DISTRIBUTED DATABASE AN OVER VIEW

A Distributed database is a collection of data which belong logically to the same system but they are spread over the sites of a network. Two important aspects of distributed database are

1) Distribution

2) Logical correlation

Distribution:

The data are not resident in the same site that is it can be distributed among various sites within the network**.**

Logical correlation:

The data has some properties which tie them together so that a distributed database form a set of local database or files which are resident at different sites of computer network. It has to specify what type of network can be used for connecting the database and files**.**

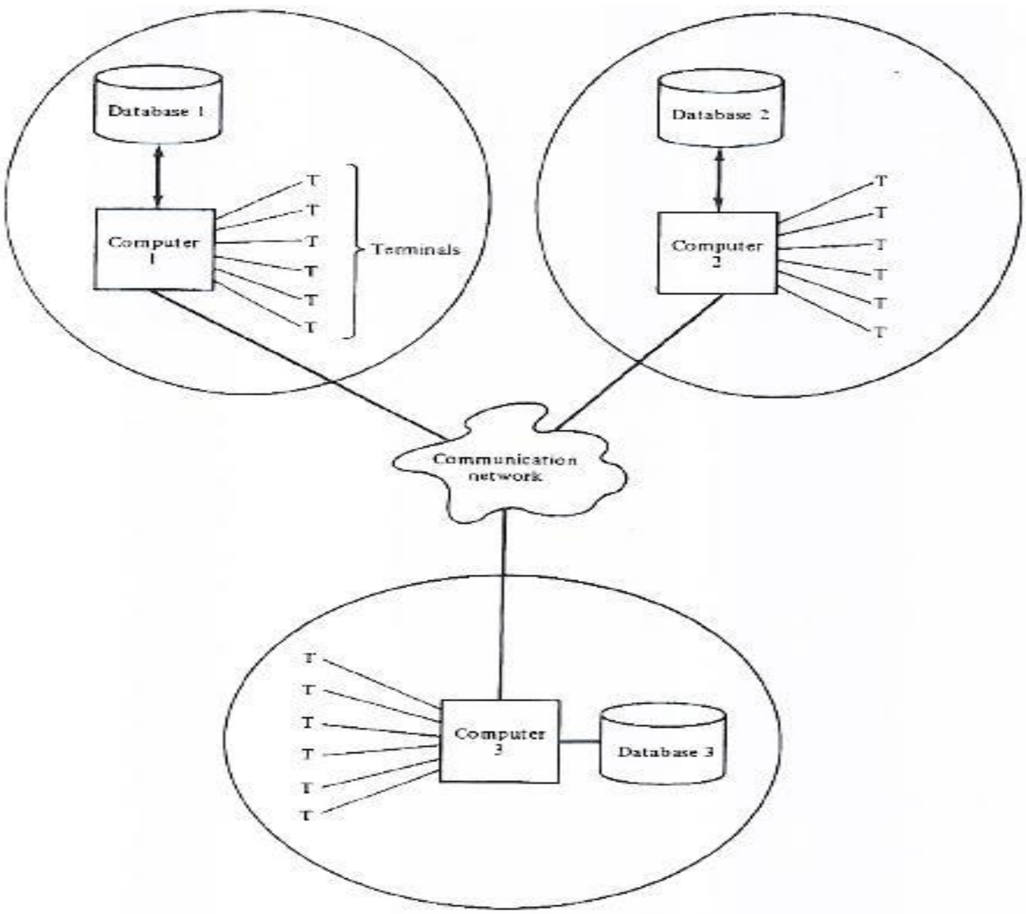


Figure 1.1 A distributed database on a geographically dispersed network.

A distributed database on a geographically dispersed network. In this example 3 branches are located at 3 different places each branch contains its own computer terminals and databases. Each branch can access the data locally or globally. It is otherwise known as local application or global application.

The global access of data is a tedious task so local network are used.

Example:2 Distributed database on a local network

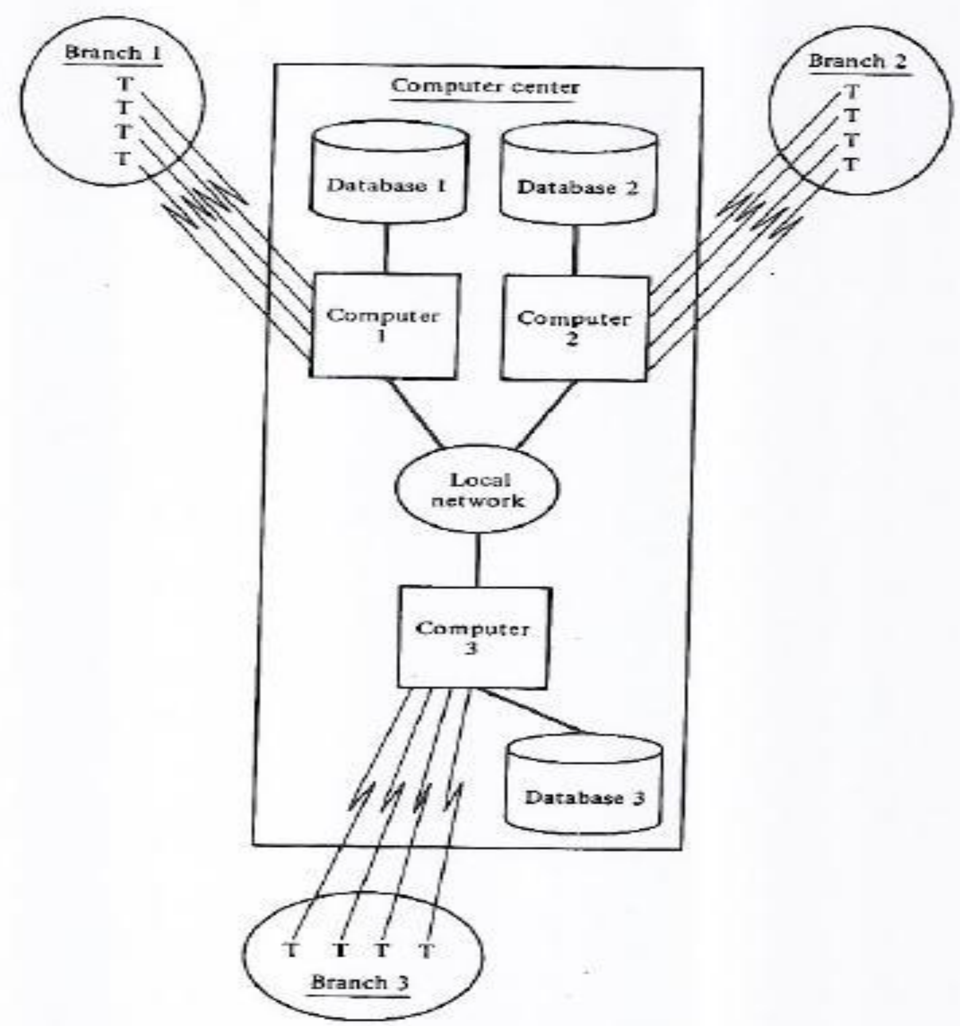


Figure 1.2 A distributed database on a local network.

The physical structure of a connection can be changed and the data that are accesses locally is to be considered local but the locality of the terminals is not defined with respect to the geographical distribution.

Local network will provide high throughput and reliability because the databases can be distributed.

Example 3: Multiprocessor system

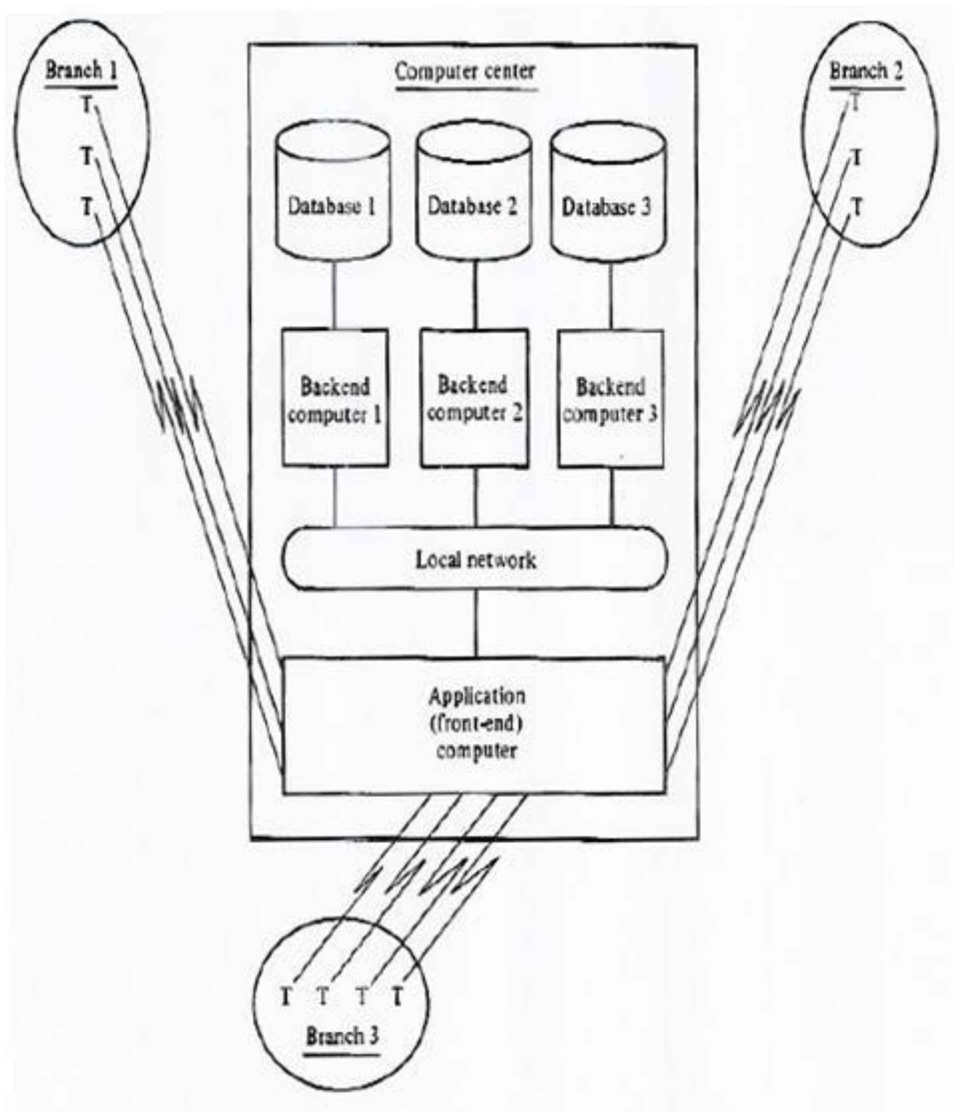


Figure 1.3 A multiprocessor system

Here the data is physically distributed over different processors but distribution is not based on the application or region. Here no computer is capable of executing an application by itself everything can be accessed thru the front end processor called front end application so the multiprocessor system is not a distributed system.

- So from these examples the distributed database can be defined as the co-operation database can be connected by the systems thru the region and each region or site has its autonomy for accessing the local and global applications.

FEATURES OF DISTRIBUTED VERSUS CENTRALIZED DATABASE:

- Centralized Control
- Data independence
- Reduction of redundancy
- Complex physical structures for efficient access
- Integrity , Recovery & Concurrency control.
- Privacy & Security.

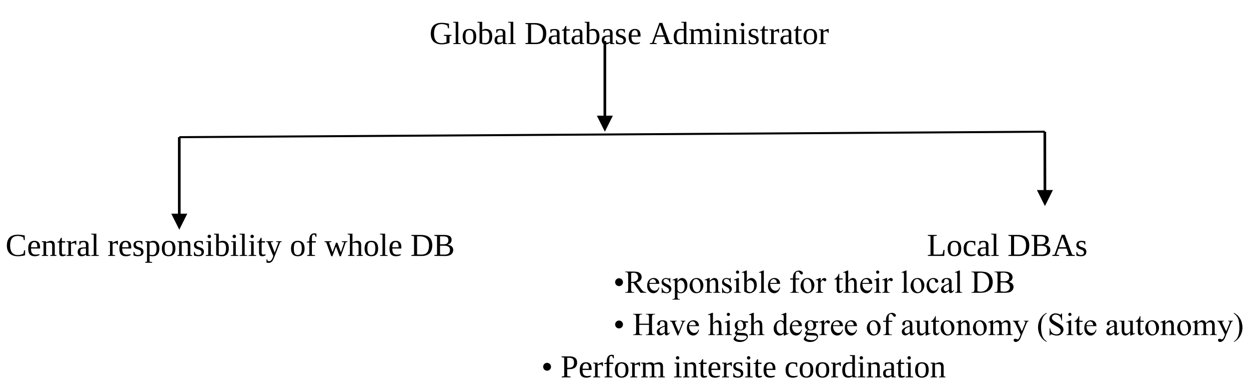
CENTRALIZED CONTROL:

• Provide centralized control over the information resources of a whole enterprise or organization

In DDB

• Depends on architecture (Example 1.2 lends to centralize control than 1.1)

• Identify **Hierarchical control structure**



• Site Autonomy vary from complete with no centralized DBA to completely centralized control

DATA INDEPENDENCE:

The actual organization of data is transparent to the application programmer. Programs written having conceptual view of data (conceptual schema) & unaffected by changes in physical organization of data.

In Traditional DB

- Multilevel architecture having different data description & mapping Conceptual , Storage and external schema developed.

In DDB

• Same importance as traditional DB.

• Introduce **Distribution Transparency**

• Programs can be written as if the database were not distributed.

• Correctness of programs unaffected by data movement from site to another while speed of execution is affected

• Obtained by introducing new levels and schemata

REDUCTION OF REDUNDANCY:

In Traditional DB

• Reduced by data sharing (several application access same files and records) for 1. Inconsistencies among several copies of the same logical data 2. Storage space saved

In DDB

• Data redundancy needed for 1. Increase locality of application if data replicated at all sites 2. Increase availability of the system as site failure does not stop application execution at other site.

• Data redundancy reduced for reasons same as Traditional DB.

• **Data replication** convenience **increase** with **ratio of retrieval accesses** (any copy) **versus update accesses** (all copies) performed by applications to it.

COMPLEX PHYSICAL STRUCTURE FOR EFFICIENT ACCESS:

In Traditional DB

• Secondary indexes, interfile chains & others.

• Their support is important for DBMSs

• Used to obtain efficient access to data

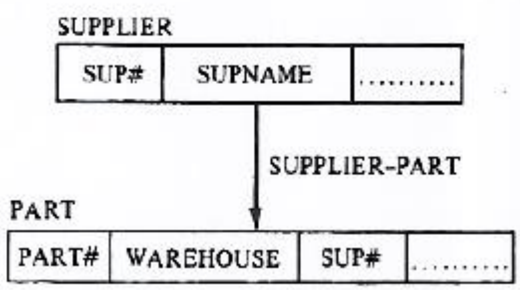
In DDB
• Not right tool for efficient access.

• Efficient access can’t be provided by this structure as
1. Very difficult to build and maintain such structures.
2. Not convenient to **navigate** at record level in DDB

Navigation example

• Find all PART records supplied by supplier S1

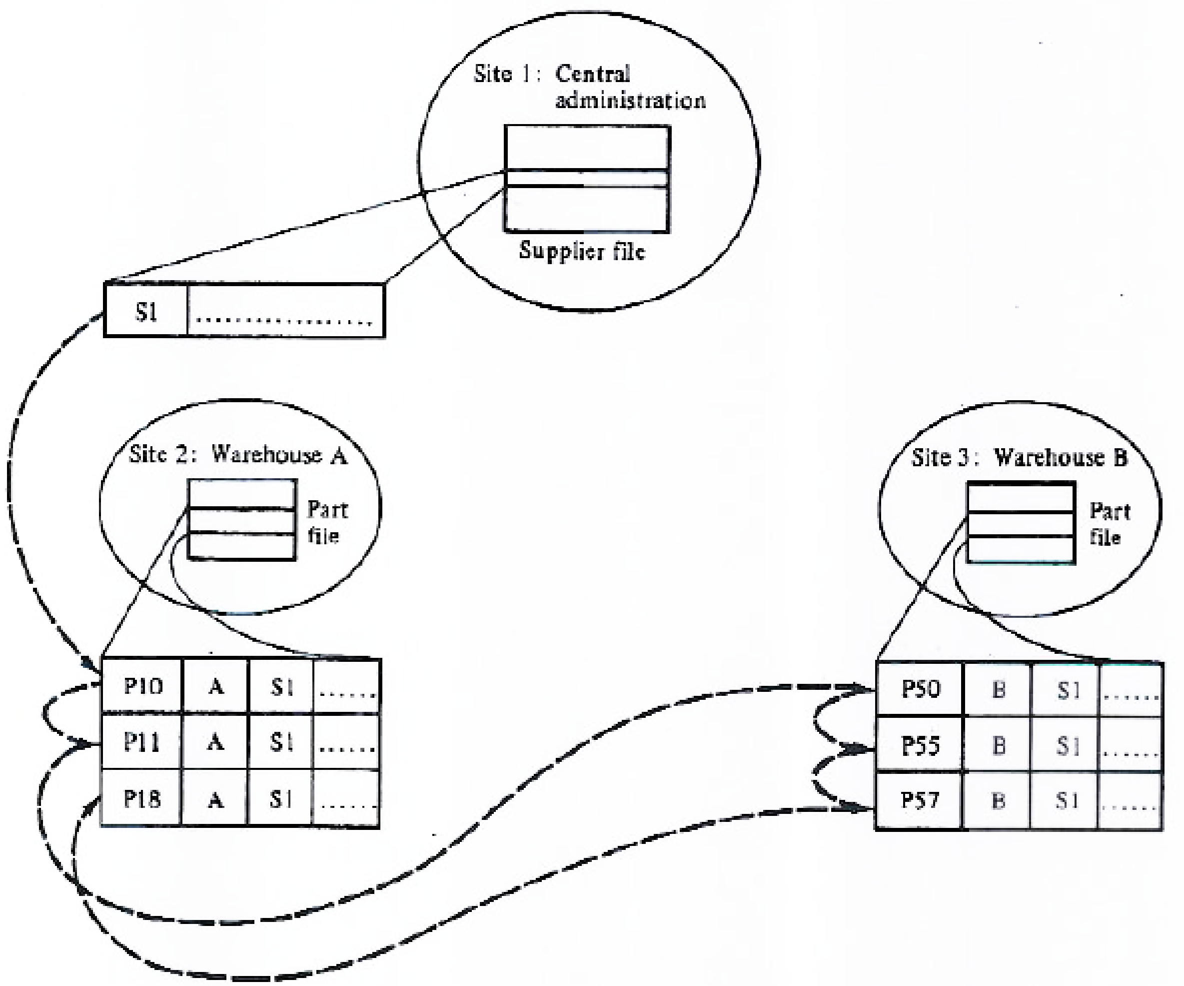
• Application run from site1



(a) A Codasyl database schema.

Find SUPPLIER record with SUP# = S1; Repeat until “no more members in set” Find next PART record in SUPPLIER-PART set; Output PART record;

(b) Codasyl-DBMS-like program



(c) Distribution of the SUPPLIER-PART set.

Figure 1.4 A distributed Codasyl-like database.

NAVIGATION EXAMPLE:

1) *At site 1*
   Send sites 2 and 3 the supplier number SN
2) *At sites 2 and 3*
   Execute in parallel, upon receipt of the supplier number, the following program:

   Find all PARTS *records having*
   SUP # = SN;
   Send result to site 1.

3) *At site 1*
   Merge results from sites 2 and 3;
   Output the result.

Figure 1.5 Example of access plan.

• More efficient implementation (grouping processes)

•**Distributed Access Plan :** how the data must be accessed
• As navigational programming in centralized DB.
• **Steps** 1. Execution of program local at single site
2. Transmission of files between sites
• **Can be written by programmer or automatically produced by optimizer.**

Optimizers' Design problems

Categories

Global Optimization

• Which data must be accessed at Which site& which data files must Consequently be transmitted between sites

Local Optimization

• how to perform local DB access at each site.
• Typical to traditional , no distributed DB pr problems

• Optimization parameter:
  • Communication cost
  • Accessing the local DBs cost

• Importance of these factors depend on **relation between communication cost** & **disk access cost** , which depend on communication network.

• Research here aids in understanding how DDB can be **efficiently accessed** even if access plans not produced automatically

INTERGRITY, RECOVERY AND CONCURRENCY CONTROL

• Strongly Correlated issues .

• **Solution** : providing transactions.

• **Transaction**
•Definition: Atomic unit of execution – set of operations performed entirely or not at all.

• **Example:** Funds transfer example (debit & credit)

• **Problem:** debit at an operation site & credit at non operational site

• **How to act ?**! Abort transaction or find smart way to execute transfer even if sites not simultaneously operating ?

• **Transaction atomicity** enemies

  • Failures

  • Concurrency

DB integrity

• **Transaction** atomicity assure **DB integrity** by assuring all actions transfer DB from consistent state to another are performed or initial consistent state is preserved.

• **Recovery:** Deals with preserving transaction atomicity in the presence of failures.

• **Concurrency Control:** Deals with ensuring transaction atomicity in the presence of concurrent execution of transactions.

Problems : Synchronization harder in DDB than in centralized DB

PRIVACY AND SECURITY

In Traditional centralized DB

• DBA has centralized control

• DBA ensures only authorized access is performed

• Without specialized control procedures, is weak to privacy & security violations than older separate files based approaches

In DDB

• Local DBAs face same DBA problems in traditional DB.

• In DDB with very high degree of autonomy, local DBA more protected through enforcing their own protection instead of central DBA.

• Communication networks represents a weak point with respect to protection

•Problems of privacy & security

WHY DISTRIBUTED DATABASES

1. Organizational and economic reasons.

• Many decentralized organizations structurally fitted by DDB

• Economy of scale motivation for having large centralized computer centers.

2.Interconnection of existing DBs

• Necessity of performing global applications for DBs exist in organizations

• Creating bottom-up DDB from existing local DBs having less effort from completely new centralized DB creation

3. Incremental growth.

•Adding new relatively autonomous branches for organizations

•With centralized approach would have to Either take care for future dimension expansion in initial design – difficult & expensive Or the growth will have major impact on existing applications

4. Reduce communication overhead

• w.r.t. centralized DB as in example 1.1

• Maximization of locality of application is 1 primary objective in DDB design

5. Perform considerations

• Several autonomous processors

• High degree of parallelism – increase performance

• In DDB decomposition of data reflects application dependence criteria, maximize application locality ; mutual interference between different processors minimized.

• Load is shared between different processors

• Bottlenecks as communication network itself or common services of the whole system are avoided.

6. Reliability and availability

•Autonomous processing capability of sites do not guarantee reliability but insures **Graceful degradation property: failures** in DDB is can be **higher** than in centralized DB for greater **# of components** but failure **affect** only applications using failed site , complete system crash is rare.

• **Why DDB development begun ?**

1. Small computers instead of large mainframes constitutes necessary h/w needed.
2. DDB development depends on Computer Network& Database technologies Which are developed sufficiently.

Distributed Database Management Systems (DDBMSs)

Services provided by above type of systems are

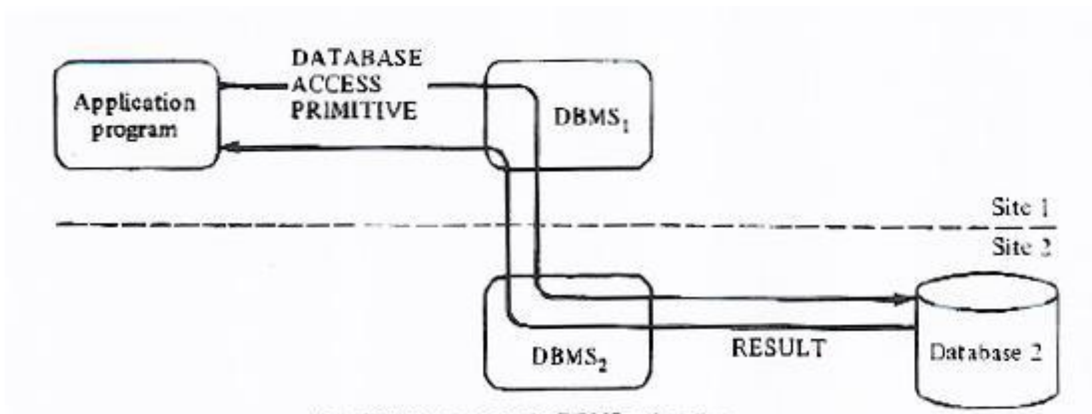
• Remote DB access by an application program

• Some degree of distribution transparency.

• Support for database administration & control

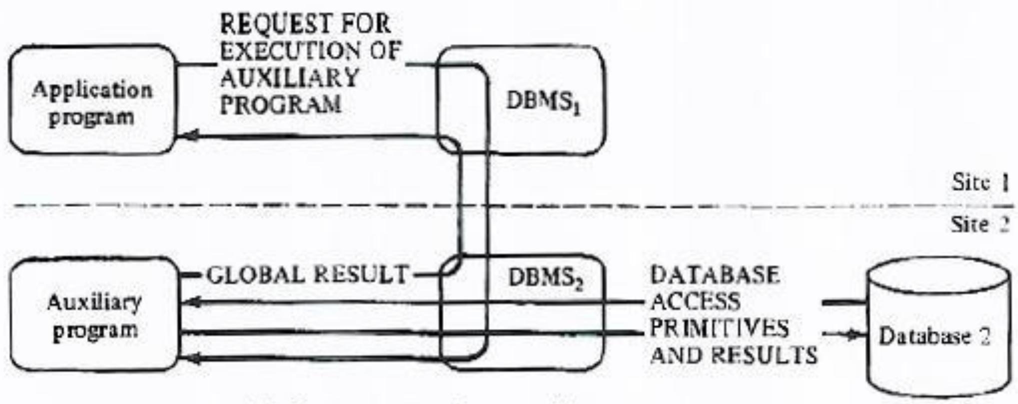
• Some support for concurrency control & recovery of distributed transactions

DDBMSs provides access remote DB by an application through



(a) Remote access via DBMS primitives.

- Units shipped between Systems by
  1.DB access primitive
  2.Result obtained by executing it
- Assures distribution transparency



(b) Remote access via an auxiliary program.

Figure 1.7 Types of accesses to a distributed database.

• Auxiliary program executed at remote site is required by application which
1.Access remote DB
2.Return the result to requesting application

•Efficient if many DB access is required for auxiliary program perform all required access and send only result back.

Homogeneity and Heterogeneity of DDBMSs

• Can be over
- Hardware
- Operating system
- Local DBMSs

} Managed by communication software

• **Homogenous DDBMS :**
  • DDBMSs with same DBMS at each site.
  • Preferred to be built in case of top-down without preexisting system development of DDB

• **Heterogeneous DDBMS :**
  • At least two different DBMSs.
  • Added translating between different models of DBMSs problem.(Ch.15) • Used in case of integrating preexisting DBs .
  • Actually systems supported some degree of it with no translation between different data model
  • Some systems support communication between different DC components

LEVELS OF DISTRIBUTION TRANSPERANCY

At different levels the application programmer view the distributed database depending on how much distribution is provided to DDBMS

<u>Reference architecture for Distributed Databases</u>

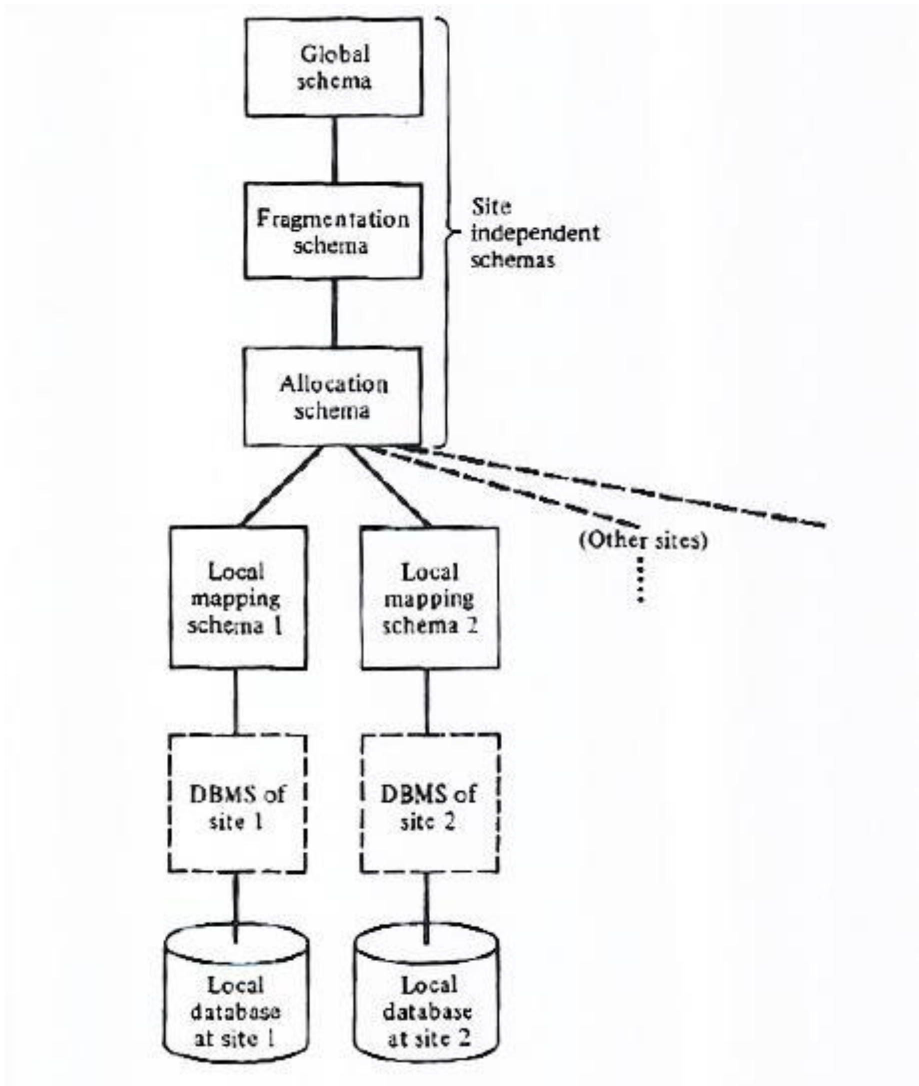


Figure 3.1 A reference architecture for distributed databases

Global Schema:

- Define all data contained in DDB as if DB is not distributed.
- Using relational model - Consists of the definitions of a set of **global relations.**
- Can be spitted to several no overlapping **Fragments.**

Fragmentation Schema:

• Defines the mapping between global relations and fragments(1:M mapping)

• Logical portions of physical global relations located at 1 or several sites of network

• Notation: **Ri** where R is the global relation , Ri is the *ith* fragment of R

Allocation Schema:

• At which site(s) the fragment is located.

• Type of mapping defined here determines DDB is redundant(1:M) or not(1:1).

• **Rj** indicates physical image of global relation R at site j

•A **copy of a fragment** at given site Donated using global relation name & 2 indexes(fragment index and site index) Indicates copy of fragment R2 located at site 3

Local mapping Schema:

• Map physical images to the objects which are manipulated by the local DBMSs.

•Depends on type of DBMS (different mapping in heterogeneous system) .

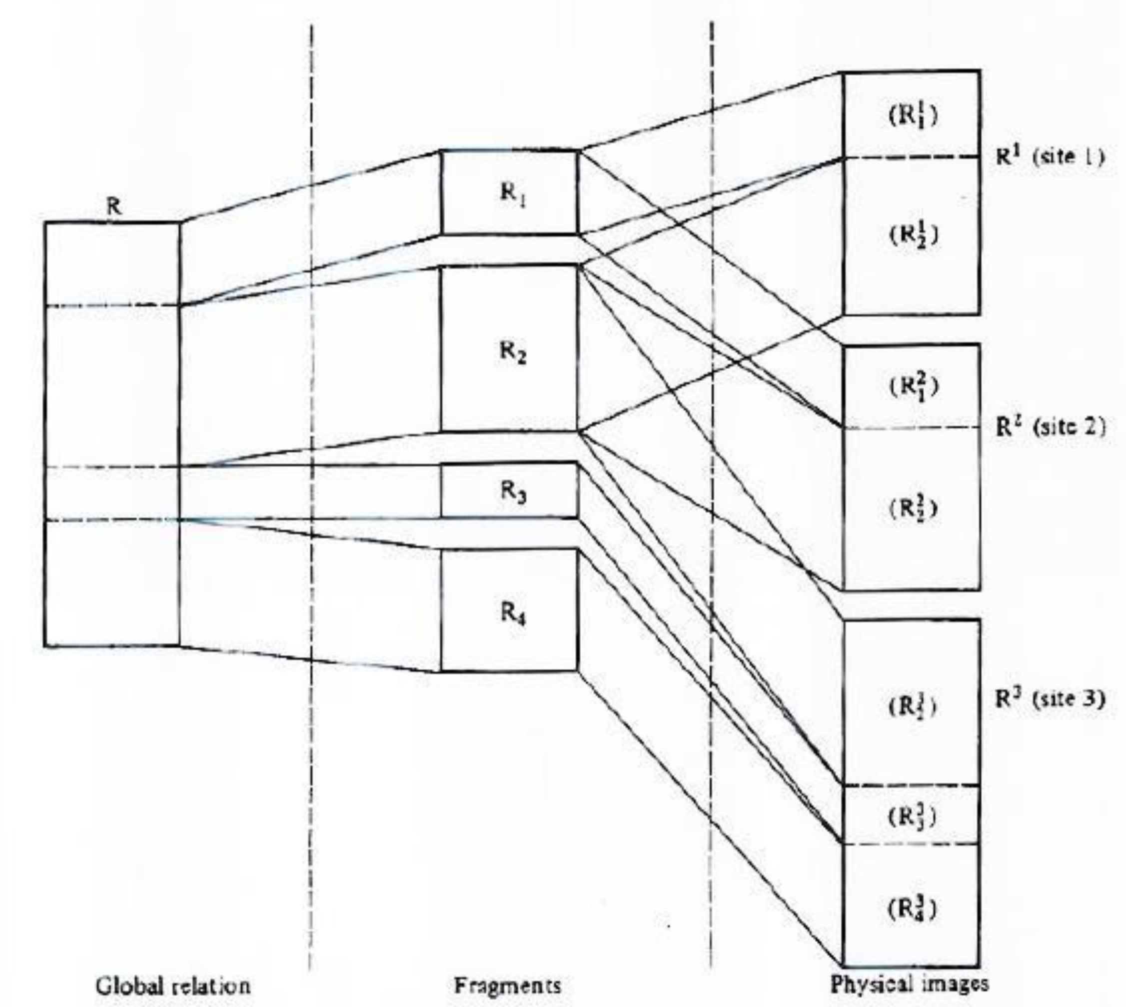


Figure 3.2 Fragments and physical images for a global relation.

Objectives motivate the architecture features:

1. Separating the data fragmentation concept from data allocation concept. Allow distinguish

Transparency

Fragmentation transparency	Location transparency
• Highest degree of transparency • Require user or application programmer Works on global relations.	• Lower degree of transparency • require user or application Programmer works on fragments Instead of global

2. Explicit control of redundancy at fragmentation level (R2 & R3 overlapping i.e. contain common data)
3. Independence from local DBMSs(called **Local Mapping transparency**) Allow study DDBM problems without taking in account specific data models of local DBMSs.

Replication Transparency:

- Implied by location transparency (not distinguish in book)
- User unaware of fragments replication.

Types of Data Fragmentation

HORIZONDAL FRAGMENTATION | **VERTICAL FRAGMENTATION**

• **A Fragment :** Expression in a relational language, taking global relations as operands and produces the fragment as a result.

• **Rules on defining fragments:**

1. Completeness condition: No data item do not belong to any fragment. - Set of qualifications (conditions) of all fragments must be complete
2. Reconstruction condition: Must be able to construct global relation from its fragment
3. Disjointness condition: Fragment be disjoint; so that replication of data can be controlled explicitly at allocation level. (HZ fragmentation)

• **Horizontal Fragmentation:**

• Partition tuples of global relation into subsets

• Example

$$SUPPLIER(SNUM, NAME, CITY)$$

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

$$SUPPLIER_1 = \mathbf{SL}_{CITY=\text{"SF"}} SUPPLIER$$
$$SUPPLIER_2 = \mathbf{SL}_{CITY=\text{"LA"}} SUPPLIER$$

•**Applying Rules of fragmentation:**

1. Completeness condition if “SF” and “LA” are only cities values
2. Reconstruction condition.

$$SUPPLIER = SUPPLIER_1 \text{ UN } SUPPLIER_2$$

3. Disjointness verified.

• **Derived Horizontal Fragmentation:**

• Example:

$$SUPPLY(SNUM, PNUM, DEPTNUM, QUAN)$$

$$SUPPLY_1 = SUPPLY\, \mathbf{SJ}_{SNUM=SNUM}\, SUPPLIER_1$$
$$SUPPLY_2 = SUPPLY\, \mathbf{SJ}_{SNUM=SNUM}\, SUPPLIER_2$$

•**Applying Rules of fragmentation:**

1. Completeness condition (Referential integrity constraint) no supplier # in SUPPLY not contained also in SUPPLIER.
2. Reconstruction condition
3. Disjointness verified if tuple in SUPPLY does not corresponds to 2 tuples of SUPPLIER relation which belong to 2 different fragments

• **Vertical Fragmentation:**

• Example:

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

A vertical fragmentation of this relation can be defined as

$$EMP_1 = \mathbf{PJ}_{EMPNUM,NAME,MGRNUM,DEPTNUM}\ EMP$$
$$EMP_2 = \mathbf{PJ}_{EMPNUM,SAL,TAX}\ EMP$$

This fragmentation could, for instance, reflect an organization in which salaries and taxes are managed separately. The reconstruction of relation *EMP* can be obtained as

$$EMP = EMP_1\ \mathbf{JN}_{EMPNUM=EMPNUM}\ EMP_2$$

For example, consider the following vertical fragmentation of relation *EMP*:

$$EMP_1 = \mathbf{PJ}_{EMPNUM,NAME,MGRNUM,DEPTNUM}\ EMP$$
$$EMP_2 = \mathbf{PJ}_{EMPNUM,NAME,SAL,TAX}\ EMP$$

The attribute *NAME* is replicated in both fragments. We can explicitly eliminate this attribute when we reconstruct relation *EMP* through an additional projection operation:

$$EMP = EMP_1\ \mathbf{JN}_{EMPNUM=EMPNUM}\ \mathbf{PJ}_{EMPNUM,SAL,TAX}\ EMP_2$$

- **Mixed Fragmentation:**
- Example:

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

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

$$EMP_1 = \mathbf{SL}_{DEPTNUM \leq 10} \mathbf{PJ}_{EMPNUM, NAME, MGRNUM, DEPTNUM}\, EMP$$
$$EMP_2 = \mathbf{SL}_{10 < DEPTNUM \leq 20} \mathbf{PJ}_{EMPNUM, NAME, MGRNUM, DEPTNUM}\, EMP$$
$$EMP_3 = \mathbf{SL}_{DEPTNUM > 20} \mathbf{PJ}_{EMPNUM, NAME, MGRNUM, DEPTNUM}\, EMP$$
$$EMP_4 = \mathbf{PJ}_{EMPNUM, NAME, SAL, TAX}\, EMP$$

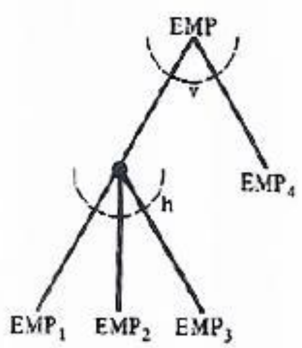


Figure 3.3 The fragmentation tree of relation *EMP*.

The reconstruction of relation *EMP* is defined by the following expression:

$$EMP = \mathbf{UN}\,(EMP_1, EMP_2, EMP_3)\,\mathbf{JN}_{EMPNUM = EMPNUM}\; \mathbf{PJ}_{EMPNUM, SAL, TAX}\, EMP_4$$

Global schema

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

$DEPT(DEPTNUM, NAME, AREA, MGRNUM)$

$SUPPLIER(SNUM, NAME, CITY)$

$SUPPLY(SNUM, PNUM, DEPTNUM, QUAN)$

Fragmentation schema

$EMP_1 = \mathbf{SL}_{DEPTNUM \leq 10} \mathbf{PJ}_{EMPNUM, NAME, MGRNUM, DEPTNUM}(EMP)$

$EMP_2 = \mathbf{SL}_{10 < DEPTNUM \leq 20} \mathbf{PJ}_{EMPNUM, NAME, MGRNUM, DEPTNUM}(EMP)$

$EMP_3 = \mathbf{SL}_{DEPTNUM > 20} \mathbf{PJ}_{EMPNUM, NAME, MGRNUM, DEPTNUM}(EMP)$

$EMP_4 = \mathbf{PJ}_{EMPNUM, NAME, SAL, TAX}(EMP)$

$DEPT_1 = \mathbf{SL}_{DEPTNUM \leq 10}(DEPT)$

$DEPT_2 = \mathbf{SL}_{10 < DEPTNUM \leq 20}(DEPT)$

$DEPT_3 = \mathbf{SL}_{DEPTNUM > 20}(DEPT)$

$SUPPLIER_1 = \mathbf{SL}_{CITY = \text{“SF”}}(SUPPLIER)$

$SUPPLIER_2 = \mathbf{SL}_{CITY = \text{“LA”}}(SUPPLIER)$

$SUPPLY_1 = SUPPLY \, \mathbf{SJ}_{SNUM = SNUM} SUPPLIER_1$

$SUPPLY_2 = SUPPLY \, \mathbf{SJ}_{SNUM = SNUM} SUPPLIER_2$

Figure 3.4 The global and fragmentation schemata of EXAMPLE_DDB.

Distribution transparency for Read-only Applications Language definitions:

Language definitions:

- All variables: strings(arrays)
- Input : read(filename, variable)
- Output: write(filename, variable)
- Filename : “terminal” if I/O performed at terminal
- Pascal var used in SQL statement: prefixed with $ symbol
- Pascal var used for Success or failure of a required DB operation: prefixed with # symbol
- SQL I/O

```
Select NAME into $NAME
from SUPPLIER
where SNUM = $SNUM
```

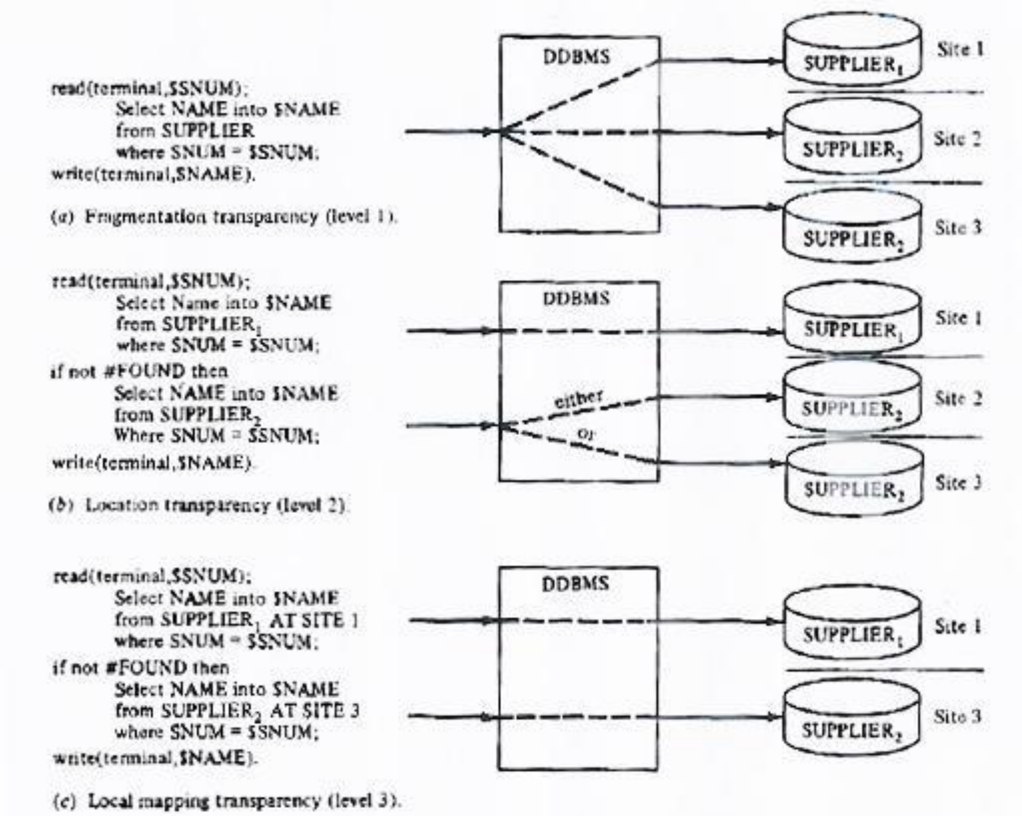


Figure 3.5 The read-only application SUPINQUIRY at different levels of distribution transparency.

(SUPINQUIRY) In 3.5-b can be written as

```
SUPINQUIRY:
        read (terminal, $SNUM);
        read (terminal, $CITY);
        case $CITY of
            "SF": Select NAME into $NAME
                  from SUPPLIER1
                  where SNUM = $SNUM;
            "LA": Select NAME into $NAME
                  from SUPPLIER2
                  where SNUM = $SNUM
        end;
        write (terminal, $NAME).
```

(SUPINQUIRY)

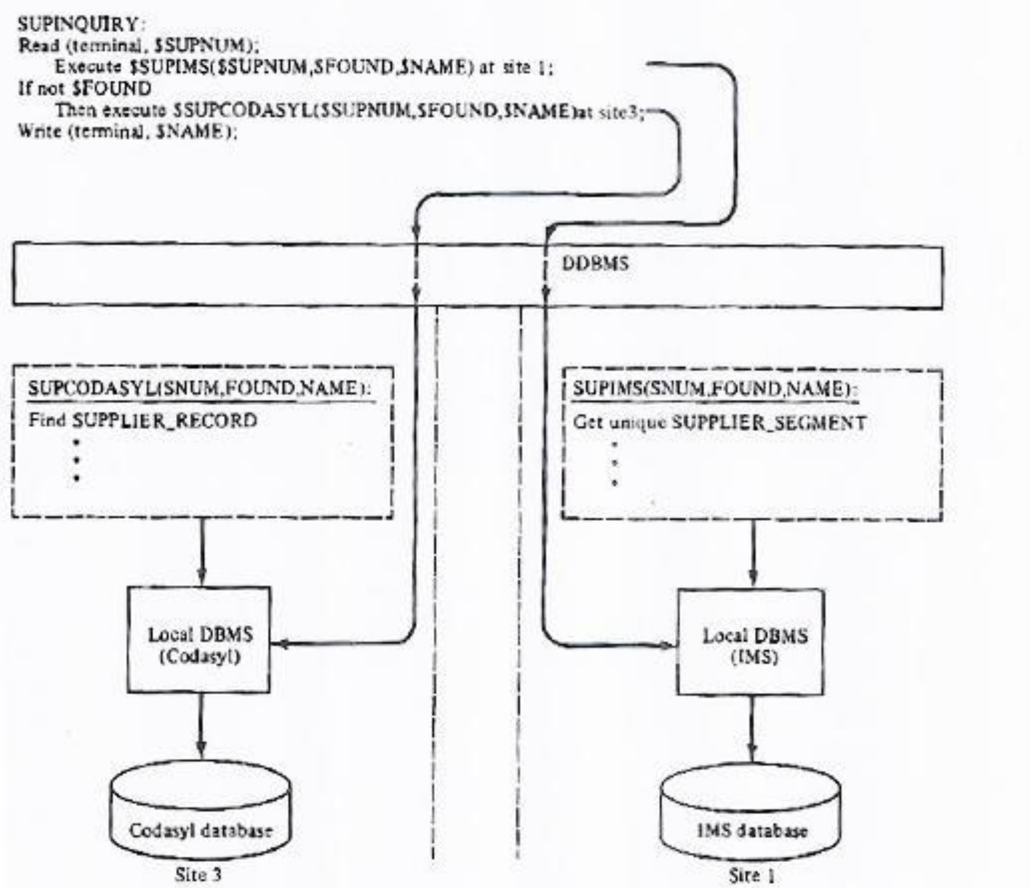


Figure 3.6 An application on a heterogeneous distributed database without transparency.

Complex Application(SUPOFPART) : retrieve name of the supplier who supplies a given part.

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

(a) Fragmentation transparency (level 1)

```
read(terminal, $PNUM);
Select NAME into $NAME
from SUPPLIER, SUPPLY
where SUPPLIER1.SNUM=SUPPLY1.SNUM
and SUPPLY1.PNUM=$PNUM;
if not #FOUND then
          Select NAME into $NAME
          from SUPPLIER2.SUPPLY2
          where SUPPLIER2.SNUM=SUPPLY2.SNUM
             and SUPPLY2.PNUM=$PNUM;
write(terminal, $NAME).
```

(b) Location transparency (level 2)

```
read(terminal, $PNUM);
Select SNUM into $SNUM
from SUPPLY1 at site 3
where PNUM=$PNUM;
if #FOUND then
          begin
              send $SNUM from site 3 to site 1;
              Select NAME into $NAME
              from SUPPLIER1 at site 1
              where SNUM=$SNUM
          end
else begin
              Select SNUM into $SNUM
              from SUPPLY2 at site 4
              where PNUM=$PNUM;
              send $SNUM from site 4 to site 2;
              Select NAME into $NAME
              from SUPPLIER2 at site 2
              where SNUM=$SNUM
          end;
write(terminal, $NAME).
```

(c) Local mapping transparency (level 3)

Figure 3.7 The read-only application SUPOFPART at different levels of distribution transparency.

Distribution transparency for Update Applications

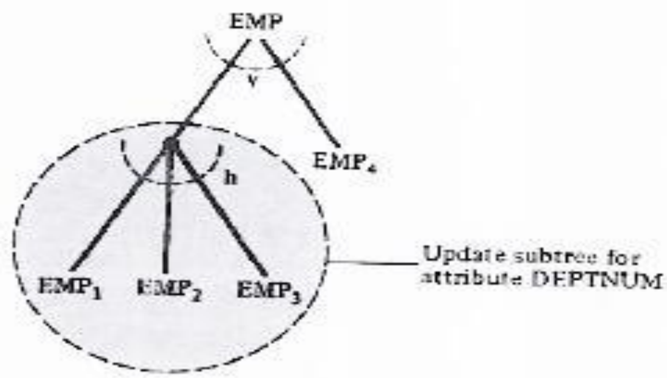
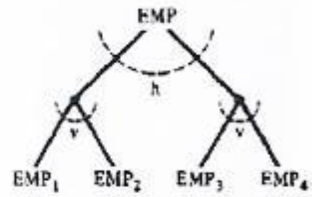


Figure 3.8 The update subtree of the attribute DEPTNUM in the fragmentation tree of relation EMP.



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

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

Before update

(EMP_1):

EMPNUM	NAME	SAL	TAX
100	SMITH	10000	1000

(EMP_2):

EMPNUM	MGRNUM	DEPTNUM
100	20	3

After update

(EMP_3):

EMPNUM	NAME	DEPTNUM
100	SMITH	15

(EMP_4):

EMPNUM	SAL	TAX	MGRNUM
100	10000	1000	20

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

Figure 3.9 An update application.

Update *EMP*
set *DEPTNUM*=15
where *EMPNUM*=100.

(a) Fragmentation transparency (level 1)

Select *NAME, SAL, TAX* into $NAME, $SAL, $TAX
from EMP_1
where *EMPNUM*=100;
Select *MGRNUM* into $MGRNUM
from EMP_2
where *EMPNUM*=100;
Insert into EMP_3 (*EMPNUM, NAME, DEPTNUM*):
(100, $NAME, 15);
Insert into EMP_4 (*EMPNUM, SAL, TAX, MGRNUM*):
(100, $SAL, $TAX, $MGRNUM);
Delete EMP_1 where *EMPNUM*=100;
Delete EMP_2 where *EMPNUM*=100.

(*b*) Location transparency (level 2)

```
Select NAME, SAL, TAX into $NAME, $SAL, $TAX
from EMP1 at site 1
where EMPNUM=100;
Select MGRNUM into $MGRNUM
from EMP2 at site 2
where EMPNUM=100;
Insert into EMP3 (EMPNUM, NAME, DEPTNUM)
        at site 3: (100, $NAME, 15);
Insert into EMP3 (EMPNUM, NAME, DEPTNUM)
        at site 7: (100, $NAME, 15);
Insert into EMP4 (EMPNUM, SAL, TAX, MGRNUM)
        at site 4: (100, $SAL, $TAX, $MGRNUM);
Insert into EMP4 (EMPNUM, SAL, TAX, MGRNUM)
        at site 8: (100, $SAL, $TAX, $MGRNUM);
Delete EMP1 at site 1 where EMPNUM=100;
Delete EMP1 at site 5 where EMPNUM=100;
Delete EMP2 at site 2 where EMPNUM=100;
Delete EMP2 at site 6 where EMPNUM=100.
```

(c) Local mapping transparency (level 3)

Figure 3.10 An update application at different levels of distribution transparency

Distribution Database Access Primitives

•**Language definitions:**

•For DB access Query returns Several values not just 1 as before

•Suffix REL : file by Pascal like & relation by SQL statement

```
Select EMPNUM,NAME into $EMP_REL($EMPNUM, $NAME) from EMP
```

```
repeat
    read(terminal, $SNUM);
    Select PNUM into $PNUM_REL($PNUM)
    from SUPPLY
    where SNUM=$SNUM;
    repeat
        read($PNUM_REL, $PNUM);
        write(terminal, $PNUM)
    until END-OF-$PNUM_REL
until END-OF-TERMINAL-INPUT.
```

(a) The database is accessed for each $SNUM value

```
repeat
    read(terminal, $SNUM);
    write($SNUM_REL($SNUM), $SNUM)
until END-OF-TERMINAL-INPUT;
Select PNUM into $PNUM_REL($PNUM)
from SUPPLY, $SNUM_REL
where SUPPLY.SNUM=$SNUM_REL.$SNUM;
repeat
    read($PNUM_REL, $PNUM);
    write(terminal, $PNUM)
until END-OF-$PNUM_REL.
```

(b) The database is accessed after all the values of $SNUM have been collected

```
Select PNUM, SNUM into $TEMP_REL($TEMP_PNUM, $TEMP_SNUM)
from SUPPLY;
repeat
    read(terminal, $SNUM);
    Select $TEMP_PNUM into $TEMP2_REL($TEMP2_PNUM)
    from $TEMP_REL
    where $TEMP_PNUM=$SNUM;
    repeat
        read($TEMP2_REL, $TEMP2_PNUM);
        write(terminal, $TEMP2_PNUM)
    until END-OF-$TEMP2_REL
until END-OF-TERMINAL-INPUT.
```

(c) The database is accessed before collecting the values of $SNUM

Figure 3.11 Different ways of writing an application with fragmentation transparency.

Integrity constraints in DDBs

Integrity Constraints samples:

- Which data values are allowed (age must be between 0 and 100)
- Which transactions are allowed(age cannot decrease)
- Can involve single or multiple relations
- All values of a given attribute of a relation exist also in some other relation for ensuring correctness of derived fragmentation
- Example

```
Delete *
from SUPPLIER
where SNUM = $SNUM
```

(* indicates the entire tuple). This operation could violate the above referential integrity constraint. In order to verify that the constraint is not violated, it is possible to modify the program as follows:

```
Select $SNUM
from SUPPLY
where SNUM = $SNUM;
if not #FOUND then
        Delete *
        from SUPPLIER
        where SNUM = $SNUM
```