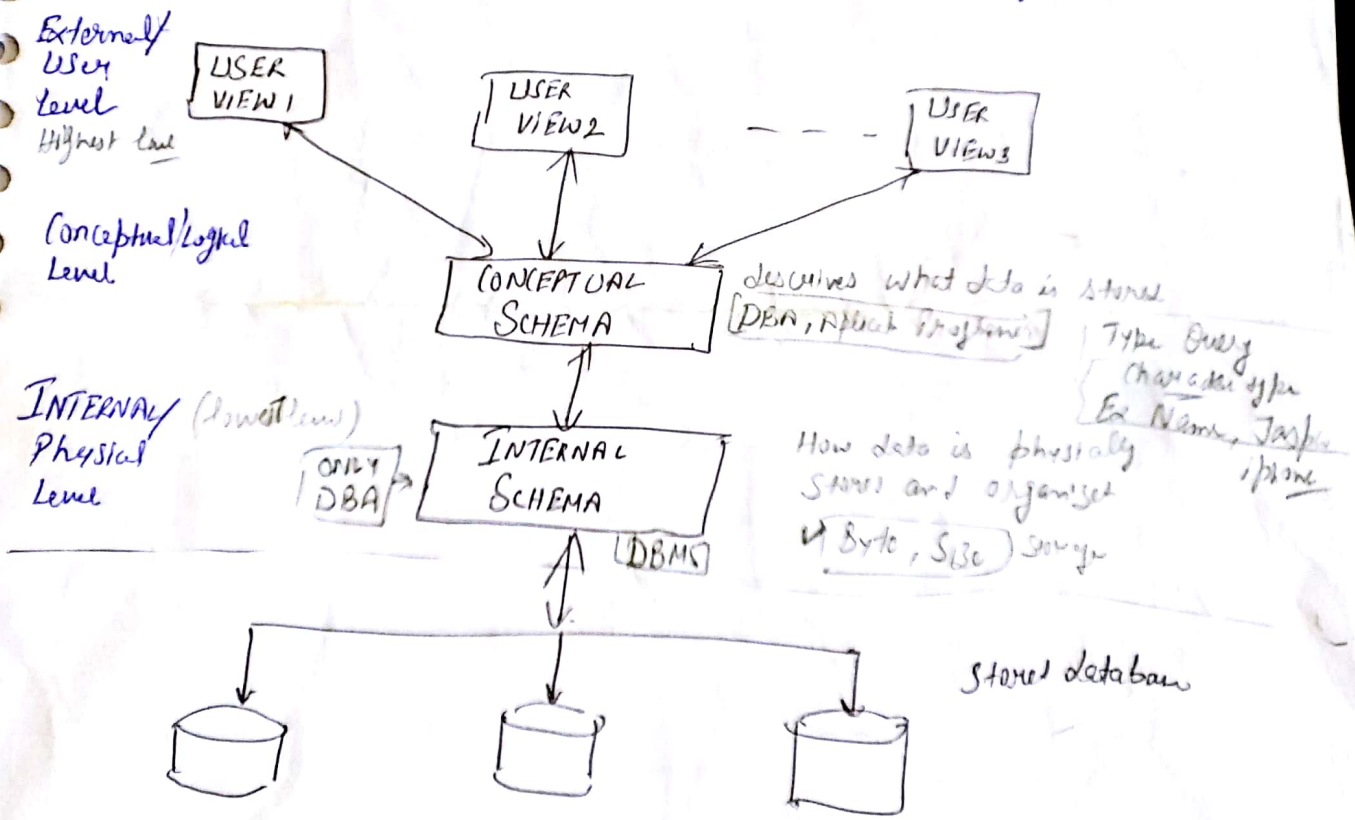
Jaspreet Singh OE Lecture-2 (9592252401)

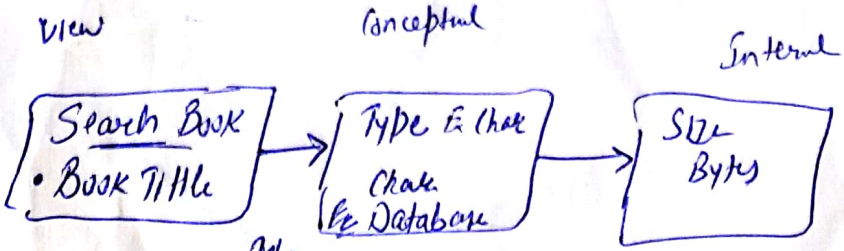
DBMS ARCHITECTURE → describes how data in database viewed by user.

- DBMS architecture is called three-level architecture.
- The goal of three level architecture is to separate the user application and physical database.
- The major purpose of DBMS is to provide user with an abstract view of data.

Abstraction means to hide certain details of how data is stored and maintained & complexity



Ex



① Book → OOPS' → SDE

② ISBN NUMBER → 12 Digit Number → SDE.

Ex When Book name = OOPS'

Conceptual

1 Physical Level

Ex (char name [20])
Int ID [10]
Physical

- It is the lowest level of abstraction.
- It describes how data is actually stored
- Physical level has an internal schema which describes the physical storage structure of database.

2 Conceptual Level

Table

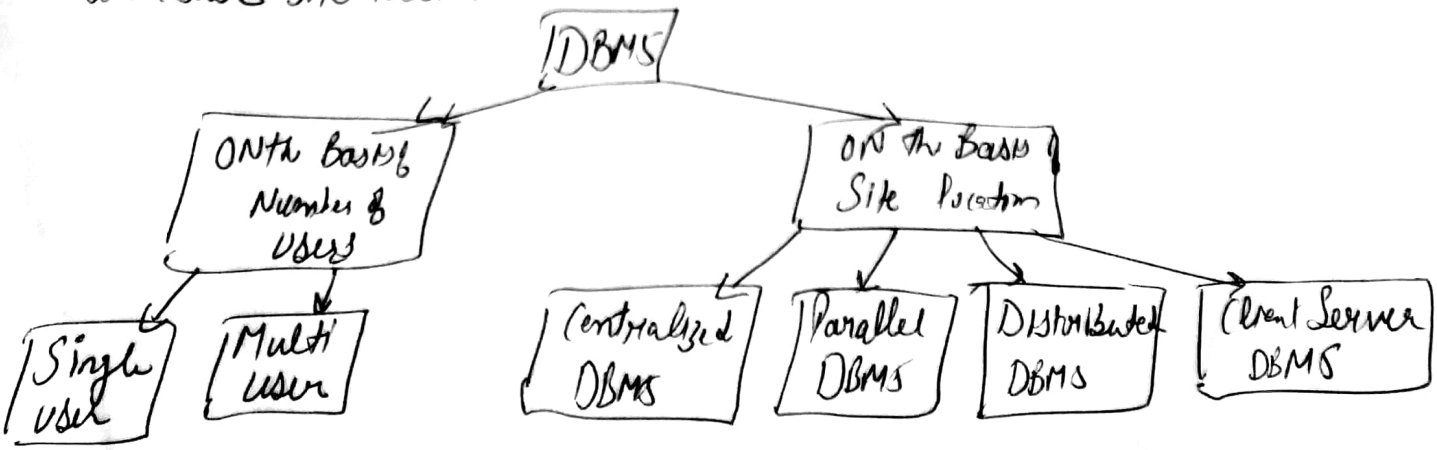
- It is the next higher level of abstraction.
- This schema hides the details of physical storage structure and concentrates on describing entities, data types, relationships, and constraints.

3 View Level

- This is highest level of abstraction.
- The view level includes number of user views.
- Different users may need only a part of the entire database. So it describes only required part.

Types of DataBase System

DBMS is classified on the basis of number of users and the database site location.



Classification of DBMS

1) On the Basis of Numer of Users

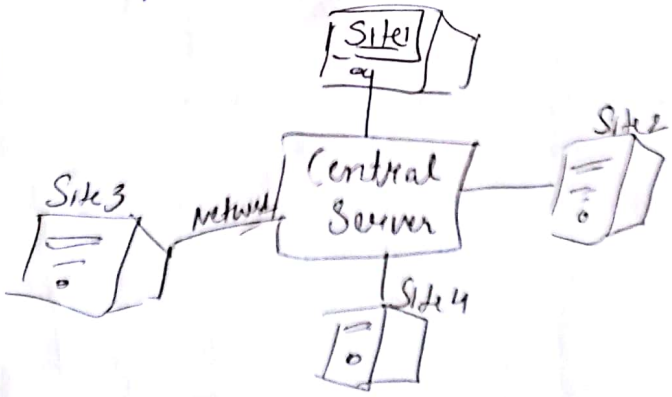
(1) Single User DBMS :- In single user DBMS, database resides on one computer and it is only accessed by one user at a time.

(2) Multiuser DBMS :- The multiple users access the data frome one central storage area so that the database integrated. [Integrate means data is not stored in two or more places] => One place only.

2) On the Basis of Site Location

1) Centralized System.

- In centralized system, the database resides on some single central location.
- A number of processors can access this central database via ~~some~~ some computer network.
- These processors can connected to the central database via some computer network.



Centralized DBMS

Eg Railway Reservation System.

- Where the database is centrally located at New Delhi.

Disadvantage

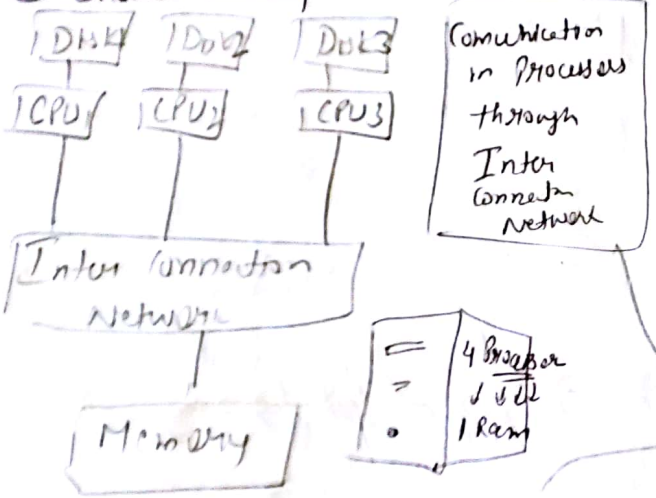
When central site computer goes down, Then the users are blocked from using system until the system comes back.

2 Parallel Database System

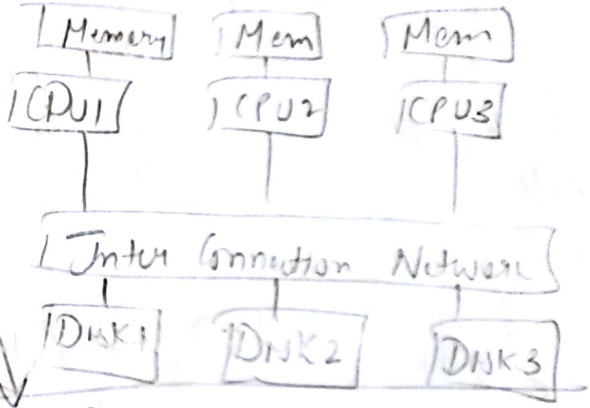
- Parallel database system architecture consists of Central Processing Units and data storage disks in parallel.
- They improve processing and Input/Output (I/O) speed.
- Parallel database systems are used in the applications that have to process an extremely large number of transactions per second or have to query large databases.

There are three architecture in Parallel DBMS

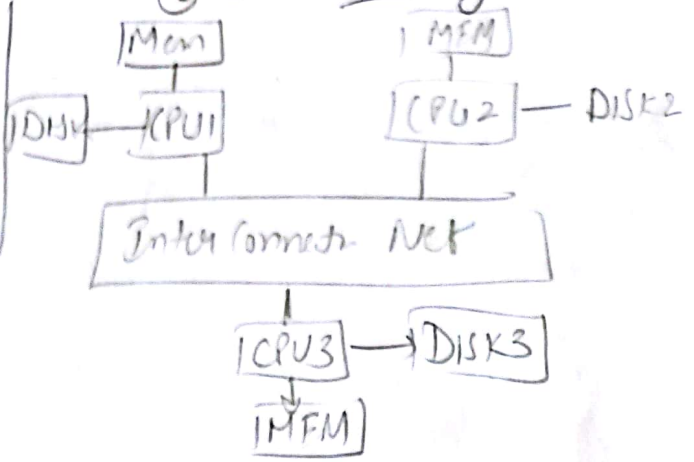
① Shared Memory



② Shared Disk



③ Shared nothing



ⓐ Shared Memory (Ram)

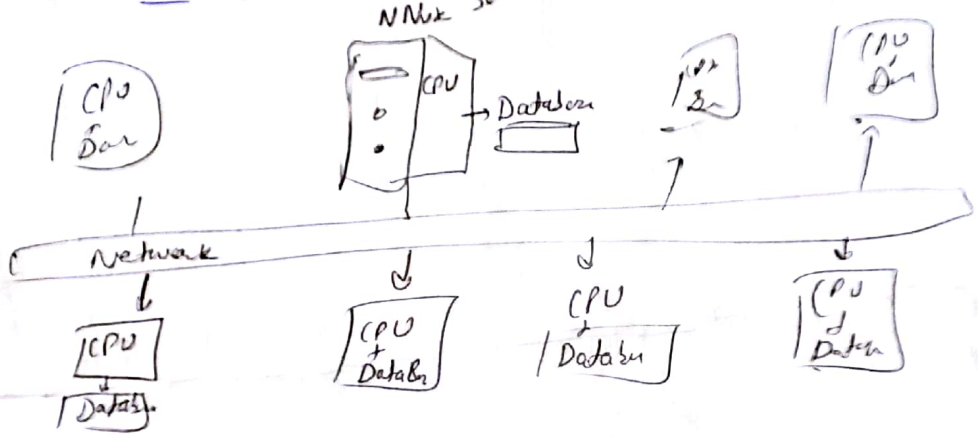
Blocks of Ram can be used by different CPUs (Processors) in Multiprocessor Computer System

Disadvantage

- It uses share resources.
- For eg - If Disk 1 corrupt Not any system can use/access Disk1 data,

3) Distributed Database System

- In distributed DBMS, the database is split into number of fragments.
- Each fragment is stored on one or more computers.
- These computers are connected by a communication network.
- The computers in distributed system may vary in size and function.
  known as nodes or sites



Advantage

- Scalability
- No problem if any node fails. (Replication factor)

Disadvantage

- Recovery from failure is complex.

4) Client-Server DBMS

- Client/Server architecture of database management system has two logical components, namely client and server.
- Clients are generally personal computers or workstations
- whereas servers are large workstations.
- The application and tool of DBMS run on one or more client platforms while DBMS software reside on the server.
- Server handles clients requests for access to database access & update
- Client is front end, Server back end.

imp. *

Data Independence

- In old days, programs stored data in system files.
- Each program has to maintain its own data
  - ✓ huge overhead.
  - ✓ error prone.
- Development of DBMS helped to achieve data independency (transparency).
- Provide centralised controlled data maintainance and Access.
- Changes made at one level of DB system does not effects other levels.

(1) Logical DI :

Capacity to change conceptual schema without changing external schema/application program.

i.e we can change conceptual Schema

⇒ expand Database by adding record type, data

⇒ To change constraints

⇒ To reduce Database (or removing data items).

(2) Physical DI :

Capacity to change internal schema without having to change conceptual Schema.

⇒ Changes in internal schema needed because :-

- Physical files were organized.
- Improve performance of retrieval or update.

Lecture 3

Jaspreet Singh (9592232481) (8) 507

PL/SQL Concepts

- SQL has some disadvantages : PL/SQL come into existence

Disadvantages of SQL

1) SQL doesnot have procedural capabilities :
   - SQL doesnot provide the programmers with the techniques of condition checking, looping and branching.

2) SQL statements passed to the Oracle engine one at a time. This lead to increase in traffic & decrease in the speed of data processing

3) ~~SQL Statements are passed to the oracle engine :~~

3) SQL has no facility for program handling of errors during manipulation of data.
   Oracle engine display their own errors.
   But in PL/SQL we have exception handling facility. Ex exception %

Advantages of PL/SQL

1 Block Structure :
- PL/SQL consists of blocks of code which can be nested with in each other
- It send entire block of SQL statement to oracle engine all in ~~one~~ one go.
- Reduce network traffic, it process the code much faster
- All changes are made to the data in the table are done or undone in one go.

2 Procedural Language capabilities:

(9)

- It consists of Procedural language constructs such as conditional statements and loops.

3 Better Performance

- PL/SQL process multiple SQL statements in a single block of code
- Reduce network traffic & Improve the performance time of Oracle engine.

4 Error Handling :

- Errors or exceptions are effectively handled during the execution of a program.
- Once an exception is caught specific actions can be taken depending on the type of exception.

5 Portability

- Applications written in PL/SQL are portable to any computer hardware or operating system where Oracle is operational.

⇒ PL/SQL What is PL/SQL

- PL/SQL stands for Procedural Language extension of SQL
- PL/SQL is combination of SQL along with procedural features of programming languages.
- It was developed by Oracle Corporation in early 90's to enhance capabilities of SQL

PL/SQL Block Structure

PL/SQL Block Consists of three sections

1. The Declaration Section (optional)
2. The ~~Exception~~ Section (mandatory)
   Execution
3. The Exception Section (or Error) Optional.

Sample

```
DECLARE
  Variable Declaration
BEGIN
  Program Execution
EXCEPTION
  Exception handling
END;
```

- Declaration Section (optional)

Declaration section start with DECLARE key word.
- Used to declare placeholder like variables, constants, records as cursors which stores data temporarily.

2 Execution Section

- Start with keyword BEGIN and Ends with END.
- This is mandatory section
- Where the program logic is written to perform any tasks.
- Like loops & conditional statements.

3 Exception Section

- Starts with EXCEPTION keyword.
- This section is optional
- Any errors in the program can be handled in this section

PL/SQL Block Structure

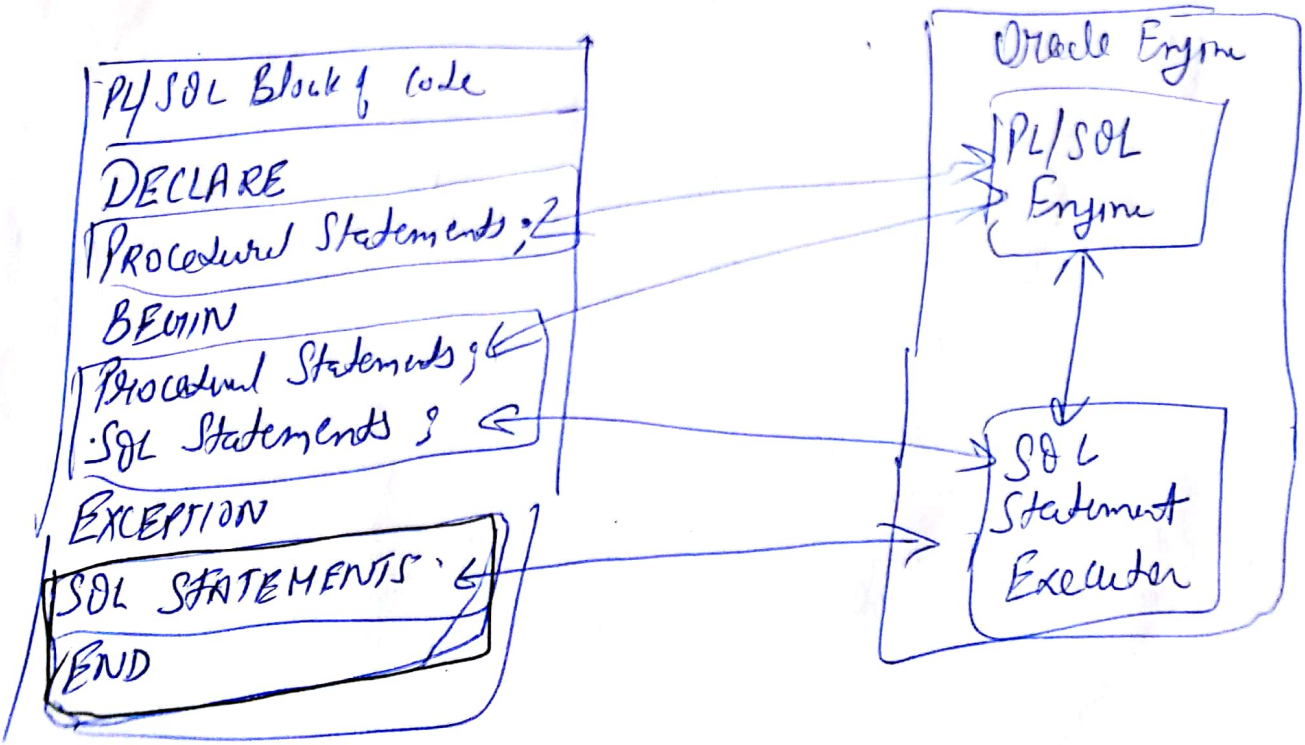
Declare	Declaration of memory variable Constants, cursors etc.
BEGIN	Executable Statements Queries., loops,
Exception	Code to handle errors.
End	END of PL/SQL Block

PL/SQL Execution environment

- Oracle Engine Consists of

PL/SQL Engine → Procedural Statements are Executed

SQL Statement Executor → SQL Statements are Executed



PL/SQL Datatypes

1. Number → For storing numeric data : 123
   number id (10) — decimal places
   id (10,3) → 12345
2. Char → For storing character data ex a, b, c
3. Date → For storing date & time data.
4. Boolean → For storing True, False or Null value
5. Varchar → For storing Alpha Numeric value

Varchar
① 2000 byte
Length difference.
② Occupy space for null values.

Varchar 2
4000 byte
• Doesn't occupy

Place holders

2. Literal, (Variable, Constant)

① Literal - value

It is numeric value character or a string.

Numeric Literal
- Integer → 25
- Float → 2.35

String Literal
Ex 'TOM'
② Single character 'a', 'a'

Characters as a set of characters enclosed within single quotes.

PL/SQL Placeholder

- Placeholders are temporary storage area.
- Placeholders can be variable or constant.

Variable

- Variable name must begin with a character & can be followed by maxm 29 characters.
- Reserved words can not be used as variable names unless enclosed in double quotes.

Syntax

Variable name Datatype (Size);

Eg salary number (10);

Assigning Value

1) Salary number (10) := 10000;

2) Salary := 20,000;
   Variable name := value;

3) Selecting or fetching table data value into variable.
   (variable) (table)

Eg Select ~~Salary~~ Stipend into ~~Salary~~ Salary from emp
   where id = 2;

XML

- Stands For Extensible Markup Language.
  - Extensible → to extend its programming
  - Markup → tags designed for process, definition, presentation & text layout
  - Language → that uses tags to define elements within a doc.
- Widely used on World Wide Web as a means of communication between different types of web components.
- It is a W3C (World Wide Web Consortium) Standard
  International Web Development Organisation
- XML is Meta Language. This means that you use it for creating languages. (You can create your own markup languages)
- XML is an extensible concept → It is designed to be a self descriptive.
  - we can create tags.
  - create own tags.

XML Usage :

1. XML is mainly used for Transporting, Storing and Sharing Data.

   Ex. • Software Updates.
   - Web based feeds, such as weather reports, stock updates, Jobs etc.
   - It is also used in many other aspects of Web Development and Third party Applications.

- Every XML document is text based and platform independent.
- XML is a Markup language much like HTML.
- A simplified version of SGML (Standard Generalized Markup Language)
- It can describe data in tree or graph structures.

Lecture - 5

XML

XML Stands for eXtensible Markup Language. (9)

Markup :-

(1) Plain Text
Easy to Edit
Platform Independent

Important (Importance) Characteristics of XML: (1) Platform Independent

(1) XML is extensible:
It allows you to create your own self-descriptive tags or language, that suits your application. → and document structure

(2) XML carries the data doesn't present it.
It is markup language much like HTML
But XML allows you to store data, irrespective of how it will presented.

(3) XML is a public Standard:
XML was developed by organisation called the World wide Web (W3C).

XML is markup language that defines set of rules for incoming document in a format, that is both human readable & machine readable.
Markup is information added to a document that enhances its meaning in certain ways.

Ex <message>
<text> Hello GFG </text>
</message>

Markup symbols or tags. in above are <message> ... </message>

- XML Separates Data From HTML.
  - If you need to display dynamic data in your HTML document it will take a lot of work to edit HTML each time the data changes.
  - With XML, data can be stored in separate XML files. In this way you can concentrate on using HTML/CSS for display & layout only.
- XML Simplifies Data Transport.
  - One of the most time consuming challenges for developers is to exchange data between incompatible systems over the Internet.
  - Exchanging data as XML greatly reduces this complexity. Because XML is platform independent, either hardware or software.

XML is not a Replacement for HTML.

- It is most important to understand that XML is not a replacement for HTML.
In most applications. XML is used to transport data, while HTML is used to format and display the data.

→ XML Issues / Lacks

XML

Jaspreet Singh
9592 232401

1 Structured Data

- Relational Databases are highly Structured.
- You must define schema before entering any data.
- Every Row confirms to the table schema.
- Changing the schema is hard and may break many things.

2 Unstructured Data

- Data is Free form
- There is no predefined, and its hard to (like we can insert any data, / Ex restrict stuff)
- Reader need to infer structures and meanings. (obtain data)

web file, audio video, image.

3 Semi-Structured Data

- In Semistructured data, the schema information is mixed with the data values
- Combination of structured and unstructured data.

html code,
Book: - Chapters, sections, titles, Paragraphs, references, index etc.

There are two main structuring concepts to construct an XML.

① Element (tags)

② Attributes (

Elements

Everything from Element start tag to the element's end tag.

RULES

1) XML tags are Case Sensitive.

</Adress> This is a wrong </adress>

<Adress> correct </Adress> ✓

② XML tags must be closed in an appropriate order i.e an XML tag opened inside another element must be closed before the outer element is closed.

```
<outer_elent>
<internal_elent>
(Dntent
</internal_elent>
</outer_elent>
```

```
<Project>
<Worker>
  There are 5 workers.
</Worker>
</Projects>
```

③ An element name can contain any alphanumeric characters.

The only punctuation marks allowed in names are the hyphen (-), underscore (_) and period (.).

Attributes

- An attribute defines the property of the element.
- Attributes of element are separated by white spaces.
- It associates a name with a value.

attname = "value"

name is follow by an = sign and string value inside " " or ' ' single quotes.

Syntax

< element name attribute 1 attribute 2 >
... content
</ element name >

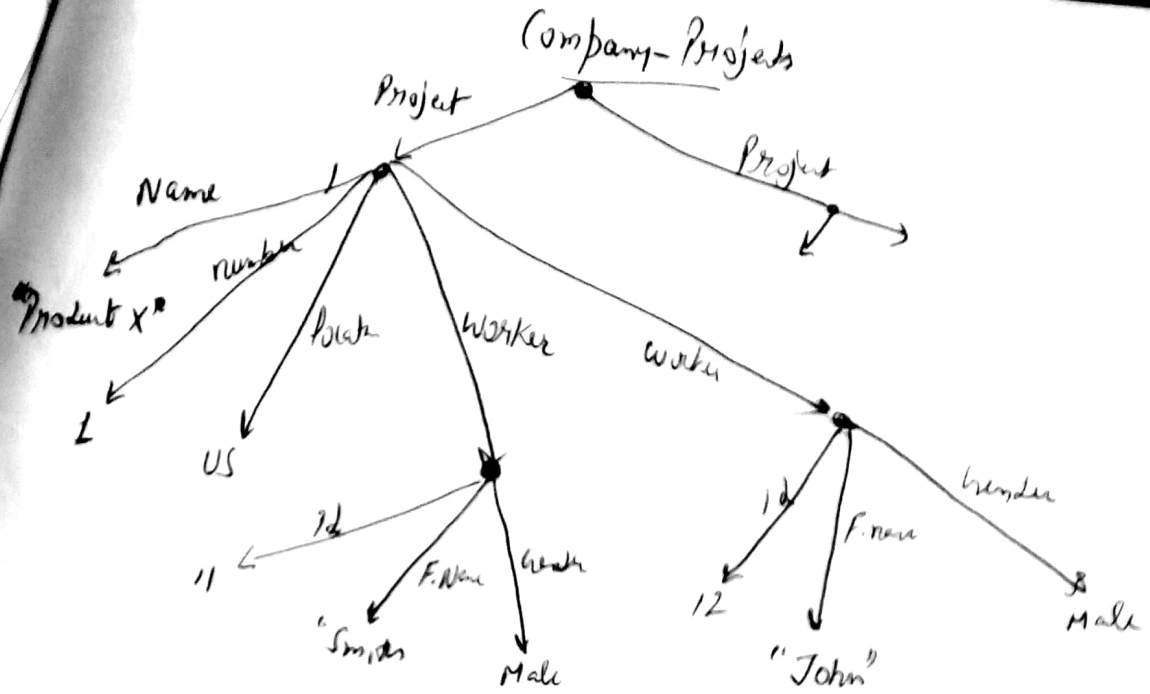
Empty Element

- Element with no content.

< Contact_info > </ Contact info >

Ex

< recipe name = "Bread" prep time = "5 mins" >
< title > Basic Bread </ title >
< ingredient amount = "8" unit = "grams" >
Flour
</ ingredient >
< ingredient amount "10" unit = "teaspoon" > Salt </ ingredient >
</ recipe >



XML Hierarchical (Tree) Data Model

Representing Semistructured data as a graph.

<Company_Projects>
<Project>
<name> Product X </name>
<Number> 1 </number>
<Location> US </location>
~~<Project>~~
<Worker>
<Id> 11 </Id>
<F.Name> Smith </F.name>
<Gender> Male </Gender>
</Worker>
<Worker>
<Id> 12 </Id>
<F.Name> John </F.name>
<Gender> Male </Gender>
</worker>
</Project>
</Company_Projects>

Validation is a process by which an XML document is validated. An XML document is said to be valid if its contents match with the elements, attributes, and associated document type declaration (DTD), and if the document complies with the constraints expressed in it. Validation is dealt in two ways by the XML parser. They are:

- Well-formed XML document
- Valid XML document

Well-formed XML Document

An XML document is said to be **well-formed** if it adheres to the following rules:

- Non DTD XML files must use the predefined character entities for **amp(&)**, **apos(single quote)**, **gt(>)**, **lt(<)**, **quot(double quote)**.
- It must follow the ordering of the tag. i.e., the inner tag must be closed before closing the outer tag.
- Each of its opening tags must have a closing tag or it must be a self-ending tag (<title>....</title> or <title/>).
- It must have only one attribute in a start tag, which needs to be quoted.
- **amp(&)**, **apos(single quote)**, **gt(>)**, **lt(<)**, **quot(double quote)** entities other than these must be declared.

Example

Following is an example of a well-formed XML document:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE address
[
   <!ELEMENT address (name,company,phone)>
   <!ELEMENT name (#PCDATA)>
   <!ELEMENT company (#PCDATA)>
   <!ELEMENT phone (#PCDATA)>
]>
<address>
   <name>Tanmay Patil</name>
   <company>TutorialsPoint</company>
   <phone>(011) 123-4567</phone>
```

WELL FORMED AND VALID XML

1. A doc is Well Formed XML Document =>
   - When it is structurally and syntactically correct. (that means tag are nested properly, case sensitive, elements close properly)
   - Adheres to general XML rules.

- It is not necessary that a well formed XML Doc is Valid. (Well Formed XML Doc: may be Valid or in valid.)
  - either it follows XML rules or it is syntactically correct.

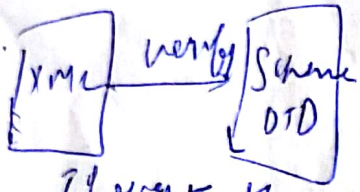
=> Because ~~Valid simply means~~ an XML Doc is Valid.
- When it ~~that this document~~ satisfies a schema document.

This mean there is another document, this XML document points to.

- This schema Document is known as DTD [Document Type Declaration]

Valid doc

If an XML document is well formed and has an associates ~~Document~~ DTD, Then it is said to be a valid XML doc.



If right then it is vali

```
</address>
```

The above example is said to be well-formed as:

- It defines the type of document. Here, the document type is element type.
- It includes a root element named as address.
- Each of the child elements among name, company, and phone is enclosed in its self-explanatory tag.
- Order of the tags is maintained.

Valid XML Document

If an XML document is well-formed and has an associated Document Type Declaration (DTD), then it is said to be a valid XML document. We will study more about DTD in the chapter XML - DTDs.

15. XML – DTDs

The XML Document Type Declaration, commonly known as DTD, is a way to describe XML language precisely. DTDs check vocabulary and validity of the structure of XML documents against grammatical rules of appropriate XML language.

An XML DTD can be either specified inside the document, or it can be kept in a separate document and then liked separately.

Syntax

Basic syntax of a DTD is as follows:

```
<!DOCTYPE element DTD identifier
[
   declaration1
   declaration2
   ........
]>
```

In the above syntax,

- The **DTD** starts with <!DOCTYPE delimiter.
- An **element** tells the parser to parse the document from the specified root element.
- **DTD identifier** is an identifier for the document type definition, which may be the path to a file on the system or URL to a file on the internet. If the DTD is pointing to external path, it is called **External Subset.**
- **The square brackets [ ]** enclose an optional list of entity declarations called **Internal Subset**.

Internal DTD

A DTD is referred to as an internal DTD if elements are declared within the XML files. To refer it as internal DTD, standalone attribute in XML declaration must be set to **yes**. This means, the declaration works independent of an external source.

Syntax

Following is the syntax of internal DTD:

```
<!DOCTYPE root-element [element-declarations]>
```

where *root-element* is the name of root element and *element-declarations* is where you declare the elements.

Example

Following is a simple example of internal DTD:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE address [
   <!ELEMENT address (name,company,phone)>
   <!ELEMENT name (#PCDATA)>
   <!ELEMENT company (#PCDATA)>
   <!ELEMENT phone (#PCDATA)>
]>
<address>
   <name>Tanmay Patil</name>
   <company>TutorialsPoint</company>
   <phone>(011) 123-4567</phone>
</address>
```

Let us go through the above code:

Start Declaration - Begin the XML declaration with the following statement.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
```

DTD - Immediately after the XML header, the *document type declaration* follows, commonly referred to as the DOCTYPE:

```
<!DOCTYPE address [
```

The DOCTYPE declaration has an exclamation mark (**!**) at the start of the element name. The DOCTYPE informs the parser that a DTD is associated with this XML document.

DTD Body - The DOCTYPE declaration is followed by the body of the DTD, where you declare elements, attributes, entities, and notations.

```
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone_no (#PCDATA)>
```

Several elements are declared here that make up the vocabulary of the <name> document. <!ELEMENT name (#PCDATA)> defines the element *name* to be of type "#PCDATA". Here, #PCDATA means parse-able text data.

End Declaration - Finally, the declaration section of the DTD is closed using a closing bracket and a closing angle bracket (**]>**). This effectively ends the definition, and thereafter, the XML document follows immediately.

Rules

- The document type declaration must appear at the start of the document (preceded only by the XML header) — it is not permitted anywhere else within the document.
- Similar to the DOCTYPE declaration, the element declarations must start with an exclamation mark.
- The Name in the document type declaration must match the element type of the root element.

External DTD

In external DTD elements are declared outside the XML file. They are accessed by specifying the system attributes which may be either the legal *.dtd* file or a valid URL. To refer it as external DTD, standalone attribute in the XML declaration must be set as **no**. This means, declaration includes information from the external source.

Syntax

Following is the syntax for external DTD:

```
<!DOCTYPE root-element SYSTEM "file-name">
```

where *file-name* is the file with *.dtd* extension.

Example

The following example shows external DTD usage:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE address SYSTEM "address.dtd">
<address>
  <name>Tanmay Patil</name>
  <company>TutorialsPoint</company>
  <phone>(011) 123-4567</phone>
</address>
```

The content of the DTD file **address.dtd** is as shown:

```
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
```

Types

You can refer to an external DTD by using either **system identifiers** or **public identifiers**.

System Identifiers

A system identifier enables you to specify the location of an external file containing DTD declarations. Syntax is as follows:

```
<!DOCTYPE name SYSTEM "address.dtd" [...]>
```

As you can see, it contains keyword SYSTEM and a URI reference pointing to the location of the document.

Public Identifiers

Public identifiers provide a mechanism to locate DTD resources and is written as follows:

```
<!DOCTYPE name PUBLIC "-//Beginning XML//DTD Address Example//EN">
```

As you can see, it begins with keyword PUBLIC, followed by a specialized identifier. Public identifiers are used to identify an entry in a catalog. Public identifiers can follow any format, however, a commonly used format is called **Formal Public Identifiers**, or FPIs.

// Element Parsed Character DATA PCDATA

<!ELEMENT element_name (#PC DATA)

<!ELEMENT Name (#PCDATA)

Ex: <Name> Umesh </Name>

DTD

1) Element
2) Attribute

Jaspreet Singh
9592 2324..

Attributes

<! ATTLIST element-name attribute name attribute-type attribute-value>

<! ATTLIST Employee Employee-Number CDATA #REQUIRED>

< Employee Employee-Number = "105" > (element, attribute)
< /Employee >

XML

< Employee >
< /Employee >

Wrong

- Attribute Type

CDATA ⇒ The value is character Data (any string)

ID ⇒ Unique.
PC DATA ⇒ Parsed Character Data.

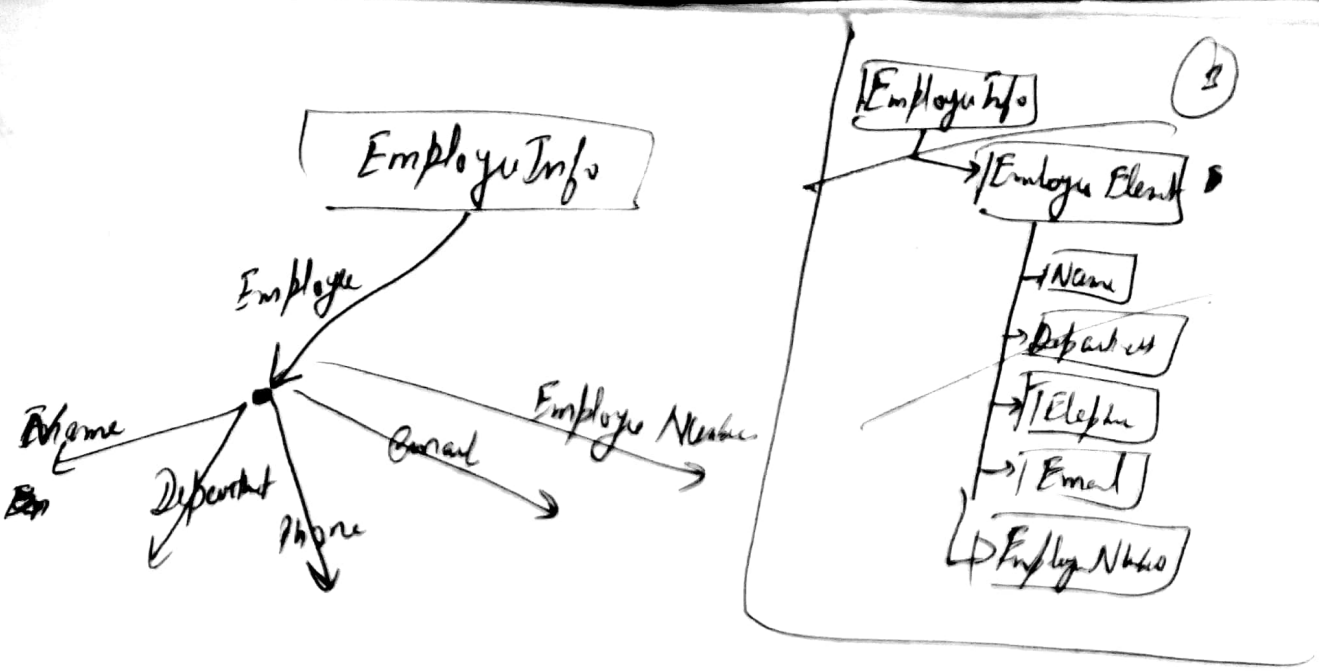
- Attribute-Value

(1) ~~Value~~ → default value of attribute
(2) # REQUIRED → The attribute is required
(3) # IMPLIED → The attribute is optional
(4) # FIXED Value → The attribute value is fixed

(1) <! ATTLIST Employee Employee Number CDATA # REQUIRED >
(2) <! ATTLIST Employee Employ-Number CDATA # IMPLIED >

Implies (optional)
< Employee Employees Number = "05" > | < Employee > ✓
< /Employee > ✓ | < /Employee >

(3) <! ATTLIST Employee Employ-Number CDATA #FIXED "105" >



EmployeInfo Element : Doesnot include text data.

→ EmployeElement Element : 0 or more times : Doesnot include text data.

→ Name Element : Once : Text Data

→ Department Element : Once : Text Data

→ Phone Element : Once : Text Data

→ EMail Element : Once : Text Data

→ EmployeNumber Attribute : Integer Data : Required

External DTD

<! XML version = "1.0" encoding = "UTF-8" Standalone = "no" ?>

<! DOCTYPE employee-Info SYSTEM "employee-Info.dtd" >

Note: XML FILE

<Employee-Info>

</Employee-Info>

	Challenges, Characteristics and Tools	Computational Intelligence in Data Science (ICCIDS 2017) (IEEE INDEXING)	Communicated

(4)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE Employee-Info [
  <!ELEMENT Employee-Info (Employee)+>
  <!ELEMENT Employee (Name, Department, Phone, Email)>
  <!ELEMENT NAME (#PCDATA)>
  <!ELEMENT DEPARTMENT (#PCDATA)>
  <!ELEMENT PHONE (#PCDATA)>
  <!ELEMENT Email (#PCDATA)>
<!ATTLIST EMPLOYEE. EMPLOYEE_NUMBER CDATA #REQUIRED>
]>
```

```
<Employee_Info>
<Employee Employee_Number="105">
<Name> Umesh </name>
<Department> CSE </Department>
<Phone> 9592000001 </Phone>
<email> jps@gmail.com </email>
</Employee>
</Employee_Info>
```

Element Occurrences

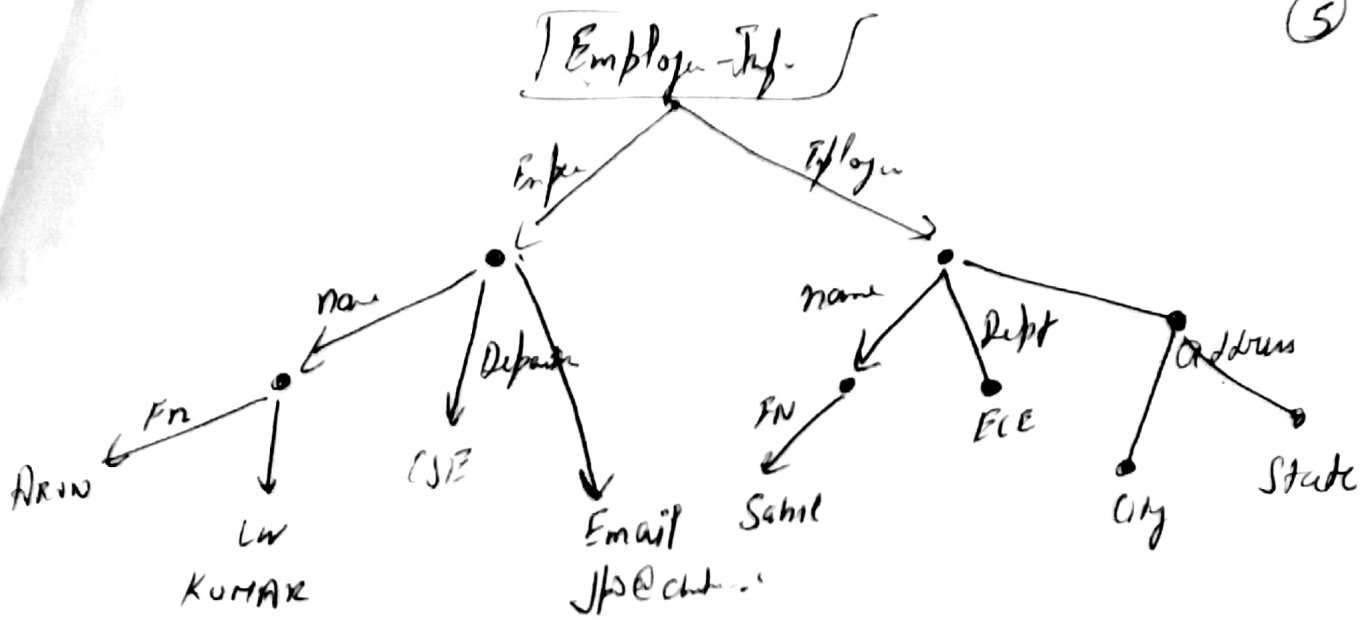
(i) + 1 or more Minimum one occurrence of Element.
<! Element Employ-details (Employee)+ >

(ii) * 0 or more Zero or more

(iii) ? 0 or 1 occurrence

(iv) <! Element Employee-Info (Employee) >

Only one occurrence



<!DOCTYPE Employee-Info [

<! ELEMENT Employee-Info (Employee)+>

<! ELEMENT EMPLOYEE (Name, Department, Email*, address*)

<! ELEMENT Name (FN, LN*)>

<! ELEMENT FN (#PCDATA)>

<! ELEMENT LN (#PCDATA)>

<! ELEMENT Department (#PCDATA)>

<! ELEMENT EMAIL (#PDATA)>

<! ELEMENT Address (City, State)>

<! ELEMENT City (#PCDATA)>

<! ELEMENT State (#PCDATA)>

]>

XML → DTD

XML

```
<employe id = "2" name = "Rahul" >
  ABC
  <age> 22 </age>
  <gender> Male </gender>
  <salary> 20,000 </salary>
</employe>
```

[id unique
name - optional]

DTD

```
<?xml version = "1.0" encoding = "UTF-8" ?>
<!DOCTYPE employe [
<!ELEMENT employe (#PCDATA, age, gender, salary) >
<!ELEMENT age (#PCDATA) >
<!ELEMENT gender (#PCDATA) >
<!ELEMENT salary (#PCDATA) >
<!ATTLIST employe id ID #REQUIRED >
<!ATTLIST employe name CDATA #IMPLIED >
]>
```

```
<! Element Workers ( Worker* )>
<! Element Worker ( SSN, LastName?, FirstName?,
   hours ) >
<! ELEMENT SSN (#PCDATA)>
<! Element LastName (#PCDATA )>
<! Element FirstName (#PCDATA)>
<! Element hours (#PCDATA)>
]>
```

XML DTD Notations

i) A * following the element name means that the element can be repeated zero or more times in the document. This can be called an optional multivalued (repeating) element.

ii) A + following the element name means that the element can be ~~repeated~~ repeated one or more times in the document. This can be called a required multivalued (repeating element)

iii) A ? following the element name means that the element can be repeated zero or one times. This can be called

Experiment Number

an optional single valued (non-repeating) element.

iv) An element appearing without any of the preceding three symbols must appear exactly once in the document. This can be called required single-valued (non-repeating) element.

v) The type of element is specified via parenthesis following the element.

a) If the parenthesis include names of other elements, these would be the children of the element in the tree structure.

b) If the parenthesis include the keyword #PCDATA or one of the other types available in XML DTD, the element is a leaf node. ~~#PCDATA~~ PCDATA stands for parsed character data, which is roughly similar to a string data type.

vi) Parenthesis can be nested when specifying elements.

vii) A bar symbol (e1 | e2) specifies that either e1 or e2 can appear in the document.

Teacher's Signature

Limitations of ~~DTD~~ XML DTD

i) The data types in DTD are not very general.

ii) DTD has its own special syntax & so it requires specialised processors.

⇒ It would be advantageous to specify XML schema documents using the syntax rules of XML itself so that the same processors for XML documents can process XML schema descriptions.

iii) All DTD elements are always forced to follow the specified ordering the document so unordered elements are not permitted.

Advantages

i) XML DTD is used to describe XML language precisely.

ii) To define structure of an XML document.

Ans]>

SQL	PL/SQL
i) one statement is executed at a time.	i) It is executed as a block of code.
ii) SQL tells the database what to do & not how to do it.	ii) Tells the database how to do things (Procedural)
iii) SQL is used to code queries, DML & DDL statements.	iii) It is used to code program blocks, triggers, functions, procedures & packages.
iv) We can embed SQL in a PL/SQL program.	iv) We can't embed PL/SQL within a SQL statement.
v) SQL is a data oriented language for selecting & manipulating sets of data.	v) It is a procedural language to create applications.

Experiment Number

Date __/__/__

Page

XML Schema Definition (XSD)

The XML Schema language is a standard for specifying the structure of XML documents.

XML Schema is based on the tree data model with elements & attributes as the main structuring concepts. It borrows additional concepts from database such as keys, references & identifiers.

	DTD	XSD
i)	Doesn't support data types. It has only #PCDATA as the data type for the elements.	i) Supports data types. a) Primitive / fundamental data types: String, decimal, float, boolean. b) Custom data types I) Complex Type: A datatype that contains child elements or attributes & also the mixed contents. II) Simple Type: A datatype that contains only values.
ii)	It doesn't support namespaces. It has its own set of keywords for defining a Schema.	ii) It uses its own set of namespaces & elements for defining the Schema.

Teacher's Signature

e.g. ! DOCTYPE for root tag
! ELEMENT for an element

iii) There is no restriction of datatype validation.

iv) It is more suitable for small XML data.
e.g. bookname

v) doesn't define order for child elements.

vi) e.g. <! DOCTYPE Address[
<! ELEMENT Address (Name)
<! ELEMENT Name (#PCDATA)
]>

iii) It allows us to specify restriction on data.
e.g. → <price></price> tag.
We can write only digits here

iv) It is used in large XML data.
e.g. → Web Services

v) order is defined.

e.g. <xsd: element name="Address">
<xsd: ComplexType>
<xsd: sequence>
<xsd: element name="name" type=
"xsd: string">
</xsd: sequence>
</xsd: ComplexType>
</xsd: element>

Schema

1. Scheme description and XML Name Spaces
2. Annotations, documentation, language used (optional)
3. Elements and Types → (root element structure) ex = complex
4. Specify first level elements of root element
   - ex dept, emp, project
5. Specify elementtype and minimum and maximum occurrences.
   - (1) element type =
   - (2) Minimum = Null → 0
   - (3) Max = multivalues → unbounded. (Similar to ? + of DTD)

Ex `<xsd: element name = "employee" type = "employee" minOccurs = "0" maxOccurs = "Unbounded" />`

6. Specify Keys (constraints)
   - (1) Unique Key (xsd: unique)
   - (2) Primary Key (xsd: key)
   - (3) Foreign key (xsd: keyref)

```
<xsd: key name = "deptNumKey">
<xsd: selector xpath = "department" />
<xsd: field xpath = "DepartmentName" />
</xsd: key>
<xsd: "emp DeptNumber KeyRef" refer = "DeptNumKey" />
```

7. Specify structure of complex elements via complex type

```
<xsd: complexType name "Department" >
<xsd: Sequence >
<xsd: element name = "DepartmentNum" type = "xsd: string" />
</xsd: sequence >
</xsd: complexType >
```

8. Composite attributes:

Name → FN, LN

(1) <? xml version = "1.0" encoding = "UTF-8" ?>
<xsd : Schema xmlns : xsd = "http://www.w3.org/2001/XMLSchema">

end of doc tag

</xsd : schema>

(2) tag
<xsd : element name = "Company">
<xsd : complexType>
<xsd : Sequence>
(4) (5)
<xsd : element name = "depatmt" type = "Department" minOccurs = "0"
maxOccurs = "unbounded" />

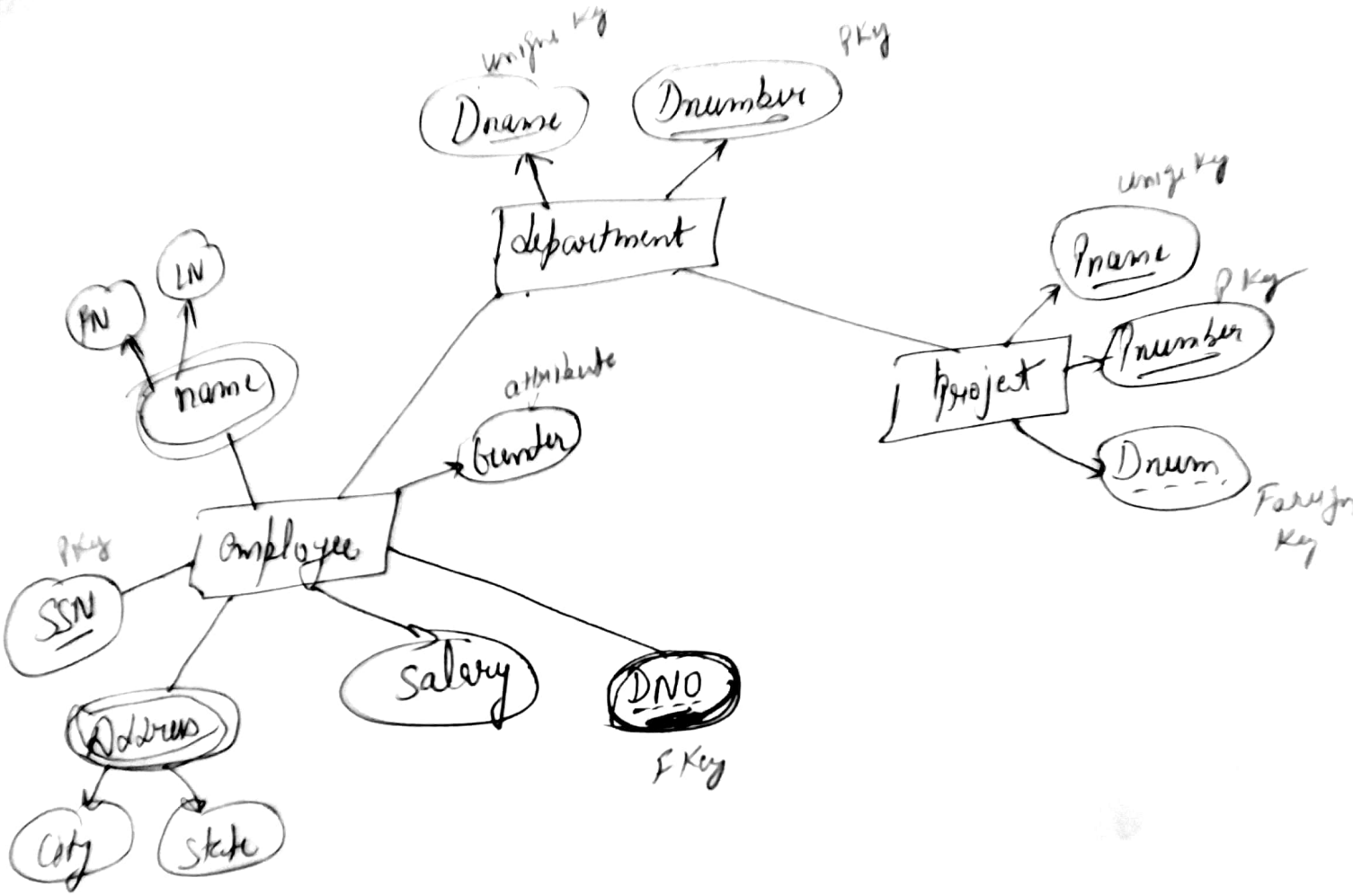
</xsd : sequence>
</xsd : complexType>

(6)
↓
Keyless

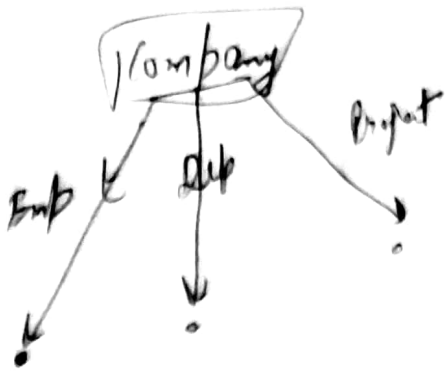
</xsd : element>
tag

(7)
(8)

Company Database



Ex



XML Schema File Company

```
<?xml version = "1.0" encoding = "UTF-8" ?>
<xml: schema xmlns: xsd = "http://www.w3.org/2001/XMLSchema">

<xsd: element name = "company">
   <xsd: complexType>
      <xsd: sequence>
         <xsd: element name = "department" type = "Department"
                  minOccurs = "0" maxOccurs = "unbounded" />
         <xsd: element name = "employee" type = "Employee"
                  minOccurs = "0" maxOccurs = "unbounded" />
         <xsd: element name = "project" type = "Project"
                  minOccurs = "0" maxOccurs = "unbounded" />
      </xsd: sequence>
   </xsd: complexType>

   <xsd: unique name = "departmentNameUnique">
      <xsd: selector xpath = "department" />
      <xsd: field xpath = "Dname" />
   </xsd: unique>
   <xsd: unique name = "projectNameUnique">
      <xsd: selector xpath = "project" />
      <xsd: field xpath = "Pname" />
   </xsd: unique>

   <xsd: key name = "departmentNumberKey">
      <xsd: selector xpath = "department" />
      <xsd: field xpath = "Dnumber" />
   </xsd: key>
   <xsd: key name = "employeeNumberKey">
      <xsd: selector xpath = "employee" />
      <xsd: field xpath = "SSN" />
   </xsd: key>
```

```
<xsd:key name = "projectNumberKey" >
    <xsd: selector xpath = "project" />
    <xsd: field    xpath = "Pnumber" />
</xsd:key>

<xsd:keyref name = "employeeDepartmentNumberKeyRef" refer = "departmentNumberKey" />
    <xsd: selector xpath = "employee" />
    <xsd: field    xpath = "Dno" />
</xsd:keyref>

<xsd:keyref name = "projectDepartmentNumberKeyRef" refer = "departmentNumberKey" />
    <xsd:selector xpath = "project" />
    <xsd: field   xpath = "Dnum" />
</xsd:keyref>

<xsd: keyref name "WorksOnEmployeNumberKey" refer = "EmployeNumberKey" />
    <xsd: selector xpath = "
    <xsd: field xpath = "
</xsd: element>
```

01

```
<xsd:complexType name = "Department" >
    <xsd:sequence>
        <xsd: element name = "Dname" type = "xsd:String" />
        <xsd: element name = "Dnumber" type = "xsd: string" />
    </xsd: sequence>
</xsd:complexType>

<xsd: complexType name = "Employee" >
    <xsd: sequence>
        <xsd: element name = "Name" type = "name" />
        <xsd: element name = "SSN"  type = "xsd:string" />
```

02

```
<xsd:element name="Salary" type="xsd:string" />
<xsd:element name="Address" type="address" />
<xsd:element name="Dno" type="xsd:string" />
<xsd:attribute name="Gender" type="xsd:string" />

</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Project">
    <xsd:sequence>
        <xsd:element name="Pname" type="xsd:string" />
        <xsd:element name="Pnumber" type="xsd:string" />
        <xsd:element name="Dnum" type="xsd:string" />
    </xsd:sequence>
</xsd:complexType>
```

SLT Link

(or)

```
<xsd:complexType name="Name">
    <xsd:sequence>
        <xsd:element name="FN" type="xsd:string" />
        <xsd:element name="LN" type="xsd:string" />
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Address">
    <xsd:sequence>
        <xsd:element name="City" type="xsd:string" />
        <xsd:element name="State" type="xsd:string" />
    </xsd:sequence>
</xsd:complexType>
```

Further complex elements

XML Languages

There are several XML query languages
Two standard query languages are:

① XPath → (to locate element)

② XQuery → (XPath + full features DL)

XSLT
XLink
X

I XPath:

- XPath provides language constructs for specifying path expressions to identify certain nodes (elements) or attributes within an XML document.

Two main separators are used when specifying path:

① Single slash (/)

② Double slash (//)

① Single slash (/): A single slash before a tag specifies that the tag must appear as a direct child of previous (parent) tag.

② Double slash (//): // specifies that the tag can appear as a decendent of previous tag at any level.

Attribute

@

To specify attribute

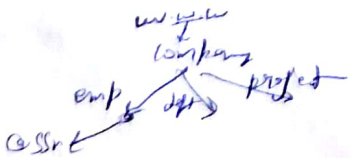
@ ISN

Wild card

* any element

For example, if a company XML document is stored at the location

- www.company.com/info.XML



XPath expressions

① doc(www.company.com/info.XML)/company

② doc(www.company.com/info.XML)/company/department

③ doc(www.company.com/info.XML)//employee

④ doc(www.company.com/info.XML)//employee[salary gt 70000]/employeeName

⑤ doc(www.company.com/info.XML)//company/employee[salary gt 70000]/Name

wildcard

⑥ doc(www.company.com/info.XML)/company/employee[salary gt 70000]/*

attribute

⑦ doc(www.company.com/info.XML)/company/employee/[salary gt 70000]/@SSN

Restriction/Limitation of XPath.

(Inequality) (pattern) (multiple conditions *)

- Path expression is that which specify the path of the item to retrieves.
- Here in XPath path expression specifies the pattern also specify the item to be retrieves.
- Hence, it is difficult to specify certain conditions on the pattern.
- If we need result efficiently.

The XQuery language separates these two concerns, and provides more powerful constructs for specifying queries.

- XQuery : Specify Queries in XML

The typical form of query in XQuery known as FLWR expression. Four main clauses of XQuery.

FOR LET WHERE RETURN.
M/F O/P

1 FOR $x IN doc(www.company.com/info.xml)/company/employee
[employee. Salary gt 70000]/ Name
RETURN <res> $x/FN , $x/LN </res>

2 FOR $x IN //employee
doc(www.company.com/info.xml)/company/employee
WHERE $x/Salary gt 70000
RETURN <res> $x/Name/FN, $x/Name/LN </res>

3 LET $d := doc(www.company.com/info.xml)
FOR $x IN $d/company/employee
WHERE $x/Salary gt 70000
RETURN <res> $/Name/FN, $x/Name/LN </res>

- LET $d := doc(www.company.com/info.xml)
  FOR $x IN $d/Company/Project[PNumber = 5]/ProjectWorker, $y IN $d/company/employee
  WHERE $x/hours gt 20.0 AND $y.SSN = $x.SSN
  RETURN <res> $y/Name/FN, $x/hours </res>

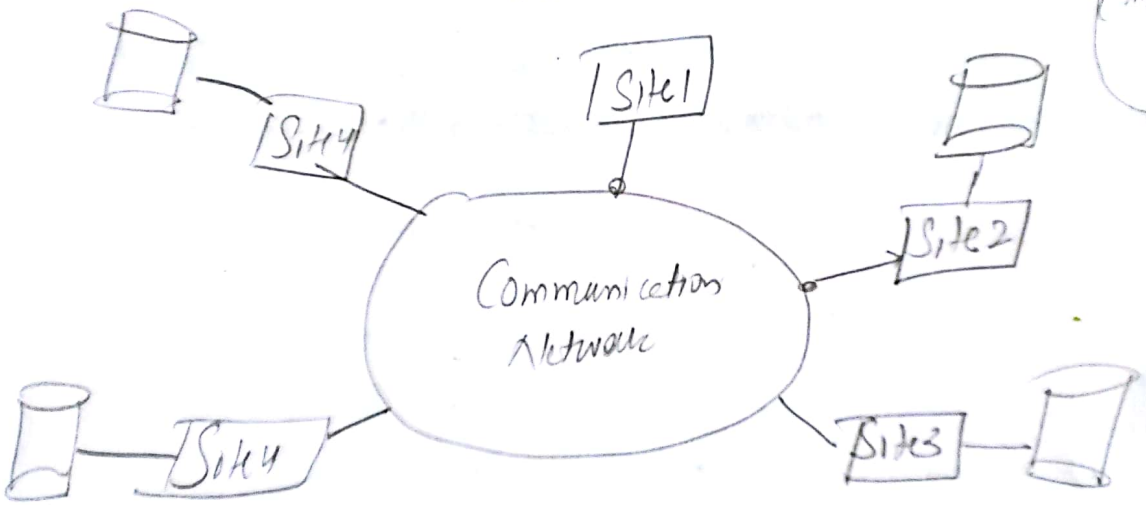
(1) DISTRIBUTED DATABASE

Distributed Database is a collection of multiple logically interrelated databases, distributed over a computer Network

relationships (Foreign key)

2 DISTRIBUTED DATABASE MANAGEMENT SYSTEM.

DDBMS is a software System that manages a distributed database while and making the distribution transparent to the user. makes



DDBS ENVIRONMENT

3 Conditions for Distributed Database

For a database to be called distributed the following minimum conditions should be satisfied.

1) Connection of Database nodes over a Computer Network.

. There are multiple computers known as sites or nodes. These sites must be connected by a communication network to transmit

• Promises / Features of DDBMS

(1) Transparency (2) Autonomy (3) Reliability & Availability.

a) Transparency

It refers to the idea of hiding implementation details from end users. The advantage of fully transparent DBMS is the high level of support that it provides for the development of complex applications.

Types of Transparency

(1) Data Organisation Transparency.
(2) Replication Transparency
(3) Fragmentation Transparency.
(4)

(1) Data Organisation Transparency: (also known as distribution or n/w transparency)
This refers to freedom for the user from the operational details of the network & placement of the data in the distributed system. It is further divided into location & naming transparency.

1.1) Location Transparency

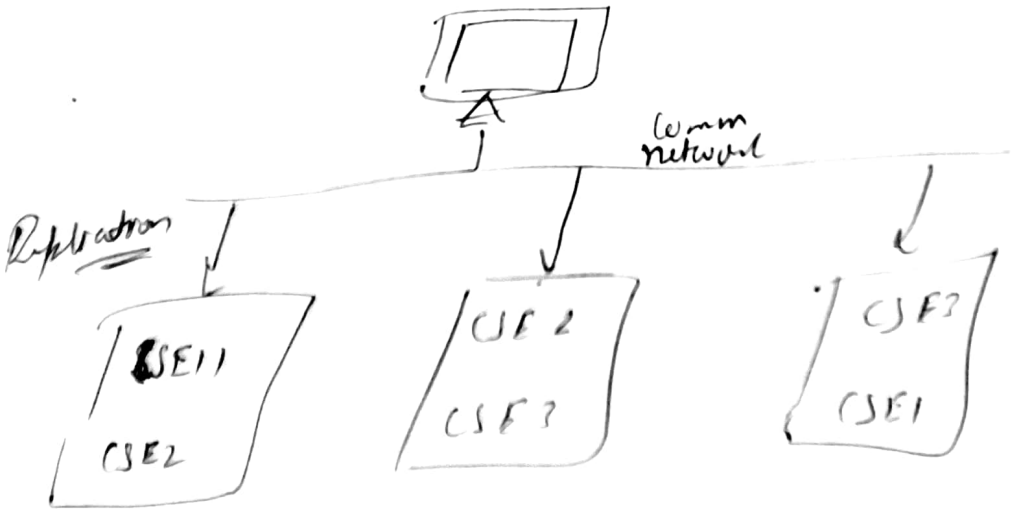
It refers to the fact that the command used to perform a task is independent of the location of data & the location of the nodes & the system on which an operation is carried out.

1.] Naming Transparency:

It means that a unique name is provided for each object in the database. In the absence of naming transparency, users are required to embed the location name as part of the object.
is embedded as a part of object name

2] Replication Transparency:

The copies of the same data objects may be stored at multiple sites for better availability, performance and reliability. Replication transparency makes the user unaware of the ~~existing~~ existance of these copies.



3] Fragmentation Transparency:

Two types of fragmentation are possible. Horizontal & Vertical

- Horizontal Fragmentation distributes a relation (table) into sub relations that are subsets of the tuples (rows) in the original relations.

Vertical Fragmentation:
distributes a relation into subrelations where each subrelation is defined by a subset of the columns of the original relation.

Fragmentation transparency makes the user ~~unknown~~ unaware of the existance of fragments.

4) Design Transparency & Execution Transparency:
Users have no idea how the distributed database is designed and how a transaction executes.

b) Autonomy
It determines the extent to which individual node or database in a ~~connected~~ DDB can operate independently.
A high degree of autonomy is desirable for increased flexibility customised maintainance of an individual node.
Autonomy can be applied to design, communication & execution.

1) Design Autonomy: Refers to independence of data model usage & transaction management techniques among nodes

2) Communication Autonomy: It determines the extent to which each node can decide on sharing of information with other nodes.

3) Execution Autonomy: It refers to the independence user to act as they please.

3] Availability & Reliability
(Trustworthy)

Reliability is defined as the probability that a system is running (not down) at a certain time point.
(performing)

Availability is a probability that the system is continuously available during a time interval.

Reliability and Availability are related to database failures, faults & errors.

Advantages of Distributed database

(1) Improved ease & flexibility of Applications development
Developing and maintaining applications at geographically distributed sites of an organisation is facilated owing to transparency of data distribution & control.

(2) Increased Reliability & availability:
This is achieved by the isolation of faults to their site of origin without affecting the other databases connected to the network.
When the data & DDBMS software are distributed over several sites, one site may fail while other sites continue to operate.
Only the data and software that exist at the failed site can not be accessed. This improves both reliability & availability.
Replication → User can recover all the data of failed site from other sites because, In distributed database data is replicated on multiple nodes.
- In Central ×
- In Parallel ×

(3) Improved Performance:
A distributed DBMS fragments the database by keeping the data closer to where it is needed most. Data localization reduces the contention for CPU & I/O services.

4) Easier Expansion: (Scalability)
In distributed environment expansion of the system in term of adding more data increasing database sizes, or adding more processors is much easier. It is the most important feature of distributed database system. which is also known as scalability.

Functions of Distributed Database

imp

1) Keeping track of data distribution
The ability to keep track of the data distribution, fragmentation, & replication by expanding the DDBMS catalog.

2) Distributed Query Processing:
The ability to access remote sites and transmit queries & data among the various sites via a communication network.

3) Distributed Transaction Management: The ability to devise execution strategies for queries and transactions that access data from more than one site and to synchronize the access to distributed data and maintain the integrity of the overall database.

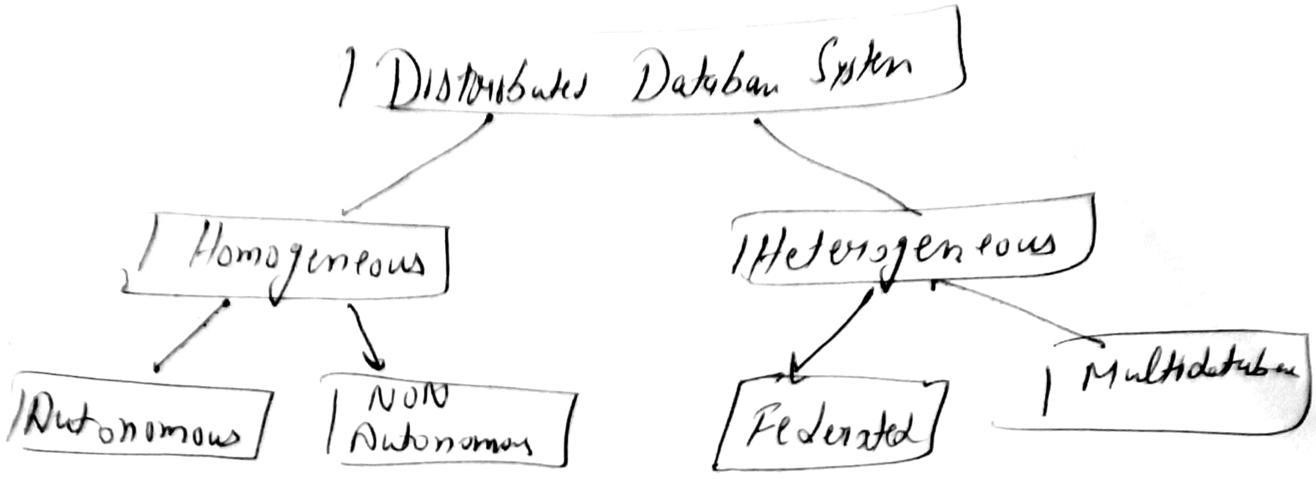
4 Replicated Data management :
The ability to decide which copy of a replicated data item to access and to maintain the consistency of copies of a replicated data item.
[means har site te same copy
nahi chahiye hai]

5 Distributed Database Recovery :
The ability to recover from individual site crashes and from new type of failures.
various

6 Security :
Distributed transaction must be executed with the proper management of the security of the data and the authorization/access privileges of users.

7 Distributed directory (catalog) management :
A directory contain information (metadata) about the data in the database. The directory may be global for the entire DDB or local for each site.

Types of Distributed Database System



(1) Homogeneous Distributed System:

i) All the servers and client use identical software and operating system.

ii) Each site is aware of all other sites & cooperates with other sites to process user requests.

iii) The database is accessed through a single interface as if it is a single database
i.e it appears to user as a single system.

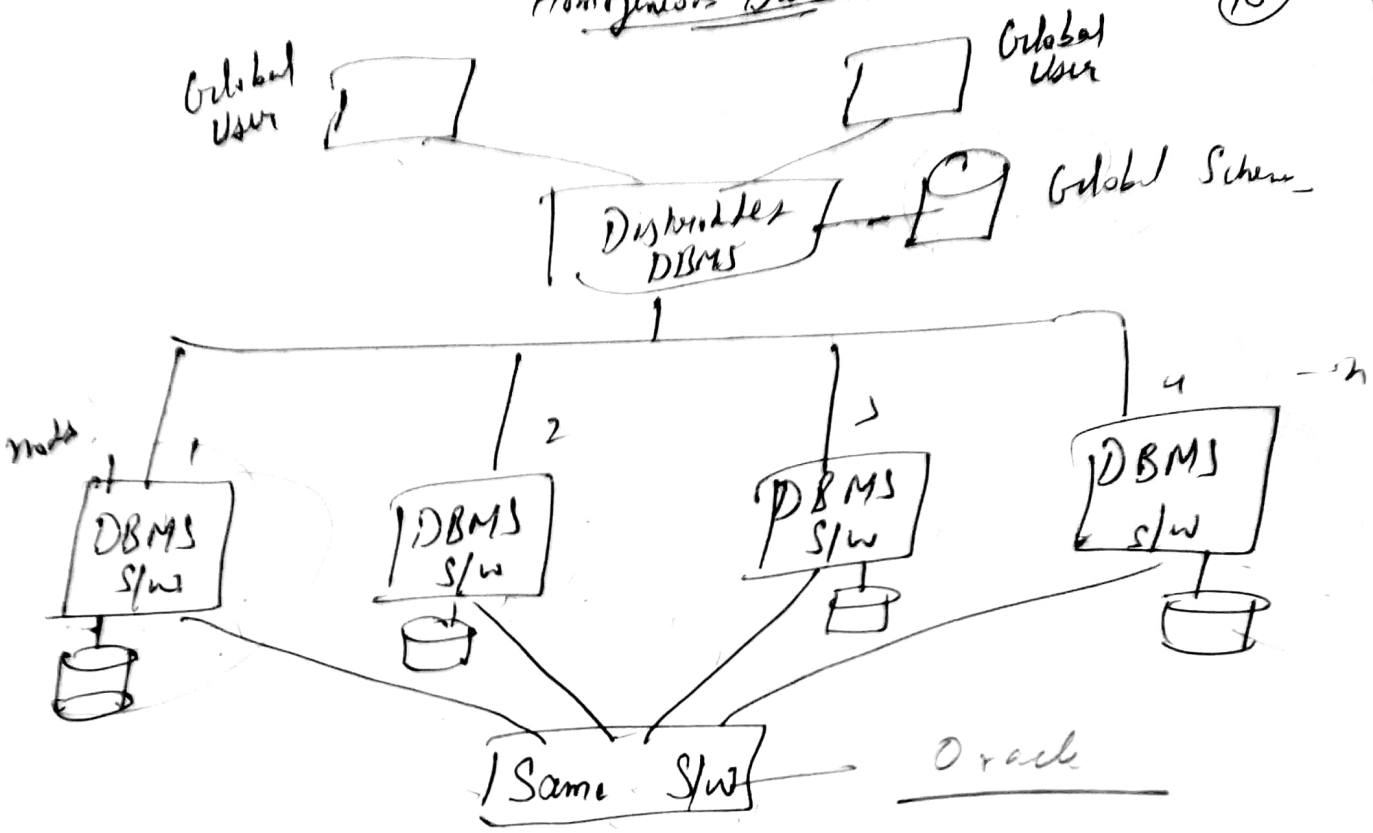
iv) It is easy to design & manage.

v) Difficult for most organisations to force a homogeneous environment

(Har company different configuration ya OS use karti hai so agar bandation ni lga sakte k ik hi use karna) So ek difficult hai apply karna har company ke

Scanned by CamScanner

Homogenious Database



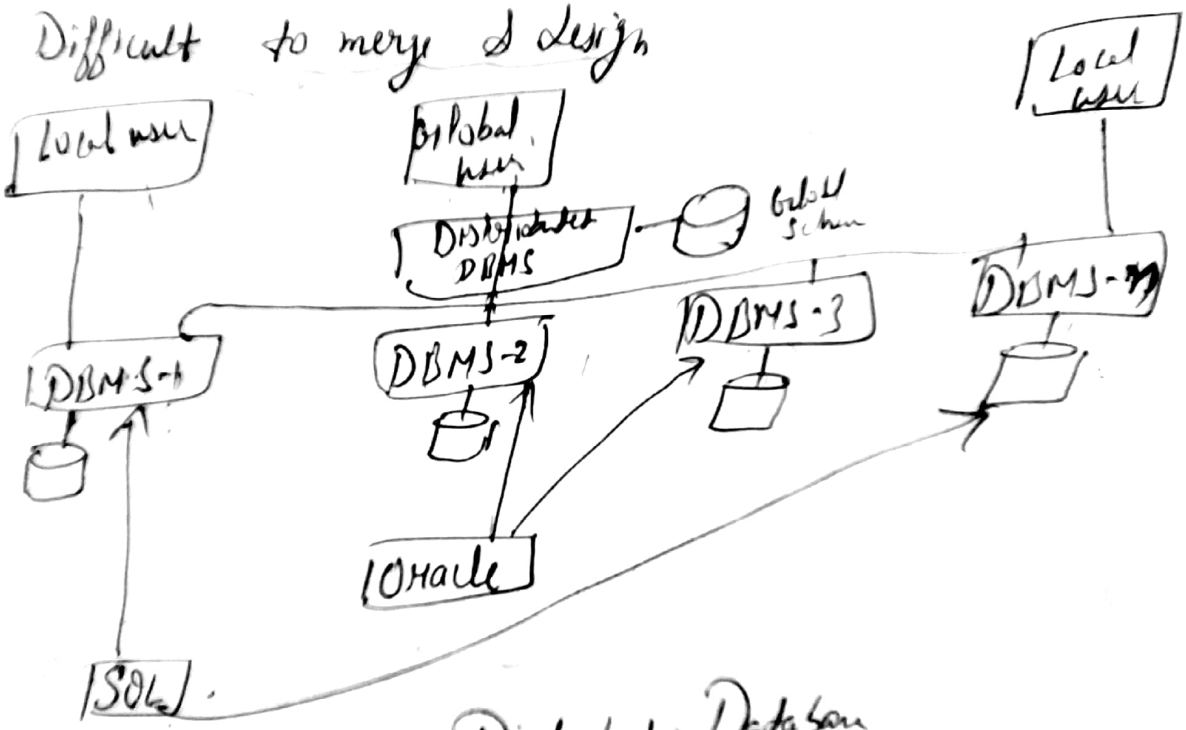
Types of Homogenious & distributed database.

1 Autonomous : (Identical)
Each database is independent that functions on its own. They are integrated by a controlling application & use message passing to share data updates.

2 Non-Autonomous : Data is distributed across the homogenious nodes and a master DBMS co-ordinates. data updates across the sites.

Heterogeneous Distributed Database

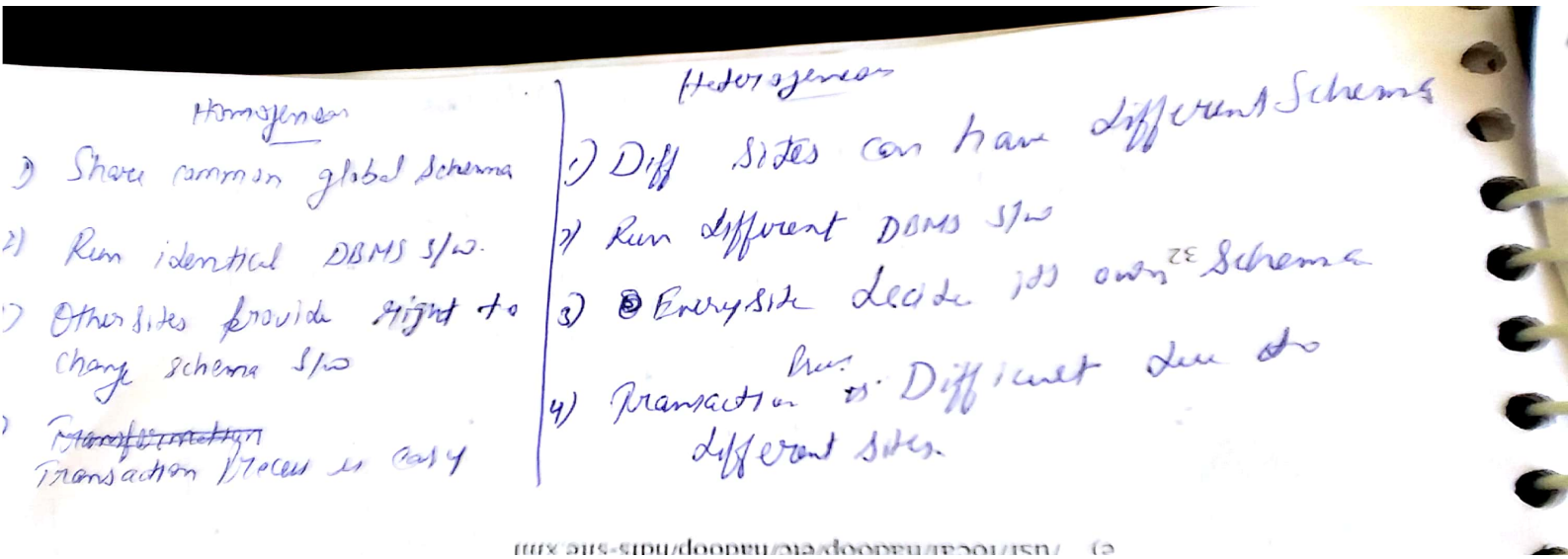
1) Different sites have different operating systems, data models and DBMS S/W.
2) Query processing and transaction processing is complex due to dissimilar ~~Site~~ S/W.
3) A site may not be aware of other sites & so there is limited co-operation in processing user requests
4) Difficult to merge & design



Types of Heterogeneous Distributed Database

1) **Federated:**
The Heterogeneous database systems are independent in nature & integrated together so that they function as a single database system. Global schema or view is shared by the applications.

2) **Multidatabase System:**
There is no global schema, but it constructs one (may be present) as needed by the applications.



difference very imp types bus likh lo baki types ko explain karne ki jaroort nhi sidha age diagram bana do

Parallel DBMS

1) Machines are physically close to each other, Ex same server room.

2) Machines are connected with high-speed LAN's & switches.

3) Communication cost is small

4) Follow shared memory, shared-disk or shared nothing architecture

5) It is a software system where multiple processors are used to execute & run queries in parallel.

6) ~~Execution speed is fast~~

7) Nodes should be homogenious.

8) Backup at on site only.

9) Diagram: Shared Disk, Shared memo, Shared nothing.

10) Replication feature not exist.

Distributed DBMS

1) Machines can be far from each other ex: in different continent.

2) Machines can be connected using public network ex: Internet.

3) Communication cost is high.

4) Follows shares nothing architecture.

5) It is a software system that manages multiple logically interrelated databases distributed over a communication network.

6) Nodes may be homogenous or heterogenous

7) Backup at multiple sites.

8) Diagram: Distributed Database.

9) Replication feature exists. (when a node fails with the help of Replication feature we can recover data from any node)

one site

difference very imp

Architectural Models of DDBMS

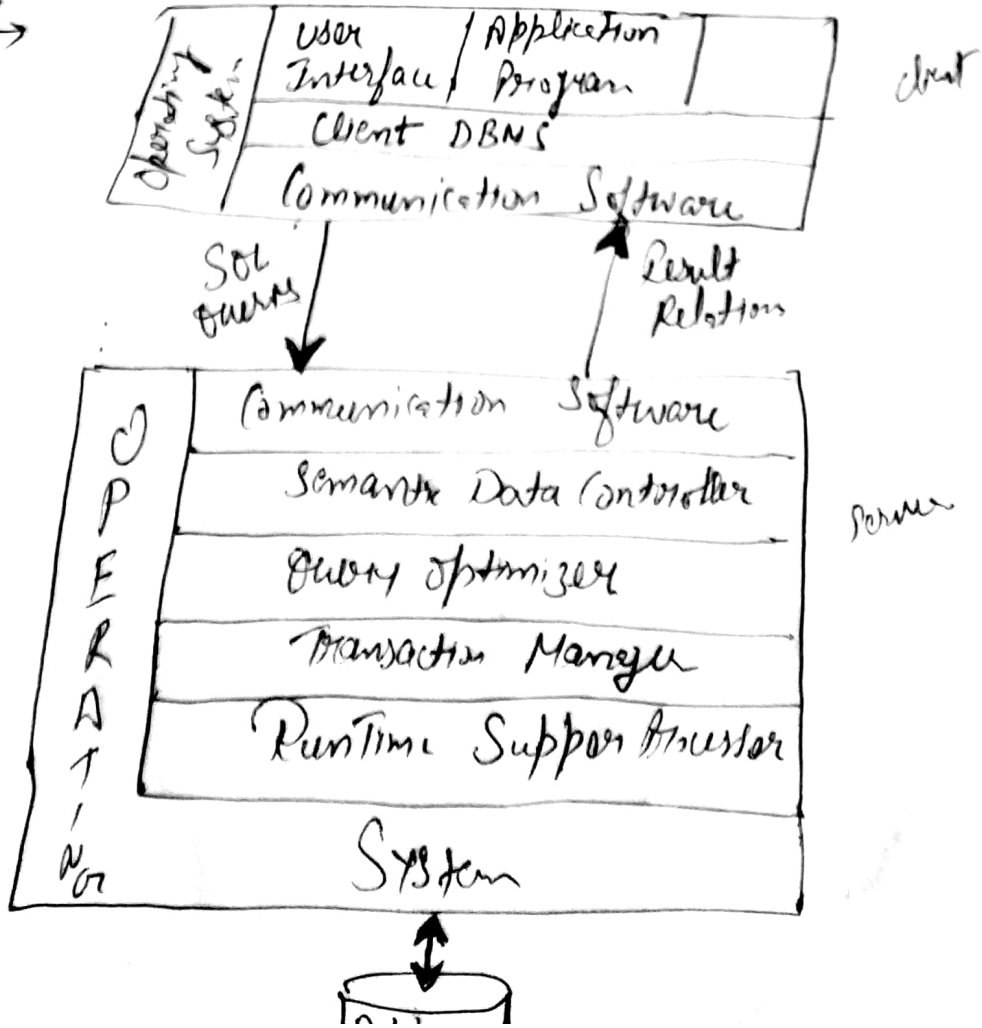
1) Client-Server Architecture.
2) Peer-to-Peer Architecture.
3) Multi DBMS Architecture.

1 Client-Server Architecture

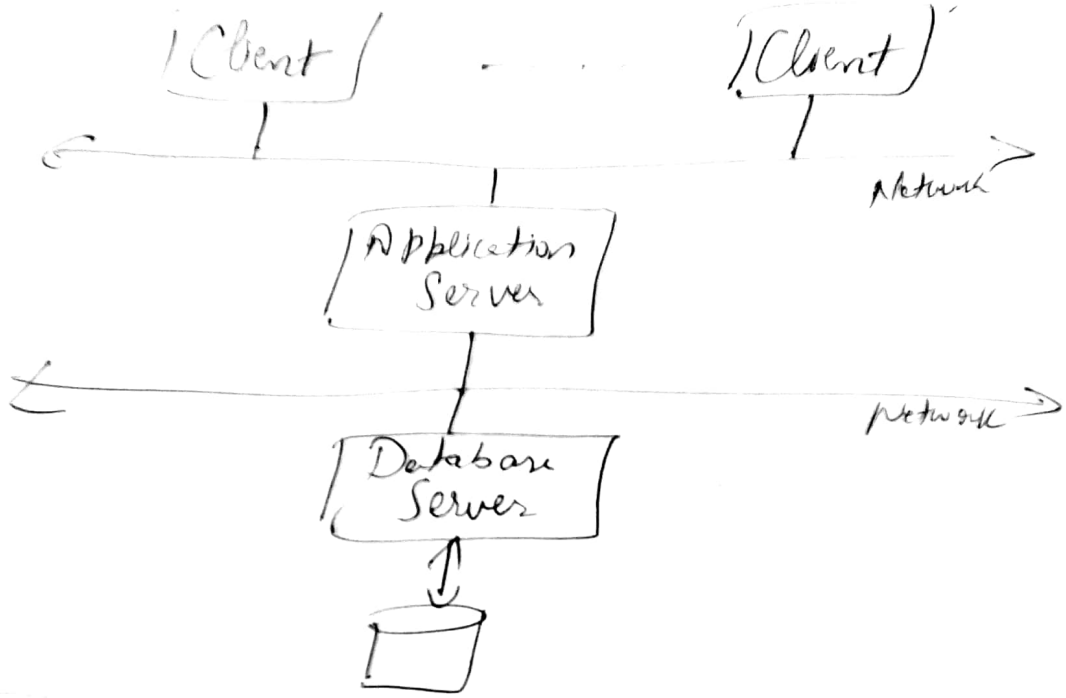
It is a 2 level architecture. where functionality is divided into servers & clients.

- **Server** does most of the data management work. This means that all the query processing & optimization, transaction management & storage management is done at server ~~level~~ end.
- **Client** function include applications user interface

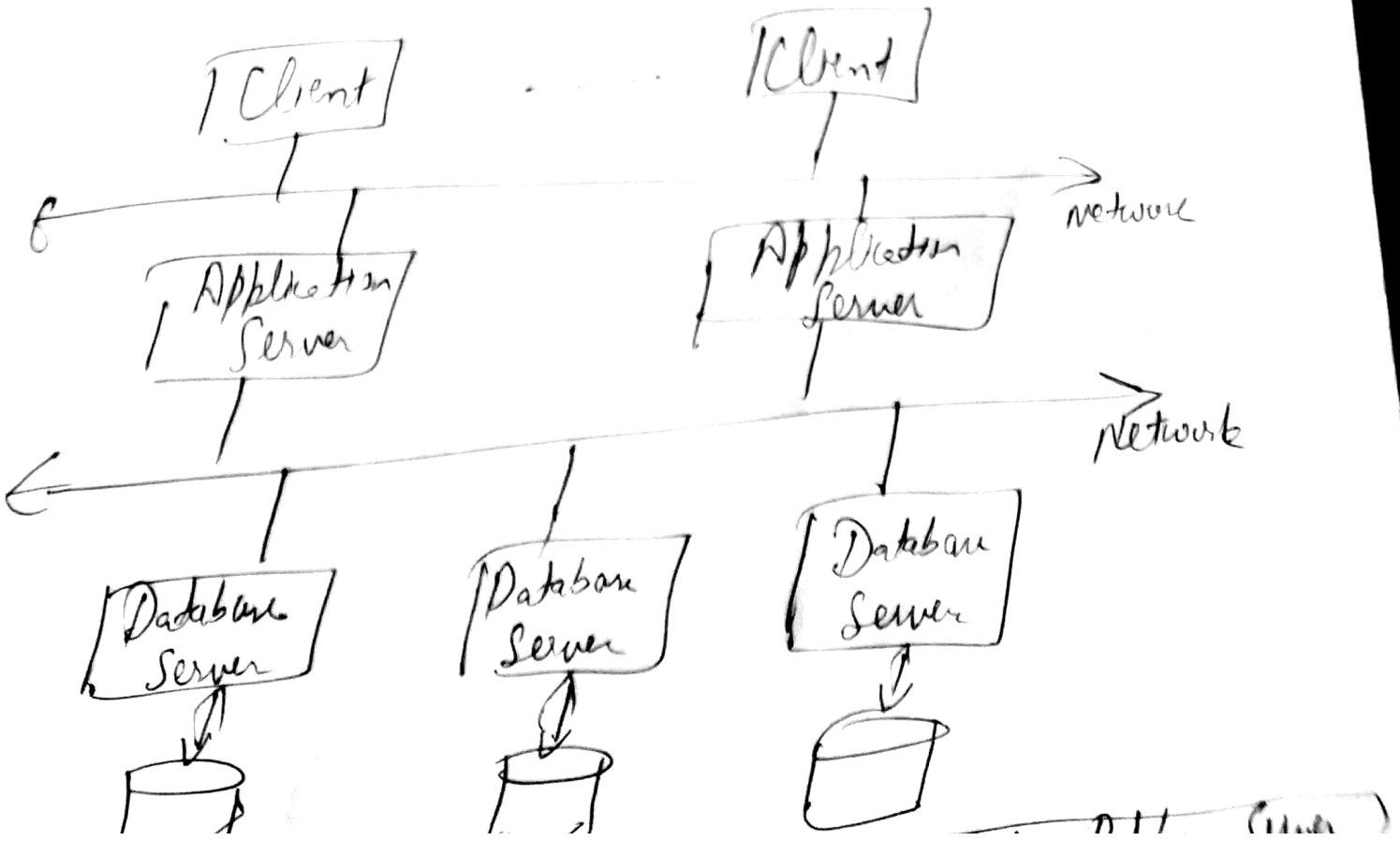
Client/Server Architecture →



① Single Server / Multiple Clients



② Multiple Server / Multiple clients

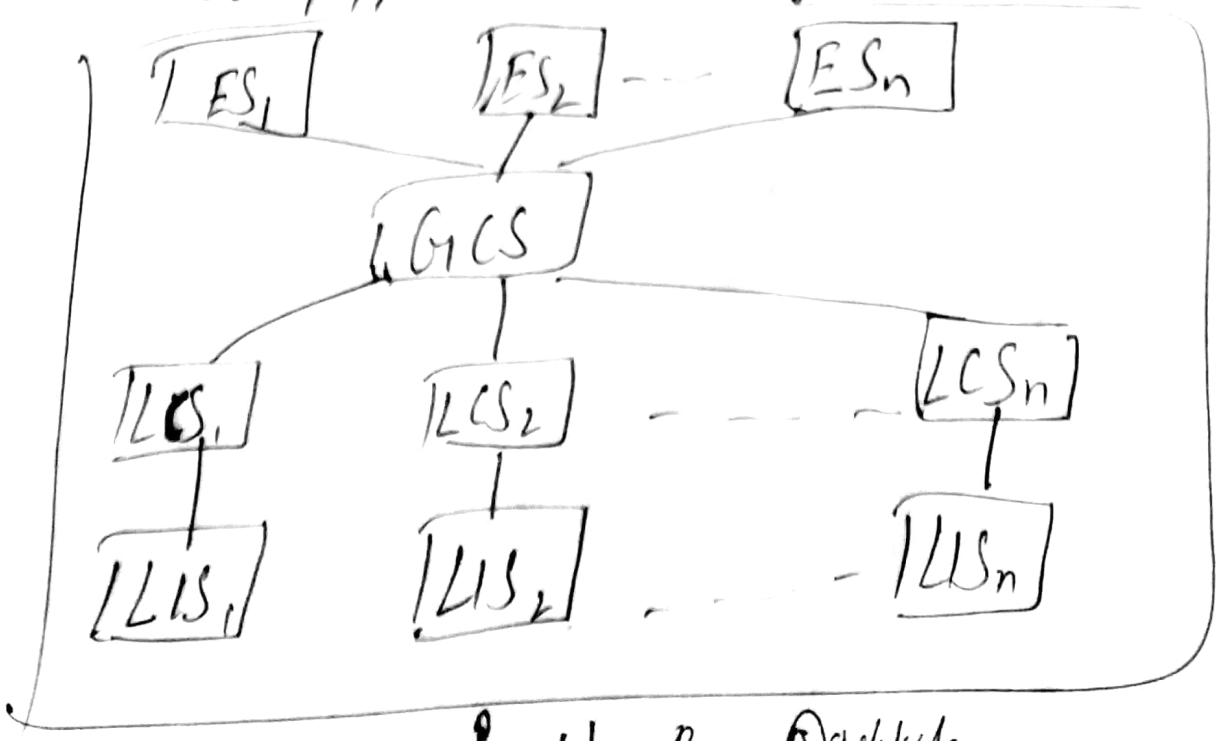


⑦ Peer-to-Peer System

Every system acts a full DBMS & can communicate with others systems both as client & server to execute queries & transactions.

The peer-to-peer architecture has 4 levels of schema:

1) Global Conceptual Schema (GCS):
   - global logical view of data
   - Union of logical conceptual schemas (LCS)
2) LCS Logical Conceptual Schema:
   - Describes logical data organisation at each site. Enterprise view of entire DBs.
3) Local internal Schema: LIS
   - Local Physical data organisation at each site.
4) External Schema: (ES)
   - Describes the user/application view on the data.



Peer-to-Peer Architecture

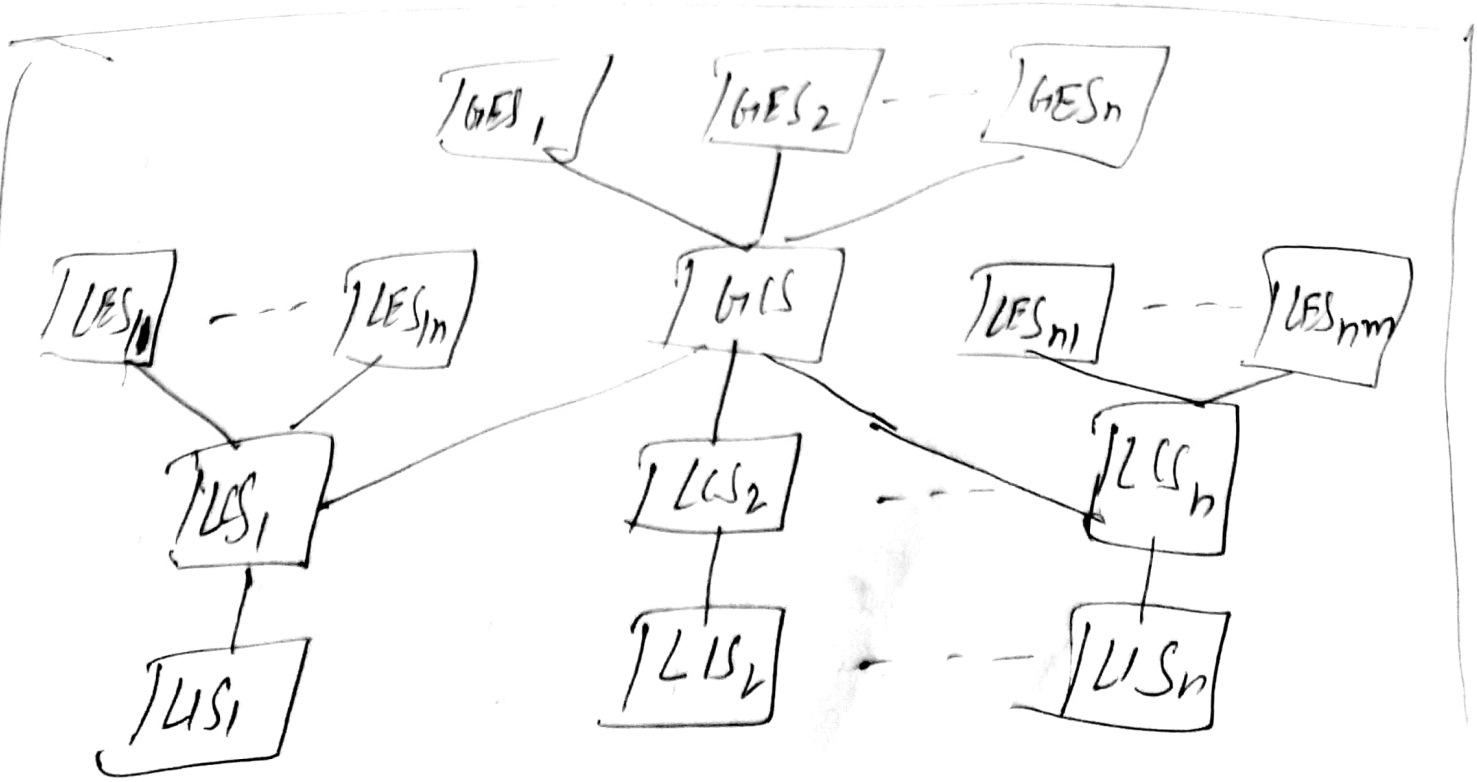
3) Multi - DBMS Architecture

Fundamental difference to peer-to-peer DBMS is the definition of the global conceptual schema (GCS).

- In multi-DBMS, the GCS represents only the collection of some of local databases that each local DBMS want to share.
- Data Distribution across different sites & multi DB to local data mapping.
- Two different architecture models:
  1) Models with a GCS.
  2) Models without GCS.

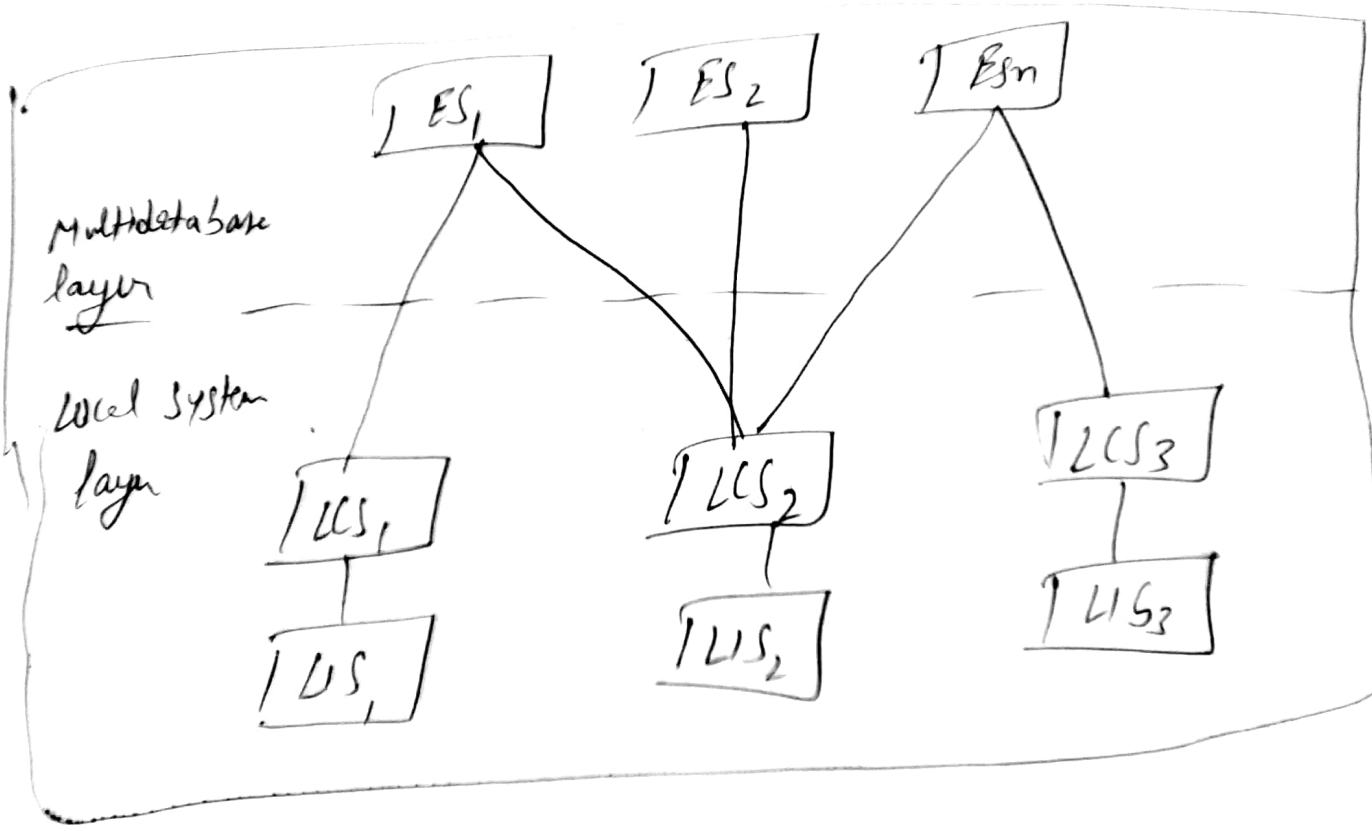
1) Model with a GCS

- GCS is a union of parts of the local DBMS - ~~defining~~
- Define their own views on the local DB



2. Model without GICS:

- The local DBMSs present to the multi-database layer the part of their local DB they are willing to share.
- External views are defined on top of LCSs.



Unit

⇒ Data Fragmentation

Types
1) Horizontal
2) Vertical
3) Mixed (Vertical + Horizontal) (Hybrid)

Fragment of a relation is an appropriate unit of distribution.

This means :-
1) Application views are subsets of relations.
2) Locality of access of application is defined on subset of relations.
3) Number of transactions are executed concurrently by a transaction manager.
Transaction manager maintains before and after database images, log of transactions & concurrency control
4. Parallel execution of single query.

5) ~~Semantic data control is more difficult~~.

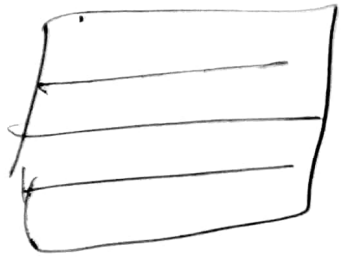
⇒ Fragmentation aims to improve :
1) Reliability
2) Performance
3) Balance storage capacity & costs.
4) Communication costs.
5) Security.

Types of Fragmentation

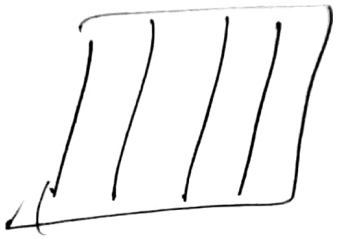
1) Horizontal Fragmentation:
Partitions a relation along its tuples (rows)

2) Vertical:
Partitions a relation along its attributes. (columns)

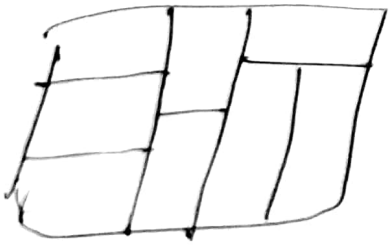
3) Mixed (Hybrid)
Combination of horizontal & vertical



Horizontal



Vertical



Mixed

Question PROJ

PNO	Pnam	Budget	Loc
P_1	Inst	150000	Montreal
P_2	OD	135000	New York
P_3	CAD/CAM	250000	New York
P_4	Maint	310000	Paris
P_5	CAD/CAM	500000	Boston

1) Horizontal Fragmentation :

1) PROJ1 : projects with budget less than 200000
2) PROJ2 : projects with budgets greater than or equal to 200000.

PROJ1 σ Budget < 200000 (PROJ)

PNO	PNam	Budget	Loc
P1	Inst	150000	Montreal
P2	DD	135000	New York

PROJ2 σ Budget >= 200000 (PROJ)

PNO	PName	Budget	LOC
P3	CAD/CAM	250000	New York
P4	Maint	310000	Paris
P5	CAD/CAM	500000	Boston

2) Vertical Fragmentation

1) PROJ1 : Information about project budgets.
2) PROJ2 : Information about project names & locations

PROJ1 π PNO, Budget (PROJ)

PNO	Budget
P1	150000
P2	135000
P3	250000
P4	310000
P5	500000

PROJ2 π PNO, PNAME, LOC (PROJ)

PNO	PName	LOC
P1	Inst	Montreal
P2	DD	New York
P3	CAD/CAM	New York
P4	Maint	Paris
P5	CAD/CAM	Boston

3) Hybrid Fragmentation

PROJ1

[PROJ1 :- Proj number, Project name & location with Budget > 200000]

Pno	Pname	Loc
P_3	CAD/CAN	New York
P_4	Maint	Paris
P_5	CAD/CAN	Boston

→ π pno, pname, Loc (σ budget > 200000)

π Pno, Pname, Loc (σ budget > 200000);

Most Imp ★★ 2 marks

ques aae ga kaa table given hoga te os ch fragmentation hogi te us table nu apa dasna

→ Correctness Rules of Fragmentation

1) Completeness:

Decomposition of relation R into fragments $R_1, R_2, \ldots R_n$ is complete if each data item in R can also be found in some R_i.

2) Reconstruction: If a relation R is decomposed into fragments $R_1, R_2, R_3 \ldots R_n$ then there should exist same relation From its fragments.

$R = R_1 \nabla \cdots \nabla R_n$

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

3) Disjointness: (Data item in one fragment should not placed in other frag)

If a relation R is decomposed into fragments $R_1, R_2 \ldots R_n$, & data item d_i appears in fragment R_j,

then d_i should not appear in any other fragment R_k $k \neq j$;

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

Replication

5 marks

It is useful in improving the availability of data by copying data at multiple sites.

- A relation or a fragment can be replicated at one or more sites.
- Fully redundant databases are those in which every site contains a copy of the entire database.

There are 3 types of Replication

1) Full Replication
2) No Replication
3) Partial Replication.

1) Full Replication

Whole database is copied at every site. This improves availability because the system can continue to operate as long as atleast one site is up.

2) Partial Replication

Some fragments of the database may be replicated others may not.

⇒ Data may be replicated row by row, table by table, database by database.

mcq on transaction management

+

PL/SQL

View, Sequence, Cluster, Index

Exception Handling

data fragmentation nd its types very very imp question pakka in paper

with eg

STI

Slabus Completed

ek sequence da ques ya fir cluster vagara vale topic cho 5 marks da question

fragmentation nd replication cho 5 ya 10 marks vala question very very imp