SUBJECT: ADVANCED DATABASES (IV B.TECH II SEM CSE)

UNIT-1

1. DEFINITION OF DISTRIBUTED DATABASE AND INTRODUCTION

A distributed database is a collection of data which belong logically to the same system but are spread over the sites of a computer network.

In a distribution, the data are not residing at the same site, so that we distinguish a distributed database from a single and centralized database

There is **logical correlation** for the data with respect to properties, however we distinguish the distributed database from a set of local databases or files which resides at different sites of the computer network.

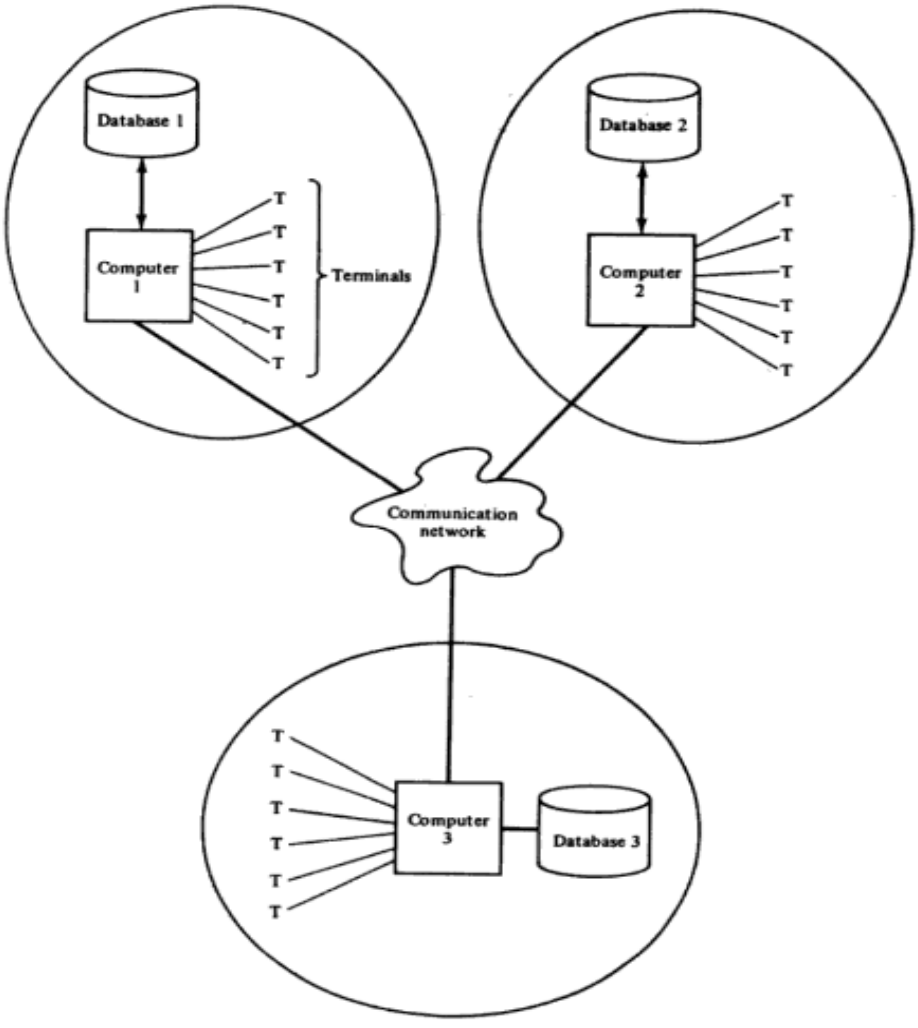


Fig.1: A Distributed Database on a Geographically dispersed network

Structure of distributed database on geographically dispersed network shown in Fig. 1. It illustrates the sample bank processing; bank has three branches and each maintains the separate database. At each branch, we control eight teller terminals and user requested operations are updated on its own distributed database. User operations requested and executed at each branch

called as local applications. Some applications access the data at more than one branch, these are called as **global applications or distributed applications**.

Example:

Typical global application is a transfer of funds from an account of one branch to an account of another branch. This application requires updating the databases at two different branches.

" *A distributed database is a collection of data which are distributed over different computers of a computer network. Each site has its autonomous capability and can perform local applications and they can also participate at least one global application"*

2. FEATURES OF DISTRIBUTED VERSUS CENTRALIZED DATABASES

The following features are analyzed for comparison between distributed databases and centralized databases.

1. Centralized Control
2. Data Independency
3. Reduction of Redundancy
4. Complex Physical Structures and Efficient Access
5. Integrity, recovery, and concurrency control
6. Privacy and Security

1. Centralized Control

Providing centralized control over the information resources of a whole enterprise or organization is considered as strongest motivation for the databases. In databases, each application has its own private files and the database administrator (DBA) has guarantee for safety of the data.

In a distributed database, the feature of 'Centralized Control' is less emphasized. The distributed databases much emphasized to identify the hierarchical control structure based on 'global database administrator', which has the central responsibility of the whole database and the local database administrators. Local database administrators have full control on its respective local databases.

Local database administrators may have a high degree of autonomy; suppose the case that a global database is completely missing, in those cases, intersite coordination is performed by the local administrators. Such characteristic is known as **site autonomy**.

2. Data Independence

Data independency means that actual organization of data is transparent to the application programmer.

Programs are written having conceptual view of the data, called as conceptual schema. Advantage of data independence is that programs are unaffected by changes in the physical organization of data.

In distributed databases, new aspect is added when compared to centralized databases. This aspect is known as 'distribution transparency', in which correctness of the program is

unaffected by the movement of data from one site to another site, however, their speed of execution is affected.

3. Reduction of Redundancy

In traditional centralized databases, 'reduction of redundancy' is considered with following two reasons

a. Inconsistencies among several copies of the same logical data should be avoided by single copy of the data
b. Storage space is saved due to elimination of redundancy data

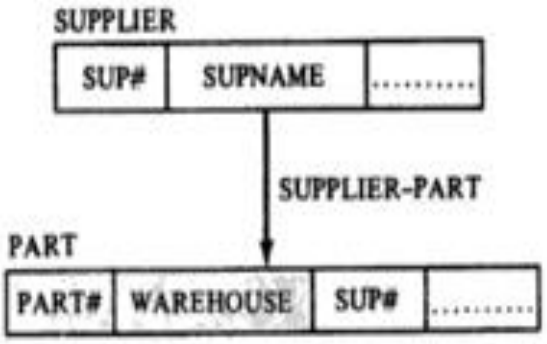
In distributed databases, redundancy of data is considered as desirable feature with following reasons

a. Locality of application is increased if the data is replicated at all sites where the applications need it
b. Availability of the system can be increased (Site failure does not stop the execution of applications at other sites if the data is replicated)

4. Complex Physical Structures and Efficient Access

Complex accessing structures, like secondary indexes, interfile chains etc., are majorly used in traditional databases for efficient access of data in database management system (DBMS).

Complex accessing structures are not the right tool for distributed databases. Thus, efficient access is a major problem in distributed databases. Efficient access to a distributed database cannot be provided by inter-site physical structures. It is not easy to navigate the data at record level in distributed databases.



(*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*) A Codasyl-DBMS-like program for finding parts supplied by supplier S1.

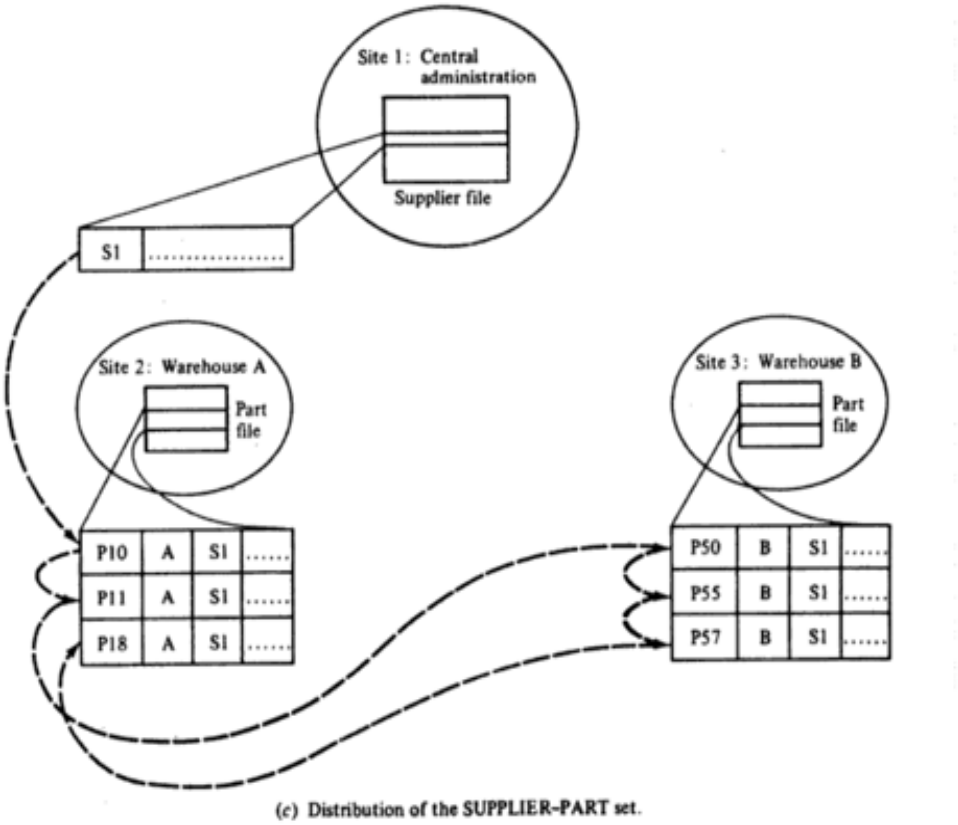


Fig. 2 Distributed Database

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.

Fig. 3 Distributed Access Plan

Two records SUPPLIER and PART are shown in Fig 2a, for the query mentioned in Fig 2b ("Find all PART records supplied by the supplier S1), Fig 2c shows the databases is distributed over three sites as per shown in Fig 2c for getting the information from connected records of SUPPLIER and PART at site1, site2 and site3.

Fig 3 shows the efficient implementation of distributed access plan with specification of two types of operations, the execution of programs which are local at single sites, and the transmission of files between sites. This distributed access plan written by the programmer

itself and it is similar to navigational programming. It is also required to solve two problems with distributed access plan, these are, **global optimization** and **local optimization**.

Global optimization consists of determining which data must be accessed at which sites and which data sites must consequently be transmitted between sites.

Local optimization consists of deciding how to perform the local database access at each site.

5. Integrity, Recovery, and Concurrency Control

The database issues, **integrity, recovery, and concurrency control** are the different problems but they are strongly correlated. A **transaction** of database provides the solution of these problems. A transaction is an **atomic unit of execution**, i.e. it is a sequence of operations which are either entirely performed or not performed at all.

For example, transferring the funds from debit account to credit account, perform either all debit and credit operation from respective accounts or none is performed in respective accounts.

Two dangerous enemies of transaction atomicity are performed, they are failures and concurrency. Failures may cause the interrupting the operations, thus, violating atomicity requirement. Such cases**, recovery** preserve the transaction atomicity problem.

Concurrent execution of different transactions permits one transaction to observe an inconsistent, transient state created by another transaction during its execution.

6. Privacy and Security

In traditional databases, the database administrator having centralized control to ensure only authorized access to the data is performed. The centralized database without specialized control procedures is facing more problem with privacy and security violations. In distributed database, we facing the same issues. However, two aspects are worth mentioning with respect to privacy and security, these are as follows: a) owners of local data that the data is more protected due to high degree of site autonomy b) security problems are intrinsic to distributed systems.

3. WHY DISTRIBUTED DATABASES

The following reasons are motivated for development of distributed databases.

1. Organizational and Economic Reasons
2. Interconnection of Existing Databases
3. Incremental Growth
4. Reduced Communication Overhead
5. Performance Considerations
6. Reliability and Availability

1. Organizational and Economic Reasons

Many organizations are decentralized, so that distributed database approach is fits more naturally the structure of the organization. Organizational structure and economy are considered for developing distributed database

2. Interconnection of Existing Databases

Distributed databases are giving natural solutions when local databases are already existing in an organization and the necessity of performing global applications. In such cases, restructuring of the databases is required and distributed database is created from the preexisting local databases.

3. Incremental Growth

If the organization require addition of data units along with existing data units, such cases, distributed database smoothly performs the addition of required units with minimal degree of the impact for incremental growth of the organization

4. Reduced Communication Overhead

As per the understanding of geographically distributed database, it is clear that many applications are local that reduces the communication overhead with respect to a centralized database. Thus, maximization of locality of the applications is one of the primary objective in distributed database design

5. Performance Considerations

We achieve high degree of parallelism with the design of distributed databased. Scalability issues are solved for improving the performance of distributed database systems when comparing with traditional databases.

6. Reliability and Availability

The distributed database approach with redundant data can be used to obtain higher reliability and availability. Failures are occurred as more in distributed database than centralized database. Failure cause data loss at one site, however, the data is replicate in another site, thus it is available and recovered from data replicated site.

3. REVIEW OF DATABASES

The following concepts are used for describing and manipulating access strategies to distributed databases.

3.1 The Relational Model

In relational databases, data are stored in tables, called **relations**. Each relation has a (fixed) number of columns, called as **attributes** and a number of rows, called **tuples.** There is correspondence between fields of traditional file systems and attributes of relational model, and between records of files and tuples of relations. The number of attributes of a relation called its **grade**; and the number of tuples is called its **cardinality**.

Ex:

Table 1: EMP Relation

EMPNUM	*NAME*	*AGE*	*DEPTNUM*
3	Jones	27	1
7	Smith	34	2
11	Bob	18	1
15	Jane	23	3
18	Mary	31	1

In above example, EMP consisting of four attributes, thus grade is 4. It consists of five tuples, thus cardinality is 5.

The relation name and the names of attributes appearing in it are called the **relation schema** of the relation.

Example of Relation Schema:

EMP (EMPNUM, NAME, AGE, DEPTNUM)

The set of possible values for a given attribute is called its **domain**.

The **relational algebra** is a collection of **operations on relations.** Each of which takes one or two relations as operands and produces one relation as result.

Operations of Relational Algebra

a. Unary Operation: It take only one relation as operand. Selection and Projection are unary operations.

- **i. Selection (SL_F R):** R is the operand to which the selection is applied and F is a formula which expresses a selection predicate, produce a result relation with the same relation schema as the operand relation.
- **ii. Projection (PJ_{Attr} R):** 'Attr' denotes the subset of the attributes of the operand relation, produces a result having these attributes as relation schema. The tuples of the result are derived from the tuples of operand relation schema by suppressing the values of attributes which do not appear in Attr

b. Binary Operations: It take two relations as operands; union, difference, Cartesian product, join, and semi-join are the binary operations

- i. **Union** (R **UN** S): It is meaningful only when R and S, have same relation schema. All tuples are appearing in the resulting schema when the tuples are either appearing in R or S or in both.
- ii. **Difference** (R **DF** S):): It is meaningful only when R and S, have same relation schema; it produces a relation with the same relation schema of operands schema. It results the tuples appearing in R but not in S
- iii. **Cartesian Product** (R **CP** S): Produces a relation whose relation schema includes all the attributes of R and S. Every tuple of R is combined with every tuple of S to form one tuple of the result.
- iv. **Join** of two relations are denoted between R and S as 'R $\mathbf{JN_F}$ S', here F specifies the join predicate formula.

$$R \mathbf{JN_F} S = \mathbf{SL_F} (R \mathbf{CP} S)$$

Thus, relation schema of the result of the join includes all the attributes of R and S, all the tuples from R and S which satisfy the join predicate 'F'.

- v. **Natural Join** R **NJN** S of two relations R and S is an equi-join in which all attributes with the same names in the two relations are compared.

Since these attributes have both the same name and the same values in all the tuples of the result, one of the attributes is omitted from the result.

vi. **Semi-Join** of two relations R and S is denoted as R $\mathbf{SJ_F}$ S, where F is a formula that express a join predicate.

$$R\ \mathbf{SJ_F}\ S = \mathbf{PJ}_{Attr\,(R)}\,(R\ \mathbf{JN_F}\ S)$$

Whrere Attr(R) denotes the set of all attributes of R. Thus, the result of semi-join is the subset of the tuples of R

vii. **Natural Join** of two relations R and S, denoted as R **NSJ** S, is obtained by considering a semi-join with the same join predicate as in the natural join.

R

A	*B*	*C*
a	1	a
b	1	b
a	1	d
b	2	f

S

A	*B*	*C*
a	1	a
a	3	f

T

B	*C*	*D*
1	a	1
3	b	1
3	c	2
1	d	4
2	a	3

(*a*) Operand relations.

A	*B*	*C*
a	1	a
a	1	d

(*b*) Selection $\mathbf{SL}_{A=a}\ R$

A	*B*
a	1
b	1
b	2

(*c*) Projection $\mathbf{PJ}_{A,B}\ R$

A	*B*	*C*
a	1	a
b	1	b
a	1	d
b	2	f
a	3	f

(*d*) Union *R* **UN** *S*

R.A	*R.B*	*R.C*	*S.A*	*S.B*	*S.C*
a	1	a	a	1	a
b	1	b	a	1	a
a	1	d	a	1	a
b	2	f	a	1	a
a	1	a	a	3	f
b	1	b	a	3	f
a	1	d	a	3	f
b	2	f	a	3	f

(*f*) Cartesian Product *R* **CP** *S*

A	*B*	*C*
b	1	b
a	1	d
a	2	f

(*e*) Difference *R* **DF** *S*

A	*R.B*	*R.C*	*T.B*	*T.C*	*D*
a	1	a	1	a	1
a	1	a	2	a	3
b	1	b	3	b	1
a	1	d	1	d	4

(*g*) Join $R\ \mathbf{JN}_{R.C=T.C}\ T$

A	B	C	D
a	1	a	1
a	1	d	4

(*h*) Natural join *R* **NJN** *T*

A	*B*	*C*
a	1	a
b	1	b
a	1	d

(*i*) Semi-join $R\ \mathbf{SJ}_{R.C=T.C}\ T$

A	*B*	*C*
a	1	a
a	1	d

(*j*) Natural join *R* **NSJ** *T*

3.2 Database Applications, Programs, and Transactions

Database Applications: It is a sequence of operations which can be requested by an end-user with a single activation request.

An application can be online, in which it is requested by the user at a terminal and requires the response in a short time. Each application is associated with **application code**, which is used to request its execution.

Online applications can be classified as **simple applications** or **conversational applications. Simple applications** receive one input message and produce one output message. **Conversational applications** exchange several messages with the user.

A **transaction** is an atomic unit database access, which is either completely executed or not executed at all

A program is the definition of computation, including the database access. A program is also considered as a unit having its own address space which communicates with other programs only through messages.

4. DISTRIBUTED DATABASE MANAGEMENT SYSTEMS (DDBMSs)

A distributed database management system supports the creation and maintenance of distributed databases. The following software components are necessary for building distributed database.

1. The database management component (DB)
2. The data communication component (DC)
3. The data dictionary (DD), it can represent the information about the distribution of data in the network
4. The distributed database component (DDB)

The term 'distributed database management system' refer to the above set of components. DDB is the specialized distributed database component.

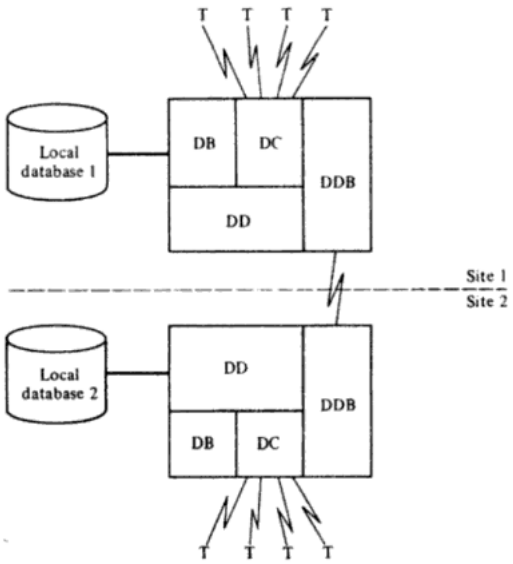


Fig. 4: Components of Distributed Database

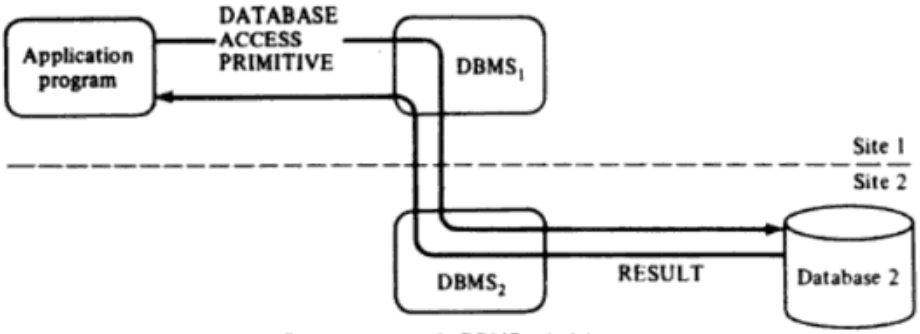
Key services of distributed database system are described as follows:

1. Remote database access by an application program; it is most important feature and provided by all the systems which have a distributed database component.
2. Feature of distributed transparency is supported by different systems
3. Support for the database administration; we use tools for monitoring the database, gather the information about the database utilization, providing global view of data at the various sites.
4. Support for concurrency control and recovery of distributed transactions.

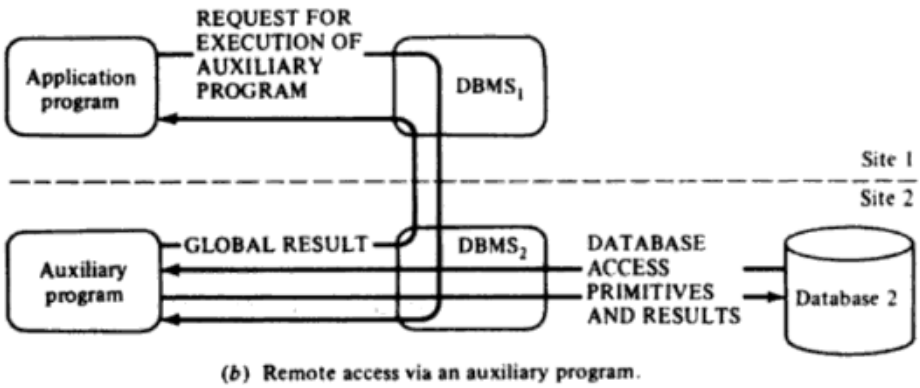
Types of Access in Distributed Databases:

DDBMSs provide two types of remote access

1. Remote Access via DBMS Primitives
2. Remote Access via an Auxiliary Program



(a) Remote access via DBMS primitives.



(b) Remote access via an auxiliary program.

Fig. 5. Types of Access to Distributed Database

Fig. 5a shows that the application issues a database access request which refers to remote data. This request is automatically routed by the DDBMS to data located site. After, request is executed at that site, and the result is returned.

Fig. 5b shows a different approach, in which the application requires the execution of an auxiliary program at the remote site. The auxiliary program written by the application programmer, accesses the remote database and returns the result to the requesting application

5. REVIEW OF COMPUTER NETWORKS

Computers are capable for performing of autonomous work are connected by a communication network. The computers are called **hosts** and **site** denote a host computer.

Communication Network: It is itself constituted of communication links of various kinds, include, coaxial cable, satellite links etc., often includes several computers.

Basic Facility of Computer Network: A process is running at any site can send a message to a process running at any other site of the computer network.

Parameters for Characterizing the Computer Networks:

Delay:

If the network is heavily used, then the delay becomes larger and queuing analysis is required for evaluating the delay.

Cost:

Fixed cost is associated with each message plus an additional cost is defined based on the length of the message.

Reliability:

It is the probability that the message is correctly delivered at its destination.

Model of the computer network is shown in Fig. 6

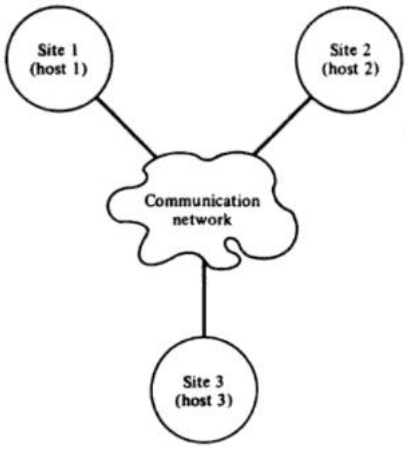


Fig. 6: Model of Computer Network

Types of Communication Networks

1. Point-to-Point or Store-and-Forward Network

The communication network is composed of set of dedicated processors, known as IMPs (interface message processors). Each site is connected to an IMP. When a site sends a message to another IMP, which also stores it and sends it to further IMP, and so on, until the message reaches the destination IMP. This type of network is called as a **point-to-point or store-and-forward** network.

2. Broadcast Network

In a broadcast network, broadcasting a message to many sites is not more expensive than sending a message to one site only. A process at one site send the same message to processes at all other sites; this operation is called as broadcast.

These two types of networks are shown in following Fig. 7a and 7b respectively.

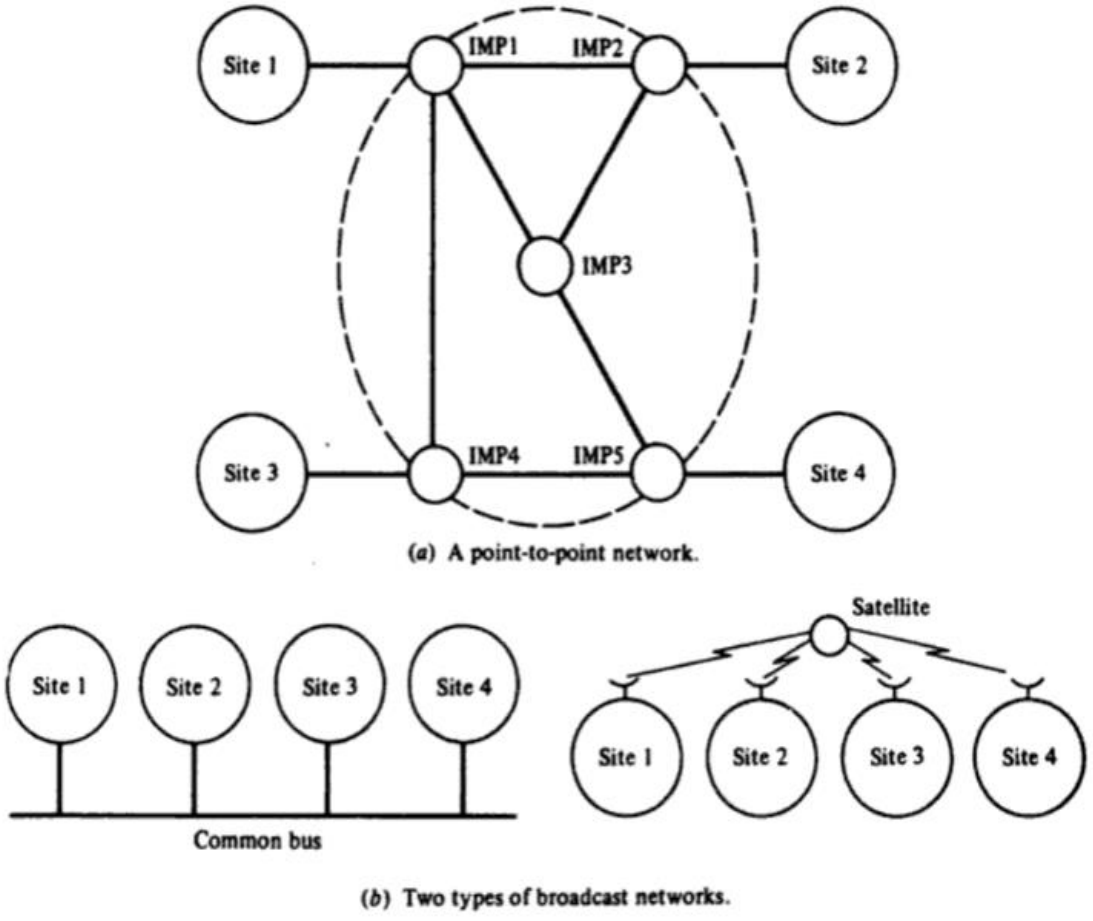


Fig 7: Types of communication Networks

Communication networks are distinguished based on topologies. The basic topologies of star, hierarchical, the ring, and completely connected, which are shown in Fig 8.

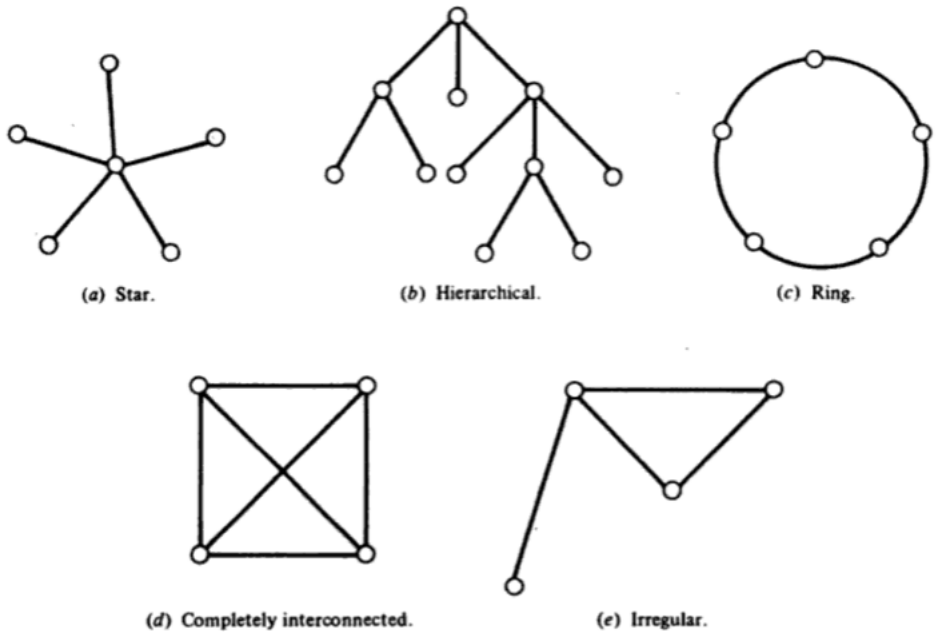


Fig. 8 Network Topologies

Protocols and Sessions

Two processes want to communicate by exchanging messages must follow some rules. The rules which are followed by two (or more) processes for communicating are called a **protocol.**

A session is established between two processes which we want to communicate and is held until all necessary messages have been exchanged.

The ISO/OSI Reference Architecture

(collect the ISO/OSI diagram and concept from CN textbook)

*******End of Unit-1********

UNIT-2

LEVELS OF DISTRIBUTION TRANSPARENCY

2.1 Reference Architecture for Distributed Databases

Reference architecture is not explicitly defined in all the distributed databases; however, its levels are conceptually relevant to organization of any distributed database. Following Fig. 2.1 shows a reference architecture for a distributed database.

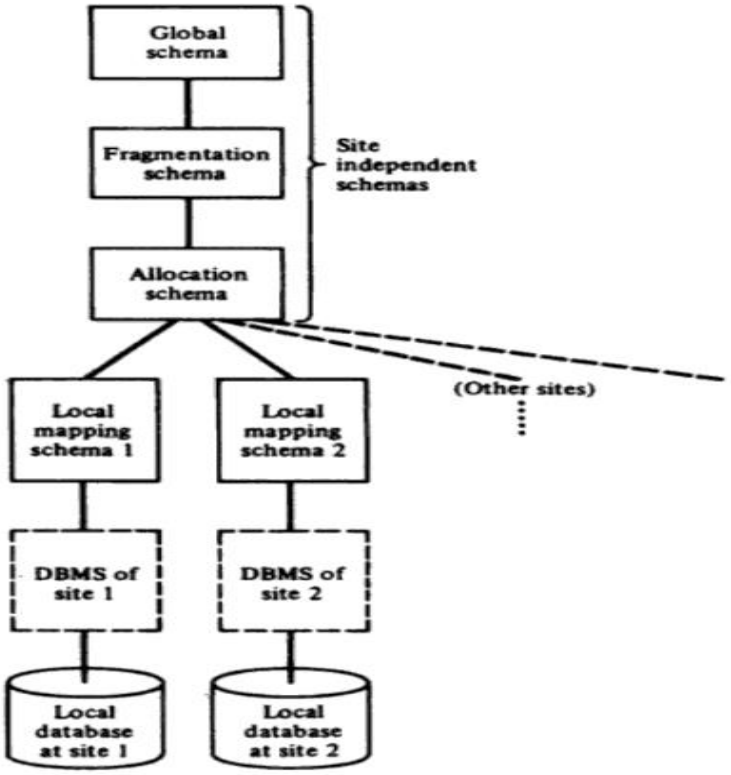


Fig. 2.1: A Reference Architecture for Distributed Databases

The **global schema** defines all the data which are contained in the distributed database as if the database were not distributed at all. We use relational model for defining the data model of the global schema. Global schema consists of the definition of a set of global relations.

Each global schema can be split into several non-overlapping portions are called **fragments.** The relation between global schema and fragments is defined in the **fragmentation schema**. Several fragments are corresponding to a single global relation. Hence, it is one to many. Fragments are indicated by a global relation name with an index (fragment index); For example, R_i indicates the i^{th} fragment of global relation R.

Allocation schema defines at which site(s) a fragment is located. **Type of mapping** in the allocation schema determines the whether the distributed database is redundant or non-redundant. In the redundant case, type of mapping is one to many, whereas as if it is non-redundant, then type of mapping is one to one.

All the fragment which corresponds to same global relation R and are located at the same site j constitute physical image of global relation at site j.

Example is shown in Fig 2.2, in which global relation is split into four fragment R_1 ,R_2 , R_3 , and R_4 . These four fragments are allocated redundantly at three sites of the computer network and building images are R^1, R^2, R^3. Copy of the fragment is denoted by R_i^j , i.e., copy of the fragment R_i is located at site j.

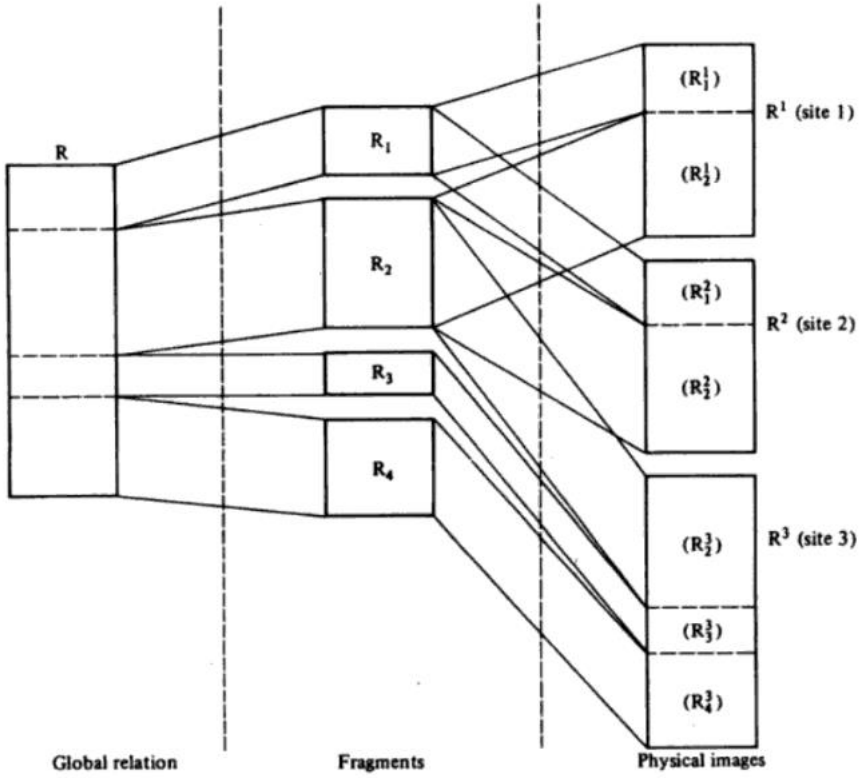


Fig. 2.2 Fragments of Physical Relation for a Global Relation

At lower level, it is necessary to map the physical images to the objects which are manipulated by the local DBMS. This mapping is called as local mapping schema and depends on the type of local DBMS.

Following three most important objectives are motivating the features of this architecture are data fragmentation and allocation, control of redundancy, and independency from local DBMS.

1. Separating the concept of data fragmentation from the concept of data allocation – this separation allows us to distinguish two different levels of distribution transparency, namely **fragmentation transparency** and **location transparency**. Fragmentation transparency is the highest degree of transparency for the fact that the user or application programs works on global relations. Location transparency is the lower degree of transparency for the fact of the user or applications to work on fragments instead of global relations.

2. **Explicit control of redundancy** The reference architecture provides the explicit control of redundancy at fragments level. For example, Fig. 2.2 shows that two physical images R^2 . and R^3 are overlapping. i.e., they contain common data.
3. **Independence from local DBMS** This feature called **local mapping transparency** , allows to study several problems of distributed database management without having to take account the specific models of local DBMSs.

2. 2 Types of Data Fragmentation

Decomposition of global relations into fragments can be performed by applying two different types of fragmentation: **horizontal fragmentation** and **vertical fragmentation.** A fragment can be expressed by a relational language. Following rules are applied during the fragmentation.

1. Completeness condition
2. Reconstruction condition
3. Disjoint condition

In completeness condition, all the data of global relation must be mapped into fragments

In reconstruction method, it must always possible to reconstruct the global relation from its fragments.

In disjoint condition, the fragments must be disjoint. So that the replication of the data can be controlled explicitly at the allocation level.

2.2.1 Horizontal Fragmentation

Horizontal fragmentation consists of partitioning the tuples of a global relation into subsets. It can be defined by the 'select' operation on the global relation

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

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

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

The above fragmentation satisfies the completeness condition if "SF" and "LA" are the only possible values of the *CITY* attribute; otherwise we would not know to which fragment the tuples with other *CITY* values belong.

The reconstruction condition is easily verified, because it is always possible to reconstruct the *SUPPLIER* global relation through the following operation:

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

The disjoint property is also clearly verified.

We will call the predicate which is used in the selection operation which defines a fragment its **qualification.**

Example:

q_1 : $CITY$ = "SF"
q_2 : $CITY$ = "LA"

2.2.2 Derived Horizontal Fragmentation

In some cases, the horizontal fragmentation of a relation cannot be based on a property of its own attributes, but is derived from the horizontal fragmentation of another relation.

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

where *SNUM* is a supplier number. It is meaningful to partition this relation so that a fragment contains the tuples for suppliers which are in a given city. However, city is not an attribute of the *SUPPLY* relation, it is an attribute of the *SUPPLIER* relation considered in the above example. Therefore we need a semi-join operation in order to determine the tuples of *SUPPLY* which correspond to the suppliers in a given city. The derived fragmentation of *SUPPLY* can be therefore defined as follows:

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

The effect of the semi-join operations is to select from *SUPPLY* the tuples which satisfy the join condition between $SUPPLIER_1$ or $SUPPLIER_2$ and *SUPPLY*, thus determining those tuples of *SUPPLY* which refer to suppliers in San Francisco or Los Angeles, respectively.

2.2.3 Vertical Fragmentation

The vertical fragmentation of a global relation is the sub-division of its attributes into groups; fragments are obtained by projecting the global relation over each group.

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

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

because *EMPNUM* is a key of *EMP*. In general, the inclusion of a key of the global relation into each fragment is the most straightforward way to guarantee that the reconstruction through a join operation is possible. An alternative way to provide the reconstruction property is to generate **tuple identifiers** which are used as system-controlled keys. This can be convenient in order to avoid the replication of large keys; moreover, tuple identifiers cannot be modified by users.

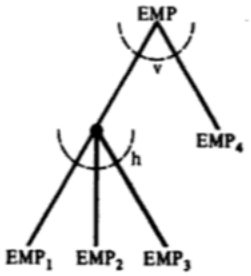
2.2.4 Mixed Fragmentation

We use both select and project operations as per defined horizontal and vertical fragmentations in mixed fragmentation. The reconstruction can be obtained by applying the reconstruction rules in inverse order.

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*:

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



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

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

Mixed fragmentation can be conveniently represented by a **fragmentation tree**. In a fragmentation tree, the root corresponds to a global relation, the leaves correspond to the fragments, and the intermediate nodes correspond to the intermediate results of the fragment-defining expressions. The set of nodes which are sons of a given node represent the decomposition of this node by a fragmentation operation (vertical or horizontal). For example, Figure 3.3 shows the fragmentation tree of relation *EMP*. The root (relation *EMP*) is vertically fragmented into two portions: one portion corresponds to a leaf node of the tree (EMP_4); the other portion is horizontally partitioned, thus generating the other three leaves, corresponding to fragments EMP_1, EMP_2, and EMP_3.

The EXAMPLE_DDB

Figure 3.4 shows the global and fragmentation schemata of EXAMPLE_DDB which will be used in the rest of this book for the development of examples. Most of the global relations of EXAMPLE_DDB and their fragmentation have been already introduced. A *DEPT* relation, horizontally fragmented into three fragments on the value of the *DEPTNUM* attribute, is added. The features of EXAMPLE_DDB will be discussed when they will be used to exemplify specific topics. Note, as a

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

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

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

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

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

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

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

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

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

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

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

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

2.3 DISTRIBUTION TRANSPARENCY FOR READ-ONLY APPLICATIONS

We analyze the different levels of distribution transparency that can be provided by distributed database management system (DDBMS)

3.3.1 A simple application

A simple application: ‘Supplier Inquiry’ :

“find the supplier name for given supplier number from the query system”

Query:

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

In this query analysis, input parameters of an SQL query appear in the where-clause and output parameters are appeared in the select clause. Thus, in a procedural language, $SNUM received as input and $NAME takes the output.

Different levels of distribution transparency is shown in 2.3, in which the following transparencies are achieved

1. Fragmentation transparency
2. Location transparency
3. Replication Transparency

Fragmentation transparency is the highest degree of transparency and consists of the fact that the user or application programmer works on global relation

Location transparency is a lower degree of transparency and requires the user or programmer to work on fragments instead of global relations

Replication transparency means that the user is unaware of the replications of fragments.

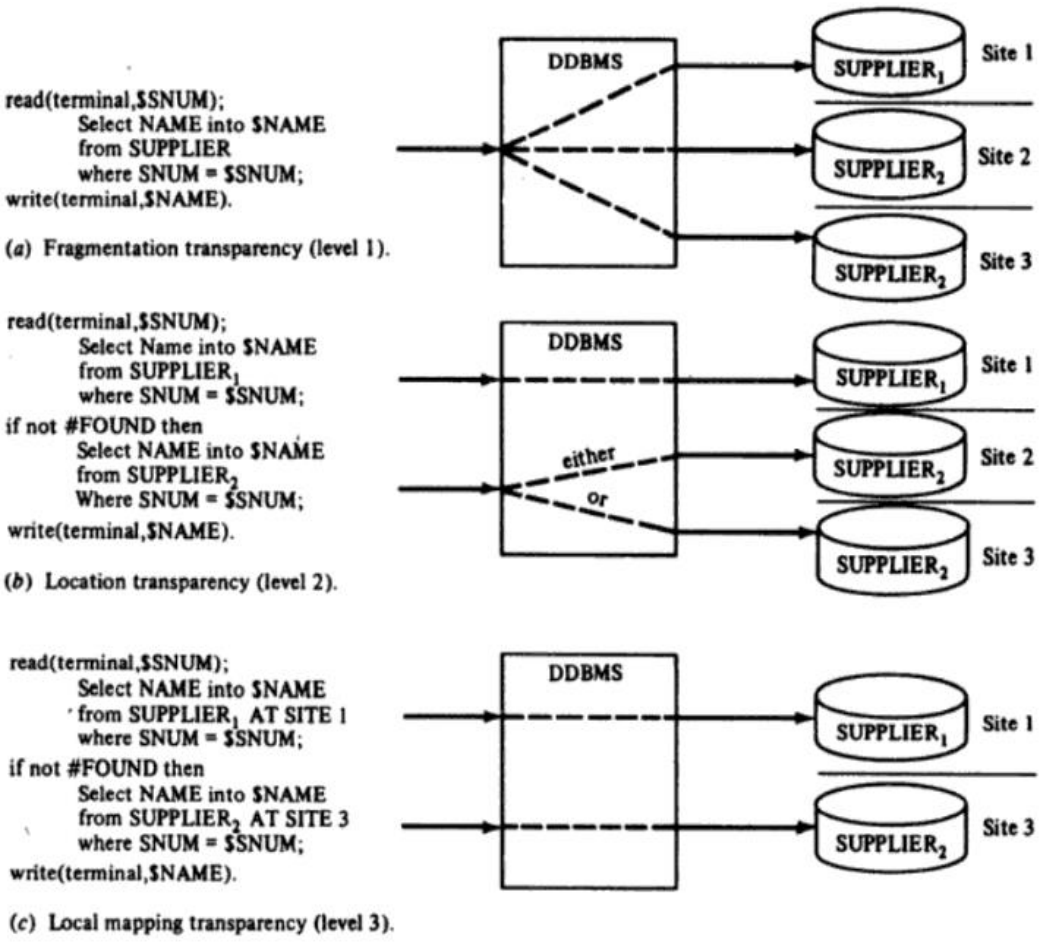


Fig. 2.3 Different levels of distribution transparency.

Level 1: Fragmentation Transparency:

It is shown in Fig. 2.3a, first it reads the supplier number from the terminal, then it accesses the database. The specified SQL statement represents a single distributed database access primitive, which receives the variable $SNUM as input parameter and returns the variable $NAME as output parameter. The DDBMS access this primitive at any one of the three sites.

Level 2: Location Transparency:

Location transparency provided by DDBMS is shown in Fig. 2.3b, in which request for supplier number is issued for the fragment SUPPLIER 1, if the DDBMS returns as 'not FOUND', same request is issued to another fragment SUPPLIER 2.

Level 3: Local Mapping Transparency:

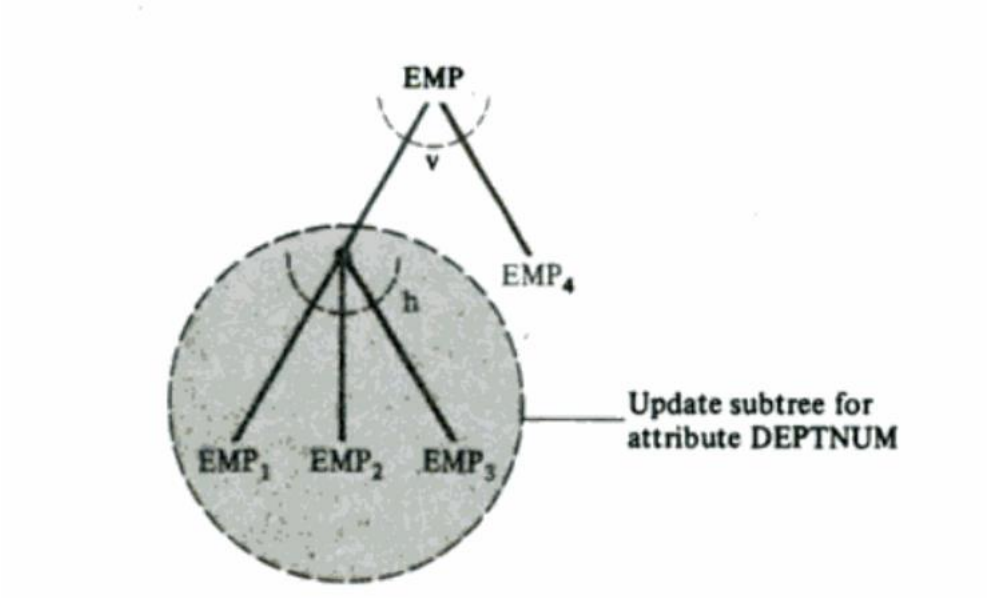
At this level, we assume that the application still refers to objects using names. It is shown in Fig. 2.3c. The site names are indicated in the SQL statements "at clause" to the "from clause".

2.4 DISTRIBUTION TRANSPARENCY FOR UPDATE APPLICATIONS

Same levels of distribution transparency is considered for read only applications and also for update applications. Update must be performed on all copies of the data; retrieval can be performed on single copy of the data. It means that DDBMS cannot provide location and replication transparency for update applications, application programmer has the responsibility for providing these transparencies for update applications.

Example:

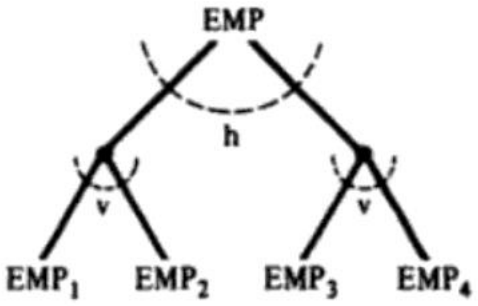
Suppose CITY attribute of a supplier is modified and clearly knows that supplier tuple is moved from one fragment to another. The tuples of SUPPLY relation which refers to the same supplier also must be change fragment, because the SUPPLY relation has a derived fragmentation. Following figure shows the more understanding of data movement operations for updating of a fragmentation attribute.



Consider some attribute A of EMP is used to select predicate of a horizontal fragmentation. The update subtree of A is the subtree having as root node representing the above horizontal fragmentation.

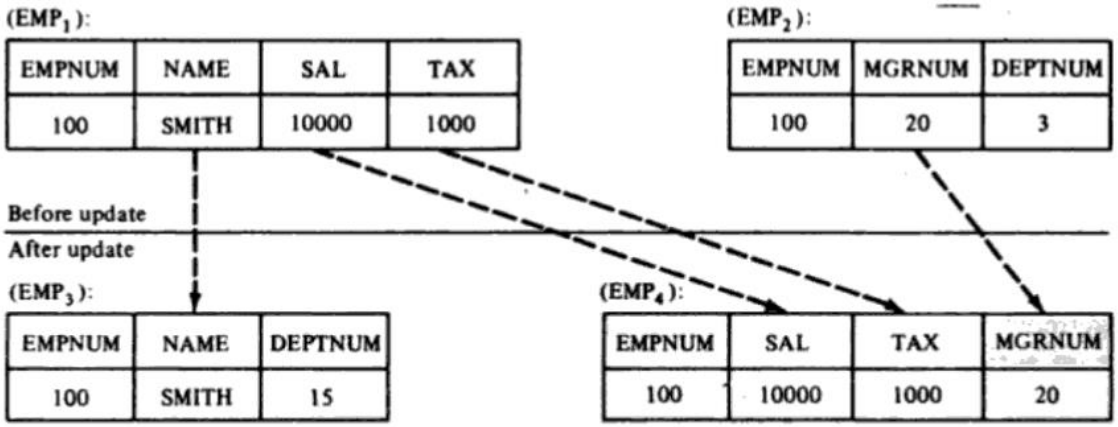
It also shows the update subtree for attribute DEPTNUM. Change of attributes reflects only leave of its update subtree. A change of attribute DEPTNUM affects only EMP_1, EMP_2, and EMP_3. A tuple can migrate (change fragments) between two of the three former fragments because of the update.

Fig. 2.4 shows an example of update application, in which DEPTNUM is the same as whole segmentation tree. It shows the effect of changing the value of DEPTNUM from 3 to 15 for the EMPNUM =100. Its updates can be performed by following transparencies.



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

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



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

Fig. 2.4 An Update Application

Level 1: Fragmentation Transparency:

At this level, the application programmer must perform UPDATE operation by writing following query in Level 1-Fragmentation Transparency.

The query is shown as follows:

Update EMP

Set DEPTNUM = 15

Where EMPNUM = 100

Level 2 : Location Transparency

At this level, the application programmer must deal explicitly with the fragments. Example of location transparency is shown as follows

```
Select NAME, SAL, TAX into $NAME, $SAL, $TAX
from EMP1
where EMPNUM=100;
Select MGRNUM into $MGRNUM
from EMP2
where EMPNUM=100;
Insert into EMP3 (EMPNUM, NAME, DEPTNUM):
        (100, $NAME, 15);
Insert into EMP4 (EMPNUM, SAL, TAX, MGRNUM):
        (100, $SAL, $TAX, $MGRNUM);
Delete EMP1 where EMPNUM=100;
Delete EMP2 where EMPNUM=100.
```

First two select statements are used for collecting the where they are stored (possibly in EMP_1 and EMP_2). Third and Fourth statements shows that updating of DEPTNUM = 15 for EMPNUM =100 and then deletes the old tuples.

Level 3: Location Mapping Transparency

At this level, the application programmer must deal explicitly with location of the fragments, in which updates must be performed for the account of replication. Assume the fragments of relations are allocated in the following way:

EMP_1 : sites 1 and 5

EMP_2 : sites 2 and 6

EMP_3 : sites 3 and 7

EMP_4 : sites 4 and 8

The following queries are performed by application programmer for achieving of location mapping transparency.

```
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.
```

UNIT-3

1. A FRAMEWORK FOR DISTRIBUTED DATABASE

We apply following steps in the framework of distributed database

1. Designing the "conceptual schema" – it describes the integrated database, in which all the data used by database applications
2. Designing the "physical database" – it maps the conceptual schema to different storage areas.
3. Designing the "fragmentation" – it determines how the global schema are divided into horizontal, vertical, or mixed fragmentation
4. Designing the "allocation of fragments" – determine how fragments are mapped to physical images

First two steps show the design of the global schema and design of the local databases at each site in distributed databases. Step 3 and 4 shows the characterization of design of data distribution, i.e. in detailed fragmentation of global relation and physical placement of the data at various sites.

Application requirements may also influence the schema design. Application requirements are included as follows:

1. The site from which the application is issued, know as **site of origin** of application
2. The frequency of activation of the application
3. The number, type, and the statistical information of data objects of application

Objectives of the Design of Data Distribution

1. Processing Locality

Place the data near to the application site where the application is to run is known as processing locality. We maximize the processing locality by increasing the local references and remote references to each candidate fragmentation.

Applications are completely executed at their sites of origin is known as **complete locality**.

2. Availability and Reliability of Distributed Data

A high degree of availability for read-only applications achieved by storing multiple copies of the same information.

Reliability is also achieved by storing multiple copies of the same information for recovering of data during either crashes or physical destruction

3. Workload Distribution

Distributing the workload over the sites is an important feature of distributed computing systems. It maximizes the degree of parallelism of execution of applications.

4. Storage costs and availability

Database distribution should reflect the cost and availability of storage at the different sites. It is possible to have specialized sites in the network for data storage.

Top-Down and Bottom-Up Approaches to the Design of Data Distribution

Top-Down Approach

In the top-down approach, we start the designing the global schema and we proceed by designing the fragmentation of the database, and then allocating the fragments to the sites, creating the physical images.

(explain the horizontal and vertical fragmentation also)

Bottom-Up Approach

When the distributed database is developed as the aggregation of existing databases is known as bottom-up approach. This approach is based on the **integration** of existing schemata into a single, global schema.

The bottom-up design of a distributed database requires the following developments

1. The **selection** of a **common database model** for describing the global schema of the database
2. The **translation** of each local schema into the common data model
3. The **integration** of the local schemata into a common global schema

2. THE DESIGN OF DATABASE FRAGMENTATION

The purpose of fragmentation design is to determine "non-overlapping fragments", which are "logical units of allocation".

Designing of fragmentation consists of grouping tuples as fragments in horizontal fragmentation and grouping the attributes as fragments in vertical fragmentation.

Each group of tuples or attributes having the "same properties", constitute as fragments.

Example:

Consider the horizontal fragmentation for a global relation EMP.

Distributed database application: Determine employees are members of projects from EMP

Let each department be a site of the distributed database and application can be issued at any department. Existing employees are distributed in different departments.

Each project is made in one department. In our application, we expect the members of project are in the same department. Thus, tuples of each department are collected by determining the employees who work at the same department.

Vertical fragmentation for relation EMP

Assume that attributes SAL and TAX are used only by administrative applications, which always use these attributes together. Thus, pair of SAL and TAX are showing appropriate vertical fragmentation.

2.1 Horizontal Fragmentation

Two types of horizontal fragmentation are called as primary and derived. Derived fragmentation is derived in terms of primary fragmentation.

Horizontal fragmentation deals with following

1. Logical properties of the data – fragmentation predicates

2. Statistical properties of the data – define number of references of applications to fragments

2.1.1 Primary Fragmentation

Primary horizontal fragments are defined using selections on global relations, in which each tuple can be selected in one and only one fragments. Determining the primary fragmentation of a global relation requires a set of disjoint and complete selection predicates. Therefore, we use following definitions:

1. A simple predicate is a type of the predicate
   Attribute = value
2. A minterm predicate y for a set of P of simple predicates is the conjunction of all predicates appearing in P
   $y = \bigwedge_{p_i \in p} p_i^*$, where ($p_i^* = p_i$ or $p_i^* = NOT\ p_i$) and y≠false
3. A fragment is the set of all tuples for which a minterm predicate holds.

Example:

Determine the employee's information who are the members of projects and also require the data of employees who are the programmers

Let assume that there are only two departments 1 and 2 and required project members are in DEPT = 1 only.

DEPT = 1 is same as DEPT ≠ 2 and also vice-versa

Simple predicate of give example is DEPT =1 and JOB = "P"

Possible Minterm Predicates are

DEPT =1 and JOB = "P"

DEPT =1 and JOB ≠ "P"

DEPT ≠ 1 and JOB = "P"

DEPT ≠ 1 and JOB ≠ "P"

Let P = {p1, p2, ……pn} be a set of simple predicates. If fragmentation is efficient and correct if and only if P must be complete and minimal.

Complete - we say the predicate P is **complete** if and only if any two tuples belong to same fragment are referenced with same fragment by any application

Minimal – we say that P is **minimal** if all its predicates are relevant

Example

P1={DEPT = 1} is not complete

P2 = {DEPT = 1, JOB = "P"} is complete and minimal

P3 = {DEPT = 1, JOB = “P”, SAL >50 } is complete but not minimal

General Example 1:

Consider the distributed database for a company in California having three sites at San Francisco (site 1), Fresno (site 2), and Los Angeles (site 3). Fresno is located between site 1 and site 3. There are 30 departments, in which first 10 are close to site 1, 11 to 20 are close to site 2, and 21 to 30 are close to site 3.

Global schema : EXAMPLE_DDB

Other Relations : EMP, DEPT, SUPPLIER, SUPPLY

Database Application : Names of suppliers with a given SNUM for relation SUPPLIER

(SNUM, NAME, CITY)

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

This distributed application is issued at any site.

If it is issued at site 1, it references SUPPLIERS whose CITY is San Francisco with 80% probability

If it is issued at site 2, it references SUPPLIER of San Francisco (SF) and Loss Angeles (LA) with equal probability

If it is issued at site 3, it references SUPPLIER of Loss Angeles (LA) with 80% probability.

We can apply this application for producing the possible simple predicates p1 and p2 as per follows

p1: CITY = “SF”

p2: CITY =”LA”

the set {p1, p2} is now complete and minimal.

General Example 2:

Global Relation : DEPT (DEPTNO, NAME, AREA, MGRNUM)

Database Application : Administrative application is issued at sites 1 and 3 for the respective departments

Possible set of Predicates :

p1: DEPTNO ≤ 10

p2 : 10 < DEPTNO ≤ 10

p3 : DEPTNO > 20

p4 : AREA = “North”

P5 : AREA = “South”

There are implications between AREA and DEPTNO, thus reduced fragmentation is defined as follows:

y1: DEPTNO ≤ 10

y2 : (10 < DEPTNO) ≤ 10 and (AREA=”North”)

y3 : (10 < DEPTNO) ≤ 10 and (AREA=”South”)

y4 : DEPTNO > 20

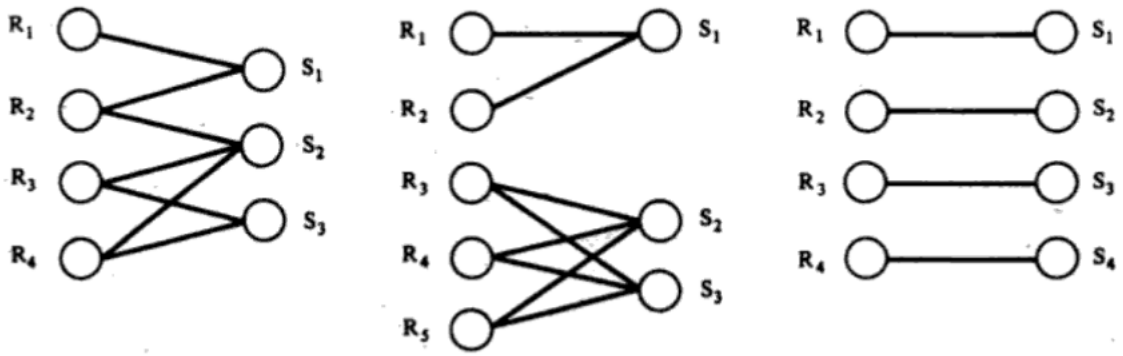
Therefore, final fragmentation of relation DEPT is shown in following table.

	p4: AREA = “North”	p5: AREA = “South”
p1: DEPTNO ≤ 10	y1	FALSE
p2 : 10 < DEPTNO ≤ 10	y2	y3
p3 : DEPTNO > 20	FALSE	y4

2.1.2 Derived Horizontal Fragmentation

The derived horizontal fragmentation of a global relation R is not based n properties of its own attributes, but it is derived from the horizontal fragmentation of another result. Derive fragmentation is used to facilitate join between fragments.

A **distributed join** is a join between horizontally fragmented relations. A distributed join is represented using **join graphs.** The join graph G of the distributed join R JN S is a graph < N, E>, where nodes N represent fragments of R and S and non-directed edges between nodes represent joins between fragments. The join graphs are shown in following figure.



(a) Join graph. (b) Partitioned join graph. (c) Simple join graph.

2.1.3 Vertical Fragmentation

Determining the vertical fragmentation of a global relation R requires grouping the attributes as set of fragments. The following approaches are used for vertical fragmentation

1. **The split approach**- in which global relations are progressively split into fragments
2. **The grouping approach** – in which attributes are progressively aggregating to constitute fragments.

In above cases, formulas are used to indicate which is the best splitting or grouping

General Example

Global Relation : EMP (EMPNO, NAME, SAL, TAX, MGRNUM, DEPTNO)

Vertical Fragmentation Results:

EMP1(EMPNO, NAME, SAL, TAX)

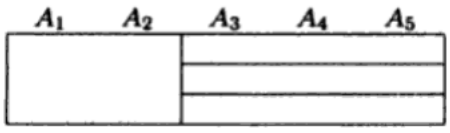
EMP2(EMPNO, MGRNUM, DEPTNO)

2.1.4 Mixed Fragmentation

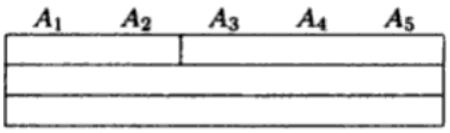
The mixed fragmentation consists of :

1. Applying horizontal fragmentation to vertical fragments
2. Applying vertical fragmentation to horizontal fragments

The following figures shows the mixed fragmentation



(a) Vertical fragmentation followed by horizontal fragmentation



(b) Horizontal fragmentation followed by vertical fragmentation

UNIT-4

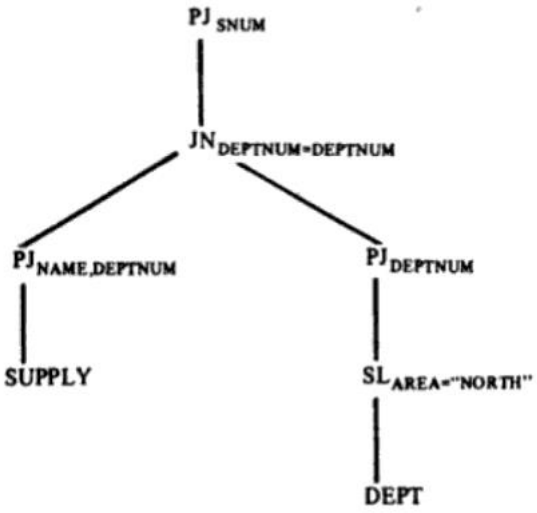
4.1 TRANSFORMING GLOBAL QUERIES INTO FRAGMENT QUERIES

A. Canonical Expression of a Fragment Query

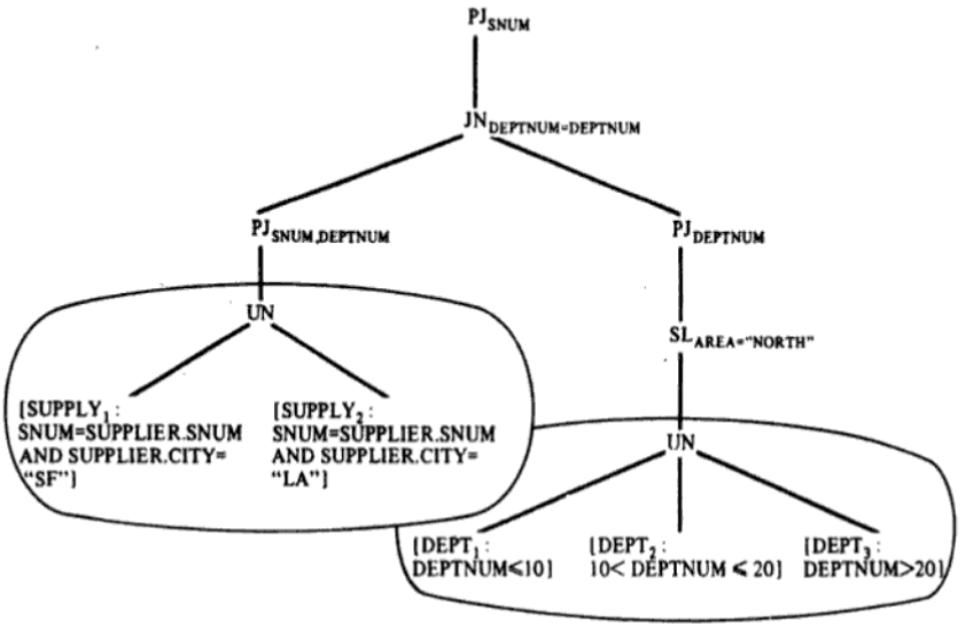
A canonical expression for the algebraic expression over the global schema is obtained as follows:

"Reconstruction of global relation is obtained from fragments. We map an operator tree on the global schema, in which collecting the fragments of all global relations into temporary relations and then applying to these temporaries the global query. "

Example: Operator tree of Q1 as follows



The following figure shows the canonical expression of above tree



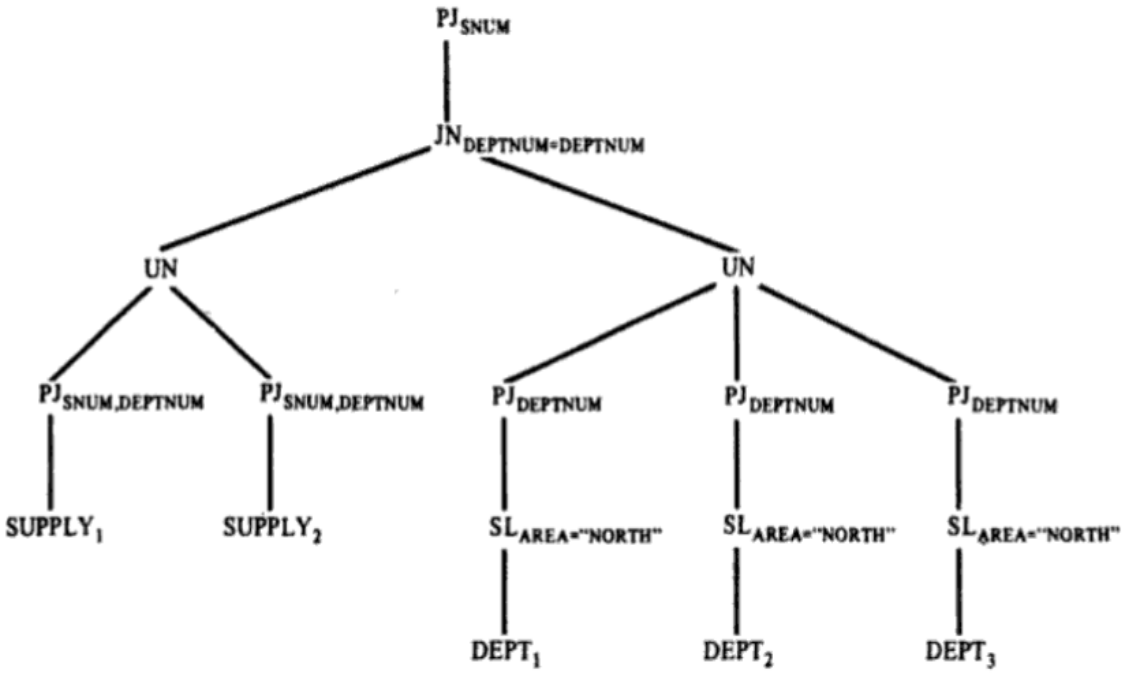
(*a*) Canonical form of query Q1.

We apply following criteria 1 and criteria 2 for further simplification of above canonical expression.

Criterion 1 : Use Idempotence of selection and projection to generate appropriate selections and projections for each operand relation

Criterion 2 : Push selections and projections down in the tree as far as possible.

After applying these two criterions, the resulting operator are described as follows:



(*b*) Pushing selections and projections down in the operator tree.

B. Algebra of Qualified Relations

A qualified relation is a relation extended by a qualification and the qualification possessed by all the tuples of the relation.

Qualified relation can be denoted by a pair **[R :q_R],** where **R** is a relation called the **body of the qualified relation and q_R is a predicate called the qualification of the qualified relation. Horizontal fragments are** typical examples of **qualified relations**.

The following rules define the result of applying the operations of relational algebra to qualified relations:

Rule 1 $$\mathbf{SL}_F[R : q_R] \Rightarrow [\mathbf{SL}_F R : F \text{ AND } q_R]$$

This rule states that the application of a selection $\mathbf{SL}_F$ to a qualified relation $[R : q_R]$ produces a qualified relation having the relation $\mathbf{SL}_F R$ as its body and the predicate F AND q_R as its qualification. The extension of the qualification to F AND q_R after a selection reflects the fact that F holds on all the selected tuples as well as q_R.

Rule 2 $$\mathbf{PJ}_A[R : q_R] \Rightarrow [\mathbf{PJ}_A R : q_R]$$

The qualification of the result of a projection remains unchanged, even if the projection eliminates some of the attributes upon which the qualification was evaluated.

Rule 3 $[R : q_R]\ \mathbf{CP}\ [S : q_S] \Rightarrow [R\ \mathbf{CP}\ S : q_R\ \text{AND}\ q_S]$

The extension of the qualification to q_R AND q_S is rather intuitive; notice that the two qualifications apply to disjoint attributes of $R\ \mathbf{CP}\ S$.

Rule 4 $[R : q_R]\ \mathbf{DF}\ [S : q_S] \Rightarrow [R\ \mathbf{DF}\ S : q_R]$

However, the following problem arises with rule 4: consider the intersection operation of traditional relational algebra, defined as $R\ \mathbf{IN}\ S = R\ \mathbf{DF}\ (R\ \mathbf{DF}\ S)$. From the definition, it is easy to show that intersection is commutative; i.e.,

$$R\ \mathbf{IN}\ S = S\ \mathbf{IN}\ R.$$

If we apply the definition of extended algebra, we come to a surprising result:

$$\begin{aligned}&[R : q_R]\ \mathbf{IN}\ [S : q_S] \Rightarrow [R : q_R]\ \mathbf{DF}\ ([R : q_R]\ \mathbf{DF}\ [S : q_S]) \Rightarrow\\ &[R : q_R]\ \mathbf{DF}\ [R\ \mathbf{DF}\ S : q_R] \Rightarrow [R\ \mathbf{DF}\ (R\ \mathbf{DF}\ S) : q_R] \Rightarrow\\ &[R\ \mathbf{IN}\ S : q_R]\end{aligned} \tag{5.4a}$$

$$\begin{aligned}&[S : q_S]\ \mathbf{IN}\ [R : q_R] \Rightarrow [S : q_S]\ \mathbf{DF}\ ([S : q_S]\ \mathbf{DF}\ [R : q_R]) \Rightarrow\\ &[S : q_S]\ \mathbf{DF}\ [S\ \mathbf{DF}\ R : q_S] \Rightarrow [S\ \mathbf{DF}\ (S\ \mathbf{DF}\ R) : q_S] \Rightarrow\\ &[S\ \mathbf{IN}\ R : q_S]\end{aligned} \tag{5.4b}$$

In fact, the result that we would like to find is

$$[R\ \mathbf{IN}\ S : q_R\ \text{AND}\ q_S]$$

Rule 5 $[R : q_R]\ \mathbf{UN}\ [S : q_S] \Rightarrow [R\ \mathbf{UN}\ S : q_R\ \text{OR}\ q_S]$

The union is extended by taking the disjunction of qualifications.

Given the above rules, let us see how derived operations of algebra, such as join and semi-joins, are extended. We have two additional rules:

Rule 6 $[R : q_R]\ \mathbf{JN}_F\ [S : q_S] \Rightarrow [R\ \mathbf{JN}_F\ S : q_R\ \text{AND}\ q_S\ \text{AND}\ F]$

Rule 7 $[R : q_R]\ \mathbf{SJ}_F\ [S : q_S] \Rightarrow [R\ \mathbf{SJ}_F\ S : q_R\ \text{AND}\ q_S\ \text{AND}\ F]$

We give following two criterion 3 and 4 based on qualification of relations in further simplification.

Criterion 3. Push selections down to the leaves of the tree, and then apply them using the algebra of qualified relations; substitute the selection result with the empty relation if the qualification of the result is contradictory.

Criterion 4. Use the algebra of qualified relations to evaluate the qualification of operands of joins; substitute the subtree, including the join and its operands, with the empty relation if the qualification of the result of the join is contradictory.

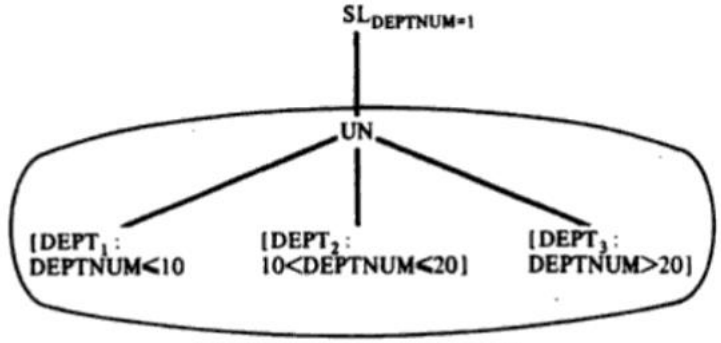
These two criterions are used in simplification of horizontally fragmented relations and joins between horizontally fragmented relations.

C. Simplification of Horizontally Fragmented Relations

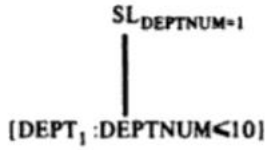
We show this kind of simplification with an example. Let us consider query Q3 on relation *DEPT*, which is horizontally fragmented:

$$Q3 : \mathbf{SL}_{DEPTNUM=1}\ DEPT$$

The canonical form of the query is shown in Figure 5.5a. We push the selection toward the leaves by distributing it with respect to union. Then we apply selections according to criterion 3; the qualification of results of the selections on $DEPT_2$ and $DEPT_3$ is a contradiction;



(a) Canonical form of query Q3.



(b) Simplification of query Q3.

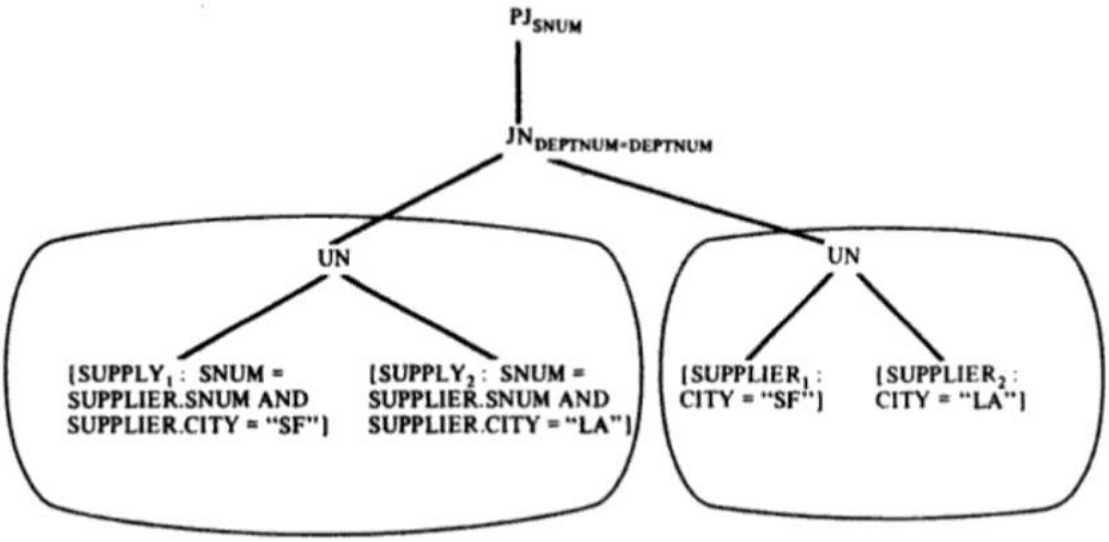
D. Simplification of Joins between Horizontally Fragmented Relations

Let us consider, for simplicity, the join between two fragmented relations R and S. There are two distinct possibilities of joining them; the first one requires collecting all the fragments of R and S before performing the join. The second one consists of performing the join between fragments and then collecting all the results into the same result relation; we refer to this second case as "distributed join."

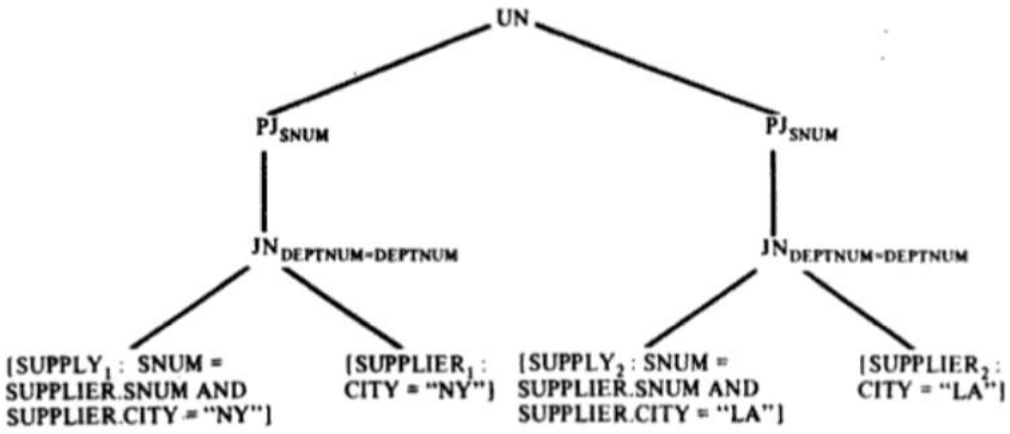
Criterion 5. In order to distribute joins which appear in the global query, unions (representing fragment collections) must be pushed up, beyond the joins that we want to distribute.

Building a join graph requires, then, applying criterion 5 (for distributing the join) followed by criterion 4

Figure 5.6a shows the canonical form of the query. We recall that the fragmentation of *SUPPLY* is derived from the fragmentation of *SUPPLIER*; i.e., each tuple of *SUPPLY* is stored either in fragment $SUPPLY_1$, if it refers to a supplier of San Francisco, or in fragment $SUPPLY_2$, if it refers to a supplier of Los Angeles. Applying criterion 5, we push the two unions up beyond the join; thus, we generate four joins between fragments. We then apply criterion 4, and we discover that two of them are intrinsically empty because their qualification is contradictory. The empty joins are those of $SUPPLIER_1$ (in "SF") with $SUPPLY_2$ (of "LA" suppliers), and likewise of $SUPPLIER_2$ (in "LA") with $SUPPLY_1$ (of "SF" suppliers). Thus, the operator tree reduces to that of Figure 5.6*b*. Assuming that fragments with the same index are placed at the same site (i.e., that data about *SUPPLY* are stored together with data about *SUPPLIERS*), this operator tree corresponds to an efficient way of evaluating the query, because each join is local to one site.



(*a*) Canonical form of query Q4.



(*b*) Distributed join for query Q4.

Figure 5.6 Simplification of joins between horizontally fragmented relations.

E. Using Inference for Further Simplifications

Assume that the following knowledge is available to the query optimizer:

1. The North area includes only departments 1 to 10.
2. Orders from departments 1 to 10 are all addressed to suppliers of San Francisco.

We use the above knowledge to "infer" contradictions that allow eliminating subexpressions.

a. From 1 above, we can write the following implications:

$$AREA = \text{"North"} \Rightarrow \text{NOT}\,(10 < DEPTNUM \leq 20)$$
$$AREA = \text{"North"} \Rightarrow \text{NOT}\,(DEPTNUM > 20)$$

Using criterion 3, we apply the selection to fragments $DEPT_1$, $DEPT_2$, and $DEPT_3$ and evaluate the qualification of the results. By virtue of the above implications, two of them are contradictory. This allows us to eliminate the subexpressions for fragments $DEPT_2$ and $DEPT_3$. Thus, the operator tree of Figure 5.4*b* reduces to that of Figure 5.7a.

b. We then apply criterion 5 for distributing the join; in principle, we would need to join the subtree including $DEPT_1$ with both subtrees including $SUPPLY_1$ and $SUPPLY_2$. But from 1 above, we know that:

to join the subtree including $DEPT_1$ with both subtrees including $SUPPLY_1$ and $SUPPLY_2$. But from 1 above, we know that:

$$AREA = \text{"North"} \Rightarrow DEPTNUM \leq 10$$

and from 2 above we know that:

$$DEPTNUM \leq 10 \Rightarrow$$
$$\text{NOT}\,(SNUM = SUPPLIER.SNUM \text{ AND } SUPPLIER.CITY = \text{"LA"})$$

By applying criterion 4, it is possible to deduce that only the subtree including $SUPPLY_1$ needs to be joined. The final operator tree for query Q1 is shown in Figure 5.7*b*.

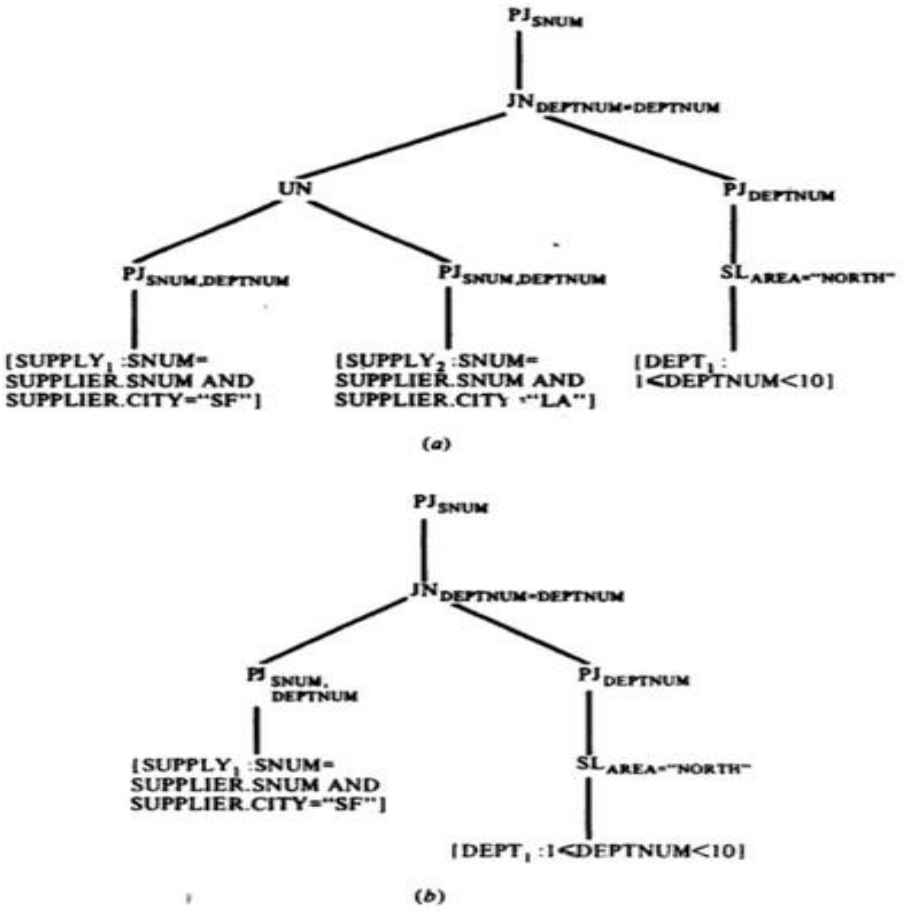
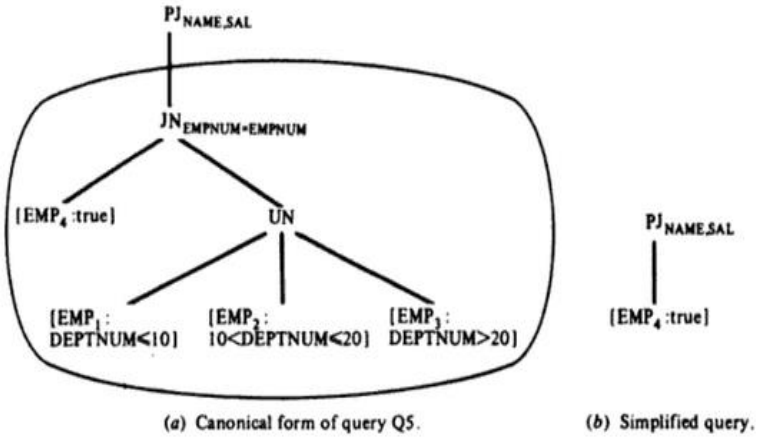


Figure 5.7 Simplification of an operator tree using inference.

F. Simplification of Vertically Fragmented Relations

Simplification is to determine a proper subset of the fragments which is sufficient for answering the query, then eliminate all other fragments from the query expression, as well as the joins which are used in the inverse of the fragmentation schema for reconstructing global relations.

For the following query, simplification of vertical fragmented relations are as follows:



(a) Canonical form of query Q5. (b) Simplified query.

Figure 5.8 Simplification of vertically fragmented relations.

4.2 EQUIVALENCE TRANSFORMATIONS FOR QUERIES:

It is possible to transform most SQL queries into equivalent expressions of relational algebra.

4.2.1 Operator Tree of a query:

In order to have a more practical representation of queries, in which expression manipulation is easier to follow, we introduce **operator trees**. Let us consider

query Q1, which requires the supplier number of suppliers that have issued a supply order in the North area of our company. Query Q1 corresponds to the following expression of the relational algebra:

$$Q1 : \mathbf{PJ}_{SNUM}\,\mathbf{SL}_{AREA=\text{"North"}}(SUPPLY\,\mathbf{JN}_{DEPTNUM=DEPTNUM}\,DEPT)$$

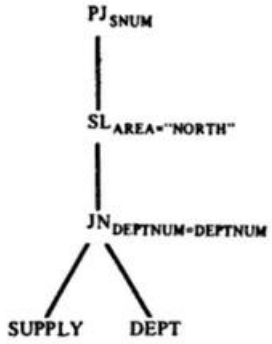


Figure 5.1 An operator tree for query Q1.

The operator tree of an expression of relational algebra can be regarded as the parse tree of the expression itself, assuming the following grammar:

$R \rightarrow$ identifier
$R \rightarrow (R)$
$R \rightarrow un\_op\ R$
$R \rightarrow R\ bin\_op\ R$
$un\_op \rightarrow \mathbf{SL}_F \mid \mathbf{PJ}_A$
$bin\_op \rightarrow \mathbf{CP} \mid \mathbf{UN} \mid \mathbf{DF} \mid \mathbf{JN}_F \mid \mathbf{NJN}_F \mid \mathbf{SJ}_F \mid \mathbf{NSJ}_F$

4.2.2 Equivalent transformations for the Relational Algebra

Equivalence transformations can be given systematically for small expressions, i.e., expressions of two or three operand relations. These transformations are classified into categories according to the type of the operators involved. Let U and B denote unary and binary algebraic operations, respectively. We have:

- **Commutativity** of unary operations:

$$U_1\ U_2\ R \leftrightarrow U_2\ U_1\ R$$

- **Commutativity** of operands of binary operations:

$$R\,B\,S \leftrightarrow S\,B\,R$$

- **Associativity** of binary operations:

$$R\,B\,(S\,B\,T) \leftrightarrow (R\,B\,S)\,B\,T$$

- **Idempotence** of unary operations:

$$U\,R \leftrightarrow U_1\,U_2 R$$

- **Distributivity** of unary operations with respect to binary operations:

$$U(R\,B\,S) \rightarrow U(R)\,B\,U(S)$$

- **Factorization** of unary operations (this transformation is the inverse of distributivity):

$$U(R)\,B\,U(S) \rightarrow U(R\,B\,S)$$

The tables contain in each position a **validity indicator**. A validity indicator "*Y*" means that the property can always be applied; "*N*" means that it cannot be applied. For example, the validity indicator "*Y*" in the first row and first column of Table 5.1 means that the following transformation is correct

$$\mathbf{SL}_{F1}\,\mathbf{SL}_{F2}R \rightarrow \mathbf{SL}_{F2}\mathbf{SL}_{F1}R$$

where $F1$ and $F2$ are two generic selection specifications.

Validity indicators can also be "*SNC*," specifying a condition which is necessary and sufficient for the application of the property. For example, the validity indicator SNC_1 in the second row and first column of Table 5.1 means that the transformation

$$\mathbf{PJ}_{A1}\mathbf{SL}_{F2}R \rightarrow \mathbf{SL}_{F2}\mathbf{PJ}_{A1}R$$

Table 5.1 Commutativity of unary operations

	$\mathbf{SL}_{F2}$	$\mathbf{PJ}_{A2}$
$\mathbf{SL}_{F1}(*(R)) \rightarrow *(\mathbf{SL}_{F1}(R))$	*Y*	*Y*
$\mathbf{PJ}_{A1}(*(R)) \rightarrow *(\mathbf{PJ}_{A1}(R))$	SNC_1	SNC_2

SNC_1 : $Attr(F2) \subseteq A1$

SNC_2 : $A1 \equiv A2$

Table 5.2 Commutativity of operands and associativity of binary operations

	UN	DF	CP	JN_F	SJ_F
$R * S \rightarrow S * R$	Y	N	Y	Y	N
$(R * S) * T \rightarrow R * (S * T)$	Y	N	Y	SNC_1	N

SNC_1 for $(R \text{ JN}_{F1} S) \text{JN}_{F2} T \rightarrow R \text{ JN}_{F1} (S \text{ JN}_{F2} T)$: $Attr(F2) \subseteq Attr(S) \cup Attr(T)$

Table 5.3 Idempotence of unary operations

$\text{PJ}_A(R) \rightarrow \text{PJ}_{A1} \text{PJ}_{A2}(R)$	$SNC : A \equiv A1, A \subseteq A2$
$\text{SL}_F(R) \rightarrow \text{SL}_{F1} \text{SL}_{F2}(R)$	$SNC : F = F1 \wedge F2$

In nondistributed databases, general criteria have been given for applying equivalence transformations for the purpose of simplifying the execution of queries:

Criterion 1. Use idempotence of selection and projection to generate appropriate selections and projections for each operand relation.

Criterion 2. Push selections and projections down in the tree as far as possible.

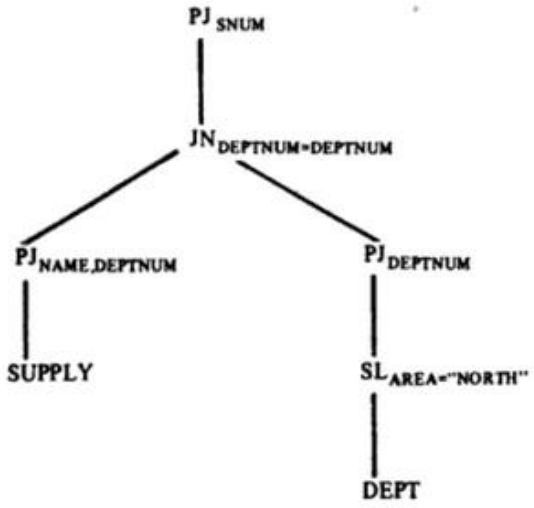


Figure 5.2 A modified operator tree for query Q1.

Figure 5.2 shows a modified operator tree for query Q1, in which the following transformations have been applied:

1. The selection is distributed with respect to the join; thus, the selection is applied directly to the *DEPT* relation.
2. Two new projection operations are generated and are distributed with respect to the join.

Notice that criterion 1 (use of idempotence) has been applied to the projection operation before distributing it; the selection operation, instead, has been directly moved to the *DEPT* relation.

4.2.3 Operator Graph and Determination of Common Sub-Expressions

An important issue in query expression is to discover its common sub-expressions; i.e., sub-expressions which appear more than once in the query; it shows that, saving the time and space if common sub-expressions are evaluated only once.

Ex:

$$Q2 : \mathbf{PJ}_{EMP.NAME}((EMP\ \mathbf{JN}_{DEPTNUM=DEPTNUM}\ \mathbf{SL}_{MGRNUM=373}DEPT)\mathbf{DF}\ (\mathbf{SL}_{SAL>35000}EMP\ \mathbf{JN}_{DEPTNUM=DEPTNUM}\mathbf{SL}_{MGRNUM=373}DEPT))$$

In this, common subexpression is as follows:

$$EMP\ \mathbf{JN}_{DEPTNUM=DEPTNUM}\ \mathbf{SL}_{MGRNUM=373}\ DEPT$$

Once common subexpressions are identified, we can use the following properties to further simplify an operator tree:

$$R\ \mathbf{NJN}\ R \leftrightarrow R$$
$$R\ \mathbf{UN}\ R \leftrightarrow R$$
$$R\ \mathbf{DF}\ R \leftrightarrow \emptyset$$

$$R\ \mathbf{NJN}\ \mathbf{SL}_F\ R \leftrightarrow \mathbf{SL}_F\ R$$
$$R\ \mathbf{UN}\ \mathbf{SL}_F\ R \leftrightarrow R$$
$$R\ \mathbf{DF}\ \mathbf{SL}_F\ R \leftrightarrow \mathbf{SL}_{\mathrm{NOT}\ F}\ R$$

$$(\mathbf{SL}_{F_1}\ R)\ \mathbf{NJN}\ (\mathbf{SL}_{F_2}\ R) \leftrightarrow \mathbf{SL}_{F_1\ \mathrm{AND}\ F_2}\ R$$
$$(\mathbf{SL}_{F_1}\ R)\ \mathbf{UN}\ (\mathbf{SL}_{F_2}\ R) \leftrightarrow \mathbf{SL}_{F_1\ \mathrm{OR}\ F_2}\ R$$
$$(\mathbf{SL}_{F_1}\ R)\ \mathbf{DF}\ (\mathbf{SL}_{F_2}\ R) \leftrightarrow \mathbf{SL}_{F_1\ \mathrm{AND\ NOT}\ F_2}\ R$$

4.3 DISTRIBUTED GROUPING AND AGGREGATE FUNCTION EVALUATION

Database applications having database access operations that can be expressed in query languages rather than specifying with relational algebra. Grouping tuples into disjoint subsets of relations and evaluating aggregate functions over them

We first give some examples, using SQL, of queries which use the above features.

Q6: Select AVG(*QUAN*)
from *SUPPLY*
where *PNUM* = "P1"

This query retrieves into a result relation having one attribute and one tuple the average quantity of supply orders for product "P1."

Q7: Select *PNUM*, *SNUM*, SUM(*QUAN*)
from *SUPPLY*
group by *SNUM*, *PNUM*

This query corresponds to partitioning the relation *SUPPLY* into groups having the same value of *SNUM* and *PNUM* (but different values of *DEPTNUM* and *QUAN*), evaluating for each such group the sum of the quantities, and retrieving *SNUM*, *PNUM*, and the sum of the quantities of each group into the result relation.

Q8: Select *SNUM*, *PNUM*, SUM(*QUAN*)
from *SUPPLY*
group by *SNUM*, *PNUM*
having SUM(*QUAN*) > 300

Extensions with Relational Algebra

Relational algebra is extended with the following **group-by** $\mathbf{GB}_{G,AF}\ R$ such that:

- G are the attributes which determine the grouping of R.
- AF are aggregate functions to be evaluated on each group.
- $\mathbf{GB}_{G,AF}R$ is a relation having:
  A relation schema made by the attributes of G and the aggregate functions of AF.

With the above operation, it is possible to write in algebra queries Q6, Q7, and Q8. We have

$$\text{Q6} : \mathbf{GB}_{\text{AVG}(QUAN)}\ \mathbf{SL}_{PNUM=\text{"P1"}}\ SUPPLY$$
$$\text{Q7} : \mathbf{GB}_{SNUM,PNUM,\text{SUM}(QUAN)}\ SUPPLY$$
$$\text{Q8} : \mathbf{SL}_{\text{SUM}(QUANT)>300}\ \mathbf{GB}_{SNUM,PNUM,\text{SUM}(QUAN)}\ SUPPLY$$

Some comments are in order. The G part corresponds to the "group-by" clause, and the AF part corresponds to the aggregate functions whose computation is required. Typically in these queries attributes upon which grouping is made are also retrieved; thus, the attributes which appear in the "group-by" clause also appear in the "select" clause.

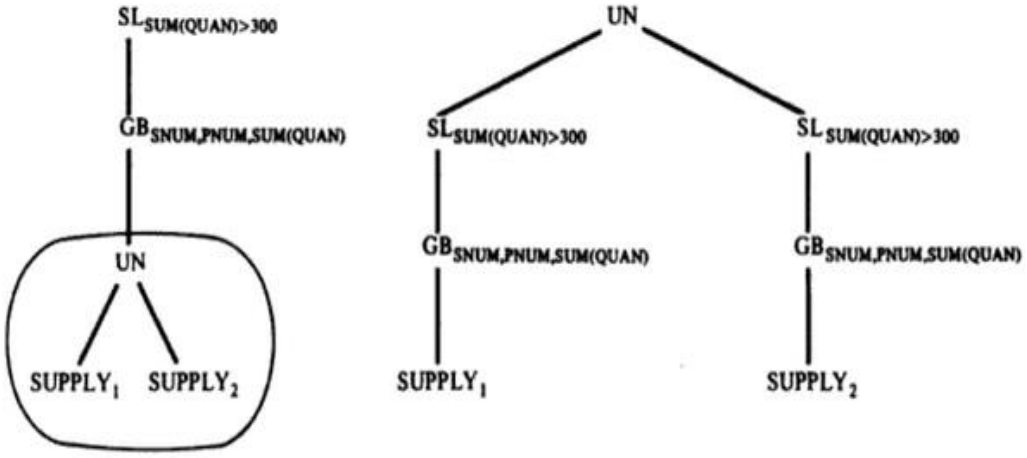
In query Q6, the G part is left empty, as the function is applied to all the tuples of *SUPPLY*. In query Q8, the usual selection operation is applied to the result of the **GB** operation; this selection corresponds to the "having" clause of SQL.

Properties of Group By Operations:

In this section, we give an equivalence property for the new operation, and we discuss, in general, the possibility of evaluating aggregate functions in a distributed way.

The property in which we are interested is the distributivity of **GB** with respect to union:

$$\mathbf{GB}_{G,AF}(R_1\ \mathbf{UN}\ R_2) \rightarrow (\mathbf{GB}_{G,AF}R_1)\mathbf{UN}(\mathbf{GB}_{G,AF}R_2), \qquad \text{Val. ind.: } SNC$$



(*a*) Canonical form of query Q8.

(*b*) Distributed version of query Q8.

Figure 5.10 A query with grouping and aggregate functions.

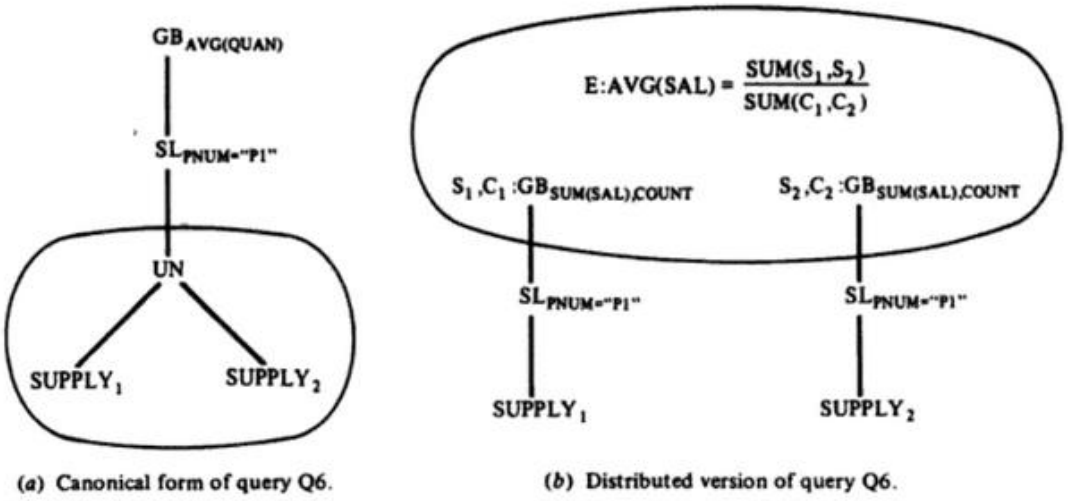
Criterion 6. In order to distribute grouping and aggregate function evaluations appearing in a global query, unions (representing fragment collections) must be pushed up, beyond the corresponding group-by operation.

An aggregate function for which it is possible to find the functions F_i and the expression $E(F_i)$ is the function average

$$\text{AVG}(S) = \frac{\text{SUM}(\text{SUM}(S_1), \text{SUM}(S_2), \ldots, \text{SUM}(S_n))}{\text{SUM}(\text{COUNT}(S_1), \text{COUNT}(S_2), \ldots, \text{COUNT}(S_n))}$$

Similarly, we have

$$\text{MIN}(S) = \text{MIN}(\text{MIN}(S_1), \text{MIN}(S_2), \ldots, \text{MIN}(S_n))$$
$$\text{MAX}(S) = \text{MAX}(\text{MAX}(S_1), \text{MAX}(S_2), \ldots, \text{MAX}(S_n))$$
$$\text{COUNT}(S) = \text{SUM}(\text{COUNT}(S_1), \text{COUNT}(S_2), \ldots, \text{COUNT}(S_n))$$
$$\text{SUM}(S) = \text{SUM}(\text{SUM}(S_1), \text{SUM}(S_2), \ldots, \text{SUM}(S_n))$$



(a) Canonical form of query Q6. (b) Distributed version of query Q6.

Figure 5.11 Distributed evaluation of aggregate functions.

4.4 PARAMETRIC QUERIES

The selection criteria of queries include parameters whose values are not known when the query is compiled. When parametric queries are executed, the user provides values which are bound to parameters, thus parametric queries allow the repeated execution of queries for different values of the parameters and return different values at each execution.

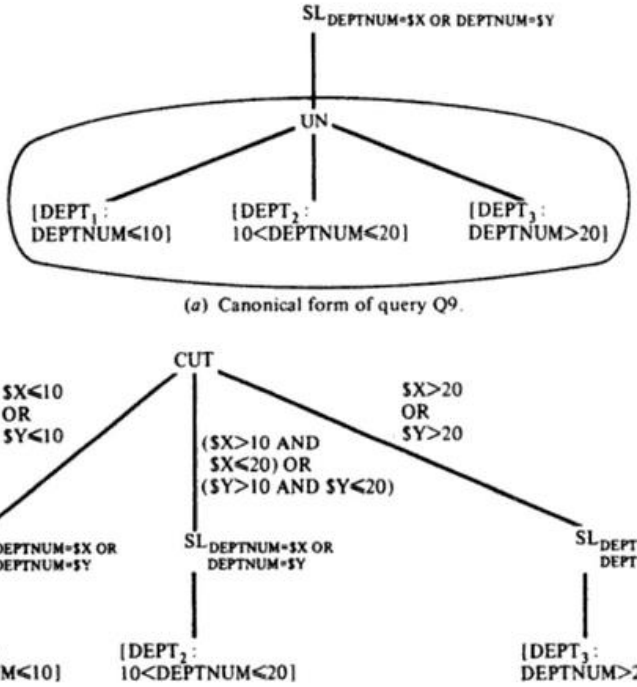
4.4.1 Simplification of Parametric queries and extension of algebra

As an example of a parametric query, let us consider query Q9, selecting tuples of the global relation *DEPT* having given department numbers. Let the selection on *DEPTNUM* be parametric:

$$Q9 : SL_{DEPTNUM=\$X \text{ OR } DEPTNUM=\$Y}\ SUPPLY$$

At run time, the actual values are assigned to the parameters $X and $Y by the program which issues the query.

The simplification of parametric queries has some distinguishing features. Consider the above query, whose canonical form is shown in Figure 5.12a.



(*a*) Canonical form of query Q9.

(*b*) Query tree with the CUT operation.

Figure 5.12 Use of CUT in a parametric query.

time, we don't know which of the fragments of the global relation *DEPT* will be addressed. However, we know that *at most two* of them will be involved in the query. At run time, when the query will be activated, it will also be possible to know, given the value of the parameters, which ones of the fragments will be involved in the query. This shows that part of the simplification of a parametric query can be done at compile time, but part of it needs to be done at run time.

Figure 5.12*b* shows the new operator tree for the parametric query. In this example, the selection is pushed below the union operation, applying criterion 2, and then the union is substituted by the **CUT** operation. The three formulas for the **CUT** operation (one for each operand of the union operation) are

$$F1 : \$X \leq 10 \text{ OR } \$Y \leq 10$$
$$F2 : (\$X > 10 \text{ AND } \$X \leq 20) \text{ OR } (\$Y > 10 \text{ AND } \$Y \leq 20)$$
$$F3 : \$X > 20 \text{ OR } \$Y > 20$$

At run time, the **CUT** operation is the first one to be evaluated; the query, for $\$X = 1$ and $\$Y = 13$, reduces to the first and second branch of the operator tree; for $\$X = 1$ and $\$Y = 5$, the query reduces to the first branch of the tree.

4.4.2 Using temporaries in Multiple Activations of Parametric Queries

For the case of repeated executions and lower the costs, it might be useful to build temporary relations at the site of origin of the query, which store a superset of the data required at each iteration.

Q10 : $\mathbf{PJ}_{NAME}\,\mathbf{SL}_{MGRNUM=\$X\ \text{AND}\ DEPTNUM=12}\ EMP$

First, we modify the query so that the operator tree of Figure 5.13a is obtained. The general goal of this transformation is to insulate subtrees which are **not parametric**, as they will constitute the required temporaries.

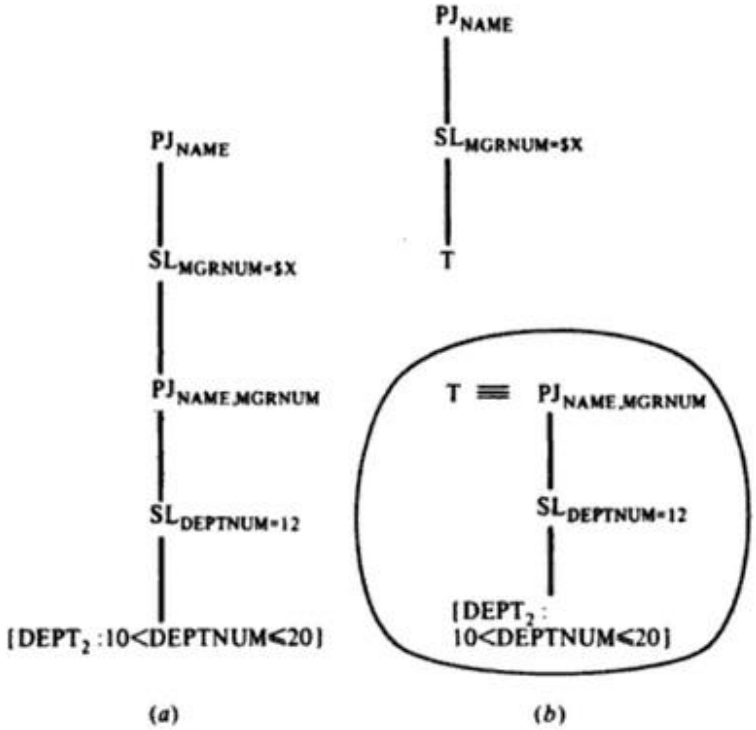


Figure 5.13 Use of temporary relations for parametric queries.

$$\mathbf{PJ}_{NAME,MGRNUM}\,\mathbf{SL}_{DEPTNUM=12}\ EMP$$

We then call T the temporary given by the above expression, and divide the operator tree into two pieces in correspondence to T (see Figure 5.13*b*). T represents the temporary relation that will be used for repeated executions of query Q10.