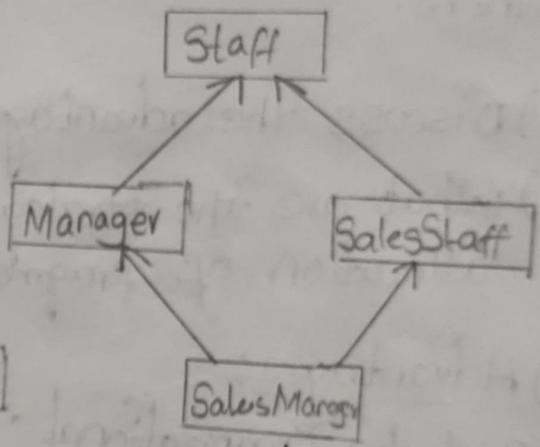
Inheritance allows one class to be defined as a special case of a more general class. these special cases are known as subclasses and the more general cases are known as superclasses. the process of forming a superclass is referred to as generalization and the process of forming a subclass is specialization. By default, a subclass inherits all the properties of its superclass and additionally, defines its own unique properties. However, as we see shortly, a subclass can also redefine inherited properties.



Multiple inheritance



Single inheritance



Repeated inheritance

Single inheritance:

where the subclasses Manager and SalesStaff inherit the properties of superclass staff. the term single inheritance refers to the fact that the subclasses inherit from no more than one superclass. the superclass staff could itself be a subclass of a superclass, person, thus forming a class hierarchy.

Multiple inheritance:-

where the subclass SalesManager inherits properties from both the superclasses Manager and SalesStaff.

Repeated inheritance:-

It is a special case of multiple inheritance where the superclasses inherit from a common superclass. Extending the previous example, the classes Manager and Salesstaff may both inherit properties from a common superclass Staff,

Selective inheritance:-

It allows a subclass to inherit a limited number of properties from the superclass. this feature may provide similar functionality to the view mechanism discussed in Section c.4. by restricting access to some details but not others.

5.) a) Discuss the advantages and disadvantages of a DDBMS?

b) what are the strategic objectives for the definition and allocation of fragments?

a) Advantages:-

- Reflects Organizational structure

Many organizations are naturally distributed over several locations.

- Improved shareability and local autonomy

the geographical distribution of an organization can be reflected in the distribution of the data; users at one site can access data stored at other sites. Data can be placed at the site close to the users who normally uses the data

- Improved availability

in a centralized DBMS, a computer failure terminates the operations of the DBMS. However, a failure at one site of a DDBMS, or a failure of a communication link making

inoperable.

- Improved reliability

As data may be replicated so that it exists at more than one site, the failure of a node or a communication link does not necessarily make the data inaccessible.

- Improved Performance

As the data is located near the site of 'greatest demand' and given the inherent parallelism of distributed DBMSs, speed of database access may be better than that achievable from a remove centralized database.

- Economics

In the 1960s, computing power was calculated according to the square of the costs of the equipment; three times the cost would provide nine times the power. this was known as Grosch's law.

- Modular Growth

in a distributed environment, it is much easier to handle expansion. New sites can be added to the network without affecting the operations of other sites

- Integration

A the same time, no one package can provide all the functionality that an organization requires. thus, it is important for organizations to be able to integrate software components from different vendors to meet their specific requirements.

• Remaining Competitive
Many enterprises have had a reorganize their business and use distributed database technology to remain competitive.

Disadvantages:-

• Complexity
A distributed DBMS that hides the distributed nature from the user and provides an acceptable level of performance, reliability and availability is inherently more complex than a centralized DBMS.

• Cost
Increased complexity means that we can expect the procurement and maintenance costs for a DDBMS to be higher than those for a centralized DBMS. Furthermore, a distributed DBMS requires additional hardware to establish a network between sites.

• Security
In a centralized system, access to the data can be easily controlled.

• Integrity control more difficult
Database integrity refers to the validity and consistency of stored data.

• Lack of standards
Although distributed DBMSs depend on effective communication, we are only now starting to see the appearence of standard communication of data access protocols

• Lack of experience
General-purpose distributed DBMSs have not been widely

accepted, although many of the protocols and problems are well understood.

- Database design more complex

Besides the normal difficulties of designing a centralized database, the design of a distributed database.

b) 1. Flexibility and modularity - fragmentation allows software systems to be more flexible and modular, providing the ability to modify, update or replace components or modules, update or replace components or modules independently.

2. Scalability - Fragmentation can facilitate effective resource allocation that can scale as the user base or data volume grows.

3. Better Performance -

Fragmentation can help with better utilization of resources, improving overall system efficiency and decreasing response time

4. Security -

Fragmentation can create a more secure system by reducing the impact of breaches, compartmentalizing sensitive data and restricting unauthorized access to specific components of the system.

5. Easy maintenance and troubleshooting-

Fragmentation allows for easier maintenance, testing, debugging and troubleshooting of specific parts of the system, without needing to analyze or modify the entire system.

6. Code reuse-

Fragmentation can facilitate code reuse, reducing the development time and costs for new applications or features.

Overall, the strategic objectives of fragment definition and allocation are aimed at creating robust and efficient software systems that can scale and adapt rapidly with changing needs or emerging technologies.

What layers of transperency should be provided with a DDBMS? Give examples to illustrate your answer. Justify your answer

A DDBMS may provide various levels of transperency. However, they all participate in the same overall objective: to make the use use of the distributed database equivalent to that of a centralized database. We can identify four main types of transperency in a DDBMS.

- distribution transperency;
- transaction transperency;
- performance transperency;
- DBMS transperency;

Distribution Transparency:-

Distribution transparency allows the user to perceive the database as a single, logical entity. If a DDBMS exhibits distribution transparency, then the user does not need to know the data is fragmented or the the location of data items

If the user needs to know that the data is fragmented and the location of fragments then we call this local mapping transparency.

Fragmentation transparency:-

Fragmentation is the highest level of distribution transparency, If fragmentation transparency is provided by the DDBMS, then the user does not need to know that the data is fragmented. As a result, database accesses are based on the global schema, so the user does not need

to specify fragment names or data locations.

Location transparency :-

Location is the middle level of distribution transperency. With location transparency, the user must know how the data has been fragmented but still does not have to know the location of the data.

Replication transparency :-

closely related to location transparency is replication transparency, which means that the user is unaware of the replication of fragments. Replication transparency is implied by location transparency. However, it is possible for a system not to have location transparency but to have replication transparency.

Local mapping transparency :-

this is the lowest level of distribution transparency. With local mapping transperency, the user needs to specify both fragment names and the location of data items, taking into consideration any replication that may exist.

Naming transparency :-

As a corollary to the above distribution transperency, we have naming transparency. As in a centralized database, each item in a distributed database must have a unique name. therefore, the DDBMS must ensure that no two sites create a database object with the same name. One solution to this problem is to create a central name server, which has the responsibility for ensuring uniqueness of all names in the system.

Transaction Transparency:-

Transaction transperancy in a DDBMS environment ensures that all distributed transactions maintain the distributed database's integrity and consistency. A distributed transaction accesses data stored at more than one location. Each transaction is divided into a number of subtransactions, one for each site that has to be accessed; a subtransaction is represented by an agent. It is very complex due to the use of fragmentation, allocation, and replication structure of DBMS.

Performance transperancy:-

Performance transperency requires a DDBMS to perform as if it were a centralized DBMS. In a distributed environment, the system should not suffer any performancy degradation due to the distributed architecture, for example the presence of the network. Performance transperency also requires the DDBMS to determine the most cost-effective strategy to execute request.

In a centralized DBMS, the query processor (QP) must evaluate every data request and find an optimal execution strategy, consisting of an ordered sequence of operations on the database! In a distributed environment, the distributed query processor (DQP) maps a data request into an ordered sequence of operation on the local database. It has the added complexity of taking into account the fragmentation, replication and allocation schemas.

DBMS Transparency :-

DBMS transperancy hides the knowledge that the local DBMSs may be different, and is therefore only applicable to heterogeneous DDBMSs. It is one of the most difficult transperencies to provide as a generalization.