Review of Database

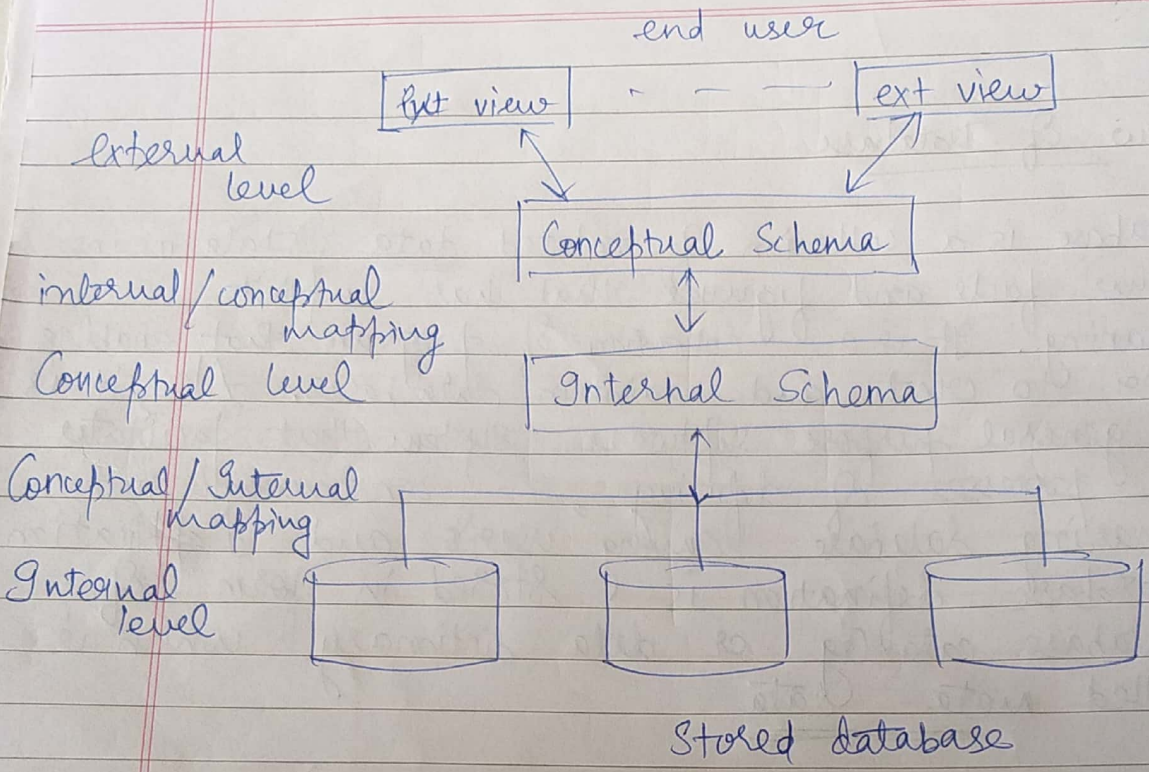
Database is a collection of related data. Data means raw facts and figures that has implicit meaning. It is a collection of program that enables user to create and maintain databases. Also, it is a general purpose software system that facilitates the processes of defining, constructing and sharing database among users and application. Database defination is stored in form of database catalog or data dictionary which is called meta data

→ **Schema** (layout + attributes)
structure or basic layout of a table

e.g
eid	no(5)
ename	varchar(50)

After adding values in table it becomes,
Instance

3 Schema Architecture

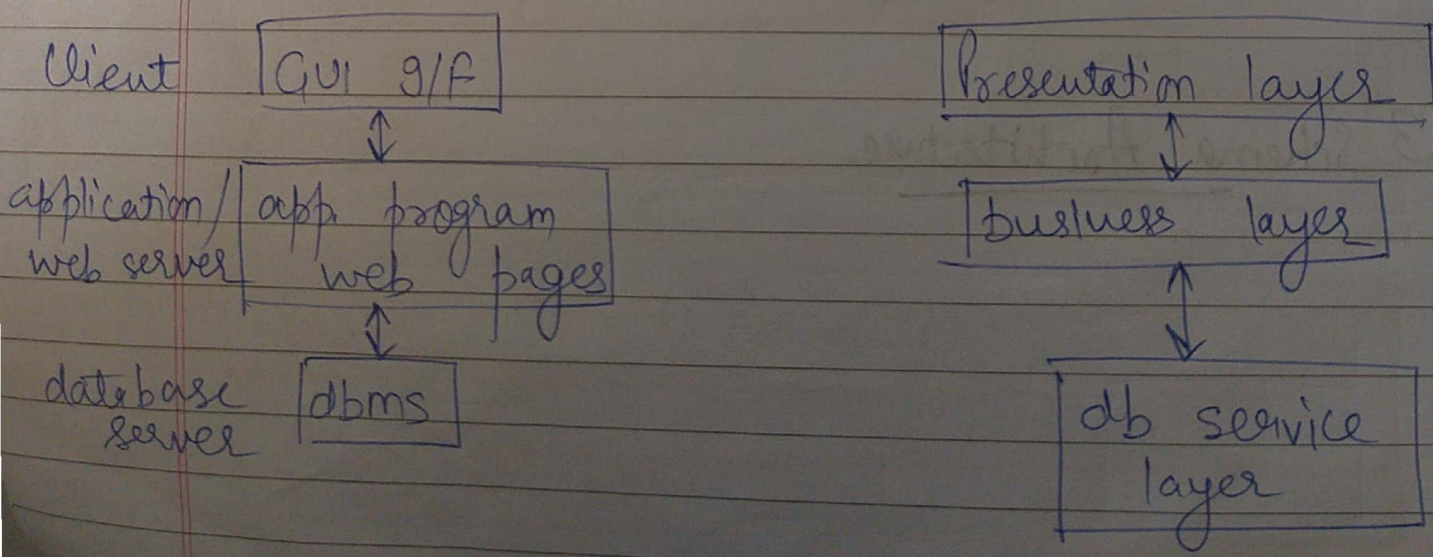


Data Independence (defines the capability of Db of changing data at one level will not affect other)

physical — int → Conceptual

logical — Conceptual → ext

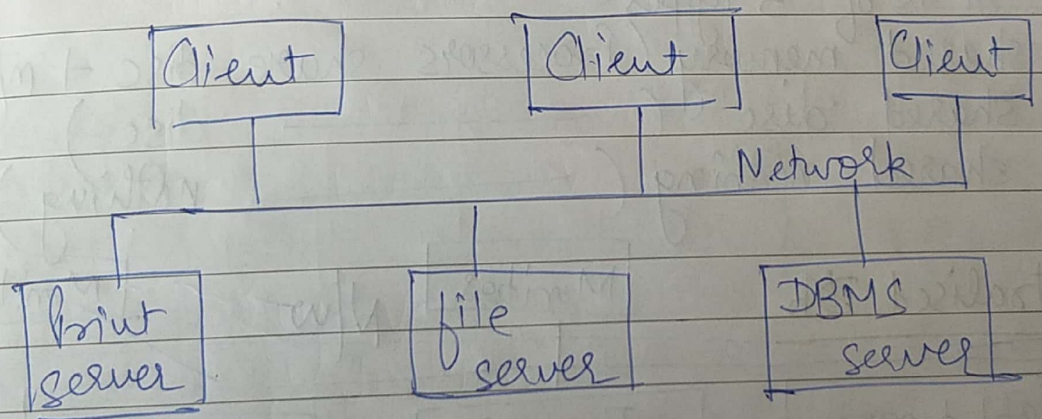
3 tier Client Server Architecture



→ Presentation layer – gt displays info. to user and allow data entry
→ Business layer – gt handles intermediate rules and constraint before data is passed up
→ db servic layer – gt includes all dbms services

* Client → application/web server, dbase server act as a 3-tier
* application/web servel – act as a intermediatary role by running application program & storing business rules. gt accepts requests from client and process the request and send dbase query and commands to dbase server.

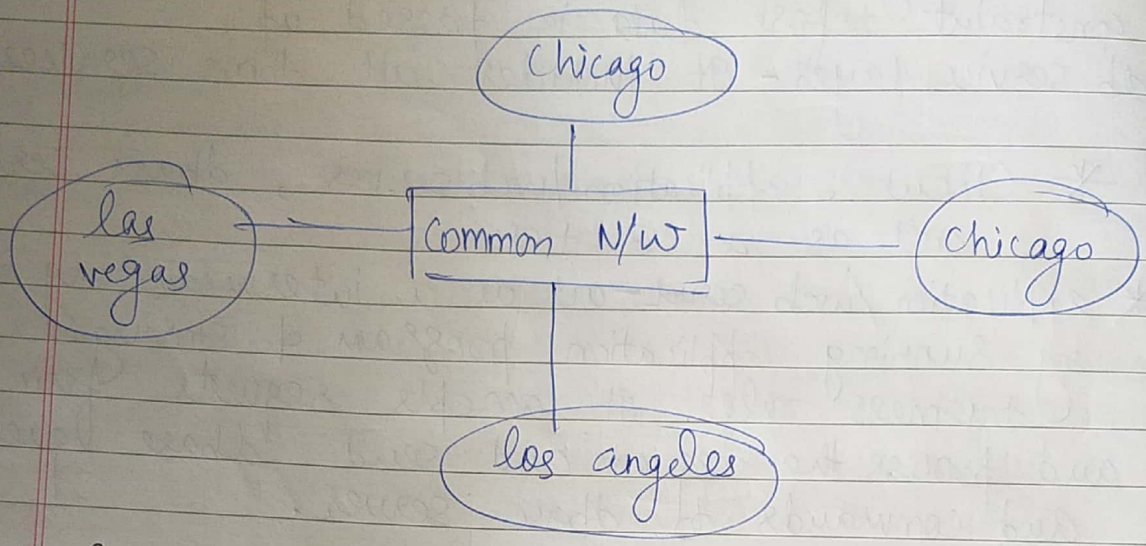
2 tier Architecture



* ① Distributed
② C/S
2 3
③ Parallel
④ Centralised

Distributed Db

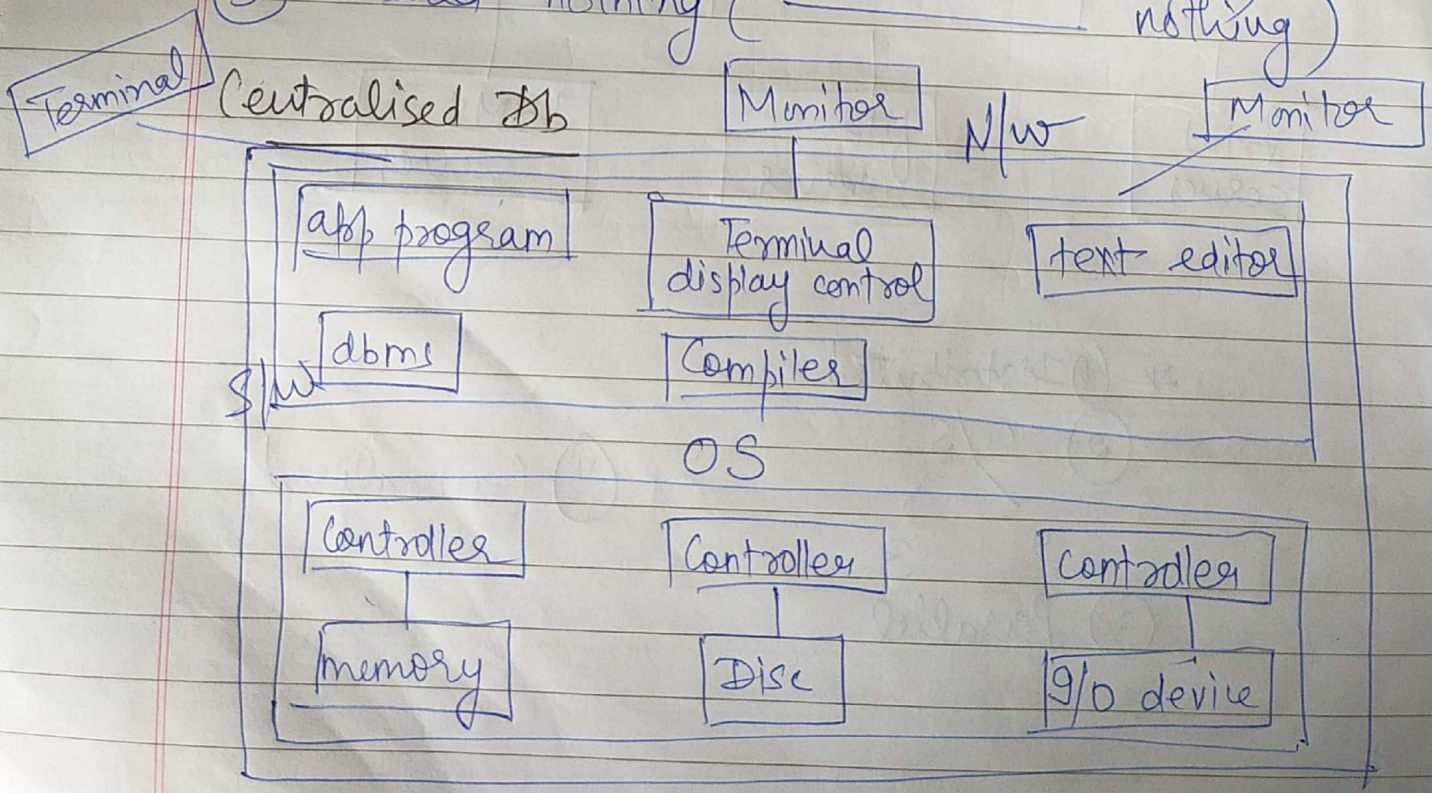
It defines multiple logically interconnected related database and distributed over different networks



Parallel Db

It is of 3 types -

1. shared memory (Processors share disc + m/m)
2. shared disc (——— disc)
3. shared nothing (——— nothing)



XML (extensible markup language)

html ⇒ <tag> -- </tag>
(used to design static web pages)

xml ⇒ <attribute name> — </attribute name>
(structuring + exchanging of data over web)

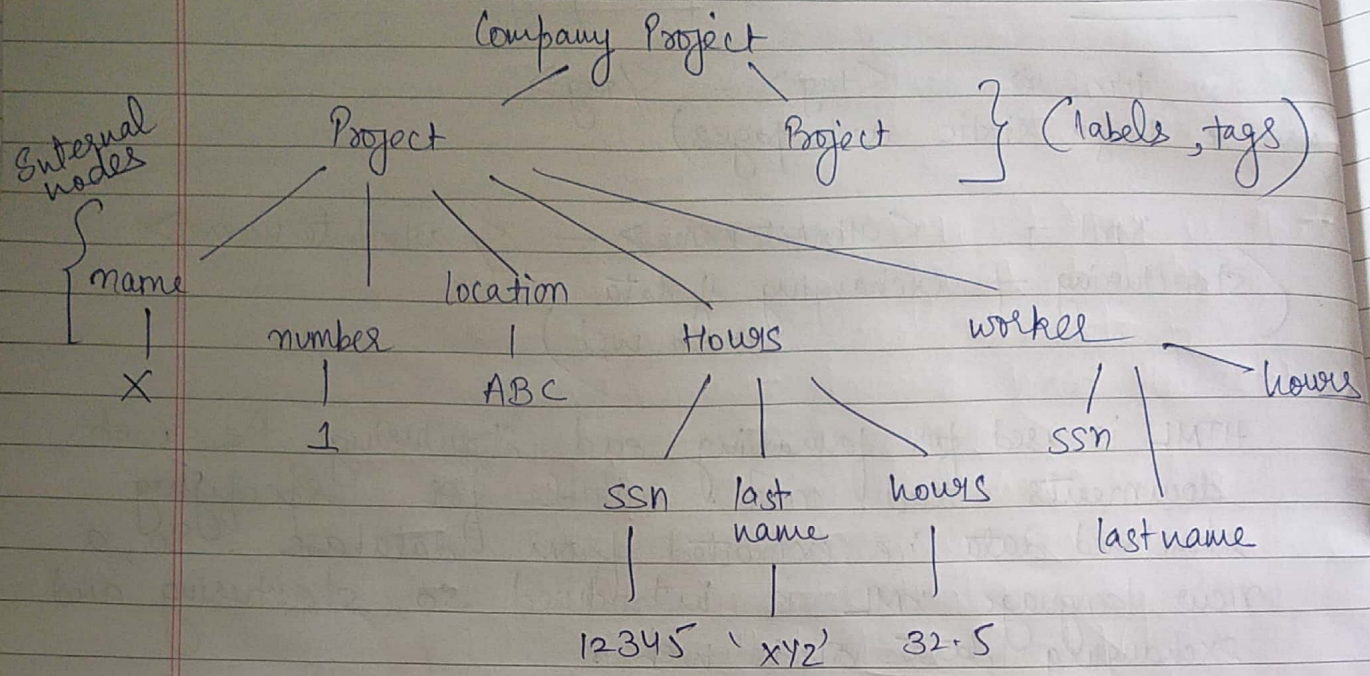
HTML is used for formatting and structuring the web documents and not suitable for specifying structured data i.e extracted from database. So, a new language XML is introduced so structuring and exchanging data over web.

Structured, semi structured & unstructured data

Structured
data having pre defined format
The DBMS checks to ensure that all data follows the structure & constraints specified in schema as it is to be represented in strict format
eg ⇒ relational table in db (in the form of rows & columns)

semi structured
The data i.e partially structured & partially unstructured.
* No pre defined schema. This data may have certain structure but not all the info collected will have identical structure
e.g ⇒ tree graph, email

unstructured
As the name indicates, it is has no structure & there is very limited indication of type of data
e.g ⇒ html code




```
<?xml version = "1.0" standalone = "yes" ?>
<Projects>
    <Project>
        <name> X </name>
        <number> 1 </number>
        <location> ABC </location>
        <deptno> </deptno>
        <worker>
            <SSN> - </->
            <lastname> - </->
            <hours> - </->
        </worker>
        <worker>
            <SSN> - </->
            <lastname> - </->
            <hours> - </->
        </worker>
```

Simple elements as they contain only data

</Project>
<Project>
<name> — </>
<number> — </>
<location> — </>
<deptno> — </>
<worker>
<ssn> — </>
<lastname> — </>
<hours> — </>
</worker>
<worker>
<ssn> — </>
<lastname> — </>
<hours> — </>
</worker>
</Project>

</Projects>

XML
- Model
- document
- DTD
- Schema

XML Hierarchical Tree Data Model

Two main structuring concepts are used to construct an XML document i.e elements and attributes. The term attribute is not used in the same manner in db terminology but it is to be used in document description language such as HTML. The following e.g of XML element called Projects (root node) and these elements are identified in a document by their start & end tag.

→ Complex & Simple elements

Complex elements are constructed from other elements hierarchally whereas simple elements contain data values.

→ The major difference b/w XML & HTML is that XML tag needs a defined tool to describe the meaning of data elements in document rather than to describe how the text is to be displayed.

→ In this tree representation, internal nodes represents complex elements whereas leaf nodes represent simple elements that's why the XML model is called tree model.

→ Simple elements are <name>, <number>, <location>, <dept no>, <ssn> <last name>, <hours>

→ Complex elements are <Project> <worker>

→ It is possible to characterize three main types of XML documents –

1. Data centric
2. document centric
3. Hybrid centric

Data Centric – These document have small data items, follows a specific structure in order to exchange it over web

Document Centric – There are document with large amount of text such as books or articles. There are few or no structured data elements in these

documents

Hybrid Centric – These document may contain structured or unstructured data.

→ XML document that doesn't follow a predefined schema of elements, names & corresponding tree structure are known as schemaless XML document.

→ When the value of standalone attribute in XML document is Yes, the document is standalone & schemaless.

XML Document, DTD, Schema

An XML Document is well formed if it follows few conditions –

1. It must start with XML declaration to indicate the version of XML being used as well as other relevant attributes
2. There should be single root element & every element must include a matching pair of start & end tag within the start & end tag of Parent element. This ensures that nested element specifies well formed tree structure

II nd Property say a document should be valid (first it should be well formed)

1. It is valid when the element tag used in start & end tag must follow the structure specified in separate XML DTD (document

type defination) file or XML schema file

Projects XML DTD

```
<!DOCTYPE Projects [
    <!ELEMENT Projects (Project+)>
    <!ELEMENT Project (name, number, location,
        deptno?, workers)
(Optional) <!ATTLIST Project
        ProjID # REQUIRED >
    <!ELEMENT Name (#PCDATA)>
    <!"  number (  )>
    <!"  location (  )>
    <!"  deptno (  )>
    <!ELEMENT workers (worker*)>
    <!ELEMENT worker (ssn, lastname?, hours)
    <!ELEMENT ssn (#PCDATA)>
    <!ELEMENT lastname (#PCDATA)>
    <!ELEMENT hours (#PCDATA)>
]>
```

while specifying the elements the following notations are used

(1) * ⇒ * following the element name means that the element can be repeated zero & more times in document. It is called as optional multivalued (repeating) element

(2) + ⇒ the element can be repeated one or more time in document & this type of element is required multivalued (repeating) element

(3) ? → The element can be repeated zero or one time. This is called single valued (non repeating) element.

(4) The elements appearing without any of preceding 3 symbols must appear exactly once in a document. This kind is required single valued (non repeating) element.

(5) If the parenthesis (()) include the key word #PCDATA (Passed character data) in XML DTD, the element is a leaf node & PCDATA is similar to string data

(6) The list of attributes that can appear within an element can be specified via keyword <!ATTLIST>

(7) Parenthesis can be nested when specifying elements

(8) A bar symbol e1/e2 specifies that either e1 or e2 can appear in the document

→ When the value of standalone attribute in XML document is no. The document needs to be checked against a separate DTD document or XML schema document. The XML DTD file should be stored in the same file system as the XML document and should be given filename as Proj.dtd

<!ATTLIST element name , att type att value >

① att type → ID
can hold values like → CDATA
ID, CDATA (character → XML
type data, XML

② att value
#REQUIRED
#FIXED
#IMPLIED (optional)
#DEFAULT

e.g → <!ATTLIST Project Proj id ID #REQUIRED>

XML Schema

<?XML version = "1.0" encoding = "UTF-8" ?>
<xsd : schema xmlns : xsd = "http://
www.w3.org/2001/XMLSchema" >
<xsd : annotation>
<xsd : documentation xml : lang = "en">
Company Schema </xsd : documentation>
</xsd : annotation>
<xsd : element name = "company" >
<xsd : complexType >
<xsd : sequence >

<xsd : elementname = "dept" type = "dept"
minOccurs = "0" maxOccurs = "unbounded" />
< ——— "employee" — "employee" ——— />

[attributename : value]

```
<xsd : uniquename = "deptName unique" />
<xsd : selector xpath = "employee Dependent" />
<xsd : field xpath = "dependentName" />
    </xsd : unique>
    </xsd : sequence>
  </xsd : complex>
```

(1) Schema Description and XML namespaces – It is used to identify specific set of XML schema language elements being used by specifying a file stored at website location. Each defination is called XML namespace because it defines the set of commands that can be used.

(2) Annotations, Documentation & language used – It is used for providing comments & description in XML document.

Types

(3) Element and ~~Tags~~ – In this we specify root element of XML schema i.e Company. The structure of company root element can be specify with <xsd : ComplexType>

(4) 1st level element in Company database – In this eg 3 first level tables are used dept, employee, Project

* If a tag has only attributes and no further sub elements or data within it. It can be ended up with '/' symbol & these are known as empty elements

(5) specifying element Type, ~~min~~ min & max occurances

(6) Specifying keys

Primary key
```
<xsd : keyname = "projectnokey" >
<xsd : selector xpath = "project" >
<xsd : field xpath = "proj number" >
</xsd : key>
```

Foreign key
```
< xsd : keyrefname =          >
_____
____
</xsd : key>
```

(7) Specifying the structure & complex elements via complex Types

(8) Composite Attributes

XML languages

XQuery — SQL query

Xpath — path specified after matching condition

(1) XPath returns a sequence of items that satisfy a certain pattern as specified by expression and these items are either values or elements or attributes. An additional qualifier condition restricts the node that satisfy the pattern. Two main separators are used to specify a path // , /

/ ⇒ It specifies that the tag must be a direct child of previous or parent tag

// ⇒ It specifies that the tag appear as a descendent of prerious tag at any level

e.g ⇒ /company , /company/dept ,
//employee [employee sal gt 70000] /
empname ,
/company/

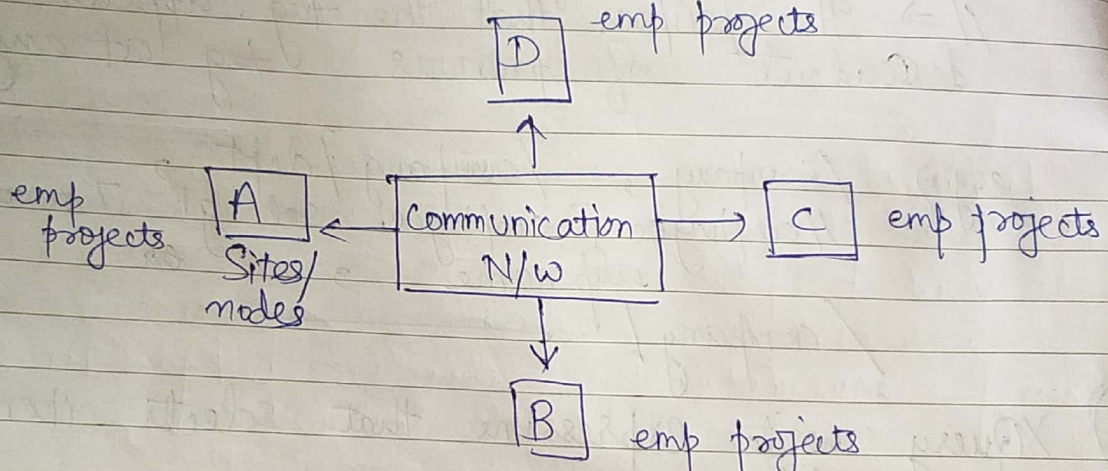
(2) XQuery write expressions that selects items from tree model

FLWR
→ for (binding to individual nodes)
→ LET (collection of —)
→ WHERE (qualifier condition)
→ RETURN (query result)

e.g → (1) LET $d = doc (www.company.com/info = xml)
FOR $x IN $d /company / project
[Prgnumber = 5-9] / Prg worker ,
$y IN $d / company / employee
WHERE $x / hours gt 20.0 AND
$y.ssn = $x.ssn
RETURN <res> $y / employee / fname / lname ,
$x / hours </res>

(2) FOR $x IN
doc ("www.company.com/info.xml")
// emp [esal gt 70000] / ename
RETURN <res> $x / fname </res>

Distributed Databases (DDBMS)



→ It is a collection of multiple logically interrelated databases distributed over a computer network and a ddbms as a software system that manages a distributed database while making the distribution transparent to the user

→ For a database to be distributed following minimum conditions should be satisfied -

(1) Connection of database nodes over a computer network
(2) logical interrelation of the connected databases
(3) Absence of homogeneity constraint among connected nodes. It is not necessary that

all the nodes be identical in terms of hardware & software.

- Transparency - It extends the general idea of hiding implementation details from end user. A highly transparent system offers a lot of flexibility to end user since, it requires no awareness of underlying details on their path. In case of traditional centralized databases, transparency pertains to logical & physical data independence for application developers.

Following types of transparency are possible -

(1) Data organization transparency / Distribution or network transparency
It refers to freedom from users from operational details of n/w & placement of data over the network.
It is of 2 types -
(1) location transparency
(2) naming "

(2) Replication transparency
Copies of the same data objects may be stored at multiple sites for better availability, performance & reliability. It makes the user unaware of the existence of these objects.

(3) Fragmentation transparency
There are 2 types of fragmentation -
(1) horizontal fragmentation
(2) vertical "

It distributes a relation/table into sub relation that are sub sets of tuples in the original relation whereas in vertical fragmentation it distributes a relation into sub relation that are subsets of columns of the original relation

(4) Design transparency
(5) Execution transparency

- **Autonomy** - It determines the extend to which individual nodes in a connected distributed database can operate independently. A high degree of autonomy is desirable for increased flexibility & customized maintainance of an individual node.

It is of 3 types -
(1) Design
(2) Communication
(3) Execution

- Design ⇒ It refers to the independence of data model use and a transaction management techniques among nodes
- Communication ⇒ This Autonomy determines the extend to which each node can decide sharing of information with other nodes.

- Execution ⇒ It determines the independence of user how they act with these distributed site / nodes.

- **Reliability + Availability**
  It specifies as the probability that a system is running at a certain ~~point~~ time point whereas Availability is the Probability that a system is continuously available during a time interval. We can directly relate reliability & availability with faults & errors These faults & errors are to be connected with fault tolerance techniques

→ To construct a system i.e reliable we can adopt several approaches. One common approach is stresses fault tolerance (It recognize that fault will occur and design mechanism that can detect & remove faults before they can result in a system failure)
Another strategy approach is to ensure that system don't contain any of the faults & this is done through exhaustive design process by extensive quality control & testing.

Advantages of DDBMS

1. Improve ease & flexibility of application development
2. Increased reliability & availability
3. Improved performance
   Data localization reduces the contention for CPU and I/O services and simultaneously reduces

access delays involved in wild area n/w. As a result, the local queries and transaction accessing data at single site have better performance because of smaller local databases. Moreover, inter query & intra query parallelism can be achieved by executing multiple queries at different sites or by breaking up a query into no of sub queries that executes in parallel so, this improves the performance.

(4) Easier expansion

Additional fx of DDBMS
↳ Function

(1) Keeping track of data distribution
(2) ~~Distribution~~ Distributed query processes
(3) Distributed transaction management
(4) Replicated data management
(5) Distributed database recovery
(6) Security
↳ Proper access or priviledges give to user
(7) Distributed directory or catalog management
↳ A directory contain information about data in database. It may be global for the entire distributed database or local for each site.

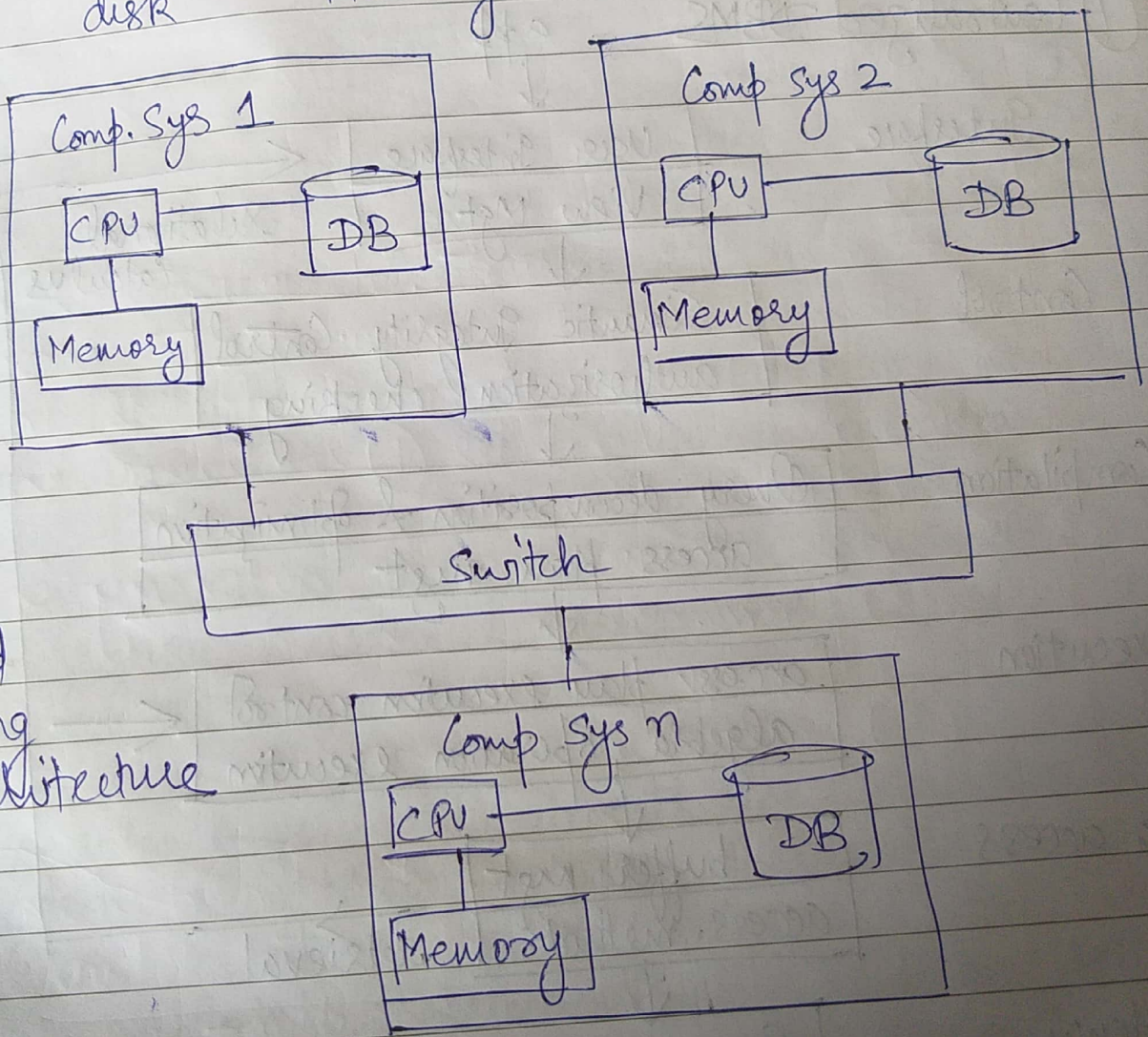
Types of DDBMS

(1) Federated
(2) peer to peer
(3) Centralized

④ Pure distributed db
⑤ 3 tier CS architecture

Parallel V/S Distributed

① Shared memory → Shared disk + memory
② Shared disk → Shared disk but own memory
③ Shared nothing
/ disk \ Memory

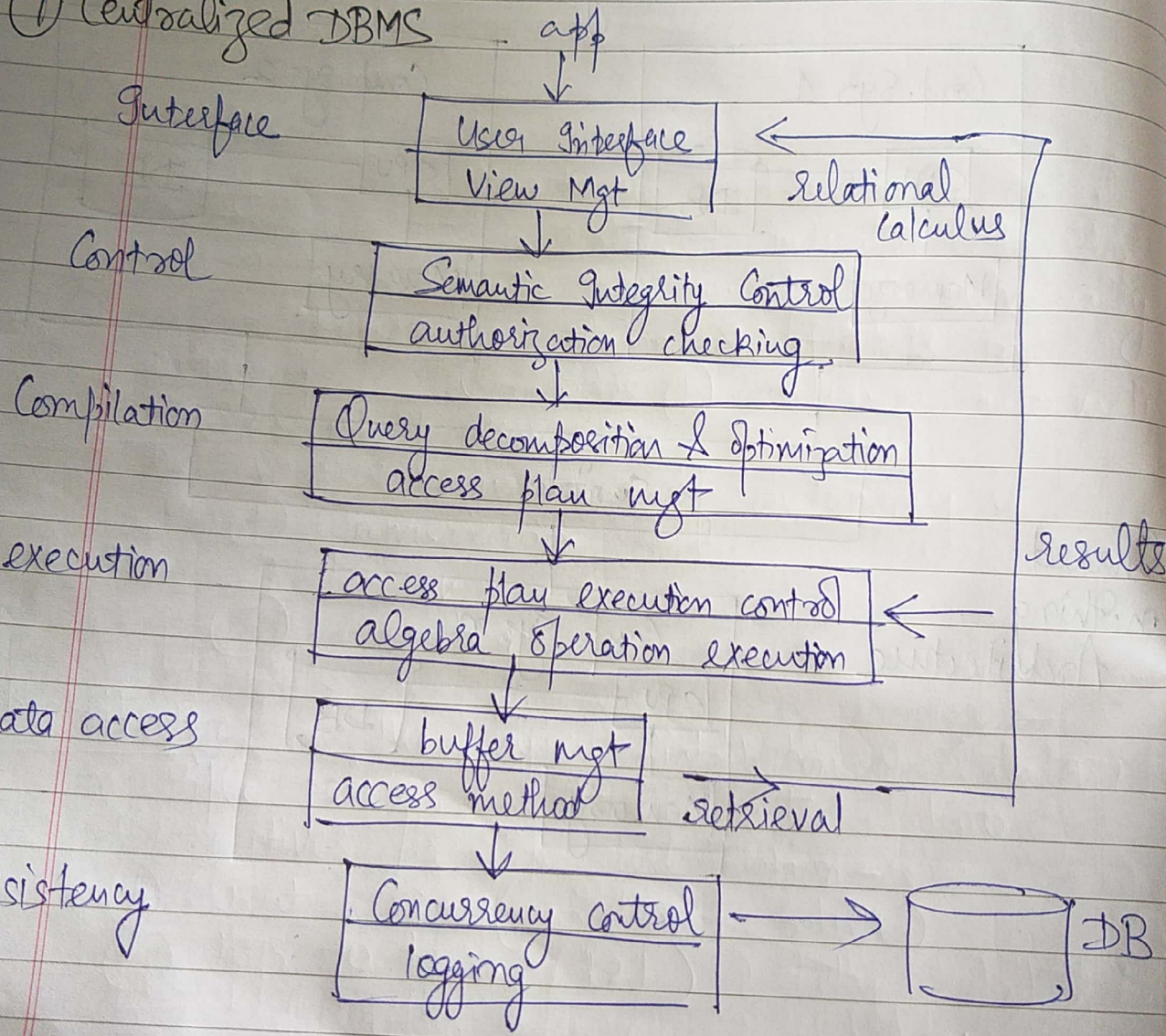


Shared nothing Architecture

Types of DDBMS

1. Client Server Architecture
   - 2-tier
   - 3-tier
2. Peer to Peer
3. Multidatabase
4. Centralized DBMS

1. Centralized DBMS



It follows top to bottom approach.
This is a centralized distributed database system so, various functional layers of Centralized DB are -

(1) I/F layer - It manages the I/F to the application there can be several I/F relational DBMS, several embedded I/F (such as C and QBE (Query By Example))

(2) Control layer - It controls the query by adding semantic predicates or constraints and authorization predicates. The O/P of this layer is enriched query in high level language accepted by the I/F.

(3) The Query processing / Compilation layer - It maps the query into sequence of optimized sequence of lower level operation. This is concerned with performance. It decomposes query into tree of algebra operation & trying to find the optimal ordering of operation.

(4) Execution layer - It directs the execution of access plan that includes transaction operation like commit revoke etc and it interprets the relational operation by calling the data access layer through the retrieval and update request

(5) Data access layer - It manages the data structure that implements the files and indices etc. Also manages the buffer by caching most

frequently accessed data.

(6) Consistency layer - It manages the concurrency control and logging for retrieval of updated data

→ To make this system different from each other, we have various factors -

(1) autonomy
(2) homogeneity
(3) distribution.

Autonomy It refers to the distribution of control, not of data. It indicates the degree to which individual DB can operate individually or independently. A no. of factors in autonomy includes (1) exchange information (2) modification etc.

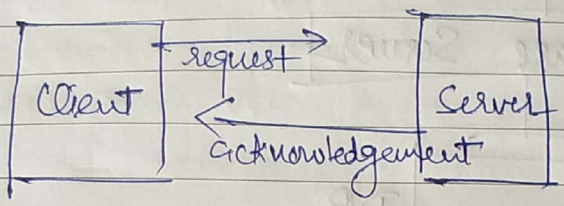
→ The various dimensions of autonomy is
(1) Communication
(2) Design
(3) execution

Distribution The Physical distribution of data over multiple sites. It abstracts two alternatives for the distribution -
(1) Client Server (2 tier + 3 tier)
(2) Peer to Peer (P2P) ⇒ fully distributed database management system.

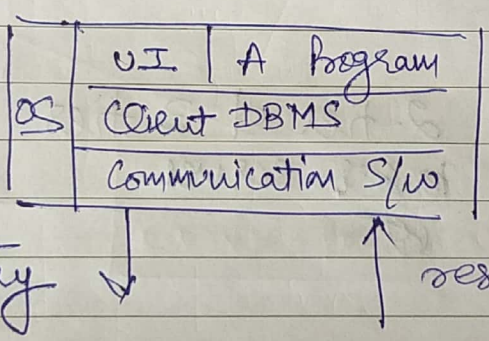
Homogenity It is of 2 types -
1. heterogeneous
2. homogeneous

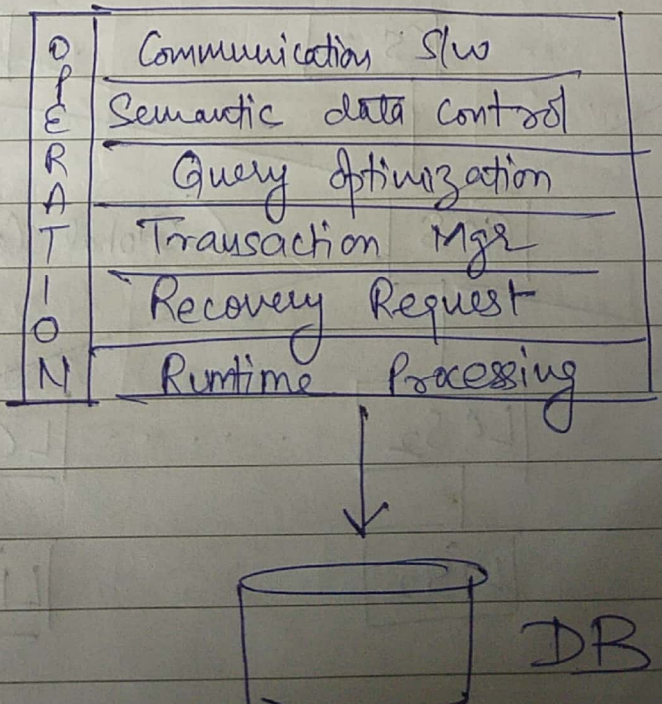
If all the server and client / user uses the identical software & hardware then that will become degree of homogenity (homogeneous) otherwise heterogeneous.

(2) Client Server Architecture

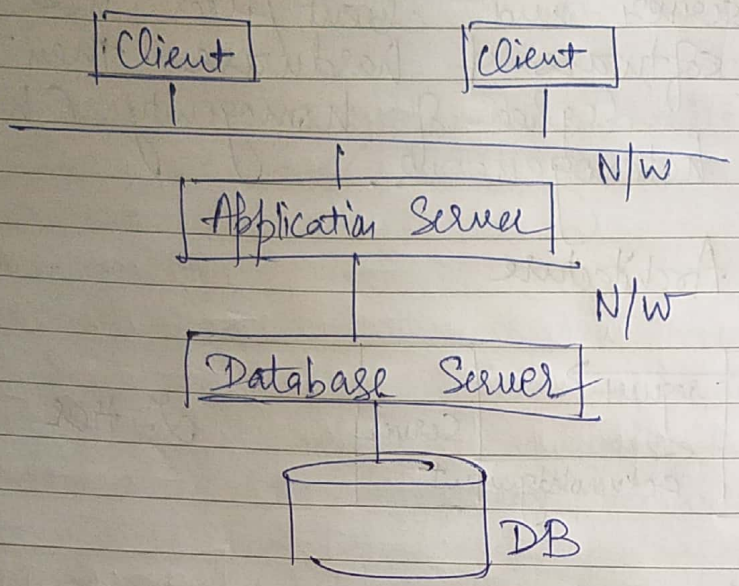


2-tier





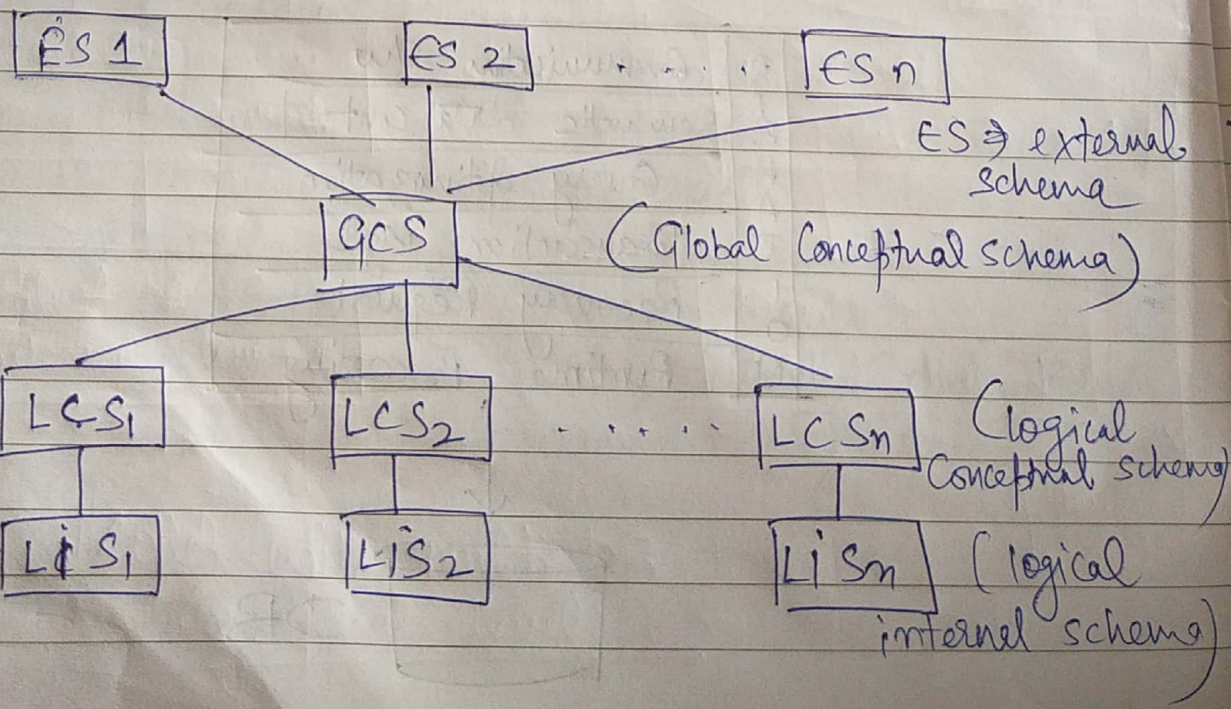
(1) Multiple Client / single server
(2) Multiple Client / Multiple server



3 tier

→ Difference b/w 2-tier & 3 tier is just of application server

(3) Peer to Peer



When an individual node consider itself as client and server then it is called as Peer to Peer architecture. It components are -

① ES
② GCS ③ LCS ④ LIS

The Physical data organization on each machine may be different. This means that there needs to be an individual internal schema defination at each site which we call as LIS (local internal schema)

→ The enterprise view of data is described by GCS (Global conceptual schema). It describes the logical structure of data at all of the sites

→ To handle fragmentation and replication the logical organization of data at each site, need to be described so, the 3rd layer in architecture is LCS included finally the user app and user access to Db is supported by ES (external schema)

→ location of replication transparency are supported by LCS and GCS with mapping whereas N/w transparency is supported by GCS alone

→ The detailed components of this architecture are as follows -

① User I/F handler - It is responsible for interpreting user commands, formatting result & sent back to user

② Semantic Data Controller - It uses the integrity constraints & authorization constraint that checks whether the user's query can be processed or not.

(3) Global query optimizer and decomposer - It determines the execution strategy to minimize post fxn

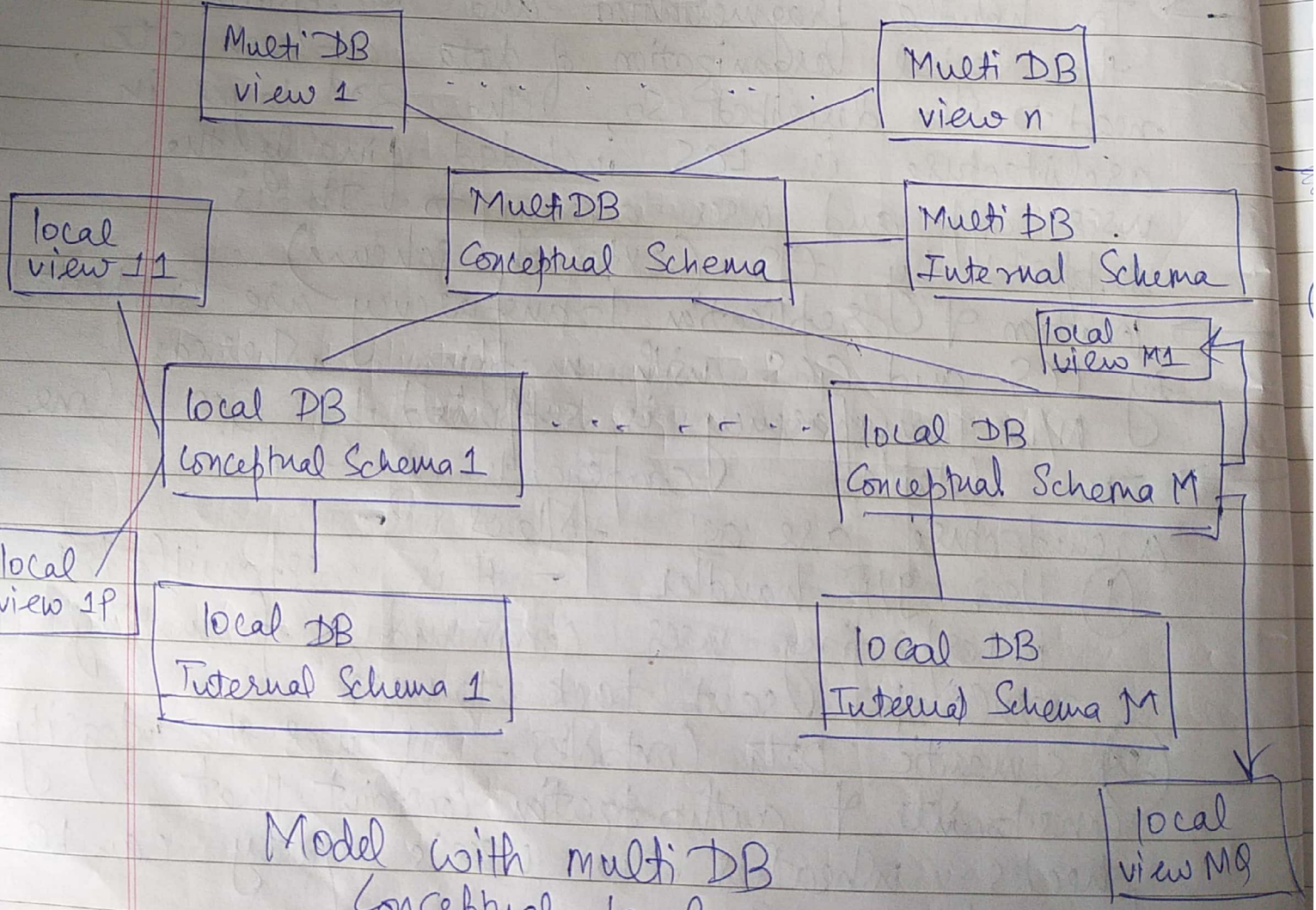
(4) Distributed execution monitor - It coordinates the distributed execution of user request.

→ LLS ⇒ how to organise data physically at all sites.

(4) Multi Database

An integrated Database System formed by a collection of two or more autonomous database system.

It can be expressed through six levels of Schema



Model with multi DB

Conceptual level

① Multi DB view level - It depicts multiple user views comprising of subsets of the integrated distributed database

② Multi DB conceptual level - It depicts integrated multi DB that comprises of local logical multi DB structured definitions.

③ Multi DB Internal ~~Schemas~~ level - It depicts the data distribution across different sites and multi DB to local data mapping.

④ Local DB view level - It depicts public view of local data.

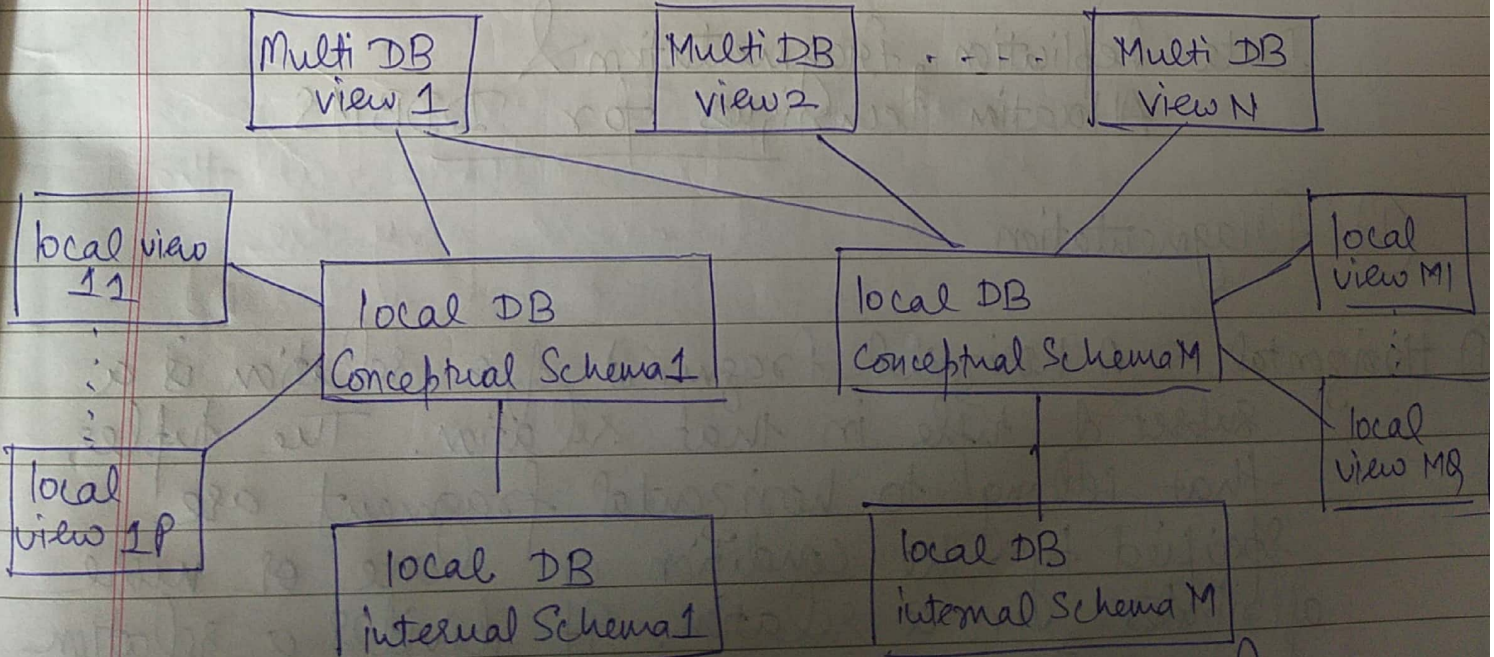
⑤ Local DB conceptual level - It depicts local data organization at each site.

⑥ Local DB Internal level - It depicts physical data organisation at each site.

→ There are two design alternatives for multi DBMS

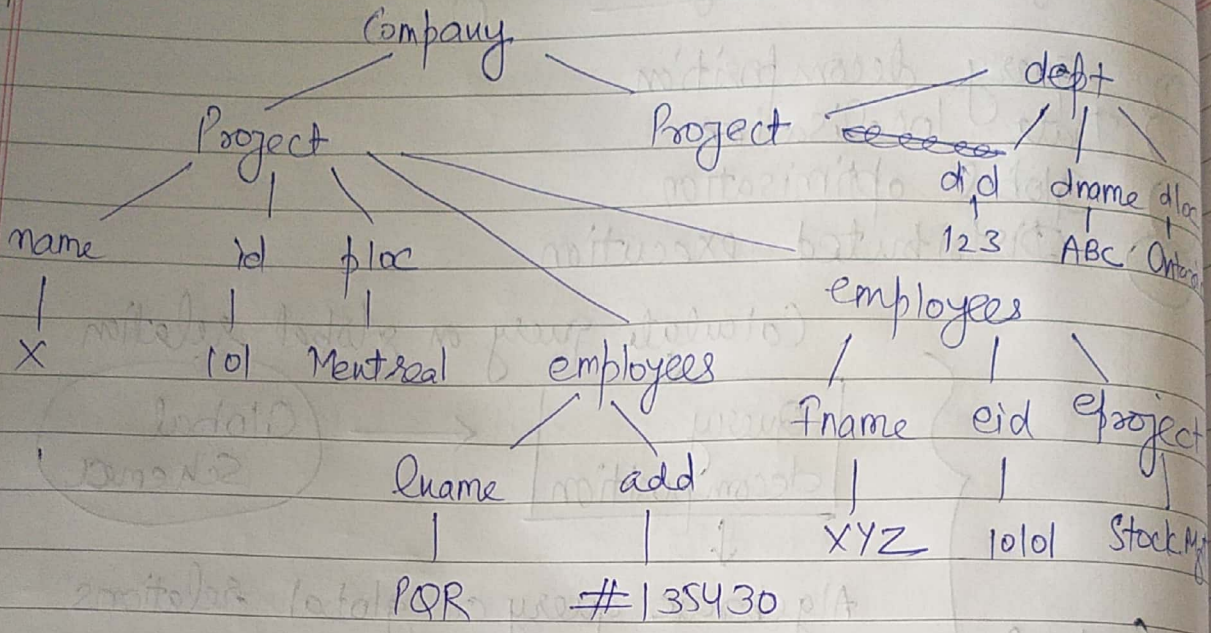
① Model with multi DB Conceptual level

② Model without multi DB Conceptual level



Model without Multi DB Conceptual level

Q Create a Document



Q Create a graph

```
<?XML Version="1.0"?>
<emp info>
<employee employee_no="105">
        <name> ABC </name>
        <dept> —— </dept>
        <telephone> — </telephone>
        <email> —— </email>
</employee>
</emp info>
```

<company>
<Project>
<name> X </name>
<id> 101 </id>
<ploc> Hertzsol </ploc>
<employee

Design Alternative

The distribution design alternatives for the tables in a DDBMS are as follows -

1. Non replicated and non fragmented - Different tables are placed at different sites
2. Fully replicated - One copy of all the database tables is stored at each site. It has negligible communication cost.
   - ↳ It has high updation cost.
3. Partially replicated - Copies of tables or portions of tables are stored at different sites.
4. Fragmented - A table is divided into two or more pieces refer to as fragments or partitions and each fragment can be stored at different sites
   1. Vertical
   2. Horizontal
   3. Hybrid
5. Mixed - combination of fragmentation & partial replication.

Data Replication, fragmentation & Allocation Techniques for DDBMS

1. Fragmentation

1. Horizontal - A horizontal fragment of a relation is a subset of tuple in that relation. The tuples that belong to horizontal fragment are specified by the condition on one or more attributes in a relation. It divides a relation horizontally by grouping rows to create subset

of tuples where each subset has its own meaning

Types of horizontal ⇒
1) Primary
2) Derived
3) Complete
4) Disjoined

Derived ⇒ It applies the partitioning of a primary relation to other secondary relation which are to be related via primary and foreign key.

Horizontal fragmentation on a relation R can be specified in relational algebra by
$\sigma C_i(R)$

Complete ⇒ A set of horizontal fragments whose conditions $C_1, C_2, \ldots C_n$ includes all tuples in R i.e every tuple in R satisfies these conditions is called Complete horizontal fragmentation

When no tuple in R satisfies (C_i and C_j) for any $i \neq j$
To reconstruct this relation for a complete horizontal fragmentation, we need to apply union operation.

eid	Name	Area
1	X	A1
2	Y	A2
3	Z	A3
4	AA	A4

eid	Name	Area
1	X	A1
2	Y	A2

Horizatal F 1

eid	Name	Area
3	Z	A3
4	AA	A4

Horizatal F 2

② Vertical – It divides a relation vertically by columns. It is necessary to include primary key or some candidate key in every vertical fragment so, that the full relation can be reconstructed from the fragments. A ~~the~~ vertical fragment on a relation R is to be represented by –

$\Pi L_i(R)$

★ A set of vertical fragments whose projection list $L_1, L_2, \ldots L_n$ include all the attributes in R but share only the primary key attribute of R is called Complete vertical fragmentation

eid	Name	Area
1	X	A1
2	Y	A2
3	Z	A3
4	AA	A4

eid	Name
1	X
2	Y

V f1

eid	Area
3	A3
4	A4

V f2

We will use full outer join operation to reconstruct this vertical fragmentation to make it complete vertical fragmentation.

(3) Hybrid/Mixed - When a relation is composed of vertical and horizontal fragmentation (rows and columns) that becomes hybrid (mixed) fragmentation

e.g select * from empname where age < 40

Q

Account no	Balance	Brauchname	Type
		Delhi	
		Mumbai	
		Pune	

Make horizontal, vertical fragments of this table

② Data Replication

Types of Replication -

① Fully replicated
② No replication
③ Partial replication

Fragmentation schema of a Db is a set of fragments that includes all attributes and tuples in a Db and satisfy the condition of the Db that whole Db can be reconstructed from the fragments by applying sequence of outer join and union operation.

Allocation schema describes allocation of fragments to sites of distributed Db. Hence it is a mapping that specifies for each fragment the site at which it is to be stored. If a fragment is stored at more than one site then it is known as replicated.

Query processing & optimization in DDBMS

Q Employee

Fname	Minit	Lname	ssn	Bdate	Addrs	sex	salary	super_ssn	Dno

10,000 records
Each record is 100 byte is long
ssn → 9 bytes
Dno → 4 bytes
Fname → 15 bytes
Lname → 15 bytes

Department

Dname	Dnumber	Mgr_ssn	Mgr_start_date

100 records
Each record is 35 byte long
Dnumber → 4 bytes
Mgr_ssn → 9 bytes
Dname → 10 bytes

a) $\Pi_{Fname, Lname, Dname}(EMPLOYEE \bowtie_{Dno = Dnumber} DEPARTMENT)$

1)

$10,000 \times 100 = 10^6$

$100 \times 35 = 3500$

$10^6 + 3500 = 1003500$

Transfer both employee & Dept relation to result site & perform the join at site 3
In this case, the total of $10^6 + 3500$
$= 1003500$ bytes must be transfered

(2) Transfer the employee relation to site 2, execute join at site 2 and send the result to site 3.
The size of query result is

$(15+15+10) * 10,000 = 400,000$
$4,000,000 + 10^6 = 1,40,0000$

(3) Transfer the Dept, relation to site 1
execute the join at site 1 and send the result to site 3:, In this case,

$400,000 + 3500 = 403500$

OR

$(15+15+10) * 10,000 = 400000$
$400,000 + 3500 = 403500$

b) Retrive Deptname, name of manager on the basis of SSn

40 x 100 = 4000

Overview of query processing

Q emp (eno, ename, title)
arg (eno, Pno, resp, dur)

① Select ename from emp, arg where emp.eno = arg.eno AND arg.resp = "Manager";
↳ relational Calallas

② Πename (σ resp = 'Manager' ∧ emp.eno = arg.eno (emp × arg));
↳ Relational Algebra

Find the name of employees who are managing a project. Two equivalent represents are shown above i.e Relational Calculas, Relational Algebra

Site 5
result = emp1' ∪ emp2'

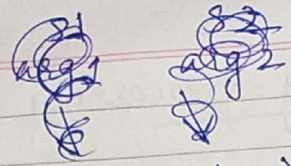
Site 3
$emp_1' = emp_1 \bowtie_{eno} arg_1'$

Site 4
$emp_2' = emp_2 \bowtie_{eno} arg_2$

Site 1
$arg_1' = \sigma_{resp='Mgr'} arg_1$

Site 2
$arg_2' = \sigma_{resp='Mgr'} arg_2$

Strategy A



Site 5
result = (emp1 ∪ emp2) ⋈ eno (σ resp='Mgr' (arg1 ∪ arg2))

Ans

S1

emp1 = σ eno ≤ 'e3' (emp) Site 3
emp2 = σ eno > 'e3' (emp) Site 4
arg1 = σ eno ≤ 'e3' (arg) Site 1
arg2 = σ eno > 'e3' (arg) Site 2

S2

S3

Strategy B

To evaluate the resource consumption of these 2 strategies we, use a simple cost model.

S

We assume that a tuple access (tupacc) is 1 unit and tuple transfer (tupletrans) is 10 units.

We assume that the relational tables emp and arg have 400 and 1000 tuples respectively and There are 20 Managers in Arg relation

We assume data is uniformly distributed among the sites.

finally we assume that relations arg and emp are locally clustered on attributes resp and eno respectively. . ., there is direct access to tuples of arg, emp based on the value of attribute resp and eno.

Ans For Strategy A

(S1) ~~Total Posts~~ Reduce arg' by selecting arg. requires $(10+10) * 1 = 20$

(S2) Transfer arg' to the site of emp requires $(10+10) * 10 = 200$

(S3) Produce emp' by joining arg' and emp. It requires
$(10+10) * 1 * 2 = 40$
↳ By default (if not given)

(S4) Transfer emp' to the result site. It requires
$(10+10) * 10 = 200$
↳ tup trans

Total Posts $= 20 + 200 + 40 + 200 = 460$

For Strategy B

(S1) Transfer emp to Site 5 requires
$400 * 10 = 4000$
↳ tup trans

(S2) Transfer arg to Site 5 requires
$1000 * 10 = 10,000$

(S3) Produce arg' by selecting arg
$1000 * 1 = 1000$
↳ tupaccess

(S4) join emp and arg' requires
$400 * 20 * 1 = 8000$
↳ tupaccess

(1) (2) (3) (4)

Total Posts = $1000 + 8000 + 4000 + 10,000$
$= 23,000$

As Total Posts in A < B
So,
Strategy A is better than B

Relational Algebra operation

Operation	Complexity
(1) Select, Project (without duplicate elimination)	O(n)
(2) Select, Project (with duplicate elimination)	O(n * log n)
(3) Join, semi join, deletion, set operator	O(n * log n)
(4) Cartesian	$O(n^2)$

layers of query Processing

1. query decomposition
2. Data localisation
3. Global optimisation
4. Distributed execution