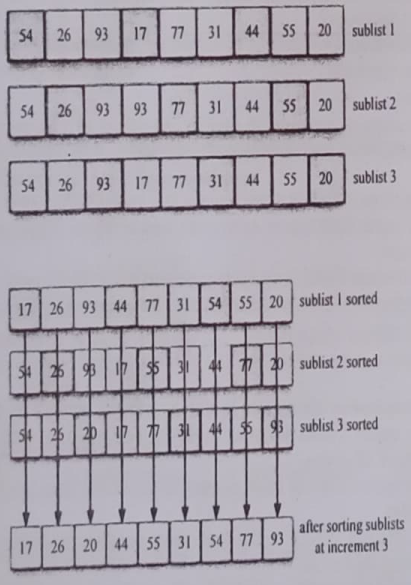
2. Explain shell sort technique. **[MODEL QUESTION]**

Answer:

The shell sort, sometimes called the "diminishing increment sort," improves on the insertion sort by breaking the original list into a number of smaller sub-lists, each of which is sorted using an insertion sort. The unique way that these sub-lists are chosen is the key to the shell sort. Instead of breaking the list into sub-lists of contiguous items, the shell sort uses an increment i, sometimes called the gap, to create a sub-lists by choosing all items that are i items apart. This can be seen in figure. This list has nine items. If we use an increment of three, there are three sub-lists, each of which can be sorted by an insertion sort. After completing these sorts, we get the list shown in figure. Although this list is not completely sorted, something very interesting has happened. By sorting the sub-lists, we have moved the items closer to where they actually belong. Figure shows a final insertion sort using an increment of one, in other words, a standard insertion sort. Note that by performing the earlier sub-list sorts, we have now reduced the total number of shifting operations necessary to put the list in its final order. for this case, we need only four more shifts to complete the process. In the way in which the increments are chosen is the unique feature of the shell sort. The function shell sort shown below uses a different set of increments. In this case, we begin with 2 sub-lists. On the next pass, 4 sub-lists are sorted. Eventually, a single list is sorted with the basic insertion sort. Figure shows the first sub-lists for our example using this increment.



DISTRIBUTED DATABASE MANAGEMENT SYSTEM

NOTE:

MAKAUT course structure and syllabus of 6th semester has been changed from 2021. **DISTRIBUTED DATABASE MANAGEMENT SYSTEM** has been introduced as a new subject in present curriculum. Taking special care of this matter we are providing chapterwise model questions & answer, so that students can get an idea about university questions patterns.

INTRODUCTION

Multiple Choice Type Questions

1. Common problem of designing DDBMS and centralized DBMS are: [MODEL QUESTION]
   a) Design of conceptual schema and design of fragmentation
   b) Design of fragmentation and design of allocation of fragments
   c) Design of allocation of fragments and design of physical database
   d) Design of conceptual schema and design of physical database

Answer: (c)

2. The data dictionary tells the DBMS [MODEL QUESTION]
   a) what files are in the database
   b) what attributes are possessed by the data
   c) what these files contain
   d) all of these

Answer: (d)

3. Which of the following models is used by distributed database system? [MODEL QUESTION]
   a) Mainframe computing model
   b) Disconnected, personal computing model
   c) Client / server computing model
   d) None of these

Answer: (c)

4. Which of the following is not benefit of site autonomy? [MODEL QUESTION]
   a) A global catalog is not necessary to access local data
   b) Node can upgrades software independently
   c) Administrators can recover from isolated system failures independently
   d) No need for backup and recovery

Answer: (d)

5. DDBMS provides better ..... over centralized DBMS. [MODEL QUESTION]
   a) decentralization b) tuning
   c) security d) transparency

Answer: (d)

6. "Fragmentation transparency cannot be achieved without location transparency." [MODEL QUESTION]
   a) True b) False c) Unknown d) None of these

Answer: (b)

7. Global Schema Fragmentation Schema and Allocation Schema reside in [MODEL QUESTION]
   a) one of the machine selected as a coordinator of the DDBMS
   b) the system virtually
   c) all the machines of the DDBMS network
   d) all of these

Answer: (c)

8. Heterogeneous data source needs ..... design for designing DDBMS. [MODEL QUESTION]
   a) Bottom-up b) Top-down c) Flat d) None of these

Answer: (b)

9. The unit of data transfer to and from disk is [MODEL QUESTION]
   a) information b) block c) pages d) file

Answer: (d)

10. Which of the following is not a feature of DDBMS? [MODEL QUESTION]
   a) network transparency b) OS transparency
   c) DBMS transparency d) hardware transparency

Answer: (b)

11. A distributed database has which of the following advantages over a centralized database? [MODEL QUESTION]
   a) software cost b) software complexity
   c) slow response d) modular growth

Answer: (d)

12. An autonomous homogenous environment is which of the following? [MODEL QUESTION]
   a) the same DBMS is at each node and each DBMS works independently
   b) the same DBMS is at each node and a central DBMS coordinates database access
   c) a different DBMS is at each node and each DBMS works independently
   d) a different DBMS is at each node and a central DBMS coordinates database access

Answer: (a)

13. A heterogeneous distributed is which of the following? [MODEL QUESTION]
   a) the same DBMS is used at each location and data are not distributed across all nodes
   b) the same DBMS is used at each location and data are distributed across all nodes
   c) a different DBMS is used at each location and data are not distributed across all nodes
   d) a different DBMS is used at each at each location and data are distributed across all nodes

Answer: (d)

14. A homogenous distributed database is which of the following [MODEL QUESTION]
   a) the same DBMS is used at each location and data are not distributed across all nodes
   b) the same DBMS is used at each location and data are distributed across all nodes
   c) a different DBMS is used at each location and data are not distributed across all nodes
   d) a different DBMS is used at each location and data are distributed across all nodes

Answer: (b)

15. Which of the following is not a feature of DDBMS [MODEL QUESTION]
   a) Replication transparency b) Fragmentation transparency
   c) Locking transparency d) Allocation transparency

Answer: (c)

16. Distributed database system do not help [MODEL QUESTION]
   a) interconnection of existing database
   b) incremental growth of the organization
   c) reduction communication overhead of geographically distributed transactions
   d) reduction of the memory requirement of distributed data

Answer: (c)

17. Top-down approach to the design of data distribution is applicable when [MODEL QUESTION]
   a) we are developing a system from scratch
   b) as the aggregation of existing databases
   c) both of these
   d) none of these

Answer: (a)

18. Distributed Systems should [MODEL QUESTION]
   a) meet prescribed time constraints b) aim better resource sharing
   c) aim better system utilization d) aim low system overhead

Answer: (b)

19. What is data about data called? [MODEL QUESTION]
   a) Metadata b) Data catalog c) Information d) Database

Answer: (a)

20. Which of the following features(s) is (are) important aspects of a distributed database? [MODEL QUESTION]
   a) Distribution b) Logical correlation
   c) Distribution & logical correlation d) Disjointness

Answer: (c)

21. A multiprocessor system where two or more processors share the same primary memory is called [MODEL QUESTION]
   a) homogeneous b) loosely coupled
   c) tightly coupled d) all of these

Answer: (c)

Short Answer Type Questions

1. What is Distributed Data Processing (DDP)? [MODEL QUESTION]

Answer:

Distributed data processing allows distribution of application programs and data among interconnected sites to satisfy the information needs of the organization. Depending on its requirements, an organization may choose to centralize or decentralize its data processing systems.

There are two types of Distributed Data Processing:

Centralized data processing:

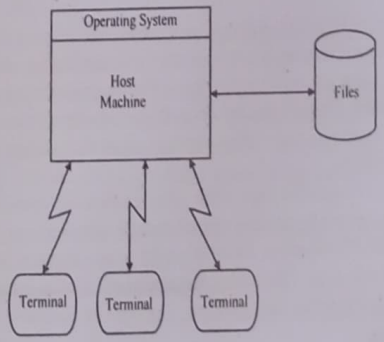
In a centralized system, one machine controls all file access and updates. A centralized system permits a high level of control over application programs and data.

A centralized data processing system is useful when:

- All data is shared across application programs.
- Many end users need access to the same data and also require the most current data available.
- Security has been established as the responsibility of the central site.
- Large volumes of data are involved. In this case, centralized control can help handle the volume of data efficiently.

In a centralized data processing environment, more attention typically is given to direct-access storage devices (that is, their cost and reliability) than to data transmission.

The following illustration shows how one machine controls all file access and updates in a centralized system.



Decentralized data processing:
In a decentralized system, multiple machines control file access and updates to serve the varied needs of end users.
A decentralized system is useful when:

- Data is primarily accessed at a single location. In this case, it is most efficient to store the data where it is used.
- The update rate is high. In this case, distribution of the data across different sites would permit more efficient update operations.
- Security has been established as a local responsibility.
- Data is used in highly specialized ways by end users.

2. Write down the advantages and disadvantages of Distributed Database System?
[MODEL QUESTION]

Answer:
The advantages of Distributed Database Systems are:

- **Modular Development:** Modular development of a distributed database implies that a system can be expanded to new locations or units by adding new servers and data to the existing setup and connecting them to the distributed system without interruption. This type of expansion causes no interruptions in the functioning of distributed databases.
- **Reliability:** Distributed databases offer greater reliability in contrast to centralized databases. In case of a database failure in a centralized database, the system comes to a complete stop. In a distributed database, the system functions even when failures occur, only delivering reduced performance until the issue is resolved.
- **Lower Communication Cost:** Locally storing data reduces communication costs for data manipulation in distributed databases. Local data storage is not possible in centralized databases.
- **Better Response:** Efficient data distribution in a distributed database system provides a faster response when user requests are met locally. In centralized databases, user requests pass through the central machine, which processes all requests. The result is an increase in response time, especially with a lot of queries.

The Disadvantages of Distributed Database Systems are:

- **Costly Software:** Ensuring data transparency and coordination across multiple sites often requires using expensive software in a distributed database system.
- **Large Overhead:** Many operations on multiple sites require numerous calculations and constant synchronization when database replication is used, causing a lot of processing overhead.
- **Data Integrity:** A possible issue when using database replication is data integrity, which is compromised by updating data at multiple sites.
- **Improper Data Distribution:** Responsiveness to user requests largely depends on proper data distribution. That means responsiveness can be reduced if data is not correctly distributed across multiple sites.

3. What is the difference between parallel database and distributed database?
[MODEL QUESTION]

Answer:
In a multi-database system, a transaction is initially created and saved in the replica database but finally committed in the master database later when the transaction is propagated there as part of the database synchronization process. The tentatively committed transaction can exist in the system for an unlimited period of time. In other words, the life cycle of the transaction is entirely different.

A **parallel database system** is one that seeks to improve performance through parallel implementation of various operations such as loading data, building indexes, and evaluating queries.
Features

- Machines are physically close to each other, e.g., same server room
- Machines connects with dedicated high-speed LANs and switches
- Communication cost is assumed to be small
- Can shared-memory, shared-disk, or shared-nothing architecture

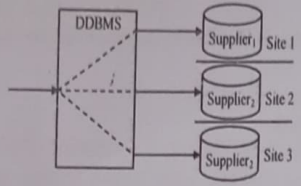
In a **distributed database system**, data is physically stored across several sites, and each site is typically managed by a DBMS that is capable of running independently of the other sites.
Features

- Machines can far from each other, e.g., in different continent
- Can be connected using public-purpose network, e.g., Internet
- Communication cost and problems cannot be ignored
- Usually shared-nothing architecture

4. What is distribution transparency? Explain different levels of distributed transparency.
[MODEL QUESTION]

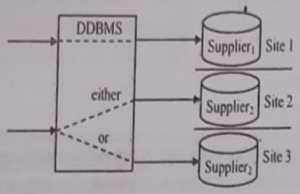
Answer:
Distribution transparency provides high-level abstractions which hide network location behind an impenetrable wall. In such a system, a process can't know where it is executing, even when it wants and needs to know.
Distribution Transparency for Read-Only Applications are:
Fragmentation transparency - reference to global relation as if the DB is not distributed
Location transparency - user accesses fragments ignoring where they are
Local Mapping transparency - refers to objects using names independent from local systems but has to specify sites of the objects, mapping of application into functions used by the local DBMS
Naming transparency - items in a database must have a unique name, but users don't need to worry about it.

```
read(terminal), $SNUM);
        Select NAME into $NAME
        from SUPPLIER
        where SNUM = $SNUM;
write(terminal, $NAME)
```



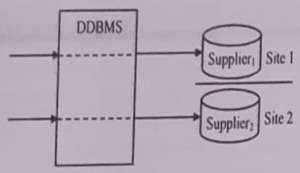
(a) Fragmentation transparency (level 1)

```
read(terminal, $SNUM);
        Select Name into $NAME
        from SUPPLIER1
        where SNUM = $SNUM;
if not #FOUND then
        Select NAME into $NAME
        from SUPPLIER2
        where SNUM = $SNUM;
write(terminal, $NAME)
```



(b) Location transparency (level 2)

```
read(terminal, $SNUM);
        Select NAME into $NAME
        from SUPPLIER1 AT SITE 1
        where SNUM = $SNUM;
if not #FOUND then
        Select NAME into $NAME
        from SUPPLIER2 AT SITE 3
        where SNUM = $SNUM;
write(terminal, $NAME)
```



(c) Local mapping transparency (level 3)

5. What are the fragmentation, replication and location transparencies? **[MODEL QUESTION]**

Answer:

Fragmentation transparency

Fragmentation transparency hides the fact from users that the data are fragmented. This is the highest level of distribution transparency. If a distributed system has fragmentation transparency, then the user must not be aware regarding the fragmentation of data. As a result, database accesses must be based on global schema and the user need not to specify the particular fragment names or data locations.

Location transparency

With location transparency, the user is aware how the data are fragmented but still does not have any idea regarding the location of the fragments. Location transparency is the middle level of distribution transparency. To retrieve data from a distributed database with location transparency, the end user or programmer has to specify the database fragment names but need not specify where these fragments are located in the distributed system.

Replication transparency

A copy of each fragment can be maintained at several sites. Data replication is the design process of deciding which fragments will be replicated. A DISTRIBUTED SYSTEM often employs DATA REPLICATION to ensure a fast response from databases and to enable the system to be resilient to hardware errors. Replication transparency is the term used to describe the fact that the user should be unaware that data is replicated.

6. "High reliability does not ensure correctness of the distributed system." Comment critically. **[MODEL QUESTION]**

Answer:

High Reliability: Distributed applications often require substantial effort to achieve levels of reliability equivalent to those expected from stand-alone applications. Detecting service failures in a stand-alone application is relatively straightforward.

When the failed site recovers or is repaired, mechanisms must be available to integrate it smoothly back into the system. Data transmission costs are also reduced here since all users at all sites are not accessing one centralized database. Distributed reliability protocols maintain the atomicity and durability properties by

(1) ensuring that either all of the actions of a transaction that is executing at different sites complete successfully (called commit) or none of them complete successfully (called abort), and

(2) ensuring that the modifications made to the database by committed transactions survive failures. The first requires atomic commit protocols (see Commit Protocols), while the second requires recovery protocols.

Correctness conditions are not satisfied by reliability factors each time.

7. What are the components which are necessary for building a distributed database? **[MODEL QUESTION]**

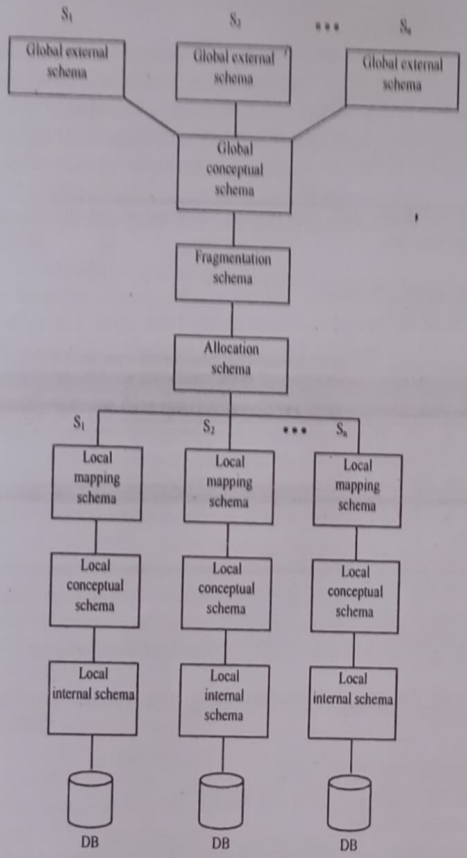
Answer:

The basic components are as follows:

- **Local data management system (DM):** This module is responsible for accessing data stored at the site.
- **The data dictionary:** The information regarding distribution of data (i.e. fragments) is stored in the data dictionary.
- **The transaction Module:** This module creates the global plan for execution of a query based on the information stored in the data dictionary. TM translates global query into queries on fragments stored at different sites. TM also ensures concurrency control, access planning, query optimization and distributed query execution.
- **The data communication module (DC):** This module is responsible for reliable inter-site communication using the links.

8. Describe the reference architecture of Distributed Database Management system with proper diagram. **[MODEL QUESTION]**

Answer:



Top-down design-The system begins with the analysis of requirements that defines the system environment. And the last step is physical design. This approach is used for Designing systems from scratch and for Homogeneous systems.

Bottom-up approach is used when the databases already exist at a number of sites or when the databases should be connected to solve common tasks. The process consists in the integration of local schemes in the global conceptual schema.

Long Answer Type Questions

1. What is Distributed Database System? Explain the features of DDBS. **[MODEL QUESTION]**

Answer:

1st Part:

A Distributed Database is a collection of multiple interconnected databases, which are spread physically across various locations that communicate via a computer network. Distributed databases utilize multiple nodes. They scale horizontally and develop a distributed system. More nodes in the system provide more computing power, offer greater availability, and resolve the single point of failure issue.

2nd Part:

Some general features of Distributed databases are:

- **Location independency** - Data is physically stored at multiple sites and managed by an independent DDBMS.
- **Distributed query processing** - Distributed databases answer queries in a distributed environment that manages data at multiple sites. High-level queries are transformed into a query execution plan for simpler management.
- **Distributed transaction management** – It provides a consistent distributed database through commit protocols, distributed concurrency control techniques, and distributed recovery methods in case of many transactions and failures.
- **Seamless integration** - Databases in a collection usually represent a single logical database, and they are interconnected.
- **Network linking** - All databases in a collection are linked by a network and communicate with each other.
- **Transaction processing** - Distributed databases incorporate transaction processing, which is a program including a collection of one or more database operations. Transaction processing is an atomic process that is either entirely executed or not at all.

2. a) What is transparency? Discuss network transparency and replication transparency with example. **[MODEL QUESTION]**

Answer:

a) 1st Part:

DBMS Transparency:

The term DBMS transparency refers to the ability of DDBMS to hide the implementation details from the user. Thus the fact that a distributed database is split into several fragments that can be stored on different computers and perhaps replicated is hidden from the user. The objective of the transparency is to make the distributed system appear like a centralized system.

We can identify four main types of transparency in a DDBMS:

- **Distribution transparency**
  a) Fragmentation transparency
  b) Location transparency
- **Transaction transparency**
  a) Performance transparency
  b) DBMS transparency

2nd Part:

Network transparency:

According to this property, a distributed database must be network transparent. Network transparency means that a user must be unaware about the operational details of the network. Network transparency is basically one of the properties of distributed database. Actually in distributed databases when a user wants to access data and if that particular data does not exist on user computer then it is the responsibility of DBMS to provide the data from any other computer where it exists. User does not know about this thing as from where data is coming. Network transparency also means roaming transparency and it gives the freedom of entering query from any workstation to a user. So a user does not know from where the results of query are coming as results may be coming from user workstation or any other workstation. Here the term roaming means that if a user moves from one place to other then all other users does not know about this.

Replication transparency:

A copy of each fragment can be maintained at several sites. Data replication is the design process of deciding which fragments will be replicated. A DISTRIBUTED SYSTEM often employs DATA REPLICATION to ensure a fast response from databases and to enable the system to be resilient to hardware errors. Replication transparency is the term used to describe the fact that the user should be unaware that data is replicated.

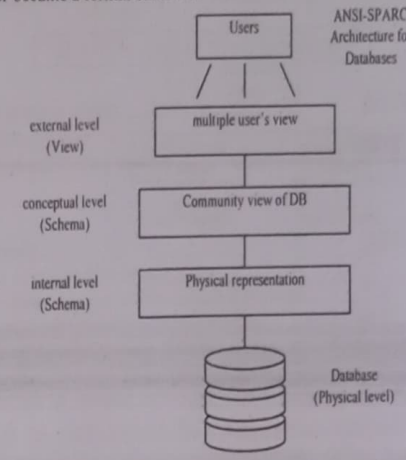
3. a) Draw the ANSI/SPARC reference architecture of Distributed Database System and discuss about the site independent schemas.
b) Show with the help of a diagram that replicated copy of R_2 of fragment R_1 is allocated into different sites as R_1^2 and R_2^1
c) When Bottom-up approach of distributed database design preferable over Top-down approach?
d) Explain the advantage of Remote access via an auxiliary program in case of heterogeneous distributed database system with the help of a diagram.
[MODEL QUESTION]

Answer:

a) 1st Part:

The ANSI-SPARC Architecture, where ANSI-SPARC stands for American National Standards Institute, Standards Planning And Requirements Committee, is an abstract design standard for a Database Management System (DBMS), first proposed in 1975.

Most modern commercial DBMS are based on this system. The ANSI-SPARC model however never became a formal standard.



A standard three level approach to database design has been agreed upon:

- **External level**
- **Conceptual level**
- **Internal level (includes physical data storage)**

The Three Level Architecture has the aim of enabling users to access the same data but with a personalised view of it. The distancing of the internal level from the external level means that users do not need to know how the data is physically stored in the database. This level separation also allows the Database Administrator (DBA) to change the database storage structures without affecting the users' views.

- **External Level (User Views):** A user's view of the database describes a part of the database that is relevant to a particular user. It excludes irrelevant data as well as data which the user is not authorized to access.
- **Conceptual Level:** The conceptual level is a way of describing what data is stored within the whole database and how the data is inter-related. The conceptual level does not specify how the data is physically stored.
  Some important facts about this level are:
  1. DBA works at this level.
  2. Describes the structure of all users.
  3. Only DBA can define this level.
  4. Global view of database.
  5. Independent of hardware and software.
- **Internal Level:** The internal level involves how the database is physically represented on the computer system. It describes how the data is actually stored in the database and on the computer hardware.

2nd Part:

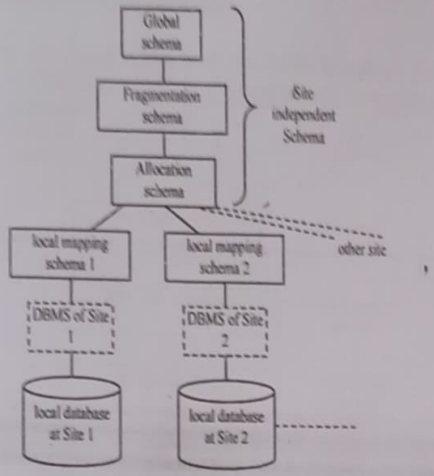
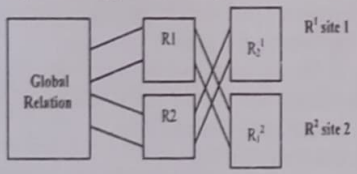


Figure shows a reference architecture for a distributed database. At due top level there in global schema. The global schema defines all the data which are contained in the distributed database as if the database were not distributed at all. For this reason, the global schema can be defined exactly in the same way as in a non distributed database. The global schema consists of the definition of a set of global relations.

Each global relation can be split into several non-overlapping portions which are called fragments. The mapping between global relations and fragments is defined in the fragmentation schema. This mapping is one to many; i.e. several fragments corresponds to one global relation, but only are global relation corresponds to one fragment.

Fragments are logical portions of global relations which are physically located at one or several sites of the network. The allocation schema defines at which site(s) a fragment is located.

b) The diagram that replicated copy of R2 of fragment R1 is allocated into different sites as R_1^2 and R_2^1 is:

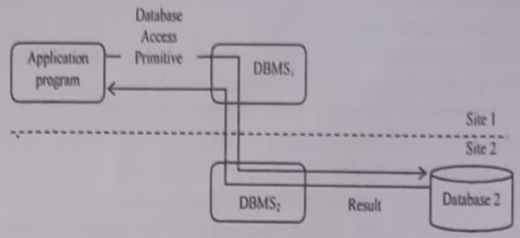


Physical images

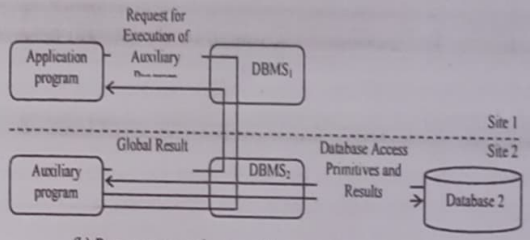
c) When the distributed database is developed as the aggregation of existing databases, it is not easy to follow the top-down approach. In fact, in this case the global schema is often produced as a compromise between existing data descriptions. It is even possible that each pair of existing databases is independently interfaced using a different translation schema, without the notion of a global schema; this, however, leads to systems which are different in conception from the reference architecture.

When existing databases are aggregated, a bottom-up approach to the design of data distribution can be used. This approach is based on the integration of existing schema into a single, global schema. It should be noticed that the bottom-up approach lends itself less easily to the development of horizontally fragmented relations.

d)



(a) Remote access via DBMS primitives



(b) Remote access via an auxiliary program

A heterogeneous DDBMS uses instead at least two different DBMSs. Heterogeneous DDBMSs add the problem of translating between the different data models of the different local DBMSs to the complexity of homogeneous DDBMSs. For this reason, if the development of a distributed database is performed top-down without a preexisting system, it is convenient to develop a homogeneous system. However, as it was stated in the previous section, in some cases the motivation for creating a distributed database is the necessity of integrating several preexisting databases; in this case it is required to develop a heterogeneous DDBMS, capable of building a global view of the database.

Although the problems of heterogeneous DDBMSs are very hard, and therefore really heterogeneous DDBMSs exist only as research prototypes, it must be stated that commercially available systems provide some degree of support for heterogeneity.

4. Write short notes on the following: **[MODEL QUESTION]**

a) Transparency

b) Heterogeneous databases

Answer:

a) DBMS Transparency:

The term DBMS transparency refers to the ability of DDBMS to hide the implementation details from the user. Thus the fact that a distributed database is split into several

fragments that can be stored on different computers and perhaps replicated is hidden from the user. The objective of the transparency is to make the distributed system appear like a centralized system.

We can identify four main types of transparency in a DDBMS:

- **Distribution transparency**
  - a) Fragmentation transparency
  - b) Location transparency
- **Transaction transparency**
  - c) Performance transparency
  - d) DBMS transparency

2nd Part:

Network transparency:

According to this property, a distributed database must be network transparent. Network transparency means that a user must be unaware about the operational details of the network. Network transparency is basically one of the properties of distributed database. Actually in distributed databases when a user wants to access data and if that particular data does not exist on user computer then it is the responsibility of DBMS to provide the data from any other computer where it exists. User does not know about this thing as from where data is coming. Network transparency also means roaming transparency and it gives the freedom of entering query from any workstation to a user. So a user does not know from where the results of query are coming as results may be coming from user workstation or any other workstation. Here the term roaming means that if a user moves from one place to other then all other users does not know about this.

Replication transparency:

A copy of each fragment can be maintained at several sites. Data replication is the design process of deciding which fragments will be replicated. A DISTRIBUTED SYSTEM often employs DATA REPLICATION to ensure a fast response from databases and to enable the system to be resilient to hardware errors. Replication transparency is the term used to describe the fact that the user should be unaware that data is replicated.

b) Heterogeneous databases:

A heterogeneous DDBMS uses instead at least two different DBMSs. Heterogeneous DDBMSs add the problem of translating between the different data models of the different local DBMSs to the complexity of homogeneous DDBMSs. For this reason, if the development of a distributed database is performed top-down without a preexisting system, it is convenient to develop a homogeneous system. However, as it was stated in the previous section, in some cases the motivation for creating a distributed database is the necessity of integrating several preexisting databases; in this case it is required to develop a heterogeneous DDBMS, capable of building a global view of the database.

Although the problems of heterogeneous DDBMSs are very hard, and therefore really heterogeneous DDBMSs exist only as research prototypes, it must be stated that commercially available systems provide some degree of support for heterogeneity.

DISTRIBUTED DATABASE DESIGN

Multiple Choice Type Questions

1. Allocation problem is easier [MODEL QUESTION]
   a) in redundant system
   b) in non-redundant system
   c) inreplicated (attributes) system
   d) both (a) and (c)

Answer: (a)

2. Clustering means [MODEL QUESTION]
   a) keeping a common data in one place
   b) keeping different types of data in different places
   c) keeping different types of data in one place
   d) keeping in a common data in different places

Answer: (a)

3. The horizontal fragmentation of a relation cannot be based on a property of its own attributes, but is derived from the horizontal fragmentation of another relation is called ..... Fragmentation. [MODEL QUESTION]
   a) horizontal
   b) vertical
   c) mixed
   d) derived horizontal fragmentation

Answer: (d)

4. Which of the following conditions cannot satisfy vertical fragmentation? [MODEL QUESTION]
   a) Disjointness b) Completeness c) Reconstruction d) Rebuild

Answer: (d)

5. Location transparency allows for which of the following? [MODEL QUESTION]
   a) users to treat data as if it is at one location
   b) programmers to treat the data as if it is at one location
   c) managers to treat the data as if it is at one location
   d) all of these

Answer: (d)

6. Storing a separate copy of the database at multiple locations is which of the following? [MODEL QUESTION]
   a) data replication
   b) horizontal partitioning
   c) vertical partitioning
   d) horizontal and vertical partitioning

Answer: (a)

7. Replication should be used when which of the following exist? [MODEL QUESTION]
   a) when transmission speeds and capacity in a network prohibit frequent refreshing of large tables

b) when using many nodes with different operating systems and DBMS and database designs
c) the application's data can be somewhat out-of date
d) all of these

Answer: (c)

3. Replication of attributes violates which of the following conditions of fragmentation? [MODEL QUESTION]
a) Completeness
b) Reconstruction
c) Disjointness
d) Both (b) and (c)

Answer: (a)

9. Local Mapping schema depends on the [MODEL QUESTION]
a) global relations
b) fragments
c) type of the local DBMS
d) location

Answer: (c)

10. Location transparency can be compared with [MODEL QUESTION]
a) Fragmentation transparency
b) Local mapping transparency
c) Replication transparency
d) none of these

Answer: (d)

11. Split approach of vertical fragmentation use [MODEL QUESTION]
a) progressing splitting
b) concurrent splitting
c) both (a) & (b)
d) none of these

Answer: (c)

12. To make non distributed system distributed we add [MODEL QUESTION]
a) DD component
b) DC component
c) DDC component
d) (a) & (c)

Answer: (d)

13. The disjointness condition is not strictly followed in [MODEL QUESTION]
a) horizontal fragmentation
b) vertical fragmentation
c) mixed fragmentation
d) hybrid fragmentation

Answer: (b)

14. Location independence represents that [MODEL QUESTION]
a) users are aware of the location of the data
b) users are unaware of the location of the data
c) users are aware of the physical names of database objects
d) None of the above

Answer: (b)

15. Processing locality conflicts with [MODEL QUESTION]
a) availability and reliability of distributed data
b) workload cost and availability
c) storage cost and availability
d) none of these

Answer: (b)

16. In DDBMSs, distributed logical data independence is provided by [MODEL QUESTION]
a) Local conceptual schema
b) Global external schema
c) Global conceptual schema
d) Local mapping schema

Answer: (b)

Short Answer Type Questions

1. Discuss horizontal, vertical, mixed, and derived fragmentations with examples. [MODEL QUESTION]

Answer:

Horizontal Fragmentation: Horizontal fragmentation consists of partitioning the tuples of a global relation into subsets; this is clearly useful in distributed databases, where each subset can contain data that have common geographical properties. It can be defined by expressing each fragment as a selection operation on the global relation.

Example: let a global relation be

SUPPLIER (SNUM, NAME, CITY)

Then the horizontal fragmentation can be defined in the following way:

$SUPPLIER_1 = \mathbf{SL}_{(CITY = \text{"Mysore"})}\text{-SUPPLIER}$

$SUPPLIER_2 = \mathbf{SL}_{(CITY = \text{"Shimoga"})}\text{-SUPPLIER}$

Vertical Fragmentation:

The *Vertical fragmentation* of a global relation is the subdivision of its attributes into groups; fragments are obtained by **projecting** the global relation over each group. This can be useful in distributed databases where each group of attributes can contain data that have common geographical properties. The fragmentation is correct; if each attribute is mapped into at least one attribute of the fragments; moreover, it must be possible to reconstruct the original relation by joining the fragments together.

Example: Consider a global relation

EMP (EMPNUM, NAME, SAL, TAX, MGRNUM, DEPTNUM)

A vertical fragmentation of this relation can be defines as

$EMP_1 = \mathbf{PJ}_{EMPNUM, NAME, MGRNUM, DEPTNUM}\ EMP$

$EMP_2 = \mathbf{PJ}_{EMPNUM, SAL, TAX}\ EMP$

Mixed Fragmentation: The fragments that are obtained by the above fragmentation operations are relations themselves, so that it is possible to apply the fragmentation operations recursively, provided that the correctness conditions are satisfied each time. The reconstruction can be obtained by applying the reconstruction rules in reverse order.

Example: Consider the same global relation

EMP (EMPNUM, NAME, SAL, TAX, MGRNUM, DEPTNUM)

The following is a mixed fragmentation, which is obtained by applying the vertical fragmentation of the previous example, followed by a horizontal fragmentation on *DEPTNUM*:

$$EMP_1 = SL_{DEPTNUM \le 10} PJ_{EMPNUM, NAME, MGRNUM, DEPTNUM} EMP$$
$$EMP_2 = SL_{10 < DEPTNUM \le 20} PJ_{EMPNUM, NAME, MGRNUM, DEPTNUM} EMP$$
$$EMP_3 = SL_{DEPTNUM > 20} PJ_{EMPNUM, NAME, MGRNUM, DEPTNUM} EMP$$
$$EMP_4 = PJ_{EMPNUM, NAME, SAL, TAX} EMP$$

Derived fragmentation: Since the process of fragmenting a single relation horizontally is a considerable effort, the question is whether such a fragmentation cannot be exploited further. In fact, there exists a good reason to do so when considering of how typically relational database schemas are constructed. In general, one finds many foreign key relationships, where one relation refers to another relation by using it's primary key as reference. Since these relationships carry a specific meaning it is very likely, that this foreign key relationship will also be used during accesses to the database, i.e. by executing join operations over the two relations. This means that the corresponding tuples in the two relations will be jointly accessed. Thus it is of advantage to keep them at the same site in order to reduce communication cost. It is of advantage to "propagate" a horizontal fragmentation that has been obtained for one relation to other relations that are related via a foreign key relationship and to keep later the corresponding fragments at the same site. We call fragments obtained in this way **derived horizontal fragments**.

2. What is the data fragmentation? **[MODEL QUESTION]**

Answer:

The database may be broken up into logical units called fragments which will be stored at different sites. The simplest logical units are the tables themselves.

Data fragmentation occurs when a piece of data in memory is broken up into many pieces. The de-fragmentation tool is to rearrange blocks on disk so that the blocks of each file are contiguous. There are mainly two types of fragmentations. They are External Fragmentation and Internal Fragmentation.

In the external fragmentation, the free space, that is available for storage is divided into many small pieces. The storage space is of many different sizes. In dynamic memory allocation, a block might be requested, but the contiguous block has a free space. There are ten blocks of 300 bytes of free space, separated by allocated regions; one still cannot allocate the requested block of 1000 bytes.

In Internal fragmentation, the space is wasted. it is often accepted in return for increased efficiency or simplicity. For example, in many file systems, files always start at the

beginning of a sector, which simplifies organization and makes it easier to grow files. Any space left over between the last byte of the file and the first byte of the next sector is internal fragmentation.

3. What is data replication? Explain with example. **[MODEL QUESTION]**

Answer:

Data replication is the design process of deciding which fragments will be replicated. A DISTRIBUTED SYSTEM often employs DATA REPLICATION to ensure a fast response from databases and to enable the system to be resilient to hardware errors. Replication transparency is the term used to describe the fact that the user should be unaware that data is replicated.

A copy of each fragment can be maintained at several sites. Data replication is the design process of deciding which fragments will be replicated. Replication, or the copying of data in databases to multiple locations to support distributed applications, is an important new tool for businesses in building competitive service advantages. New replicator facilities from several vendors are making this technology much more useful and practical than it's been in the past. In this article we will go into enough detail on replication for the reader to understand the importance of replication, its benefits and some of the related technical issues. While replication or data copying can clearly provide users with local and therefore much quicker access to data, the challenge is to provide these copies to users so that the overall systems operate with the same integrity and management capacity that is available with a monolithic, central model.

For example, if the same inventory records exist on two different systems in two different locations, say New York and Chicago, the system needs to insure that the same product isn't sold to two separate customers.

4. a) What are the advantage and disadvantage of replication?
b) What is auxiliary program? **[MODEL QUESTION]**

Answer:

a) Replication is the server process of copying data modifications from one location to another. The main characteristics of replication which make it appealing are:

The unit of replication is a transaction, not just an individual sequel (SQL) statement.

Replicated transactions are applied in the same order as they occurred on the primary side.

The replication system is able to detect whether a network connection is temporarily offline and when the component is available again. When a disruption like this occurs, the replication process should continue functioning without having to be adjusted.

Although the replication server is powerful and flexible, it is not easy to administrate. Another contributing factor is that the scope and complexity of the database administrator's job expands significantly when replication is involved. The previously independent data servers are now closely related because data is replicated between them. This effectively means that the data servers have become one large system. Thus, making a database administrator's job much more difficult.

b) One of the ways in which a remote database can be accessed by an application is via an auxiliary program, This program is written by an application programmer which returns the result to the requesting application.

5. What are correctness rules of fragmentation? Explain each rule.

Answer: [MODEL QUESTION]

Correctness Rules of Fragmentation are as follows:

a) **Completeness-** Decomposition of relation R into fragments R1, R2, . . . , Rn is complete iff each data item in R can also be found in some Ri.

b) **Reconstruction-** If relation R is decomposed into fragments R1, R2, . . . , Rn, then there should exist some relational operator ∇ that reconstructs R from its fragments, i.e., R=R1∇. . ∇Rn

- Union to combine horizontal fragments
- Join to combine vertical fragments.

c) **Disjointness-** If relation R is decomposed into fragments R1, R2, . . . , Rn and data item di appears in fragment Rj , then di should not appear in any other fragment Rk,, k ◇ j (exception: primary key attribute for vertical fragmentation)

- For horizontal fragmentation, data item is a tuple
- For vertical fragmentation, data item is an attribute

6. What is the difference among data fragmentation, data allocation and data replication?

Answer: [MODEL QUESTION]

Data fragmentation allows one to break a single object into two or more segments or fragments. The object might be a user's database, a system database, or a table. Each fragment can be stored at any site over a computer network. Information about data fragmentation is stored in the distributed data catalog (DDC), from which it is accessed by the TP to process user requests.

There are three types of data fragmentation strategies:

- Horizontal fragmentation
- Vertical fragmentation.
- Mixed fragmentation

Data allocation describes the process of deciding where to locate data. Data allocation strategies are as follows:

- With centralized data allocation, the entire database is stored at one site.
- With partitioned data allocation, the database is divided into two or more disjointed parts (fragments) and stored at two or more sites.
- With replicated data allocation, copies of one or more database fragments are stored at several sites.

Data replication refers to the storage of data copies at multiple sites served by a computer network. Fragment copies can be stored at several sites to serve specific information requirements.

7. a) What are the factors affecting allocation? [MODEL QUESTION]
b) What is minterm predicate?

Answer:

a) The factors affecting allocation are:

- response time constraints
- availability constraints
- network topology
- security restrictions

b) Minterm predicate y for a set of P of simple predicates refers to conjunction of all predicates appearing in P either taken in natural form or negated provided that the expression is not a contradiction. Simple and negated simple predicates.

Examples:

PNAME = "Maintenance" AND BUDGET ≤ 200000
NOT(PNAME = "Maintenance") AND BUDGET ≤ 200000

8. a) What is vertical clustering? [MODEL QUESTION]
b) What is partial union?

Answer:

a) In Vertical clustering an attribute may be contained in more than one fragments. The vertical clustering causes replication of some data elements. The replication is more advantageous for the read-only applications than for the read-write application. In the latter case, the same update operation is performed on several sites.

b) Let R and S be two union-compatible relations. Then their union $R \cup S$ is a relation which contains tuples from both relations: $R \cup S = \{x : x \in R \text{ or } x \in S\}$.

Union is a partial operation on relations: it is only defined for some (compatible) relations, not for all of them.

9. What is Mixed and Horizontal Fragmentation? Explain with an example.

Answer: [MODEL QUESTION]

Mixed Fragmentation: The fragments that are obtained by the above fragmentation operations are relations themselves, so that it is possible to apply the fragmentation operations recursively, provided that the correctness conditions are satisfied each time. The reconstruction can be obtained by applying the reconstruction rules in reverse order.

Example: Consider the same global relation

EMP (EMPNUM, NAME, SAL, TAX, MGRNUM, DEPTNUM)

The following is a mixed fragmentation, which is obtained by applying the vertical fragmentation of the previous example, followed by a horizontal fragmentation on *DEPTNUM*:

$$EMP_1 = SL_{DEPTNUM \leq 10}\, PJ_{EMPNUM,\ NAME,\ MGRNUM,\ DEPTNUM}\, EMP$$
$$EMP_2 = SL_{10 < DEPTNUM \leq 20}\, PJ_{EMPNUM,\ NAME,\ MGRNUM,\ DEPTNUM}\, EMP$$
$$EMP_3 = SL_{DEPTNUM > 20}\, PJ_{EMPNUM,\ NAME,\ MGRNUM,\ DEPTNUM}\, EMP$$
$$EMP_4 = PJ_{EMPNUM,\ NAME,\ SAL,\ TAX}\, EMP$$

Horizontal Fragmentation: Horizontal fragmentation consists of partitioning the tuples of a global relation into subsets; this is clearly useful in distributed databases, where each subset can contain data that have common geographical properties. It can be defined by expressing each fragment as a selection operation on the global relation.

Example: let a global relation be

SUPPLIER (SNUM, NAME, CITY)

Then the horizontal fragmentation can be defined in the following way:

$$SUPPLIER_1 = SL_{CITY = "Mysore"}\text{-SUPPLIER}$$
$$SUPPLIER_2 = SL_{CITY = "Shimoga"}\text{-SUPPLIER}$$

10. Explain different types of fragmentation indicating their advantage and disadvantage. **[MODEL QUESTION]**

Answer:

Fragmentation can be of three types: horizontal, vertical, and hybrid (combination of horizontal and vertical).

In **vertical fragmentation**, the fields or columns of a table are grouped into fragments.

In order to maintain re-constructiveness, 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 groups the tuples of a table in accordance to values of one or more fields.

Horizontal fragmentation should also confirm to the rule of re-constructiveness.

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";
```

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.

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 high.
- In case of recursive fragmentation, 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.

Long Answer Type Questions

1. a) Consider the schema SUPPLIER (SNO, NAME, CITY) and SUPPLY (SNO, PNO, DNO, QUAN) and the following transaction:

```
Read (tty, $PNO)
Select Name into $Name
From SUPPLIER, SUPPLY
Where SUPPLIER.SNO = SUPPLY.SNO
AND SUPPLY.PNO = $PNo
Write (tty, $Name)
```

What is the level of transparency of the above transaction-and why?

b) Discuss best-fit approach for a non-replicated allocation of horizontal fragmentation.

c) Does any directory file system provide the network transparency? If yes, explain how the transparency is achieved.

d) What is the most complex effect of update operation in distributed database system? Explain with the help of update subtree. **[MODEL QUESTION]**

Answer:

a) The level of transparency of the above transaction is Fragmentation transparency. Because, the transaction here hides the fact from users that the data are fragmented. This is the highest level of distribution transparency. If a distributed system has fragmentation transparency, then the user must not be aware regarding the fragmentation of data. As a result, database accesses must be based on global schema and the user need not to specify the particular fragment names or data locations.

b) Determining a non redundant final allocation is easier. The simplest method is a "best-fit" approach, a measure is associated with each possible allocation, and the site with the best measure is selected. Replication introduces further complexity in the design, because:

1. The degree of replication of each fragment becomes a variable of the problem.
2. Modeling read applications is complicated by the fact that the applications can select among several alternative sites for accessing fragments. For determining the redundant allocation of fragments, either of the following two methods can be used:
   i) Determine the set of all sites where the benefit of allocating one copy of the fragment is higher than the cost.
   ii) For non replicated problem, introduce replicated copies starting from the most beneficial.

c) Directory file system provide the network transparency, ***Network transparency*** allows applications to access remote resources through network without needing to know whether they are remote or local. ***In a distributed directory file system***, network transparency involves three aspects: naming, performance, and failure resiliency.
Naming plays an important role in distributed file systems. A directory file system is said to provide name transparency if it satisfies the following three requirements. ***First***, any file is accessible from any location. ***Second***, the same name is used at every location. And ***third***, file location is not reflected in the name.

d) In providing distribution transparency to update applications, there is another problem which is more complex than performing the updates on all copies of a data item. Consider for example what happens if the *CITY* attribute of a supplier is modified. Clearly, the supplier tuple must be moved from one fragment to another. Moreover, in our example database, the tuples of the *SUPPLY* relation which refer to the same supplier also must change fragment, because the *SUPPLY* relation has a derived fragmentation. It is intuitively clear that changing the value of an attribute which is used in the definition of the fragmentation schema can have rather complex effects. The degree to which these effects are managed by the DDBMS characterizes the level of distribution transparency for updates.

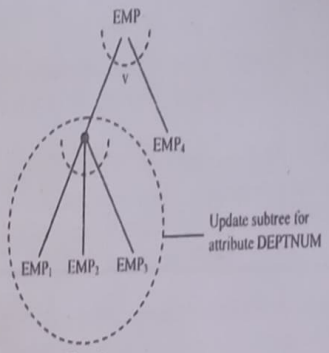
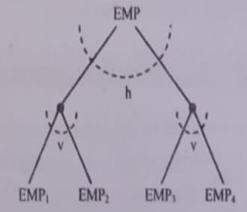


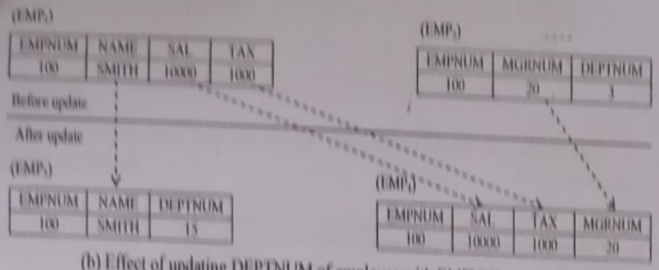
Fig.: 1

In order to understand which data movement operations are required by an update of a fragmentation attribute, it is useful to look at the fragmentation tree. Consider an attribute *A* which is used in a selection predicate of a horizontal fragmentation. The **update subtree** of *A* is the subtree having as root the node representing the above horizontal fragmentation. For example, we indicate in the above figure 1 the update subtree for attribute *DEPTNUM*. The effects of a change of an attribute are limited to the fragments which are leaves of its update subtree. For example, a change of attribute *DEPTNUM* affects only EMP_1, EMP_2, and EMF_3, but not EMP_1. A tuple can migrate (change fragments) between two of the three former fragments as a consequence of the update.
In general, the update subtree of an attribute *A* can be more complex than the simple one shown in Figure 1, and the effect of changing a value of the attribute *A* is therefore more complex. For example, suppose that relation EMP has the fragmentation tree shown in Figure 2.a. In this case the update subtree of attribute *DEPTNUM* is the same as the whole fragmentation tree. The effect of changing the *DEPTNUM* value of a tuple having *EMPNUM* = 100 from 3 to 15 is shown in Figure 2.b. This update is such that the tuple, originally stored in fragments belonging to the left subtree of Figure 2.a, becomes then part of the right subtree. We see that not only the data has been moved between fragments, but the tuple has also been composed in a different way.



$EMP_1 = PJ_{EMPNUM,NAME,SAL,TAX}SL_{DEPTNUM < 10}(EMP)$
$EMP_2 = PJ_{EMPNUM,MGRNUM,DEPTNUM}SL_{DEPTNUM < 10}(EMP)$
$EMP_3 = PJ_{EMPNUM,NAME,DEPTNUM}SL_{DEPTNUM < 10}(EMP)$
$EMP_4 = PJ_{EMPNUM,SAL,TAX,MGRNUM}SL_{DEPTNUM < 10}(EMP)$

(a) A different fragmentation and fragmentation tree for relation EMP



(b) Effect of updating DEPTNUM of employee with EMPNUM = 100

Fig. 2

2. Discuss best fit, all beneficial site and additional beneficial site strategy for allocation of fragments. **[MODEL QUESTION]**

Answer:

Best-fit approach:

The sites will be evaluated how many accesses are required from the applications located on this site:

$B^1_{ij} = \sum_k f_{kj} n_{ki}$

where

- f_{kj} the frequency of application k at site j
- n_{ki} the number of accesses from application k to fragment i

The site with greatest B_{ij} value is selected for the fragment i.

All beneficial site approach:

The goodness of a site is measured by the benefit of the local read accesses and the costs of remote update accesses:

$B^2_{ij} = \sum_k f_{kj} r_{ki} - C \times \sum_k \sum_{l \neq j} f_{kl} u_{ki}$

where

- f_{kj} the frequency of application k at site j
- r_{ki} the number of read accesses from application k to fragment i
- u_{ki} the number of update accesses from application k to fragment i
- C update cost relative to the read

Additional replica ion approach:

Beside the costs involved in B^2_{ij} a so called reliability factor is contained in the measure too. This factor increases with the increasing number of replications.

3. a) Consider the two relation schemas PAY = {TITLE, SAL} and EMP = {ENO, ENAME, TITLE} and where owner (L) = PAY and member (L) = EMP and L is the link between the two relation schemas. Define a derived horizontal fragmentation on EMP with the predicates SAL < = 200000 and SAL > 20000.

b) Write the BEA algorithm for vertical fragmentation.

c) Discuss "all beneficial sites" and "additional replication" methods for replicated allocation of horizontal fragments. **[MODEL QUESTION]**

Answer:

a) We can group engineers into 2 groups according to their salary:

≤ 200.000 and > 200000. The 2 fragments

EMP_1 and EMP_2 are defined:

$EMP_1 = EMP \propto PAY_1$

$EMP_2 = EMP \propto PAY_2$

where

$PAY_1 = \sigma_{SAL \leq 200000}(PAY)$

$PAY_2 = \sigma_{SAL > 200000}(PAY)$

b) Bond Energy Algorithm (BEA) has been used for clustering of entities. BEA finds an ordering of entities (in our case attributes) such that the global affinity measure (AM) is maximized.

$AM = \sum_i \sum_j$ (affinity of A_i and A_j with their neighbors)

Input: The AA matrix

Output: The clustered affinity matrix CA which is a perturbation of AA

1. **Initialization:** Place and fix one of the columns of AA in CA.
2. **Iteration:** Place the remaining n-i columns in the remaining i+1 positions in the CA matrix. For each column, choose the placement that makes the most contribution to the global affinity measure.
3. **Row order:** Order the rows according to the column ordering.

c) All beneficial sites – In this approach the set of all sites where the benefit of allocation of one copy of the fragment is higher than the cost, and allocate a copy of the fragment to each element of this set.

Additional replication – Determine the solution of the non-replicated problem. Progressively introduce replicated copies starting from the most beneficial; the process is terminated when no additional replication is beneficial.

4. Write short notes on the following: **[MODEL QUESTION]**

a) Vertical and derived fragmentation

b) Fragmentation

Answer:

a) Vertical and derived fragmentation:

Horizontal Fragmentation: Horizontal fragmentation consists of partitioning the tuples of a global relation into subsets; this is clearly useful in distributed databases, where each subset can contain data that have common geographical properties. It can be defined by expressing each fragment as a selection operation on the global relation.

Example: let a global relation be

SUPPLIER (SNUM, NAME, CITY)

Then the horizontal fragmentation can be defined in the following way:

$$SUPPLIER_1 = SL_{CITY = "Mysore"} \text{ SUPPLIER}$$
$$SUPPLIER_2 = SL_{CITY = "Shimoga"} \text{ SUPPLIER}$$

Vertical Fragmentation:

The *Vertical fragmentation* of a global relation is the subdivision of its attributes into groups; fragments are obtained by **projecting** the global relation over each group. This can be useful in distributed databases where each group of attributes can contain data that have common geographical properties. The fragmentation is correct; if each attribute is mapped into at least one attribute of the fragments; moreover, it must be possible to reconstruct the original relation by joining the fragments together.

Example: Consider a global relation

EMP (EMPNUM, NAME, SAL, TAX, MGRNUM, DEPTNUM)

A vertical fragmentation of this relation can be defines as

$$EMP_1 = PJ_{EMPNUM,NAME, MGRNUM, DEPTNUM} \text{ EMP}$$
$$EMP_2 = PJ_{EMPNUM, SAL,TAX} \text{ EMP}$$

b) Fragmentation:

Fragmentation is a process of dividing the whole or full database into various sub-tables or sub relations so that data can be stored in different systems. The small pieces of sub relations are called *fragments*. These fragments are called logical data units and are stored at various sites. It must be made sure that the fragments are such that they can be used to reconstruct the original relation.

In the fragmentation process, let's say, If a table T is fragmented and is divided into a number of fragments say T1, T2, T3....TN. The fragments contain sufficient information to allow the restoration of the original table T. This restoration can be done by the use of UNION or JOIN operation on various fragments. This process is called ***data fragmentation***. All of these fragments are independent which means these fragments can not be derived from others. The users needn't be logically concerned about fragmentation which means they should not concerned that the data is fragmented and this is called *fragmentation Independence* or we can say *fragmentation transparency*.

Advantages:

- As the data is stored close to the usage site, the efficiency of the database system will increase.
- Local query optimization methods are sufficient for some queries as the data is available locally.
- In order to maintain the security and privacy of the database system, fragmentation is advantageous.

Disadvantages:

- Access speeds may be very high if data from different fragments are needed.
- If we are using recursive fragmentation, then it will be very expensive.

DISTRIBUTED QUERY OPTIMIZATION

Multiple Choice Type Questions

1. Replication of attributes violets which of the following conditions of fragmentation? [MODEL QUESTION]
   a) Completeness b) Reconstruction
   c) Disjointness d) Both (b) and (c)

Answer: (a)

2. Idempotence of unary operation implies (U = >unary operation, B = >binary operation) [MODEL QUESTION]
   a) U1 U2 R ↔ U2 U1 R b) U (R B S) ↔ U (R) B U (S)
   c) U R ↔ U1 U2 R d) U (R) B U (S) ↔ U (R B S)

Answer: (c)

3. Two-phase commitment protocol is used for [MODEL QUESTION]
   a) concurrency control b) integrity control
   c) recovery d) redundancy

Answer: (a)

4. Time stamp mechanism is used for [MODEL QUESTION]
   a) concurrency control b) integrity control
   c) recovery c) redundancy

Answer: (a)

5. In a relational data model, the columns are called [MODEL QUESTION]
   a) relation b) tuples c) degree d) attributes

Answer: (b)

6. Primary key never takes [MODEL QUESTION]
   a) date b) NULL c) constant value d) not NULL

Answer: (b)

7. In how many ways m rows and n attributes can be represented? [MODEL QUESTION]
   a) $m*n$ b) $n!*m$ c) $n*m!$ d) $m!*n!$

Answer: (a)

8. Spurious tuples may occur due to [MODEL QUESTION]
   i) Bad normalization
   ii) Theta joins
   iii) Updating tables from join
   a) (i) and (ii) b) (ii) and (iii) c) (i) and (iii) d) (i), (ii) and (iii)

Answer: (a)

9. A, B, C is a set of attributes. The functional dependency is as follows: [MODEL QUESTION]

$AB \to B, AC \to C, C \to B$

a) is in 1NF b) is in 2NF c) is in 3NF d) is in BCNF

Answer: (a)

10. What is the maximum number of functional dependencies (trivial and non-trivial) of a relation R of degree n? [MODEL QUESTION]

a) 2n b) 2^{2n} c) n! d) 2^n

Answer: None of these. It is $(2^n-1)^2$

11. Is determination of consistent view of the network has the same complexity as determination of network failure? [MODEL QUESTION]

a) Yes b) No c) Unknown d) Possible

Answer: (d)

12. Semi-join is required to [MODEL QUESTION]

a) reduced network traffic b) reduced memory usage

c) increased speed d) none of these

Answer: (a)

13. Which of the following is not a phase of optimistic method for distributed concurrency control? [MODEL QUESTION]

a) read phase b) verification phase

c) validation phase d) write phase

Answer: (b)

14. A serializable schedule always [MODEL QUESTION]

a) produces same result as all possible serial schedules of participating transactions

b) produces same result as a serial schedule of participating transactions

c) produces same result as a concurrent schedule of participating transactions

d) is same as a serial schedule of participating transactions

Answer: (b)

15. A semi join is which of the following? [MODEL QUESTION]

a) only the joining attributes are sent from one site to another and then all of the rows returned

b) all of the attributes are sent from one site to another and then only the required rows are returned

c) only the joining attributes are sent from one site to another and then only the required rows are returned

d) all the attributes are sent from one site to another and then only the required rows are returned

Answer: (c)

16. $PJ_A[R:q_R]$ implies [MODEL QUESTION]

a) $[PJ_AR:q_R]$ b) $[PJ_AR:A\ AND\ q_R]$

c) $[PJ_AR:A\ OR\ q_R]$ d) $[PJ_A\ qr:R]$

Answer: (c)

17. Allocation schema is a [MODEL QUESTION]

a) site independence schema b) modern schema

c) site dependence schema d) none of these

Answer: (a)

18. DWFG stands for [MODEL QUESTION]

a) Distributed wait for graph b) Domain window Firewall Gateway

c) Distributed weighted Forward Graph d) none of these

Answer: (a)

19. Which of the following is a replica control protocol? [MODEL QUESTION]

a) Distributed 2PL b) Biased protocol

c) Majority locking d) All of these

Answer: (d)

20. Which of the following is a ROWA protocol? [MODEL QUESTION]

a) Primary copy 2PL b) Majority locking

c) Distributed 2PL d) Centralized 2PL

Answer: (c)

21. Which of the following operation is conflicting? [MODEL QUESTION]

a) Read-Write b) Write-Read c) Write-Write d) All of these

Answer: (b)

22. If a distributed system has *n* sites, the total number of message transfers in centralized 2PL is [MODEL QUESTION]

a) $2n$ b) $3n+2$ c) $2n+3$ d) $5n$

Answer: (c)

23. Which of the following refers to the operation of copying and maintaining database object in multiple databases belonging to a distributed system? [MODEL QUESTION]

a) Backup b) Recovery c) Replication d) None of these

Answer: (c)

24. Which of the following is the recovery management technique for distributed system? [MODEL QUESTION]

a) Deferred update b) Immediate update

c) Two-phase commit d) None of these

Answer: (c)

25. Local autonomy means [MODEL QUESTION]
a) local query will be processed locally
b) local data will be accesses by local site only
c) local DBA is sole authority of local data
d) each site is a DBMS in its own right
Answer: (a)

26. Database profile includes [MODEL QUESTION]
a) cardinality b) size c) distinct values d) all of these
Answer: (d)

Short Answer Type Questions

1. What are the objectives of Distributed Query Processing? [MODEL QUESTION]

Answer:

Objectives of Distributed Query Processing

1. The main objectives of query processing in a distributed environment is to form a high level query on a distributed database, which is seen as a single database by the users, into an efficient execution strategy expressed in a low level language in local databases.
2. An important point of query processing is query optimization. Because many execution strategies are correct transformations of the same high level query the one that optimizes (minimizes) resource consumption should be retained.
3. The good measure of resource consumption are:
   i) The total cost that will be incurred in processing the query. It is the dome of all times incurred in processing the operations of the query at various sites and intrinsic communication.
   ii) The resource time of the query. This is the time elapsed for executing the query. Since operations can be executed in parallel at different sited, the response time of a query may be significantly less than its cost.
4. In a distributed system, the total cost to be minimized includes CPU, I\O, and communication costs. This cost can be minimized by reducing the number of I\O operation through fast access methods to the data and efficient use of main memory. The communication cost is the time needed for exchanging the data between sited participating in the execution of the query.

2. What do you meant by Query Processing? Write down the characteristics of query Processors. [MODEL QUESTION]

Answer:

1st Part:

A Query Processor is a module in the Distributed Database System that performs the tasks to process, to optimize and to generate execution strategy for a high-level query.

The Query Processing does data localization for the query based on the fragmentation scheme and generates the execution strategy that incorporates the communication operations involved in processing the query.

2nd Part:

Characteristics of Query Processors:

Languages

- Input language can be relational algebra or calculus; output language is relational algebra (annotated with communication primitives). The query processor must efficiently map input language to output language.

Types of Optimization

- The output language specification represents the execution strategy. There can be many strategies, the best one can be selected through exhaustive search, or by applying heuristic (minimize size of intermediate relations). For distributors databases semi-joins can be applied to reduce data transfer.

Statistics

- Fragment cardinality and size
- Size and number of distinct values for each attribute. Detailed histograms of attribute values for better selectivity estimation.

Decision Sites

- One site or several sites participate in selection of strategy.

Exploitation of network topology

- Wide area network communication cost.
- Local area network parallel execution

Exploitation of replicated fragments

- Larger number of possible strategies.
- Use of Semi joins.
- Reduce size of data transfer.
- Increase # of massages and local processing

3. What is locking? What are the shared and exclusive locks? Briefly discuss Timestamp protocol in relation to distributed database system? [MODEL QUESTION]

Answer:

Any Database management system whether it is distributed or centralized, utilizes locks to provide concurrency control. Common uses of locks are to ensure that only one user can modify a record at a time and that data cannot be read while it is being modified. Locking mechanisms can be enforced at the row, table or page level.

There are two mechanisms for locking data in a database: pessimistic locking, and optimistic locking.

Exclusive locks

These locks protect updates to file resources, both recoverable and non-recoverable. They can be owned by only one transaction at a time. Any transaction that requires an exclusive lock must wait if another task currently owns an exclusive lock or a shared lock against the requested resource.

Shared locks

A shared lock reserves its object for reading only. They ensure that a record is not in the process of being updated during a read-only request. Shared locks can also be used to prevent updates of a record between the time that a record is read and the next synchronous point.

It prevents the object from changing while the lock remains. More than one program can place a shared lock on the same object.

Timestamp protocol

The principal idea behind the time stamping scheme is that each transaction is given a unique time stamp used to decide the serialization order.

Two primary methods are used to generate unique timestamps, one is centralized and one is distributed. In the centralized scheme, a single site is chosen for distributing the timestamps. The site can use a logical counter or its own local clock for this purpose.

In the distributed scheme, each site generates a unique local type stamp using either a logical counter or the local clock. The global unique timestamp is obtained by concatenation of the unique local type stamp with the site identifier, which must be unique. We use the site identifier in the least significant position to ensure that the global timestamps generated in one site are not always greater than those generated in another site.

We may still have a problem if one site generates local timestamps at a faster rate than does the other site. In such a case, the fast site's logical counter would be larger than that of other sites. Therefore, all timestamps generated by the fast site will be larger than those generated by other sites. A mechanism is needed to ensure that the local timestamps are generated fairly across the system. To accomplish the fair generation of timestamps, we define within each site S_i a logical clock (LC_i) which generates the unique local timestamp. To ensure that the various logical clocks are synchronized, we require that a site S_i advance its logical clock whenever a transaction T_i with timestamp <x,y> visits that site and x is greater than the current value of LC_i. In this case, site S_i advances its logical clock to the value x+y.

4. What are the difference between semi join and natural join? Explain with example. Show that semi join in not commutative. [MODEL QUESTION]

Answer:

The Natural Join of two relations R and S is defined as R ⋈ S is an equijoin in which all attributes with the same names in the two relations are compared.

The Semi-Join of two relations R and S is denoted as $R \ltimes S$ where F is a formula which specifies a join predicate, and is derived from the projection of all attributes of the first operand relation after the join operation of the two relations.

The major difference between Natural Join and Semi Join is that Semi Join adds an additional projection operation on all attributes of the first operand relation after natural join operation. The size of the resultant relation of Semi Join operation is less than that of Natural Join. Therefore, the performance of Semi Join operation is increased. ***Note that the semi-join is not commutative***, i.e.

$$R \ltimes_{A=B} S \neq S \ltimes_{A=B} R$$

For an example consider the tables *Employee* and *Dept*.

Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales

Dept

DeptName	Manager
Finance	George
Sales	Harriet
Production	Charles

The natural join of the two tables is given by

Employee ⋈ *Dept*

Name	EmpId	DeptName	Manager
Harry	3415	Finance	George
Sally	2241	Sales	Harriet
George	3401	Finance	George
Harriet	2202	Sales	Harriet

Consider the following Dept table

Dept

DeptName	Manager
Sales	Bob
Sales	Thomas
Production	Katie
Production	Mark

The semijoin of Employee and department is given by

Employee ⋉ *Dept*

Name	EmpId	DeptName
Sally	2241	Sales
Harriet	2202	Production

5. Consider the following PROJECT relation. This relation is fragmented in two fragments namely, P1 and P2 as given below: **[MODEL QUESTION]**

PNO	ENO	PTYPE	PDESC
P001	E001	DEVELOPMENT	Proj1
P002	E005	DEVELOPMENT	Proj2
P003	E0014	DEVELOPMENT	Proj3
P004	E002	MAINTENANCE	Proj4

P1: $\sigma_{PTYPE= "DEVELOPMENT"}$ (PROJECT), P2: $\sigma_{PTYPE= "MAINTENANCE"}$ (PROJECT)
Show the correctness of the fragmentation.

Answer:

P1: $\sigma_{PTYPE= "DEVELOPMENT"}$ (PROJECT)

PNO	ENO	PTYPE	PDESC
P001	E001	DEVELOPMENT	Proj1
P002	E005	DEVELOPMENT	Proj2
P003	E0014	DEVELOPMENT	Proj3

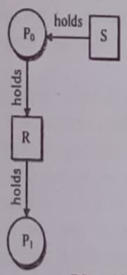
P2: $\sigma_{PTYPE= "MAINTENANCE"}$ (PROJECT)

PNO	ENO	PTYPE	PDESC
P004	E002	MAINTENANCE	Proj4

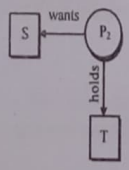
6. What is false deadlock? **[MODEL QUESTION]**

Answer:

Suppose machine A has a process P0, which holds the resource S and wants resource R, which is held by P1. The local graph on A is shown in Figure below.



Another machine, machine B, has a process P2, which is holding resource T and wants resource S. Its local graph is shown below.

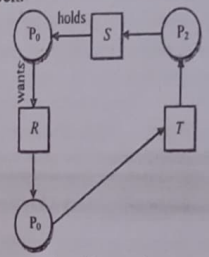


Both of these machines send their graphs to the central coordinator, which maintains the union. There are no cycles and hence no deadlock. Now two events occur. Process P1 releases resource R and asks machine B for resource T. Two messages are sent to the coordinator:

message 1 (from machine A): "releasing R"

message 2 (from machine B): "waiting for T"

This should cause no problems (no deadlock). However, if message 2 arrives first, the coordinator would then construct the graph in Figure below and detect a deadlock. Such a condition is known as false deadlock.

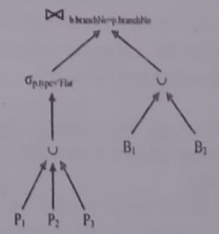


A way to fix this is to use Lamport's algorithm to impose global time ordering on all machines. Alternatively, if the coordinator suspects deadlock, it can send a reliable message to every machine asking whether it has any release messages. Each machine will then respond with either a release message or a negative acknowledgement to acknowledge receipt of the message.

7. Optimize the following query: **[MODEL QUESTION]**
List the flats that are for rent along with the corresponding branch details.
Relations: BRANCH (Branch No., Street, Postcode)
PROPFORRENT(PNO., Rent Amount, Owner No., Type, Branch No.)
Consider the following fragments:
P1: σBranch_no= "B003" ^ type = "House" (PROPFORRENT)
P2: σBranch_no= "B003" ^ type = "Flat" (PROPFORRENT)
P3: σBranch_no!= "B003" (PROPFORRENT)
B1: σBranch_no= "B003" (BRANCH)
B2: σBranch_no!= "B003" (BRANCH)
Write the SQL. Write the corresponding relational algebraic expression. Then optimize the query.

Answer:

SELECT * FROM BRANCH b, PROPERTY p
WHERE b, Branch No. = p.Branch No.
AND p.type = "Flat";



8. What is deadlock prevention? [MODEL QUESTION]

Answer:

Distributed Deadlock Prevention:

With a deadlock prevention approach, a transaction is aborted and restarted if there is a risk that deadlock might occur. Since deadlocks can never occur in this way, deadlock prevention eliminates the need for deadlock detection and resolution. Deadlock prevention is done in the following way: if a transaction T_1 requests a resource which is held by another transaction T_2, then a "prevention test" is applied; if the test indicates that there is a risk of deadlock, then T_1 is not allowed to enter a wait state. Instead, either T_1 is aborted and restarted (nonpreemptive method), or T_2 is aborted and restarted (preemptive method).

The prevention test must ensure that if T_1 is allowed to wait for T_2, then deadlock can never occur. This is typically obtained by ordering transactions, using for instance the lexicographical ordering of their identifiers, $T_1$is allowed to wait for T_j only if $i < j$. The reason why this method works is that it is impossible to build a closed chain

$$i_1 \rightarrow i_2 \rightarrow \ldots\ldots \rightarrow i_n \rightarrow i_1$$

such that if $i_l \rightarrow i_k$, then $i_l < i_k$ for all j, k.

In order to generate unique identifiers for transactions in a distributed system, it is sufficient that each site generate locally unique identifiers and append the site identifier to them. However, for the performance of the algorithm, it is better that identifiers reflect the "time" of transaction initiation. Therefore, timestamps are used for this purpose.

A nonpreemptive method for deadlock prevention based on timestamps is the following: If T_i requests a lock on a data item which is already locked by T_j then T_i is permitted to wait only if T_i is older than T_j. If T_i is younger than T_k then T_i is aborted and restarted with the same timestamp. The rationale of this method is the following: It is better always to restart the younger transaction.

A preemptive method works with the opposite rule: if T_i requests a lock on a data item which is already locked by T_j, then T_i is permitted to wait only if it is younger than T_j; otherwise, T_j is aborted and the lock is granted to T_i. The rationale of this method is the following: Since we still want to abort the younger transaction, with a preemptive method we must allow older transactions to preempt younger ones, and therefore only younger transactions wait for older ones.

9. Consider the following: [MODEL QUESTION]

Select Ename, Resp from Emp, Asg, Proj.
where Emp.Eno = Asg.Eno and Pname = "CAD/CAM"
and Dur > = 36 and draw its query graph.

Answer:

The query graph of the following query:

Select Ename, Resp from Emp, Asg, Proj.
where Emp.Eno = Asg.Eno **and** Pname = "CAD/CAM"
and Dur > = 36

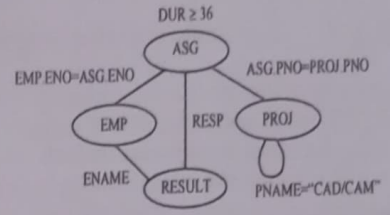


Fig: Query graph

10. Consider the following relation algebra: [MODEL QUESTION]

π_{cname} (σ_{mgrno} =374 (EMP dno ⋈ = dno DEPT) -
(σ_{mgrno} =374 AND sal >= 35000 (EMP ⋈ dno = dno DEPT)))

The above expression generate names of the employee whose salary are less than 35000 and manager number is 374. Draw operator tree of the above algebraic expression and optimize it into the highest level.

Answer:

The corresponding operator-tree is shown in fig. (a) we start by merging leaves corresponding to EMP and DEPT relation. Then we factorize the selection on 'sal' with respect to join. Now we can merge the nodes. Corresponding to the selection on 'mgrno' and finally the node corresponding to the join; we came to the operator tree of fig. (b). We recognize the following sub-expression:

$\sigma_{mgrno=374}$ ($EMP_{dno} \bowtie_{dno}$ DEPT)

We can use the property:

$R - (\sigma_F R) \leftrightarrow \sigma_{nF} R$

reducing the operator tree to that in fig. (c). Finally we can apply idempotence to the projection and push selections and projections toward the leaves of the tree generating the operator tree of fig. (d).

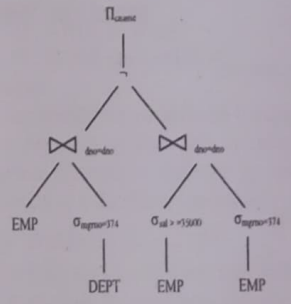


Fig. (a)

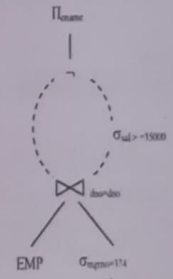


Fig. (b)

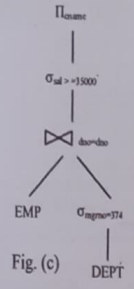


Fig. (c)

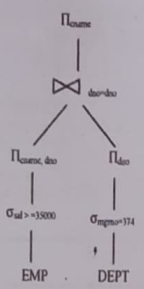


Fig. (d)

11. "2PL protocol only ensure that the schedule is conflict serializable or not; but it cannot generate all possible combination of valid serilizable schedule." Comment critically with example. **[MODEL QUESTION]**

Answer:

2PL ensures conflict serializability.

Every schedule produced by two-phase locking, i.e.,

- every read and write is preceded by a lock and followed by an unlock
- no DB item is simultaneously locked by more than one transaction
- for each transaction, the last lock precedes the first unlock is conflict-serializable. Transactions can be serialized according to their lock points.

But schedules may cause cascading aborts. i.e. if a transaction aborts after it releases a lock, it may cause other transactions that have accessed the unlocked data item to abort as well.

12. Prove that $[R:q_R]SJ_F[S:q_S] \geq [R\,SJ_F S:q_S \text{ AND } q_S \text{ AND } F]$. **[MODEL QUESTION]**

Answer:

Semi-join is derived from the use of projection and join.

So,

$$[R:q_R]SJ_F[S:q_S]$$

$$\Rightarrow \quad PJ_{Attr(R)}([R:q_R]JN_F[S:q_S])$$

$$\Rightarrow \quad PJ_{Attr(R)}([RJN_F S:q_R \text{ AND } q_S \text{ AND } F])$$

$$\Rightarrow \quad [PJ_{Attr(R)}(RJN_F S):q_R \text{ AND } q_S \text{ AND } F]$$

$$\Rightarrow \quad [RSJ_F S:q_R \text{ AND } q_S \text{ AND } F]$$

Thus, proved

This is rule 7 of the algebra of qualified relations. Since it can be shown that its equal to. So, as per the join rules and theory, values that are less than or below the limit will also be true.

13. Explain the significance of the semi-join program in context with DDBMS. **[MODEL QUESTION]**

Answer:

Given an equi-join $R\,JN_{A=B}\,S$ where A and B are attributes (or, more generally, sets of attributes) or R and S, the semi-join program for it is given by

$$S\,JN_{A=B}\,(R\,SJ_{A=B}PJ_B S)$$

Let us start by considering a simple example of a semi-join program and its transmission cost. A semi-join program between two relations R and S over two attributes A and B is defined as $(R\,SJ_{A=B}S)JN_{A=B}S$, and is equivalent to $R\,JN_{A=B}\,S$. Assume that R and S are allocated at sites r and s, respectively. Then, performing the semi-join program corresponds to:

1. Sending $PJ_B(S)$ to site r, at a cost:

$$C_0 + C_1 \times \text{size}(B) \times \text{val}(B[S])$$

2. Computing the semi-join on r, at a null cost; let

$$R' = R\,SJ_{A=B}S$$

3. Sending R' to s, at a cost:

$$C_0 + C_1 \times \text{size}(R) \times \text{card}(R')$$

4 Computing the join on s, at a null cost

The overall cost is

$$C_{SJ} = 2 \times C_0 + C_1 \times (\text{size}(B) \times \text{val}(B[S]) + \text{size}(R) \times \text{card}(R'))$$

Semi-join is not symmetric, and clearly the other semi-join program $(S\,SJ\,R)JN\,R$ has a different cost. Comparing their two costs solves the problem of determining the optimal semi-join program for the join of R and S. Notice that the two programs produce results which are allocated on different sites; thus, the possible difference in cost in the transmission of the results also should be taken into consideration.

Moreover, it is important to verify that the semi-join program is profitable with respect to the use of joins as a query processing tactic, which consists in executing the join directly, without previous semi-joins. The cost of using joins as a query processing tactic is that of transmitting one of the operands to the site of the other one. If we perform the join at site s, the corresponding cost is

$$C_{JN} = C_0 + C_1 \times \text{size}(R) \times \text{card}(R)$$

Comparing C_{JN} and C_{SJ}, the semi-join program is profitable if

$$C_0 + C_1 \times (\text{size}(B) \times \text{val}(B[S]) + \text{size}(R) \times \text{card}(R')) < C_1 \times \text{size}(R) \times \text{card}(R)$$

In many cases the above inequality holds, because $\text{size}(B)$ and $\text{val}(B[S])$ are small while $\text{card}(R)$ is larger than $\text{card}(R')$.

14. Explain the different between the terms "deadlock prevention" and "deadlock avoidance". **[MODEL QUESTION]**

Answer:

Deadlock Prevention:

- Preventing deadlocks by constraining how requests for resources can be made in the system and how they are handled (system design).
- The goal is to ensure that at least one of the necessary conditions for deadlock can never hold.

Deadlock Avoidance:

- The system dynamically considers every request and decides whether it is safe to grant it at this point,
- The system requires additional information regarding the overall potential use of each resource for each process.
- Allows more concurrency.

15. Write down the query optimization algorithm of INTEGERS. **[MODEL QUESTION]**

Answer:

Algorithm: Dynamic*-QOA

Input: *MRQ*: multirelation query
Output: result of the last multirelation query
begin
  for *each detachable ORQ$_i$ in MRQ* **do** {*ORQ* is monorelation query}
    run(*ORQ$_i$*)
  MRQ' _list ← REDUCE(*MRQ*) {MRQ repl. By *n* irreducible queries}
  while $n \neq 0$ **do** {*n* is the number of irreducible queries}
    {choose next irreducible query involving the smallest fragments}
    MRQ' ← SELECT_QUERY (*MRQ' _list*);
    {determine fragments to transfer and processing site for *MRQ'*}
    Fragment-site-list ← SELECT_STRATEGY (*MRQ'*);
    {move the selected fragments to the selected sites}
    for *each pair* (*F*, *S*) *in Fragment-site-list* **do**
      move fragment *F* to site *S*
    execute *MRQ'*;
    $n \leftarrow n-1$
  {output is the result of the last *MRQ'*}
end

16. Explain distributed query optimization. **[MODEL QUESTION]**

Answer:

Distributed query optimization refers to the process of producing a plan for the processing of a query to a distributed database system. The plan is called a query execution plan. In a distributed database system, schema and queries refer to logical units of data. In a relational distributed relation database system, for instance, logical units of data are relations. These units may be fragmented at the underlying physical level. The fragments, which can be redundant and replicated, are allocated to different database servers in the distributed system.

Long Answer Type Questions

1. a) What is Serializability in a distributed database?
b) Give a brief introduction to 2PL as a distributed concurrency control method.
c) Let two objects, y be stored at site S1, z, w be stored at site S2. Determine for each of the following executions, whether the execution is Serializable or not.
If yes, determine all possible total order of transactions.
If no, then prove that there is no total order possible.

Execution 1:
S1 : $R_i(x)$ $R_j(x)$ $W_j(x)$ $W_i(x)$
S2 : $R_i(w)$ $R_j(z)$ $W_j(w)$ $W_i(w)$

Execution 2:
S1 : $R_i(x)$ $R_j(x)$ $W_j(y)$ $W_i(y)$
S2 : $W_i(z)$

Execution 3:
S1 : $R_i(x)$ $R_j(x)$ $W_i(x)$ $W_j(y)$
S2 : $R_i(z)$ $R_j(z)$ $W_j(z)$ $W_i(w)$

Execution 4:
S1 : $R_i(y)$ $R_j(x)$ $W_j(x)$
S2 : $W_i(z)$ $R_i(w)$ $R_j(w)$ $W_j(w)$ **[MODEL QUESTION]**

Answer:

a) Serializability in a Distributed Database:

In a distributed database, each transaction performs operations at several sites. The sequence of operation performed by transactions at a site is a *local schedule*. An execution of *n* distributed transactions $T_1, T_2, \ldots\ldots, T_n$ at *m* sites is modeled by a set of local schedules *S*1, *S*2, ..., *Sm*.

If we apply at each node a local concurrency control mechanism, we can ensure that all local schedules are serializable. However, the serializability of local schedules is not sufficient to ensure the correctness of the execution of a set of distributed transactions. Consider, for example, the following two schedules.

$$S1(\text{site 1}): R_i(x)W_i(x)R_j(x)W_j(x)$$

$$S2(\text{site 2}): R_j(y)W_j(y)R_i(y)W_i(y)$$

Both local schedules are serial; however, there is no global serial sequence of execution of both transactions because $T_i < T_j$ in Serial (S1) and $T_j < T_i$ in Serial (S2). In order to guarantee the serializability of distributed transactions, a stronger condition than the serializability of local schedules is required.

The execution of transactions $T_1,.....T_n$ is correct if:

1. Each local schedule Sk is serializable
2. There exists a total ordering of $T_1,.....T_n$ such that, if $T_i < T_j$ in the total ordering, then there is a serial schedule Sk' such that Sk is equivalent to Sk' and $T_i < T_j$ in Serial (Sk'), for each site k where both transactions have executed some action.

b) In strict two-phase locking a transaction cannot write into the database until it has reached its commit point and a transaction cannot release any locks until it has finished writing into the database; therefore locks are not released until after the commit point.

When locking is used for concurrency control, the objects remain locked & are unavailable for other transactions during the atomic commit protocol, although an aborted transaction releases its locks after phase 1 of the protocol.

The advantages are that transaction read only values of committed transactions and there are no cascaded aborts. The disadvantages are that there is limited concurrency and deadlocks may occur.

If all sites use strict two-phase locking and participate in a two-phase commit protocol, the transactions are globally Serializable and hence correct.

c) Execution 1:

S1 : R_i (x) R_j (x) W_j (x) W_i (x)

S2 : R_i (w) R_j (z) W_j (w) W_i (w)

In case of above execution, it is not Serializable. Because there is a conflict between W_j **(x) and W_i (x).**

Here in T_i and T_j both transactions, write operation is being performed on same site S1 and on same object 'x'. And the same conflict is continued in site S2 as Write operation is being performed on same object 'w' by T_i and T_j.

Execution 2:

S1 : R_i (x) R_j (x) W_j (y)W_i (y)

S2 : W_i (z)

In case of above execution, it is not Serializable. Because there is a conflict between W_j **(y) and W_i (y).**

Here in T_i and T_j both transactions, write operation is being performed on same site S1 and on same object 'y'.

Execution 3:

S1 : R_i (x) R_j (x) W_i (x)W_j (y)

S2 : R_i (z) R_j (z) W_j (z) W_i (w)

There is no conflict in site S1. But in site S2, The log is not serial because it is not supporting the total ordering. All the operations of T_i does not precedes T_j completely.

Execution 4:

S1 : R_i (y) R_j (x) W_j (x)

S2 : W_i (z) R_i (w) R_j (w) W_j (w)

Here both the schedules are serial. Here is a global serial sequence of execution of both the transactions. Because $T_i < T_j$ in both serials (S1) and (S2). So, the above execution is Serializable.

All possible total orders,

S1 : R_j (x) W_j (x) R_i (y)

S2 : R_j (w) W_j (w) W_i (z) R_i (w)

Or,

S1 : R_i (y) R_j (x) W_j (x)

S2 : R_i (w) W_i (z) R_j (w) W_j (w)

Etc.

2. a) EMP (ENO, ENAME, TITLE) **[MODEL QUESTION]**

ASG (ENO, PNO, RESP, DUR)

Simplify the following query in SQL, based on the above relations using idem potency rules and give the query graph.

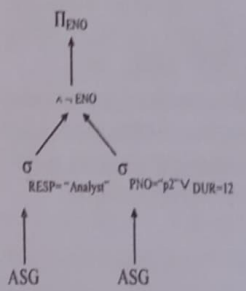
```
Select ENO
From ASG
Where RESP = "Analyst"
And NOT (PNO = "P2" or DUR = 12)
```

b) What are the different layers/steps of query processing?

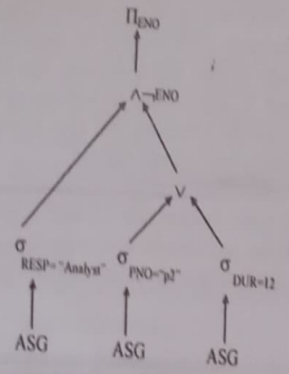
Answer:

a) The relation is: $\Pi_{ENO}(\sigma_{RESP="Analyst" \wedge \neg(PNO="p2" \vee DUR=12)}(ASG))$.

The query graph:



The simplified graph is:



b)

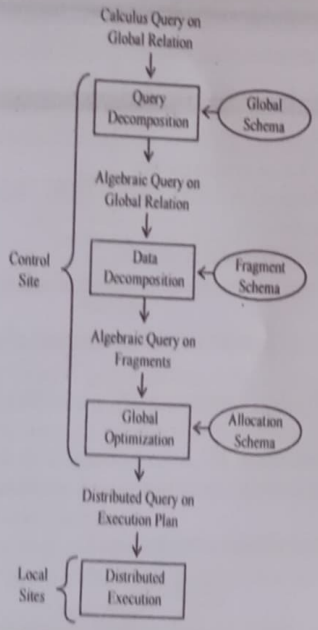


Fig: The Different Layers / Steps of Query Processing

In the above Figure a generic layering scheme for query processing is shown where each layer solves a well-defined sub-problem. Four main layers are involved in distributed query processing.

The first three layers map the input query into an optimized distributed query execution plan. They perform the functions of *query decomposition, data localization, and global*

query optimization. Query decomposition and data localization correspond to query rewriting. The first three layers are performed by a central control site and use schema information stored in the global directory. The fourth layer performs *distributed query execution* by executing the plan and returns the answer to the query. It is done by the local sites and the control site.

Query Decomposition: The first layer decomposes the distributed calculus query into an algebraic query on global relations. The information needed for this transformation is found in the global conceptual schema describing the global relations.

Data Localization: The input to the second layer is an algebraic query on distributed relations. The main role of the second layer is to localize the query's data using data distribution information. This layer determines which fragments are involved in the query and transforms the distributed query into a fragmented query.

Global Query Optimization: The input to the third layer is a fragment query, that is, an algebraic query on fragments. The goal of query optimization is to find an execution strategy for the query, which is close to optimal. An execution strategy for a distributed query can be described with relational algebra operations and *communication primitives (send / receive operations)* for transferring data between sites.

Local Query Optimization: The last layer performed by all the sites having fragments involved in query. Each sub-query executing at one site, called a *local query,* is then optimized using the local schema of the site. At this time, the algorithms to perform the relational operations may be chosen. Local optimization uses the algorithms of centralized systems.

3. Consider the join $RJN_{A=B}S$. Assume that *R* and *S* are at different sites; and disregarded the cost of collection the result of the join. Let $C_0 = 0$ and $C_1 = 1$.

The following profiles are given:
size (*R*) = 50; card (*R*) = 100; val(A[R]) = 50; Size (A) = 3
size (*S*) = 5; card (*S*) = 50; val(B[S]) = 50; Size (B) = 3

$RSJ_{A=B}S$ has selectively $\rho = 0.2$

$SSJ_{B=A}R$ has selectively $\rho = 0.8$

Give the transmission cost of:

i) **performing the join at the site of *R* using semi-join reduction**
ii) **performing the join at the site of *S* using semi-join reduction**
iii) **performing the join at the site of *R* without semi-join reduction**
iv) **performing the join at the site of *S* without semi-join reduction**

Which is the best solution? **[MODEL QUESTION]**

Answer:

i) Performing the join at the site of R using semi-join reduction, at a cost

$$C_0 + C_1 \times size(B) \times val(B[S]) = 0 + 1 \times 3 \times 50 = 150$$

ii) Performing the join at the site of S using semi-join reduction, at a cost

$$C_0 + C_1 \times size(R) \times card(R) = 0 + 1 \times 50 \times 100 = 5000$$

iii) Performing the join at the site of R without semi-join reduction at null cost;

Let, $R' = R \, SJ_{A=B} S$

iv) Performing the join at the site of s without semi-join reduction at null cost;

The overall cost is:

$$2 \times C_0 + C_1 \times (size(B) \times val(B[S]) + size(R) \times card(R'))$$

$$= 2 \times 0 + 1 \times (3 \times 50 + 50 \times 100) = (150 + 5000) = 5150$$

4. a) Why are distributed deadlocks occurred? **[MODEL QUESTION]**
b) What are distributed wait-for graph and local wait-for graph? How wait-for graph helps in deadlock detection?

Answer:

a) Distributed deadlocks can occur in distributed systems when distributed transactions or concurrency control is being used.

b) A **wait-for graph** is a directed graph used for deadlock detection where each site keeps a local wait-for graph.

The nodes of the graph correspond to all the processes that are currently either holding or requesting any of the resources local to that site. A global wait-for graph is maintained in a single coordination process; this graph is the union of all local wait-for graphs.

A global wait-for-graph can be reconstructed under the following conditions:

- On removing or inserting new edge in local wait for graph
- Periodically when number change occurs in locl wait-for graph
- When coordinator invokes detection algorithm,

Centralized Approach may be used for distributed deadlock detection. Here a deadlock-detection coordinator constructs *global wait-for graph,* detects deadlock, and selects victim process to rollback. Due to incomplete or delayed information, the coordinator's constructed global-wait-for graph may not exactly match the real situation. *False cycles/deadlocks* might be incorrectly detected with unnecessary processes being rolled back

5. a) What is query optimization? **[MODEL QUESTION]**
b) Explain distributed cost model with an example.
c) What do you mean by the cardinality of selection?
d) Explain centralized query optimization.

Answer:

a) Having long-running queries not only consumes system resources that makes the server and application run slowly, but also may lead to table locking and data corruption issues. So, query optimization (QO) becomes an important task. As there are many equivalent transformations of same high-level query, aim of QO is to choose one that minimizes resource usage. In general it also reduces total execution time of query. Since the problem is computationally intractable with large number of relations, so strategy adopted is reduced to finding near optimum solution.

b) The distributed cost model includes cost functions to predict the cost of operators, database statistics, base data and formulas to calculate the sizes of intermediate results. Two different types of **cost functions** can be used- they are reduce **total time and** reduce **response time**

Reduce **total time**

- Reduce each cost component (in terms of time) individually, i.e., do as little for each cost component as possible
- Optimize the utilization of the resources (i.e., increase system throughput)

The **total time** is given by the Sum of the time of all individual components. These are local processing time given by CPU time + I/O time and Communication time: fixed time to initiate a message + time to transmit the data

Total time =TCPU _ #instructions + TI/O _ #I/Os + TMSG _ #messages + TTR _ #bytes

Reduce **response time**

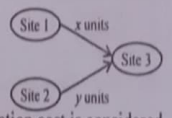
- Do as many things in parallel as possible
- May increase total time because of increased total activity

The response time is the elapsed time between the initiation and the completion of a query

Response time =TCPU _ #seq instructions + TI/O _ #seq I/Os + TMSG _ #seq messages + TTR _ #seq bytes

where #seq x (x in instructions, I/O, messages, bytes) is the **maximum number** of x which must be done sequentially. Any processing and communication done in parallel is ignored.

Example: Query at site 3 with data from sites 1 and 2.



Assume that only the communication cost is considered

Total time = TMSG _ 2 + TTR _ (x + y)

Response time = max{TMSG + TTR _ x, TMSG + TTR _ y}

e) Cardinality is the specification of the number of occurrences of one object that can be related to the number of occurrences of another object.
For example, one object can relate to only one other object (1:1 relationship); one object can relate to many objects (1:N relationship); Some number of occurrences of an object can relate to some other number of occurrences of another object (M:N relationship). Cardinality defines "the maximum number of objects that can participate in a relationship". However, it does not provide an indication of whether or not a particular data object must participate in the relationship.

d) *Centralized of Query Processing*
The main steps in processing a high-level query are illustrated in figure I

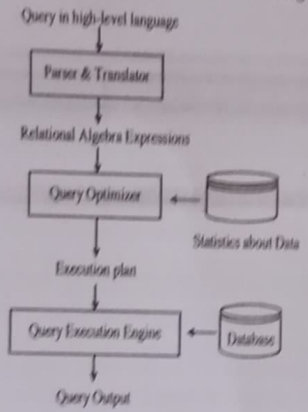


Fig: Steps in query processing process

The functions of Query Parser is parsing and translating a given high-level language query into its immediate form such as relational algebra expressions. The parser need to check for the syntax of the query and also check for the semantic of the query (it means verifying the relation names, the attribute names in the query are the names of relations and attributes in the database). A parse-tree of the query is constructed and then translated into relational algebra expression.
A relational algebra expression of a query specifies only partially how to evaluate a query, there are several ways to evaluate an relational algebra expression. For example, consider the query:
SELECT Salary FROM EMPLOYEE WHERE Salary >= 50,000;
The possible relational algebra expressions for this query are:

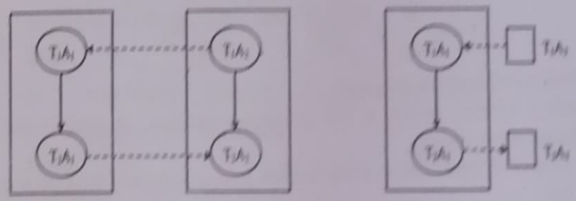
- $\Pi_{Salary}(\sigma_{Salary>=50000}(EMPLOYEE))$
- $\sigma_{Salary>=50000}(\Pi_{Salary}(EMPLOYEE))$

Further, each relational algebra operation can be executed using various algorithms. For example, to implement the preceding selection, we can do a linear search in the EMPLOYEE file to retrieve the tuples with Salary >= 50000. However, if an index

available on the Salary attribute, we can use the index to locate the tuples. Different algorithms might have different cost.
Thus, in order to specify fully how to evaluate a query, the system is responsible for constructing a query execution plan which made up of the relational algebra expression and the detailed algorithms to evaluate each operation in that expression. Moreover, the selected plan should minimize the cost of query evaluation. The process of choosing a suitable query execution plan is known as query optimization This process is perform by Query Optimizer.
One aspect of optimization occurs at relational algebra level. The system attempt to find an expression that is equivalent to the given expression but that is more efficient to execute. The other aspect involves the selection of a detail strategy for processing the query, this relates to choosing the processing algorithm, choosing the indices to use and so on.
Once the query plan is chosen, the Query Execution Engine lastly take the plan, executes that plan and return the answer of the query.

6. What are the additional threat to handle deadlock from centralized to distributed DBMS? Explain centralized and hierarchical deadlock detector. Discuss the effect of replication to create deadlock. [MODEL QUESTION]

Answer:
The problem of deadlock detection is more difficult in a distributed database than in a centralized one, because the circular waiting situation which determines a deadlock can involve several sites and not just one.



(a) A distributed Wait-for graph showing a distributed deadlock (b) A local Wait-for graph

Distributed and local wait-for graphs

There several methods for dealing with deadlocks in a distributed system:

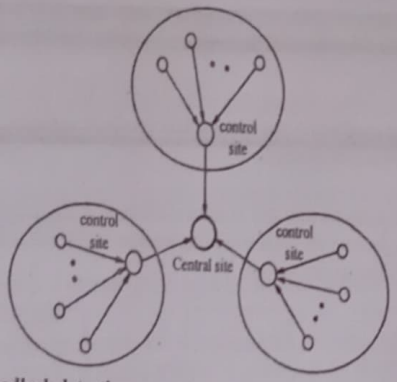
1. Deadlock detection using centralized or hierarchical control.
2. Distributed deadlock detection.
3. Deadlock prevention.

1. Deadlock detection using centralized or hierarchical control:
In hierarchical algorithm, sites are (logically) arranged in hierarchical fashion, and a site is responsible for detecting deadlocks involving only its children sites. These algorithms take advantage of access patterns that are localized to a cluster of sites to optimize performance.

The Menasce-Muntz Algorithm

In the hierarchical deadlock detection algorithm of Menasce and Muntz, all the controllers are arranged in tree fashion. (A controller manages a resource or is responsible for deadlock detection.) The controllers at the bottom-most level (called leaf controllers) manage resources and others (called nonleaf controllers) are responsible for deadlock detection. A leaf controller maintains a part of the global TWF graph concerned with the allocation of the resources at that leaf controller. A nonleaf controller maintains all TWF graphs spanning its children controllers and is responsible for detecting all deadlocks involving all of its leaf controllers.

Whenever a change occurs in a controller's TWF graph due to a resource allocation, wait, or release, it is propagated to its parent controller. The parent controller makes changes in its TWF graph, searches for cycles, and propagates the changes upward, if necessary. A nonleaf controller can receive up-to-date information concerning the TWF graph of its children continuously or periodically.



2. Distributed deadlock detection:

Deadlock detection attempts to find and resolve deadlocks. A detection scheme is evaluated by two criteria:

1. ***Progress***-If there exists an actual deadlock, it must be detected in a finite amount of time.
2. ***Safety***-The scheme must not find a deadlock that is not actually there.

These strategies rely on a Wait-For Graph (WFG) that in some schemes is explicitly built and analyzed for cycles. In the WFG, the nodes represent processes and the edges represent the blockages or dependencies. Thus, if process P_0 is waiting for a resource held by process P_1, there is an edge in the WFG from the node for process P_0 to the node for process P_1.

In the AND model (resource model), a cycle in the graph indicates a deadlock. In the OR model, a cycle may not mean a deadlock since any of a set of requested resources may unblock the process. A knot in the WFG is needed to declare a deadlock. A knot exists when all nodes that can be reached from some node in a directed graph can also reach that node.

In the fully distributed deadlock-detection algorithm, all controllers share equally the responsibility for detecting deadlock. In this scheme, every site constructs a wait-for graph that represents a part of the total graph. The idea is that, if deadlock exists, a cycle will appear in (at least) one of the partial graphs.

Deadlock Handling can be done using Process Termination or Resource Preemption

Process Termination

After a detection algorithm determines that a deadlock exists, there are different approaches for system recover from the deadlock automatically:

1. Abort all deadlocked processes to beak the circular wait
2. Abort one process at a time until the deadlock cycle is eliminated

To start a partial termination, we need to decide in which order to abort the deadlocked processes. Usually, the sequence of termination is decided by:

1. Priority of processes
2. Duration of processes
3. Number of types of resources held by processes
4. Number of resources being requested
5. Number of processes to be terminated

Resource Preemption

To eliminate deadlocks using resource preemption, resources held by processes need to be reallocated to other processes successively until a deadlock cycle is broken.

1. **Selecting a Victim**
   The order of preemption will be determined based on minimum cost. Cost factors include number of resources being held by deadlocked processes, amount of time the deadlocked processes consumed, etc.
2. **Rollback**
   If a resource is preempted from a process, this process must be rolled back to a safe state, and restarted from that state. Usually the simplest way is total rollback, i.e. abort the process and then restart it. The effective way is roll back the process only as far as necessary to break the deadlock.
3. **Starvation**
   Same process may always be picked as victim. As a result this process will never be completed. To avoid starvation, number of rollback should be included as cost factor.

7. Justify the following statements: [MODEL QUESTION]

a) Unique timestamp generation is difficult in DDBMS than centralized DBMS.

b) Query graph identifies redundant relation in an SQL.

Answer:

a) Unique timestamp generation is difficult in DDBMS than centralized DBMS:

A timestamp is a unique identifier created by the DBMS to identify a transaction. Typically, timestamp values are assigned in the order in which the transactions are submitted to the system, so a timestamp can be thought of as the *transaction start time*.

Concurrency control techniques based on timestamp ordering do not use locks; hence, *deadlocks cannot occur.*

Generation of Timestamp:

Timestamps can be generated in DBMS several ways. One possibility is to use a counter that is incremented each time its value is assigned to a transaction. The transaction timestamps are numbered 1, 2, 3, . . . in this scheme. A computer counter has a finite maximum value, so the system must periodically reset the counter to zero when no transactions are executing for some short period of time. Another way to implement timestamps is to use the current date/time value of the system clock and ensure that no two timestamp values are generated during the same tick of the clock.

In case to DDBMS, the timestamp generation is a bit more complicated as the site must also be identified. The general approach for timestamping in DDBMS is to use the concatenation of the local timestamp with a unique identifier, <local timestamp, site identifier>. The site identifier is placed in the least significant position to ensure that events can be ordered according to their occurrence as opposed to their location

b) Query graph identifies redundant relation in an SQL:

In a Query graph Nodes represent operand or result relation and Edge represents a join if both connected nodes represent an operand relation, otherwise it is a projection. It can be used to simplify an SQL query by eliminating redundancies, e.g., redundant predicates. Redundancies are often due to semantic integrity constraints expressed in the query language e.g., queries on views are expanded into queries on relations that satisfy certain integrity and security constraints.

Consider the following query:

```
SELECT TITLE FROM EMP
WHERE
EMP.ENAME = "J. Doe"
OR
(NOT(EMP.TITLE = "Programmer")
AND
( EMP.TITLE = "Elect. Eng."
OR
EMP.TITLE = "Programmer" )
AND
NOT(EMP.TITLE = "Elect. Eng."))
```

Let p1 be ENAME = "J. Doe", p2 be TITLE = "Programmer" and p3 be TITLE = "Elect.Eng."

Then the qualification can be written as p1 ∨ (¬p2 ∧ (p2 ∨ p3) ∧ ¬p3) and then be transformed into p1

So the simplified query is as follows:

```
SELECT TITLE
FROM
EMP
WHERE
EMP.ENAME = "J. Doe
```

8. Write the R* algorithm. **[MODEL QUESTION]**

Answer:

The input to the algorithm is the localized query expressed as a relational algebra tree, the location of relations and their statistics.

Input: QT: query tree

Output: strat: minimum cost strategy

```
begin
    for each relation R_i ∈ QT do
    begin
        for each access path AP_ij to R_i do
            determine cost (AP_ij)
        end-for
        best-AP_i ← AP_ij with minimum cost
    end
    for each order (R_i1, R_i2, ...., R_in) with i = 1,...,n! do
    begin
        build strategy (...((bestAP_i1, ....,⋈ R_i2) ⋈ R_i3) ⋈ ... R_in)
        compute the cost of strategy
    end-for
strat ← strategy with minimum cost
for each site k storing a relation involved in QT do
begin
    LS_k ← local strategy (strategy, k)
    send (LS_k , site k) {each local strategy is optimized at site k}
end-for
end
```

9. Explain the distributed SDD-1 query optimization protocol. **[MODEL QUESTION]**

Answer:

SDD-1 Algorithm

Step 1: In the execution strategy (call it ES), include all the local processing

Step 2: Reflect the effects of local processing on the database profile

Step 3: Construct a set of beneficial semijoin operations (BS) as follows:
BS ⇒ Ø For each semijoin SJi BS ← BS ∪ SJi if cost(SJi) < benefit(SJi)

Iterative Process

Step 4: Remove the most beneficial SJi from BS and append it to ES
Step 5: Modify the database profile accordingly
Step 6: Modify BS appropriately – compute new benefit/cost values – check if any new semijoin need to be included in BS
Step 7: If BS ≠ Ø, go back to Step 4.

Assembly Site Selection

Step 8: Find the site where the largest amount of data resides and select it as the assembly site

Postprocessing

Step 9: For each Ri at the assembly site, find the semijoins of the type Ri Rj where the total cost of ES without this semijoin is smaller than the cost with it and remove the semijoin from ES.
Step 10: Permute the order of semijoins if doing so would improve the total cost of ES.

10. a) Consider the following schema: **[MODEL QUESTION]**

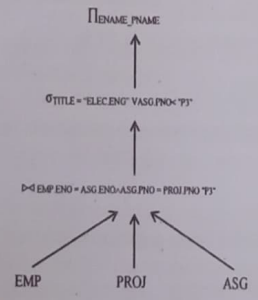
EMP = (ENO, ENAME, TITLE)
ASG = (ENO, PNO, RESP, DUR)
PROJ = (PNO, PNAME, BIDGET, LOC)
Consider the following query:

```
SELECT ENAME, PNAME
FROM EMP, ASG, PROJ
WHERE EMP.ENO = ASG.ENO
AND ASG.PNO = PROJ.PNO
AND (TITIE = 'ELECT.ENG'OR ASG.PNO<'.P3')
```

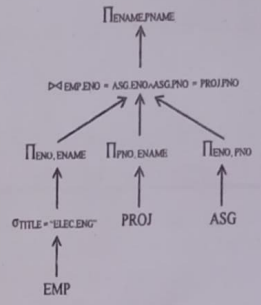
Draw the canonical tree and then transform it to an optimised tree.

Answer:

The canonical tree for the above query is as follows:



The optimized query tree for the above query is listed below:



b) Simplify the following query by using the idempotency rules: **[MODEL QUESTION]**

Select ENO from ASG where (not(TITLE = 'programmer')
and (TITLE = 'programmer' or TITLE = 'Elect.eng')
and not (TITLE = 'Elect.eng') or ENAME = 'Sachin'.

Answer:

The idempotency rules:

1. $p \wedge p = p$
2. $p \vee p = p$
3. $p \wedge true = p$
4. $p \vee false = p$
5. $p \wedge false = false$
6. $p \vee true = true$
7. $p \wedge \neg p = false$
8. $p \vee \neg p = true$
9. $p_1 \wedge (p_1 \vee p_2) = p_1$
10. $p_1 \vee (p_1 \wedge p_2) = p_1$

Let the simplification proceeds as follows:

Let p_1 be TITLE = "Programmer",
p_2 be TITLE = "Elect. Eng.", and
p_3 be ENAME = "Sachin".

The query qualification is:

$$(\neg p_1 \wedge (p_1 \wedge p_2) \wedge \neg p_2) \vee p_3$$

Elimination of Redundancy

The disjunctive normal form for this qualification is obtained by applying rule 5 which yields:

$$(\neg p_1 \wedge ((p_1 \wedge \neg p_2) \vee (p_2 \wedge \neg p_2))) \vee p_3$$

and then rule 3 yields:

$$(\neg p_1 \wedge p_1 \wedge \neg p_2) \vee (\neg p_1 \wedge p_2 \wedge \neg p_2) \vee p_3$$

By applying rule 7, we obtain

$(\text{false} \wedge \neg p_2) \vee (\neg p_1 \wedge \text{false}) \vee p_3$

By applying the same rule, we get

$(\text{false} \wedge \text{false}) \vee p_3$

Which is equivalent to p_3 by rule 4.

So the simplified SQL is

Select ENO from ASG where ENAME = 'Sachin'.

11. Assume the fragmentation of a global relation R (A,B,C) is described by the following predicates:

P1 : (1 < = A < = 4)
P2 : (5 < = A < = 7) AND (1 < = B < = 5)
P3 : (5 < = A < = 7) AND (6 < = BN < = 10)
P4 : (8 < = A < = 10) AND (1 < = C < = 5)
P5 : (8 < = A < = 10) AND (6 < = C < = 10)

Introduce the CUT operation in the following parametric queries:

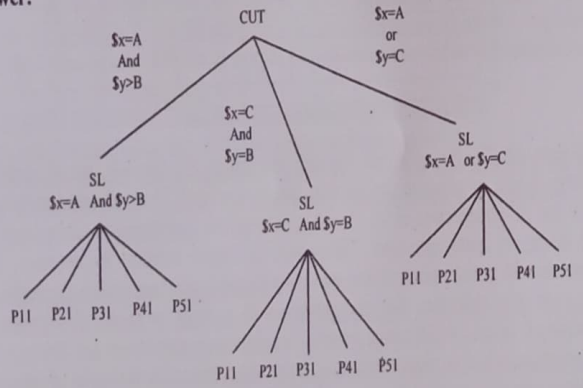
i) $SL_{A=\$X \text{ AND } B<\$Y}R$

ii) $SL_{C=\$X \text{ AND } B=\$Y}R$

iii) $SL_{A=\$X \text{ OR } C=\$Y}R$

[MODEL QUESTION]

Answer:



Where

P1 : (1 < = A < = 4)
P2 : (5 < = A < = 7) AND (1 < = B < = 5)
P3 : (5 < = A < = 7) AND (6 < = BN < = 10)
P4 : (8 < = A < = 10) AND (1 < = C < = 5)
P5 : (8 < = A < = 10) AND (6 < = C < = 10)

12. Consider the following global, fragmentation and allocation schema:
Global schema: STUDENT (STUD-ID, DEPT, NAME)
Fragmentation schema: $STUDENT_1 = SL_{DEPT="EE"}$ STUDENT
$STUDENT_2 = SL_{DEPT="CS"}$ STUDENT
(Assume that EE and CS are only possible value for DEPT).
(Allocation schema: $STUDENT_1$ at sites 1, 2 and $STUDENT_2$ at sites 3, 4.)
Write a query that moves the student having number 802 from department "EE" to department "CS", at levels 1, 2 and 3 of transparency. **[MODEL QUESTION]**

Answer:

Fragmentation Transparency (Level 1)

```
Update STUDENT
Set DEPT="CS"
Where STUD-ID =802;
```

Location Transparency (level 2)

```
Select NAME into $NAME from STUDENT
Where STUD-ID=802;
Insert into STUDENT (STUD-ID, DEPT, NAME)
Values (802, 'CS', $NAME);
DELETE from STUDENT where STUD-ID=802;
```

Local Mapping Transparency (Level 3)

```
Select NAME into $NAME from STUDENT at SITE 1
Where STUD-ID=802;
Insert into STUDENT, AT SITE 3 (STUD-ID, DEPT, NAME)
Values (802, 'CS', $NAME);
DELETE from STUDENT at SITE 1 where STUD-ID=802;
DELETE from STUDENT at SITE 2 where STUD-ID=802;
```

13. Write sort notes on the following: **[MODEL QUESTION]**

a) Primary copy locking
b) Weight-majority locking
c) Write-locks-all
d) Quorum based protocol in distributed concurrency control
e) Wound wait protocol
f) Conservative timestamp ordering protocol
g) Relation between database integrity and database security

Answer:

a) Primary copy locking:

In the primary copy locking approach, all locks for a data item x are requested at the site of the primary copy. We will assume first that also all the read and write operations are performed on this copy; however, writes are then propagated to all other copies. Considering a database, if we choose $x1$, $y1$, and $z2$ as primary copies, then the availability of the database for the 12 transactions is the same as in the majority approach (Figure), because the assignment of votes which we had applied determined a primary

copy situation. This will always happen if we apply weighted voting to items having only two copies. However, this is a particular case; if many copies exist, so that different majorities can be built, then the two copy approach by a particular assignment of votes.
Several enhancements of the primary copy approach exist which make it more attractive. The principal ones are:

1. Allowing consistent reads at different copies than the primary, even if read locks are requested only at the primary; this enhances the locality of reads
3. Allowing the migration of the primary copy if a site crash makes it unavailable; this enhances availability.
4. Allowing the migration of the primary copy depending on its usage pattern. This also enhances the locality aspect.

b) Weight-majority locking:

Transactions

Partitions	1	2	3	4	5	6	7	8	9	10	11	12	10′	11′	12′
A	-	-	-	-	1	1	-	1	1	-	1	1	2	(1,2)	(1,2)
B	-	-	-	-	2	2	-	2	2	-	2	2	2	2	(1,2)
C	2	2	2	2	2	2	2	2	2	2	2	2	2	2	(1,2)
D	-	-	-	-	1	1	-	1	1	-	1	1	-	1	(1,2)

Weighted majority, with

$V(x_1) = V(y_1) = V(z_2) = 2$ and
$V(x_2) = V(y_3) = V(z_3) = 2$

Weight-majority locking: The pure majority locking approach is not very suitable for our example, because two copies of each data item exist; hence to lock a majority we must lock both. We will therefore consider a weighted majority approach, or quorum approach, which adopts the same rules which has been used for quorum-based commitment and termination protocols.
These rules, applied to the locking problem, consist of assigning to each data item x a total number of votes $V(x)$, and assigning votes $V(xt)$ to each copy z, in such a way that $V(x)$ is the sum of all $V(x_i)$. A read quorum $V_r(x)$ and a write quorum $V_w(x)$ are then determined, such that:

$$V_r(x)+V_w(x)>V(x)$$

$$V_w(x)>V(x)/2$$

A transaction can read (write) x if it obtains read (write) locks on so many copies of x that the sum of their votes is greater than or equal to $V_r(x)(V_w(x))$. Due to the first condition, all conflicts between read and write operations are determined, because two transactions which perform a read and a write operation on x cannot reach the read and write quorum using two disjoint subsets of copies. Likewise, because of the second condition, all conflicts between write operations are determined. Notice that the second condition can be omitted if transactions read all data items which are written.

Let us assign votes for the copies of data items in the following way:
$V(x) = V(y) = V(z) = 3$
$V(x1) = V(y1) = V(z2) = 1$
$V(z2) = V(y3) = V(z3) = 1$
With this assignment we can now consider the availability of the system in the case of partitions. We choose the read and write quorums to be 2 for all data items. The availability for the 12 transactions is shown in Figure. The following can be observed:
1. Transactions 1, 2, 3, 4, 7, and 10 have all the same availability. They are characterized by the fact that they access all three data items either for reading or for writing or for both. Since the read quorum is equal to the write quorum, it makes no difference whether the data item is read or written from the viewpoint of availability. For the same reason, transactions 5, 6, 8, and 11, which access only data items x and y, have the same availability. Also, transactions 9 and 12 have the same availability of the latter group, because the copy with highest weight for data item x resides at the same site as the copy with highest weight for y.
2. The availability for update transactions is greater with the weighted majority approach than with write-locks-all, while the availability for read-only transactions is smaller.
3. With this method, read-only transactions increase their availability if they can read an inconsistent database, i.e., if they do not need to lock items; in fact, columns 10′, 11′ and 12′ are the same for the majority approach as for the write-locks-all approach

c) Write-locks-all:

Transactions

Partitions	1	2	3	4	5	6	7	8	9	10	11	12	10′	11′	12′
A	-	-	-	-	-	-	-	-	-	2	(1,2)	(1,2)	2	(1,2)	(1,2)
B	-	-	-	-	-	-	-	-	-	2	2	(1,2)	2	2	(1,2)
C	-	-	-	-	-	-	2	2	2	2	2	2	2	2	2
D	-	-	-	-	-	-	-	-	-	-	1	(1,2)	-	1	(1,2)

Write-locks-all, read-locks-one

Write-locks-all: The Figure shows, for each transaction and for each type of partition, the group, if any, in which the transaction can be executed. The read-only transactions 10′, 11′ and 12′ are the same as 10, 11 and 12, except that consistency is not required for them; hence, no locks are used. The values of Figure confirm the intuition: For transactions with a small write-set and especially for read-only transactions, the system is much more available than for transactions with a large write-set. For read-only transactions it makes no difference whether consistency is required or not, because the copy which is locked is required anyway for reading. Note that read-only transactions sometimes can run in more than one group, because if a data item has two copies in two different groups, then no update transaction can write on it and read-only transactions can use each copy consistently.
If we make the assumption that no partitions occur, but only site crashes, then the same approach can be used as with a non-redundant database; i.e., the updates of non-available copies of data items can be spooled. In this case, the availability of the database for update transactions increases very much. In fact, since only the read-set matters in this

case, transactions 1, 4, and 7 have the same availability as transaction 10; transaction 2, 5, and 8 as transaction 11; and transactions 3, 6 and 9 as transaction 12. Obviously, the example must be carefully interpreted. The fact that a transaction can run in a given group means now that it can be run if all other sites are down, instead of building separate groups. The high increase in availability is obtained at the risk of catastrophic partitions.

d) Quorum-based protocol:

*A **quorum** is the minimum number of members of a deliberative body necessary to conduct the business of that group. Ordinarily, this is a majority of the people expected to be there, although many bodies may have a lower or higher quorum.*

Quorum-based voting can be used as a replica control method, as well as a commit method to ensure transaction atomicity in the presence of network partitioning.

Quorum-based voting in commit protocols

In a distributed database system, a transaction could be executing its operations at multiple sites. Since atomicity requires every distributed transaction to be atomic, the transaction must have the same fate (commit or abort) at every site. In case of network partitioning, sites are partitioned and the partitions may not be able to communicate with each other. This is where a quorum-based technique comes in. The fundamental idea is that a transaction is executed if the majority of sites vote to execute it.

Every site in the system is assigned a vote Vi. Let us assume that the total number of votes in the system is V, and the abort and commit quorums are Va and Vc, respectively. Then the following rules must be obeyed in the implementation of the commit protocol:

1. Va + Vc > V, where 0 Va, Vc V.
2. Before a transaction commits, it must obtain a commit quorum Vc.
3. Before a transaction aborts, it must obtain an abort quorum Va.

The first rule ensures that a transaction cannot be committed and aborted at the same time. The next two rules indicate the votes that a transaction has to obtain before it can terminate one way or the other.

Quorum-based voting for replica control

In replicated databases, a data object has copies present at several sites. To ensure serializability, no two transactions should be allowed to read or write a data item concurrently. In case of replicated databases, a quorum-based replica control protocol can be used to ensure that no two copies of a data item are read or written by two transactions concurrently.

In the quorum-based voting for replica, each copy of a replicated data item is assigned a vote. Each operation then has to obtain a read quorum (Vr) or a write quorum (Vw) to read or write a data item, respectively. If a given data item has a total of V votes, the quorums have to obey the following rules:

1. Vr + Vw > V
2. Vw > V/2

The first rule ensures that a data item is not read and written by two transactions concurrently. The second rule ensures that two write operations from two transactions cannot occur concurrently on the same data item. The two rules ensure that one-copy serializability is maintained.

e) Wound wait protocol:

One of the problems of locking mechanisms is the potential for deadlock. Deadlock occurs when two or more transactions are mutually waiting for each other's resources. To avoid such deadlocks, some database management systems give priority to older transactions. That is, if an older transaction requires access to a data item that is locked by a younger transaction, the younger transaction is forced to release all of its data items, its activities are rolled back (based on the log), the older transaction is given access to the data item, and the younger transaction is forced to start again. If a younger transaction is repeatedly preempted, it will grow older in the process and ultimately become one of the older transactions. This protocol, known as the wound-wait protocol (old transactions wound young transactions, young transactions wait for old ones), assures that every transaction will ultimately be allowed to complete its task.

f) Conservative timestamp ordering protocol:

It is a timestamp ordering scheme for every site (i.e, a transaction manager, TM) in the system. There is a scheduler process that keeps track of all the transaction requests arriving at a certain data manager (DM). This scheduler also maintains two internal queue data structures for each TM, namely a READ and a WRITE queue. These queues, as the name suggests, hold READ and WRITE requests for a TM and are ordered by using the timestamp ordering scheme previously mentioned.

When a TM wants to perform a READ or a WRITE operation to a certain DM, it sends a request to the scheduler responsible for that DM. A request consists of a READ/WRITE command, a variable-value pair to be read/written (value is NIL in case of a READ-op), and timestamp (TS). The scheduler is then responsible for either allowing the request execution or for buffering the request in case of a potential conflict (we will define "potential" later, don't worry). The scheduler is also responsible for making sure that each request is processed by the algorithm when it comes in, and for testing if any of the buffered requests can be executed.

Whenever a read or write request is buffered or executed, buffered requests are tested to see if any of them can be executed, by running the algorithm for each one of them.

The algorithm guarantees serialization simply by ordering every action, be it conflictant or not. That's why it's called conservative ordering algorithm, by the way. As an example, assume the following ordering of requests arriving to a DM-scheduler:

1st to arrive, w/ TS=1 : REQ_READ (TM_1, DM_1, x, 1);
2nd to arrive, w/ TS=2 : REQ_READ (TM_2, DM_1, x, 2);
3rd to arrive, w/ TS=3 : REQ_READ (TM_2, DM_1, y, 3);
4th to arrive, w/ TS=4 : REQ_WRITE(TM_1, DM_1, y, 4);

These operations are clearly serializable, for it represents the execution of transaction 2 before transaction 1. One can see, however, that all the requests are buffered because of empty queues in the system, and because the final write request has a timestamp greater than the ones in the system. This example also illustrates a very undesirable situation that may occur when using CTO algorithms: **the system can be blocked**, even though it has no conflicting actions.

It should be clear also from the simple example above, that with only two transactions and one write request, every possible ordering of actions is valid and serializable, and there is no reason for blocking. The write request of transaction 1 could, for example, be executed in any order.

However, in the case of large number of requests arriving from a large number of transactions, or when the possibility of conflict among requests is big, the conservative algorithm does its job fairly well. By making sure that all the requests are processed in timestamp order, it guarantees a consistent and serializable execution of events.

To handle those blocking situations, where termination is not guaranteed, it is recommended that "dummy" requests be sent regularly to the scheduler.

g) Relation between database integrity and database security:

Data are the most important asset to any organization. Therefore, it must be made sure that data is valid and secure all the time. Data integrity and Data security are two important aspects of making sure that data is useable by its intended users. Data integrity makes sure that the data is valid. Data security makes sure that data is protected against loss and unauthorized access.

Data integrity and data security are two different aspects that make sure the usability of data is preserved all the time. Main difference between integrity and security is that integrity deals with the validity of data, while security deals with protection of data. Backing up, designing suitable user interfaces and error detection/correction in data are some of the means to preserve integrity, while authentication/authorization, encryptions and masking are some of the popular means of data security. Suitable control mechanisms can be used for both security and integrity.

TRANSACTION MANAGEMENT

Multiple Choice Type Questions

1. Atomicity of transaction demands [MODEL QUESTION]
   a) all the transaction's operations will be performed
   b) none of the transaction's operations will be performed
   c) no stable state
   d) none of these

Answer: (a) or (b)

2. The unit of transfer to and from disk is [MODEL QUESTION]
   a) file b) block c) page d) information

Answer: (a)

3. Granularity means [MODEL QUESTION]
   a) size of memory b) size of data c) locks d) transaction

Answer: (d)

4. Which of the following refers to the operation of copying and maintaining database object in multiple databases belonging to a distributed system? [MODEL QUESTION]
   a) Backup b) Recovery
   c) Replication d) None of these

Answer: (c)

5. Which component has the right to communicate distributed information with another component of different machine for running distributed transaction correctly? [MODEL QUESTION]
   a) Root agent b) DTM c) LTM d) None of these

Answer: (b)

6. 3PC protocol ensures non-blocking in case of ..... failure. [MODEL QUESTION]
   a) site b) network c) partition d) coordinator

Answer: (d)

7. The highest level in the hierarchy of data organization is called [MODEL QUESTION]
   a) data bank b) database c) data file d) data record

Answer: (b)

8. Which of the following strategies ensures that either all the database are updated or none of them? [MODEL QUESTION]
   a) read one write all b) two phase commit
   c) two phase locking d) update propagation

Answer: (b)

9. Which of the following is correct? [MODEL QUESTION]
   a) remote data accessing and distributed database is same
   b) derived fragmentation reduces redundancy
   c) 3 phase commit is an example of blocking commitment protocol
   d) basic 2 phase locking protocol may lead to cascading rollback

Answer: (d)

10. Which of the following is incorrect? [MODEL QUESTION]
   a) redundancy is allowed in distributed database to increase reliability
   b) cold restart is required after catastrophic failure
   c) majority locking protocol cannot ensure serializabilty
   d) primary copy method reduces reliability of distributed database

Answer: (c)

11. A transaction manager is which of the following? [MODEL QUESTION]
   a) maintains a log of transactions
   b) maintains before and after database images
   c) maintains appropriate concurrency control
   d) all of these

Answer: (d)

12. Which of the following is true concerning a global transaction? [MODEL QUESTION]
   a) the required data are at one local site and the distributed DBMS routes requests as necessary
   b) the required data are located in at least one nonlocal site and the distributed DBMS routes requests as necessary
   c) the required data are at one local site and the distributed DBMS passes the request to only the local DBMS
   d) the required data are located in at least one nonlocal site and the distributed DBMS passes the request to only the local DBMS

Answer: (b)

13. The name of the agent at the site of origin of a transaction is [MODEL QUESTION]
   a) main agent b) root agent
   c) principle agent d) none of these

Answer: (b)

14. Which of the following techniques is used when the information on stable storage is lost? [MODEL QUESTION]
   a) Shadow paging b) Check point
   c) Cold restart d) None of these

Answer: (c)

15. Which of the following failures does not occur in a distributed system? [MODEL QUESTION]
   a) Disk failures b) Link failures c) Message Loss d) None of these

Answer: (c)

16. Which of the following refers to the operation multiple physical database belonging to a distributed system? [MODEL QUESTION]
   a) Backup b) Recovery c) Replication d) All of these

Answer: (c)

17. In basic TO algorithm [MODEL QUESTION]
   a) younger transaction are executed first
   b) older transaction are restarted
   c) older transaction are executed first
   d) both older and younger transaction are restarted

Answer: (c)

Short Answer Type Questions

1. Explain unilateral abort capability in the context of 2-phase commit protocol. [MODEL QUESTION]

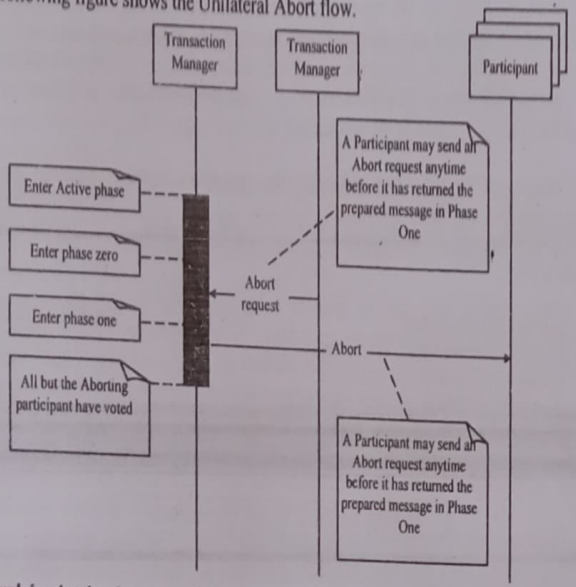
Answer:

Until a participant votes on the outcome of the transaction, any participant can decide to unilaterally stop the transaction by issuing an Abort request to its transaction manager. This ability is known as a Unilateral Abort.

After a transaction manager receives an Abort request from one of its participants, it immediately transitions the transaction to the Aborting state, which guarantees an Abort outcome. All other participants will be notified of the Abort outcome, although it is possible that the root application does not discover the Abort outcome until it attempts to complete the transaction or perform some other operation involving the transaction manager or another participant.

After a specified transaction manager enters the Aborting state, it does not issue any further Phase Zero notifications or Phase One requests to vote. For a transaction that spans two or more transaction managers due to propagation, it is possible for the Abort outcome decision to race with other Phase Zero or Phase One activity as it is communicated between the transaction managers.

The following figure shows the Unilateral Abort flow.



2. Explain checkpoint and cold restart of a distributed database system. **[MODEL QUESTION]**

Answer:

Check point: A checkpoint is a synchronization record placed in the log to note a point when all concluded transactions are safely on disk. It limits the log segment needed to recover from a failure.

Cold restart: After a fatal failure of the system, the actual consistent state of the DBMS is lost. An archived older consistent state is to be recovered. As a DDBMS has global consistent state, a site cannot change their state independently from the other sites. *If one site has to make cold restart then all of the other sites must make cold restart.*

For each transaction T, the state contains the updates performed by all subtransactions of T at any site (T is contained in the global state), or it does not contain any of them.

If the T transaction is contained in the global state, then all conflicting transactions which have preceded T in the serialization order, are also contained in the global state.

The duration in the ACID concept cannot be guaranteed as the effect of some transactions will be lost due to the fatal failure. But all of the transactions before a given point of time are recovered.

The consistency requirements are met as recovery replies the earlier consistent transactions.

The checkpoint de **replication** notes a point of time when a snapshot is taken about a consistent state of the database system. A global checkpoint is a set of local checkpoints which are performed at all sites of the system and are synchronized by the following condition:

If a sub transaction of a transaction T is contained in the local checkpoint at some time then all other sub transaction of T must be contained in the corresponding local checkpoints at other sites.

In the case of cold restart

- take the latest available local checkpoint
- take the corresponding global checkpoint and re-establish the local states at every sites according to this global checkpoint

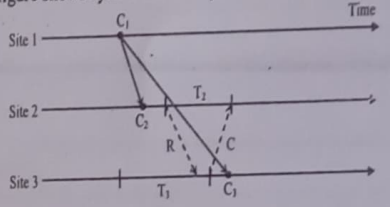
Generation of global checkpoint:

The snapshot, the checkpoint must be performed when there is no active transaction. The exact solution needs a lot of idle time as site A should wait for site B.

Loosely synchronized checkpoints:

The sites can perform the checkpoint within a given interval. Every checkpoint is identified by an order number. The TM should guarantee that if a T finishes before Ci then all of its sub-transactions should finishes before the local Ci.

The following figure shows synchronization problem for global checkpoints.



T_2 and T_3 are sub-transactions of transaction T; T_3 is coordinator for 2-phase-commitment.
C_1, C_2, C_3 are local checkpoints (started from site 1)

⟶ "Write checkpoint" messages

- - - -> Messages of 2-phase-commitment protocol (R = READY, C = COMMIT)

Synchronization problems for global checkpoints

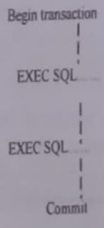
3. What is the flat transaction? Explain with example. **[MODEL QUESTION]**

Answer:

Flat Transaction

Consists of:

- Computation on local variables not seen by DBMS; hence will be ignored in most future discussion
- Access to DBMS using call or statement level interface
- This is **transaction schedule**; commit applies to these operations
- No internal structure
- Accesses a single DBMS
- Adequate for simple applications



4. Explain the cold starts and warm starts. **[MODEL QUESTION]**

Answer:

Cold restart:

After a fatal failure of the system, the actual consistent state of the DBMS is lost. An archived older consistent state is to be recovered. As a DDBMS has global consistent state, a site cannot change their state independently from the other sites. If one site has to make cold restart then all of the other sites must make cold restart.

Local failure - global recovery

Consistent global state:

- ***Atomicity concept:***

For each transaction T, the state contains the updates performed by all subtransactions of T at any site (T is contained in the global state), or it does not contain any of them.

- ***Serializability concept:***

If the T transaction is contained in the global state, then all conflicting transactions which have preceded T in the serialization order, are also contained in the global state.

The duration in the ACID concept cannot be guarantied as the effect of some transactions will be lost due to the fatal failure. But all of the transaction before a given point of time are recovered.

The consistency requirements are met as recovery replies the earlier consistent transactions.

Recovery of the participant is performed by the **warm restart** protocol, based on the last record written in the log:

- when it's an action or abort, actions are ***undone***; when it's a commit, actions are ***redone***; either way, the failure has occurred before starting the commit protocol;
- when it's ready, failure has occurred during 2PC; participant is ***in doubt*** about the result of the transaction.
- During a warm restart, the identifiers of the transactions in doubt are collected in the ***ready set***. For each of them the final transaction outcome must be requested from the TM.
- This can happen as a result of a direct ***(remote recovery)*** request from the RM or as a repetition of the second phase of the protocol.

5. What is local autonomy? **[MODEL QUESTION]**

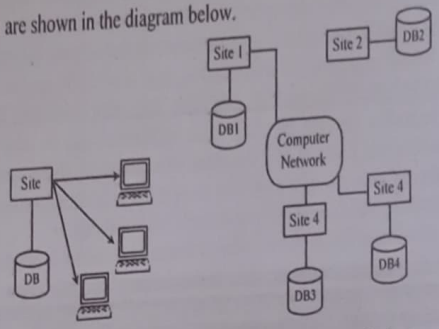
Answer:

Local autonomy means that all the data in the distributed network is owned and managed locally. For example, a site in one location might have a remote database that participates in a national distributed system. While functioning as a part of the distributed network, the local database continues to process local operations independently from the overall distributed system, and does not rely on the distributed system to function.

6. Discuss the reasons why an organization with a centralized database system might like to move to a distributed database environment. Outline the advantages and disadvantages of such a decision. **[MODEL QUESTION]**

Answer:

The two systems are shown in the diagram below.



Centralized Database System Distributed Database System

Changes in the corporate environment toward decentralized operations have prompted organizations to move toward distributed database systems that complement the new decentralized organization.

Today's global enterprise may have many local-area networks (LANs) joined with a WAN, as well as additional data servers and applications on the LANs. Client applications at the sites need to access data locally through the LAN or remotely through the WAN. For example, a client in Tokyo might locally access a table stored on the Tokyo data server or remotely access a table stored on the New York data server.

In a distributed database environment, mainframe computers may be needed at corporate or regional headquarters to maintain sensitive corporate data, while clients at remote sites use minicomputers and server-class workstations for local processing.

Both centralized and distributed database systems must deal with the problems associated with remote access:

- Network response slows when WAN traffic is heavy. For example, a mission-critical transaction-processing application may be adversely affected when a decision-support application requests a large number of rows.
- A centralized data server can become a bottleneck as a large user community contends for data server access.
- Data is unavailable when a failure occurs on the network.

Distributed databases can be defined as a collection of multiple, logically interrelated databases distributed over a computer network. All the database must be logically related that are managed by DDBMS (distributed database management system).

While a centralized database has all its data on one place. As it is totally different from distributed database which has data on different places. In centralized database as all the data reside on one place so problem of bottle-neck can occur, and data availability is not efficient as in distributed database.

7. Write the difference between centralized and distributed DBMS with respect to DBA, redundancy, indexing, reliability and performance. [MODEL QUESTION]

Answer:

The features which characterize the traditional database approach are centralized control, data independence, reduction of redundancy, complex physical structures for efficient access, integrity, recovery, concurrency control, privacy, and security.

- The possibility of providing ***centralized control*** over the information resources of a whole enterprise or organization was considered as one of the strongest motivations for introducing databases; they were developed as the evolution of information systems in which each application had its own private files. The fundamental function of a **database administrator (DBA)** was to guarantee the safety of data; the data itself was recognized to be an important investment of the enterprise which required a centralized responsibility.
  In distributed databases, the idea of centralized control is much less emphasized.
- In distributed databases, data independence has the same importance as in traditional databases; however, a new aspect is added to the usual notion of data independence, namely, ***distribution transparency***.
- ***Reduction of redundancy*** In traditional databases, redundancy was reduced as far as possible for two reasons: first, inconsistencies among several copies of the same logical data are automatically avoided by having only one copy, and second, storage space is saved by eliminating redundancy. Reduction of redundancy was obtained by data sharing, i.e., by allowing several applications to access the same files and records. In distributed databases, however, there are several reasons for considering data redundancy as a desirable feature: first, the locality of applications can be increased if the data is replicated at all sites where applications need it, and second, the availability of the system can be increased, because a site failure does not stop the execution of applications at other sites if the data is replicated.
- ***Integrity, recovery, and concurrency control*** In databases the issues of integrity, recovery, and concurrency control, although they refer to different problems, are strongly interrelated. To a large extent, the solution of these problems consists of providing transactions.
- **Privacy *and security*** In traditional databases, the database administrator, having centralized control, can ensure that only authorized access to the data is performed. Note, however, that the centralized database approach in itself, without specialized control procedures, is more vulnerable to privacy and security violations **than** the older approaches based on separate files.

In distributed databases, local administrators are faced essentially with the **same** problem as database administrators in a traditional database. However, **two** peculiar aspects of distributed databases are worth mentioning: first, in a distributed database with a very high degree of site autonomy, the owners of local data feel more protected because they can enforce their own protections instead of depending on a central database administrator; second, security problems are intrinsic to distributed systems in general, because communication networks can represent a weak point with respect to protection.

Long Answer Type Questions

1. Write down the 2-phase commitment protocol. What are different communication structure for this protocol? What is log write-ahead protocol? [MODEL QUESTION]

Answer:

1st Part:

The two-phase commit protocol provides an automatic recovery mechanism in case a system or media failure occurs during execution of the transaction. The two-phase commit protocol ensures that all participating database servers receive and implement the same action (either to commit or to roll back a transaction), regardless of local or network failure.

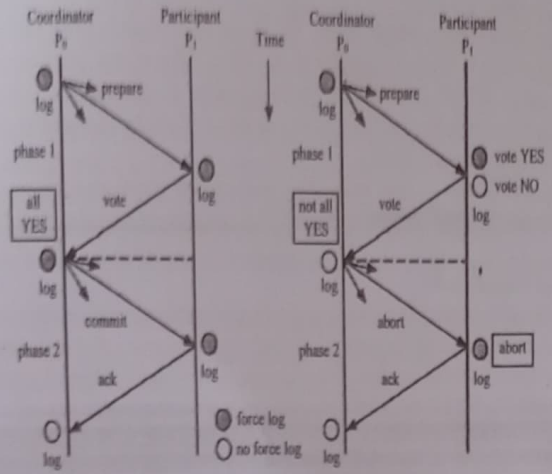
If any database server is unable to commit its portion of the transaction, all database servers participating in the transaction must be prevented from committing their work.

With a two-phase commit protocol, the distributed transaction manager employs a *coordinator* to manage the individual resource managers.

The commit process proceeds as follows:

- ***Phase 1***
  - Each participating resource manager coordinates local operations and forces all log records out:
  - If successful, respond "OK"
  - If unsuccessful, either allow a time-out or respond "OOPS"
- ***Phase 2***
  - If all participants respond "OK":
    - Coordinator instructs participating resource managers to "COMMIT"
    - Participants complete operation writing the log record for the commit
  - Otherwise:
    - Coordinator instructs participating resource managers to "ROLLBACK"
    - Participants complete their respective local undos

In order for the scheme to work reliably, both the coordinator and the participating resource managers independently must be able to guarantee proper completion, including any necessary restart/redo operations. The following figure describes the basic protocol.



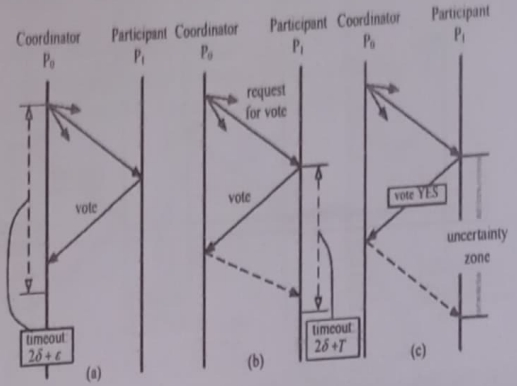
If the coordinator has detected a participant's failure, it must decide abort. If a participant has detected the coordinator's failure it first tries to determine whether a decision has been made, by consulting the other participants. If it fails to get an answer, it starts a protocol to elect a new coordinator.

2nd Part:

When the coordinator or a participant restarts after a failure, it uses the log records to determine the state of the transaction, and to act accordingly. For instance, if it finds itself in phase 1, it restarts the transaction; if it finds itself in phase 2, it sends the logged decision to the participants; if the last log record is the end of the transaction, there is nothing to do.

Note that a participant that voted YES enters an "uncertainty zone" until it has received a decision from the coordinator, or until timeout. This is because another participant may have already made either a commit or an abort decision (none is excluded for the time being). This may lead to a blocking scenario: suppose p_i has voted YES and is in the uncertainty zone; suppose that the coordinator made a decision, sent it to some participants, and then failed; suppose, in addition, that these latter participants also failed. Then a decision has actually been made, but p_i, although being correct, has no means to know it, at least until after the coordinator has been repaired (if the coordinator has logged the decision on stable storage before failing, the decision can be retrieved and resent to the participants).

The protocol assumes that there is stable storage at each node with a ***write-ahead log***, that no node crashes forever, that the data in the write-ahead log is never lost or corrupted in a crash, and that any two nodes can communicate with each other. The last assumption is not too restrictive, as network communication can typically be rerouted. The first two assumptions are much stronger; if a node is totally destroyed then data can be lost. The following figure shows Two-phase commit on failure detection.



2. a) Describe the different failures possible in 2-phase commit protocol. What are the outcomes of these failures? **[MODEL QUESTION]**

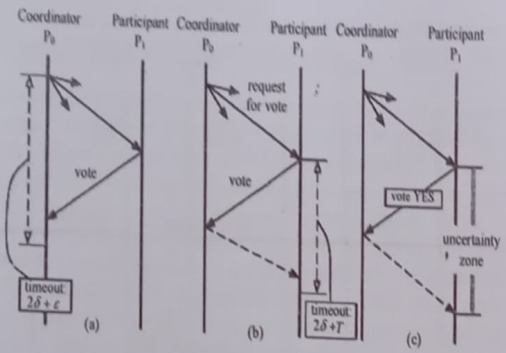
Answer:

Failures and their outcomes in 2-phase commit protocol -

When the coordinator or a participant restarts after a failure, it uses the log records to determine the state of the transaction, and to act accordingly. For instance, if it finds itself in phase 1, it restarts the transaction; if it finds itself in phase 2, it sends the logged decision to the participants; if the last log record is the end of the transaction, there is nothing to do.

Note that a participant that voted YES enters an "uncertainty zone" until it has received a decision from the coordinator, or until timeout. This is because another participant may have already made either a commit or an abort decision (none is excluded for the time being). This may lead to a blocking scenario: suppose p_i has voted YES and is in the uncertainty zone; suppose that the coordinator made a decision, sent it to some participants, and then failed; suppose, in addition, that these latter participants also failed. Then a decision has actually been made, but p_i, although being correct, has no means to know it, at least until after the coordinator has been repaired (if the coordinator has logged the decision on stable storage before failing, the decision can be retrieved and resent to the participants).

The protocol assumes that there is stable storage at each node with a ***write-ahead log***, that no node crashes forever, that the data in the write-ahead log is never lost or corrupted in a crash, and that any two nodes can communicate with each other. The last assumption is not too restrictive, as network communication can typically be rerouted. The first two assumptions are much stronger; if a node is totally destroyed then data can be lost. The following figure shows Two-phase commit on failure detection.



b) What is the difference between reliability and availability? What are the factors affecting the allocation? What is nested transaction? **[MODEL QUESTION]**

Answer:

1st Part:

Distributed reliability protocols maintain the atomicity and durability properties by

(i) ensuring that either all of the actions of a transaction that is executing at different sites complete successfully (called commit) or none of them complete successfully (called abort), and

(ii) ensuring that the modifications made to the database by committed transactions survive failures. The first requires atomic commit protocols (see Commit Protocols), while the second requires recovery protocols

Availability is simply the time an item is able to operate (operating time+standby time) as a ratio to the time in service. It tells us how much the cash machine can yield if we feed it with the required inputs e.g. raw material, power etc. and remove the product as it is made. It can be defined as the percentage of time over a well-defined period that a system or service is available for users. So, for example, if a system is said to have 99.999%, or five nines, availability, this system must not be unavailable more than five minutes over the course of a year.

2nd Part:

The factors affecting allocation:

The data allocation problem for object databases involves allocation of both methods and classes. The method allocation problem is tightly coupled to the class allocation problem. Since applications on object-oriented databases invoke methods, the application of methods affects the performance of applications. However, allocation of methods hat need to access multiple classes at different sites is a problem that has been not yet been tackled. Four alternatives can be identified.

1. ***Local behavior – local object.*** This is the most straightforward case and is included to form the baseline case. The behavior, the object to which it is to be applied, and

the arguments are all co-related. Therefore, no special mechanism is need to handle this case.

2. ***Local behavior – remote object.*** This is one of the cases in which the behavior and the object to which it is applied are located at different sites. There are two ways of dealing with this case. One alternative is to move the remote object to the site where the behavior is located. The second is to ship the behavior implementation to the site where the object is located.
3. ***Remote behavior – local object.*** This case is the reverse of case (2).
4. ***Remote function – remote argument.*** This case is the reverse of case (1).

Nested Transaction:

An alternative transaction model is to permit a transaction to include other transactions with their own begin and commit points. Such transactions are called ***nested transaction***. These transactions that are embedded in another one are usually called sub-transactions.

Example:

```
Begin_transaction Reservation
 begin
 Begin_transaction Airline
  ...
 end. {Airline}
 Begin_transaction Hotel
  ...
 end. {Hotel}
 Begin_transaction Car
  ...
 end. {Car}
end.
```

Nested transactions have received considerable interest as a more generalized transaction concept. The level of nesting is generally open, allowing subtractions themselves to have nested transactions. This generality is necessary to support application areas where transactions are more complex than in traditional data processing.

3. Describe different types of failure in DDBMS. **[MODEL QUESTION]**

Answer:

Communication Failures in Distributed Databases

Recovery mechanisms for distributed transactions require understanding also the failures which may occur in the communications between sites. In this section, we classify the main types of communication failures, and state some basic assumptions on the communication network which are relevant in this chapter.

When a message is sent from site *X* to site *Y*, we require from a communication network the following behavior:

1. *X* receives a positive acknowledgment after a delay which is less then some maximum delay DMAX,

2. The message is delivered at *Y* in proper sequence with respect to other X-Y messages.

3. The message is correct.

There is a great variety of possible failures with respect to the above specification; for example, the message might not be correct, the message might be out of order, *X* might not receive the acknowledgment with the message being delivered, *X* might receive the acknowledgment without the message being delivered, and. so on. Most communication networks are capable of eliminating most of these errors, so that we can assume that:

1. If a message from *X* to *Y* is delivered at *Y*, then the message is correct and is in sequence with respect to other *X*-*Y* messages.
2. If X receives an acknowledgment, then the message has been delivered.

There are two basic types of possible communication errors; lost messages and partitions.

When a site *X* does not receive an acknowledgment of a message from a site *Y* within a predefined time interval, X is uncertain about the following things;

1. Did a failure, occur at all, or is the system simply slow?
2. if a failure occurred, was it a communication failure, or a crash of site *Y*?
3. Has the message been delivered at *Y* or not?

The answer to question 3 does not depend on the answer to question 2, because either the communication failure or the crash can have happened before or after the delivery of the message.

Therefore, several levels of reliability algorithms can be designed, which are capable of dealing with the following failures, in order of increasing difficulty:

Class 1. Site failures only.
Class 2. Site failures and lost messages, but no partitions.
Class 3. Site failures, lost messages, and partitions.

The algorithms of class 1 consider the communication network completely reliable and deal with site failures under this assumption. Hence, if a communication failure occurs, the system behaves in an erroneous way. We will call ***catastrophic failures*** for a reliability mechanism all the failures which cannot be dealt with by it. A network partition is a catastrophic failure for the algorithms of classes 1 and 2.

Multiple failures and K-resiliency:

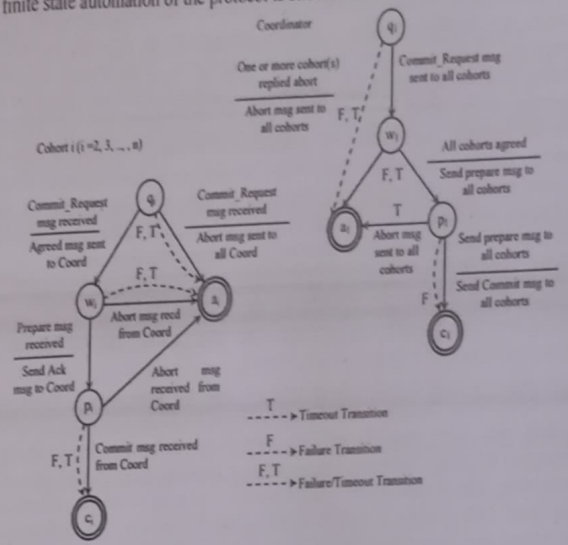
Unfortunately failures do not occur one at a time. A system which can tolerate *K* failures is called K-resilient. In distributed databases, this concept is applied to site failures and/or partitions.

With respect to site failures, an algorithm is said to be K-resilient if it works properly even if *K* sites are down. With respect to partitions, K-resiliency refers to the number of sub partitions which have been generated. An extreme case of multiple failure is a **total failure**, where all sites are down. Many algorithms do not deal with this case, as it is considered extremely rare.

4. a) Briefly discuss 3-phase commitment protocol.
b) Will 3 PC work in case of partition (type of failure) of network? If not, discuss an algorithm that works in case of partition. **[MODEL QUESTION]**

Answer:

a) Coordinator -receives a transaction request.
Cohort- receives a can Commit? Message from the coordinator.
The finite state automation of the protocol is shown below.



Assumptions

- each site uses the write-ahead-log protocol
- atmost one site can fail during the execution of the transaction

Basic Idea is that before the commit protocol begins, all the sites are in state q. If the coordinator fails while in state q1, all the **cohorts** perform the **timeout transition**, thus aborting the transition. Upon recovery, the **coordinator** performs the **failure transition.**

Phase I

During the first phase, the coordinator is in state w1, and each cohort is either in state a (in which case the site has already sent an abort message to the coordinator) or w or q depending on whether it has received the Commit_Request message or not. If a cohort fails, the coordinator times out waiting for the agreed message from the failed cohort. In this case, the coordinator aborts the transaction and sends abort messages to all the cohorts.

Phase II

In the second phase, the coordinator sends a Prepare message to all the cohorts if all the cohorts have sent Agreed messages in Phase I. Otherwise, the coordinator will send an Abort message to all the cohorts. On receiving a Prepare message, a cohort sends an acknowledge message to the coordinator. If the coordinator fails before sending Prepare

messages (i.e., in state w1), it aborts the transaction upon recovery, according to the failure transition. The cohorts time out waiting for the prepare message, and also abort the transaction as per the timeout transition.

Phase III

In the third phase, on receiving acknowledgements to the Prepare messages from all the cohorts, the coordinator sends a Commit message to all the cohorts. A cohort, on receiving a Commit message, commits the transaction. If the coordinator fails before sending the commit message (i.e., in state p1), it commits the transaction upon recovery, according to the failure transition from state p1. The cohorts time out waiting for the Commit message. They commit the transaction according to the timeout transition from state pi. However, if a cohort fails before sending an acknowledgement message to a Prepare message, the coordinator times-out in state p1. The coordinator aborts the transaction and sends Abort messages to all the cohorts. The failed cohort, upon recovery, will abort the transaction according to the failure transaction from state wi.

3 phase commit is **non-blocking** as it places an upper bound on the amount of time required before a transaction either commits or aborts. This property ensures that if a given transaction is attempting to commit via this protocol and holds some resource locks, it will release the locks after the timeout.

b) In a distributed database system, a transaction could be executing its operations at multiple sites. Since atomicity requires every distributed transaction to be atomic, the transaction must have the same fate (commit or abort) at every site. In case of network partitioning, sites are partitioned and the partitions may not be able to communicate with each other. This is where a quorum-based technique comes in. The fundamental idea is that a transaction is executed if the majority of sites vote to execute it.

Every site in the system is assigned a vote Vi. Let us assume that the total number of votes in the system is V, and the abort and commit quorums are Va and Vc, respectively.

Algorithm for Read Quorum Construction:

***Function Read Quorum (trees)*: QUORUM;**

```
var NPSQuorum, children: QUORUM;
var node: NODE;
begin
node = root of the tree;
if Empty (trees) then
return ({ });
else if all nodes of NPS (node) are accessible then
return (each node of NPS(node));
else
begin
children = children of node;
for each node ε children
NPSQuorum = ReadQuorum(child Subtrees);
If all nodes of NPS(node) are inaccessible then
```



```
return ({ });
else
return (NPSQuorum);
end
end;
```

Algorithm for Write Quorum Construction:

```
Function Write Quorum(Trees): QUORUM;
var NPSQuorum, children: QUORUM;
var node: NODE;
begin
node = root of the tree;
if Empty (Trees) then
return ({ });
else if all nodes of NPS (node) is already taken then
for each child ε children
NPSQuorum = NPSQuorum U
WriteQuorum = (child subtree);
else if all nodes of NPS (node) are accessible then
begin
NPSQuorum = NPSQuorum U
(any node among node and is siblings);
children = children of node;
for each child ε children
NPSQuorum = NPSQuorum U
WriteQuorum = (child subtree);
If unable to take any node from NPS(node)
return ({ });
else
return (NPSQuorum);
end
else
return ({ });
end;
```

5. What are the uses of catalogs in DDBMS? What are the contents of catalog? How catalogs are allocated to different site in DBMS?

OR,

Describe how you will manage the storage and allocation of a global system catalogue in a distributed DBMS. **[MODEL QUESTION]**

Answer:

a) The Global System Catalog holds information specific to the distributed nature of the system, such as the fragmentation and allocation schemas. A site catalog describes all objects (fragments, replicas) at a site and keeps track of replicas of relations created at this site.

Contents of catalog: It stores not only data regarding relations, indexes, users, etc, but also all the necessary control information for independence in the following manner:

1. **Centralized:** violate no reliance on a central site.
2. **Fully replicated:** loss of local autonomy.
3. **Partitioned:** nonlocal operations are very expensive.
4. **Combination of (1) and (3):** violate no reliance on a central site.

The distributed database catalog entries must specify site(s) at which data is being stored in addition to data in a system catalog in a centralised DBMS. Because of data partitioning and replication, this extra information is needed. There are a number of approaches to implementing a distributed database catalog.

- Centralised - Keep one master copy of the catalog
- Fully replicated - Keep one copy of the catalog at each site
- Partitioned - Partition and replicate the catalog as usage patterns demand
- Centralised/partitioned - Combination of the above

6. Justify the following statements: **[MODEL QUESTION]**
3-phase commitment protocol overcome limitation of 2-phase commitment protocol.

Answer:

3-phase commitment protocol overcome limitation of 2-phase commitment protocol:
A Two-phase commit protocol cannot dependably recover from a failure of both the coordinator and a cohort member during the Commit phase. If only the coordinator had failed, and no cohort members had received a commit message, it could safely be inferred that no commit had happened. If, however, both the coordinator and a cohort member failed, it is possible that the failed cohort member was the first to be notified, and had actually done the commit. Even if a new coordinator is selected, it cannot confidently proceed with the operation until it has received an agreement from all cohort members ... and hence must block until all cohort members respond.
The Three-phase commit protocol eliminates this problem by introducing the Prepared to commit state. If the coordinator fails before sending preCommit messages, the cohort will unanimously agree that the operation was aborted. The coordinator will not send out a doCommit message until all cohort members have ACKed that they are Prepared to commit. This eliminates the possibility that any cohort member actually completed the transaction before all cohort members were aware of the decision to do so (an ambiguity that necessitated indefinite blocking in the Two-phase commit protocol).

7. a) Explain how network partitioning is handled in replicated databases. **[MODEL QUESTION]**

Answer:
Network partitions separate replica managers into two or more subgroups, in such a way that the members of a subgroup can communicate with one another but members of different subgroups cannot communicate. There are two approaches for this – 1) Optimistic approaches use available copies with validation and 2) Pessimistic approaches which use Quorum consensus.

b) What is allocation? **[MODEL QUESTION]**

Answer:
Data allocation describes the process of deciding where to locate data. Data allocation strategies are as follows:

- With centralized data allocation, the entire database is stored at one site.
- With partitioned data allocation, the database is divided into two or more disjointed parts (fragments) and stored at two or more sites.
- With replicated data allocation, copies of one or more database fragments are stored at several sites.

c) Write the horizontal Algorithm and explain it with proper example. **[MODEL QUESTION]**

Answer:
Primary horizontal fragments are defined using selections on global relations; the correctness of primary fragmentation requires that each tuple of the global relation be selected in one and only one fragment. Thus, determining the primary fragmentation of a global relation requires determining a set of disjoint and complete selection predicates. The property that we require for each fragment is that the elements of them must be referenced homogeneously by all the applications.
The horizontal fragmentation function distributes a relation based on selection predicates. The following example is used in subsequent discussions.

EMP

ENO	ENAME	TITLE
E1	J. Doe	Elec. Eng
E2	M. Smith	Syst. Anal.
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E5	B. Casey	Syst. Anal.
E6	L. Chu	Elect. Eng.
E7	R. Davis	Mech. Eng.
E8	J. Jones	Syst. Anal.

Relation EMP(ENO, ENAME, TITLE) of the above table can be split into three horizontal fragments EMP_1, EMP_2, and EMP_3, defined as follows:

$$EMP_1 = \sigma_{ENO \le "E3"}(EMP)$$

$$EMP_2 = \sigma_{"E3" < ENO \le "E6"}(EMP)$$

$$EMP_3 = \sigma_{ENO > "E6"}(EMP)$$

The localization program for an horizontally fragmented relation is the union of the fragments. In our example we have

$$EMP = EMP_1 \cup EMP_2 \cup EMP_3$$

Thus the localized form of any query specified on EMP is obtained by replacing it by $EMP_1 \cup EMP_2 \cup EMP_3$.

8. Write shorts notes on the following: **[MODEL QUESTION]**
a) Network partitioning
b) Site failure
c) Wait-Die and Wound-Wait algorithm
Answer:

a) Network partitioning:

One of the most difficult failures in a distributed system is ***network partitioning*** (when a set of computers are networked together but they are isolated from another set of computers that are interconnected). When network partitioning happens, there is a danger that computers in each partition unilaterally decide to commit or abort the transaction. If the two partitions do not decide on the same action (either both committing or both aborting the transaction), the integrity of the database will be lost. Therefore, we also examine a Quorum-based commit protocol that deals with network partitioning.

Network partitioning can completely destroy mutual consistency in the worst case, and this fact has led to a certain amount of restrictiveness, vagueness, and even nervousness in past discussions, of how it may be handled. In some environments it is desirable or necessary to permit users to continue modifying resources such as files when the network is partitioned. A network operating system would be a good example. In such environments mutual inconsistency becomes a fact of life which must be dealt with. Once inconsistency is detected, some reconciliation steps are needed. In those cases where the semantics of the operations involved are straightforward, automatic reconciliation may be possible.

b) Site Failure:

From the viewpoint of local recovery, the most important characteristic of failures is the amount information which is lost because of the failure. Therefore, in the classification of failures we consider essentially the memory component. From this viewpoint failures can be classified as follows:

1. ***Failures without loss of information:*** In these failures, all the information stored in memory is available for the recovery. These failures include, for example, the abort of transactions because an error condition is discovered like an arithmetic overflow or a division by zero.
2. ***Failures with loss of volatile storage:*** In these failures, the content of main memory is lost; however, all the information which is recorded on disks is not affected by the failure. Typical failures of this kind are system crashes,
3. ***Failures with loss of nonvolatile storage:*** In these failures, called media **failures,** also the content of disk storage is lost. Typical failures of this kind are head crashes. The probability of failures of the third type is less than that of the other two types. Moreover, it is possible to make the probability of the failures of the third type arbitrarily small by replicating the information on several disks having **independent failure modes.** Two disks have independent failure modes if the probability of failure of one of them does not depend on the operational status of the other one.
   This idea is the basis for the development of **stable storage.** Stable storage is the most resilient storage medium available in the system. Stable storage is typically

implemented by replicating the same information on several disks with independent failure modes and using the so-called **careful replacement** strategy: at every update operation, first one copy of the information is updated, then the correctness of the update is verified, and finally the second copy is updated.

Having introduced stable storage, we can introduce a new type of failure:

4. ***Failures with loss of stable storage:*** In these failures, some information stored in stable storage is lost because of several, simultaneous failures of the third type. Although we can make the probability of failures of stable storage arbitrarily small by using more replication and careful replacement strategies, it is impossible to reduce this probability to zero.

c) Wait-Die and Wound-Wait algorithm:

Wait/Die and Wound/Wait are deadlock avoidance algorithms which use a symmetry-breaking technique. In both these algorithms there exists an older process (O) and a younger process (Y). Process age can be determined by a timestamp at process creation time. Smaller timestamps are older processes, while larger timestamps represent younger processes.

The Wait-Die algorithm:

- Allow wait only if waiting process is older.

Since timestamps increase in any chain of waiting processes, cycles are impossible. When the younger process restarts and requests the resource again, it may be killed once more.

This is the less efficient of these two algorithms.

The Wound-Wait algorithm:

- Otherwise allow wait only if waiting process is younger.

Here timestamps decrease in any chain of waiting process, so cycles are again impossible. It is wiser to give older processes priority. When the younger process re-requests resource, it has to wait for older process to finish. This is the better of the two algorithms.

PARALLEL DATABASE SYSTEMS

Short Answer Type Questions

1. Explain: Parallel Query Processing. **[MODEL QUESTION]**

Answer:

The steps for Parallel Query Processing are:

1. **Translation:** Conversion of the relational algebra expression to a query tree.
2. **Optimisation:** Reordering of join operations in the query tree and choose among different join algorithms to minimise the cost of the execution.
3. **Parallelisation:** Transforming the query tree to a physical operator tree and loading the plan to the processors.
4. **Execution:** Running the concurrent transactions.

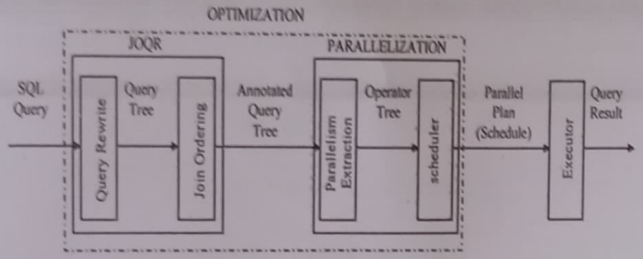


Fig: Parallel query processing

2. What are the difference between distributed system and parallel system? **[MODEL QUESTION]**

Answer:

Features	Parallel Systems	Distributed Systems
i) Memory	Tightly coupled system shared memory	Weakly coupled system Distributed memory
ii) Control	Global clock control	No global clock control
iii) Processor Interconnection	Order of Tbps	Order of Gbps
iv) Main focus	Performance Scientific computing	Performance (cost & scalability) Reliability/availability Information/resource & sharing.

3. Write down the advantages and disadvantages of parallel system. **[MODEL QUESTION]**

Answer:

The advantages of the Parallel systems are:

- Provide Concurrency (do multiple things at the same time)

- Taking advantage of non-local resources
- Cost Saving
- Overcoming memory constraints
- Save time and money
- Global address space provides a user-friendly programming perspective to memory.

The Disadvantages of the Parallel systems are:

- Primary disadvantage is the lack of scalability between memory and CPUs.
- Programmer responsibility for synchronization constructs that ensure "correct" access of global memory.
- It becomes increasingly difficult and expensive to design and produce shared memory machines with ever increasing numbers of processors.

Long Answer Type Questions

1. a) Describe scale up ad speedup factors in parallel databases.
b) Define different types of data partitioning techniques in parallel databases with example. **[MODEL QUESTION]**

Answer:

Following are the major factors which affect the efficient parallel operation and can reduce both speedup and scaleup.

Startup costs

For every single operation which is to be done in parallel, there is an associated cost involved in starting the process. That is, breaking a given big process into many small processes consumes some amount of time or resources. For example, a query which tries to sort the data of table need to partition the table as well as instructing various parallel processors to execute the query before any parallel operation begins.

Example: **SELECT * FROM Emp ORDER BY Salary;**

This query sorts the records of Emp table on Salary attribute. Let us assume that there are 1 million employees, so one million records. It will consume some time to execute in a computer with minimal resources. If we would like to execute the query in parallel, we need to distribute (or redistribute) the data into several processors (in case of Shared Nothing Architecture), also, we need to instruct all the processors to execute the query. These two operations need some amount of time well before any parallel execution happens.

Interference

Since processes executing in a parallel system often access shared resources, a slowdown may result from the interference of each new process as it competes with existing processes for commonly held resources, such as a system bus, or shared disks, or even locks. Both speedup and scaleup are affected by this phenomenon.

Skew

When breaking down a single task into number of parallel small tasks, it is very hard to make them equal in size. Hence, the performance of the system depends on the slowest CPU which processes the larger sub-task. This type of uneven distribution of a job is called skew. For example, if a task of size 100 is divided into 10 parts, and the division is skewed, there may be some tasks of size less than 10 and some tasks of size more than 10; if even one task happens to be of size 20, the speedup obtained by running the tasks in parallel is only five, instead of ten as we would have hoped.

b) Different types of data partitioning techniques in parallel databases

i) Round Robin: The simplest portioning strategy distributes tuples among the fragments in a round-robin fashion. This is the partitioned version of the classic entry-sequence file. Assume that there are n disks, $D_1, D_2, \cdots, D_n$, among which the data are to be partitioned. This strategy scans the relation in some order and sends the ith tuple to disk number $D_{i \bmod n}$. The round-robin scheme ensures an even distribution of tuples across disks.

Round-robin partitioning is excellent for applications that wish to read the entire relation sequentially for each query. The problem with round-robin partitioning is that it is very difficult to process point queries and range queries.

ii) Hash partitioning: This strategy choose one or more attributes from the given relational schema as the partitioning attributes. A hash function is chosen whose range is within 0 to $n-1$, if there are n disks. Each tuple of the original relation is hashed based on the partitioning attributes. If the has function returns i then the tuple is placed on disk D_i

Hash Partitioning is ideally suited for applications that want only sequential and associative accesses to the data. Tuples are stored by applying a hashing function to an attribute. The hash function specifies the placement of the tuple on a particular disk.

ii) Range Partitioning: Range portioning clusters together tuples with similar attribute values in the same partition. This strategy distributes contiguous attribute-value ranges to each disk. It chooses a partitioning attribute as a partitioning vector. Using range-partitioning technique, a relation is partitioned in the following way. Let $\{v_0, v_1, \cdots v_{n-2}\}$ denote the partition vector of the partitioning attribute such that if $i < j$, then $v_i < v_j$. Consider a tuple t such that $t[A] = x$. If $x < v_0$ then t goes on disk D_0 and if $x > v_{n-2}$, then t goes on disk D_{n-1}. If $v_i \le x < v_{i+1}$, then t goes on disk D_{i+1}.

Range portioning is good for sequential and associative access, and is also good for clustering date.

ADVANCED TOPICS

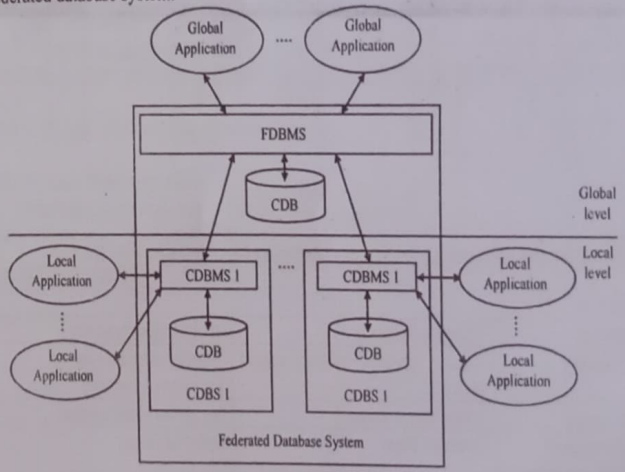
Short Answer Type Questions

1. What do you mean by federated database? What are loosely and tightly coupled systems in reference with distributed database management system? [MODEL QUESTION]

Answer:

1st Part:

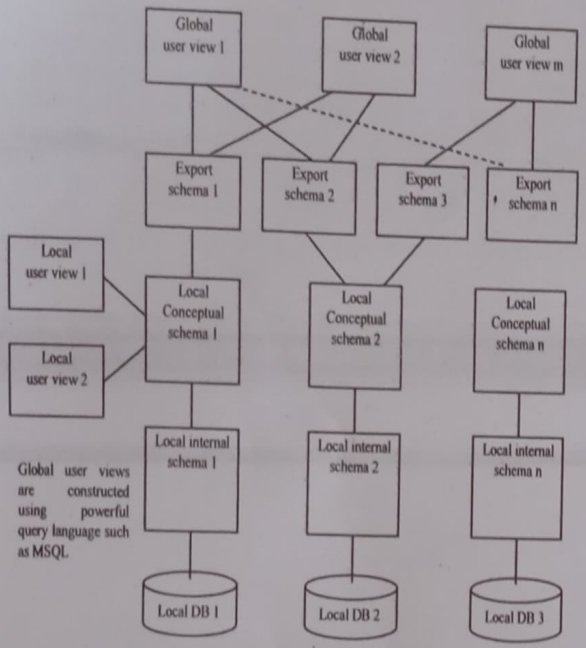
The purpose of federated database systems is to provide means to transparently access and modify data dispersed over heterogeneous database systems in a uniform manner. A **federated database system** is a type of meta-database management system (DBMS). The constituent databases are interconnected via computer network, and may be geographically decentralized. Since the constituent database systems remain autonomous, a federated database system is a contrastable alternative to the (sometimes daunting) task of merging together several disparate databases. A federated database, or **virtual database**, is the fully integrated, logical composite of all constituent databases in a federated database system.



2nd Part:

A Tightly coupled system has a global schema and translation is between local and global schema. This means that more work has to happen on the database administrator and less work on the user: query is written based on a global schema.

A loosely coupled system is shown below:



Export schema describes data a database site is willing to share

In Loosely-coupled system does not have a global schema and translation is between external **schemas** and local conceptual schemas. This means that there is more work on users.

2. What is Distributed Object Architecture? **[MODEL QUESTION]**

Answer:

Distributed object systems are the foundation for modern three-tier architectures. In a three-tier architecture, as shown in the figure, the presentation logic resides on the client (first tier), the business logic resides on the middle tier (second tier), and other resources, such as the database, reside on the backend (third tier).

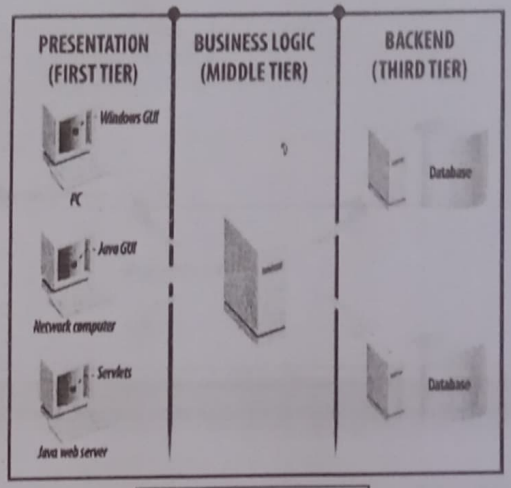


Fig: Three Tier Architecture

Long Answer Type Questions

1. Explain the Term: Distributed data management. **[MODEL QUESTION]**

Answer:

A distributed database is a collection of multiple, logically interrelated databases distributed over a computer network. A distributed DBMS is then defined as the software system that permits the management of the distributed database and makes the distribution transparent to the users.

This definition is based on the following implicit assumptions:

1. Data is stored at a number of sites. Each site is assumed to logically consist of a single processor.
2. The processors at these sites are interconnected by a computer network rather than a multiprocessor configuration.

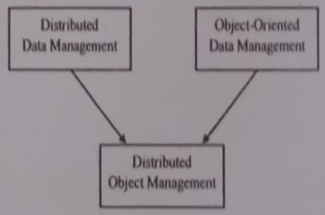


Fig. 1: Technologies Contributing to DOM

The two assumptions make the distinction between distributed DBMS and parallel DBMS.

Distributed DBMSs provide transparent access to a physically distributed database. User queries and transactions access the database based on external schemas (ES) defined over a global conceptual schema (GCS) which is partitioned and stored at different sites forming the local conceptual schemas (LCS) at those sites. The LCS at a given site is then mapped to a local internal schema (LIS) which describes the physical organization of data at that site.

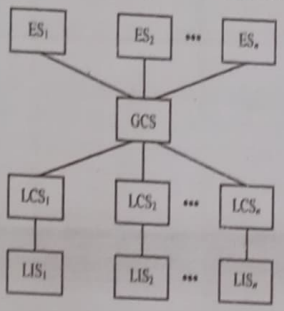


Fig. 2: Datalogical Distributed DBMS Architecture

2. Describe the following: **[MODEL QUESTION]**
a) Federated Database
b) OO4O

Answer:

a) A federated database system is a type of meta-database management system (DBMS), which transparently maps multiple autonomous database systems into a single federated database. The constituent databases are interconnected via a computer network and may be geographically decentralized. Since the constituent database systems remain autonomous, a federated database system is a contrastable alternative to the (sometimes daunting) task of merging several disparate databases. A federated database, or virtual database, is a composite of all constituent databases in a federated database system. There is no actual data integration in the constituent disparate databases as a result of data federation.

Through data abstraction, federated database systems can provide a uniform user interface, enabling users and clients to store & retrieve data from multiple noncontiguous databases with a single query -- even if the constituent databases are heterogeneous. To this end, a federated database system must be able to decompose the query into subqueries for submission to the relevant constituent DBMS's, after which the system must composite the result sets of the subqueries. Because various database management systems employ different query languages, federated database systems can apply wrappers to the subqueries to translate them into the appropriate query languages.

b) Oracle Objects for OLE (OO4O) is a COM-based data access driver that combines seamless and optimized access to Oracle databases with easy to use interfaces. OO4O can

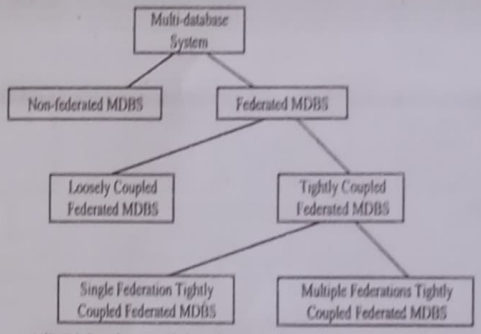
be used in a variety of environments ranging from web applications to n-tier client/server applications. It can be used from virtually any programming or scripting language that supports the Microsoft COM Automation technology, such as Visual Basic, Visual C++, VBA in Excel, Active Server Pages, PowerBuilder, Delphi, Microsoft Internet Information Services, and Microsoft Distributed Transaction Coordinator.

Because it is a native driver, OO4O generally provides the fastest performance on Windows clients to Oracle databases. It does not incur the overhead of ODBC and OLE DB drivers. OO4O has been developed and has evolved specifically for use with Oracle database servers. It provides easy access to features that are unique to Oracle, but are otherwise cumbersome or inaccessible to use from ODBC and OLE DB-based components, such as ADO.

3. Write sort notes on the following: **[MODEL QUESTION]**
a) MDBS
b) Peer-to-Peer Architecture
c) Distributed database administration

Answer:

a) MDBS:



In recent years, MDBS has been gaining the attention of many researchers who attempt to logically integrate several different independent DDBMSs while allowing the local DBMSs to maintain complete control of their operations. Complete autonomy means that there can be no software modifications to the local DBMSs in a DDBMS. Hence, an MDMS, which provides the necessary functionality, is introduced as an additional software layer on top of the local DBMSs.

An MDBS is a software that can be manipulated and accessed through a single manipulation language with a single common data model in a heterogeneous environment without interfering with the normal execution of the individual database systems. The MDBS has developed from a requirement to manage and retrieve data from multiple databases within a single application while providing complete autonomy to individual database systems. To support DBMS transparency, MDBS resides on top of existing databases and file systems and presents a single database to its users, An MDBS maintains a global schema against which user issue queries and updates, and this global schema is constructed by integrating the schemas of local databases.