

Distributed Databases

Unit-I Distributed Database

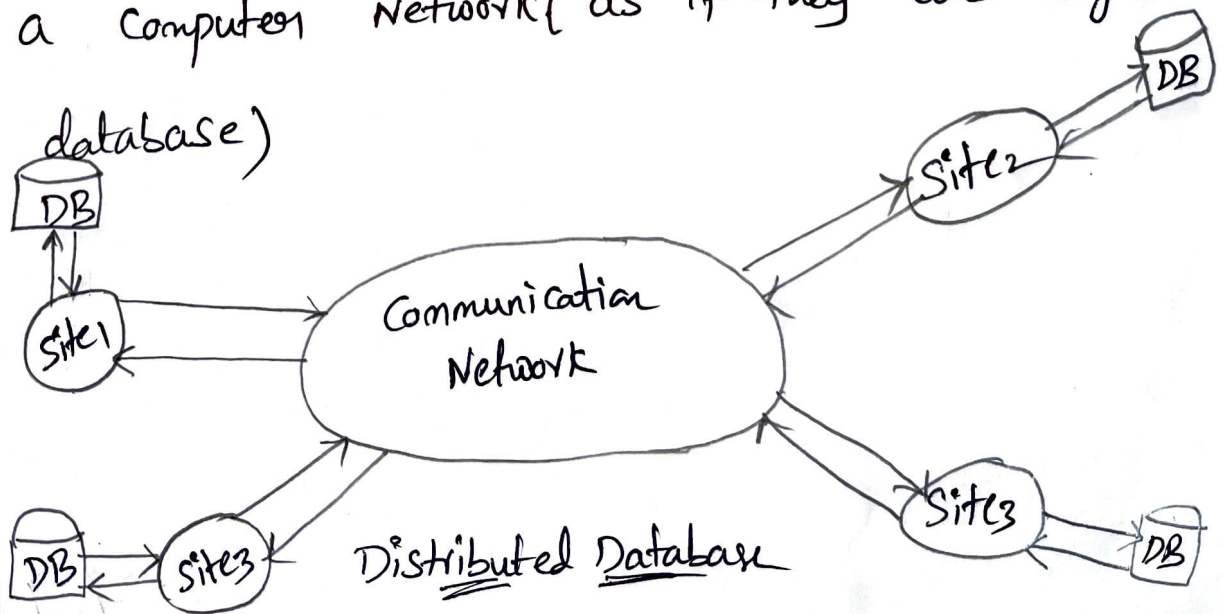
Objective: Distributed databases are used for horizontal scaling, and they are designed to meet the workload requirements without having to make changes in the database application.

Introduction:

A distributed database is a combination of two contrasting technologies

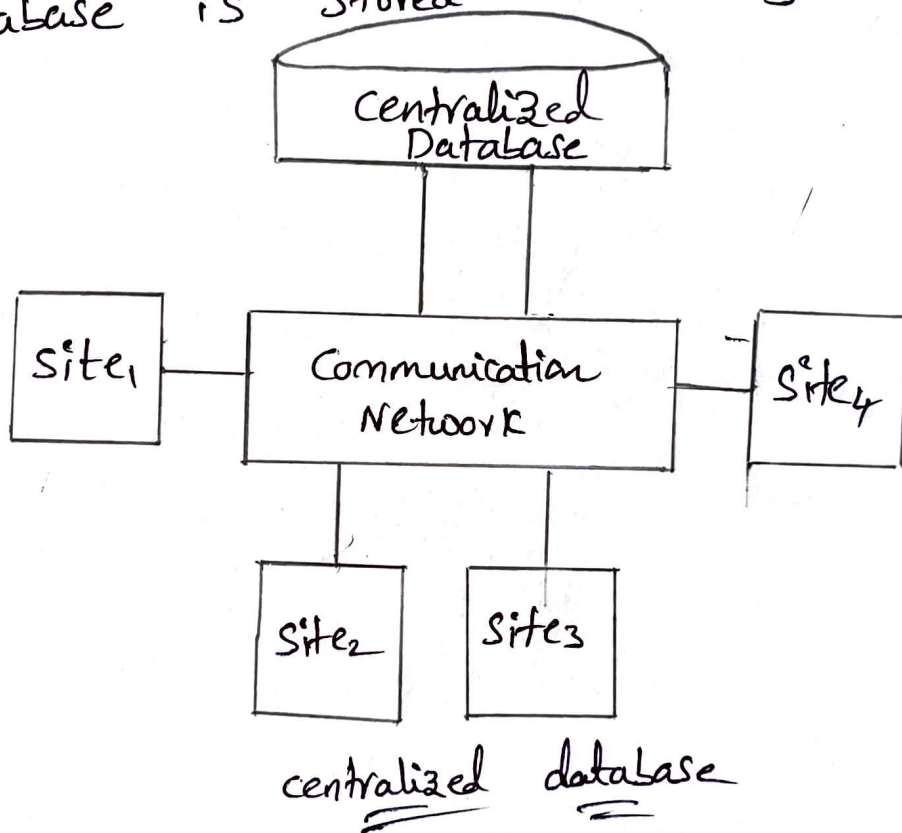
- 1) Database System
- 2) Computer Network

A distributed database is a collection of multiple interrelated databases distributed over a computer network (as if they are single database)



The diagram is an example of distributed database. In which communication channel is used to communicate with the different locations and every system has its own database.

centralized database: In centralized DBMS the database is stored in a only one site.



Advantages of Distributed Database

Modular Development: -

If the system needs to be expanded to new locations or new units, in centralized database systems, the action requires substantial efforts and disruption in the functioning.

However, in distributed databases the work simply requires adding new computers and local data to the new site and finally connected them to the distributed system, with no interruption in current functions.

More Reliable: In case of database failures, the total system of centralized databases comes to halt. However, in distributed system when a component fails the functioning of the system continues may be at a reduced performance. Hence DDBMS is more reliable.

Better Response: If data is distributed in an efficient manner, the user requests can be met from local data itself, thus providing faster response. On the other hand, in centralized systems all queries have to pass through the central computer for processing, which increases the response time.

Lower Communication cost: - In distributed database systems, if data is stored or located locally where it is mostly used, then the communication costs for data manipulation can be minimized. This is not feasible in centralized systems.

Disadvantages of Distributed Databases

Need for Complex and Expensive Software:

DDBMS demands complex and often expensive software to provide data transparency and co-ordination across the several sites.

Processing overhead: - Even simple operations may require a large no. of communications and additional calculations to provide uniformity in data across the sites.

Data Integrity: The need for updating data in multiple sites pose problems of data integrity.

Overheads for improper data distribution:

Responsiveness of queries is largely dependent upon proper data distribution. often leads to very slow response to user requests.

Answer the following:

- 1) What is distributed database?
- 2) Distinguish between centralized and distributed databases?
- 3) What are the advantages and disadvantages of distributed databases?

Distributed Data Processing:

Processing of data that is done online by different interconnected computers is known as distributed data processing.

Objective:

By breaking down complex tasks into smaller subtasks distributed across nodes, the system can process data more quickly and efficiently.

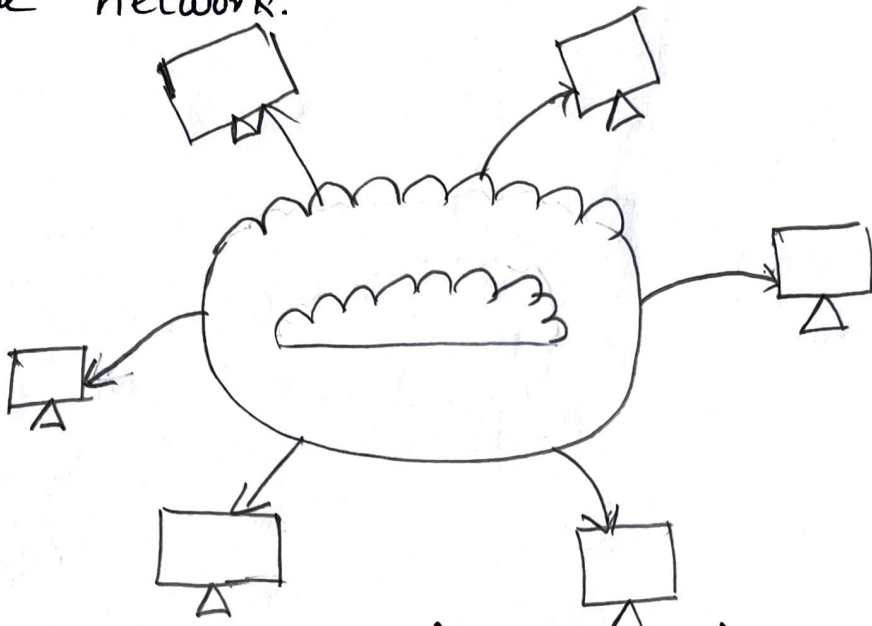
Introduction:

We host our website on the online server nowadays cluster hosting is also available in which website data is stored in different clusters (remote computers).

When a visitor comes to the website then website pages are loaded from the server which is near to the user. Google also uses distributed data processing.

Google has database servers in all major countries. When computer user visits Google site from China then google website is loaded from China server.

In Distributed Processing, there exists one main server which controls all other computers in the network.



Distributed Data Processing

Distributed Data processing is done with high internet speed like querying the databases.

Advantages :-

In Expensive :- Some companies buy a mainframe and super computers to do large scale processing online but it cost those a hundred thousand dollars.

Buying mainframe and supercomputers tend to centralized processing if that computer mal-function then all company data get into risk.
on the other hand doing processing by connecting personal computers from different

locations can save money because they cost them a thousand bucks, data is also distributed so adding and removing (nodes) is easy.

Easy to replace remote computers:-

Microsoft Windows Server has a feature called fail over clustering that helps to remove faulty computers if any computers on the network fails or corrupted by some means then that computer is automatically replaced by other computers.

Optimized Processing:-

Managing data on online server solves slow processing on the personal computer, we can do extra tasks also, doing extra tasks consumes processor power. But the online computer is dedicated one type of processing and it is more likely to increase processing powers. Database server can only handle database queries and file server stores files. So data processing is optimized.

Easy to Expand:-

Suppose your company needs more data processing than expected you can easily attach more

computers to the distributed network.

Parallel Processing:-

Adding and removing computers from the network cannot disturb flow. All data from different computers are processed in parallel processing means data is updated at the same time from all nodes.

Better Performance:-

The overall performance of the company gets better and data is filtered and processed more rapidly in the distributed environment.

Backup of data:-

Data can be backup from any computer connected to the network so the user can backup data at a different time and work with that data locally and then upload the data to the Server.

Local Data Synchronization:-

All the computers on the network can have local storage of important data. Suppose there are different office branches interconnected to each other. All branch computers have a local copy of data. office users edit and update and upload to the main server.

So, the data is synched and available to all computers. Working locally with data is easy and fast and when the user thinks that his work is complete then at the end of the day he can sync that data with the main server.

Data Recovery:-

If some data like the database is a loss in any computer then it can be recovered by another interconnected computer i.e. main database server.

Disadvantages:-

Complexity:-

Computers attached in Distributed data processing are difficult to trouble shoot, design and administer.

Planning data Synchronization is difficult:-

Doing the correct synchronization of data is difficult to develop. Sometimes data is updated in wrong order, so administrators have to keep the focus on it before making a distributed n/w.

Data Security:- If the unauthorized computer is connected to distributed n/w then it can affect other computer performance and data can be a loss also.

Distributed Database System

A distributed database is basically a database that is not limited to one system, it is spread over different sites, i.e., on multiple computers or over a network of computers.

A distributed database system is located on various sites that don't share physical components. This may be required when a particular database needs to be accessed by various users globally, it needs to be managed such that for the users it looks like one single database.

Types:-

1) Homogeneous Database:-

In a homogeneous database, all different sites store database identically. The operating system, database management system, and the data structure used.

All are the same at all sites. Hence they are easy to manage.

2) Heterogeneous Database:-

In a heterogeneous distributed database different sites can use different schema and software that can lead to problems in query processing transactions, also a particular site might be completely unaware of the other sites.

Different computers may use a different database application. They may even use different data models for the database. Hence translations are required for different sites to communicate.

Distributed data Storage:-

There are 2 types in which data can be stored on different sites. These are

1) Replication:-

In this approach, the entire relationship is stored redundantly at 2 or more sites. If the entire database is available at all sites, it is a fully redundant database.

Hence in replication systems maintain copies of data. This is advantageous as it increases the availability of data at different sites. Also now query requests can be processed in parallel.

However, it has certain disadvantages as well. Data needs to be constantly updated any change made at one site needs to be recorded at every site that relation is stored or else it may lead to inconsistency. This is a lot of overhead. Also concurrency control becomes way more complex as concurrent access now needs to be checked over a no. of sites.

2) Fragmentation:-

In this approach, the relations are fragmented (i.e., they are divided into smaller parts) and each of the fragments is stored in different sites where they are required. It must be made sure that the fragments are such that they can be used to reconstruct the original (i.e., there is not only loss of data).

Fragmentation is advantageous as it does not create copies of data, consistency is not a problem.

Fragmentation of relation can be done in two ways.

1) Horizontal Fragmentation (splitting by rows):

The relation is fragmented into groups of tuples so that each tuple is assigned to at least one fragment.

2) ~~Horizontal~~ Vertical Fragmentation (splitting by columns):

The schema of the relation is divided into smaller schemas. Each fragment must contain a common candidate key so as to ensure a lossless join.

In certain cases, an approach that is hybrid of fragmentation and replication is used.

Promises of DDBMS:-

Objective: The DBMS should be responsible for providing a high level of data independence together with replication and fragmentation transparencies.

Introduction:

A distributed Database Management System is then defined as the Software System that permits the management of the distributed database make the distribution transparent to the users.

There are Four Fundamentals which is viewed as Promises of DBMS Technology.

- 1) Transparent Management of distributed and replicated data.
- 2) Reliable access to data through distributed transactions.
- 3) Improved Performance
- 4) Easier System Expansion.

Explanation:-

Transparent Management of distributed and replicated data:- A transparent system is a system which "hides the implementation details from users".

Let us start our discussion with an example
Consider an engineering firm that has offices
in Boston, Waterloo, Paris and San Francisco.

They run projects of each of these sites
and would like to maintain a database of their
employees, the projects and other related data.

Assuming that the database is relational
we can store information in 2 relations.

EMP (ENO, ENAME, TITLE) and

PROJ (PNO, PNAME, BUDGET)

We also introduce a third relation to store
Salary information SAL (TITLE, AMT) and a fourth
relation ASG which indicates which employees
have been assigned to which projects for what
duration with what responsibility.

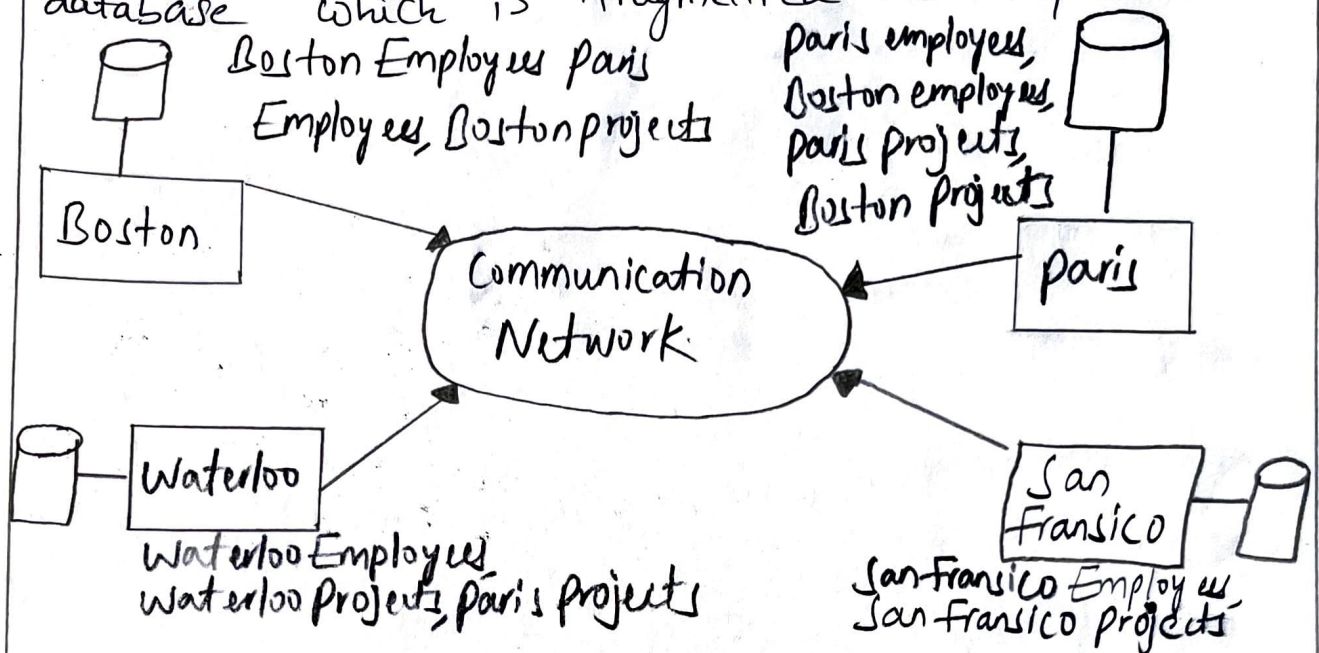
ASG (ENO, PNO, RESP, DUR)

If all this data were stored in a centralized
DBMS, and we wanted to find out the names
and employees who worked on a project for more
than 12 months. we would specify this using the

following SQL query

```
SELECT ENAME, AMT  
FROM EMP, ASG, SAL  
WHERE ASG.DUR > 12  
AND EMP.ENO = ASG.ENO  
AND SAL.TITLE = EMP.TITLE
```

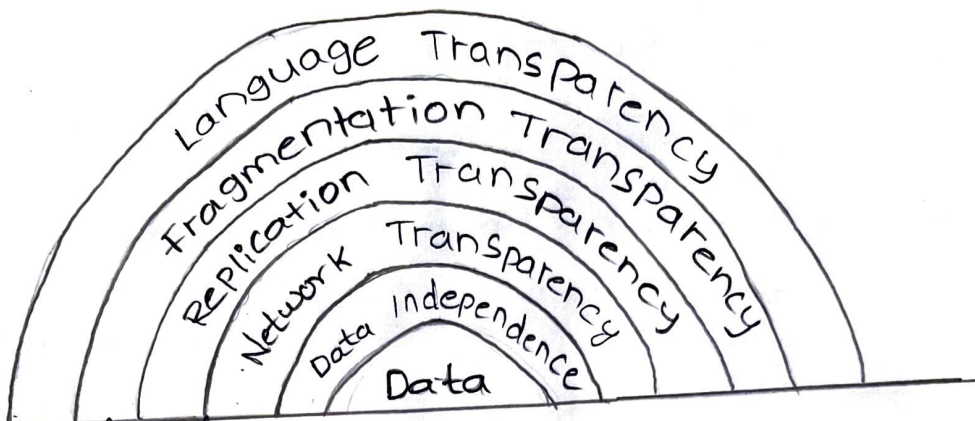
Furthermore, it may be preferable to duplicate some of this data at other sites for performance and reliability reasons. The result is a distributed database which is fragmented and replicated.



Fully Transparent access means that the users can still pose the query as specified above, without paying any attention to the fragmentation, location or replication of data and let the system worry about resolving these issues.

For a system to adequately deal with this type of query over a distributed fragmented and replicated database, it needs to deal with a number of different types of transparencies.

- 1) Data Independence
- 2) Network Transparency
- 3) Replication Transparency
- 4) Fragmentation Transparency



Data Independence:-

Data Independence is a fundamental form of Transparency that we look for within a DBMS. There are 2 types of data independence.

Logical Data Independence:- It refers to the 'immunity' of user appl's to changes in the logical structure of the database.

Physical Data Independence:- It deals with hiding the details of the storage structure from user application.

When a user application is written, it should not need to be modified when data organization changes occur due to performance considerations.

Network Transparency:-

User should be protected from the operational details of the network possibly even hiding the existence of the network. This type of transparency is referred to as network transparency or distribution transparency.

Replication Transparency:-

For performance, reliability and availability reasons it is usually desirable to be able to distribute data in a replicated fashion across the machines on a network.

Furthermore, if one of the machines fails, a copy of data are still available on other machine on the network, in fact the decision as to whether to replicate or not and how many copies of any database object to have depends to a considerable degree on user applications.

Fragmentation Transparency:-

There are two general types of fragmentation alternatives.

Horizontal Fragmentation:— A relation is Partitioned into a set of Sub relations, each of which have a subset of the tuples (rows) of the original relation.

Vertical Fragmentation:—

Where each sub relation is defined on a subset of the attributes (columns) of the original relation.

2) Reliability through Distributed transaction:

→ Distributed DDBMS are designed to improve reliability by having replicated components result in eliminating failure. So the failure create problem to the entire system.

→ In distributed Proper care is taken such that instead of failure part user may be permitted to access other facts of the distributed database.

→ This is useful to support for distributed transactions. Ex: Transaction based on the engineering firm.

→ Assuming that there is an applⁿ that updates the salaries of all the employees by 10%.

→ In the middle of this transaction of a system failure we would like the DBMS to be able to determine upon recovery, where it left off and continues with its operation.

→ Distributed transactions execute at a no. of sites at which, they access the local database.

→ Here we are providing a facility that there is no interruption in any transaction.

3) Improved Performance: -

A distributed DBMS fragments the conceptual database, this is called as data localization.

Advantages: -

- 1) Since each site handles only a portion of the database, contention for CPU and I/O services is not severe.
- 2) Localization reduces remote access delays.
- 3) Implementation of inherent parallelism of distributed system.
- 4) It has inter-query and intra-query parallelism.
- 5) Inter query parallelism to execute multiple queries at the same time.
- 6) Intraquery parallelism is achieved by breaking up a single query into a number of sub queries each of which is executed at a different site accessing ~~each of which is executed at a~~ different part of the distributed database.

4) Easier System Expansion: -

In a distributed environment it is much easier to accommodate increasing database sizes.

→ In general expansion can be handled by adding processing and storage power to the n/w.

This also depends on the overhead of distribution.

→ one aspect of easier system expansion is economics.

→ It normally costs much less to put together a system of "smaller" computers with equivalent power of a single big machine.

Distributed DBMS Problem Areas

Objective: Distributed database would allow to end user to create and stored data anywhere in the n/w where database is situated, while storing and accessing data from distributed database through computer network. There are various problems occurs such as deadlock, concurrency and data allocation using fragmentation.

Introduction:-

The following are problem areas of Distributed database.

- 1) Distributed Database Design
- 2) Query Processing
- 3) Directory Management
- 4) Concurrency Control
- 5) Deadlock Management
- 6) Reliability
- 7) ~~operating system support~~ Heterogeneous databases
- 8) Replication

1) Distributed Database Design:- The question that is being addressed is how the database and the applications that run against it should be placed across the sites.

There are two basic alternatives of placing data: Partitioned (or non-replicated) and replicated.

In the Partitioned Scheme the database is divided into a number of disjoint partitions each of which is placed at a different site.

Replicated designs can be either fully replicated where the entire database is stored at each site, or partially replicated (or partially duplicated) where ~~the entire database is~~ each partition of the database is stored at more than one site but not all the sites.

The two fundamental design issues are fragmentation, the separation of the database into partitions called fragments, and distribution, the optimum distribution of fragments.

2) Query Processing:-

Query Processing deals with designing algorithms that analyze queries and convert them into a series of data manipulation operations.

→ The problem is how to decide on a strategy for executing each query over the network in the most cost effective way, however cost is defined.

→ The factors to be considered are the distribution of data, communication costs, and lack of sufficient locally available information.

→ The objective is to optimize where the inherent Parallelism is used to improve the performance of executing the transaction, subject to the mentioned constraints.

Directory Management:

There are 3 levels of directories

- 1) Conceptual
- 2) Logical
- 3) Physical

Directories are consulted for most database operations. There are many issues that concern whether to distribute or centralized the directories.

Distributed Concurrency Control:-

Concurrency control involves the synchronization of accesses to the distributed database, such that the integrity of the database is maintained.

one not only has to worry about the integrity of a single database, but also about the consistency of multiple copies of the database. The condition that requires all the values of multiple copies of every data item to converge to the same value is called mutual consistency.

→ Deadlock Management

Heterogeneous databases:- Sometimes called multi-databases. Distributed databases are fully Autonomous usually databases already exist and distributed database system integrates them.

Deadlock Management:-

Similar to operating systems deadlock management. Well known solutions are

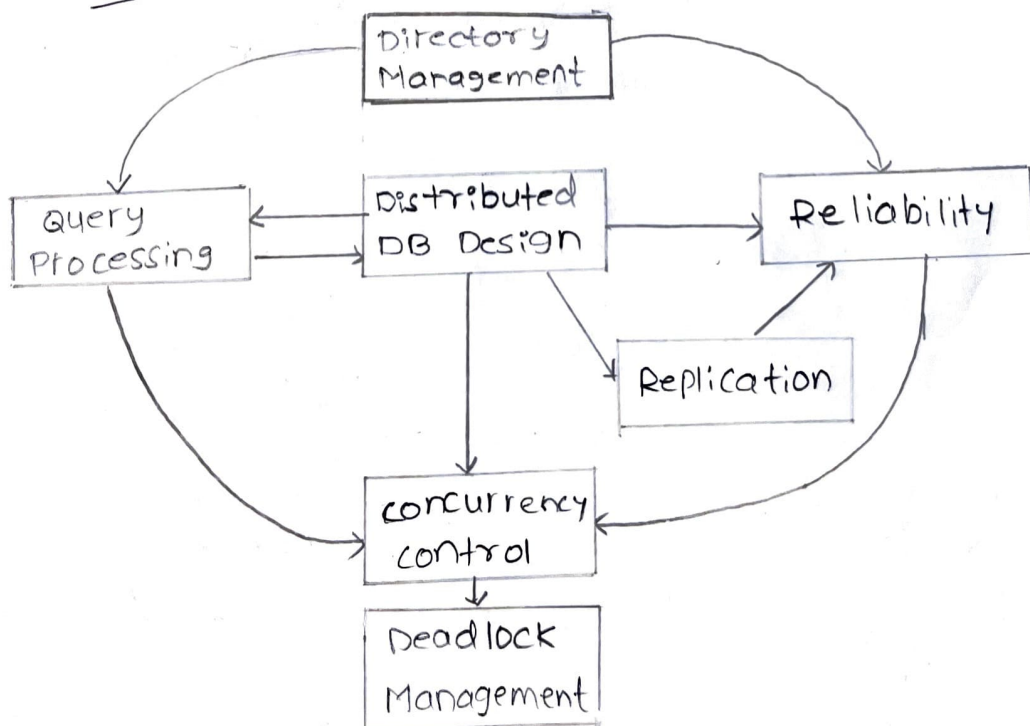
- Prevention
- Avoidance
- Detection/ Recovery.

The competition among users for access to a set of resources (data) can result in deadlock.

Reliability:- The one of the potential advantage of distributed systems is improved reliability and availability.

Replication:- If the distributed database is (partially or fully) replicated, it is necessary to implement protocols that ensure the consistency of the replica i.e. copies of the same data item have the same value.

Relationship between issues:-



Distributed DBMS Architecture

Objective: This method involves redundantly storing the full relationship at two or more locations.

DBMS architecture helps users to get their requests done while connecting to the database. We choose database architecture depending on several factors like the size of the database, number of users, and relationships b/w the users.

Introduction

There are 2 types of distributed DBMS architectures

- 1) ANSI/SPARC Architecture
- 2) A Generic centralized DBMS Architecture.

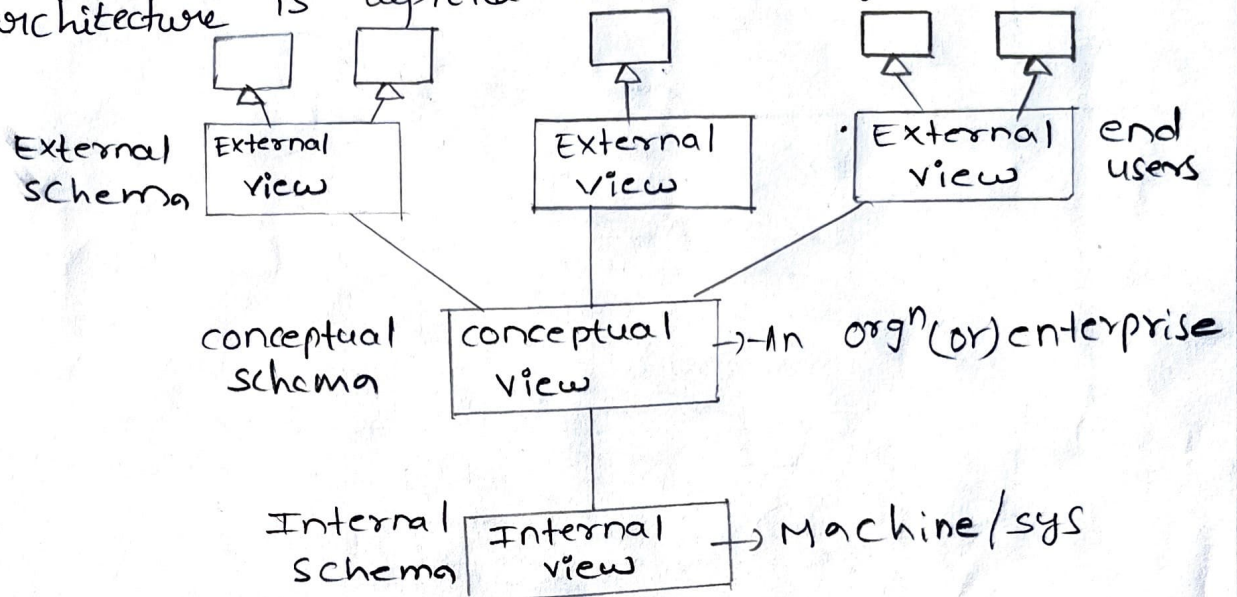
ANSI/SPARC Architecture: In late 1972, the Computer and Information Processing Committee (X3) of the American National Standards Institute (ANSI) established a Study Group on Database Management Systems under the auspices of its Standards Planning Requirements Committee (SPARC).

The mission of the study group was to study feasibility of setting up standards in this area, as well as determining which aspects should be standardized if it was feasible.

The study group issued its interim report in 1975 [ANSI/SPARC, 1975], and its final

report in 1977. The Architectural framework proposed in these reports came to be known as the "ANSI/SPARC architecture".

A simplified version of the ANSI/SPARC architecture is depicted in below figure.



There are 3 views of data

- 1) The external view :- which is that of the end user, who might be a programmer.
- 2) The internal view :- that of system or machine
- 3) The conceptual view :- that of the enterprise.

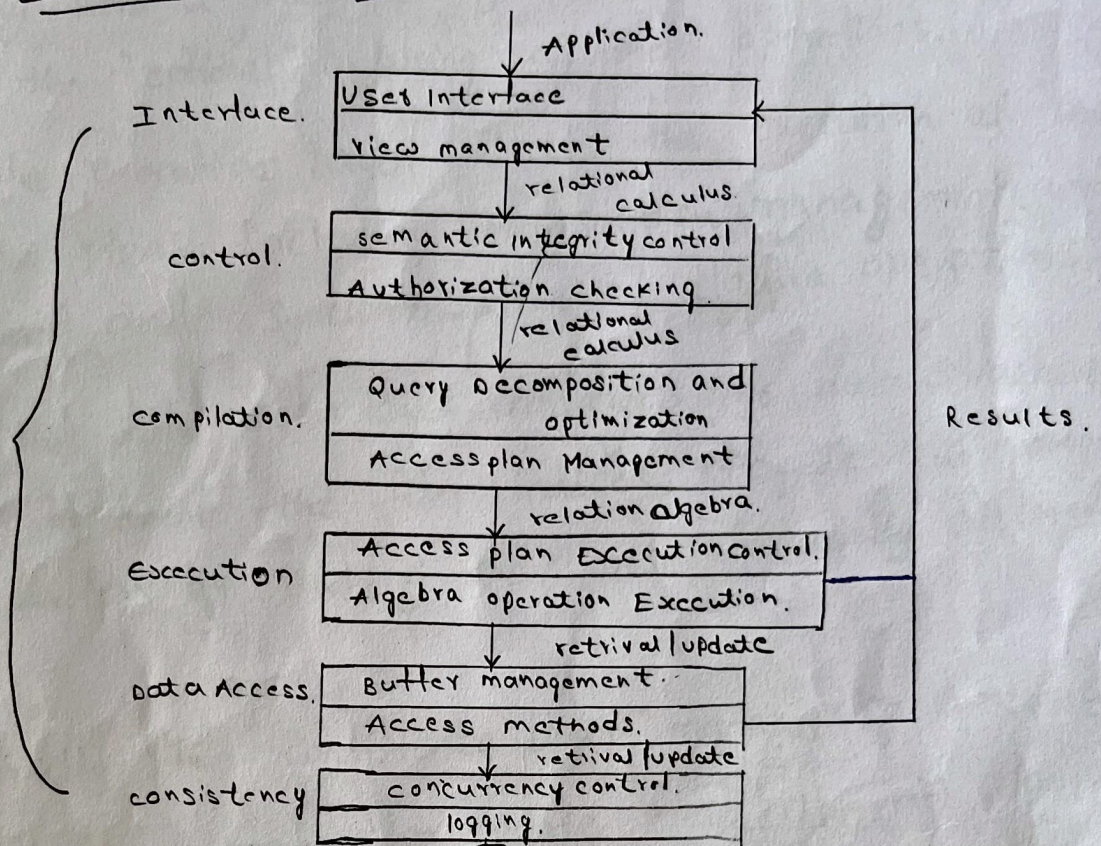
For each of these views, an appropriate Schema-defⁿ is required.

→ At the lowest level of the architecture is the internal view, which deals with the physical defⁿ and organization of data.

→ The separation of the external schemas from the conceptual schema enables logical data independence,

While the Separation of the Conceptual Schema from the internal Schema allows Physical data independence.

2) A Generic centralized DBMS Architecture:



The functions performed by a DBMS can be layered as in above figure, where the arrows indicate the direction of the data and control flow. Taking a top-down approach, the layers are interface, control, compilation, execution, data access, and consistency management.

- 1) The interface layer manages the interface to the applications. There can be several interfaces.
- 2) The control layer control the query by adding semantic integrity predicates and authorization predicates. The o/p of this layer is an enriched query in the high-level language accepted by the interface.

3) The query processing layer maps the query into an optimized sequence of lower-level operations. This layer is concerned with performance. It decomposes the query into a tree of algebra operations and tries to find the "optimal" ordering of the operations.

4) The execution layer directs the execution of the access plans, including transaction management (commit, restart) and synchronization of algebra operations.

5) The data access layer manages the data structures that implement the files, indices, etc. It also manages the buffers by caching the most frequently accessed data.

6) Finally the consistency layer manages concurrency control and logging for update requests. This layer allows transaction, system and media recovery after failure.

Architectural Models for Distributed DBMS:-

Objective:-

The core objective of any database architecture is efficient data management.

Introduction:-

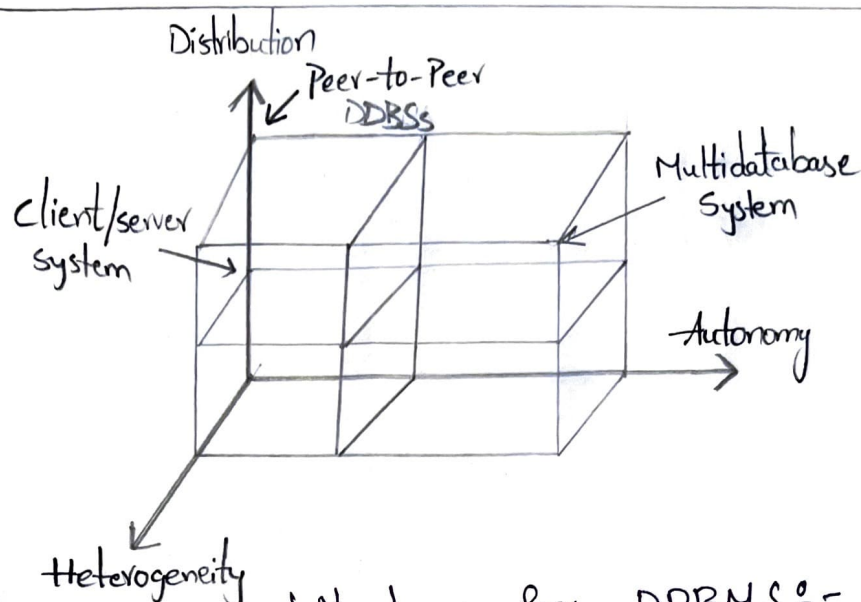
DDBMS architectures are generally developed developing on three parameters

- 1) Distribution: It indicates the physical distribution of data across the different sites.
- 2) Autonomy: It indicates the distribution of control of the database system and the degree to which each constituent DBMS can operate independently.
- 3) Heterogeneity: It refers to the uniformity or dissimilarity of the data models system components and databases.

Architectural Models:

Some of the common Architectural Models are:

- 1) Client-server architecture for DDBMS
- 2) Peer to Peer Architecture for DDBMS
- 3) Multi DBMS Architecture.



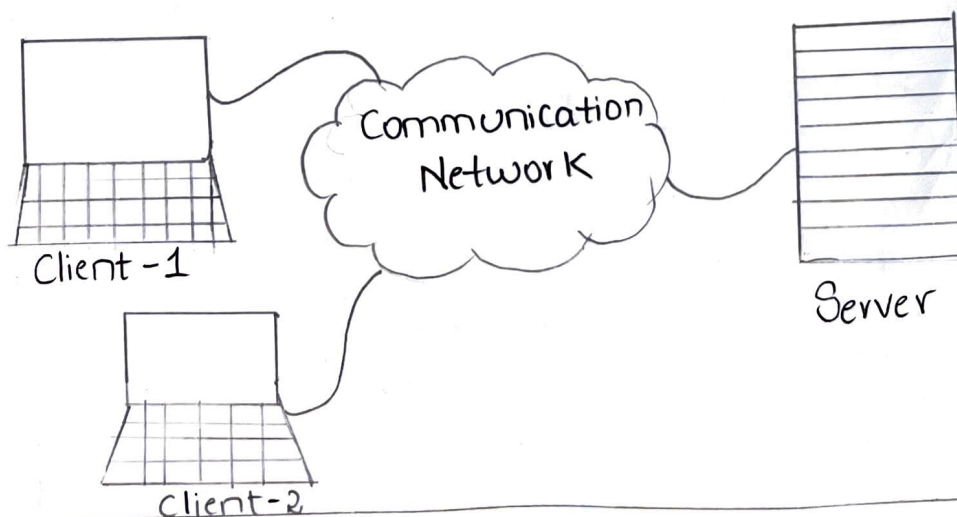
client-server architecture for DDBMS:-

This is a two level architecture where the functionality is divided into Servers and clients. The Server functions primarily encompass data management, query processing, optimization and transaction Management.

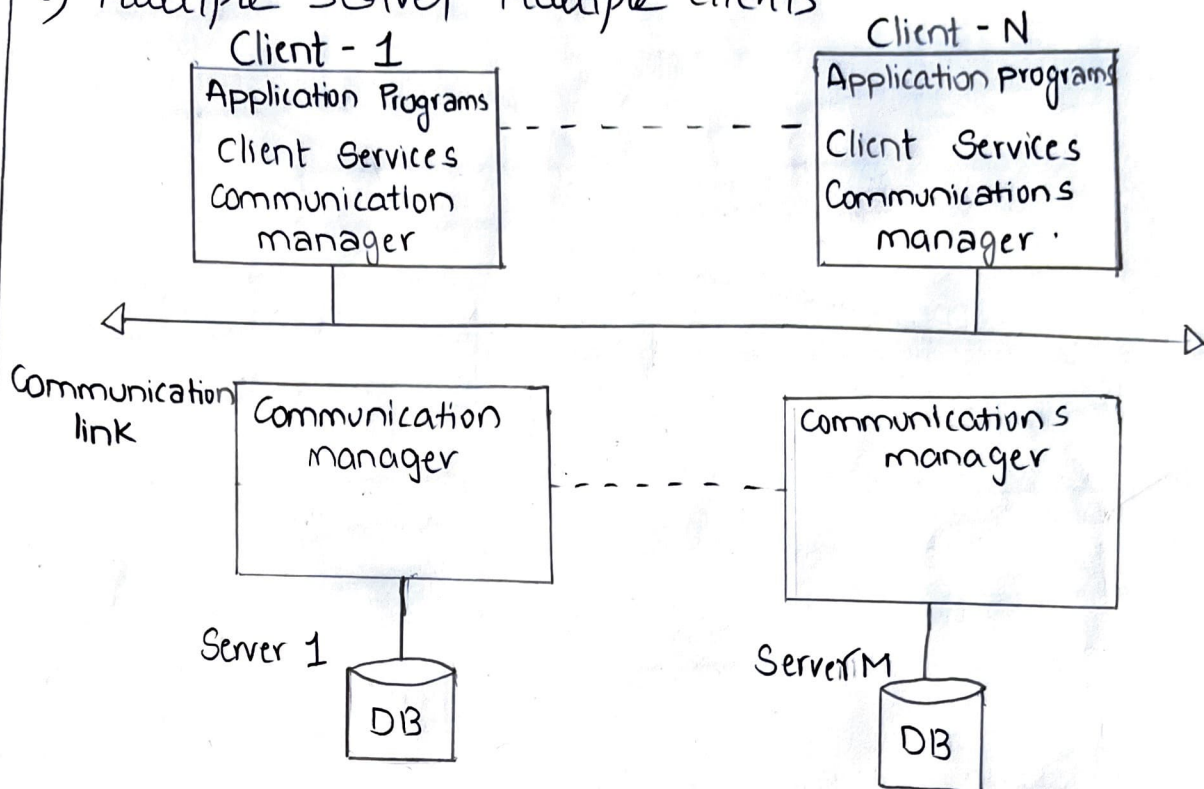
client functions include mainly user interface however they have some functions like consistency checking and transaction management.

The two different client-server architectures are

1) Single Server multiple client



2) Multiple Server Multiple Clients



2) Peer to Peer Architecture for DDBMS

In these Systems each Peer acts both as a client and a server for imparting database services. The peers share their resource with other peers and co-ordinate their activities.

This architecture generally has four levels of

Schemas

Global Conceptual Schema: Depicts ^{the global view of} logical data organization

Local Conceptual Schema: Depicts logical data organization at each site.

Local Internal Schema: Depicts physical data organization at each site.

External Schema: Depicts user view of data.

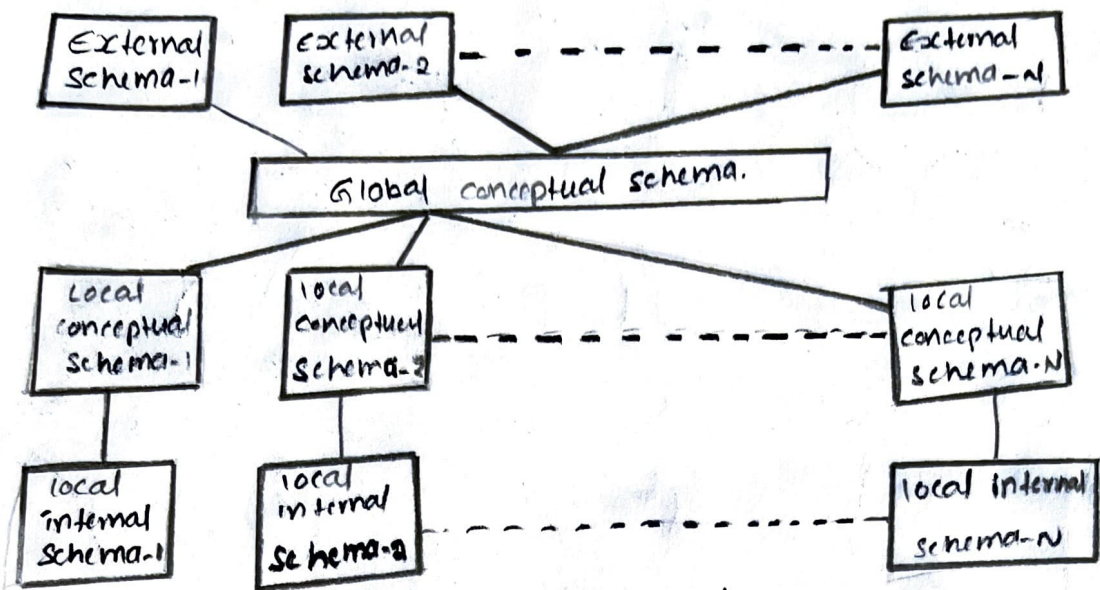


Figure: Framework

Multi-DBMS Architecture:

This is an integrated database system formed by a collection of two or more autonomous database systems.

Multi-DBMS can be expressed through six levels of schemas.

Multi-database view level of schemas:

Multi-database view level: - Depicts multiple user views comprising of subset of the integrated distributed database.

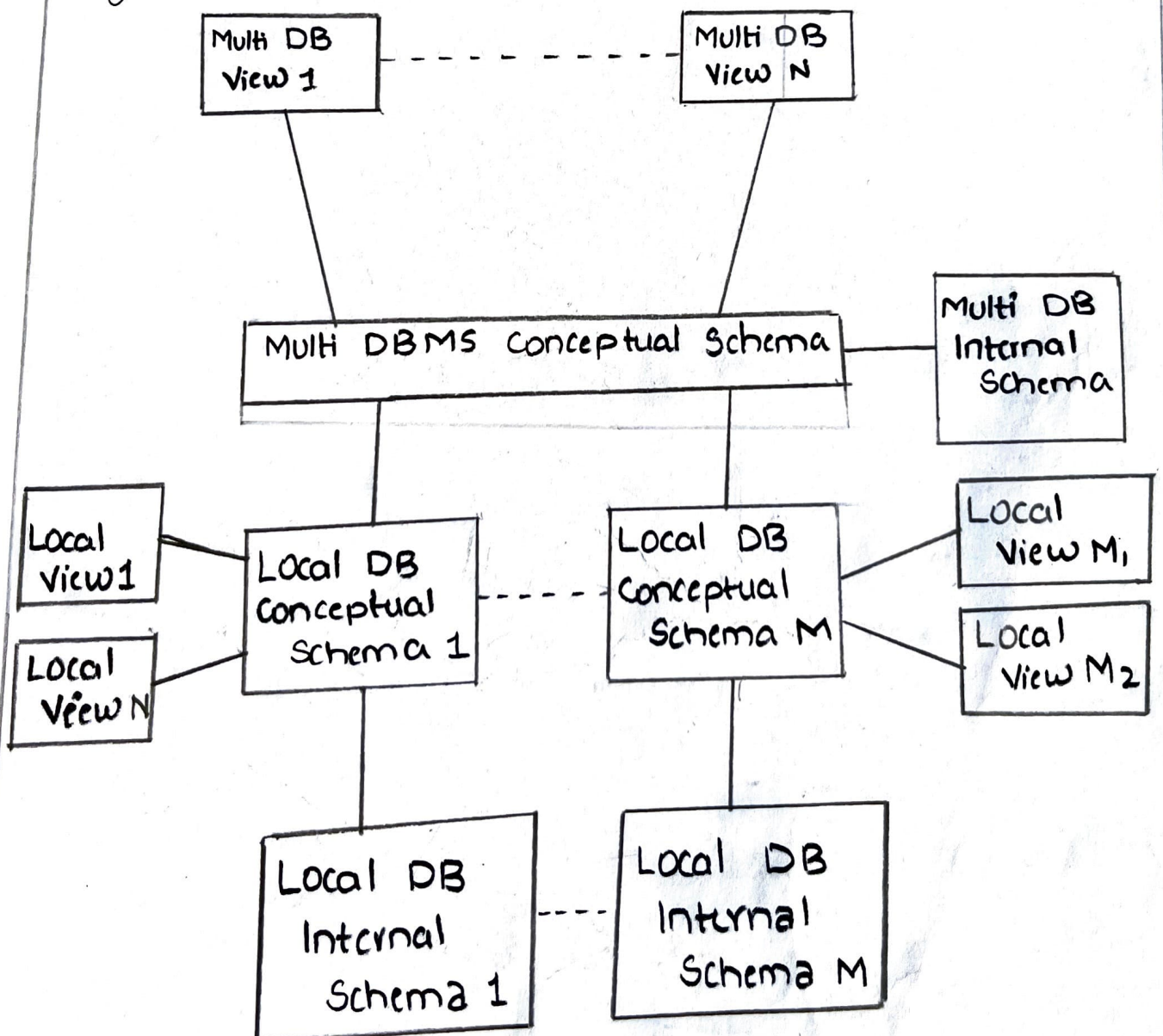
Multi-database conceptual level: Depicts integrated multi-database that comprises of global logical multi-database structure definitions.

Multi-database internal level: depicts the data distribution across different sites and multi-database to local data mapping.

Local database view level: depicts view of local data.

Local database conceptual level: depicts local data organization at each site.

Local database internal level: depicts physical data organization at each site.



Distribution Design Issues

Objective: Handling Increased load, as the number of users or requests increases, the system must scale accordingly without performance degradation.

Introduction:

The following set of interrelated questions cover the entire issues

- 1) Why fragment at all?
- 2) How to fragment (or) How should we fragment?
- 3) How much should we fragment?
- 4) Is there any way to test the correctness of decomposition?
- 5) How should we allocate?
- 6) What is the necessary information for fragmentation and allocation?

Reasons for Fragmentation

→ The important issue is the appropriate unit of distribution. For a number of reasons it is only natural to consider subsets of relations as distribution units.

→ If the application that have views defined on a given relation reside at different sites, two alternatives can be followed, with the entire

relation being the unit of distribution. The relation is not replicated and is stored at only one site, or it is replicated at all or some of the sites where the applications reside.

→ The fragmentation of relations typically results in the parallel execution of single query by dividing it into a set of subqueries that operate on fragments. Thus fragmentation typically increases the level of concurrency and therefore the system throughput.

Fragmentation Alternatives

There are two alternatives

1) Horizontal Fragmentation

PROJ

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop	135000	New York
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris
P5	CAD /CAM	500000	Boston

PROJ Relation divided horizontally into two relations.

PROJ₁ contains information about projects whose budgets are less than 2,000,000

PROJ₁

PNO	PNAME	BUDGET	Loc
P1	Instrumentation	150000	Montreal
P2	Database Develop	135000	New York

Where as PROJ₂ stores information about projects with larger budgets means $> 2,000,000$

PROJ₂

PNO	PNAME	BUDGET	Loc
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris
P5	CAD/CAM	500000	Boston

Vertical Fragmentation

PROJ relation Partitioned Vertically into two sub relations PROJ₁ and PROJ₂.

PROJ₁ contains only the information about project budgets, where as PROJ₂ contains project names and locations. It is important to notice that the Primary key to the relation (PNO) is included in both fragments.

PROJ ₁	
PNO	BUDGET
P ₁	150000
P ₂	135000
P ₃	250000
P ₄	310000
P ₅	500000

PROJ ₂		
PNO	PNAME	Loc
P ₁	Instrumentation	Montreal
P ₂	Database Develop.	New York
P ₃	CAD/CAM	New York
P ₄	Maintenance	Paris
P ₅	CAD/CAM	Boston.

Degree of Fragmentation

- The extent to which the database should be fragmented is an important decision that affects the performance of query execution.
- The degree of fragmentation goes from one extreme, that is, not to fragment at all, to the other extreme, to fragment to the level of individual tuples (in case of horizontal fragmentation) or to the level of individual attributes (in case of vertical fragmentation).

Correctness rules for Fragmentation

Completeness: If a relation instance R is decomposed into fragments $R_1, R_2, \dots, R_n$ each data item that can be found in R can be also found in one or more of R_i 's. This property

is also important in fragmentation since it ensures that the data in a global relation is mapped into fragments without any loss.

Reconstruction: If a relation R is decomposed into fragments $R_1, R_2, \dots, R_n$ it should be possible to define a relational operator ∇ (Nabla) such that:

$$R = \nabla R_i, \quad \forall R_i \in F_R$$

The reconstructability of the relation from its fragments ensures that constraints defined on the data in the ~~data~~ in the form of dependencies are preserved.

Disjointness: If a relation R is horizontally decomposed into fragments $R_1, R_2, \dots, R_n$ and data item d_i is in R_j , it is not in any other fragment R_k ($k \neq j$). This criterion ensures that the horizontal fragments are disjoint.

→ If a relation R is vertically decomposed its primary key attributes are typically repeated in all its fragments. Therefore, in case of vertical partitioning, disjointness is defined only on the nonprimary key attributes of a relation.

Allocation Alternatives

A nonreplicated db (commonly called a partitioned db) contains fragments that are allocated to sites, and there is only one copy of any fragment on the NW.

→ In case of replication, either the database exists in its entirety at each site (fully replicated db), or fragments are distributed to the sites in such a way that copies of a fragment may reside in multiple sites (Partially replicated db)

	Full replication	Partial replication	Partitioning
Query Processing	Easy	← Same difficulty →	
Directory Management	Easy or nonexistent	← Same difficulty →	
Concurrency control	Moderate	Difficult	Easy
Reliability	Very high	High	Low
Reality	Possible Appln	Realistic	Possible Appln

Comparison of Replication Alternatives

Information Requirements

→ The information needed for distributed design can be divided into four categories

- 1) Database Information
- 2) Application information
- 3) Communication information
- 4) Computer System information.

Distributed Database Design

Objective: Distributed database design refers to the given database and its workload, how should the database be split and allocated to sites so as to optimize certain objective function.

Introduction

There are ~~two~~^{four} issues:

- 1) Data fragmentation, which determines how the data should be fragmented.
- 2) Data allocation, which determines how the fragments should be allocated.
- 3) Data ~~allocation~~^{dupli}cation, which increases the availability and improves the performance of the system.
- 4) Location Transparency, which enables a user to access data without knowing, or being concerned with, the site at which the data resides. The location of data is hidden.

Fragmentation:-

Fragmentation is the task of dividing a table into a set of smaller tables. The subsets of the table are called fragments.

Fragmentation can be of three types:

1) Horizontal

2) Vertical

3) Hybrid (combination of horizontal and vertical)

Horizontal fragmentation can further be classified into two techniques ^{Primary} horizontal fragmentation and derived horizontal fragmentation.

Fragmentation should be done in a way so that the original table can be reconstructed from the fragments. This is needed so that the original table can be reconstructed from the fragments whenever required. This requirement is called "reconstructiveness".

Advantages:

- 1) Permits a number of transaction to executed concurrently
- 2) Results in parallel execution of a single query.
- 3) Increase level of concurrency, also referred to as, intra query concurrency.
- 4) Increased System throughput
- 5) Since data is stored close to the site of usage, efficiency of the database.
- 6) Local query optimization techniques are sufficient for most queries since data is locally available.
- 7) Since irrelevant data is not available at the sites, security and privacy of the database.

Disadvantages:

- 1) Applications whose views are defined on more than one fragment may suffer performance degradation, if apps have conflicting requirements.
- 2) Simple tasks like checking for dependencies, would result in chasing after data in a no. of sites.
- 3) When data from different fragments are required, the access speeds may be very high.
- 4) In case of recursive fragmentations, the job of reconstruction will need expensive techniques.
- 5) Lack of back-up copies of data in different sites may render the database ineffective in case of failure of a site.

Vertical Fragmentation:

In vertical fragmentation, the fields or columns of a table are grouped into fragments. In order to maintain reconstructiveness, each fragment should contain the primary key field(s) of the table. Vertical fragmentation can be used to enforce privacy of data.

Grouping:

starts by assigning each attribute to one fragment

→ At each step, joins some of the fragments until some criteria is satisfied.

→ Results in overlapping fragments.

Splitting:

→ Starts with a relation and decides on beneficial partitioning based on the access behaviour of apps to the attributes.

→ Fits more naturally within the top-down design

→ Generates non-overlapping fragments.

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 Reg_NO, Fees FROM STUDENT;
```

Horizontal Fragmentation

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 reconstructiveness.

Each horizontal fragment must have all columns of the original base table.

→ Primary horizontal fragmentation is defined by a selection (operator) operation on the owner relation of a database schema.

→ Given relation R_i , its horizontal fragments are given by $R_i = \sigma_{F_i}(R)$ $1 \leq i \leq W$

F_i selection formula used to obtain fragment R_i .

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 data base as

```
CREATE COMP_STD AS SELECT * FROM STUDENT  
WHERE COURSE = "Computer Science";
```

Desired Horizontal Fragmentation

→ Defined on a number relation of a link according to a selection operation specified on its owner.

→ Link between the owner and member relations is defined as equi-join.

→ An equi-join can be implemented by means of semi joins.

→ Given a link L where $\text{owner}(L) = S$ and $\text{member}(L) = R$, the derived horizontal fragments of R are defined as

$$R_i = R \propto S_i, 1 \leq i \leq W$$

Where $S_i = \sim F_i(S)$

W is the max no of fragments that will be defined on. F_i is the formula using which the primary horizontal fragment S_i is defined.

Hybrid Fragmentation

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 expensive task.

Hybrid Fragmentation can be done in two alternative ways

- 1) At first, generate a set of horizontal fragments; then generate vertical fragments from one or more of the horizontal fragments.
- 2) At first, generate a set of vertical fragments; then generate horizontal fragments from one or more of the vertical fragments.

Data Replication :- Data replication is the process of storing copies of the database at two or more sites. It is a popular fault tolerance technique of distributed databases.

Advantages of Data Replication :-

- 1) Reliability - In case of failure of any site, the database system continues to work since a copy is available at another site(s).
- 2) Reduction in Network load - Since local copies of data are available, query processing can be done with reduced network usage, particularly during prime hours. Data updating can be done at non-prime hours.
- 3) Quicker Response - Availability of local copies of data ensures quick query processing and consequently quick response time.
- 4) Simpler Transactions - Transactions requiring less number of joins of tables located at different sites and minimal coordination across the network. Thus, they become simpler in nature.

Disadvantages

- 1) Increased Storage Requirements - Maintaining multiple copies of data is associated with

Storage costs. The storage space required is in multiples of storage required for centralized system.

2) Increased cost and complexity of data updating -

Each time a data item is updated, the update needs to be reflected in all the copies of the data at the different sites. This requires complex synchronization techniques and protocols.

3) Undesirable Application - Database coupling -

If complex update mechanisms are ^{used,} not removing data inconsistency requires complex co-ordination at application level. This results in undesirable application-database coupling.

Some commonly used replication ^{techniques} are snapshot replication, Near real-time replication, Pull replication.

Steps of Distributed Database design

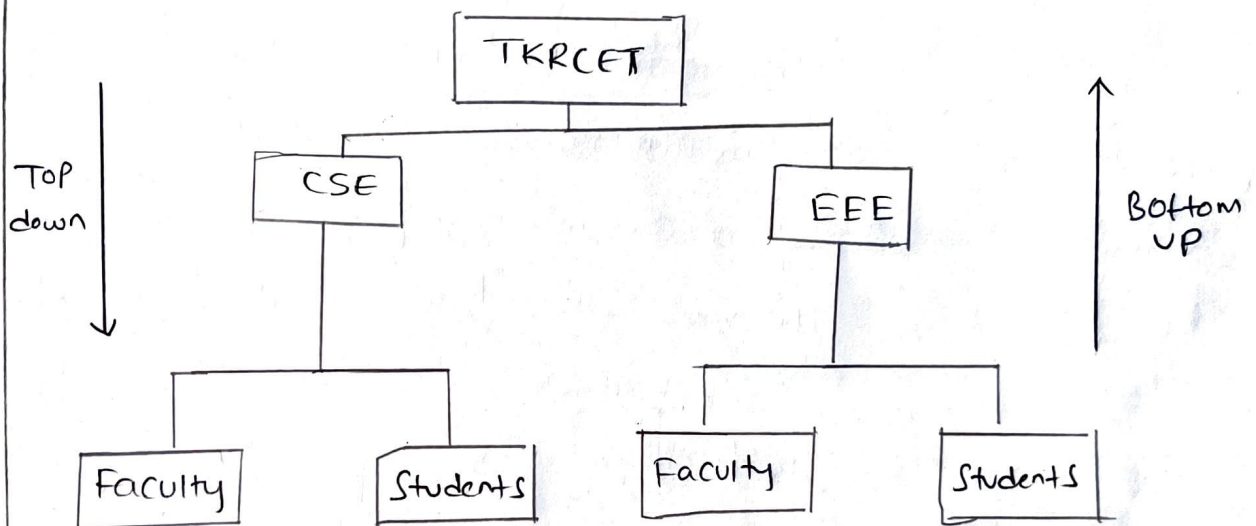
There are in general several design alternatives

Top-down approach: first the general concepts, the global framework are defined, after then the details.

Down-top approach (or) Bottom-up approach: first the details modules are defined, after then the global framework.

If the system is built up from a scratch, the top-down method is more accepted.

→ If the system should match to existing systems or some modules are yet ready, the down-top method is usually used.



Design Alternatives

The distribution design alternative for the tables in a DDBMS are

- 1) Non replicated and non-fragmented
- 2) Fully replicated
- 3) Partially replicated
- 4) Mixed.

Non-replicated and non-fragmented

In this design alternative, different tables are placed at different sites. Data is placed so that it is at a close proximity to the site where it is used most. It is most suitable for database systems where the percentage of queries needed to join information in tables placed at different sites is low. If an appropriate distribution strategy is adopted, then this design alternative helps to reduce the communication cost during data processing.

Fully Replicated: In this design alternative, at each site, one copy of all the database tables is stored. Since, each site has its own copy of the entire database, queries are very fast requiring negligible communication cost. On the contrary, the massive redundancy in data requires huge cost during update operations. Hence, this is suitable

For systems where a large number of queries is required to be handled whereas the number of database updates is low.

Partially Replicated: Copies of tables or portions of tables are stored at different sites. The distribution of the tables is done in accordance to the frequency of access. This takes into consideration the fact that the frequency of accessing the tables vary considerably from site to site. The number of copies of the tables (or portions) depends on how frequently the access queries execute and the site which generate the access queries.

Mixed Distribution: This is a combination of fragmentation and partial replications. Here the tables are initially fragmented in any form (horizontal or vertical), and then these fragments are partially replicated across the different sites according to the frequency of accessing the fragments.

Fragmentation :

There are 3 types of fragmentation.

- 1) Horizontal Fragmentation.
- 2) Vertical Fragmentation.
- 3) Hybrid Fragmentation.

Horizontal Fragmentation

Horizontal fragmentation divides a relation (table) horizontally into the group of rows to create subsets of tables.

Types of Horizontal Fragmentation

- 1) Primary Horizontal Fragmentation
- 2) Derived Horizontal Fragmentation

Primary HF: The relation is fragmented based on simple conditions like $\langle , > , = , \neq , \leq , \geq \rangle$

Example

Employee			
Id	Name	City	Salary
6	Sam	Pune	3500
5	Punit	Pune	4000
7	Kunal	Pune	6900
8	Sanam	Delhi	6900
2	Naitik	Mumbai	6800
3	Sneha	Chennai	6600
4	Pratiksha	Delhi	6200

Now we will create fragments based on city

Fragment 1: Employee whose city is Pune

Query: create table fragment₁ as
select * from employee where city = 'Pune';

Id	Name	city	Salary
6	Sam	Pune	3500
5	Penit	Pune	4000
7	Kunal	Pune	6900

Fragment₁ =

$\sigma_{city = "Pune"}(Employee)$

Fragment₁

Fragment₂: Employee whose city is 'Mumbai'

Query: create table fragment₂ as select
* from Employee where city = 'Mumbai';

Id	Name	city	Salary
2	Natik	Mumbai	6500

Fragment₂ =

$\sigma_{city = "Mumbai"}(Employee)$

Fragment₂

Fragment₃: Employee whose city is 'Delhi'

Query: create table fragment₃ as select
* from Employee where city = 'Delhi';

Id	Name	city	Salary
8	Sonam	Delhi	6900
4	Pratiksha	Delhi	6200

Fragment₃ =

$\sigma_{city = "Delhi"}(Employee)$

Fragment₃

Fragment₄: Employee whose city is 'chennai'

Query: create table fragment₄ as select *
from Employee where city = 'chennai';

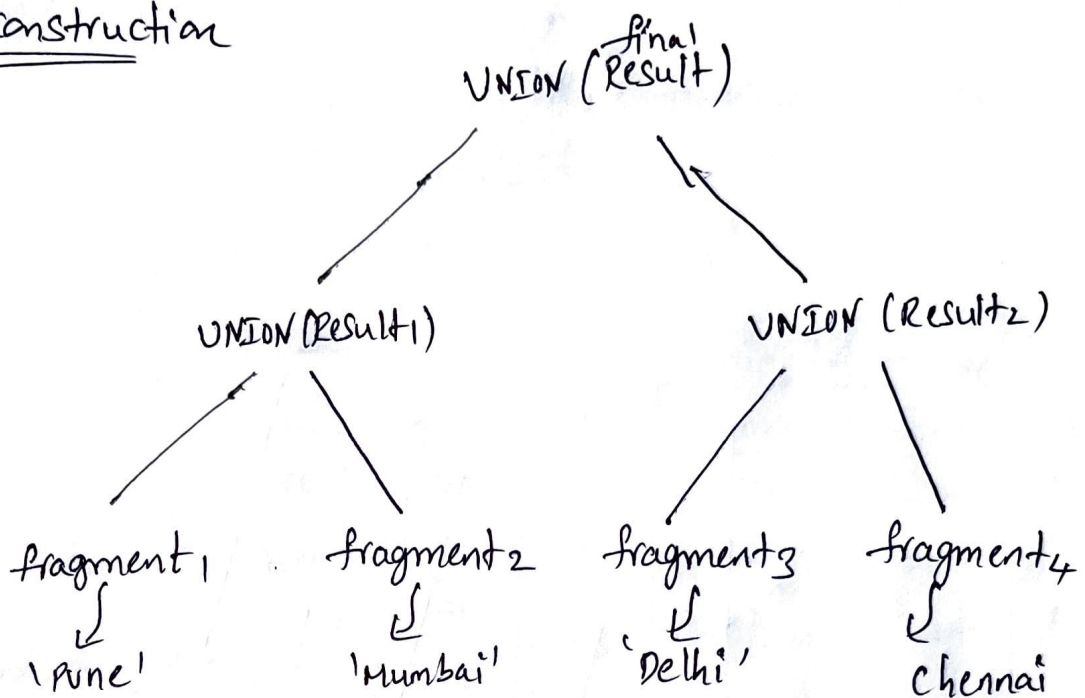
Id	Name	city	Salary
3	Sneha	chennai	6600

Fragment₄ =

$\sigma_{city = "chennai"}(Employee)$

Fragment₄

Reconstruction



Correctness rules

- 1) we should not lose data because of fragmentation
- 2) we should not get redundant data because of fragmentation, $R_1 \cup R_2 \cup R_3 \dots R_n = R$
- 3) Reconstruction $R_1 \cap R_2 \cap R_3 \dots R_n = \text{Nullset}$

Derived Horizontal Fragmentation

In derived horizontal fragmentation, a relation R_1 may get fragmented by attribute of some other relation R_2 .

→ Relation in a DHF are classified as:

- 1) owner relation (has a Primary key)
- 2) Member relation (has a foreign key)
uses the owner's Primary key as foreign key.

Example:

owner relation R_1 (emp-id, emp-name, emp-sal) ^{P.K}

Member relation R_2 (Proj-id, emp-id, role)
_{P.K} _{F.K}

Department table

DNo	DName	Budget	Loc
D ₁	Mgmt	7,50,000	Mumbai
D ₂	Payroll	5,00,000	New Delhi
D ₃	Production	4,00,000	New Delhi
D ₄	Maintenance	3,00,000	Lucknow

owner table

Project table

PNo	PName	Budget	DNo
P ₁	DB Design	1,35,000	D ₂
P ₂	Maintenance	3,10,000	D ₃
P ₃	CAD/CAM	5,00,000	D ₁
P ₄	Architecture	3,00,000	D ₁
P ₅	Documentation	4,50,000	D ₁

Member table

Step₁: Create fragments on owner relation

Create table HF₁ as select * from department where
 loc = 'Mumbai';

DNo	DName	Budget	Loc
D ₁	Mgmt	7,50,000	Mumbai

Create table HF₂ as select * from department
 where loc = 'New Delhi';

DNo	DName	Budget	Loc
D ₂	Payroll	5,00,000	New Delhi
D ₃	Production	4,00,000	New Delhi

Create ^{table} fragment HF₃ as select * from department
 where loc = 'Lucknow';

DNo	DName	Budget	Loc
D ₄	Maintenance	3,00,000	Lucknow

Step 2: Perform Semi-join operation on Project relation & fragments.

Project ~~X~~ HF₁

Project X HF₂

Project X HF₃

Project X HF₁

Project table

PNO	PName	Budget	DNO
P ₁	DB Design	1,35,000	D ₂
P ₂	Maintenance	3,10,000	D ₃
P ₃	CAD/CAM	5,00,000	D ₂
P ₄	Architecture	3,00,000	D ₁
P ₅	Documentation	4,50,000	D ₁

<u>HF₁</u>			
DNO	DName	Budget	Loc
D ₁	Mgmt	7,50,000	Mum bai

create table Project₁ as select PNO, PName, Budget, Project.DNO from Project, HF₁
where Project.DNO = HF₁.DNO

O/P:
Project₁

PNO	PName	Budget	DNO
P ₄	Architecture	3,00,000	D ₁
P ₅	Documentation	4,50,000	D ₁

Perform Project X HF₂ and
Project ~~X~~ HF₃ also.

Min-term predicate:

Min-term predicate which can be written as

$$M_i = P_1 \wedge P_2 \wedge P_3 \wedge P_4 \wedge \dots \wedge P_n$$

Here P_1 means P_1 or $\neg(P_1)$
↳ not

P_2 mean P_2 or $\neg(P_2)$.

Using the conjunctive form of various simple predicates in different combination, we can derive many such min-term predicate.

Example 1

AC No	Balance	Branch-name
A101	5000	Mumbai
A103	10000	New Delhi
A104	2000	Chennai
A102	12000	Chennai
A110	6000	Mumbai
A115	6000	Mumbai
A120	2800	New Delhi

Account table

→ for the table Account, if simple conditions are $\text{Balance} < 10000$, $\text{Balance} > 10000$ then $\rightarrow P_1$

→ we can derive set of min-term ^{above} of predicates using the rules stated ^{above} as follows

→ we will get 2^n min-term predicates, where n is the number of simple predicates in the given predicate set.

For p_1 , we have 2 simple predicates, Hence we will get $2^2 = (4)$ possible combinations of min-term predicates

$$M_1 = \{ \text{Balance} < 10000 \wedge \text{Balance} > 10000 \}$$

$$M_2 = \{ \text{Balance} < 10000 \wedge \neg (\text{Balance} > 10000) \}$$

$$M_3 = \{ \neg (\text{Balance} < 10000) \wedge \text{Balance} > 10000 \}$$

$$M_4 = \{ \neg (\text{Balance} < 10000) \wedge \neg (\text{Balance} > 10000) \}$$

our next step is to choose the min-term predicates which can satisfy certain conditions to fragment a table, and eliminate the others which are not useful.

$$\text{Account}_1 = \overline{\text{Balance} < 10000 \wedge \text{Balance} > 10000} \quad (\text{Account})$$

SQL query:

SELECT * from account where Balance < 10000
AND Balance > 10000

The first query for fragment $\text{Account}_1 (M_1)$ is invalid as any record in a table cannot have two values for any attribute in one record. i.e., $(\text{Balance} < 10000 \wedge \text{Balance} > 10000)$ requires the values for balance must both be less than 10000 and greater than and equal to 10000, which is not possible. Hence the condition violates and can be eliminated.

$Account_2 = \neg (Balance < 10000 \wedge \neg (Balance \geq 10000)) (Account)$

Query:

SELECT * from Account where balance < 10000
AND NOT Balance $\geq$ 10000;

Where NOT balance $\geq$ 10000 is equivalent to balance < 10000

→ For fragment $Account_2$ (Min-term predicate m_2), the condition is (balance < 10000 and balance < 10000) which ultimately means balance < 10000 which is correct.

Account₂

AcNo	Balance	Branch-name
A101	5000	Mumbai
A104	2000	Chennai
A110	6000	Mumbai
A115	6000	Mumbai
A120	2500	New Delhi

$Account_3 = \neg (Balance < 10000) \wedge Balance \geq 10000 (Account)$

Query:

Select * from account where NOT Balance
< 10000 AND Balance $\geq$ 10000

where NOT Balance < 10000 is equivalent to balance $\geq$ 10000
Likewise fragment 3 is valid

Accounts		
Acno	Balance	Branch-name
A103	10000	New Delhi
A102	12000	Chennai

→ Account4 = $\neg(\text{Balance} < 10000) \wedge \neg(\text{Balance} > 10000)$ (Account)

Query:

Select * from account where not Balance < 10000
AND NOT Balance > 10000;
Account4 condition is not valid and must
be eliminated.

Vertical Fragmentation

Vertical fragmentation divides a relation (table)
vertically into group of columns to create
subsets of tables.

Example

Eid	Ename	Salary	Bonus	Project	Exp
1	ABC	50,000	5000	P1	5
2	DEF	40,000	7000	P2	3
3	GHI	45,000	8000	P3	2

→ We will make fragments based on relevant
information.

→ Primary key is included in every fragment

Fragment₁ - Basic Employee details (Eid, Ename, Salary)

Eid	Ename	Salary
1	ABC	50,000
2	DEF	40,000
3	GHI	45,000

Fragment₁

Fragment₂ - Bonus on Projects (Eid, Project, Bonus)

Eid	Bonus	Project
1	5000	P ₁
2	7000	P ₂
3	8000	P ₃

Fragment₂

Reconstruction:

Reconstruction of vertical fragmentation is performed by using full outer join operation on fragments.

→ Select * from fragment₁ NATURAL JOIN fragment₂

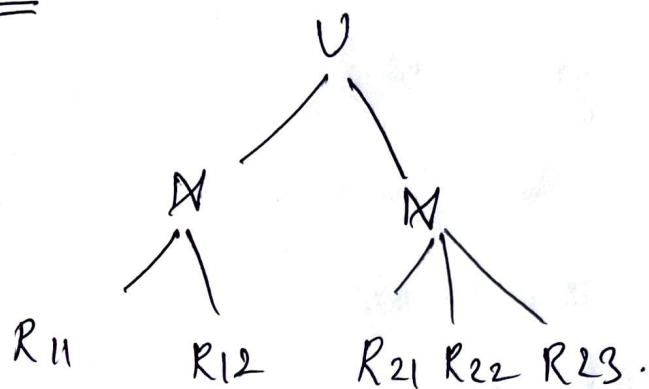
Hybrid Fragmentation is combination

Hybrid Fragmentation of horizontal and vertical fragmentation techniques are used.

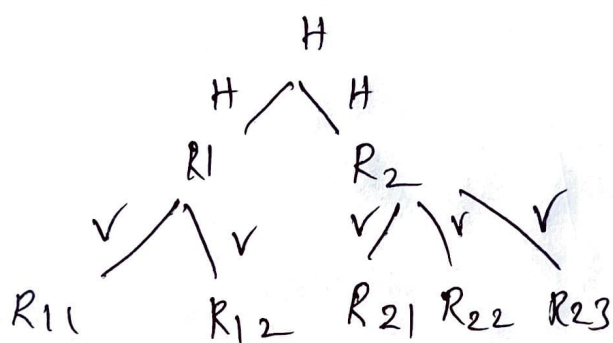
→ Mixed fragmentation is group of rows and columns in relation.

However, reconstruction of the original table is often a expensive task, original relation obtained by combination of join and union operations.

Reconstruction



Hybrid



Allocation :-

Objective : The objective of data allocation in distributed database design is to minimize the total cost of data transfer while executing queries. This is done by finding the best way to place fragments of a database across multiple sites.

Introduction :

Allocation of data is one of the key design issues of distribution database.

→ A database that consists of two or more data files located at different sites on a computer network is known as data allocation.

→ Data allocation describes the process of deciding where to locate data.

→ Data allocation strategies are

- 1) Centralized — Consists of a single DB and DBMS stored at one site with users distributed across the network.
- 2) Partitioned — Database is partitioned into disjoint fragments, each fragment assigned to one site.

3) Replicated — with replicated data allocation, copies of one or more database fragments are stored at different sites.

Factors affecting Data allocation

- Performance and data availability goals.
- Size, the number of rows, and the number of relations that an entity maintains with other entities.
- Types of transactions to be applied to the database and the attributes accessed by each of these transactions.

Allocation model:

Information Requirements

- Database Information
 - Selectivity of fragments
 - Size of a fragment
- Application Information
 - Number of read accesses of a query to a fragment.
 - Number of update accesses of query to a fragment.
 - A matrix indicating which queries updates which fragments.
 - A similar matrix for retrievals.

- originating site of each query.

Site Information

- unit^{cost} of storing data at a site.
- unit cost of processing at a site.

Network Information

- Communication cost / frame between two sites.
- frame size (frames are units of data in the link layer (e.g. WiFi, Bluetooth, Ethernet, etc))

It has two components

- 1) Processing component
- 2) Transmission component

1) Processing component:-

Three factors

1) Access cost (AC)

$$AC_i = \sum_{\text{all sites}} \sum_{\text{all fragments}} (U_{ij} * UR_{ij} + \pi_{ij} * RR_{ij}) * LPCK * x_{jk}$$

U_{ij} = equals 1 when update else 0 on f_j

UR_{ij} = no. of update access

π_{ij} = equals 1 when read else 0 on f_j

RR_{ij} = no. of read access

$LPCK$ = cost of access at a site K

x_{jk} = This ensures that f_j is stored at site k

The decision variable is x_{jk} , which is defined as

$$x_{jk} = \begin{cases} 1 & \text{if the fragment} \\ 0 & \text{otherwise if is stored at site } k. \end{cases}$$

2) Integrity enforcement and 3) Concurrency control algorithm execution time costs are IE & CC respectively.

2) Transmission Component

It has two component

- 1) Processing update cost
- 2) Processing retrieval cost-

→ In update queries it is necessary to inform all the sites where replicas exist.

→ In addition, at the end of an update request, there is no data transmission back to the originating site other than confirmation message.

Processing update cost

$$\begin{aligned} \text{TCU}_i &= \sum_{\substack{\text{All} \\ \text{sites}}} \sum_{\substack{\text{All} \\ \text{fragments}}} \text{update message cost} + \\ &\quad \sum_{\substack{\text{All} \\ \text{sites}}} \sum_{\substack{\text{All} \\ \text{fragments}}} \text{ACKnowledge cost} \end{aligned}$$

Transmission cost for update

2) Processing Retrieval Cost

→ In retrieval queries, it is sufficient to access only one of the copies.

→ The equation states that among all the sites with copies of the same fragment, only the site that yields the minimum total transmission cost should be selected for the execution of the operation.

$$\begin{array}{l} \text{Transmission} \\ \text{Cost for} \\ \text{Retrieval} \end{array} TCR_i = \sum_{\substack{\text{All} \\ \text{Site}}} \min_{\text{all sites}} \left(\begin{array}{l} \text{cost of retrieval command} \\ + \text{cost of sending back the result} \end{array} \right)$$

Now the transmission cost function for query q_i can be specified as

Transmission cost function

$$TC_i = TCU_i + TCR_i.$$

UNIT-II

Query Processing objectives

- 1) The main objectives of query processing in a distributed environment is to form a high level query on distributed database, which is seen as a single database by 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 sum of all times incurred in processing the operations of the query at various sites 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 sites, the response time of the query may be significantly less than its cost.

4). obviously the total cost should be minimized.

i) 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.

ii) ^{In} centralized systems, only CPU and I/O cost have to be considered.

Characterization of query processors

Important characteristics of query processor are

- 1) Languages
- 2) Types of optimization
- 3) Optimization Timing
- 4) Statistics
- 5) Decision Sites
- 6) Exploitation of the Network Topology
- 7) Exploitation of Replicated fragments
- 8) Use of Semi Join.

Languages :- Input language to the query processor is based on high level relational DBMS. i.e, input is in calculus query.

on the other hand output language is based on lower level relational DBMS. i.e, output is in algebraic query.

The operations of the output language are implemented directly in the computer system.

Query processing must perform efficient mapping from input language to the output language.

types of optimization :- Among all possible strategies for executing query the one in which less time and space are required is the best solution for the optimization query.

A comparative study for all the strategies should perform such that how much cost will be required for each solution that strategy should be chosen in which there will be less cost, but keep in mind that choosing the low cost strategy may require much space. so care should be taken in this case also.

optimization Timing:- The actual time required to operate the execution of a query is an important factor, if less time is required, then it is the best solution for query processing.

Statistics:- The effectiveness of query optimization relies on statistical information of the database, i.e., how many fragments query will be needed, which operation should be done first.

Size and number of distinct values of each attribute of the relation, histograms of the attributes for minimizing probability error etc.

Decision Sites:- In a distributed database, several sites may participate for answering the query. most systems are centralized decision approach, in which a single site generates the strategy. However the decision process could be distributed among various sites.

Exploitation of Network Topology :-

Distributed query processor uses computer network. So its performance depends also on which topology it is using. The cost function, speed, utilization of various network, resources are important factors for executing query process or in a distributed environment.

Exploitation of Replicated Fragments

A distributed relation is usually divided in to relation fragments.

Distributed queries expressed on global relations are mapped in to queries on physical fragments of relations by translating relations in to fragments. we call this process localization because its main function is to localize the data involved in the query. For higher reliability and better performance, it is useful to have fragments replicated at different sites.

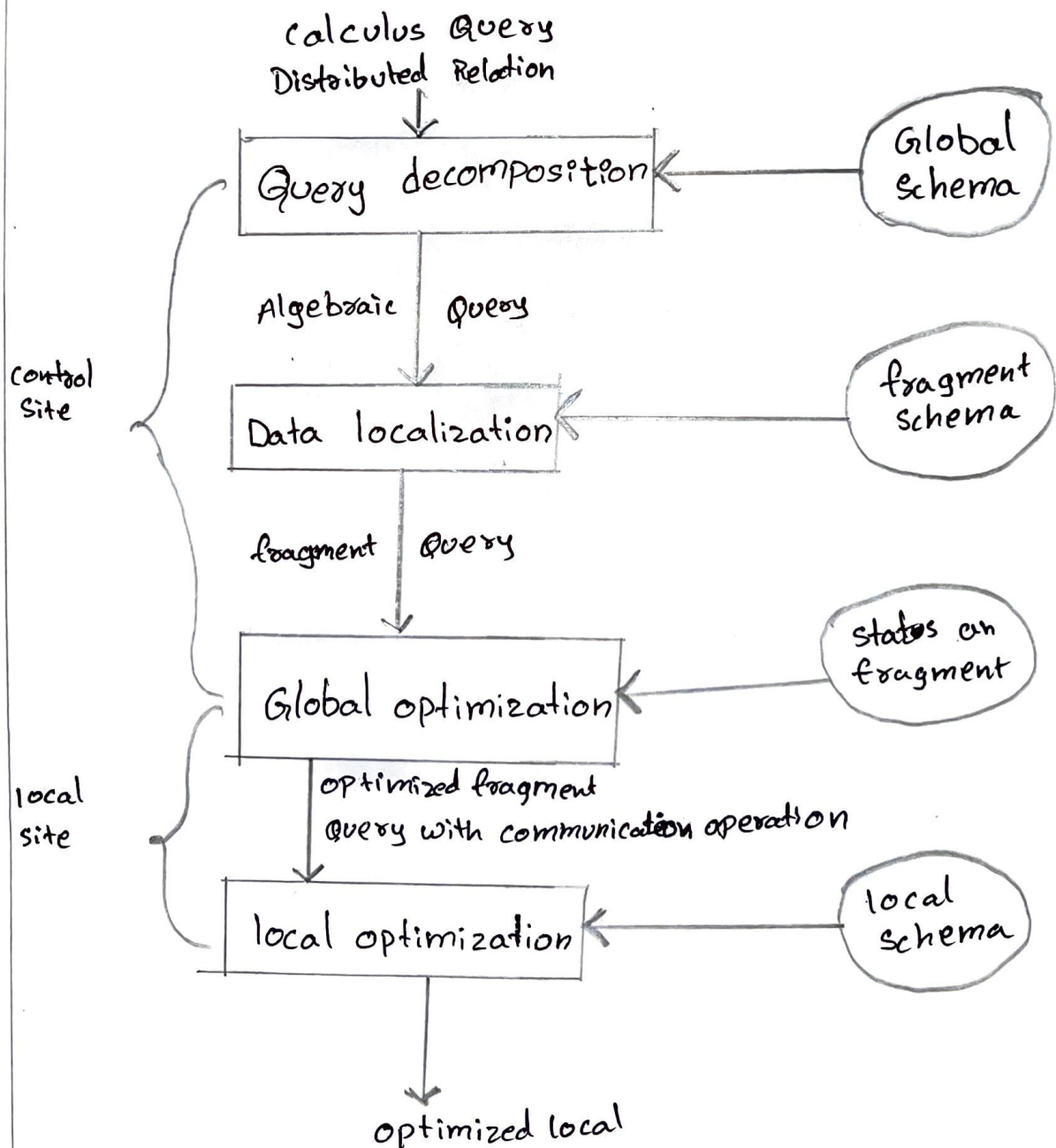
Use of Semi join operation :-

In distributed system, it is better to use semi join operation instead of join operation for minimizing data communication.

Layers of Query Processing

Query processing has 4 layers:

- 1) Query decomposition
- 2) Data localization
- 3) Global query optimization
- 4) Distributed query execution.



Query Decomposition

The first layer decomposes the 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.

→ query decomposition can be viewed as four successive steps:

- 1) Normalization
- 2) Analysis
- 3) Simplification (or) Elimination of redundancy.
- 4) Restructure (or) Rewriting.

Normalization:- First the calculus query is rewritten in a normalized form that is suitable for subsequent manipulation. Normalization of a query generally involves the manipulation of the query quantifiers and of the query qualification by applying logical operator priority.

with relational languages such as SQL, the most important transformation is that of the query qualification (the WHERE clause), which may be an arbitrarily complex, quantifier free predicate, preceded by all necessary quantifiers ($\forall$ or $\exists$).

There are two possible normal forms for the Predicate, one giving Procedure to the AND($\wedge$) and the other to the OR($\vee$).

The conjunctive normal form is a conjunction ($\wedge$ Predicate) of disjunctions ($\vee$ Predicates) as follows:

$$(P_{11} \vee P_{12} \vee \dots \vee P_{1n}) \wedge \dots \wedge (P_{m1} \vee P_{m2} \vee \dots \vee P_{mn})$$

where P_{ij} is a Simple Predicate.

A qualification is disjunctive normal form, on the other hand, is as follows:

$$(P_{11} \wedge P_{12} \wedge \dots \wedge P_{1n}) \vee \dots \vee (P_{m1} \wedge P_{m2} \wedge \dots \wedge P_{mn})$$

well known equivalence rules for logical operations

($\wedge$, $\vee$, and $\neg$):

1) $P_1 \wedge P_2 \Leftrightarrow P_2 \wedge P_1$

2) $P_1 \vee P_2 \Leftrightarrow P_2 \vee P_1$

3) $P_1 \wedge (P_2 \wedge P_3) \Leftrightarrow (P_1 \wedge P_2) \wedge P_3$

4) $P_1 \vee (P_2 \vee P_3) \Leftrightarrow (P_1 \vee P_2) \vee P_3$

5) $P_1 \wedge (P_2 \vee P_3) \Leftrightarrow (P_1 \wedge P_2) \vee (P_1 \wedge P_3)$

6) $P_1 \vee (P_2 \wedge P_3) \Leftrightarrow (P_1 \vee P_2) \wedge (P_1 \vee P_3)$

7) $\neg (P_1 \wedge P_2) \Leftrightarrow \neg P_1 \vee \neg P_2$

8) $\neg (P_1 \vee P_2) \Leftrightarrow \neg P_1 \wedge \neg P_2$

9) $\neg (\neg P) \Leftrightarrow P$

Ex: Find the names of employees who have been working on project P_1 for 12 or 24 months

The query expressed in SQL is

Select Ename from Emp, ASG where
 $\text{Emp.ENO} = \text{ASG.ENO}$ AND $\text{ASG.PNO} = "P_1"$ AND
 $\text{DUR} = 12$ OR $\text{DUR} = 24$.

The qualification in conjunctive normal form is

$\text{Emp.ENO} = \text{ASG.ENO} \wedge \text{ASG.PNO} = "P_1" \wedge$
 $(\text{DUR} = 12 \vee \text{DUR} = 24)$

The qualification in disjunctive normal form is

$(\text{Emp.ENO} = \text{ASG.ENO} \wedge \text{ASG.PNO} = "P_1" \wedge \text{DUR} = 12)$

$\vee (\text{Emp.ENO} = \text{ASG.ENO} \wedge \text{ASG.PNO} = "P_1" \wedge \text{DUR} = 24)$

In the latter, treating the two conjunctions independently may lead to redundant work if common subexpressions are not eliminated.

Analysis: - Second, the normalized query is analyzed semantically so that incorrect queries are detected and rejected as early as possible. Techniques to detect incorrect queries exist only for a subset of relational calculus. Typically, they use some sort of graph that captures the semantics of the query.

A query is type incorrect if any of its attribute or relation names are not defined in global schema, or if operations are being applied to attributes of the wrong type.

Ex:

Select E# from Emp where Ename > 200

In the above query, first attribute E# is not declared in the schema, second, the operation "> 200" is incompatible with the type string of Ename.

→ A query is semantically incorrect if its components do not contribute in any way to the generation of the result.

2 Techniques are used for whether the query is semantically correct or not

- 1) Connection Graph or Query Graph
- 2) Join Graph.

1) Connection Graph:- In a query graph, one node indicates the result relation, and any other node indicates an operand relation.

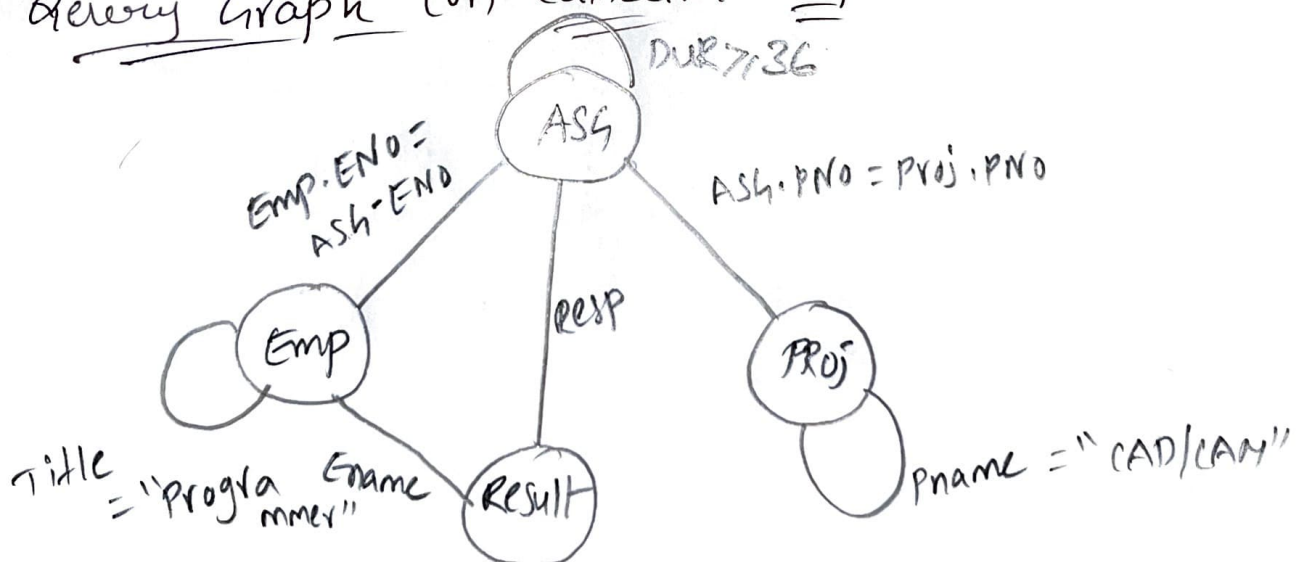
→ An edge between two nodes one of which does not correspond to the result represents a join. Where as edge whose destination node is the result represents a ~~join~~ project.

Furthermore, non-result node may be labeled by a select or a self join (join of the relation with itself) Predicate.

Ex: Find the names and responsibilities of programmers who have been working on the CAD/CAM project for more than 3 years.

```
select Ename, Resp  
from Emp, ASG, Proj  
where Emp.Eno = ASG.Eno  
AND ASG.Eno = Proj.Eno  
AND Pname = "CAD/CAM"  
AND DUR > 36  
AND TITLE = "Programmer"
```

Query Graph (or) Connection Graph

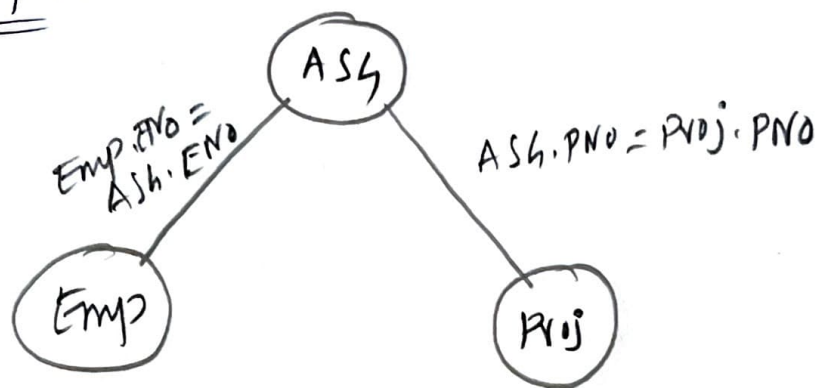


→ Nodes represent operand (or) result relation

→ Edge represent a join if both are connected nodes otherwise projection.

An important subgraph of the query graph is the join graph, in which only the joins are considered. The join graph is particularly useful in the query optimization phase.

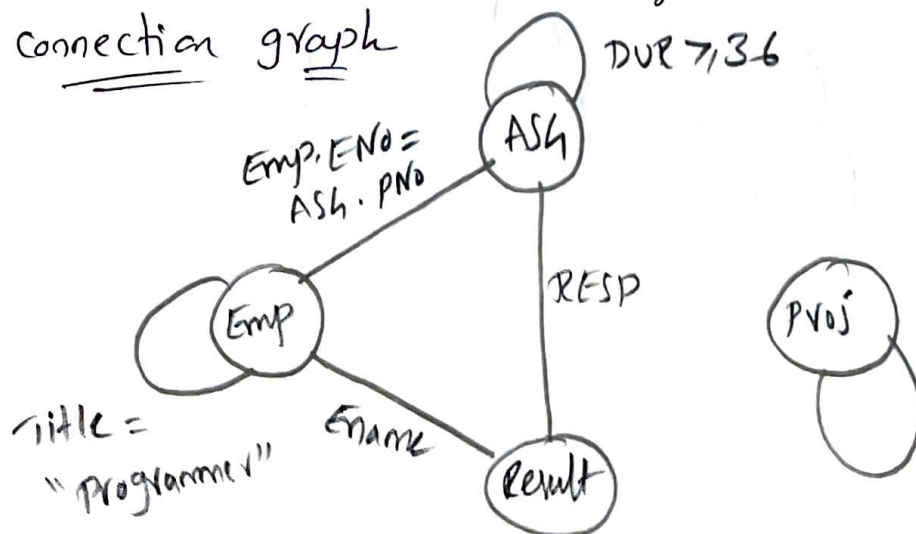
Join graph:-



→ The query is semantically incorrect if its query graph is not connected.

Ex: select Ename, RESP
 from Emp, ASG, PROJ
 Where Emp.ENO = ASG.ENO
 AND Pname = "CAD/CAM"
 AND DUR > 36
 AND TITLE = "Programmer"

connection graph



The figure shows that the query graph is disconnected, which tells us that the query is semantically incorrect.

3) Simplification (or) Elimination of Redundancy

Third, the correct query (still expressed in relational calculus) is simplified. one way to simplify a query is to eliminate redundant predicates.

→ A naive evaluation of qualification with redundancy can well lead to duplicated work. Such redundancy and thus redundant work may be eliminated by simplifying the qualification with the well-known idempotency rules:

$$1) P \wedge P \Rightarrow P$$

$$2) P \vee P \Rightarrow P$$

$$3) P \wedge \text{true} \Rightarrow P$$

$$4) P \vee \text{false} \Rightarrow P$$

$$5) P \wedge \text{false} \Rightarrow \text{false}$$

$$6) P \vee \text{true} \Rightarrow \text{true}$$

$$7) P \wedge \neg P \Rightarrow \text{false}$$

$$8) P \vee \neg P \Rightarrow \text{true}$$

$$9) P_1 \wedge (P_1 \vee P_2) \Rightarrow P_1$$

$$10) P_1 \vee (P_1 \wedge P_2) \Rightarrow P_1$$

Ex: Select title
 from emp
 where (NOT (TITLE = "Programmer")
 AND (TITLE = "Programmer" OR TITLE =
 "Elec. Engg.") AND NOT (TITLE = "Elec. Engg.")
 OR Ename = "J. Doe"

The Simplification Proceeds as follows

Let P_1 be TITLE = "Programmer",
 P_2 be TITLE = "Elec. Engg" and
 P_3 be Ename = "J. Doe"

The query qualification is

$$(\neg P_1 \wedge (P_1 \vee P_2) \wedge \neg P_2) \vee P_3$$

The disjunctive normal form for this qualification
 is obtained by applying rule 5 defined in
 normalization

$$(\neg P_1 \wedge ((P_1 \wedge \neg P_2) \wedge (P_2 \wedge \neg P_2))) \vee P_3$$

then rule 3 defined in normalization

$$((\neg P_1 \wedge P_1 \wedge \neg P_2) \vee (\neg P_1 \wedge P_2 \wedge \neg P_2)) \vee P_3$$

Applying rule 7 defined above

$$(false \wedge \neg P_2) \vee (\neg P_1 \wedge false) \vee P_3$$

Applying the same rule

$$false \vee false \vee P_3$$

which is equivalent to P_3 by rule 4

Simplified query is

SELECT TITLE

FROM Emp

WHERE ENAME = "J. DOE".

ReWriting (or) Reconstruction:

Fourth, the calculus query is restructured as an algebraic query.

→ For the sake of clarity it is customary to represent the relational algebra query graphically by an operator tree.

→ An operator tree is a tree in which leaf node is a relation stored in the database, and a non-leaf node is an intermediate relation produced by a relational algebra operator.

The sequence of operations is directed from the leaves to the root, which represents the answer to the query.

In SQL the leaves are immediately available in the FROM clause.

→ Second the root node is created as a Project operation involving the result attributes. These are found in the SELECT clause in SQL.

Third, the qualification (SQL WHERE clause) is translated into the appropriate sequence of relational operations (Select, Join, union etc) going from the leaves to the root. The sequence can be given directly by the order of appearance of the Predicates and operators.

Ex: Find the names of employees other than J. Doe who worked on the CAD/CAM project for either one or two years

Select Ename

from Emp, ASG, Proj

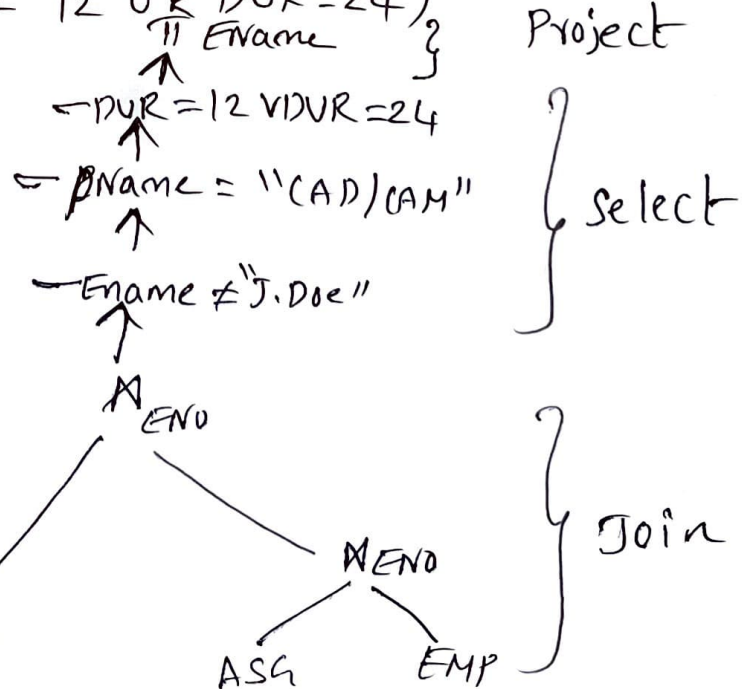
where ASG. ENO = Emp. ENO

AND ASG. PNO = Proj. PNO

AND Ename $\neq$ "J. Doe"

AND Proj. Pname = "CAD/CAM"

AND (DUR = 12 OR DUR = 24)



Data Localization

The input to the second layer is an algebraic query on global relations. The main role of the second layer is to localize the query's data using distribution information in the fragment schema.

→ This layer determines which fragments are involved in the query and transforms the distributed query into a query fragments.

→ A global relation can be reconstructed by applying the fragmentation rules, and then deriving a program called a localization program, of relational algebra operators, which then acts on fragments.

→ Generating a query on fragments is done in two steps

- 1) The query is mapped into a fragment query by substituting each relation by its reconstruction program.
- 2) Second, the fragment query is simplified and restructured to produce another "good" query.

Reduction techniques that generates simpler and optimized queries are

Reduction for Primary Horizontal Fragmentation:

The horizontal fragmentation function distributes a relation on selection predicates.

Ex: Relation Emp (Eno, Ename, Title) can be split into three horizontal fragments Emp₁, Emp₂, Emp₃ defined as follows:

$$\text{Emp}_1 = \sigma_{\text{Eno} \leq "E_3"}(\text{Emp})$$

$$\text{Emp}_2 = \sigma_{E_3 < \text{Eno} \leq "E_6"}(\text{Emp})$$

$$\text{Emp}_3 = \sigma_{\text{Eno} > "E_6"}(\text{Emp})$$

The union of the fragments

$$\text{Emp} = \text{Emp}_1 \cup \text{Emp}_2 \cup \text{Emp}_3$$

Reduction with Selection:-

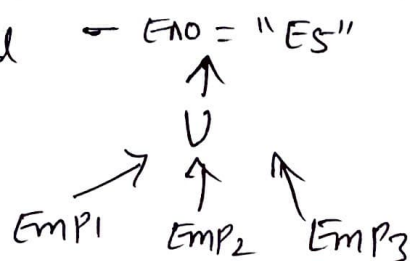
Given a relation R that has been horizontally fragmented as R₁, R₂, ..., R_w, where $R_j = \sigma_{P_j}(R)$, the rule can be stated formally as

$$\text{Rule : } \sigma_{P_i}(R_j) = \emptyset \text{ if } \forall x \text{ in } R : \neg (P_i(x) \wedge P_j(x))$$

where P_i and P_j are selection predicates, x denotes a tuple, and P(x) denotes "predicate P holds for x".

Select * from Emp where Eno = "E5"

a) Localized query



b) $\sigma_{\text{Eno} = "E_5"}$



By Commuting the Selection with the Union operation, it is easy to detect that the Selection Predicate contradicts the predicates of Emp₁ and Emp₃, thereby Producing empty relations. The reduced query is simply applied to Emp₂.

Reduction with Join

Joins on horizontally fragmented relations can be simplified when the joined relations are fragmented according to join attribute. The simplification consists of distributing joins over joins and eliminating useless joins. The distribution of join over Union can be stated as

$$(R_1 \cup R_2) \bowtie S = (R_1 \bowtie S) \cup (R_2 \bowtie S)$$

where R_i are fragments of R and S is a relation.

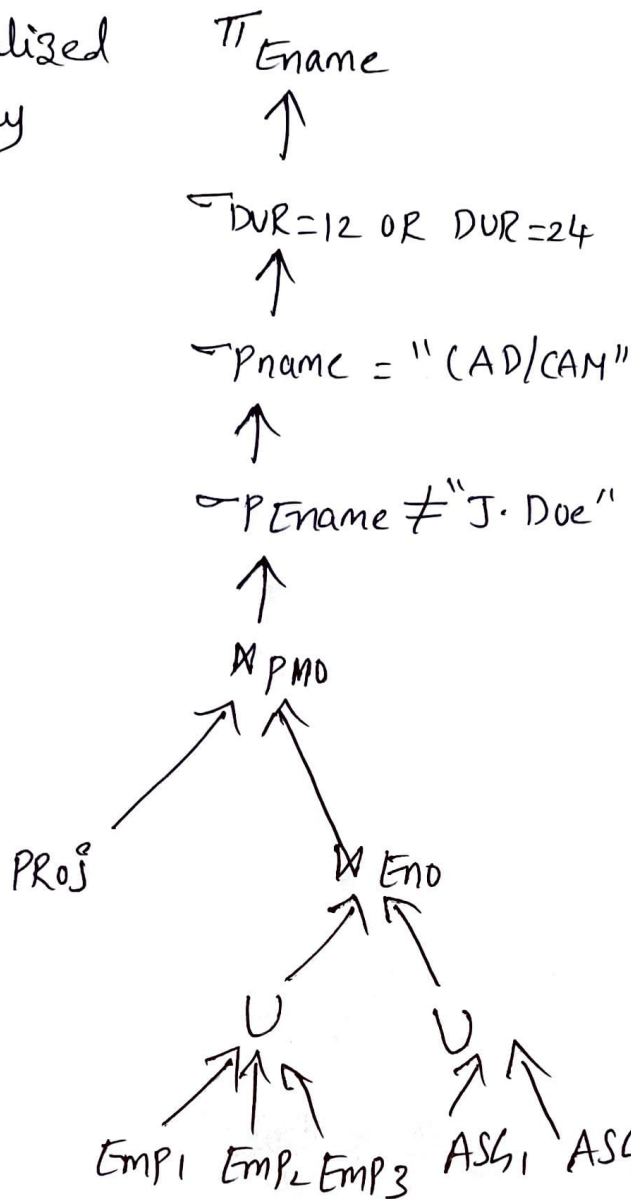
Rule : $R_i \bowtie R_j = \emptyset$ if $\forall x \text{ in } R_i, \forall y \text{ in } R_j :$
 $\neg (P_i(x) \wedge P_j(x))$

Ex: Assume Emp is fragmented as Emp₁, Emp₂, Emp₃ and ASG is fragmented as

$$ASG_1 = \neg_{ENO \leq "E_3"} (ASG)$$

$$ASG_2 = \neg_{ENO > "E_3"} (ASG)$$

a) Localized query

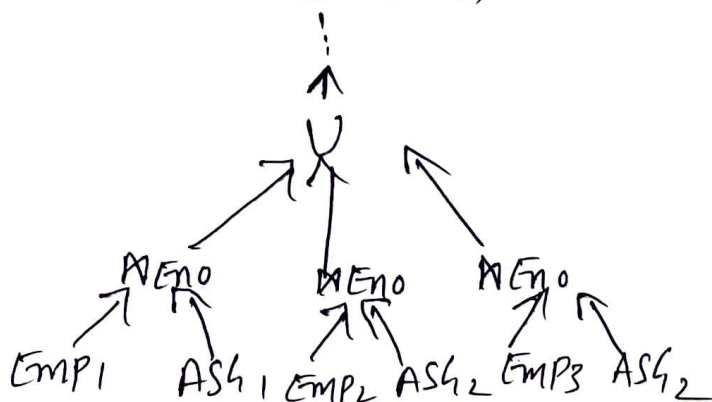


unnecessary work can be eliminated

$EMP_3 \bowtie ASG_1$ gives an empty result

$$EMP_3 = \sigma_{ENO \neq "E_6"}(EMP)$$

$$ASG_1 = \sigma_{ENO \leq "E_3"}(EMP)$$



b) Reduced query

Reduction for Vertical Fragmentation

The vertical fragmentation function distributes a relation based on projection attributes. Since the reconstruction operator for vertical fragmentation is the join, the localization program for a vertically fragmented relation consists of the join of the fragments on the common attribute.

Ex: EmpRelation can be divided into two vertical fragments where the key attribute Eno is duplicated

$$EMP_1 = \pi_{Eno, Ename} (EMP)$$

$$EMP_2 = \pi_{Eno, TITLE} (EMP)$$

The localization program is

$$EMP = EMP_1 \bowtie_{Eno} EMP_2$$

Given Relation R defined over attributes

$A = \{A_1, \dots, A_n\}$ which is vertically fragmented

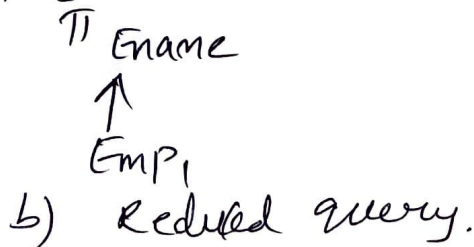
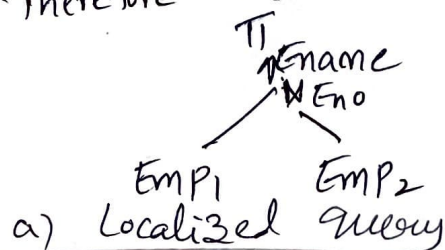
as $R_i = \pi_{A'}(R)$ where $A' \subseteq A$

Rule: $\pi_{D,K}(R_i)$ is useless if the set of Projection attributes D is not in A' .

Select Ename from Emp

EMP_2 is useless because Ename is not in EMP_2 .

Therefore Projection needs to apply only to EMP_1



Reduction for Derived Fragmentation

Derived fragmentation is another way to distributing two relations so that joints processing of select and join is improved.

→ Since tuples of R are placed according to the tuples of S, derived fragmentation should be used only for one-to-many (hierarchical) relationships of the form $S \rightarrow R$.

Ex: ASG (Eno, Pno, Resp, DUR) can be indirectly fragmented

$$ASG_1 = ASG \bowtie_{Eno} Emp_1$$

$$ASG_2 = ASG \bowtie_{Eno} Emp_2$$

$$Emp_1 = \sigma_{Title = "Programmer"} (Emp)$$

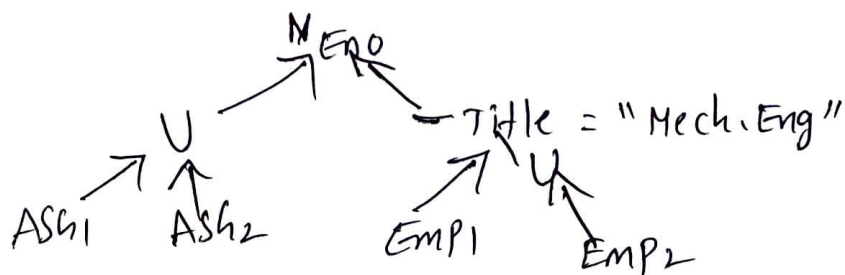
$$Emp_2 = \sigma_{Title \neq "Programmer"} (Emp)$$

$$ASG_1 \bowtie Emp_2 = \emptyset$$

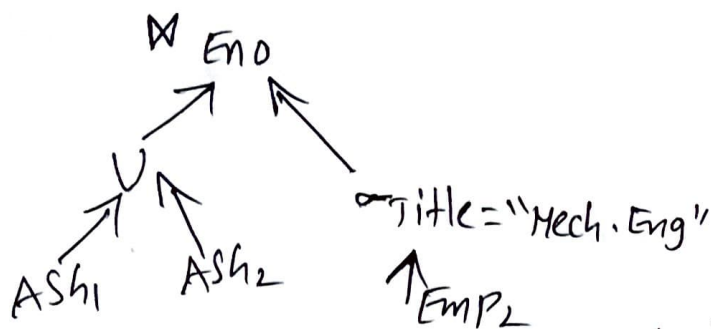
Select * from Emp, ASG

Where ASG.Eno = Emp.Eno

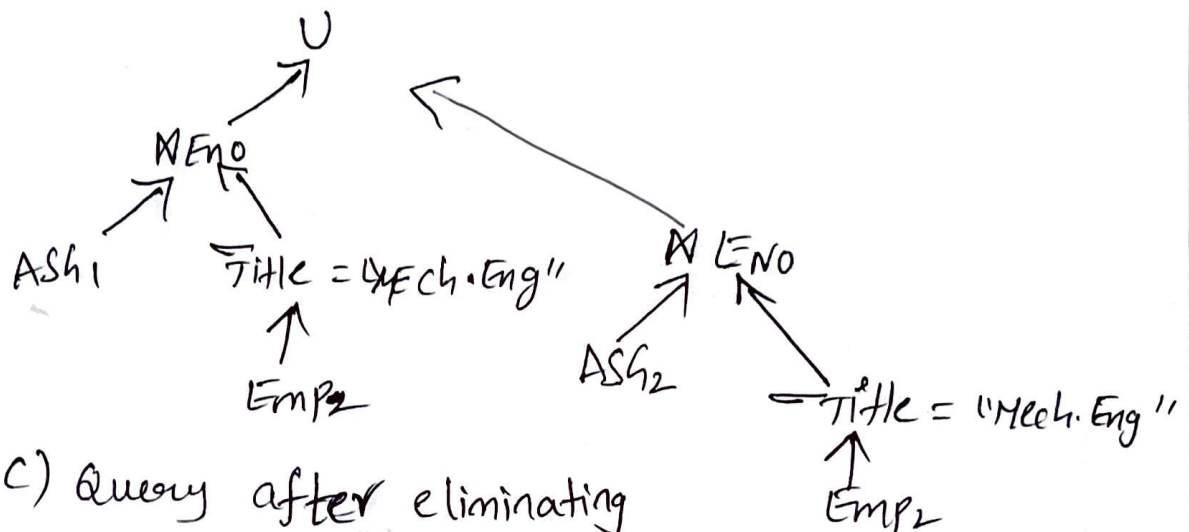
AND Title = "Mech.Eng."



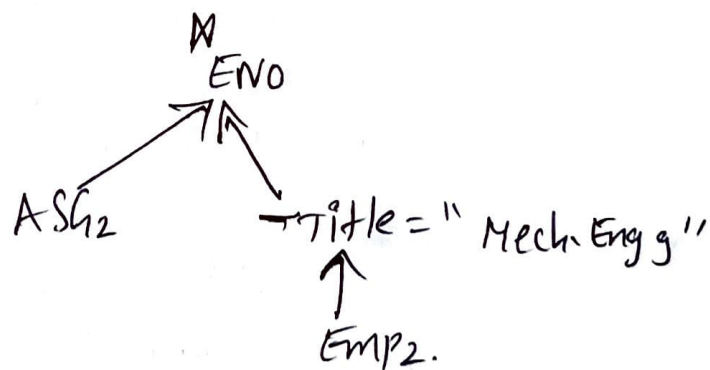
a) Localized query



b) Query after pushing selection down



c) Query after eliminating the left subtree



d) Reduced query after eliminating the left subtree.

Reduction for Hybrid Fragmentation

The goal of hybrid fragmentation is to support, efficiently, queries involving Projection, Selection, and Join.

Hybrid fragmentation based on selection-projection will make selection only, or projection only, less efficient than with horizontal fragmentation.

The localization for a hybrid fragmented relation uses unions and joins of fragments.

$$\underline{\text{Ex:}} \quad \text{Emp}_1 = \sigma_{\text{ENO} \leq "E_4"} (\pi_{\text{ENO}, \text{ENAME}} (\text{Emp}))$$

$$\text{Emp}_2 = \sigma_{\text{ENO} > "E_4"} (\pi_{\text{ENO}, \text{ENAME}} (\text{Emp}))$$

$$\text{Emp}_3 = \pi_{\text{ENO}, \text{TITLE}} (\text{Emp})$$

Localization Pgm is

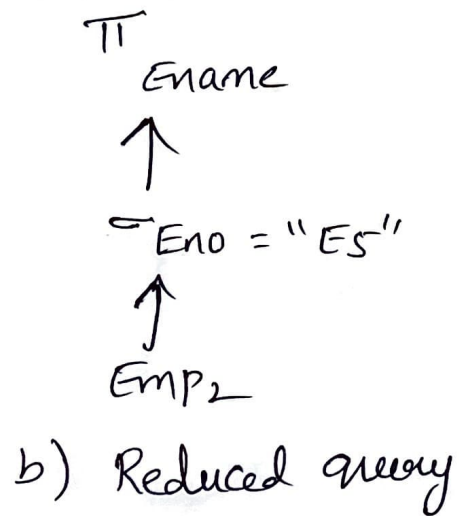
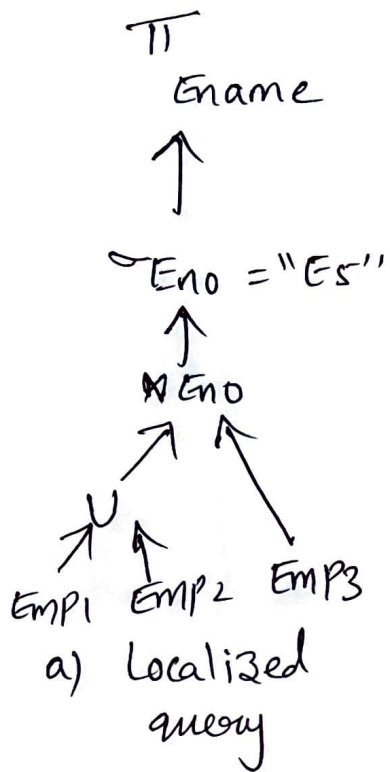
$$\text{Emp}_4 = (\text{Emp}_1 \cup \text{Emp}_2) \bowtie_{\text{ENO}} \text{Emp}_3$$

Queries on hybrid fragments can be reduced by combining the rules used, respectively in primary horizontal, vertical and derived horizontal fragmentation.

- 1) Remove empty relations generated by contradicting selection on horizontal fragments.
- 2) Remove useless relations generated by projections on vertical fragments.
- 3) Distribute joins over unions in order to isolate and remove useless joins.

Select ENAME from Emp

Where ENO = "E5"

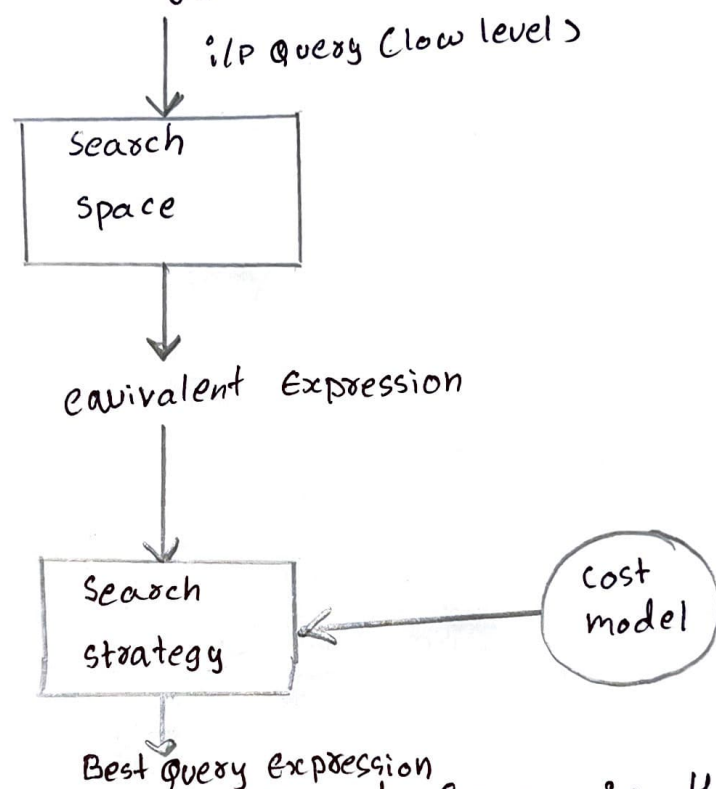


Query optimization

- The input to the third layer is an algebraic query or fragments. The goal of query optimization is to find an execution strategy for the query which is close to optimal.
- The previous layers have already optimized the query, for example by eliminating redundant expressions, this optimization is independent of fragment characteristics such as fragment allocation and cardinalities.
- Query optimization consists of finding the "best" ordering of operators in the query, including communication operation that minimizes a cost function.
- The output of query optimization layer is a optimized algebraic query with communication operators included in fragments. It is typically represented and saved as a distributed query execution plan.

A query optimizer, the software module that performs query optimization, is usually seen as consisting of three components.

- 1) Search space
- 2) Cost model
- 3) Search strategy.



Search Space: The Search Space is the set of alternative execution plans that represent the input query.

→ These plans are equivalent, in the sense that they yield the same result, but they differ in the execution order, of the operations and way these operations are implemented.

→ The Search space is obtained by applying transformation rules.

2) The Cost model:- Predicts the cost of a given execution plan. To be accurate, the cost model must have good knowledge about the distributed execution environment.

3) The Search strategy:- explores the search space and select the best plan, using the cost model. It defines which plans are examined and in which order. The details of the environment (centralized versus distributed) are captured by the search space and cost model.

Query optimization involves three steps

- 1) Query tree generation
- 2) Query plan generation
- 4) Code generation.

Query tree generation: A query tree is a tree data structure representing a relational algebra expression.

Query plan generation: After query tree is generated, a query plan is made. A query plan is an extended query tree that includes access paths for all operations in the query tree.

Code generation:- Code generation is the final step in query optimization. It is the executable form of the query, whose form depends upon the type of the underlying operating system. Once the query code is generated, the execution Manager runs it and produces the results.

Example

Student

Roll No	Name	Add
1	Rama	China
2	Krishna	Mumbai
3	Suresh	Hyderabad
4	Nareesh	Kolkata

Result

Roll No	Sub id	Marks
1	S ₁	90
2	S ₂	60
3	S ₁	75
4	S ₃	85

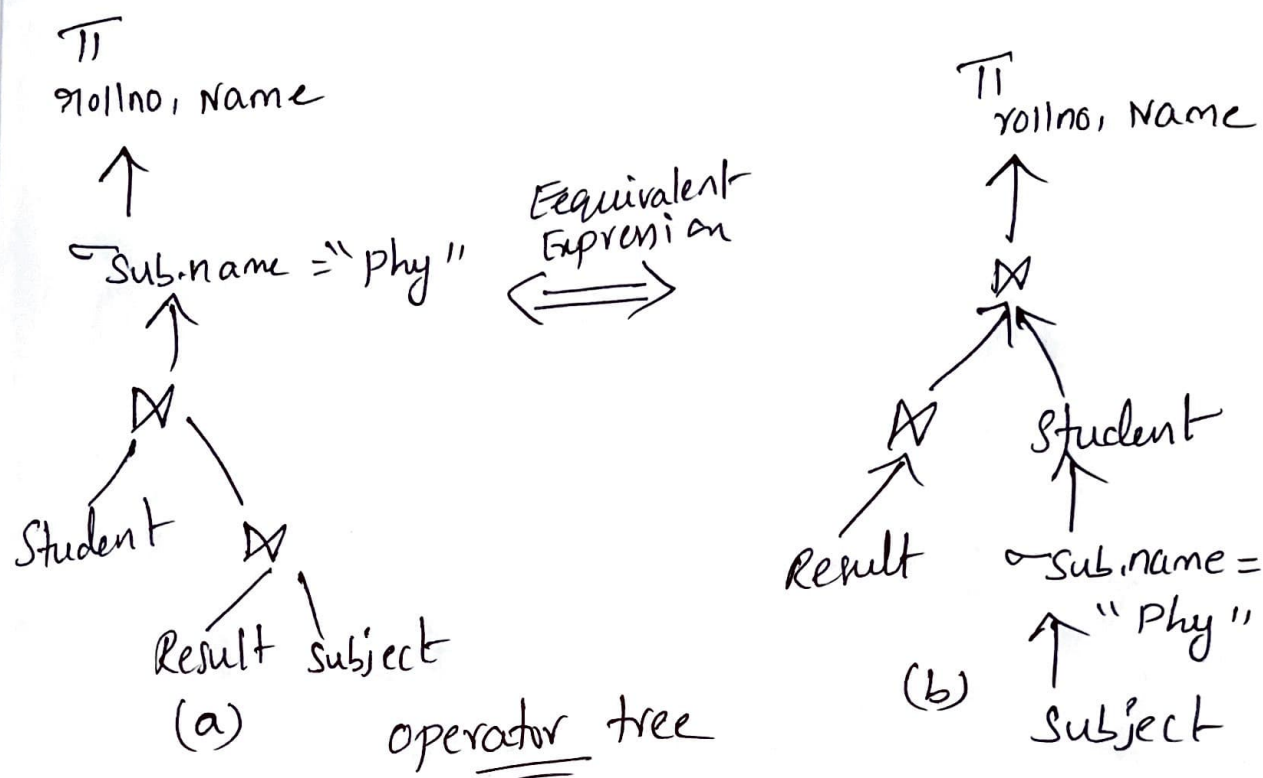
Subject

Sub id	Sub name
S ₁	Phy.
S ₂	Chemistry
S ₃	Maths

→ List the rollno, name of the students who have enrolled in Physics

$\pi_{roll, Name} (\sigma_{Subname = 'phy'} (Student \bowtie Result \bowtie subject))$

This expression consists a large intermediate table, However, we can interested in only a few records of this relation.



Roll No	Name	Add	Sub id	Marks	Subname
1	Rama	China	S1	90	phy
2	Krishna	Mumbai	S2	60	che
03	Suresh	Hyd	S1	75	Phy
04	Nareesh	Kolkata	S2	85	Maths

Student $\bowtie$ Result $\bowtie$ Subject

Now Select Subject name who have enrolled in physics

Roll No	Name	Add	Sub id	Marks	Subname
1	Rama	China	S1	90	Phy
3	Suresh	Hyd	S1	75	Phy

Now Projection of rollno and Name who have enrolled in Subject physics.

Rollno	Name
1	Rama
3	Suresh.

b) 1) select Subject name physics record from subject table

Sub id	Sub name
S ₁	physics

Now perform (X) operation with result table

Roll No	Sub id	Marks	Subname
1	S ₁	90	Phy
3	S ₁	75	Phy

Now perform (X) operation with Student table

Roll No	Name	Sub id	Add	Marks	Sub name
1	Rama	S ₁	china	90	Phy
3	Suresh	S ₁	Hyd	75	Phy

Now perform π rollno, name

Roll No	Name
1	Rama
3	Suresh

Given a relational algebra expression. It is the job of query optimizer to come up with a query Evaluation plan that compute the same result as the given expression and is the least costly way of generating result.

4) Query Execution:

The last layer is performed by all the sites having fragments involved in the query.

→ Each sub query executing alone site, called local query, is then optimized using the local schema of the site and executed.

Centralized Query Optimization:

In this we present the main query optimization techniques for centralized systems.

→ This presentation is a prerequisite to understanding distributed query optimization for three reasons.

→ First, a distributed query is translated to local queries, each of which is processed in a centralized way.

→ Second, distributed query optimization techniques are often extensions of the techniques for centralized systems.

Finally centralized query optimization is a simpler problem, the minimization of communication costs makes distributed query optimization more complex.

In a centralized system, query processing is done with the following aim

→ Minimizing of response time of query.
(Time taken to produce the results to user's query.)

→ Maximize system throughput (the number of requests that are processed in a given amount of time)

→ Reduce the amount of memory and storage required for processing.

→ Increase Parallelism.

Centralized query optimization has divided into 3 approaches

- 1) Static query optimization.
- 2) Dynamic query optimization.
- 3) Hybrid query optimization.

Dynamic Query Optimization

Dynamic query optimization combines two phases of query decomposition and optimization with execution.

→ The QEP is dynamically constructed by the query optimizer which makes calls to the DBMS engine for executing the query's operations. Thus there is no need for cost model.

The most popular dynamic query optimization algorithm is that of INGRES algorithm one of the first relational DBMS.

→ The algorithm recursively breaks up a query expressed in relational calculus (i.e., RCL)

in to smaller pieces which are executed along the way.

→ First the query is decomposed in to a sequence of queries having a unique relation in common, then each mono relation query is processed by selecting based on the Predicate, the best access method to that relation (Eg: Index, Sequential Scan)

Example

INGRES Algorithm

Question : To illustrate the detachment technique, we apply it to the following query.

"Names of employees working on the CAD/CAM Project"

Answer : This query can be expressed in SQL by the following query q_1 on the Engineering database.

```
 $q_1$  : Select Emp. Ename  
      From Emp, ASG, Proj.  
      Where Emp. ENO = ASG. ENO  
      AND ASG. PNO = PROJ. PNO  
      AND Pname = "CAD/CAM"
```


After detachment of the selections, query q_1 is replaced by q_{11} followed by q_1' where JVAR is an intermediate relation.

q_{11} : Select PROJ.PNO INTO JVAR
From PROJ
Where Pname = "CAD/CAM"

q_1' : Select Emp.ename
from Emp, ASG, JVAR
Where Emp.ENO = ASG.ENO
AND ASG.PNO = JVAR.PNO

The successive detachments of q_1' may generate

q_{12} : Select ASG.ENO INTO GVAR
from Emp, GVAR
Where ASG.PNO = JVAR.PNO

q_{13} : Select Emp.ename
from Emp, GVAR
Where Emp.ENO = GVAR.ENO.

Note that other subqueries are also possible.

Thus query q_1 has been reduced to the

Subsequent queries $q_{11} \rightarrow q_{12} \rightarrow q_{13}$.

Query q_{11} is monorelation and can be executed.
 $\rightarrow$ However q_{12} and q_{13} are not monorelation
and can not be reduced by detachment.

→ Multirelation queries, which can not be further detached (e.g., q_{12} and q_{13}) are irreducible.

→ A query is irreducible if and if its query graph is a chain with two nodes.

→ Irreducible queries are converted into monorelation queries by tuple substitution.

Example Let us consider the query q_{13}

q_{13} : Select Emp. Ename
from Emp, GVAR
where Emp. ENO = GVAR. ENO

The relation GVAR is over is a single attribute (ENO). Assume that it contains only two tuples : (E_1) and (E_2) . The substitution of GVAR generates two one-relation subqueries.

q_{131} : Select Emp. Ename
from Emp
where Emp. ENO = "E₁"

q_{132} : Select Emp. Ename
from Emp
where Emp. ENO = "E₂"

Then queries may then be executed.

This algorithm works recursively until there remain no more monorelation queries to be processed.

Static Query optimization

Simple (i.e. mono relation) queries are executed according to the best access path.

→ With static query optimization there is a clear separation between the generation of the QEP at compile time and its execution by the DBMS execution engine. Thus an accurate cost model is key to predict the costs of candidate QEP's.

Execute Joins

- Determine the possible ordering of joins.
- Determine the cost of each ordering.
- Choose the join ordering with minimal cost.

For joins, two alternative algorithms

→ Nested Loops

for each tuple of external relation
(cardinality n_1)

for each tuple of internal relation
(cardinality n_2)

Joins two tuples if the join Predicate
is true

end
end

Complexity : $n_1 * n_2$

Merge Join

Sort relation, Merge relations

Complexity : $n_1 + n_2$ if relations are previously sorted and equi join.

Hybrid query optimization?

It is mixer of static and dynamic approaches, the approach is mainly static, but dynamic query optimization may take place, when high difference between predicated and actual sizes is detected.

Distributed query optimization

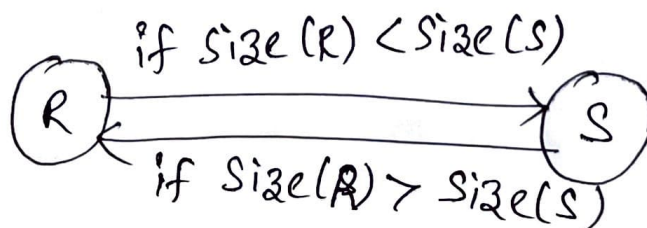
Ordering joins is an important aspect of centralized query optimization. Join ordering in a distributed context is even more important since joins between fragments may increase the communication time.

→ Two basic approaches exist to order joins in distributed queries. One tries to optimize the ordering of joins directly, whereas the other replaces joins by communications of semi-joins in order to minimize communication costs.

→ Direct join ordering of two relations/fragments located at different sites.

→ Move the smaller relation to the other site.

→ We have to estimate the size of R and S.

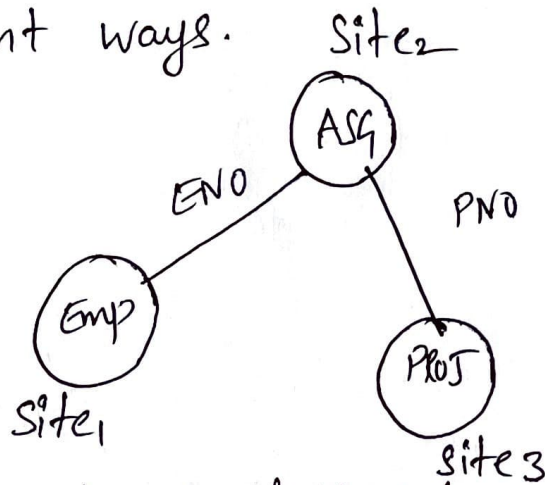


Transfer of operands in Binary operation.

Example

$PROJ \bowtie_{PNO} ASG \bowtie_{ENO} EMP$

This query can be executed in at least five different ways.



Join graph of distributed query.

Let us consider 3 Relations $Emp, ASG, Proj$ which are located at $Site_1, Site_2, Site_3$

1. $Emp \rightarrow Site_2$

Site₂ computes Emp'

$$Emp' = Emp \bowtie ASG$$

$$Emp' \rightarrow Site_3$$

Site₃ computes $Emp' \bowtie PROJ$

2. $ASG \rightarrow Site_1$

Site₁ computes Emp'

$$Emp' = Emp \bowtie ASG$$

$$Emp' \rightarrow Site_3$$

Site₃ computes $Emp' \bowtie Proj$

3. $ASG \rightarrow Site_3$

Site₃ computes $ASG' = ASG \bowtie Proj$

$$ASG' \rightarrow Site_1$$

Site₁ computes $ASG' \bowtie Emp$

4. $Proj \rightarrow Site_2$

Site₂ computes $Proj' = Proj \bowtie ASG$

$Proj' \rightarrow Site_1$, Site₁ computes $Proj' \bowtie Emp$

5). Emp $\rightarrow$ Site₂

Proj $\rightarrow$ Site₂

Site₂ computes Emp $\bowtie$ Proj $\bowtie$ AS4.

To select one of these programs the following sizes must be known or Predicated.

Size (Emp), Size (AS4), Size (Proj)

Size (Emp $\bowtie$ AS4) and Size (AS4 $\bowtie$ Proj)

Semi Join Based Algorithm

Semi Join operation can be used to decrease the total time of join queries.

The join of two relations R and ^{over}S attribute A, stored at sites 1 and 2 respectively, can be computed by replacing one or both operand relations by a semi join with the other relation, using following rules:

$$R \bowtie_A S \Leftrightarrow (R \bowtie_A S) \bowtie_A S$$

$$\Leftrightarrow R \bowtie_A (S \bowtie_A R)$$

$$\Leftrightarrow (R \bowtie_A S) \bowtie_A (S \bowtie_A R)$$

The choice between one of the three semi join strategies requires estimating their respective costs.

Let us compare the costs of the two alternatives

$R \bowtie_A S$ versus $(R \bowtie_A S) \bowtie_A S$ assuming that

$$\text{Size}(R) < \text{Size}(S)$$

cost Analysis

$(R \bowtie_{AS})$ Vs $(R$

R

A	C
1	6
2	7
4	8
5	9

Site₁

S

A	B
1	4
2	5
3	6
3	7

Site₂ R

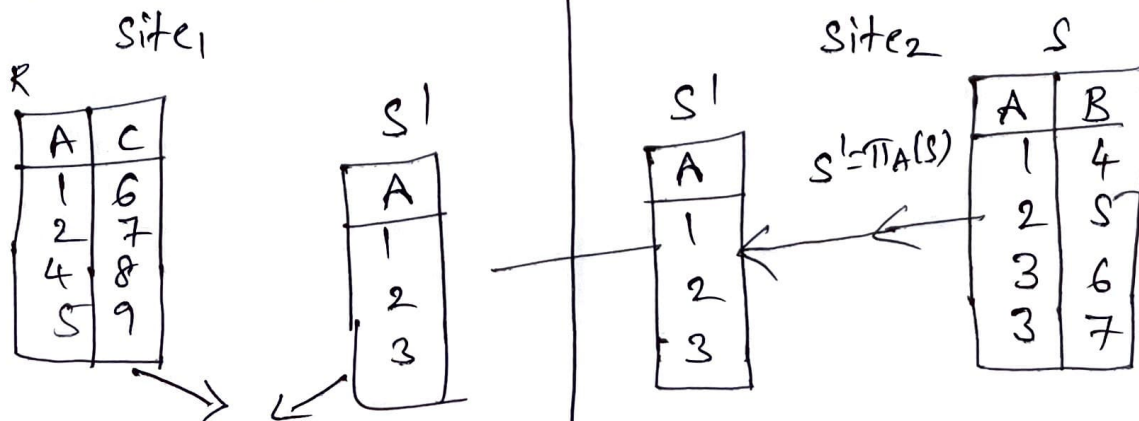
A	C
1	6
2	7
3	6
3	7

$R \bowtie S$

A	B	C
1	4	6
2	5	7

Perform Semi Joins $(R \bowtie_{AS}) \bowtie_{AS} S$

1. $S' = \pi_A(S)$
2. $S' = \text{Site}_1$
3. Site₁ computes $R' = R \bowtie S'$
4. $R' \Rightarrow \text{Site}_2$



$$R = R \bowtie_A S$$

A	C
1	6
2	7

R1

A	C
1	6
2	7

R1 MAS

A	B	C
1	4	6
2	5	7

Semi Join is better if

$$\text{Size}(\pi_A(S)) + \text{Size}(R \bowtie S) < \text{Size}(R)$$

The Semi Join approach is better acts as a sufficient reducer, i.e. if a few tuples of R participate in the join.

Hill Climbing Algorithm

Let us denote initial Strategy as ES_0

→ Then the optimizer splits into two strategies ES_1 followed by ES_2 .

ES_1 : ships are relation of join to the site of the other relation.

ES_2 : These two relations are joined locally and resulting relation is transmitted to the chosen result site.

→ If the cost of executing strategies ES_1 and ES_2 plus the cost of local join processing is less than that of ES_0 , then ES_0 is replaced in the schedule by ES_1 and ES_2

if $\text{cost}(ES_1) + \text{cost}(ES_2) + LC > \text{cost}(ES_0)$

Select ES_0

else select ES_1 and ES_2 .

Example: Let us illustrate the hill-climbing algorithm using the following query involving relations Emp, Pay, Proj and ASG of the Engineering database.

"Find the salaries of engineers who work on the CAD/CAM project"

Schemas

Emp (Eno, Ename, Title)

ASG (Eno, Pno, Resp, Dur)

Proj (Pno, Pname, Budget, Loc)

Pay (Title, Sal)

Statistics

Relation	Size	Site
Emp	8	1
Pay	4	2
Proj	1	3
ASG	10	4

The query in relational algebra is

$\pi_{SAL} (Pay \bowtie_{Title} (Emp \bowtie_{Eno} (ASG \bowtie_{Pno} (\sigma_{Pname = "CAD/CAM"} (Proj))))))$

Alternative 1 : Resulting site is site₁

$$\begin{aligned} \text{Total Cost} &= \text{Cost} (Pay \rightarrow \text{site}_1) + \text{Cost} (ASG \rightarrow \text{site}_1) \\ &\quad + \text{Cost} (Proj \rightarrow \text{site}_1) \\ &= 4 + 10 + 1 = 15 \end{aligned}$$

Alternative 2 : Resulting site is site₂

$$\begin{aligned} \text{Total Cost} &= \text{Cost} (Emp \rightarrow \text{site}_2) + \text{Cost} (Proj \rightarrow \text{site}_3) \\ &\quad + \text{Cost} (ASG \rightarrow \text{site}_2) \\ &= 8 + 1 + 10 = 19 \end{aligned}$$

Alternative 3 : Resulting site is site₃

$$\begin{aligned} \text{Total Cost} &= \text{Cost} (Emp \rightarrow \text{site}_3) + \text{Cost} (Pay \rightarrow \text{site}_2) \\ &\quad + \text{Cost} (ASG \rightarrow \text{site}_3) \\ &= 8 + 4 + 10 = 22 \end{aligned}$$

Alternative 4 : Resulting site is site4

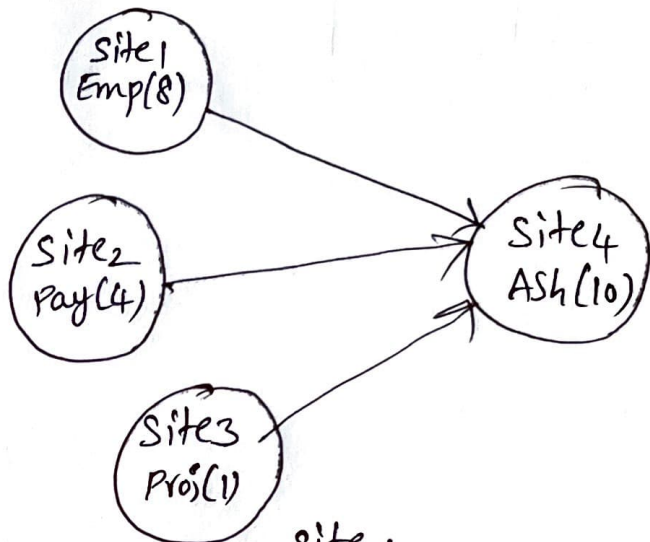
$$\begin{aligned} \text{Total cost} &= \text{cost}(\text{Emp} \rightarrow \text{site}_4) + \text{cost}(\text{pay} \rightarrow \text{site}_4) \\ &\quad + \text{cost}(\text{proj} \rightarrow \text{site}_4) \\ &= 8 + 4 + 1 = 13 \end{aligned}$$

Therefore

$$ES_0 = \text{Emp} \rightarrow \text{site}_4,$$

$$\text{pay} \rightarrow \text{site}_4$$

$$\text{proj} \rightarrow \text{site}_4$$



Candidate Split Alternative 1

$$ES_1 : \text{Emp} \rightarrow \text{site}_2$$

$$ES_2 : (\text{Emp} \bowtie \text{pay}) \rightarrow \text{site}_4$$

$$ES_3 : \text{proj} \rightarrow \text{site}_4$$

$$\text{Total Cost } (ES_1) = 8 + 8 + 1 = 17$$

Ignoring LC length of tuple is 1 for all relation

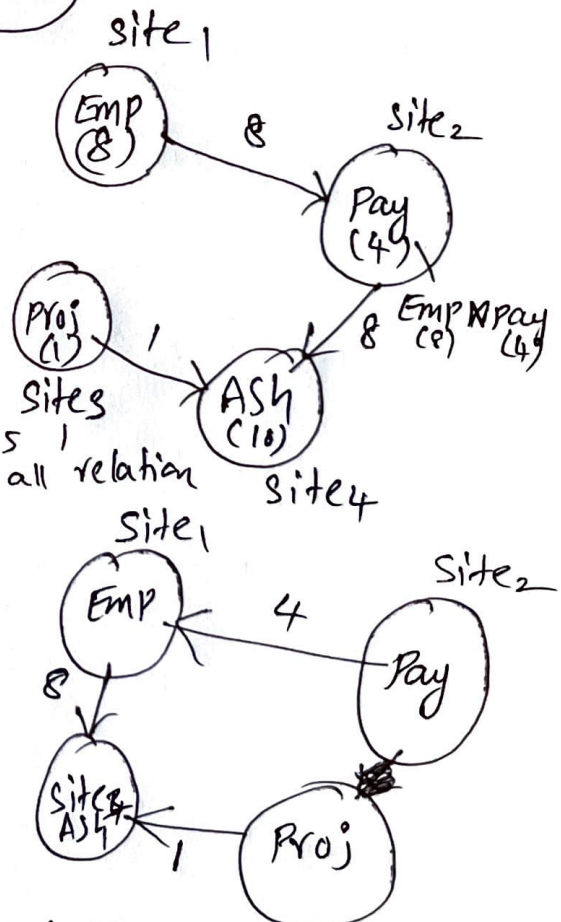
Alternative 2

$$ES_1 : \text{pay} \rightarrow \text{site}_1$$

$$ES_2 : (\text{pay} \bowtie \text{emp}) \rightarrow \text{site}_4$$

$$ES_3 : \text{proj} \rightarrow \text{site}_4$$

$$T.C (ES'') = 4 + 8 + 1 = 13$$



Since the cost of either of the site₃ alternatives is greater than equal to the cost of ES₀. ES₀ is kept as final solution.

A **transaction** is a program including a collection of database operations, executed as a logical unit of data processing. The operations performed in a transaction include one or more of database operations like insert, delete, update or retrieve data.

- **read_item()** – reads data item from storage to main memory.
- **modify_item()** – change value of item in the main memory.
- **write_item()** – write the modified value from main memory to storage.

Transaction Operations

The low level operations performed in a transaction are –

- **begin_transaction** – A marker that specifies start of transaction execution.
- **read_item or write_item** – Database operations that may be interleaved with main memory operations as a part of transaction.
- **end_transaction** – A marker that specifies end of transaction.
- **commit** – A signal to specify that the transaction has been successfully completed in its entirety and will not be undone.

- **rollback** – A signal to specify that the transaction has been unsuccessful and so all temporary changes in the database are undone. A committed transaction cannot be rolled back.

Desirable Properties of Transactions

Any transaction must maintain the ACID properties, viz. Atomicity, Consistency, Isolation, and Durability.

- **Atomicity** – This property states that a transaction is an atomic unit of processing, that is, either it is performed in its entirety or not performed at all. No partial update should exist.
- **Consistency** – A transaction should take the database from one consistent state to another consistent state. It should not adversely affect any data item in the database.
- **Isolation** – A transaction should be executed as if it is the only one in the system. There should not be any interference from the other concurrent transactions that are simultaneously running.
- **Durability** – If a committed transaction brings about a change, that change should be durable in the database and not lost in case of any failure.

States of a transaction

Active: Initial state and during the execution

Partially committed: After the final statement has been executed

Committed: After successful completion

Failed: After the discovery that normal execution can no longer proceed

Aborted: After the transaction has been rolled back and the DB restored to its state prior to the start of the transaction. Restart it again or kill it.

Types of Transactions:-

A client transaction becomes distributed. if it invokes operations in several different servers.

There are 2 different ways that distributed transactions can be structured.

1. Flat transaction.
2. Nested transaction.

1. Flat Transaction :

In a flat transaction a client makes request to more than one server.

A flat transaction completes each of its requests before going on to the next one.

Sequential access to objects.

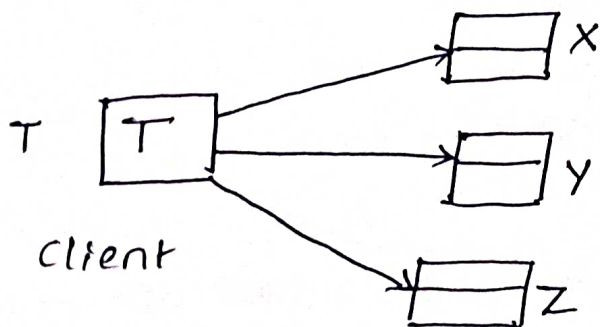


Figure:- Flat Transaction.

Nested Transaction:-

- The top level transactions can open sub transactions and each sub-transaction can open further sub-transaction.
- Sub transactions at the same level can run concurrently.
- Arranged in levels. top level can open sub transactions.
- Any depth of nesting.
- Objects in different servers can be invoked in Parallel.
- Better performance.

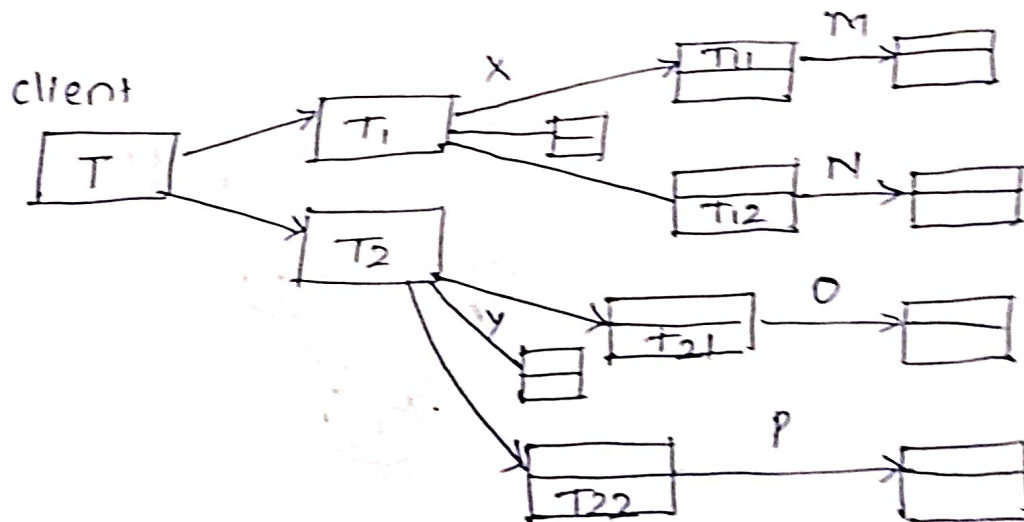


Figure:- Nested Transaction.

1. Concurrency Control

Concurrent Execution in DBMS

1. In a multi-user system, multiple users can access and use the same database at one time, which is known as the concurrent execution of the database.
2. While working on the database transactions, there occurs the requirement of using the database by multiple users for performing different operations, and in that case, concurrent execution of the database is performed.
3. The thing is that the simultaneous execution that is performed should be done in an interleaved manner, and no operation should affect the other executing operations, thus maintaining the consistency of the database.



1. Concurrency Control

Concurrency Control

Concurrency Control is the working concept that is required for controlling and managing the concurrent execution of database operations and thus avoiding the inconsistencies in the database. Thus, for maintaining the concurrency of the database, we have the concurrency control protocols.

Concurrency Control Protocols

Lock Based Concurrency Control Protocol

Time Stamp Concurrency Control Protocol

Validation Based Concurrency Control Protocol

2.Lock-Based Protocol

In this type of protocol, any transaction cannot read or write data until it acquires an appropriate lock on it. There are two types of lock:

1. Shared lock:

It is also known as a Read-only lock. In a shared lock, the data item can only read by the transaction.

It can be shared between the transactions because when the transaction holds a lock, then it can't update the data on the data item.

2. Exclusive lock:

In the exclusive lock, the data item can be both reads as well as written by the transaction.

This lock is exclusive, and in this lock, multiple transactions do not modify the same data simultaneously.

2.Lock-Based Protocol

1. Simplistic lock protocol:

It is the simplest way of locking the data while transaction.

Simplistic lock-based protocols allow all the transactions to get the lock on the data before insert or delete or update on it. It will unlock the data item after completing the transaction.

2. Pre-claiming Lock Protocol

Pre-claiming Lock Protocols evaluate the transaction to list all the data items on which they need locks.

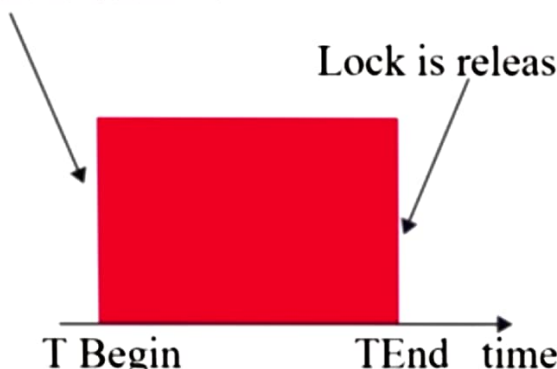
Before initiating an execution of the transaction, it requests DBMS for all the lock on all those data items.

If all the locks are granted then this protocol allows the transaction to begin. When the transaction is completed then it releases all the lock.

If all the locks are not granted then this protocol allows the transaction to rolls back and waits until all the locks are granted.

Lock is attained

Lock is released



T Begin T End time

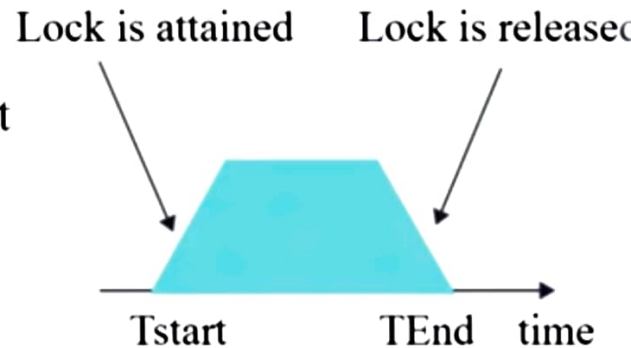
2.Lock-Based Protocol

3. Two-phase locking (2PL)

The two-phase locking protocol divides the execution phase of the transaction into three parts. In the first part, when the execution of the transaction starts, it seeks permission for the lock it requires.

In the second part, the transaction acquires all the locks. The third phase is started as soon as the transaction releases its first lock.

In the third phase, the transaction cannot demand any new locks. It only releases the acquired locks.



3. Two-phase locking (2PL)

There are two phases of 2PL:

Growing phase: In the growing phase, a new lock on the data item may be acquired by the transaction, but none can be released.

Shrinking phase: In the shrinking phase, existing lock held by the transaction may be released, but no new locks can be acquired.

In the below example, if lock conversion is allowed then the following phase can happen:

1. Upgrading of lock (from S(a) to X (a)) is allowed in growing phase.
2. Downgrading of lock (from X(a) to S(a)) must be done in shrinking phase.

3. Timestamp Ordering Protocol

The Timestamp Ordering Protocol is used to order the transactions based on their Timestamps. The order of transaction is nothing but the ascending order of the transaction creation.

The priority of the older transaction is higher that's why it executes first. To determine the timestamp of the transaction, this protocol uses system time or logical counter.

Let's assume there are two transactions T1 and T2. Suppose the transaction T1 has entered the system at 007 times and transaction T2 has entered the system at 009 times. T1 has the higher priority, so it executes first as it is entered the system first. The timestamp ordering protocol also maintains the timestamp of last 'read' and 'write' operation on a data.

3. Timestamp Ordering Protocol

Advantages and Disadvantages of TO protocol:

TO protocol ensures serializability since the precedence graph is as follows:



DBMS Timestamp Ordering Protocol

TS protocol ensures freedom from deadlock that means no transaction ever waits. But the schedule may not be recoverable and may not even be cascade- free.

4.Validation Based Protocol

Validation phase is also known as optimistic concurrency control technique.

1. Read phase:
2. Validation phase
3. Write phase

Start(T_i):

It contains the time when T_i started its execution.

Validation (T_i):

It contains the time when T_i finishes its read phase and starts its validation phase.

Finish(T_i):

It contains the time when T_i finishes its write phase.

$TS(T) = \text{validation}(T).$

Serializability in distributed database :-

In a system with a number of simultaneous transactions, a **schedule** is the total order of execution of operations. Given a schedule S comprising of n transactions, say T1, T2, T3.....Tn; for any transaction Ti, the operations in Ti must execute as laid down in the schedule S.

Types of Schedules

There are two types of schedules –

- **Serial Schedules** – In a serial schedule, at any point of time, only one transaction is active, i.e. there is no overlapping of transactions. This is depicted in the following graph –

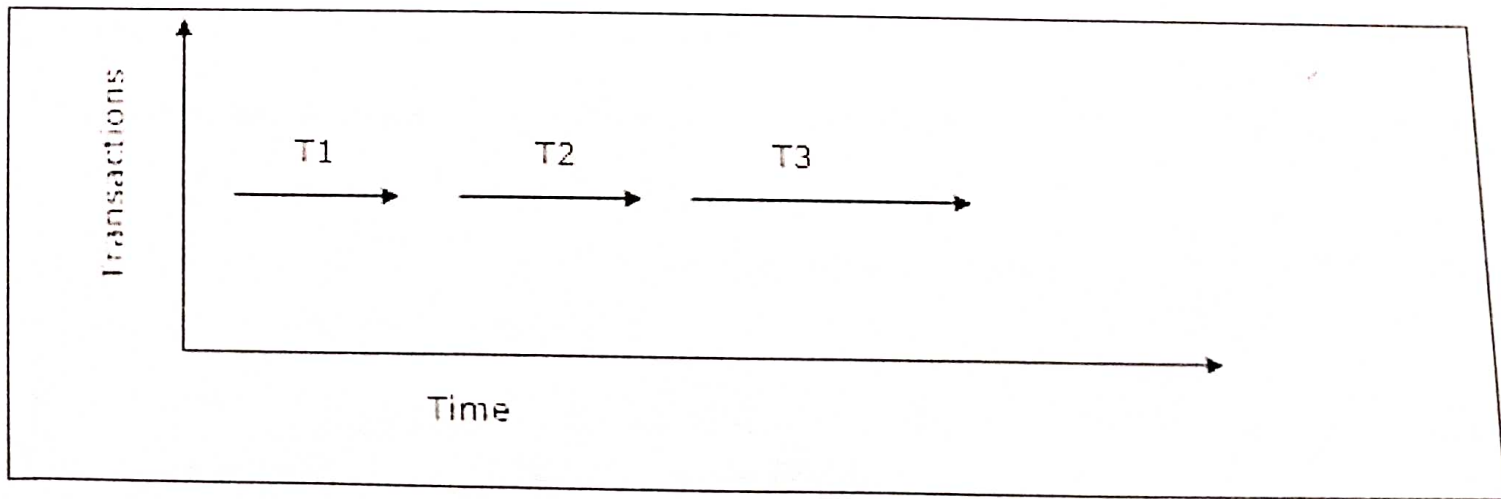


Figure 3.7 Serial Schedules

- **Parallel Schedules** – In parallel schedules, more than one transactions are active simultaneously, i.e. the transactions contain operations that overlap at time. This is depicted in the following graph –

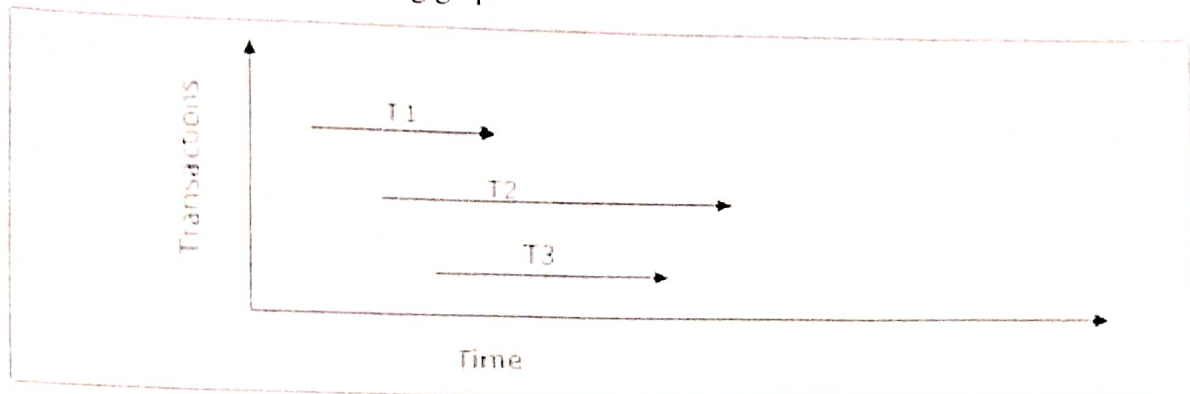


Figure 3.8 Parallel Schedules

Conflicts in Schedules

In a schedule comprising of multiple transactions, a **conflict** occurs when two active transactions perform non-compatible operations. Two operations are said to be in conflict, when all of the following three conditions exists simultaneously –

- The two operations are parts of different transactions.
- Both the operations access the same data item.
- At least one of the operations is a write_item() operation, i.e. it tries to modify the data item.

Serializability

A **serializable schedule** of 'n' transactions is a parallel schedule which is equivalent to a serial schedule comprising of the same 'n' transactions. A serializable schedule contains the correctness of serial schedule while ascertaining better CPU utilization of parallel schedule.

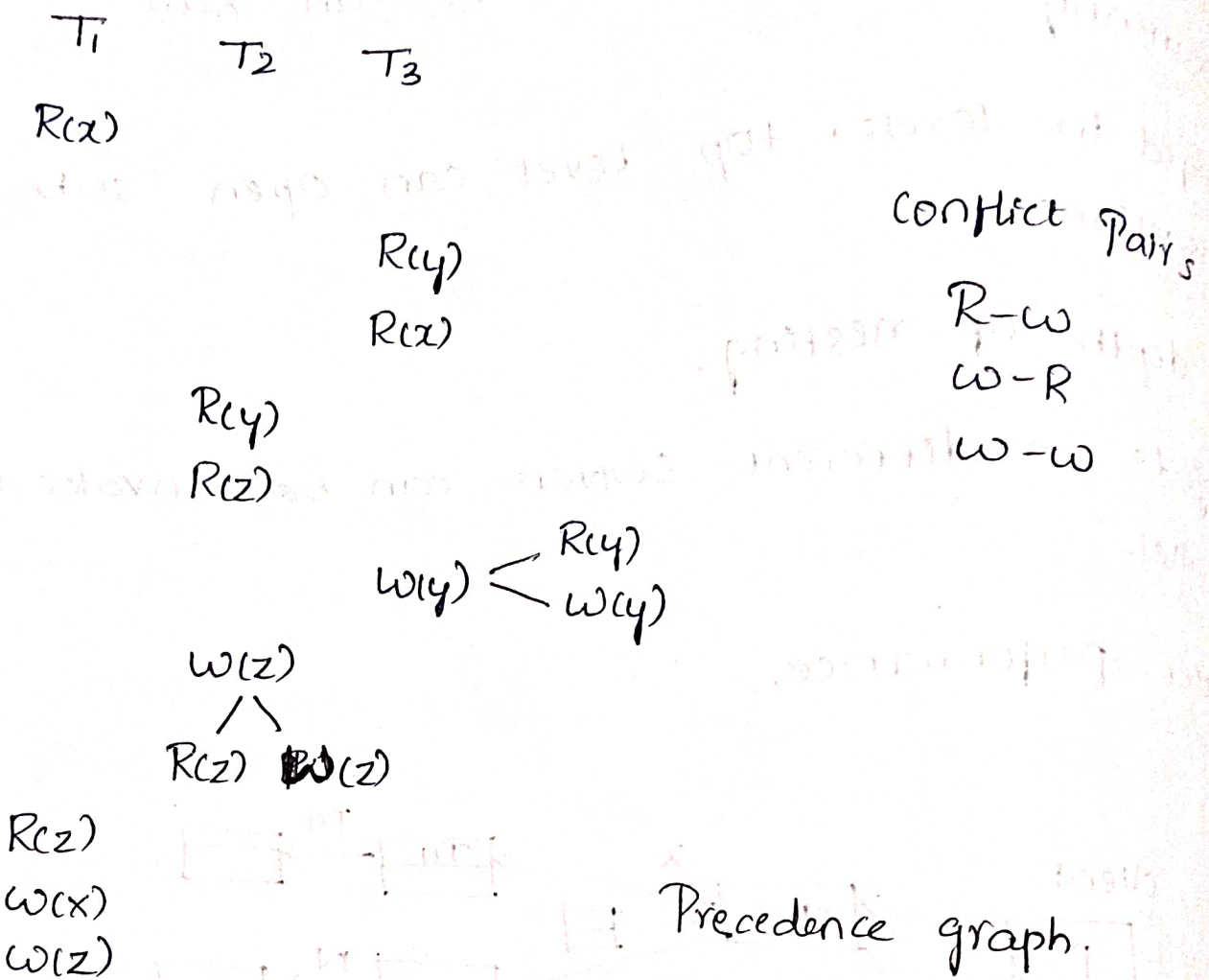
Equivalence of Schedules

Equivalence of two schedules can be of the following types ÷

- **Result equivalence** – Two schedules producing identical results are said to be result equivalent.
- **View equivalence** – Two schedules that perform similar action in a similar manner are said to be view equivalent.
- **Conflict equivalence** – Two schedules are said to be conflict equivalent if both contain the same set of transactions and has the same order of conflicting pairs of operations.

Conflict Serializability

To check conflict pairs in other transactions and draw edges.



Possibilities

- $T_1 \rightarrow T_2 \rightarrow T_3$
- $T_1 \rightarrow T_3 \rightarrow T_2$
- $T_2 \rightarrow T_1 \rightarrow T_3$
- $T_2 \rightarrow T_3 \rightarrow T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_1$ (circled)
- $T_3 \rightarrow T_1 \rightarrow T_2$
- $T_3 \rightarrow T_2 \rightarrow T_1$

No Cycle No loop

↓
conflict serializable
Schedule

↓
consistent data

Concurrency Control Techniques

- The concurrency control mechanisms can be divided into two broad classes:
- Pessimistic concurrency control methods
- Optimistic concurrency control methods
- These mechanisms are used to remove read-write and write- write operations conflicts of concurrent transactions.

Concurrency Control

Optimistic concurrency control (OCC) is a concurrency control method that assumes that multiple transactions can complete without affecting each other, and that therefore transactions can proceed without locking the data resources that they affect.

- Before committing, each transaction verifies that no other transaction has modified its data.
- If the check reveals conflicting modifications, the committing transaction rolls back. Optimistic concurrency control was first proposed by H.T. Kung

Concurrency Control

- **Pessimistic concurrency control** (or *pessimistic locking*) is called "pessimistic" because the system assumes the worst
- It assumes that two or more users will want to update the same record at the same time, and then prevents that possibility by locking the record, no matter how unlikely conflicts actually are.
- The locks are placed as soon as any piece of the row is accessed, making it impossible for two or more users to update the row at the same time.

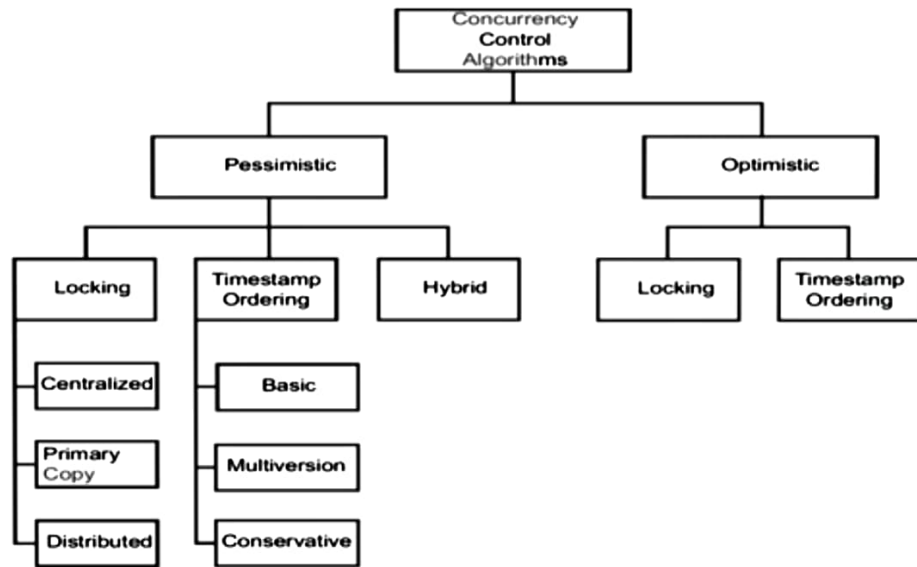


Fig. 11.4 Classification of Concurrency Control Algorithms

Locking-Based Concurrency Control Algorithms

- Ensures that a data item that is shared by conflicting operations is accessed by one operation at a time.
- This is accomplished by associating a “lock” with each lock unit
- This lock is set by a transaction before it is accessed and is reset at the end of its use.
- A lock unit cannot be accessed by an operation if it is already locked by another.

Locking-Based Algorithms

- Transactions indicate their intentions by requesting locks from the scheduler (called **lock manager**).
- Locks are either **read lock** (*rl*) [also called **shared lock**] or **write lock** (*wl*) [also called **exclusive lock**]
- Read locks and write locks conflict (because Read and Write operations are incompatible)
- Locking works nicely to allow concurrent processing of transactions.

Locks Modes

Two locks modes (a) shared (read) and (b) exclusive (write).

- Shared mode: shared lock (X). More than one transaction can apply share lock on X for reading its value but no write lock can be applied on X by any other transaction.
- Exclusive mode: Write lock (X). Only one write lock on X can exist at any time and no shared lock can be applied by any other transaction on X.

Conflict matrix

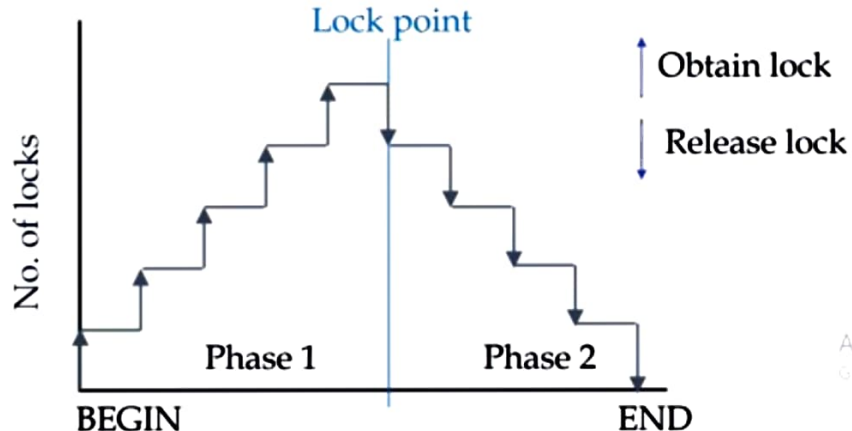
	Read	Write
Read	Y	N
Write	N	N

Activate Windows

Go to Settings to activate Windows

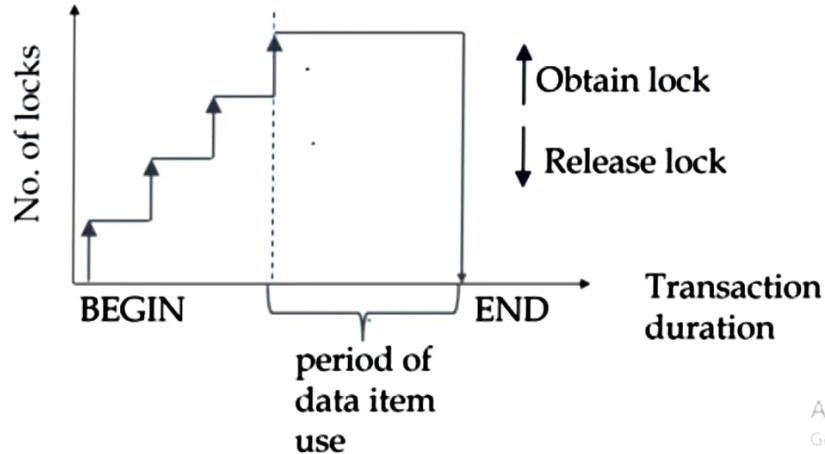
Two-Phase Locking (2PL)

- ❶ A Transaction locks an object before using it.
- ❷ When an object is locked by another transaction, the requesting transaction must wait.
- ❸ When a transaction releases a lock, it may not request another lock.



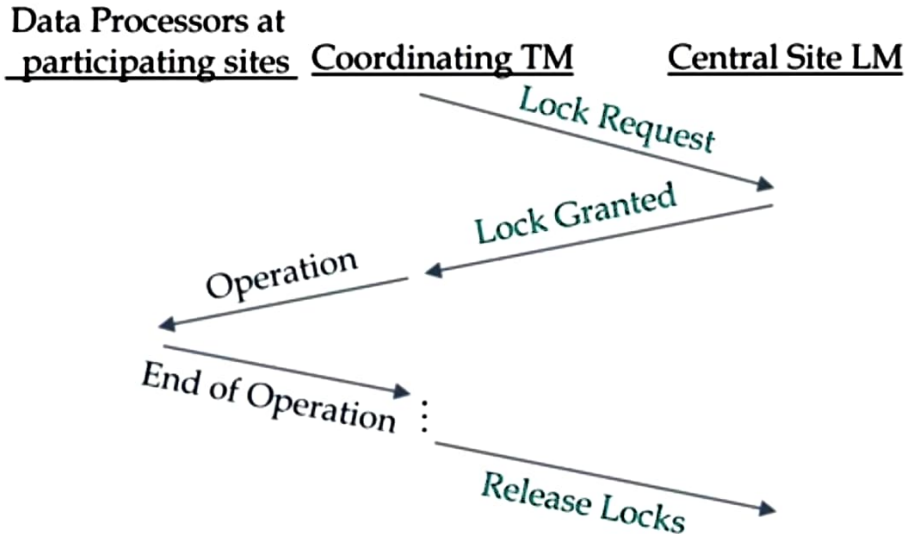
Strict 2PL

Hold locks until the end.



Centralized 2PL

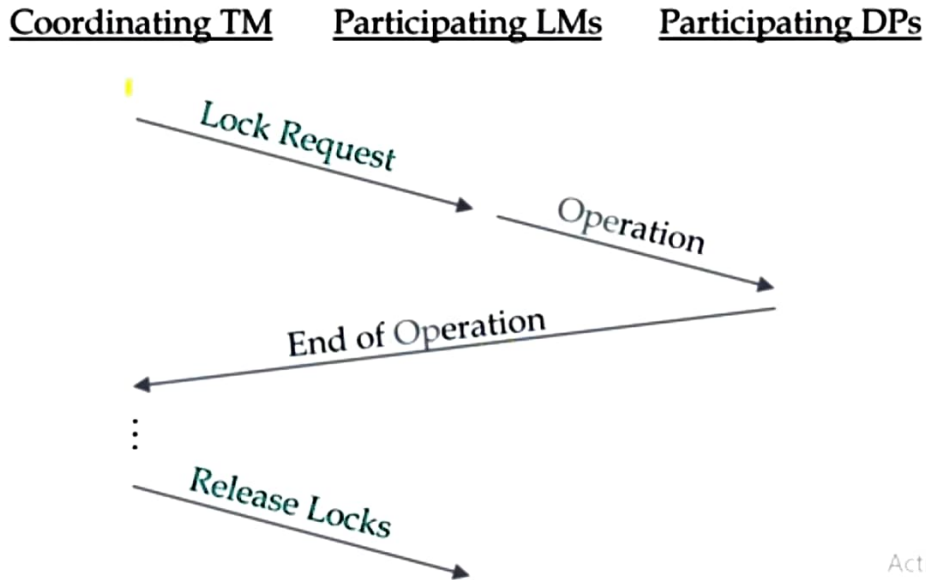
- There is only one 2PL scheduler in the distributed system.
- Lock requests are issued to the central scheduler.



Distributed 2PL

- 2PL schedulers are placed at each site. Each scheduler handles lock requests for data at that site.
- A transaction may read any of the replicated copies of item x , by obtaining a read lock on one of the copies of x . Writing into x requires obtaining write locks for all copies of x .

Distributed 2PL Execution



Time and time stamps in a distributed database

Timestamp is a unique identifier created by the DBMS to identify a transaction. They are usually assigned in the order in which they are submitted to the system. Refer to the timestamp of a transaction T as $TS(T)$. For basics of Timestamp you may refer here.

Timestamp Ordering Protocol –

The main idea for this protocol is to order the transactions based on their Timestamps. A schedule in which the transactions participate is then serializable and the only *equivalent serial schedule permitted* has the transactions in the order of their Timestamp Values. Stating simply, the schedule is equivalent to the particular *Serial Order* corresponding to the *order of the Transaction timestamps*. Algorithm must ensure that, for each items accessed by *Conflicting Operations* in the schedule, the order in which the item is accessed does not violate the ordering. To ensure this, use two Timestamp Values relating to each database item X .

- $W_TS(X)$ is the largest timestamp of any transaction that executed **write(X)** successfully.
- $R_TS(X)$ is the largest timestamp of any transaction that executed **read(X)** successfully.

Time Stamp Ordering Protocol

- Unique Value assign to every transaction.
- Tells the order (when they enters in to the system)
- Read-Ts (RTS) = last (latest) transaction no which perform read successfully.
- Write-Ts (WTS) = last (latest) transaction no which perform write successfully.

10:00

T₁

100

Older

10:10

T₂

200

Younger

10:15

T₃

300

Youngest

∴ $\boxed{Ts(T_1)}$

T ₁	T ₂	T ₃
R(A)		
	R(A)	
		R(A)

∴ $\boxed{RTS(A) = 30}$

T ₁	T ₂	T ₃
W(A)		
		W(A)
	W(A)	

∴ $\boxed{WTS(A) = 20}$

Timestamp Concurrency Control Algorithms

Timestamp-based concurrency control algorithms use a transaction's timestamp to coordinate concurrent access to a data item to ensure serializability. A timestamp is a unique identifier given by DBMS to a transaction that represents the transaction's start time.

These algorithms ensure that transactions commit in the order dictated by their timestamps. An older transaction should commit before a younger transaction, since the older transaction enters the system before the younger one.

Timestamp-based concurrency control techniques generate serializable schedules such that the equivalent serial schedule is arranged in order of the age of the participating transactions.

Some of timestamp based concurrency control algorithms are –

- Basic timestamp ordering algorithm.
- Conservative timestamp ordering algorithm.
- Multiversion algorithm based upon timestamp ordering.

Timestamp based ordering follow three rules to enforce serializability –

- **Access Rule** – When two transactions try to access the same data item simultaneously, for conflicting operations, priority is given to the older transaction. This causes the younger transaction to wait for the older transaction to commit first.
- **Late Transaction Rule** – If a younger transaction has written a data item, then an older transaction is not allowed to read or write that data item. This rule prevents the older transaction from committing after the younger transaction has already committed.
- **Younger Transaction Rule** – A younger transaction can read or write a data item that has already been written by an older transaction.

Basic Timestamp Ordering –

Every transaction is issued a timestamp based on when it enters the system. Suppose, if an old transaction T_i has timestamp $TS(T_i)$, a new transaction T_j is assigned timestamp $TS(T_j)$ such that $TS(T_i) < TS(T_j)$. The protocol manages concurrent execution such that the timestamps determine the serializability order. The timestamp ordering protocol ensures that any conflicting read and write operations are executed in timestamp order. Whenever some Transaction T tries to issue a $R_item(X)$ or a $W_item(X)$, the Basic TO algorithm compares the timestamp of T with $R_TS(X)$ & $W_TS(X)$ to ensure that the Timestamp order is not violated. This describes the Basic TO protocol in following two cases.

1. Whenever a Transaction T issues a $W_item(X)$ operation, check the following conditions:
 1. If $R_TS(X) > TS(T)$ or if $W_TS(X) > TS(T)$, then abort and rollback T and reject the operation. else,
 - Execute $W_item(X)$ operation of T and set $W_TS(X)$ to $TS(T)$.
2. Whenever a Transaction T issues a $R_item(X)$ operation, check the following conditions:
 - If $W_TS(X) > TS(T)$, then abort and reject T and reject the operation, else
 - If $W_TS(X) \leq TS(T)$, then execute the $R_item(X)$ operation of T and set $R_TS(X)$ to the larger of $TS(T)$ and current $R_TS(X)$.

Whenever the Basic TO algorithm detects two conflicting operations that occur in incorrect order, it rejects the later of the two operations by aborting the Transaction that issued it. Schedules produced by Basic TO are guaranteed to be *conflict serializable*. Already discussed that using Timestamp, can ensure that our schedule will be *deadlock free*.

One drawback of Basic TO protocol is that it **Cascading Rollback** is still possible. Suppose we have a Transaction T_1 and T_2 has used a value written by T_1 . If T_1 is aborted and resubmitted to the system then, T_2 must also be aborted and rolled back. So the problem of Cascading aborts still prevails.

Let's list the Advantages and Disadvantages of Basic TO protocol:

- Timestamp Ordering protocol ensures serializability
- Timestamp protocol ensures freedom from deadlock as no transaction ever waits.
- But the schedule may *not be cascade free*, and may not even be recoverable.

Optimistic Concurrency Control Algorithm

In systems with low conflict rates, the task of validating every transaction for serializability may lower performance. In these cases, the test for serializability is postponed to just before commit. Since the conflict rate is low, the probability of aborting transactions which are not serializable is also low. This approach is called optimistic concurrency control technique.

In this approach, a transaction's life cycle is divided into the following three phases –

- **Execution Phase** – A transaction fetches data items to memory and performs operations upon them.
- **Validation Phase** – A transaction performs checks to ensure that committing its changes to the database passes serializability test.
- **Commit Phase** – A transaction writes back modified data item in memory to the disk.

This algorithm uses three rules to enforce serializability in validation phase –

Rule 1 – Given two transactions T_i and T_j , if T_i is reading the data item which T_j is writing, then T_i 's execution phase cannot overlap with T_j 's commit phase. T_j can commit only after T_i has finished execution.

Rule 2 – Given two transactions T_i and T_j , if T_i is writing the data item that T_j is reading, then T_i 's commit phase cannot overlap with T_j 's execution phase. T_j can start executing only after T_i has already committed.

Rule 3 – Given two transactions T_i and T_j , if T_i is writing the data item which T_j is also writing, then T_i 's commit phase cannot overlap with T_j 's commit phase. T_j can start to commit only after T_i has already committed.

OPTIMISTIC CONCURRENCY CONTROL

Disadvantages of lock-based Scheme :

- Lock maintenance represents an overhead to systems that do not support concurrent access to shared data
- The use of lock can result in deadlock.
- Deadlock prevention reduces concurrency severely.
- The use of timeout and deadlock detection is not ideal for interactive programs.
- To avoid cascading aborts, locks cannot be released until the end of transaction. This may reduce the potential for concurrency

OPTIMISTIC CONCURRENCY CONTROL

Concurrency control is a concept that is used to address conflicts with the simultaneous accessing or altering of data that can occur with a multi-user system.

Optimistic concurrency control (OCC) is a concurrency control method applied to transactional systems.

The alternative approach is 'optimistic' because it is based on the observation that, in most applications, the likelihood of two clients' transactions accessing the same object is low.

OCC assumes that multiple transactions can frequently complete without interfering with each other.

Each transaction has the following phases:

- Working Phase
- Validation Phase
- Update Phase

OPTIMISTIC CONCURRENCY CONTROL

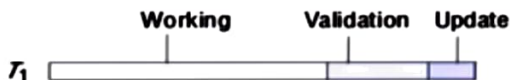
Working Phase: A transaction fetches data items to memory and performs operations upon them.

Each transaction has a tentative version of each of the objects that it updates.

This is a copy of the most recently committed version of the object

Validation Phase: When the `closeTransaction()` request is received, the transaction is validated to establish whether or not its operations on objects conflict with operations of other transactions on the same objects.

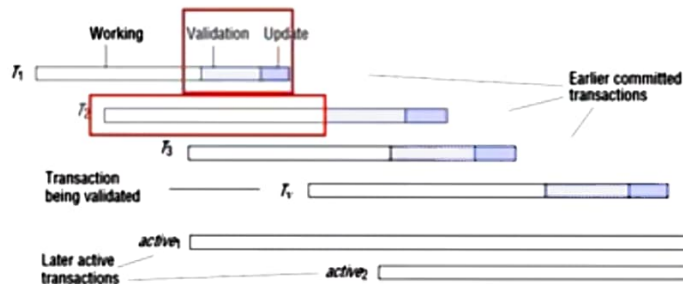
Update Phase: If a transaction is validated, all of the changes recorded in its tentative versions are made permanent.



OPTIMISTIC CONCURRENCY CONTROL

validation rules are

Tv	Ti	Rule
write	read	1. Ti must not read objects written by Tv
read	write	2. Tv must not read objects written by Ti
write	write	3. Ti must not write objects written by Tv and Tv must not write objects written by Ti



Two types of Validation,

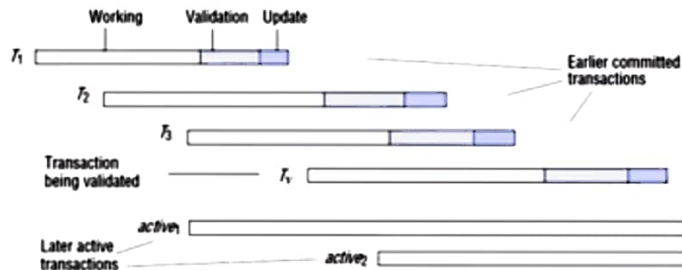
- Backward Validation

- Forward Validation

OPTIMISTIC CONCURRENCY CONTROL

validation rules are

Tv	Ti	Rule
write	read	1. Ti must not read objects written by Tv
read	write	2. Tv must not read objects written by Ti
write	write	3. Ti must not write objects written by Tv and Tv must not write objects written by Ti



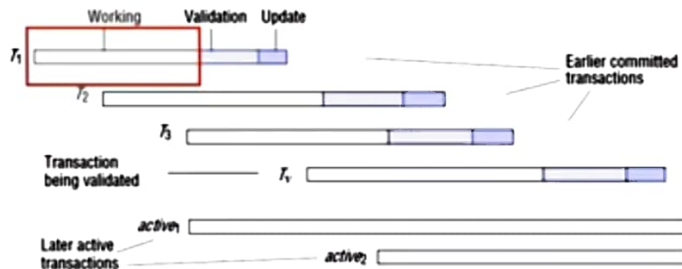
Two types of Validation,

- Backward Validation
- Forward Validation

OPTIMISTIC CONCURRENCY CONTROL

validation rules are

Tv\Ti	Ti	Rule
write	read	1. Ti must not read objects written by Tv
read	write	2. Tv must not read objects written by Ti
write	write	3. Ti must not write objects written by Tv and Tv must not write objects written by Ti



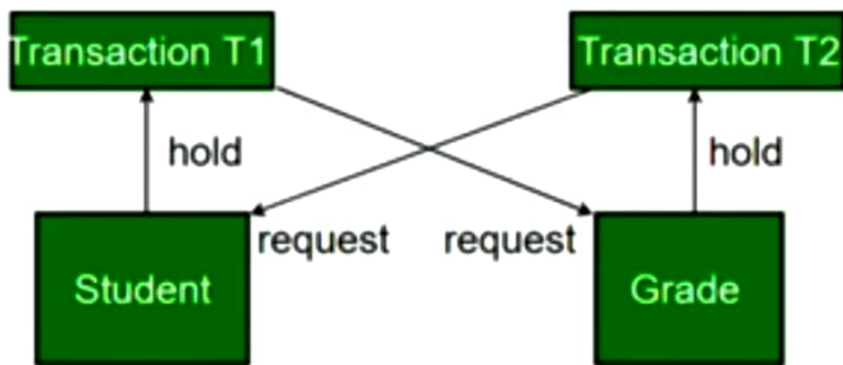
Two types of Validation,

● Backward Validation ← read - write Abort ←

● Forward Validation ← write - read Wait / Abort +

DEADLOCK

- It is a situation in which two or more transactions depend on something.



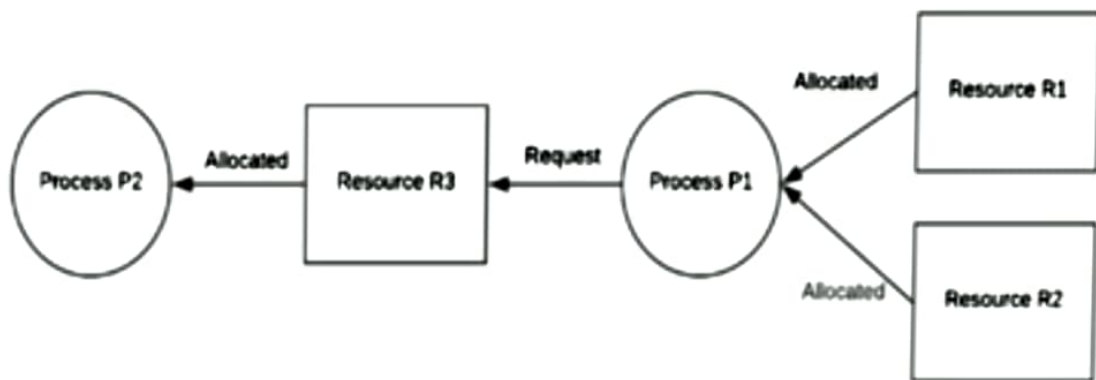
DEADLOCK

- Four conditions for deadlock:
- **Mutual exclusion condition**



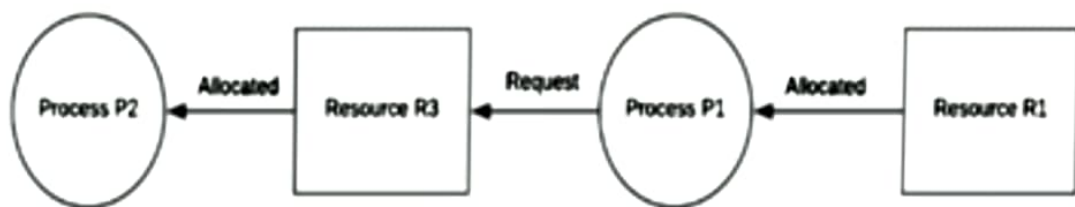
DEADLOCK

- **Hold and Wait condition**



DEADLOCK

- **No preemption condition**



DEADLOCK

- **Circular wait condition:** There must exist a set of processes
[p1, p2, □, pn] such that p1 is waiting for p2, p2 for p3, and so on □.

Deadlock can be handled using two techniques:

1. Deadlock prevention
2. Deadlock detection and deadlock recovery

Two techniques used for deadlock prevention

i) Wait-Die

ii) Wound-Wait

DEADLOCK

Wait/Die

- Older transaction waits and the younger is rolled back and rescheduled.

Wound/Wait

- Older transaction rolls back the younger transaction and reschedules it.

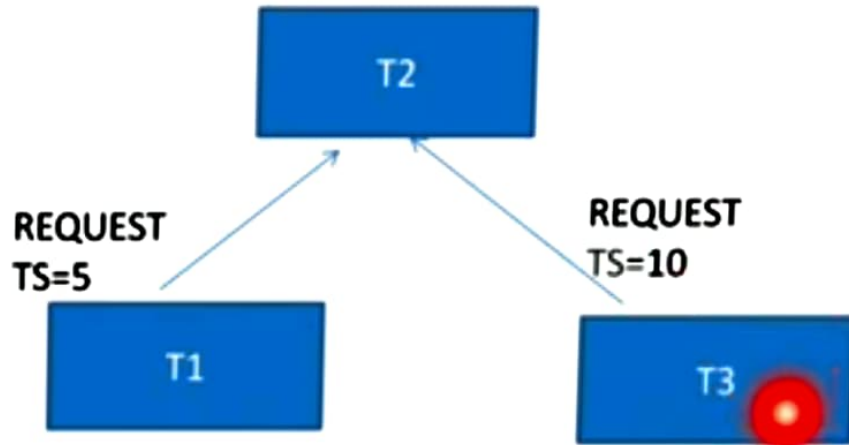
DEADLOCK

Wait/Die and Wound/Wait

Transaction requesting lock	Transaction owning lock	Wait/Die scheme	Wound/Wait scheme
T1(12343465)	T2(23454665)	•T1 waits until the T2 is completed and T2 releases its lock	•T1 preempts (rolls back)T2 •T2 is rescheduled using the same timestamp
T2(23454665)	T1(12343465)	•T2 dies(roll back) •T2 is rescheduled using the same timestamp	•T2 waits until T1 is completed and T1 releases its locks

DEADLOCK

- **Example**



DEADLOCK

- Deadlock Detection
- Deadlock detection is done using **wait for graph**
- This graph is created based on the transaction and their lock.
- If the created graph has a cycle or closed loop, then there is a deadlock.
- $G=(V,E)$
- Example

T1	T2
R(A)	
	R(A)
W(A)	
R(B)	
	W(A)
W(B)	

DEADLOCK

- Three rules for designing the wait-for graph:

1. T1 has read operation and T2 has write operation then draw an edge

T1---->T2

2. T1 has write operation and then T2 has read operation draw an edge

T2---->T1

3. T1 has write operation and T2 has read operation then draw an edge

T1---->T2

Lets draw Wait-for graph

Step1 : Draw vertices for all the transactions



DEADLOCK

Step 2:

T1	T2
R(A)	
	R(A)
W(A)	
R(B)	
	W(A)
W(B)	



DEADLOCK

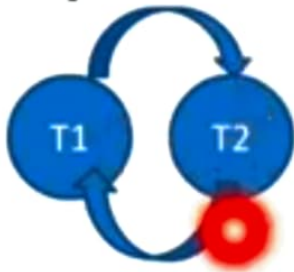
Step 3:

T1	T2
R(A)	
	R(A)
W(A)	
R(B)	
	W(A)
W(B)	



DEADLOCK

- A cycle is detected in the wait for graph.
- Deadlock is present in this transaction scenario.



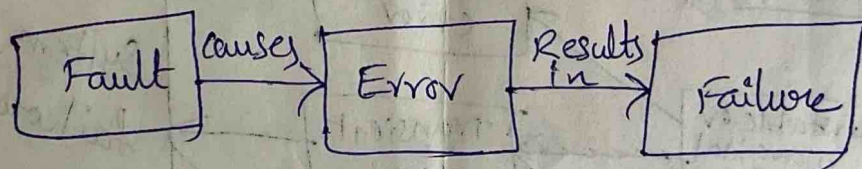
Distributed DBMS Reliability

A reliable dbms is one that can continue to process user requests even when the underlying system is unreliable. In other words, even the components of the distributed computing environment fail, a reliable distributed dbms should be able to continue executing user requests without violating database consistency.

Reliability Components and Measures

System, State, Failure

The part of the state which is incorrect is an error



Chain of Events Leading to System Failure.

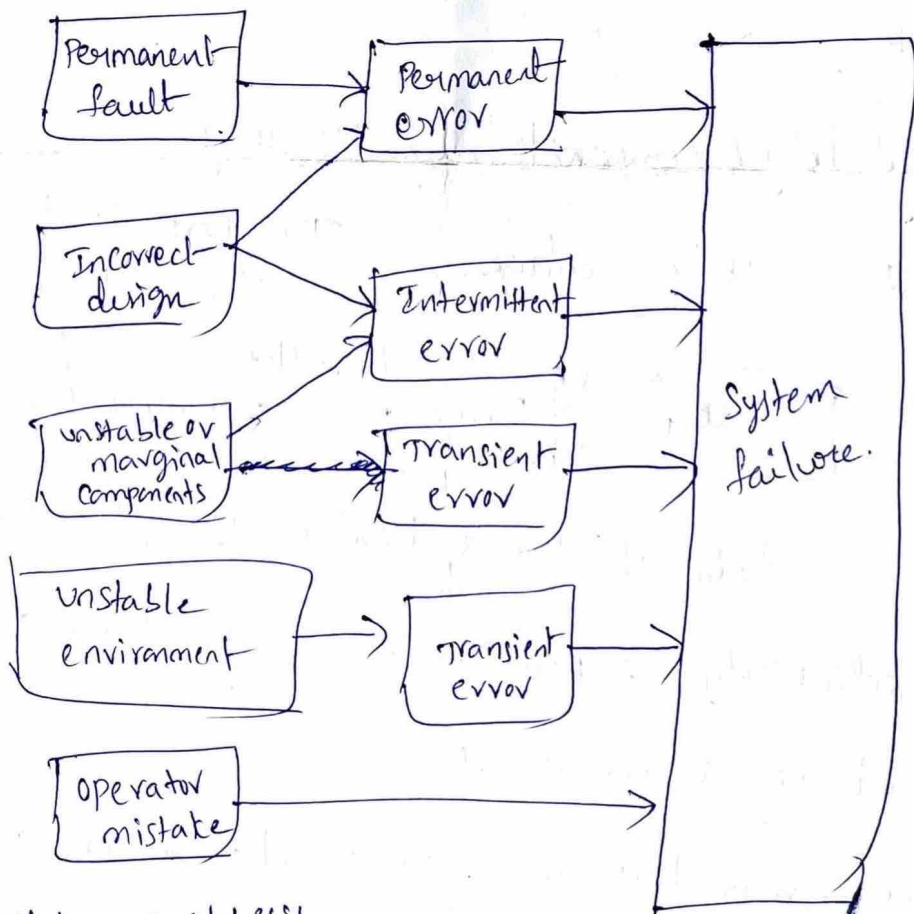
- Reliability refers to a system, that consists of a set of components.
- The system has a state, which changes as the system operates.
- The behaviour of the system in providing response to all the possible external stimuli is laid out in an authoritative specification of its behaviour. The specification indicates the valid behaviour of each system state.

→ Any deviation of a system from the behaviour described in the specification is considered as Failure.

→ We differentiate b/w errors (or faults and failures) that are permanent and those are not permanent.

2 types of faults

- 1) Hard fault → Permanent fault — irreversible change in the behaviour of the system. — cause permanent errors
- 2) Soft fault
↳ Intermittent and transient fault. — cause permanent failure



→ Reliability & Availability

Reliability refers to the probability that the system under consideration does not experience any failures in a given time interval. This is typically used to describe systems

Reliability of a system $R(t)$ is defined as

$$R(t) = \text{Pr} \{ 0 \text{ failures in time } [0, t] \mid \text{no failures at } t=0 \}$$

If we assume failures follow a Poisson distribution this reduces to

$$R(t) = \text{Pr} \{ 0 \text{ failures in time } [0, t] \}$$

Under the same assumptions

$$\text{Pr} \{ k \text{ failures in time } [0, t] \} = \frac{e^{-m(t)} [m(t)]^k}{k!}$$

where $m(t) = \int_0^t z(u) du$. Here $z(t)$ is known as the hazard function $\rightarrow$ which gives the time-dependent failure rate of the specific hardware component.

The Expected (mean) number of failures in time $[0, t]$

$$E[k] = \sum_{k=0}^{\infty} k \frac{e^{-m(t)} [m(t)]^k}{k!} = m(t)$$

Variance as

$$\text{Var}[k] = E[k^2] - (E[k])^2 = m(t)$$

$R(t)$ can be written as

$$R(t) = e^{-m(t)}$$

$$R_{\text{sys}}(t) = \prod_{i=1}^n R(t)$$

Availability ($A(t)$) refers to the probability that the system is operational according to its specification at a given point in time t . Availability refers to systems that can be repaired.

$\rightarrow$ If we assume that failures follow a Poisson distribution with a failure rate λ and that repair time is exponential with a mean repair time of $1/\mu$

$$A = \frac{\mu}{\lambda + \mu}$$

3) Mean Time between Failures / Mean Time to Repair

Mean Time between Failures (MTBF) is the expected time b/w ^{subsequent} failures in a system with repair.

$$\text{MTBF} = \int_0^{\infty} R(t) dt$$

Integral

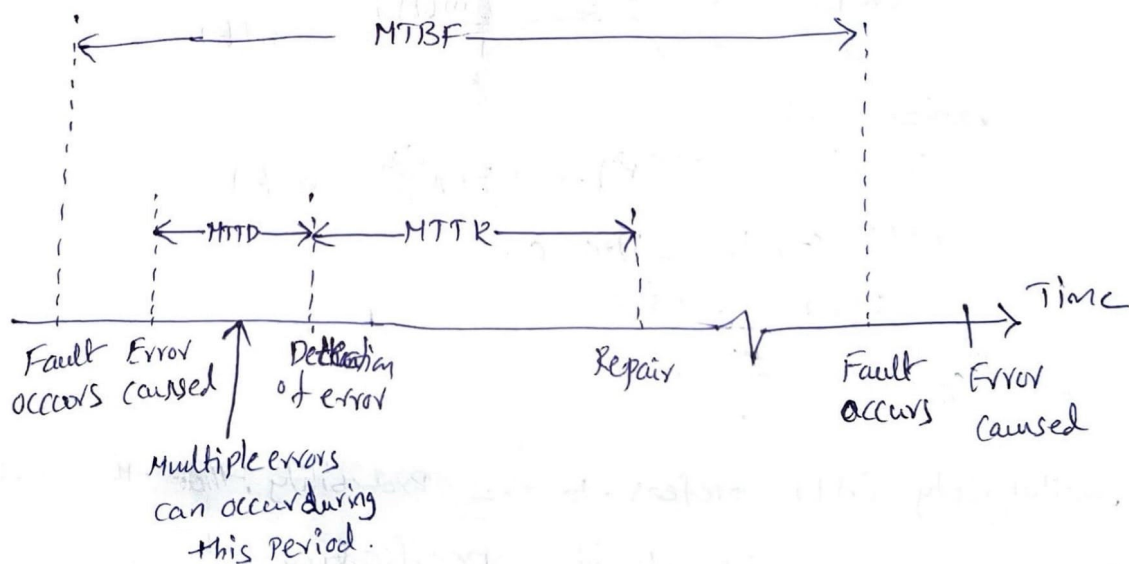
$$A = \frac{\text{MTBF}}{\text{MTBF} + \text{MTTR}}$$

MTTR is the Expected time to repair a failed System.

It is related to the repair rate as MTBF is related to the failure rate.

→ A system failures may be latent, in that a failure is typically detected some time after its occurrence.

This period is called Error Latency, and the average error latency time over a no. of identical systems is called mean time to detect (MTTD).



occurrence of Events over Time

Fault Tolerance in Distributed Systems

→ Ability of a system to continue functioning in the event ^{of a} failure. (Partial Failure)

Fault Tolerance is needed to provide 3 main features to distributed systems.

1) Availability :- Availability is defined as the property where the system is readily available to its use at any time.

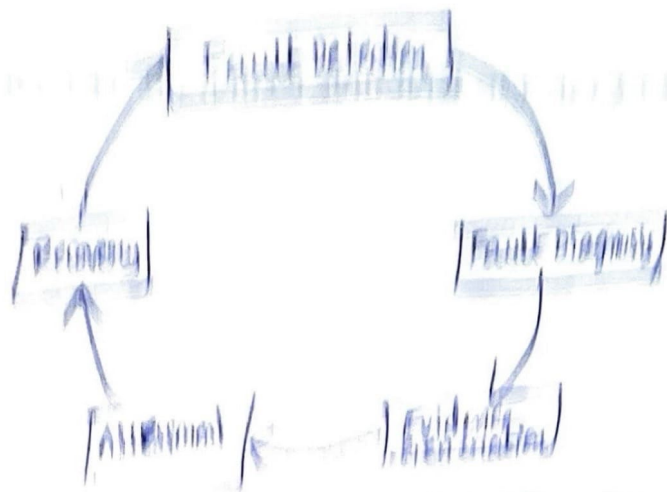
2) Reliability :- Reliability is defined as the property where the system can work continuously without any failure.

3) Safety :- Safety is defined as the property where the system can remain safe from unauthorized access even if any failure occurs.

Example - Patient Monitoring Systems, flight control Systems, Banking services etc.

Below phases carried out for fault tolerance in any distributed systems.

- 1) Fault Detection
- 2) Fault Diagnosis
- 3) Evidence Generation
- 4) Assessment
- 5) Recovery.



1. Fault Detection :- Fault Detection is the first phase where the system is monitored continuously. $\Rightarrow$ The outcomes are being compared with the expected

m/p.

$\Rightarrow$ During Monitoring if any faults can occur due to various reasons such as hardware failure, n/w failure, and s/w issues.

$\Rightarrow$ The main aim of the first phase is to detect the faults as soon as they occur so that work being assigned will not be delayed.

2. Fault Diagnosis :- Fault diagnosis is the process where the fault that is identified in the first phase will be diagnosed properly in order to get the root cause and possible nature of the faults.

• Faults diagnosis can be done manually by administrator or by using automated techniques in order to solve the fault and reformat the given data.

3) Evidence Generation :- Evidence generation is defined as the process where the report of the fault is prepared based on the diagnosis done in an earlier phase.

→ This report involves the details of the causes of the fault, the nature of faults, the solutions that can be used for fixing and other alternatives and preventions that need to be considered.

4) Assessment :- Assessment is the process where the damages caused by the faults are analysed.

→ It can be determined with the help of messages that are being passed from the component that has encountered fault. Based on the assessment further decisions are made.

5) Recovery :- Recovery is the process where the aim is to make the system fault free.

→ It is the step to make the system fault free and restore it to state forward recovery techniques and backup recovery.

→ Some of the recovery techniques such as reconfiguration and resynchronization can be used.

Types of Fault Tolerance in Distributed Systems

1) Hardware Fault Tolerance

3) System Fault Tolerance

2) Software

1) Hardware Fault Tolerance: Hardware Fault tolerance involves keeping a backup plan for hardware devices such as memory, hard disk, CPU and other hardware peripheral devices.

→ Hardware Fault tolerance is a type of fault tolerance that does not examine faults and runtime errors but can only provide hardware backup.

Two approaches ^{used} are

- 1) Fault Marking
- 2) dynamic recovery.

2) Software Fault Tolerance: - S/W Fault Tolerance is a type of fault tolerance where dedicated S/W is used in order to detect invalid output.

→ Fault Tolerance makes use of static and dynamic methods for detecting and providing the solution.

→ S/W Fault Tolerance also consists of additional data points such as recovery rollback and check points.

3) System Fault Tolerance: - System fault tolerance is a type of fault tolerance that consists of a whole system.

→ It has the advantage that it ^{not} only stores the check points but also the memory block, and program check points and detect the errors in appl's automatically.

→ The system fault tolerance is reliable and efficient.

Failures in Distributed DBMS

4 types

- 1) Transaction failures
- 2) Site (System) failures
- 3) Media (disk) failures
- 4) Communication failures.

Transaction Failure: - Transactions can fail for a no. of reasons.

→ Failure can be due to an error in transaction caused by incorrect i/p data. or detection of a present or potential deadlock. ← Reasons

→ Approach to handle such failures is to abort the transaction.

2) Site failures: - The reasons for system failure can be hardware or software failure.

→ Results in loss of main memory, processor failure or power supply, failed site becomes unreachable from other sites.

→ Total failure - refers to the simultaneous failure of all sites in the distributed system.

→ Partial failure: - indicates the failure of only some sites while the other sites remain operational.

Media Failures :- Media Failure refers to the failure of the secondary devices that store the database.

→ Such failure may be due to operating system errors as well as to hardware faults such as head crashes or controller failures.

→ Results in either all or part of the database being inaccessible or destroyed.

Communication failures :-

The above three types of failures are common to both centralized and distributed DBMS.

→ Communication failures however are unique to the distributed case. There are no of types of communication failures. The most common ones are the errors in the messages, improperly ordered messages, lost messages and communication line ~~mess~~ failures.

→ Lost or undelivered messages are typically the consequence of communication line failures or site failures.

→ If a communication line fails, in addition to losing the message in transit, it may also divide the network into two or more disjoint groups. This is called

N/W Partitioning.

→ If the N/W is partitioned, the sites in each partition may continue to operate. In this case, executing transactions that access data stored in multiple partitions becomes a major issue.

Local Reliability Protocols

Buffers are organized in Pages.

Architectural considerations

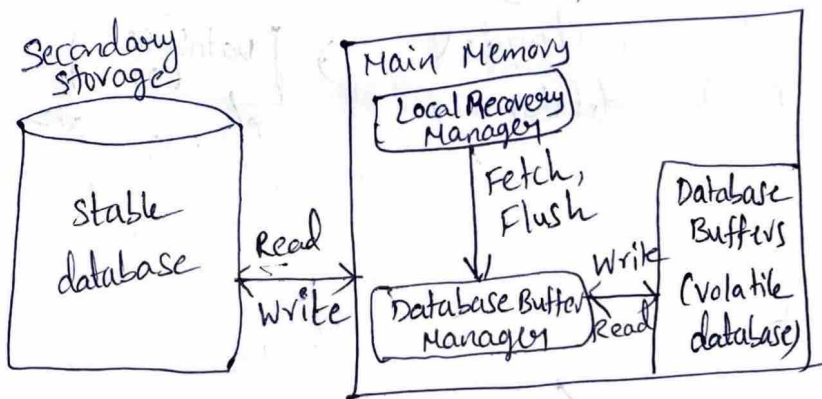
Local Recovery Manager - The execution of the commands ^{Accepted commands} such as begin-transaction, read, write, commit and abort to maintain the atomicity and durability properties of the transactions. (Maintains the atomicity and durability properties of local transactions by performing some functions.)

Volatile Storage

- Consists of the main memory of the computer system (RAM)
- The database kept in volatile storage is volatile db.

Stable Storage :- Stable storage is typically implemented by means of duplexed magnetic disks which store duplicate copies of data that are always kept mutually consistent.

- The version of the db that is kept in stable storage is the stable database. The unit of storage and access of the stable database is typically a page.



Fetch → Get a page

Flush = write Pages

Force Pages to be written from buffers to the stable DB.

Interface Between the Local Recovery Manager and the Buffer Manager.

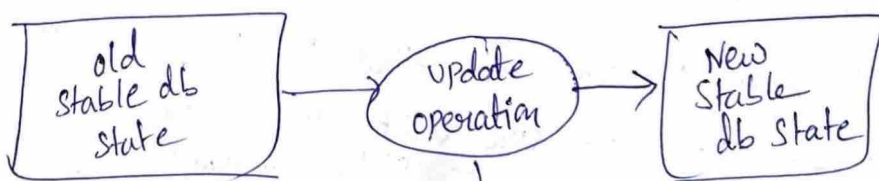
When LRM want to read a Page of the data on behalf of the transaction it indicate a fetch ^{issues} command to read, the buffer manager checks to see if that Page is already in the buffer due to previous transaction (fetch command), if so make it available to that transaction if not it read the page from the stable Database.

Recovery Information In-place update

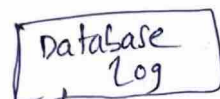
- Physically change the value of data items in stable DB. The previous values are lost.
- Do not change the value of the data items in stable DB, But maintain the new value separately.
↳ out of place update.

→ Most DBMS use in place update for Better Performance.

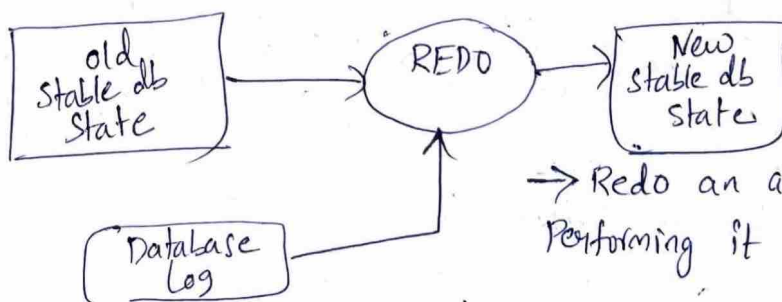
In place update Recovery Information



In Each update transaction not only changes the database but the changes also recorded in the database Log.



update operation Execution Changes are recorded in DB Log.



→ Redo an action means Performing it again.

REDO Action
= ↳ roll Forward

Database log: - Every action of transaction must not only the action but must also a log record to an append-only file.

The log contains information used by the recovery process to restore the consistency of a system. Each log record describes a significant event during transaction processing.

Types of log records

→ $\langle T_i, \text{start} \rangle$: if transaction T_i has started

$\langle T_i, x_j, v_1, v_2 \rangle$: before T_i executes a write(x_j), where v_1 is the old value before write and v_2 is the new value after the write.

$\langle T_i, \text{commit} \rangle$: If T_i has committed

$\langle T_i, \text{abort} \rangle$: if T_i has aborted

• $\langle \text{check point} \rangle$

Example for in-place update

Consider the transactions T_0 and T_1 (T_0 executes before T_1) and the following initial values

$A = 1000$ $B = 2000$ $C = 700$

T_0 : Read(A)

$A = A - 50$

Write(A)

Read(B)

$B = B + 50$

Write(B)

Possible order of actual o/p to the log file and the DB;

Log

DB

$\langle T_0, \text{start} \rangle$

$\langle T_0, A, 1000, 950 \rangle$

$\langle T_0, B, 2000, 2050 \rangle$

$\langle T_0, \text{commit} \rangle$

$A = 950$

$B = 2050$

T_1 : Read(C)

$C = C - 100$

Write(C)

T_1 (Start)

$\langle T_1, C, 700, 600 \rangle$

$\langle T_1, \text{commit} \rangle$

consider the log after some system crashes and corresponding recovery actions $C = 600$

a) $\langle T_0, \text{Start} \rangle$

$\langle T_0, A, 1000, 950 \rangle$

$\langle T_0, B, 2000, 2050 \rangle$

undo(T_0): B is restored
to 2000 and A to 1000

b) $\langle T_0, \text{Start} \rangle$

$\langle T_0, A, 1000, 950 \rangle$

$\langle T_0, B, 2000, 2050 \rangle$

$\langle T_0, \text{commit} \rangle$

$\langle T_1, \text{Start} \rangle$

$\langle T_1, C, 700, 600 \rangle$

undo(T_1) and redo(T_0):

C is restored to 700 and then
A and B are set to 950, 2050
respectively.

c) $\langle T_0, \text{Start} \rangle$

$\langle T_0, A, 1000, 950 \rangle$

$\langle T_0, B, 2000, 2050 \rangle$

$\langle T_0, \text{commit} \rangle$

$\langle T_1, \text{Start} \rangle$

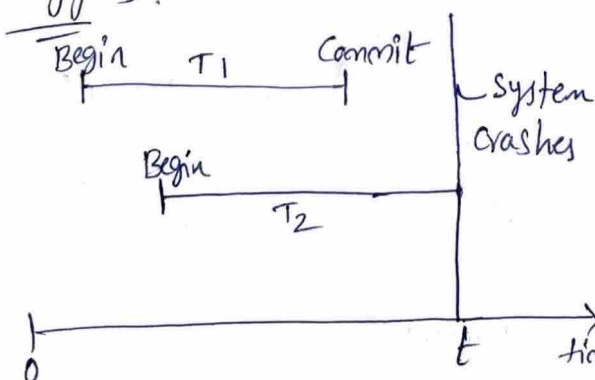
$\langle T_1, C, 700, 600 \rangle$

$\langle T_1, \text{commit} \rangle$

Redo(T_0) and redo(T_1):

A and B are Set to 950, 2050 respectively
; then C is set to 600.

Why logging?

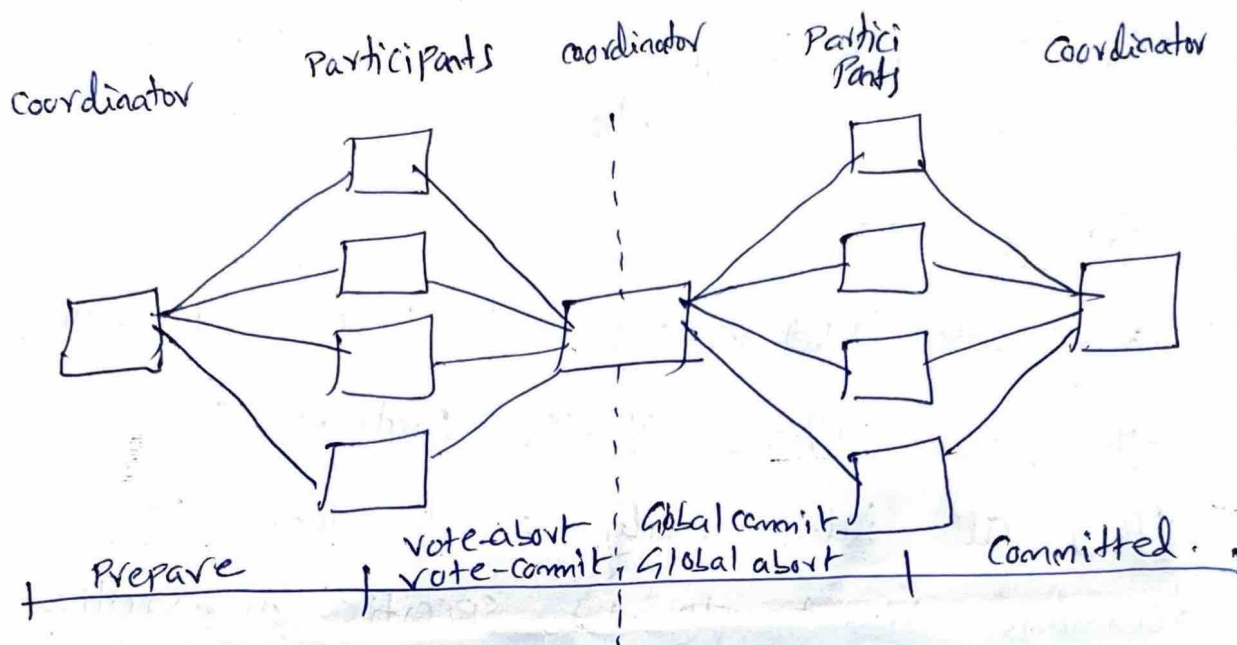


Assume buffer Pages are
written back to stable DB
only when Buffer manager
needs new buffer space.
→ T_1 : from user's view
Point, it is committed.
But updated Buffer
Pages may get lost. Redo is
needed.

T_2 : not terminated, but some updated Pages may have been
written to stable DB. Undo is needed.

Centralized 2PC

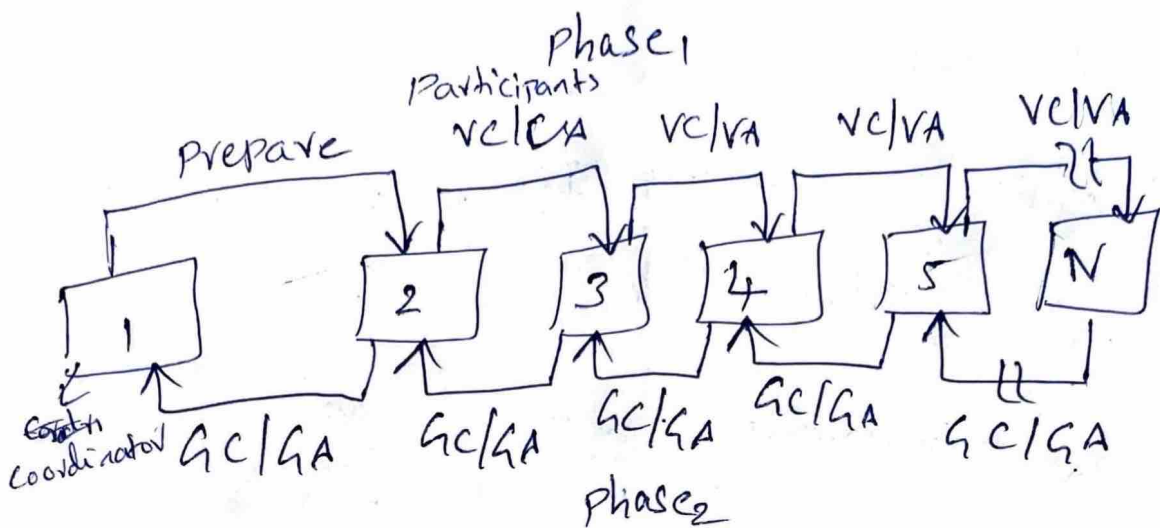
Centralized - Participants do not communicate among themselves.



Linear 2PC

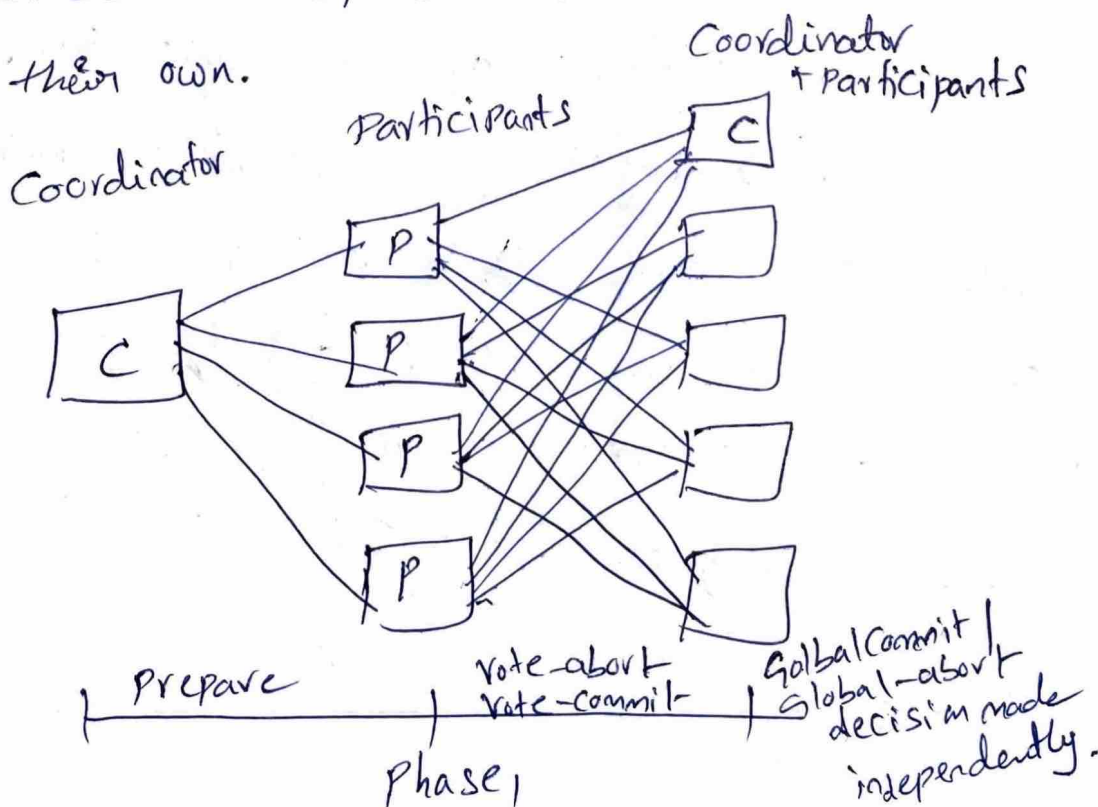
Linear - Participants communicate with one another.

There is an ordering between the sites in the system for the purpose of communication. 2PC Protocol implemented by forward communication from the coordinator 1 to N during which the first phase is completed and by a backward communication from N to the coordinator, during which the second phase is completed.



Distributed 2PC

communication (b/w) among all participant during the first phase of the protocol so that they all independently reach their termination decisions w.r.t to the specific transaction. This version called distributed 2PC, eliminates the need for the second phase of the protocol since participants can reach a decision on their own.



Site Failures

Failure of P.S (Participating Sites)

Site Fails Before sending Ready T

↳ Transaction is Aborted

Site Fails after sending Ready T

↳ Proceed in Normalway.

Recovery process when P.S restarts after failure

↳ Log file is checked.

i) in Log $\rightarrow [T, \text{Commit}]$ Redo T $\xrightarrow{\text{ack}}$
 $\searrow [T, \text{Abort}]$ Undo T (C.S)

ii) Log $\rightarrow [T, \text{Ready}] \rightarrow$ Site contacts the C.S to determine whether to Commit or abort T.

iii) No such record in Log $\rightarrow$ Undo T and abort the Transaction

Failure of Coordinator Site

All the P.S communicate with each other to determine status of 'T'

i) Any site $\begin{cases} [T, \text{Commit}] & \text{Committed} \\ [T, \text{Abort}] & \text{Aborted} \end{cases}$

iii) No case holds $\rightarrow$ All the P.S waits until C.S recovered.

Recovery

i) LOG — $\left\{ \begin{array}{l} [T, \text{Commit}] \text{ redo } T \\ [T, \text{Abort}] \text{ VNDOT} \end{array} \right\}$ shares with A1
P-S

3PC :- It is an extension of the 2PC that avoids blocking even if the coordinator.

Conditions to Avoid Blocking

ii) Atleast there is one available site

A tree diagram illustrating a search space. The root node is labeled S . It branches into five child nodes: S_1 , S_2 , S_3 , S_4 , and S_5 . The edges are labeled with the numbers 1, 2, and 3. The edges to S_1 and S_2 are marked with checkmarks.

$k=2$

- 1) [T, Prepare]
- 2) [T, Pre-commit] $\rightarrow$ Phase 4w voting & decision
- 3) [T, Commit]

S_4 - is chosen as coordinate
site

to All P.S

34, 59, 45, 24, 58

Network Partitioning

Network Partitions are due to communication line failures and may cause loss of Messages.

→ A Partitioning is called a Simple Partitioning if the N/w is divided into two components.

→ A Partitioning is called a multiple Partitioning if the N/w is Partitioned into more than two components.

Centralized Protocols

Voting Based Protocols (Quorum protocols)

The idea of majority voting has been generalized to voting with quorums.

The N/w Partitioning problem is handled by the Commit Protocol.

Every site is assigned a vote V_i .

Total no. of votes in the system V .

V_a - Abort Quorum

V_c - Commit quorum

Rules for obeying Commit Protocol

- 1) $V_a + V_c > V$, where $0 \leq V_a, V_c \leq V$
- 2) Before a transaction commits, it must obtain a commit quorum V_c .
- 3) Before a transaction aborts, it must obtain an abort quorum V_c .

The first rule ensures that a transaction cannot be committed and aborted at the same time.

If new elected coordinator fails, it is not known whether a commit or abort quorum has reached.

We introduce the PREABORT state to the ABORT state requires an abort quorum.

Parallel Database Systems

A Parallel computer or multiprocessor is a special kind of distributed system made of nodes (processors, disks and memories) connected by a very fast n/w.

→ Parallel db Systems consists of multiple processors and multiple disks connected by a fast interconnection n/w.

→ Parallel database system seeks to improve performance

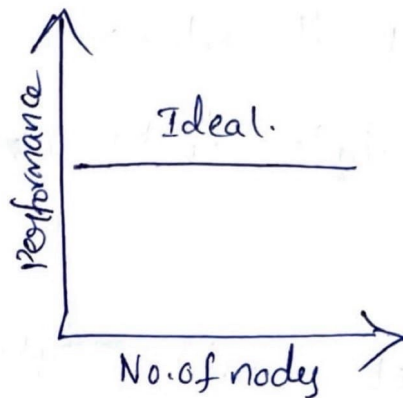
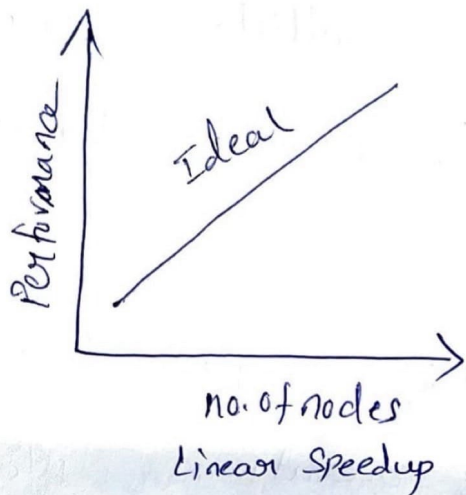
through parallelization of various operations such as loading data, building indexes and evaluating queries by using multiple CPU's and disks in parallel.

Parallel db Systems should provide following advantages

- 1) High-performance - This can be obtained through several complementary softwares - distributed database-oriented operating system support, parallel data management, query optimization and load balancing.
- 2) High availability - Increase data availability and fault-tolerance.
- 3) Extensibility - 2 advantages
 - 1) Linear speedup
 - 2) Linear Scalup.

Linear Speedup refers to a linear increase in Performance for a constant database size while the no. of nodes are increased linearly.

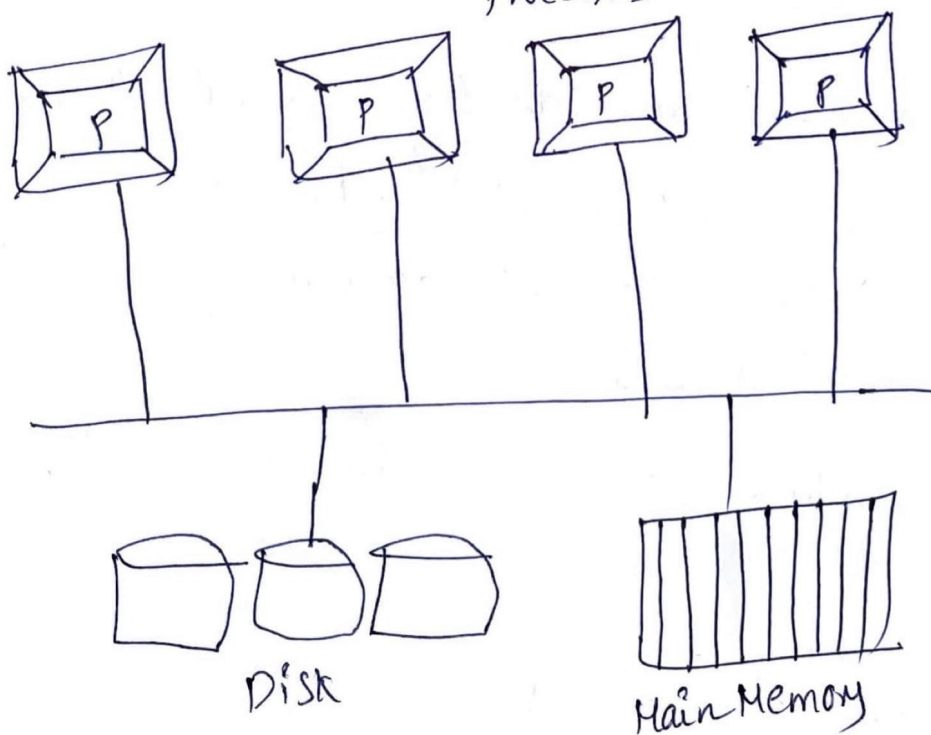
Linear Scalup refers to a Sustained performance for a linear increase in both database size and no. of nodes.



Extensibility Metrics

Parallel DB System Architectures

1) Shared Memory Architecture



Inter Communication
N/w

Parallel Data Placement:-

The Data Placement in a parallel database system exhibits similarities with data fragmentation in DD.

→ An obvious similarity is that fragmentation can be used to increase parallelism.

→ We use terms partitioning and partition instead of horizontal fragmentation and horizontal fragment.

→ Data placement must be done so as to maximize system performance, which can be measured by combining the total amount of work done by the system and the response time of individual queries.

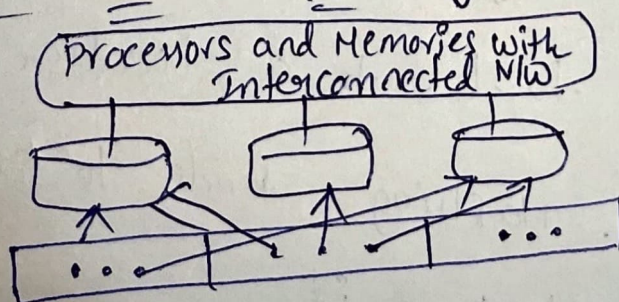
There are three basic strategies for data partitioning

1) Round-Robin

2) Hash

3) Range

Round-Robin Partitioning



Round-Robin Partitioning is the simplest strategy, it ensures uniform data distribution, with n partitions, the i th tuple insertion order is assigned to partition $(i \bmod n)$.

→ This strategy enables the sequential access to relation to be done in parallel.

Eg:

Record no. (i)	id	Name	Branch
1	1	Sam	Hyd
2	2	Ram	War
3	3	Hema	War
4	4	Haritha	NLG
5	5	Kiran	War

i = record no

n = no. of disks

It maps the i th tuple to disk $i \bmod n$

$i \bmod n$

$1 \bmod 3 = 1 \rightarrow \text{disk}_1$

$2 \bmod 3 = 2 \rightarrow \text{disk}_2$

$3 \bmod 3 = 0 \rightarrow \text{disk}_0$

$4 \bmod 3 = 1 \rightarrow \text{disk}_1$

$5 \bmod 3 = 2 \rightarrow \text{disk}_2$

No. of disks = 3

disk₀

disk₁

Round Robin.

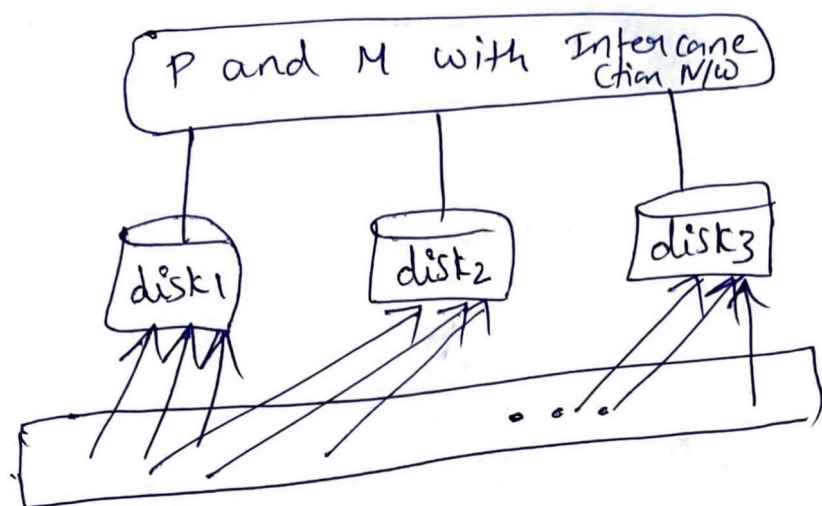
3 Hema War

1 Sam Hyd
4 Haritha

Adv: Sequential scan of all tuples in each query.

2) Hash Index Partitioning

It maps each tuple to a location based on a hash function. Hashing tends to randomise data rather than cluster it.



hash function
 $F(h) = h \bmod n$

Range Partitioning distributes tuples based on the value intervals (range) of some attribute.

Eg: The records with salary range 100 to 500 will be in disk₁, 500 to 10000 in disk₁, 10000 to 20000 to disk₂ etc.

A query with a predicate "A between A₁ and A₂" may be processed by the only node(s) containing tuples where A value in Range [A₁, A₂].

Parallel Database:-

Parallel database system seeks to improve performance through parallelization of various operations such as loading data, building indexes, and evaluating queries by using multiple cpus and disks in parallel.

Motivation for Parallel Database:

Parallel machines are becoming quite common and affordable.

- Prices of microprocessors, memory and disks have dropped sharply.

- Databases are growing increasingly large.

Large volumes of transaction data are collected and stored for later analysis.

- Multimedia objects like images are increasingly stored in databases.

1. Shared Memory Architecture :

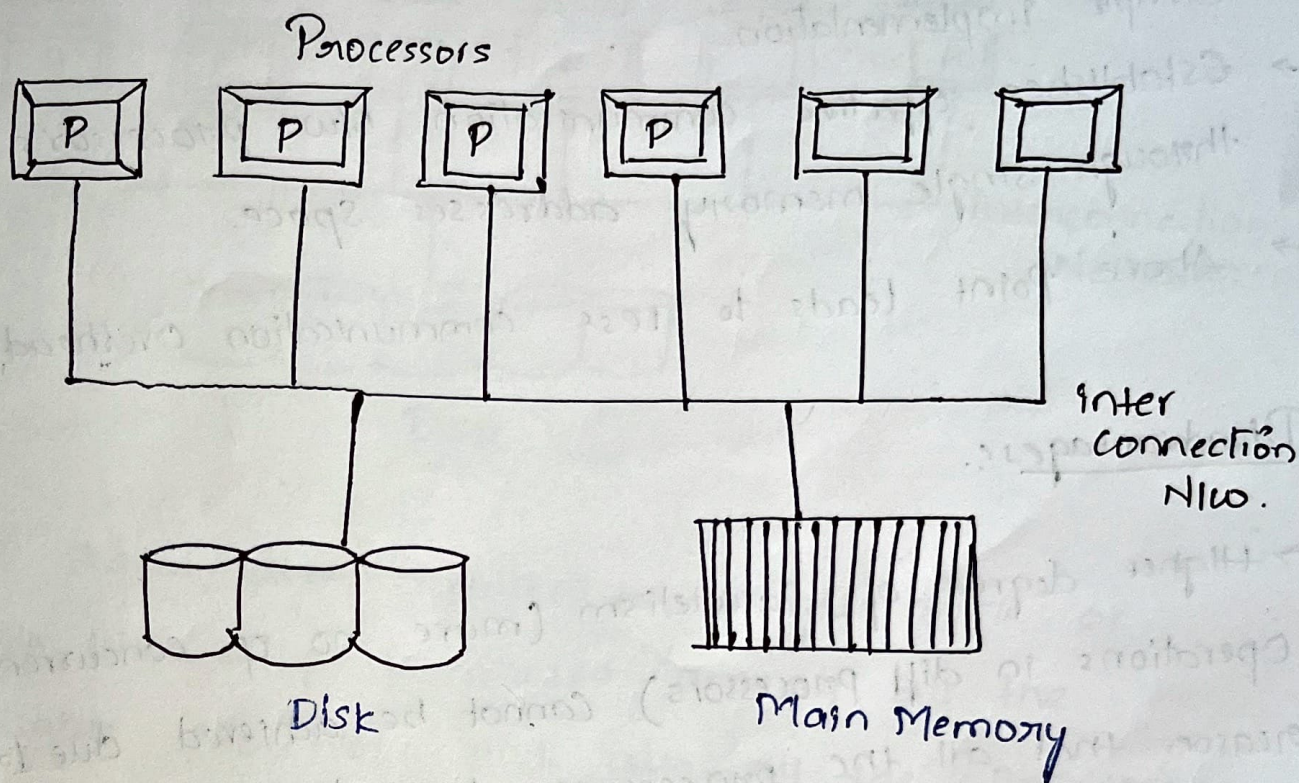


Figure: Shared Memory Architecture.

P represents processors

M represents Memory

D represents Disks/disk Setups.

In shared memory architecture, single memory is shared among many processors as shown in the figure.

Several processors are connected through an interconnection network with main memory and disk setup.

Here interconnection network is usually a high speed N/w (may be bus, mesh, or hypercube) which makes data sharing easy among the various components (processor, memory and disk)

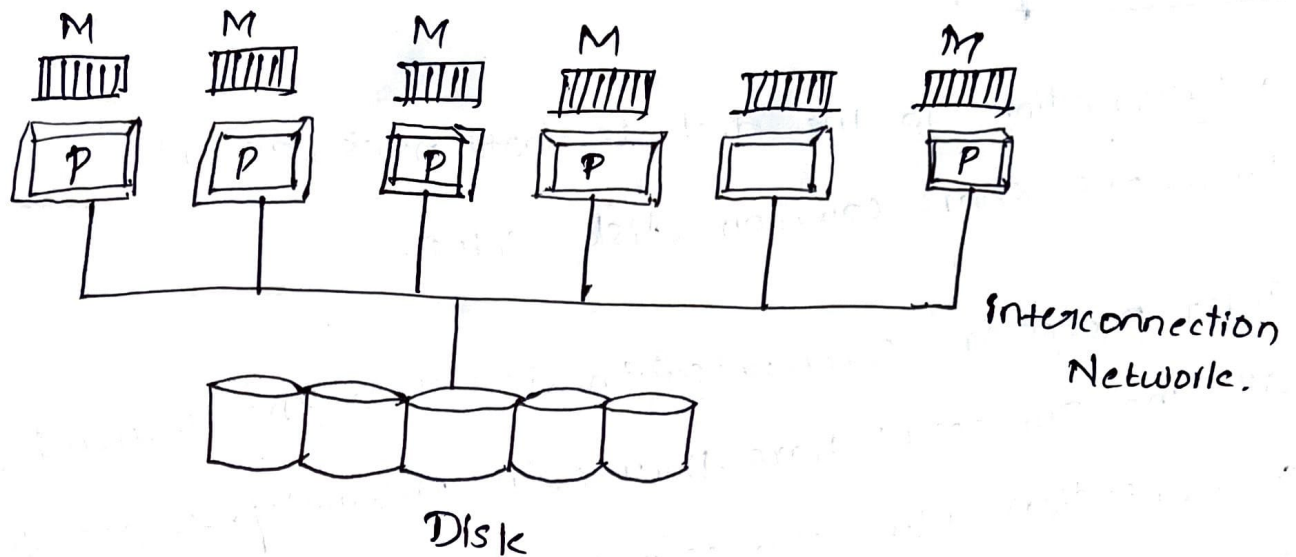
Advantages:-

- Simple implementation
- Establishes effective communication b/w processors through single memory addresses space.
- Above point leads to less communication overhead.

Disadvantages:-

- Higher degree of parallelism (more no of concurrent operations in diff processors) cannot be achieved due to the reason that all the processors share the same interconnection n/w to connect with memory. this causes Bottleneck in interconnection n/w especially in the case of bus interconnection n/w.
- Addition of processor would slow down the existing processors.
- Cache coherency should be maintained that is, if any processor tries to read the data used or modified by other processors, then we need to ensure that the data is of latest version.
- Degree of parallelism is limited. more number of parallel processes might degrade the performance.

2. Shared Disk Architecture:



In shared disk architecture, single disk or single disk setup is shared among all the available processors and also all the processors have their own private memories.

Advantages:-

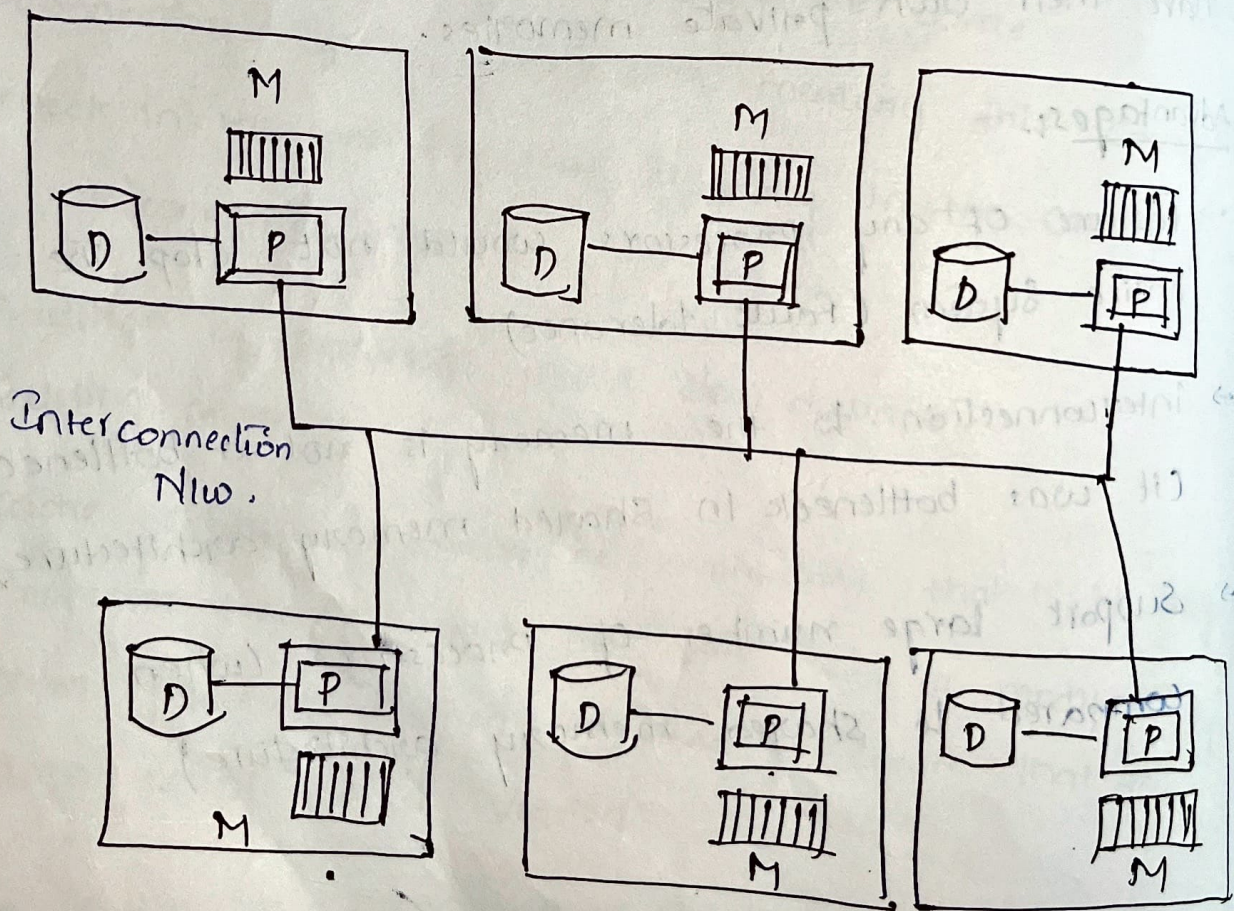
- Failure of any processors would not stop the entire system (fault tolerance)
- Interconnection to the memory is not a bottleneck (it was bottleneck in shared memory architecture)
- Support large number of processors (when compared to shared memory architecture).

Disadvantages:-

Interconnection to the disk is bottleneck as all processors share common disk setup.

Inter processor communication is slow the reason is all the processors have their own memory. Hence, the communication b/w processors need reading of data from other processors memory which needs additional software support.

3. Shared Nothing Architecture:



Advantages:-

Number of Processors used here is scalable. that is, the design is flexible to add more number of computers. Unlike in other two architectures, only the data requests which cannot be answered by local processors need to be forwarded through interconnection network.

Disadvantages:-

- Non Local disk accesses are costly. that is if one server receives the request if the required data not available, it must be routed to the server where the data is available. it is slightly complex.
- Communication cost involved in transporting data among computers.

Parallel data placement.

Parallel data placement refers to the physical placements of the data in a multiprocessor computer in order to favor placement data access and yield high performance. Most of the work on data placement has been done in the context of the shared nothing architecture.

Data placement in a parallel database system exhibits similarities with data fragmentation in distributed databases since fragmentation yields parallelism.

However, there is no need to maximize local processing at each node since users are not associated with particular nodes and load balancing is much more difficult to achieve in the presence of a large number of nodes.

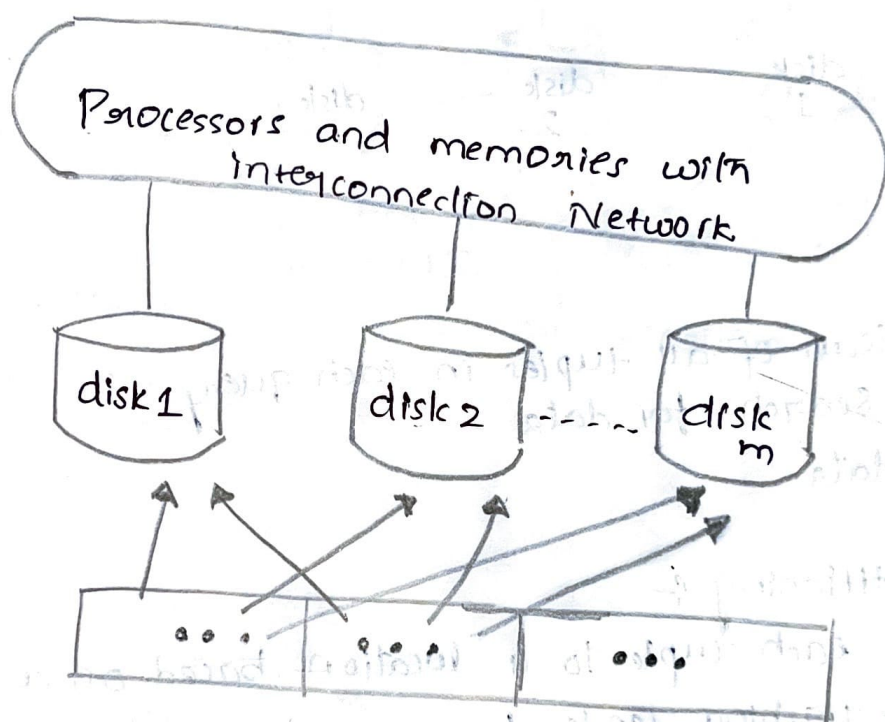
The main solution for parallel data placement is a variation of horizontal fragmentation, called

"Partitioning"

which divides database relations into partitions, each stored at a different disk node. there are 3 basic partitioning strategies.

1. Round robin partitioning:-

it maps the i^{th} tuple to disk $i \bmod n$

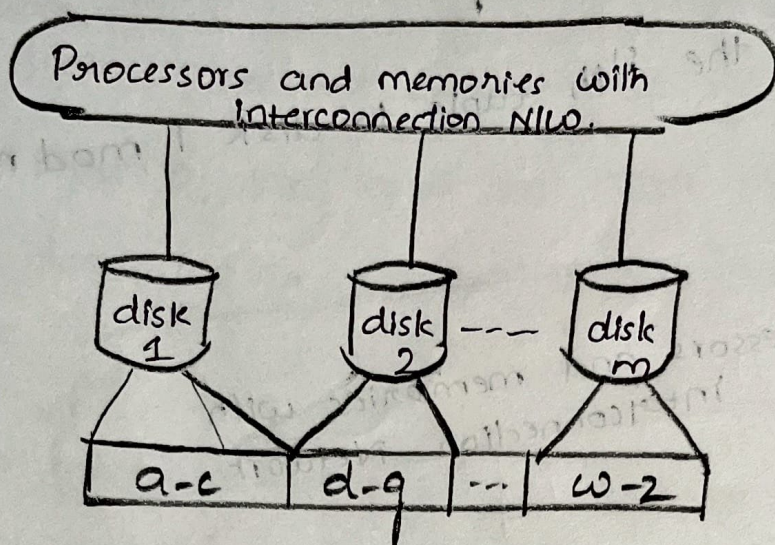


Advantage:-

Sequential Scan of all tuples in each query

2. Range Index partitioning:-

It clusters tuples with similar attributes together in the same partition.

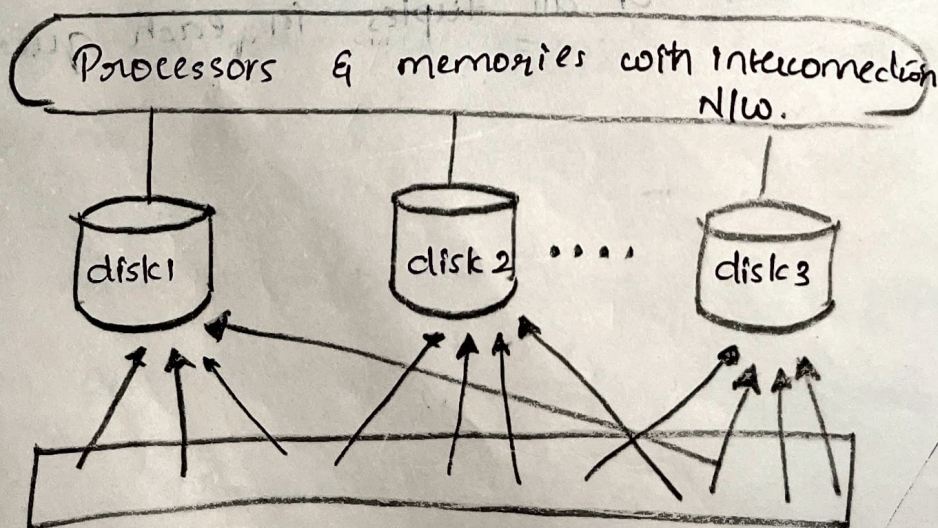


Adv:-

Sequential scan of all tuples in each query
Associative search for data
Clustering data.

3. Hash partitioning :-

It maps each tuple to a location based on a hash function. Hashing tends to randomise data rather than cluster it.



Adv:-

→ Sequential scan of all tuples in each query
→ Associative search for data.

Parallel Query Processing:-

Parallel Query Processing = Automatic translation of a query
into an efficient
execution plan.

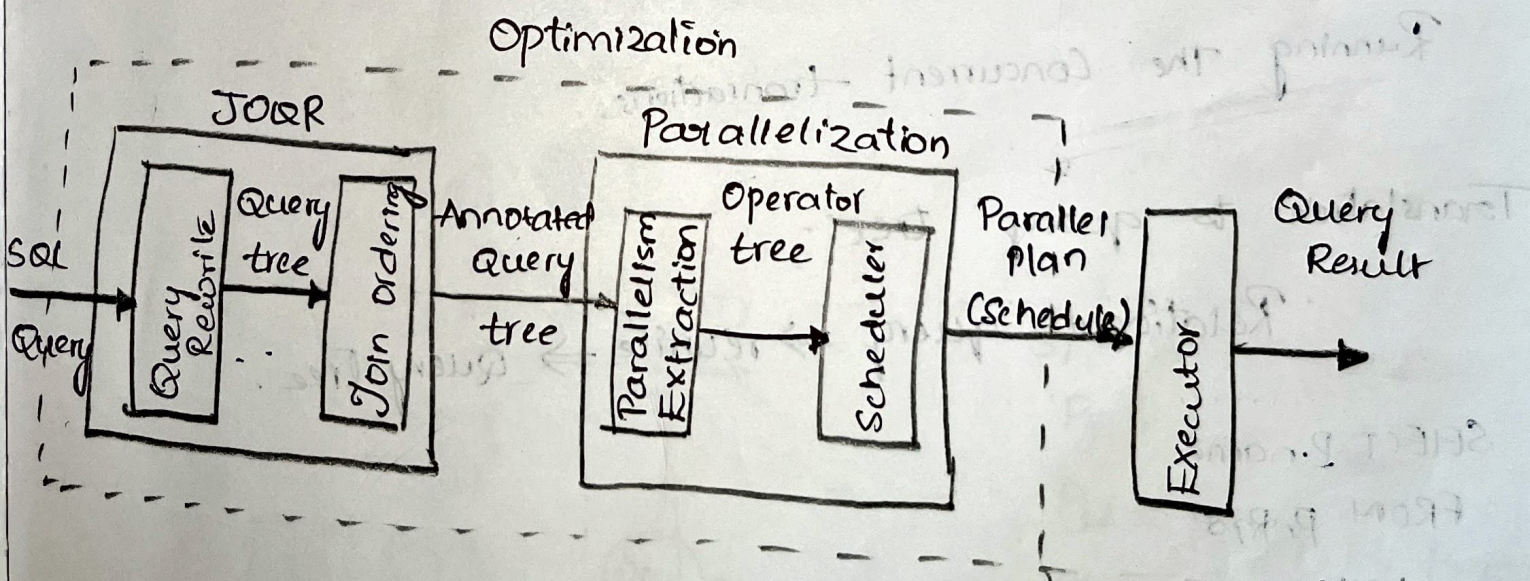
+
it's parallel execution.

The translation must be correct.

The execution plan produces the expected result

The execution plan must be optimal.

in that it minimises a cost function that captures
resource consumption.



JOQR - Join ordering and Query Rewrite

Parallel Query Processing:-

1. Translation:-

Of the relational algebra expression to a query tree.

2. Optimisation:-

Reordering of join operations in the query tree and choose among diff join algs 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.

Translation to query tree:-

Relational query $\Rightarrow$ rewrite $\Rightarrow$ Query Tree.

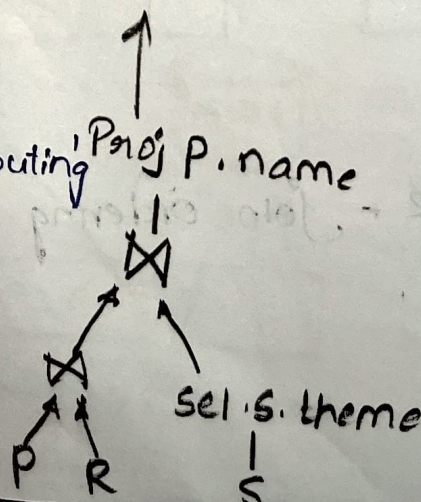
SELECT P.name

FROM P, R, S

WHERE S.theme like 'computing' AND P.name

AND S.class = R.class

AND R.id = P.id



Parallelisation of the query.

1. Extract parallelism by macro-expansion of the annotated query tree to an operator tree.

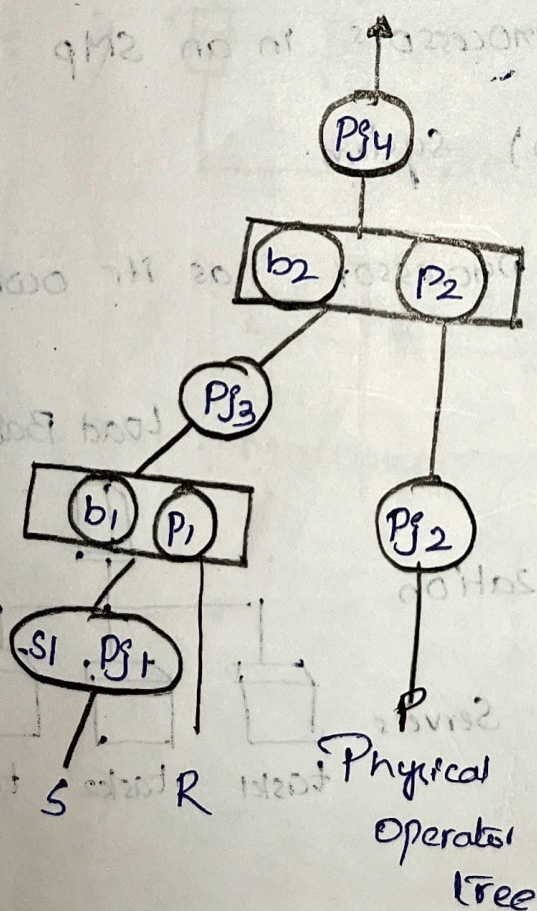
The operator tree identifies

- Atomic Code Pieces (operators)
- Timing Constraints b/w operators.

2. Schedule the operator tree on the parallel machine.

Goal: allocate machine resources so as to exploit the available parallelism which expecting timing and dataplacement constraints.

Parallelisation of the query



$S_1 = \text{Sel } S. \text{ thema}$

$P_{j1} = \text{Proj } S. \text{ class}$

$P_2 = \text{Proj } P. \text{ id, P-name}$

$P_{j3} = \text{Proj } R. \text{ id}$

$P_{j4} = \text{Proj } P. \text{ name}$

b_1, b_2

build hash table

P_1, P_2

Probe hash table

Load Balancing :-

A computer methodology to distribute workload across multiple computers or a computer cluster.

Network links, central processing units, disk drives or other resources networking.

Why do we go for Load Balancing:-

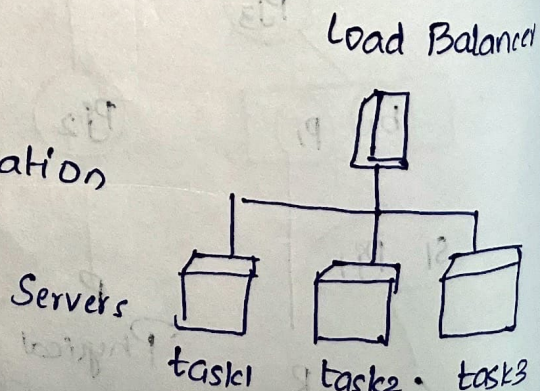
- High Availability
- High Reliability
- High Scalability
- Ease of Maintenance
- Resource sharing.

Load Balancing attempts to keep the workload evenly distributed across all processors in an SMP (Symmetric Multiprocessor system) system.

It is necessary where each processor has its own Private Queue.

Goals of Load Balancing

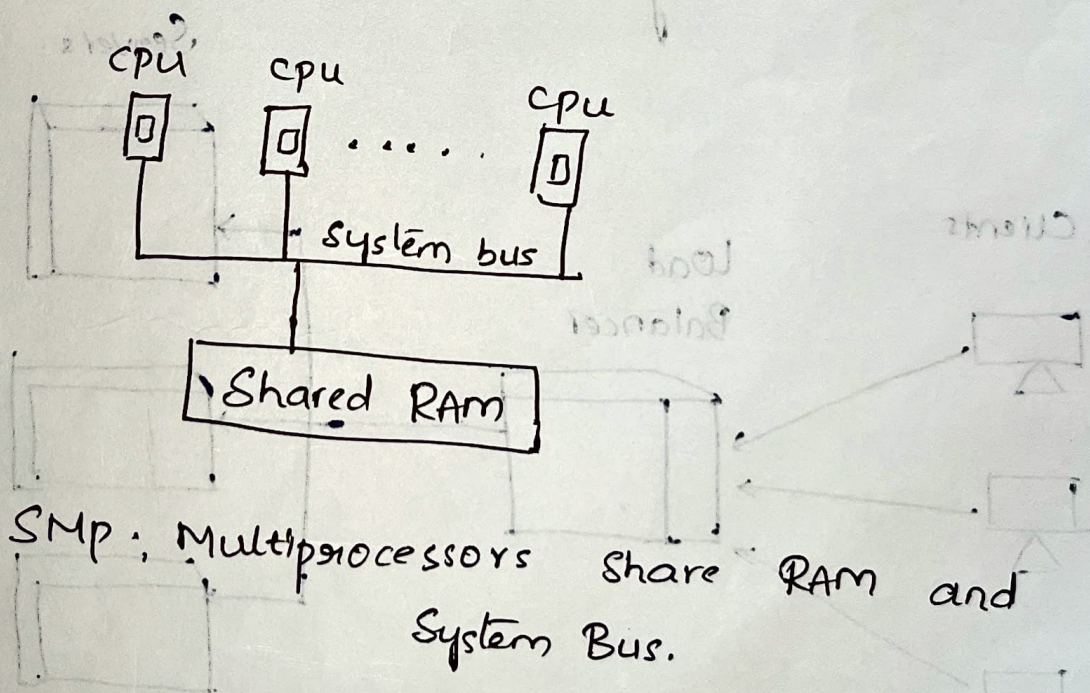
- Achieve Optimal Resource utilization
- Maximize Throughput
- Minimize Response time
- Avoid overload
- Avoid crashing.



Symmetric Multiprocessor System:-

A SMP is Multiprocessor System with centralized shared Memory called Main memory (MM) operating under a single OS with two or more homogeneous processors executing different programs and working on diff data, with the capability to share resources (memory, I/O device etc) and connected using system bus.

Usually each processor has an associated private high speed memory known as cache memory to speedup the MM data access and to reduce the system bus traffic.



SMP: Multiprocessors share RAM and System Bus.

General Approaches to Load Balancing

Push Migration

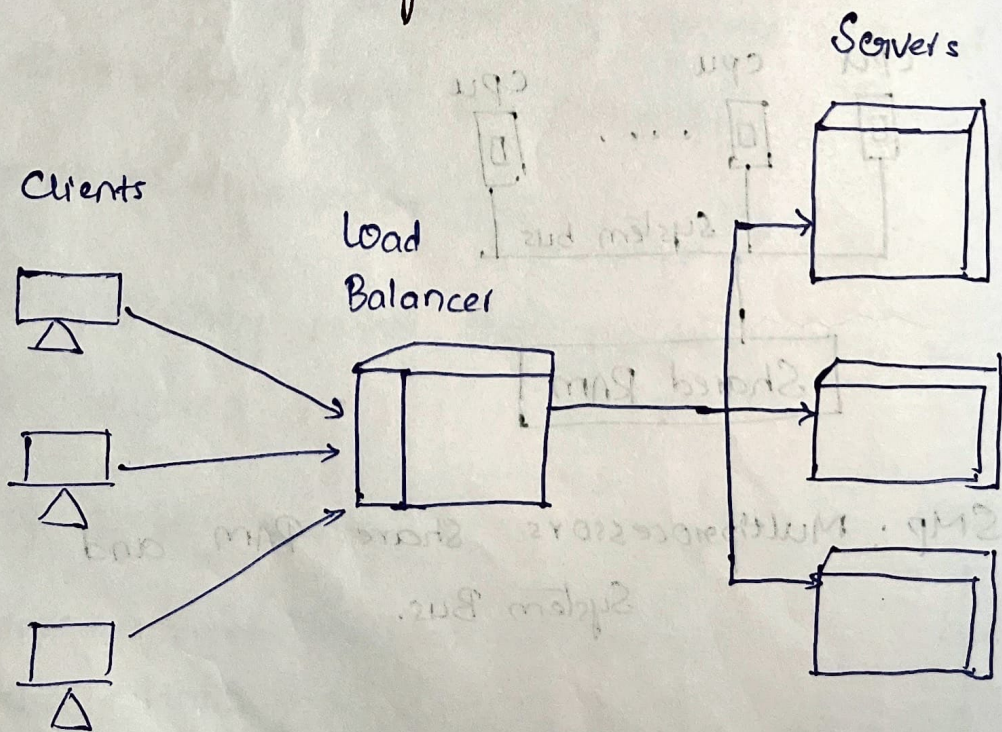
Overloaded process in one CPU (processor) is pushed to another processor for processing.

Pull Migration :-

The process which is overloaded is pulled from a processor and it's put under another processor for processing.

Types of Load Balancing :-

- Static load Balancing
- Dynamic load Balancing



Static load Balancing :-

It is the type of load Balancing which is often referred to as the mapping problem. the task is to map a static Process graph onto a fixed H/W topology. in order to minimize dilation and process load differences.

Dynamic load Balancing

it is describable in distributed system to have the system load balanced evenly among the nodes. so that the mean job response time is minimized.

Database clustering is the process of combining ~~to~~ more than one servers or instances connecting a single database

The main reasons for database clustering are its advantages a server receives.

- Data redundancy
- Load Balancing
- High Availability
- Monitoring and automation

Fundamental Object Concepts and Object Models

- An object DBMS is a system that uses an "object" as the fundamental modeling
- In a value-base data model, such as the relational model, data are identified by values. In object models, by contrast, data are identified by its OID
- One of the most important object model developed by Object Data Management Group (ODMG) that includes an object model (commonly referred to as the ODMG model), an Object Definition Language (ODL), and an Object Query Language (OQL)

There are four object concepts. They are:

1. Object:

- An object represents a real entity in the system that is being modeled.
- Most simply, it is represented as a tuple (OID, state, interface), in which OID is the object identifier, the corresponding state is some representation of the current state of the object, and the interface defines the behavior of the object.
- The state of an object is commonly defined as either an atomic value or a constructed value (e.g., tuple or set).
- E.g. Consider the following objects:
- (i1, 231)
- (i3, {i6,i11})
- (i4, {1, 3, 5})
- (i5, [LF: i7, RF: i8, LR: i9, RR: i10])

- Contrary to values, objects support a well-defined update operation that changes the object state without changing the object identifier (i.e., the identity of the object), which is immutable.
- E.g. Consider the following objects:
- (i1, Volvo)
- (i2, [name: John, mycar: i1])
- (i3, [name: Mary, mycar: i1])
- The state is represented as a set of instance variables (or attributes) that are values. These are called as the structural aspects of a model.
- The behavioral aspects of the model are captured in methods, which define the allowable operations on these objects and are used to manipulate them (Classic Example: Lift)
- The interface of an object consist of its properties. These properties include instance variables that reflect the state of the object, and the methods that define the operations that can be performed on this object.

2. Types and Classes

- **"Class"** refer to the specific object model construct and the term **"type"** to refer to a domain of objects (e.g., integer, string).
- A class is a template for a group of objects. The abstraction capability of classes, commonly referred to as encapsulation, hides the implementation details of the methods
- E.g.

type Car

attributes

```
engine : Engine
bumpers : (Bumper)
tires : (lf: Tire, rf: Tire, lr: Tire, rr: Tire)
make : Manufacturer
model : String
year : Date
serial_no : String
capacity : Integer
```

methods

```
age: Real
```

3. Composition (Aggregation)

- Composition is one of the most powerful features of object models. It allows sharing of objects, commonly referred to as referential sharing, since objects "refer" to each other by their OIDs as values of object-based attributes.
- E.g. Car type is a composite type and its instances are referred to as composite objects.

(i2, [name: John, mycar: c1])

(i3, [name: Mary, mycar: c1])

c1 is one instance of Car

4. Subclassing and Inheritance

- **Subclassing** indicate an is-a relationship between classes (types).
- Declaring a class to be a subclass of another results in **inheritance**. If class A is a subclass of B, then its its properties consist of the properties that it natively defines as well as the properties that it inherits from B. Inheritance allows reuse. A subclass may inherit either the **behavior (interface)** of its superclass, or its **implementation**, or both.
- E.g. type Vehicle as Object
 - attributes
 - engine : Engine
 - make : Manufacturer
 - model : String
 - year : Date
 - serial_no : String
 - methods
 - age: Real

In subclassing and inheritance, we also have this issue that is class extent.

1. Shallow extent
2. Deep extent

Distributed ObjectBased Systems

- Topics discussed in this chapter:
 - **CORBA** is an industry-defined standard for distributed systems.
 - **Distributed COM** can be viewed as a middleware layer implemented on top of various Windows operating systems.
 - **Globe** is a research project being developed as part of a research project on large-scale wide-area distributed systems.
 - **Comparison of CORBA, DCOM, and Globe**

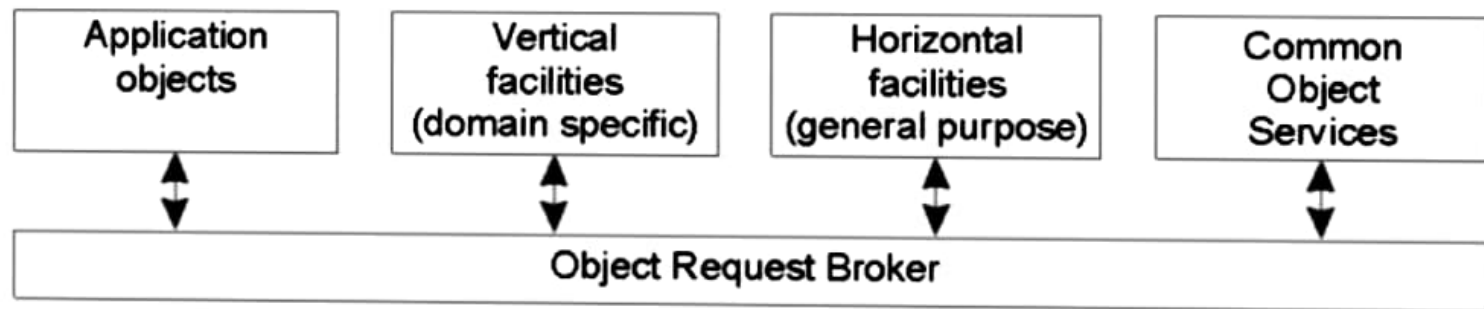
Overview of CORBA

- CORBA: Common Object Request Broker Architecture
- Background:
 - Developed by the Object Management Group (OMG) in response to industrial demands for objectbased middleware
 - Currently in version #2.4 and #3
 - CORBA is a specification: different implementations of CORBA exist
 - Very much the work of a committee: there are over 800 members of the OMG and many of them have a say in what CORBA should look like
- CORBA provides a simple distributedobject model, with specifications for many supporting services it may be here to stay (for a couple of years)

Overview of CORBA

- Architecture of CORBA
 - The **Object Request Broker (ORB)** forms the core of any CORBA distributed system.
 - **Horizontal facilities** consist of general-purpose high-level services that are independent of application domains.
 - User interface
 - Information management
 - System management
 - Task management
 - **Vertical facilities** consist of high-level services that are targeted to a specific application domain such as electronic commerce, banking, manufacturing.

Overview of CORBA



The global architecture of CORBA.

Object Model

- CORBA has a traditional remoteobject model in which an object residing at an object server is remote accessible through proxies
- All CORBA specifications are given by means of interface descriptions, expressed in an **Interface Definition Language (IDL)**.
 - An interface is a collection of **methods**, and **objects** specify which **interfaces** they implement.
 - It provides a precise syntax for expressing methods and their parameters.

Object Model

- CORBA follows an interfacebased approach to objects:
 - Not the objects, but interfaces are the really important entities
 - An object may implement one or more interfaces
 - Interface descriptions can be stored in an interface repository, and looked up at runtime
 - Mappings from IDL to specific programming are part of the CORBA specification (languages include C, C++, Smalltalk, Cobol, Ada, and Java).
- In DCOM and Globe, interfaces can be specified at a lower level in the form of tables, called **binary interfaces**.

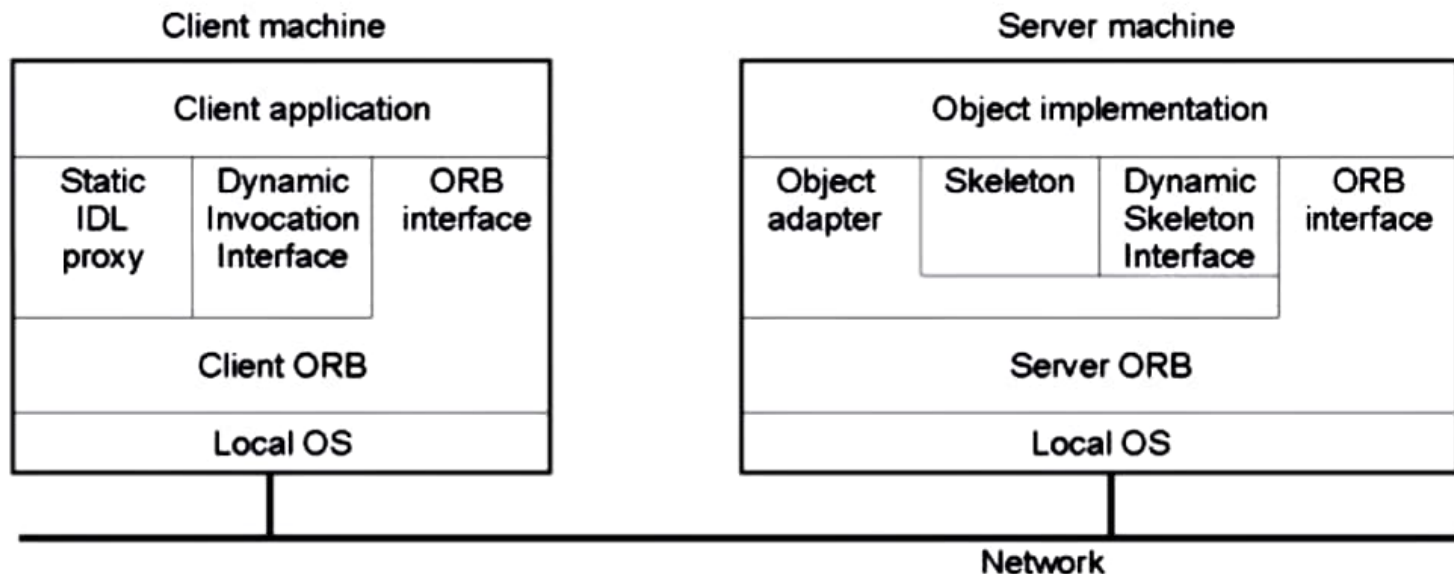
Object Model

- **Object Request Broker (ORB):** CORBA's object broker that connects clients, objects, and services
- **Proxy/Skeleton:** Precompiled code that takes care of (un)marshaling invocations and results
- **Dynamic Invocation/Skeleton Interface (DII/DSI):**
To allow clients to construct invocation requests at runtime instead of calling methods at a proxy, and having the serverside reconstruct those request into regular method invocations
- **Object adapter:** Serverside code that handles incoming invocation requests.

Object Model

- **Interface repository:**
 - Database containing interface definitions and which can be queried at runtime
 - Whenever an interface definition is compiled, the IDL compiler assigns a **repository identifier** to that interface.
- **Implementation repository:**
 - Database containing the implementation (code, and possibly also state) of objects.
 - Given an object reference, an object adaptor could contact the implementation repository to find out exactly what needs to be done.

Object Model



The general organization of a CORBA system.

Corba Services

Service	Description
Collection	Facilities for grouping objects into lists, queue, sets, etc.
Query	Facilities for querying collections of objects in a declarative manner
Concurrency	Facilities to allow concurrent access to shared objects
Transaction	Flat and nested transactions on method calls over multiple objects
Event	Facilities for asynchronous communication through events
Notification	Advanced facilities for event-based asynchronous communication
Externalization	Facilities for marshaling and unmarshaling of objects
Life cycle	Facilities for creation, deletion, copying, and moving of objects
Licensing	Facilities for attaching a license to an object
Naming	Facilities for systemwide name of objects
Property	Facilities for associating (attribute, value) pairs with objects
Trading	Facilities to publish and find the services on object has to offer
Persistence	Facilities for persistently storing objects
Relationship	Facilities for expressing relationships between objects
Security	Mechanisms for secure channels, authorization, and auditing
Time	Provides the current time within specified error margins

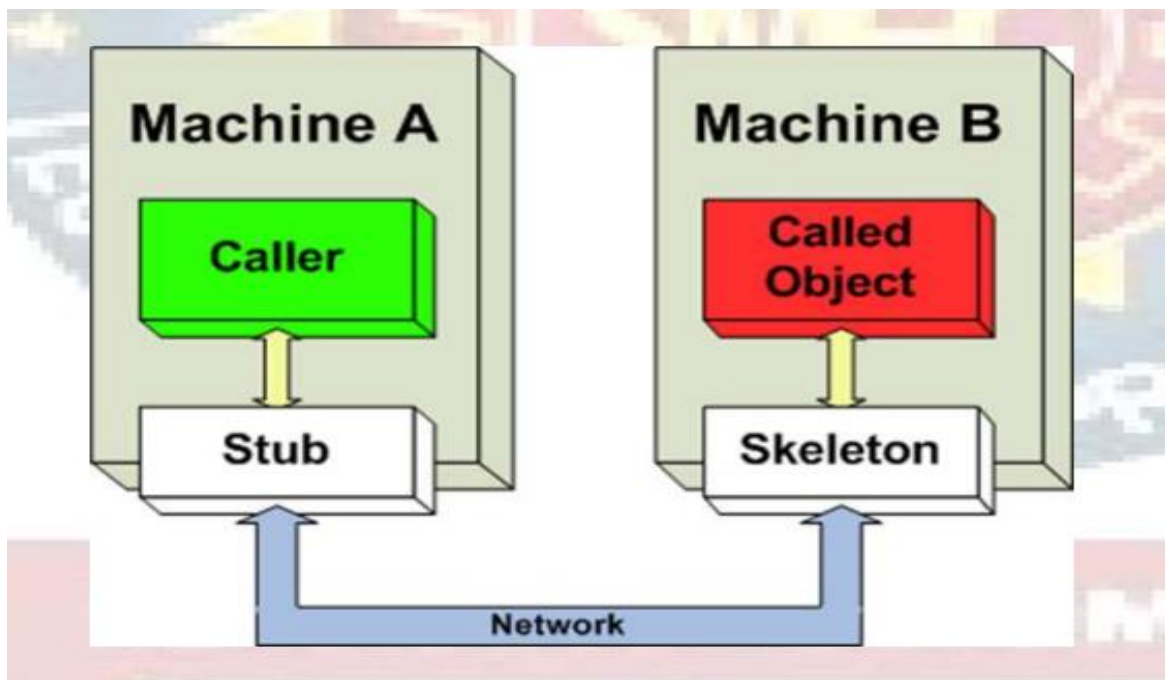
Overview of CORBA services.

Object distributed design:

In distributed computing, distributed objects [citation needed] are objects (in the sense of object-oriented programming) that are distributed across different address spaces, either in different processes on the same computer, or even in multiple computers connected via a network, but which work together by sharing data and invoking methods. This often involves location transparency, where remote objects appear the same as local objects. The main method of distributed object communication is with remote method invocation, generally by message-passing: one object sends a message to another object in a remote machine or process to perform some tasks. The results are sent back to the calling object.

Distributed objects were popular in the late 1990s and early 2000s, but have since fallen out of favour.[1] The term may also generally refer to one of the extensions of the basic object concept used in the context of distributed computing, such as replicated objects or live distributed objects. Replicated objects are groups of software components (replicas) that run a distributed multiparty protocol to achieve a high degree of consistency between their internal states, and that respond to requests in a coordinated manner. Referring to the group of replicas jointly as an object reflects the fact that interacting with any of them exposes the same externally visible state and behavior.

Live distributed objects (or simply live objects) generalize the replicated object concept to groups of replicas that might internally use any distributed protocol, perhaps resulting in only a weak consistency between their local states. Live distributed objects can also be defined as running instances of distributed multi-party protocols, viewed from the object-oriented perspective as entities that have distinct identity, and that can encapsulate distributed state and behavior.



Architectural issues:

Architectural issues in distributed data base systems Information systems. Data management systems. Database design and models. Database management system engines. Parallel and distributed DBMSs. Software and its engineering. Software organization and properties. Software system structures. Software architectures.

Object management:

A Object management system is a collection of programs that enables creation and maintenance of a database. DBMS is available as a software package that facilitates definition, construction, manipulation and sharing of data in a database. Definition of a database includes description of the structure of a database. Construction of a database involves actual storing of the data in any storage medium. Manipulation refers to the retrieving information from the database, updating the database and generating reports. Sharing of data facilitates data to be accessed by different users or programs.

Distributed object storage:

Object storage, also known as object-based storage, is a strategy that manages and manipulates data storage as distinct units, called objects. These objects are kept in a single storehouse and are not ingrained in files inside other folders. Instead, object storage combines the pieces of data that make up a file, adds all its relevant metadata to that file, and attaches a custom identifier. Object storage adds comprehensive metadata to the file, eliminating the tiered file structure used in file storage, and places everything into a flat address space, called a storage pool. This metadata is key to the success of object storage in that it provides deep analysis of the use and function of data in the storage pool.

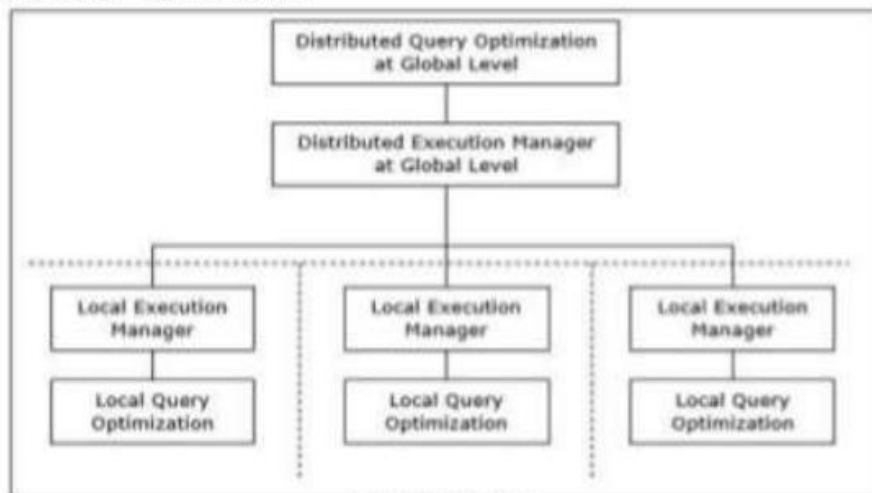
Object query Processing:

The main application of distributed query processing are classic distributed databases allowing to store and query data at different sites transparently. ...

Typical examples of such systems are object-oriented database systems but also middleware technologies such as application servers.

Query Processing in Distributed Systems

- In a distributed DBMS the catalog has to store additional information including the location of relations and their replicas. The catalog must also include system wise information such as the number of site in the system along with their identifiers'.



Object Identity

1. Object identity: An object retains its identity even if some or all of the values of variables or definitions of methods change over time.

This concept of object identity is necessary in applications but does not apply to tuples of a relational database.

2. Object identity is a stronger notion of identity than typically found in programming languages or in data models not based on object orientation.
3. Several forms of identity:
 - **value:** A data value is used for identity (e.g., the primary key of a tuple in a relational database).
 - **name:** A user-supplied name is used for identity (e.g., file name in a file system).
 - **built-in:** A notion of identity is built-into the data model or programming languages, and no user-supplied identifier is required (e.g., in OO systems).
4. Object identity is typically implemented via a *unique, system-generated* OID. The value of the OID is not visible to the external user, but is used internally by the system to identify each object uniquely and to create and manage inter-object references.
5. There are many situations where having the system generate identifiers automatically is a benefit, since it frees humans from performing that task. However, this ability should be used with care. System-generated identifiers are usually specific to the system, and have to be translated if data are moved to a different database system. System-generated identifiers may be redundant if the entities being modeled already have unique identifiers external to the system, e.g., SIN#.

OODBMS vs ORDBMS

- ORDBMS vs. OODBMS
 - similarities: both support user-defined ADTs, constructed types, reference types, object identity, query language
 - differences: ORDBMSs add new data types to RDBMS; OODBMSs add DBMS functionalities to a programming language
- Integration with host language
 - OODBMS: seamless integration with C++/Small talk
 - ORDBMS: integration is only through embedded SQL in a host language
- Application requirement
 - OODBMS:
 - few large objects fetched occasionally: few disk I/O
 - long duration transactions on in-memory objects
 - ability to cache objects in memory
 - ORDBMS:
 - large collection of data
 - extensive disk I/O
 - short transactions
- Query language
 - OODBMS:
 - Query processing is relatively inefficient
 - No standard available
 - ORDBMS:
 - Query facilities is the centerpiece
 - SQL-based standards available: SQL3, SQL4

Object-oriented data Model:

To represent the complex real world problems there was a need for a data model that is closely related to real world.

Object oriented data model represents the real world Problems easily.

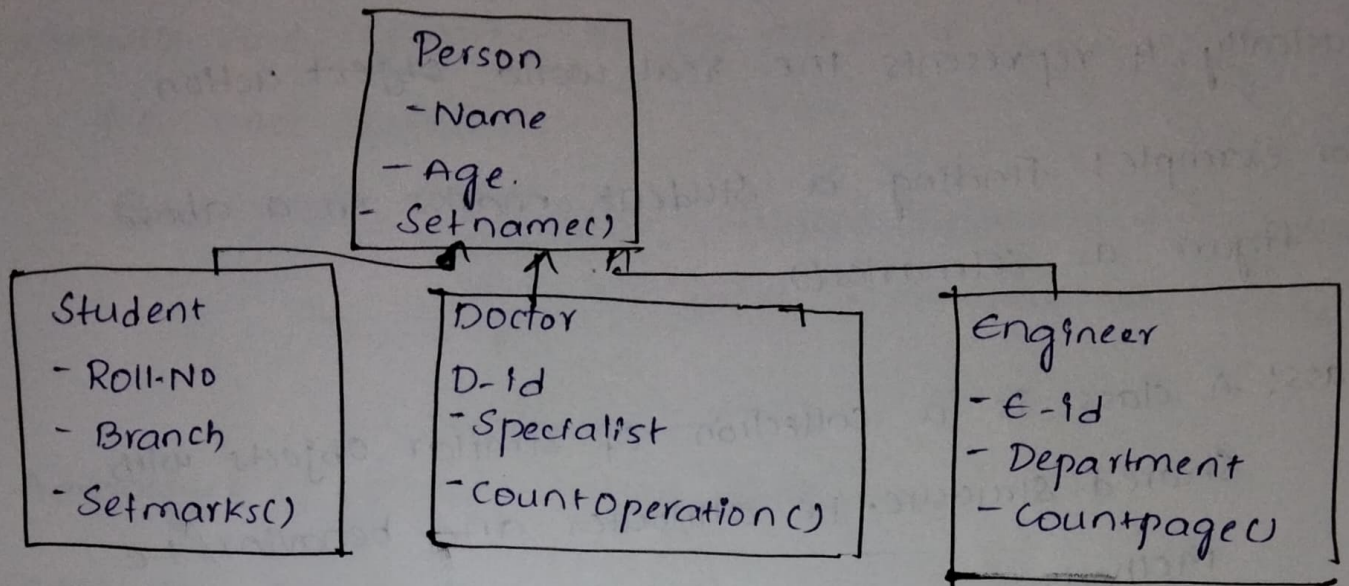
In object-oriented data model, data and their relationships are contained in a single structure which is referred as object in this data model.

In this real world problems are represented as objects with diff attributes.

All objects have multiple relationships b/w them. Basically, it is combination of object-oriented programming and relational database model. as it is clear from the foll. figure.

Object-oriented data model = Combination of object oriented programming + Relational database model.

Components of Object-oriented data model.



Objects:-

An object is an abstraction of a real world entity or we can say it is an instance of class.

Objects encapsulate data and code into a single unit which provides data abstraction by hiding the implementation details from the user.

For Example: Instances of student, doctor, engineer in above figure

Attribute: An attribute describes the properties of object.

Ex:- Object is student

Attributes are Roll.No, Branch, setmarks() in the student class.

Methods:

Method represents the behavior of an object.

Basically, it represents the real world object action.

For example: Finding a student marks in a above figure as `setmarks()`

Class: A class is a collection of similar objects with shared structure. i.e attributes and behavior. i.e methods.

An object is an instance of class.

For example: person, student, Doctor, Engineer in the above figure.

```
class student
{
    char Name [20];
    int roll-no;
    --
    --
    public ;
    void search();
    void update();
}
```

Inheritance:

By using inheritance, new class can inherit the attributes and methods of the old class.

i.e. base class.

For example: As classes Student, Doctor and Engineer are inherited from the base class Person.

Advantages of Object-oriented Datamodel

Codes can be reused due to inheritance easily understandable.

Cost of maintenance can be reduced due to reusability of attributes and functions because of inheritance.

Disadvantages of Object-oriented Datamodel.

It is not properly developed so not accepted by users easily.

Persistent programming languages

Persistent data: data that continue to exist even after the program that created it has terminated.

A persistent programming language is a programming language extended with constructs to handle persistent data. It distinguishes with Embedded SQL in at least two ways.

1. In a persistent program language, query language is fully integrated with the host language and both share the same type system. Any format changes required in database are carried out transparently.

Comparison with Embedded SQL where host and DML have different type systems, code conversion operates outside of OO type (Object-oriented type) system, and hence has a higher chance of having undetected errors

format conversion takes a substantial amount of code. Using Embedded SQL, a programmer is responsible for writing explicit code to fetch data in to memory or store data back to the database.

In a persistent program language, a programmer can manipulate persistent data without having to write such code explicitly.

Drawbacks:

1. Powerful but easy to make programming errors, but it can damage the database.
2. Harder to do automatic high-level optimization.
3. Do not support declarative querying well.

Persistence of objects:

Several approaches have been proposed to make objects persistent.

1. Persistent by class:

Declare class to be persistent!

All objects of the class are then persistent objects. Simple not flexible, since it is often useful to have both transient and persistent objects in a single class.

In many OODB (object-oriented Database) Systems, declaring a class to be persistent is interpreted as:

"Persistable" — objects in the class potentially can be made persistent.

2. Persistence by creation:

Introduce new syntax to create persistent objects.

3. Persistence by marking:

Mark an object persistent after it is created (and before the program terminates)

4. Persistence by reference:

One or more objects are explicitly declared as (root) persistent objects.

All other objects are persistent objects iff they are referred, directly or indirectly, from a root persistent object. It is easy to make the entire data structure persistent by merely declaring the root of the structure as persistent. but is expensive to follow the chains in detection for a database system.