-1 distributed data.

23/07/19

Prof. Vivekanand. (1)

Sub:- ADBS (More Over Theoretical DISTRIBUTED DATABASES (DDB) Subject)

HP 1.

- we are focusing on Distributed Database System.

2 Test (each of 15 marks)
4 Assignments (each of 5 marks)

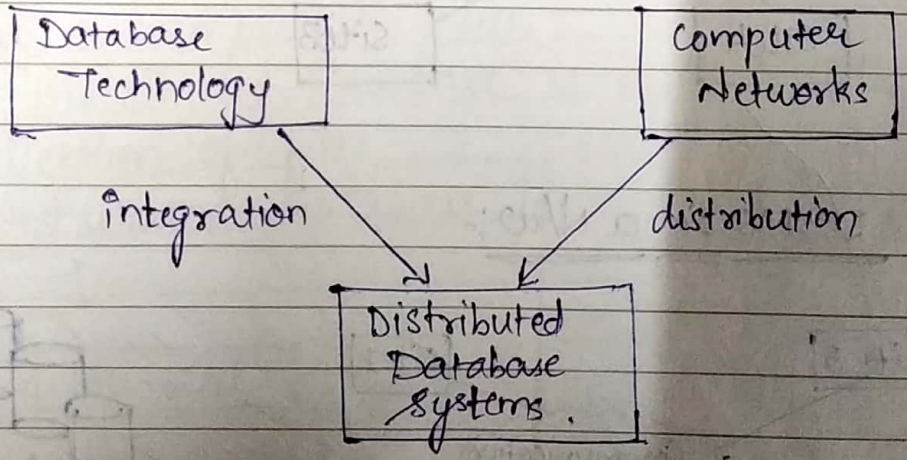
End Semester Exam → 50 Marks

Online Available

Principles of Distributed Database Systems. (Peausons Edition)
By M. Tamir Ozsu
Patrick Valduriez
S. Sridhar
} 3 authors for 1 book.

24/07/19

- Integrate Databases and Communication → DDBS



→ Distributed Computing :- A number of autonomous processing elements (not necessarily homogenous) that are interconnected by a Computer Network and that co-operate in performing their assigned task.

Q) What is distributed ?

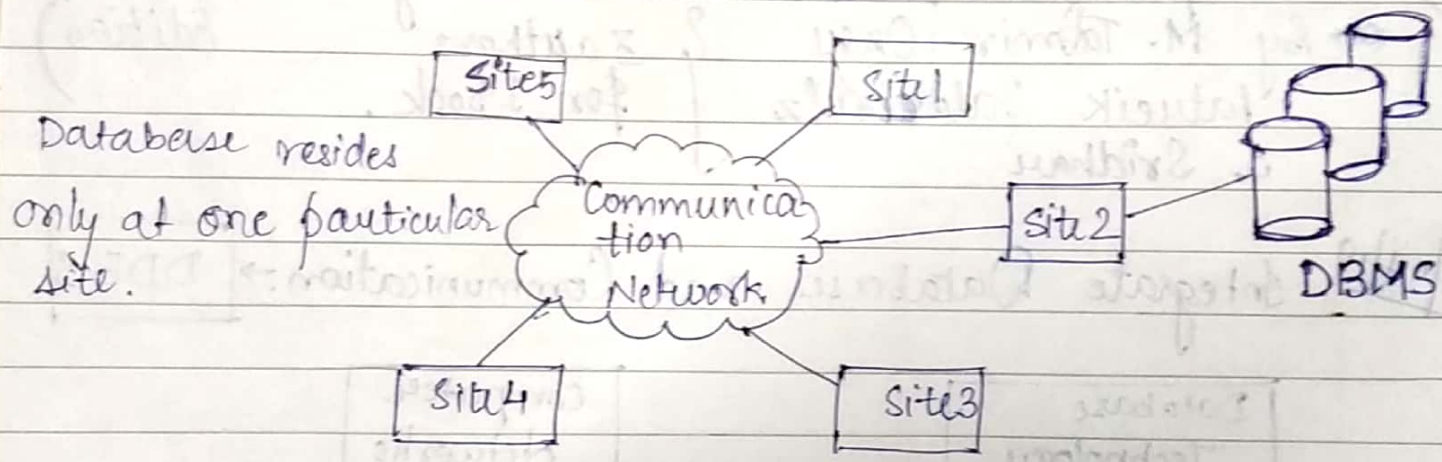
- Processing logic
- Functions
- Data.
- Control of Execution.

Q) what is not DDBS ?

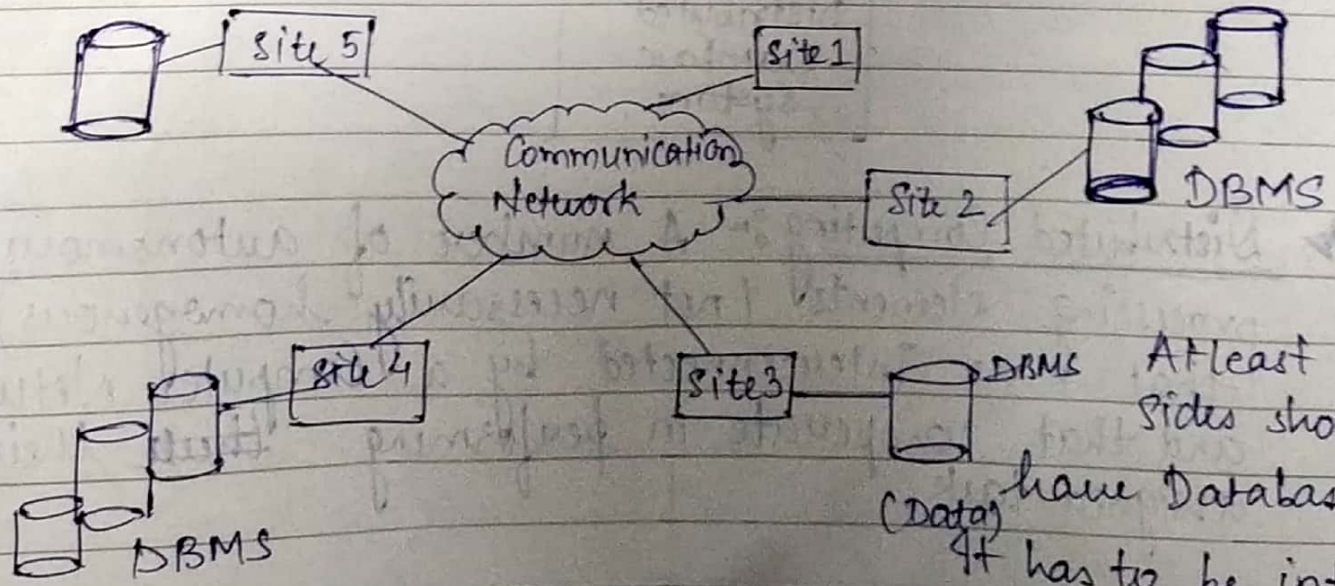
1. A time sharing computer system.
2. A loosely coupled system or tightly coupled multiprocessor system.

Loosely coupled system :- Memory shared by everyone.

3. Centralized DBMS on a Network.



Distributed DBMS on a N/w :-



Atleast 2 sides should have Databases.
(Data) It has to be interrela

eg:- Updated version of Oracle (S/w).

2.) What is a distributed Database system? (DDBS)

→ A DDB is a collection of multiple logically inter-related databases distributed over a computer N/w.

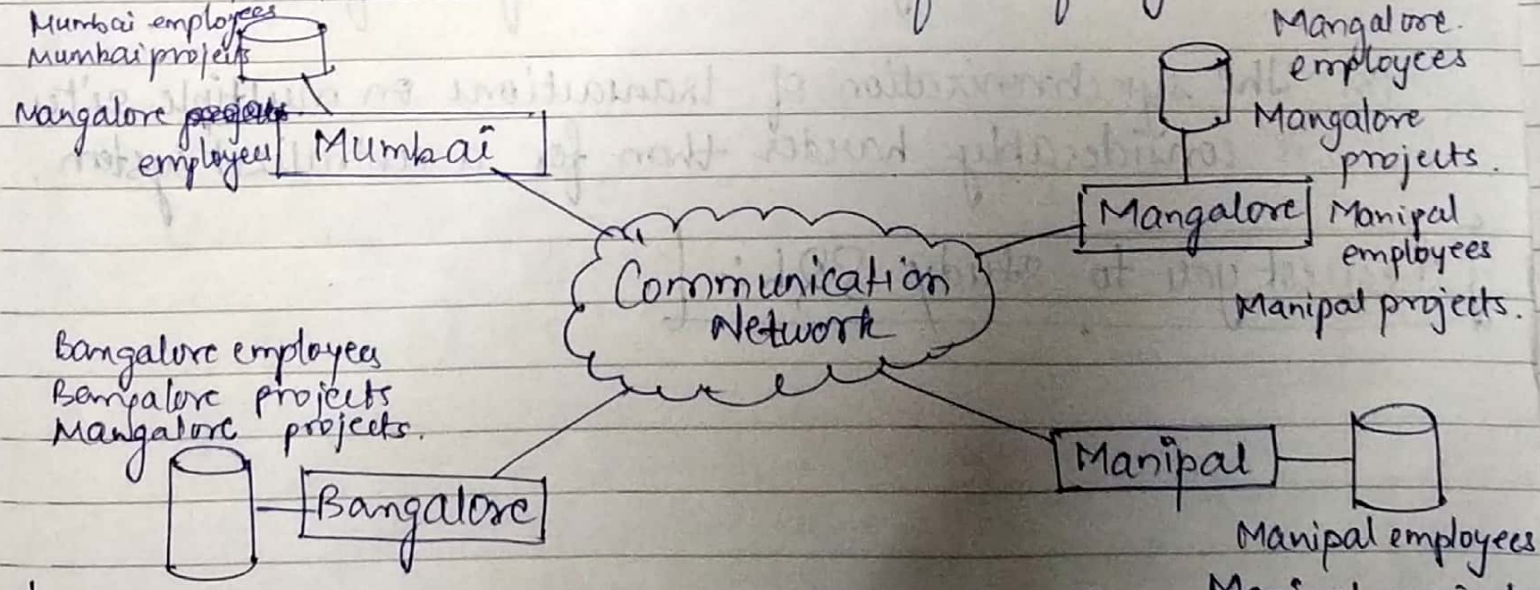
→ Distributed - DBMS (D-DBMS) :- management

A distributed DBMS is a software that manages the DDB and provides an access mechanism that makes this distribution transparent to the users.

Distributed database Systems = DB + Communication

26/07/2019

* Implicit Assumptions :-
  - Data stored at a number of sites.
  - Processors at different sites are interconnected by a computer n/w.
  - Distributed database is a database, not a collection of files.
  - Distributed Database is a full-fledged DBMS.



→ Typical eg of Distributed Database.

Distributed Database System { Pearson Edition 3

* **DDBMS promises (Advantages):-**
- Transfer and Management of distributed, fragmented & replicated data.
- Improved reliability and availability through distributed transactions.
- Improved performance.
- Easier and more economical system expansion. (scalability)

* **Complications introduced by Distribution :-**

(1.) Data may be replicated in a distributed environment.

(2.) Effect of update is reflected on each & every copy of data item.

(2.) If some site fails, while an update is being executed, the system must make sure that the effects will be reflected on the failed site as soon as the system recovers from failures.

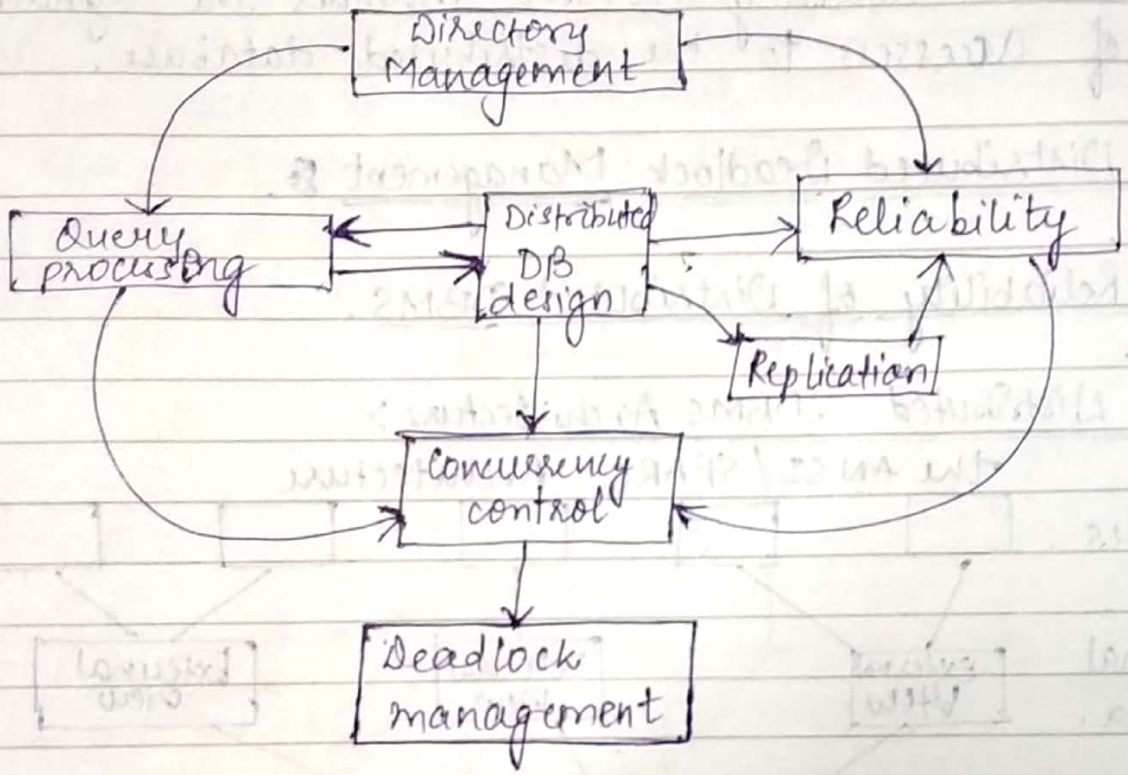
(3) The synchronization of transactions on multiple sites is considerably harder than for a centralized system.

Request you to study SQL :-

27/07/19

* Distributed DBMS Design Issues:

Relationship among various Design issues.



(1) Distributed DB Design:

Two ways of placing data: (1) partitioned [or non-replicated] and (2) replicated.

replicated: fully, partially.

(2) Distributed directory Management:

A directory contains information (such as location) about data items in the database.

(3) Distributed Query Processing:

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

(4) Distributed Concurrency control:-
Concurrency control involves the synchronization of accesses to the distributed database.

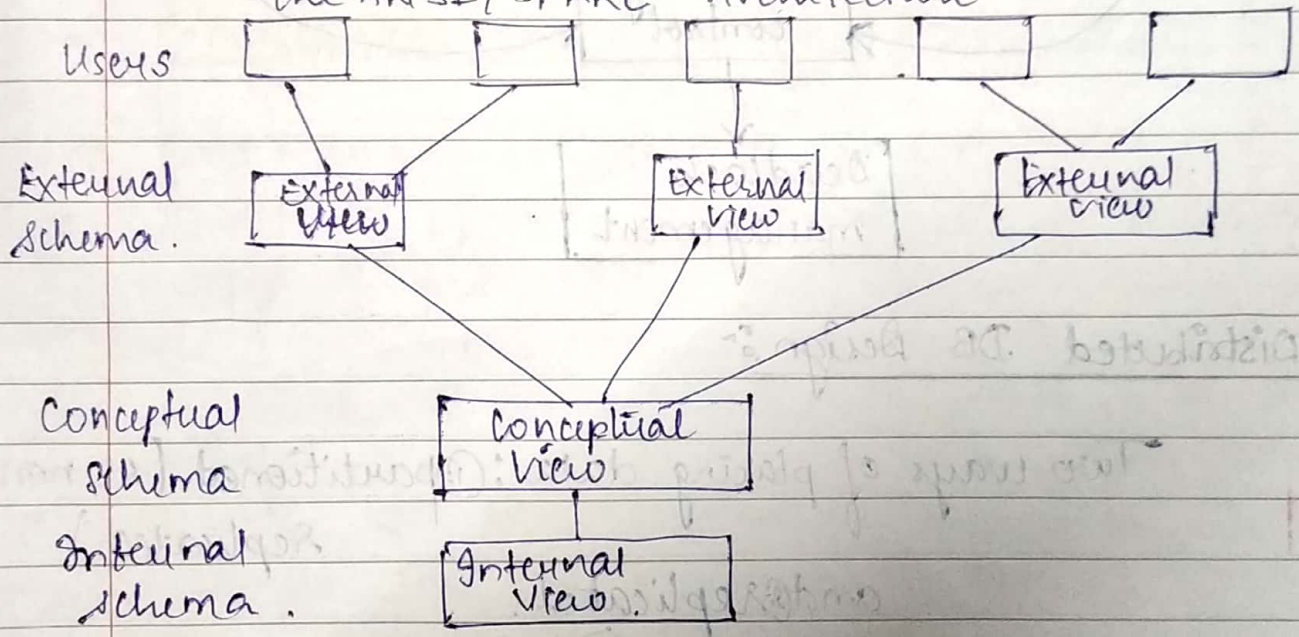
(5) Distributed Deadlock Management.

(6) Reliability of Distributed DBMS.

29/07/19

* Distributed DBMS Architecture:-

The ANSI/SPARC Architecture



(1) Internal View:- It is lowest level of the architecture. In the internal view how data actually stored in the database.

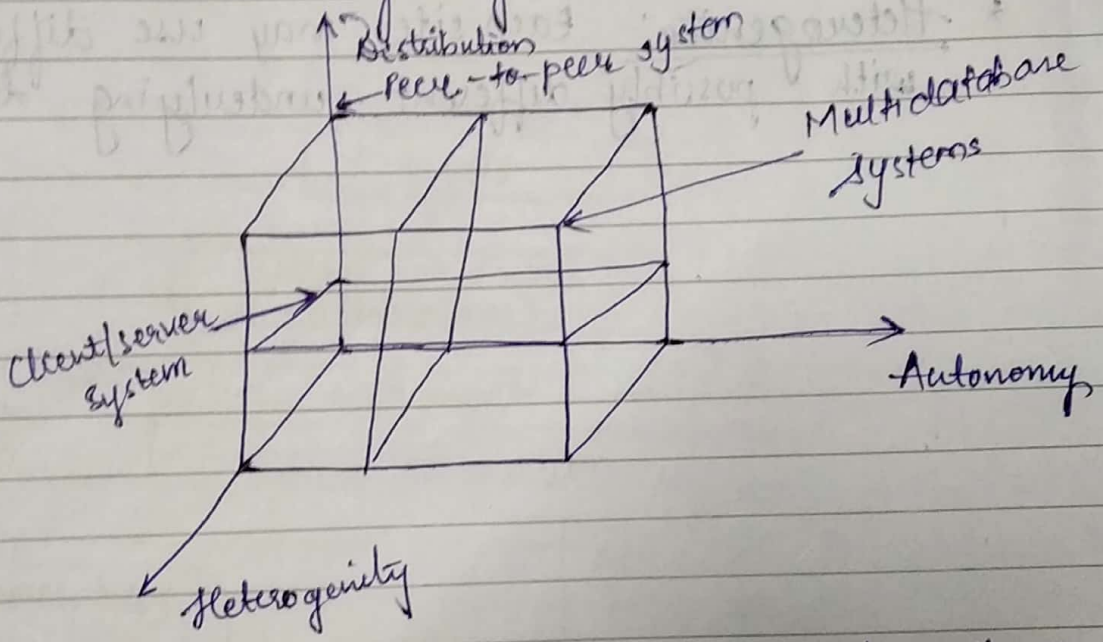
(2) Conceptual View (Logical View):- What data are stored in the database, & what relationship exist among those data.

(3) External View:- It is highest level of the architecture.

- how users view the database.

* Architectural Models of Distributed DBMSs.

The system is characterized with respect to
(1) the autonomy of local systems.
(2) their distribution.
and (3) their heterogeneity.

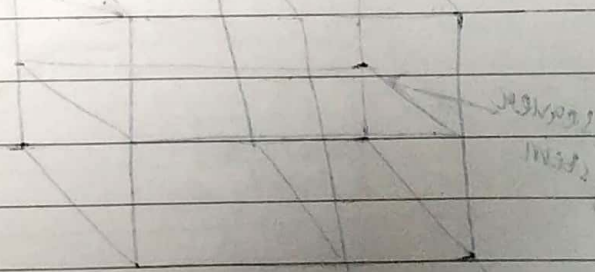


* Autonomy:- Autonomy refers to the distribution of control not of data.
It indicates the degree to which individual DBMSs can operate independently.
(1) Design Autonomy:- free to use data models that they prefer.
(2) Communication Autonomy:- each individual DBMSs is free to make its own decision or to what type of information it wants to provide to the other DBMSs.
(3) Execution Autonomy:- Each DBMS can execute the transaction in any way that it wants to.

Completed 1st chp. →

* Distribution :- Deals with data. We are considering the physical distribution of data over multiple sites.
Two types of distribution.
(1.) client / server distribution.
(2.) Peer - to - peer distribution (or full distribution).

* Heterogenity :- Each sites may use different H/W with possibly different underlying data models.



20/07/19

CHP 3:-
Distributed Database Design.

- Top - down Design process.
- Designing the system from scratch.

Requirements

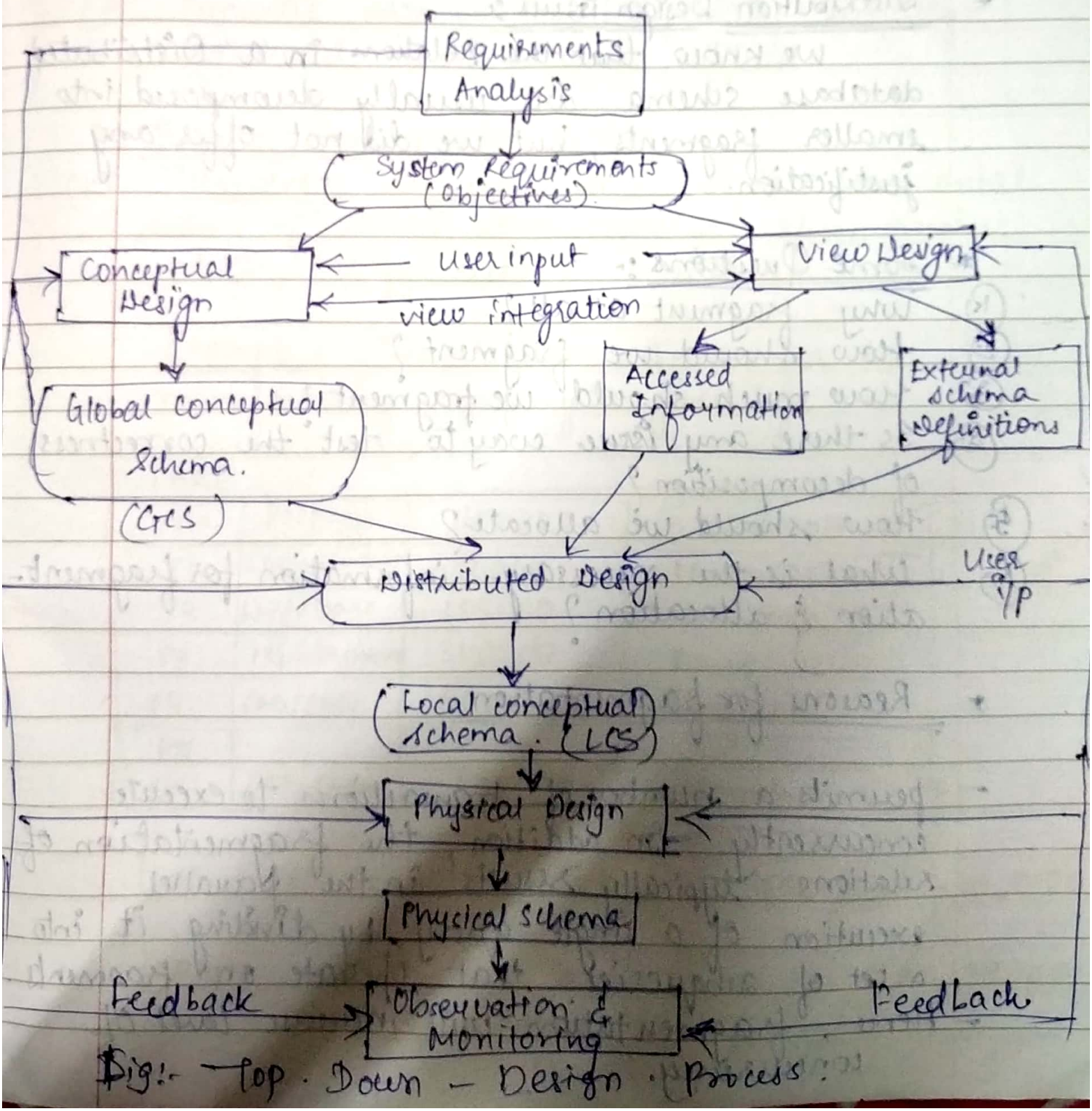


Fig:- Top. Down - Design. Process.

Requirements is input to 2 parallel activities View Design & conceptual design.

LCS distributes the entities over the sites of the distributed system.

* Distribution Design issues :-

We know that the relations in a Distributed database schema are usually decomposed into smaller fragments, but we did not offer any justification.

* Some Questions :-

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

* Reasons for fragmentation

- permits a number of transactions to execute concurrently. In addition, the fragmentation of relations typically results in the parallel execution of a single query, by dividing it into a set of subqueries that operate on fragments.
- Hence, fragmentation will increase level of concurrency.

* Fragmentation Difficulties:-

(1) Some applications whose views cannot be defined on a single fragment will request extra processing.

(2) Semantic data control (especially integrity enforcement) more difficult.

* Types of Fragmentation:-

(1) Horizontal Fragmentation:- It is a horizontal subset of a relation which contains those tuples which satisfy relation conditions.

Example:- The relation schema for the manufacturing company is given by

PROJ

PNO.	PNAME	BUDGET.	LOCATION
P1	Instrumentation	150000	Montrial
P2	Database Develop.	155000	New York.
P3	Maintainance	310000	Paris.
P4	CAD/CAM	500000	Bangkok.
P5	CAD/CAM	250000	New York

PROJ1 : Projects with budgets less than $200000.

PROJ2 : projects with budgets greater than or equal to $200000. $200,000

PROJ 1

PNO.	PNAME	Budget	Location
P1			
P2			

PROJ 2 :

PNO	PNAME	Budget	Location.
P3			
P4			
P5			

(2) Vertical Fragmentation :- It is a vertical subset of a relation which is created by a subset of columns.

Example: Table same as the previous one.

PROJ 1 : Information about project budgets.

PROJ 2 : Information about project names & locations.

PROJ 1 :-

PNO	Budget
P1	
P2	
P3	
P4	
P5	

PNO.	PNAME	LOCATION.
P1		
P2		
P3		
P4		
P5		

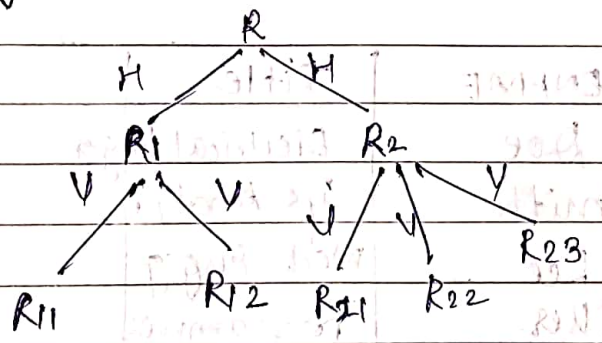
* Correctness of Fragmentation.

(1.) Completeness:-
— Decomposition of relation R into fragments $R_1, R_2, \ldots R_n$ is complete iff each data item in R can be found in some R_i.

(2.) Deconstruction:- $R = \bowtie_{1 \le i \le n} R_i$

(3.) Disjointness:- All the fragments are disjoint. if data is in R_1 then it should not be in $R_2, R_3, \ldots R_n$.

(3.) Hybrid fragmentation:-
In this case a vertical fragmentation may be followed by a horizontal one or vice-versa.



* Allocation:- (1.) Non replicated: Each fragment resides at only one side.

(2.) Replicated: (a.) fully replicated: Each fragment at each side.
(b.) partially replicated: Each fragment at some of the sites.

Study Chp 2 .

* Relational Algebra operating operations:-

(1.) Selection: Produces a horizontal subset of the operand relation

General form $\sigma_F(R) = \{t : t \in R \ \& \ F(t) \text{ is true}\}$

where R in a relation, t is a tuple variable

- F is a formula consisting of
  - operands that are constants or attributes.
  - arithmetic operations: $<, >, =, \neq, \leq, \geq$,
  - logical operators: $\wedge, \vee, \neg$ (NOT)

Example:-

EMP

ENO	ENAME	Title
E1	Doe	Electrical Engg
E2	Smith	Sys. Analyst
E3	Lee	Mech Engg
E4	Miller	Programmer
E5	Casey	Sys. Analyst.
E6	Chu	Electrical Engg.

$\sigma_{Title = 'Elect. Engg'}(EMP)$

ENO	ENAME	Title
E1		
E6		

(2) **Projection:** produces a vertical slice of a relation

General form.

$$\Pi_{A_1, \ldots, A_n}(R) = \{ t[A_1, \ldots, A_n] : t \in R \}$$

where R is a relation, t is a tuple variable $[A_1, \ldots, A_n]$ is a subset of the attributes of R over which the projection will be performed.

Example :-

PROJ

PNO	PNAME	Budget
P1	Instrumentation	150000
P2	DB Devl.	135000
P3	CAD/CAM	250000
P4	Maintain ance	310000

$\Pi_{PNO, Budget}(PROJ)$

PNO	Budget
P1	
P2	
P3	
P4	

9/08/19

(3.) Union :

(4.) Set Difference :

(5.) Cartesian Product → Natural Join
→ ~~Semi~~ Join.

General

9/08/19

Union:-

General Form

$R \cup S = \{ t : t \in R \text{ or } t \in S \}$

where R, S are relations, t is a tuple variable.

- Results contain tuples that are in R or in S, but not both (duplicates removed).

Set Difference

General Form.

$R - S = \{ t : t \in R \ \& \ t \notin S \}$

- Result contains all tuples that are in R, but not in S.

$R - S \neq S - R$.

Cartesian Product

- Given Relations
  - R of degree K_1 cardinality n_1
  - S of degree K_2 cardinality n_2

- Cartesian product (or cross product).

$R \times S = \{ t[A_1 \ldots A_K, A_{K+1} \ldots A_{K_1+K_2}] : t[A_1 \ldots A_K] \in R \ \& \ t[A_1 \ldots A_{K_2}] \in S \}$

The Result of $R \times S$ is a relation of degree $(K_1 + K_2)$ and consists of all $(n_1 \times n_2)$ tuples, where each tuple is a concatenation of one tuple of R with one tuple of S.

Example:

EMP.

ENO	ENAME	TITLE
E1	Doe	Elect. Engg.
E2	Smith	Syst. Analyst.
E3.	Lee	Mech. Engg.
E4.	Miller	Prog.
E5	Casey	Sys. Analyst
E6	Chee	Elect. Engg.
E7	David	Mech Engg.
E8	Jones	Syst. Analyst.

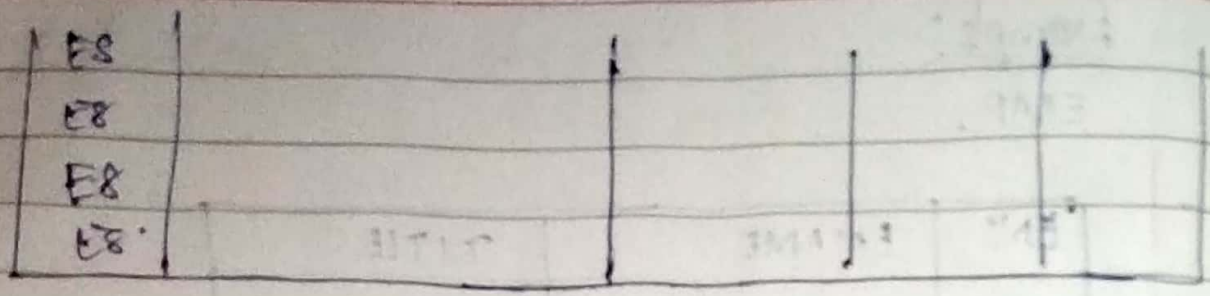
PAY

Title	Salary
Elect. Eng	55000
Syst. Analy.	70000
Mech. Engg	45000
Prog.	60000

EMP × PAY

EMPNO.	ENAME	EMP. Title	PAY. Title	Salary
E1	Doe	Elect Eng	Elect. Eng.	55000
E1	Doe	—"—	Syst Analyst	~~70000~~
E1	Doe	—"—	~~Mech Eng~~	~~45000~~
E1	Doe	—"—	~~Prog~~	~~60000~~
E2	Smith	Syst. Analyst	Elect.	55000
E2	Smith	—"—	Syst.	70000
E2	Smith	—"—	Mech	45000
E2	Smith	—"—	Prog	60000.

and so on.



~~Intersection~~

* **Intersection :-**

General form

$R \cap S = \{ t : t \in R \& t \in S \}$

$= R - (R - S)$

θ - join :- (Join operator).

The most general forms of join in the θ-join, commonly called the join.

The θ-join of 2 relations R & S is denoted as

$R \bowtie_F S$

where F is a formula defined as that of selection specifying join predicate of the form $R.A \; \theta \; S.B$, where A & B are attributes of R & S. respectively.

. join is a derivative of cartesian product.

$R \bowtie_F S = \sigma_F (R \times S)$.

* **Types of join :-**

(1) **Equi - join.** The formula F only contains equality predicate

$R \underset{R.A = S.B}{\bowtie} S$.

Mgr. no = Emp. no.

(2) Natural Join :- Equi-join of 2 relations R & S over an attribute (or attributes) common to both R and S & projecting out one copy of those attributes.

- $R \bowtie S = \pi_{R \cup S} \sigma_F (R \times S)$.

EMP × emp. title = pay. title PAY

ENO	ENAME	Title	Salary
E1	Doe	Elect. Engg	55000.
E2	Smith	Syst. Anal.	70000
E3	Lee	Mech Engg	45000
E4	Miller.	Prog	60000
E5	Casey	Elect. Eng	55000
E6	Chu	Syst. Anal.	70000
E7	David	Mech Engg	45000
E8	Jones.	Prog	60000

Semi-join

$$R \ltimes_F S = \pi_A (R \bowtie_F S) = \pi_A (R) \bowtie \pi_{A \cup B} (S) = R \bowtie_F \pi_{A \cup B} (S)$$

- R & S are relations.
- A is a set of attributes.

The semi-join of relation R, defined over the set of attributes A, by relation S, defined over set of attributes B, is the subset of the tuples of R that participate in the join of R with S.

EMP ⋈ EMP.title = PAY.title PAY

ENO	ENAME	title
E1		
E2		
E3		
E4		
E5		
E6		
E7		
E8		

* Problems:- Consider schema of the University database.

classroom(building, room number, capacity)
department(dept_name, building, budget)

course (course_id, tittle, dept_name, credits)
instructor(ID, name, dept_name, salary)
section (course_id, sec_id, semester, year, building, room_number, time_slot_id)
teacher (ID, course_id, sec_id, semester, year)

student (ID, name, dept_name, tot_cred)

takes (ID, course_id, sec_id, semester, year, grade)

advisor (s_id, i_id)

time_slot (time_slot_id, day, start_time, end_time)

prereq(course_id, pre_req_id)

(1) Find those tuples of the instructor relation where instructor is in the "physics" department.

$\sigma_{dept\_name = "physics"}$ (instructor)

(2) Find all instructors with salary greater than $90,000.

$\sigma_{salary > 90,000}$ (instructor)

(3) Find the instructors in "physics" department with a salary greater than $90,000.

$\sigma_{dept\_name = "physics" \wedge salary > 90,000}$ (instructor)

(4) List all instructor' ID, name & salary

$\Pi_{ID, name, salary}$ (instructor).

(5) Find the names of all in the physics department.

Π_{name} ($\sigma_{dept\_name = "physics"}$ (instructor)).

(6) Find the set of all courses taught in the Fall 2009 semester, the spring 2010, or both.

The set of all courses taught in the Fall 2009 semester.

Π_{course_id} ($\sigma_{semester = "Fall" \wedge year = 2009}$ (section))

the set

The set of all courses taught in the spring 2010 semester.

$$\Pi_{course\_id}(\sigma_{semester = "spring" \wedge year = 2010}(section))$$

(7) Find the set of all courses taught in the Fall 2009 semester but not in spring 2010.

$$\Pi_{course\_id}(\sigma_{semester = "Fall" \wedge year = 2009}(section)) - \Pi_{course\_id}(\sigma_{semester = "spring" \wedge year = 2010}(section))$$

(8) Find names of all instructors in the physics department together with the course-id of all courses they taught.

$$\Pi_{name, course\_id}(\sigma_{instructor.ID = teacher\_ID}(\sigma_{dept\_name = "physics"}(instructor \times teaches)))$$

(9) Find the names of all instructors together with course-id of all courses they taught

$$\Pi_{name, course\_id}(\sigma_{instructor\_ID = teaches\_ID}(instructor \bowtie teaches))$$

↓ Natural Join (performed on common relation)

If you write Cartesian products then it will contain duplicates.

$$\Pi_{name, course\_id}(instructor \bowtie teaches)$$

$(R_1 \bowtie R_2) \bowtie R_3 = R_1 \bowtie (R_2 \bowtie R_3)$ Join opⁿ is Associative

[Revise SQL Completely]

(10.) Find the names of all instructors in comp sci. dept together with the course titles of all the courses that the instructor teach.

$\Pi_{name, title}\left(\sigma_{dept\_name = "comp.sci"}(instructor \bowtie teaches) \bowtie course\right)$

* Overview of SQL:-

The SQL has several parts:

(1.) Data-definition Language (DDL): The SQL DDL provides commands for definition relation schemes, deleting relations, & modifying relation schemes;

Example:- The following SQL DDL statement defines the department table.

```
Create table department
    (dept_name char(20),
     building char(15),
     budget numeric(12,2));
```

(2.) Data-manipulation Language (DML):- The SQL DML provides the ability to query information from the DB and to insert tuples into, delete tuples from, & modify tuples in the DB.

Example:- To find name of all instructors in comp.sci. dept.

```
Select name
from instructor
where dept_name = "comp_sci".
```

Basic Query Structure.

A typical SQL query has the form

Select $A_1, A_2, \ldots, A_n$
from $R_1, R_2, \ldots, R_m$
where P

- A_i represents an attribute
- R_i represents a relation
- P is a predicate
- The result of an SQL query is a relation.
- SQL allows duplicates in relation as well as in query results.
- To eliminate duplicates, insert the keyword distinct after select
- Find the department names of all instructors, & remove duplicates.

```
select distinct dept_name
from instructor
```

The keyword all specifies that duplicates should not be removed.

```
Select all dept_name
from instructor
```

- An asterisk (*) in the select clause denotes "all attributes".

select *
from instructor

(1) Find all instructors in comp sci. dept with salary > 80000.

```
Select name
from instructor
where dept_name = 'comp.sci' and salary > 80000
```

(2) For all instructors in the university who have taught some course, find their names and course_Id of all courses they taught.

```
Select name, course_id
from instructor, teaches
where instructor_Id = teaches.Id;
  Or
Select name, course_id
from instructor natural join teaches;
```

(3) List the ~~names~~ name of instructors along with the titles of courses that they teach.

(cartesian product)

```
Select name, title
from instructor natural join teaches, course
where teaches.course_ID = course.course_Id.
```

f s..   f s .   char. → . &
f s.   1 2

(4) Increase salaries of instructors whose salary is over $1,00,000 by 3%.

Update instructor
set salary = salary × 1.03
where salary > 100000;

20/8/19

Data and Access Control.

Distributed DBMS supports semantic data control, ie. data and access control using high-level semantics.

Semantic data control typically includes

- view management
- security control.
- & semantic integrity control.

Informally, these functions must ensure that authorized users perform correct operations on the dB.

View Management :-

A view is a virtual relation, defined on the result of a query on base relation, but not materialized like a base relation, which is stored in the DB.

Example :-

manipalglobal3 | courcera.manipal.edu.

Consider EMP Relation

EMP

ENO	ENAME	Title
E1	T. Doe.	Elect. Eng
E2	M. Smith	Syst. Anal.
E3	A. Lee.	Mech. Eng
E4	J. Miller.	Prog.
E5	B. Casey	Sys. Anal
E6	L. chu.	Elect Eng.
E7	R. David	Mech. Eng
E8	T. Jonas	Sys. Analyst.

The view of system Analysts (SYSAM) derived from relation EMP by following SQL query.

```
Create view SYSAN(ENO. ENAME)
As Select ENO ENAME
from EMP
where Title = "Syst. Anal"
```

SYS.AN

ENO	ENAME
E2	M. Smith
E5	B. Casey
E8	T. Jonas

Example:- Find the name of all the system analysts with their project number & responsibility.

Consider relation schema of manufacturing company

EMP (ENO, ENAME, Title)
PAY (Title, Sal)
Proj (PNO, Pname, budget, Loc)
Asg (ENO, PNO, Resp, Dur).
→ Response.

28/08/19

(1) Find the names of all the system analysts with their project number & responsibility.

ASG → Assignment table.

eno	pno	Resp	Dur
e1	p1	Manager	12
e2	p1	Analyst	24
e2	p2	Analyst	6
e3	p3	Consultant	10
e3	p4	Engineer	48
e4	p2	Programmer	18
e5	p2	Manager	24
e6	p4	Manager	48
e7	p3	Engineer	36
e8	p3	Manager	40

Query expressed on views :-

```
SELECT ENAME, PNO, RESP
FROM SYSAN , ASG
WHERE SYSAN.ENO = ASG.ENO.
```

Query expressed on base relation :-

```
SELECT ENAME, PNO, RESP,
FROM EMP, ASG
WHERE EMP.ENO = ASG.ENO.
  AND TITLE = "sys.Analyst"
```

O/P:-

ename	pno	resp
M. Smith	P1	Analyst
M. Smith	P2	"
B. Casey	P2	Manager
T. Jones	P3	"

→ The view ESAME restricts the access by any user to those employees having same titles.

⇒ CREATE VIEW ESAME
AS SELECT *
FROM EMP E1, EMP E2.
WHERE E1.TITLE = E2.TITLE.
AND E1.END = USER

for eg:- the following query issued by the user J. Doe.

Select * from esame.

The result is:

Eno	Ename	Title
e1	J. Doe	EEE
e2	Z. Chu	"

8/19

Update through views

We can classify views as being updatable or not updatable.

A view is updatable only if the updates to the view can be propogated to the base relations without ambiguity.

Example: The view SYSAN above is updatable. The insertion of a new system analyst <201, Smith> will be mapped into the insertion of a new employee <201, Smith, syst. anal> If attributes other than Title were hidden by the view, would be assigned as null values.

Example: The view EG (ENAME, RESP) created from ASG relation. → Responsibility

```
Create view EG (ENAME, RESP)
As   Select Distinct ENAME, RESP
     from EMP, ASG
     where EMP.ENO = ASG.ENO.
```

the about view is not updatable.

Reason:

The deletion of the tuple <Smith, Analyst> cannot be propogated.

ie. It is ambiguous. ie which deletion has to be perform

System does not know first which to perform

You can delete Smith from ENAME & Analyst from Assignment but (Both are correct).

Views in Distributed DBMSs.

The definition of view is similar in a Distributed DBMS and in centralized systems.

However, a view in a distributed system may be derived from fragmented relations stored at different sites.

- when view is defined, its name & its ~~retrieved of~~ retrieval of query are stored in the catalog/ directory.
- Evaluating views derived from distributed relations may be costly.
- Use materialized view.

A materialized view stores the tuples of a view in a database relation like other database tuples, possibly with indices.

Materialized views in data warehouses typically involve aggregate functions (such as SUM & COUNT) & grouping (GROUP BY) operators, because they provide compact database summaries.

27/8/19.

Consider PROJ relation

PROJ

PNO	PNAME	Budget	Location
P1	Instrumentation	150000	Montreal
P2	DB develop	135000	New York
P3	CAD/CAM	250000	New York.
P4	Maintenance	310000	Paris

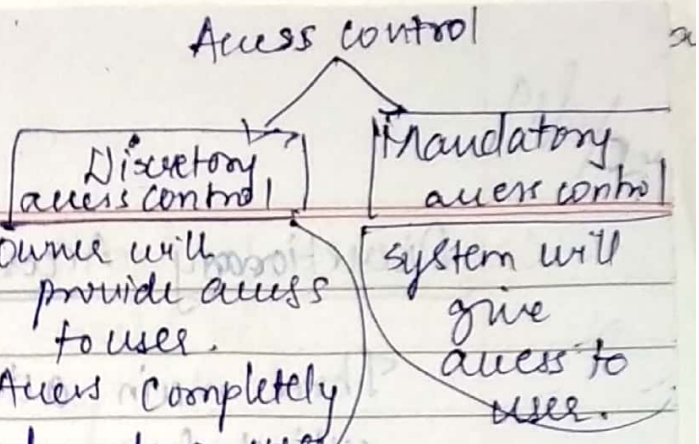
The following view over relation PROJ gives, for each location, the no. of projects and the total budget.

```
Create view PL(LOC, NBPROJ, TBUDGET)
    As Select LOC, COUNT(*), SUM(BUDGET)
        from PROJ
        group by LOC.
```

View PL has 3 tuples.

PL

LOC	NBPROJ	TBUDGET
Montreal	1	150000
NewYork	2	385000
Paris	1	310000.



* Data security:-

Data security is an important function of a database system that protects data against unauthorized access.

Data security includes 2 aspects : data protection & access control.

Data protection is required to prevent unathorized users from understanding the physical content of data.

Access control must guarantee that only authorized users perform operations they are allowed to perform on the database.

→ There are 2 approaches to database access control.

(1.) Discretionary access control (DAC):-

In DAC, the owner of the object specifies which subjects (eg. users, group of users) can access the object.

(2.) Mandatory access control (MAC) (or Multilevel access control).

In MAC, the system (i.e. & not the users) specifies which subjects can access specific data object.

30/8/19

Discretionary Access Control (DAC):-

Three main actors in DAC:-
- subjects (users, group of users) who execute op^ns.
- operations (select, Insert, update or Delete)
- objects (either Relations or attributes) on which op^ns are performed.

In an SQL-based Relational DBMS, an op^n in a high level statement such as select, insert, update or delete & rights are defined (granted or revoked using the following ~~statement~~ statements.

grant < operation type(s) > on < object > To < subject(s) >
Revoke < operation type(s) > on < object > To < subject (s) >.

The keyword "public" can be used to mean all users.

Example:- Consider authorization matrice.

	EMP	ENAME	MSG
Subjects: Carey	Update	Update	Update
Jones	Select	Select	Select where Resp ≠ "Manages".
Smith	None	Select	None.

for example one authorized op" for the pair <Jones, Relation EMP> could be
Select where title = "syst. Analyst"

which authorizes Jones to access only the employee tuples for system analysts.

DAC has limitations :-

for example, consider user A who has authorized access to relations R & S & user B. who has authorized access to relations S only. If B ~~sometime~~ somehow manages to modify an application program used by A so it write R data into S, then B can read an authorized data without violating authorization rules.

Multilevel access control answers this problem.

Multilevel Access Control.

It further improves the security by defining security levels for both subjects and data objects. Multilevel access control is based on the well known Bell & La Paduda Model.

- Security levels arranged in linear ordering.
  - Top Secret (TS) (highest)
  - Secret (S)
  - Confidential (C)
  - Unclassified (UC) lowest)

TS > S > C > UC.

Assignment 2.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Superssn	Dno

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

PROJECT

Pname	Pnumber	Plocation	Dnum

(1.) Retrieve the birth date & address of employee(s) whose name is 'John B. Smith'.

(2.) Retrieve the name and address of

$$\Pi_{\substack{Bdate,\\ Address}} \left(\sigma_{Fname = "John" \wedge Minit = "B." \wedge Lname = "Smith"} (EMPLOYEE) \right)$$

Select Bdate, Address
from EMPLOYEE
where Fname = "John" and Minit = "B" and
Lname = "Smith";

3/9/18

Continuation:-

~~Security~~ Access in read & write modes by subjects is restricted by 2 simple rules.

Read Rule (1.) A subject S is allowed to read an object of security level "l" only if level(s) ~~<~~ ≥ l. (no read up).

WriteRule.

(2.) A subject S is allowed to write an object of security level "l" only if level(S) ≤ l. (no write down).

Example:-

Security level	Subject	Object
TS	Tom	personal files
S	Samuel	Email files
C	Richard	Activity logs
UC	Harry	Telephone lists

$$l(R) \leq l(S)$$

$$l(\text{Richard}) \leq l(\text{Samuel})$$

- A multilevel relation PROJ* based on relation PROJ can be represented by adding a corresponding security level to each attribute.

PROJ* (security level)

PNO	SL1	PNAME	SL2	Budget	SL3	LOC	SL4
P1	C	Instrumentation	C	150000	C	Montreal	C
P2	C	DB Level	C	135000	S	NewYork	S
P3	S	CAD/CAM	S	250000	S	NewYork	S

3. Restricted Model: A data item has to be read before it can be updated. (written).

Ex:- T_3: { R(x), R(y), w(y), R(z), w(x), w(z), R(w), w(w), c }.

4. Restricted two-step model: Both two-steps restricted

Ex: T_4: { R(x), R(y), R(z), R(w), w(x), w(z), w(y), w(w), c }

5. Action Model: Each <read, write> pair be executed atomically.

Ex:- T_5: { [R(x), w(x)], [R(y), w(y)], [R(z), w(z)], [R(w), w(w)], c }.

Distributed Concurrency Control

The problem of synchronizing concurrent transactions such that the consistency of the database is maintained, while at the same time, maximum degree in concurrency achieved.

Anomolies:

1. Lost Updates: the effects of some transactions

... of distributed data.

are reflected on the database.

2. Inconsistent read: A transaction if it reads the same data item more than one, should always read the same value,

Execution Schedule (or History)

An order in which the operations of a set of transactions are executed.

Ex: T_1 Read (x) write (x) commit

T_2: write (x) write (y) Read (z) commit

T_3: Read (x) Read (y) Read (z) commit

$H_1 = \{ w_2(x), R_1(x), R_3(x), w_1(x), C_1, w_2(y), R_3(y), R_2(z), C_2, R_3(z), e_3 \}$.

26/10/19

Locking based concurrency control Algorithms

The main idea of locking based concurrency control to ensure that a data item that is stored by conflicting operations is accessed by one operation at a time.

Locking is an operation which secures.
a) permission to read.
b) permission to write a data item for a transaction.

Two lock models:
a) shared lock (X) (read)
More than one transaction can apply share lock on X for reading its value but no write lock can be applied on X by any other transaction.

b) Exclusive lock (X) (write)
Only one write lock on X can exist at any time & no shared lock can be applied by any other transaction on X.

conflict matrix

	read_lock (X)	write_lock (X)
read_lock (X)	compatible	not compatible
write_lock (X)	not compatible	not compatible.

Two-phase Locking Techniques.

This technique ensures conflict-serializable schedule.

The 2PL protocol execute transaction in 2 phases:

- phase 1: Growing phase
  A transaction applies locks (read or write) on desired data items one at a time.

- phase 2: Shrinking phase.
  A transaction unlocks its locked data one at a time.

Example:

T_1	T_2
read_lock(Y);	read_lock(X);
read_item(Y);	read_item(X);
unlock(Y);	unlock(X);
write_lock(X);	write_lock(Y);
read_item(X);	read_item(Y);
X = X+Y;	Y = X+Y;
write_item(X)	unlock(Y);
unlock(X)	

Transaction T_1 and T_2 will not follow 2PL.

Initial value X=20, Y=30

Result of serial execution T_1 followed by T_2
X=50, Y=80.

29/10/19

Deadlocks

2 PL does not ensure freedom from deadlock.

Example

T_1	T_2
write-lock (B)	
read (B)	
B = B − 50	
write (B)	
	read-lock (A)
	read (A)
	read-lock (B)
write-lock (B)	

Neither T_1 & T_2 can make progress
Executing write-lock (B) causes T_2 to wait for T_1 to release its lock on B.
while executing read-lock (A) causes T_1 to wait for T_2 to release its lock on A.
such a situation is called deadlock.