SIST
JUSTICE PEACE REVOLUTION
SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY
DEEMED UNIVERSITY

Levels Of Distribution-Transparency

(Unit I continued)

Transparency

- Distribution transparency is the property of distributed databases by the virtue of which the internal details of the distribution are hidden from the users
- The DDBMS designer may choose to fragment tables, replicate the fragments and store them at different sites.
- However, since users are oblivious of these details, they find the distributed database easy to use like any centralized database.

Transparency

- Transparency is the separation of the higher level semantics of a system from the lower level implementation issues.
- Fundamental issue is to provide

  data independence

  in the distributed environment

  - Network (distribution) transparency
  - Replication transparency
  - Fragmentation transparency
    - horizontal fragmentation: selection
    - vertical fragmentation: projection
    - hybrid

Distributed Database Transparency Features

- Distribution transparency
- Transaction transparency
- Failure transparency
- Performance transparency
- Heterogeneity transparency

Distributed Database Transparency Features

- Allow end user to feel like database's only user. User feels like they are working with a centralized database
- Features include:
  - *Distribution transparency* – user does not know where data is located and if replicated or partitioned
  - *Transaction transparency* – transaction can update at several network sites to ensure data integrity

Distributed Database Transparency Features

- *Failure transparency* – system continues to operate in the event of a node failure (other nodes pick up lost functionality)
- *Performance transparency* – allows system to perform as if it were a centralized DBMS. No performance degradation due to use of a network or platform differences
- *Heterogeneity transparency* – allows the integration of several different local DBMSs under a common schema

Distribution Transparency

- Allows management of a physically dispersed database as though it were a centralized database
- Supported by a distributed data dictionary (DDD) which contains the description of the entire database as seen by the DBA.
- **Database administrators** (**DBAs**) use specialized software to store and organize data.
- The role may include capacity planning, installation, configuration, database design, migration, performance monitoring, security, troubleshooting, as well as backup and data recovery.
  - The DDD is itself distributed and replicated at the network nodes.
  - The data dictionary catalogs all objects in the distributed schema, is available at every site, and is accessed identically no matter where it is viewed.
  - It defines the distributed database and shields users, including application developers, from the details of where data resides and how it is accessed.

Issues that must be addressed include:

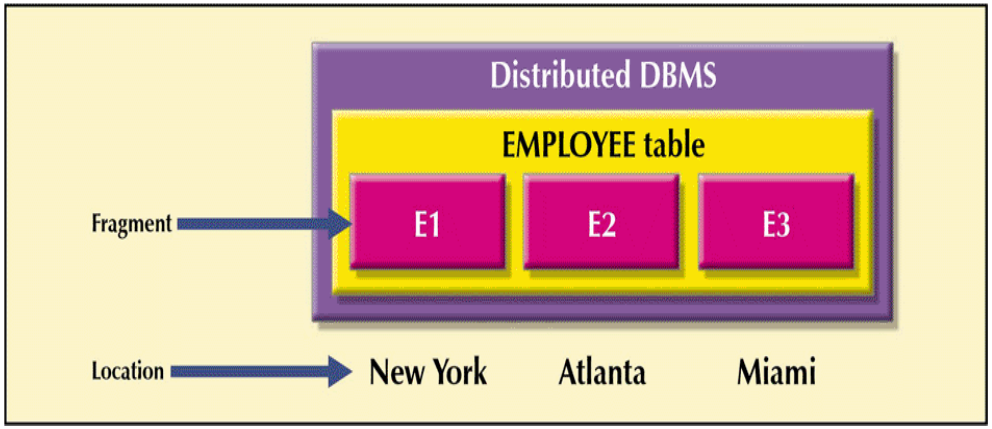
- Placement of the global data dictionary
- Object naming
- The local data dictionary
- Management of interdatabase integrity constraints
- Management of user accounts and privileges

- Three levels of distribution transparency are recognized:
  - ***Fragmentation transparency*** – user does not need to know if a database is partitioned; fragment names and/or fragment locations are not needed
  - ***Location transparency*** – fragment name, but not location, is required
  - ***Local mapping transparency*** – user must specify fragment name and location

Distribution Transparency

- The EMPLOYEE table is divided among three locations (no replication)
- Suppose an employee wants to find all employees with a birthdate prior to jan 1, 1940

FIGURE 10.9 Fragment Locations



A Summary of Transparency Feature

- Fragmentation transparency-
  - SELECT * FROM EMPLOYEE WHERE EMP_DOB < ’01-JAN-1940’;
- Location transparency-
  - SELECT * FROM E1 WHERE EMP_DOB < ’01-JAN-1940’ UNION SELECT * FROM E2 … UNION SELECT * FROM E3…;
- Local Mapping Transparency
  - SELECT * FROM E1 NODE NY WHERE EMP_DOB < ’01-JAN-1940’ UNION SELECT * FROM E2 NODE ATL … UNION SELECT * FROM E3 NODE MIA…;

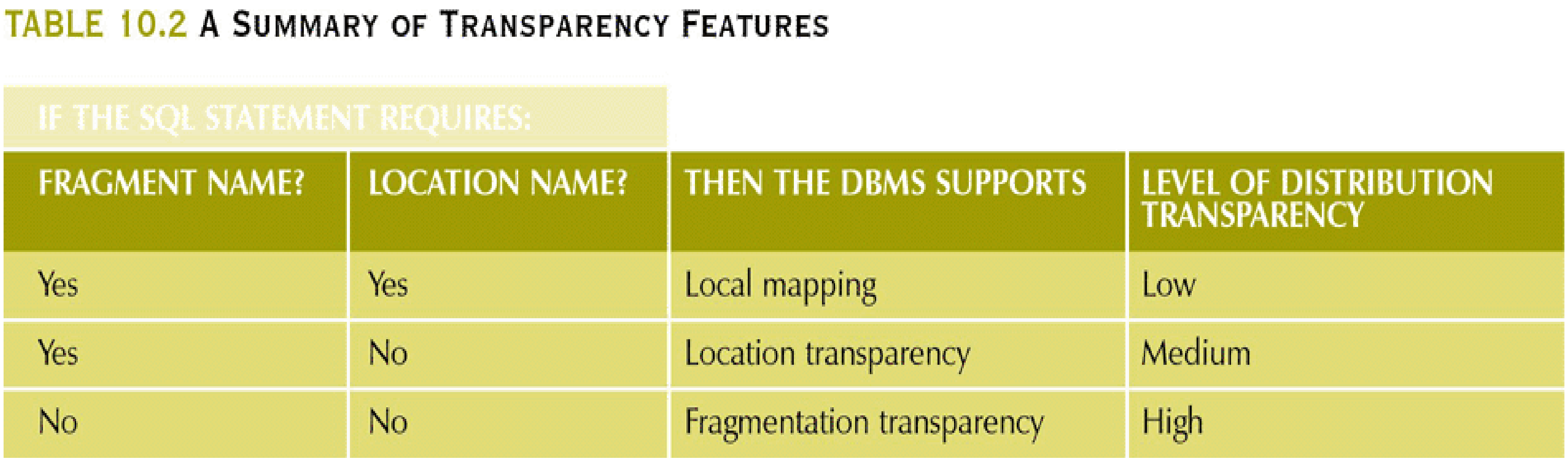
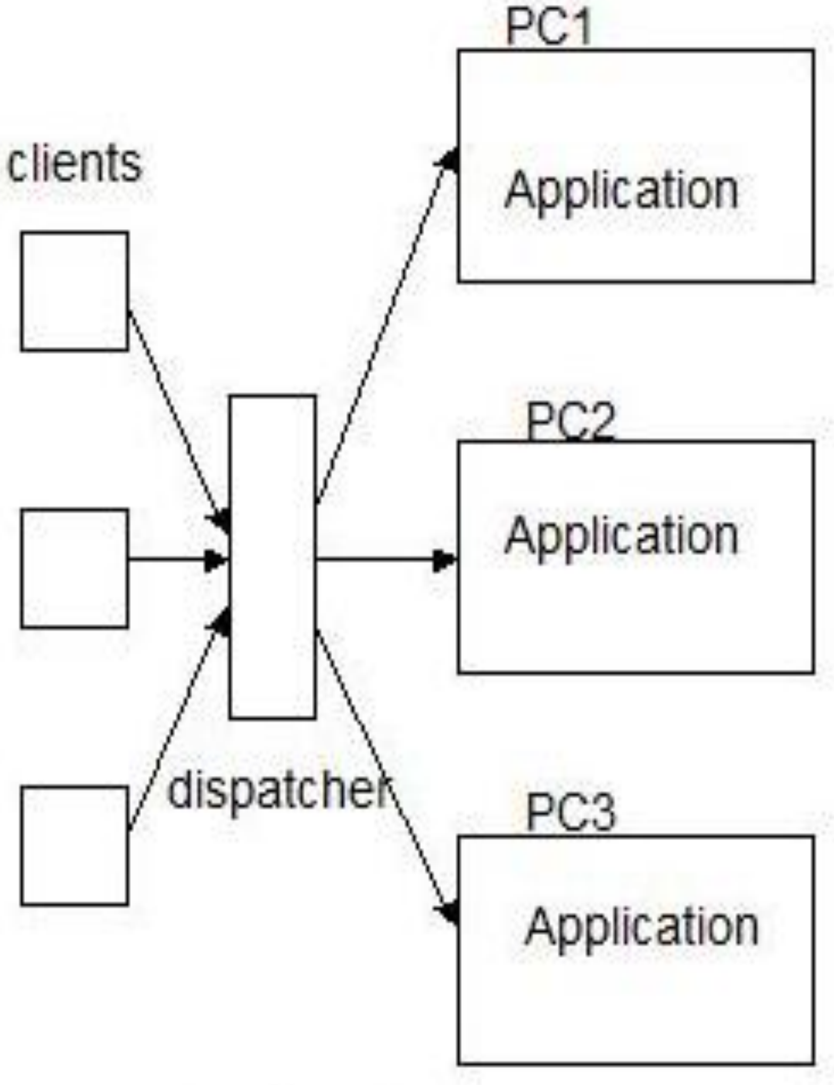


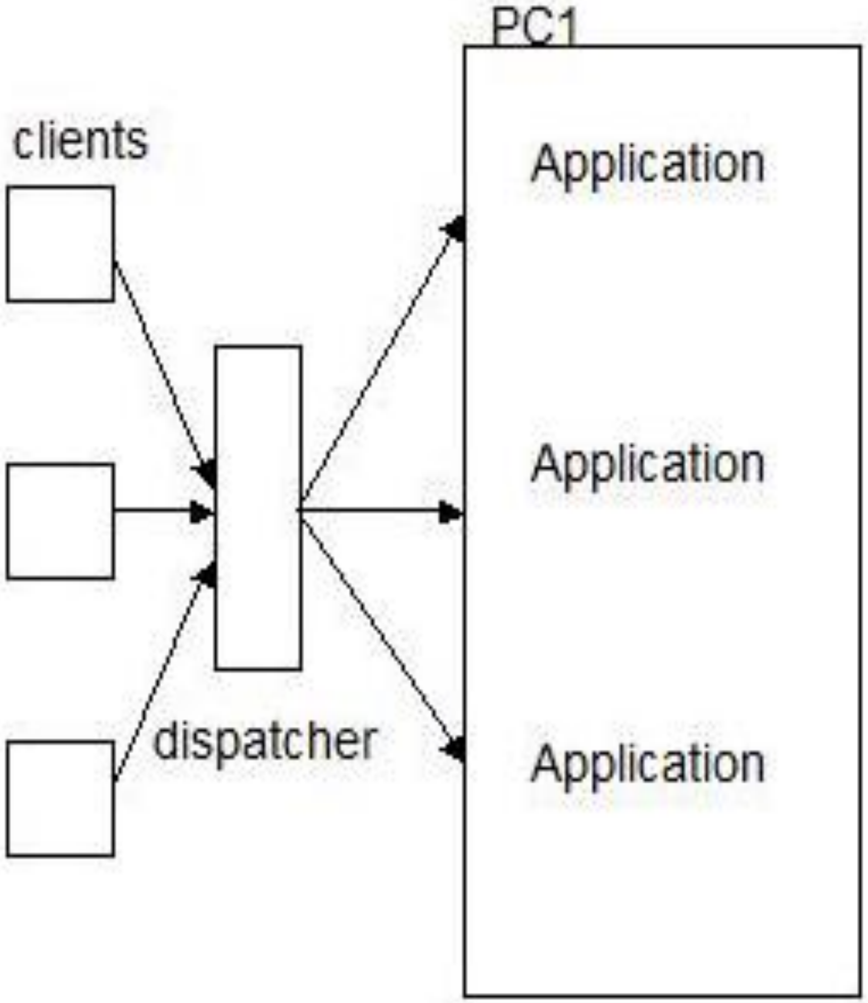
TABLE 10.2 A SUMMARY OF TRANSPARENCY FEATURES

IF THE SQL STATEMENT REQUIRES:			
FRAGMENT NAME?	LOCATION NAME?	THEN THE DBMS SUPPORTS	LEVEL OF DISTRIBUTION TRANSPARENCY
Yes	Yes	Local mapping	Low
Yes	No	Location transparency	Medium
No	No	Fragmentation transparency	High

DISTRIBUTED TRANSPARENCY



Horizontal scaling

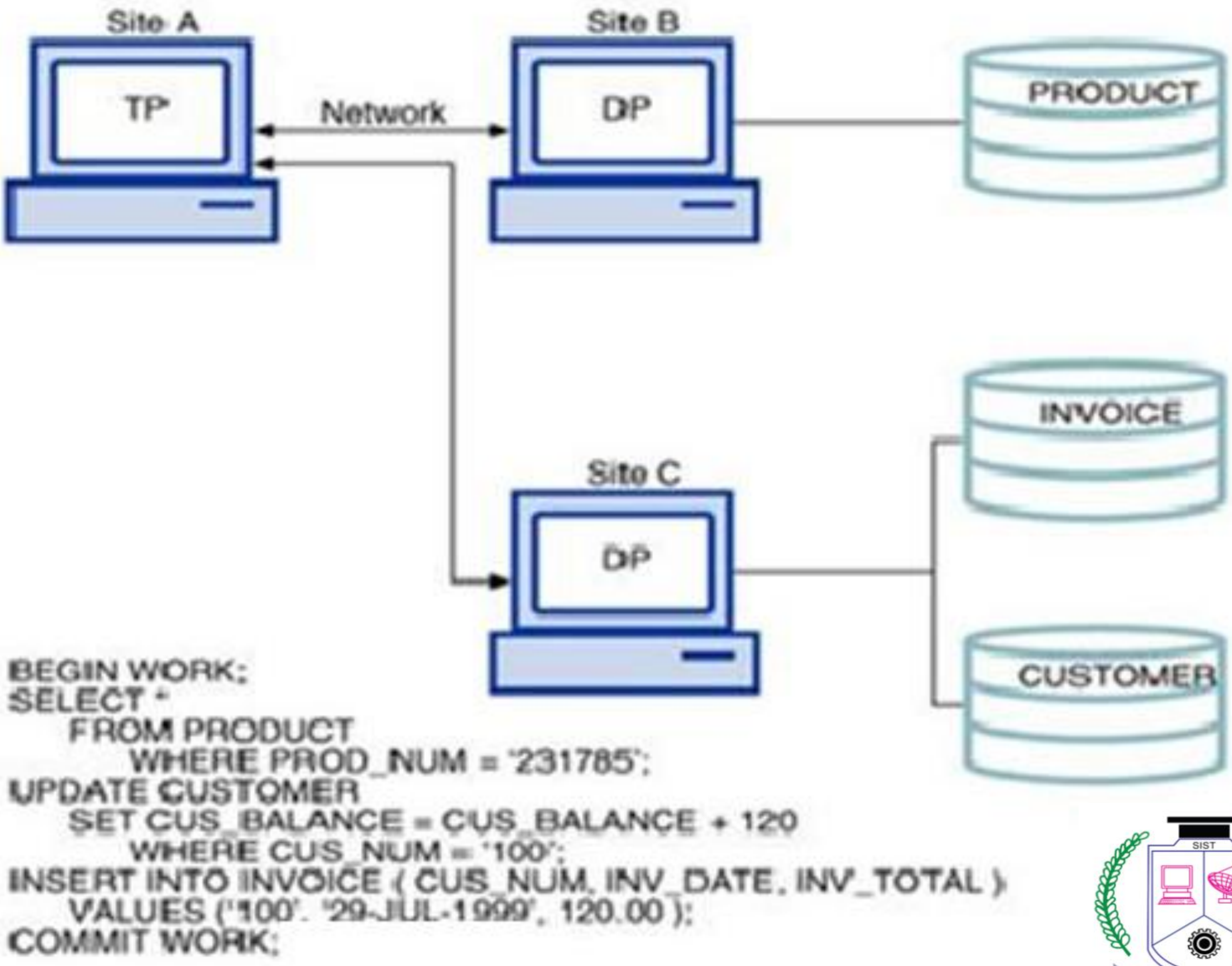


Vertical scaling

Transaction Transparency

- Ensures database transactions will maintain distributed database's integrity and consistency
- A DDBMS transaction can update data stored in many different computers connected in a network
  - Transaction transparency ensures that the transaction will be completed only if all database sites involved in the transaction complete their part of the transaction

TRANSACTION TRANSPARENCY:



```
BEGIN WORK;
SELECT *
   FROM PRODUCT
      WHERE PROD_NUM = '231785';
UPDATE CUSTOMER
   SET CUS_BALANCE = CUS_BALANCE + 120
      WHERE CUS_NUM = '100';
INSERT INTO INVOICE ( CUS_NUM, INV_DATE, INV_TOTAL )
   VALUES ('100', '29-JUL-1999', 120.00 );
COMMIT WORK;
```

A DISTRIBUTED TRANSACTION

Failure transparency

- In a distributed system, failure transparency refers to the extent to which errors and subsequent recoveries of hosts and services within the system are invisible to users and applications.
- For example, if a server fails, but users are automatically redirected to another server and never notice the failure, the system is said to exhibit high failure transparency.
- Failure transparency is one of the most difficult types of transparency to achieve since it is often difficult to determine whether a server has actually failed, or whether it is simply responding very slowly.
- Additionally, it is generally impossible to achieve full failure transparency in a distributed system since networks are unreliable.

Failures in DDBS

1. Transaction Failures
2. Site Failures
3. Media Failures
4. Communication Failures

Transaction Failures

- When a transaction fails, it aborts.
- Thereby, the database must be restored to the state it was in before the transaction started.
- Transactions may fail for several reasons.
- Some failures may be due to deadlock situations or concurrency control algorithms.

Site Failures

- Site failures are usually due to software or hardware failures.
- These failures result in the loss of the main memory contents.
- In distributed database, site failures are of two types:

  1.Total Failure where all the sites of a distributed system fail, and

  2.Partial Failure where only some of the sites of a distributed system fail

Media Failures

- Media Failures refer to the failure of secondary storage devices.
- The failure itself may be due to head crashes, or controller failure.
- In these cases, the media failures result in the inaccessibility of part or the entire database stored on such secondary storage

Communication Failures

- Communication failures, as the name implies, are failures in the communication system between two or more sites.
- This will lead to network partitioning where each site, or several sites grouped together, operates independently.
- As such, messages from one site won’t reach the other sites and will therefore be lost

Performance Transparency

- Allows the system to be reconfigured to improve the performance as the load varies
- Distributed transactions have the added complexity of having to access and update data stored at many different locations.
- In other words all sites involved in the transaction must complete their component, before the transaction results can be stored permanently.
- A DDBMS provides Transaction Transparency if it executes distributed transactions as if they were centralized transactions.

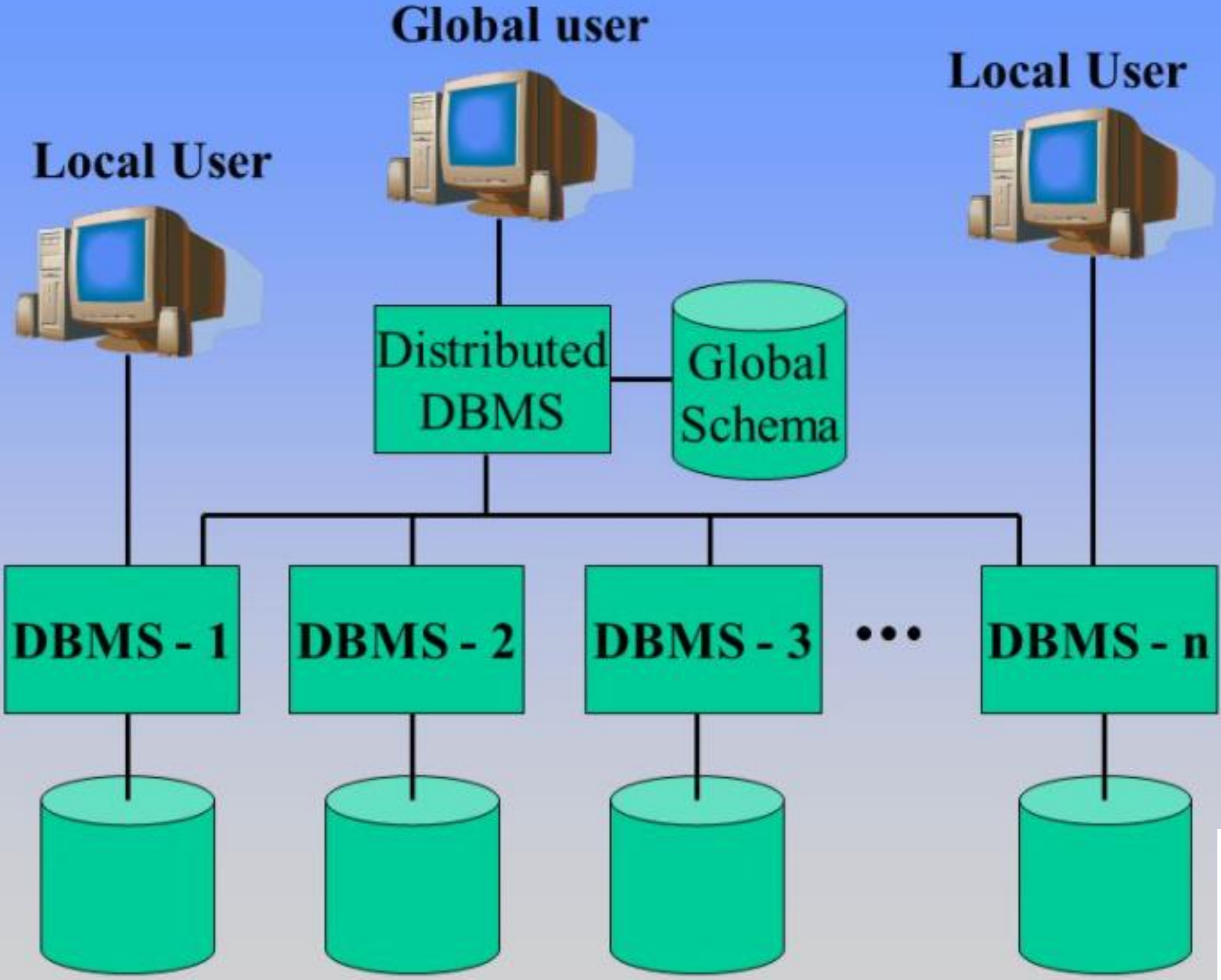
How to attain Performance Transparency

- **A transaction must display atomicity**: that is all the components of a transaction must be completed and permanently stored in the database to ensure the database's consistency and integrity. If a transaction cannot be executed in its entirety, the transaction must be aborted and completed components rolled back.
- **A transaction must display durability**: that is all changes that have been stored permanently in the database, must not be lost in the event of any type of failure.

Heterogeneity transparency

- Heterogeneous systems are preferable because organizations may have different DBMS installed at different sites and may want to access them transparently.
- Heterogeneous database systems have well-accepted standards for gateway protocols to expose DBMS functionality to external applications.
- The gateway protocols help in masking the differences of accessing database servers and bridge the differences between the different servers in a distributed system.

Heterogeneous Distributed Database Environment



Dimensions of distribution transparency

The three dimensions of distribution transparency are −

- Location transparency
- Fragmentation transparency
- Replication transparency

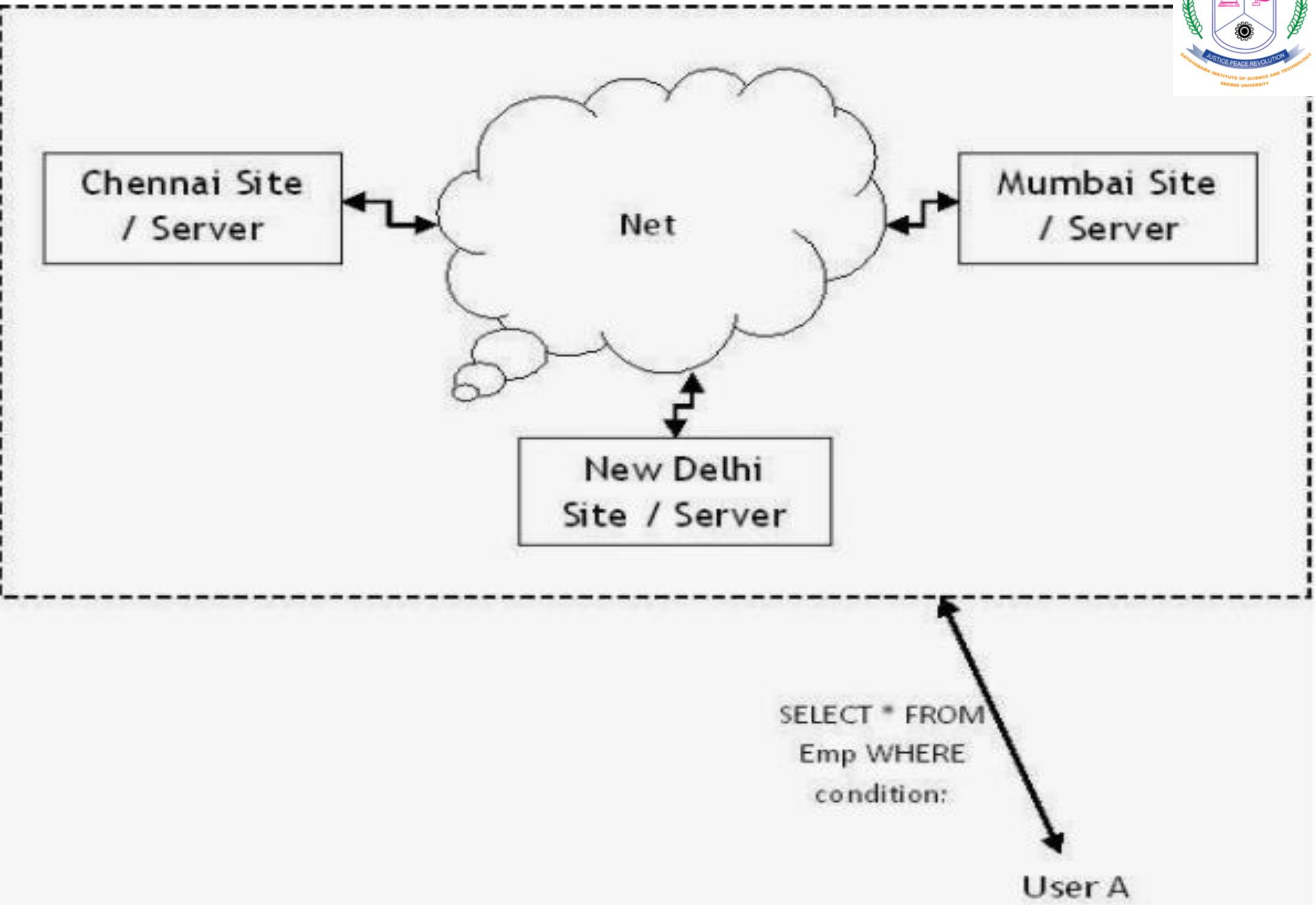
Location Transparency

- Location transparency ensures that the user can query on any table(s) or fragment(s) of a table as if they were stored locally in the user's site.
- The fact that the table or its fragments are stored at remote site in the distributed database system, should be completely oblivious to the end user.
- The address of the remote site(s) and the access mechanisms are completely hidden.
- In order to incorporate location transparency, DDBMS should have access to updated and accurate data dictionary and DDBMS directory which contains the details of locations of data.

Fragmentation Transparency

- Fragmentation transparency enables users to query upon any table as if it were defragmented.
- Thus, it hides the fact that the table the user is querying on is actually a fragment or union of some fragments.
- It also conceals the fact that the fragments are located at diverse sites.
- This is somewhat similar to users of SQL views, where the user may not know that they are using a view of a table instead of the table itself.

FRAGMENTATION TRANSPARENCY

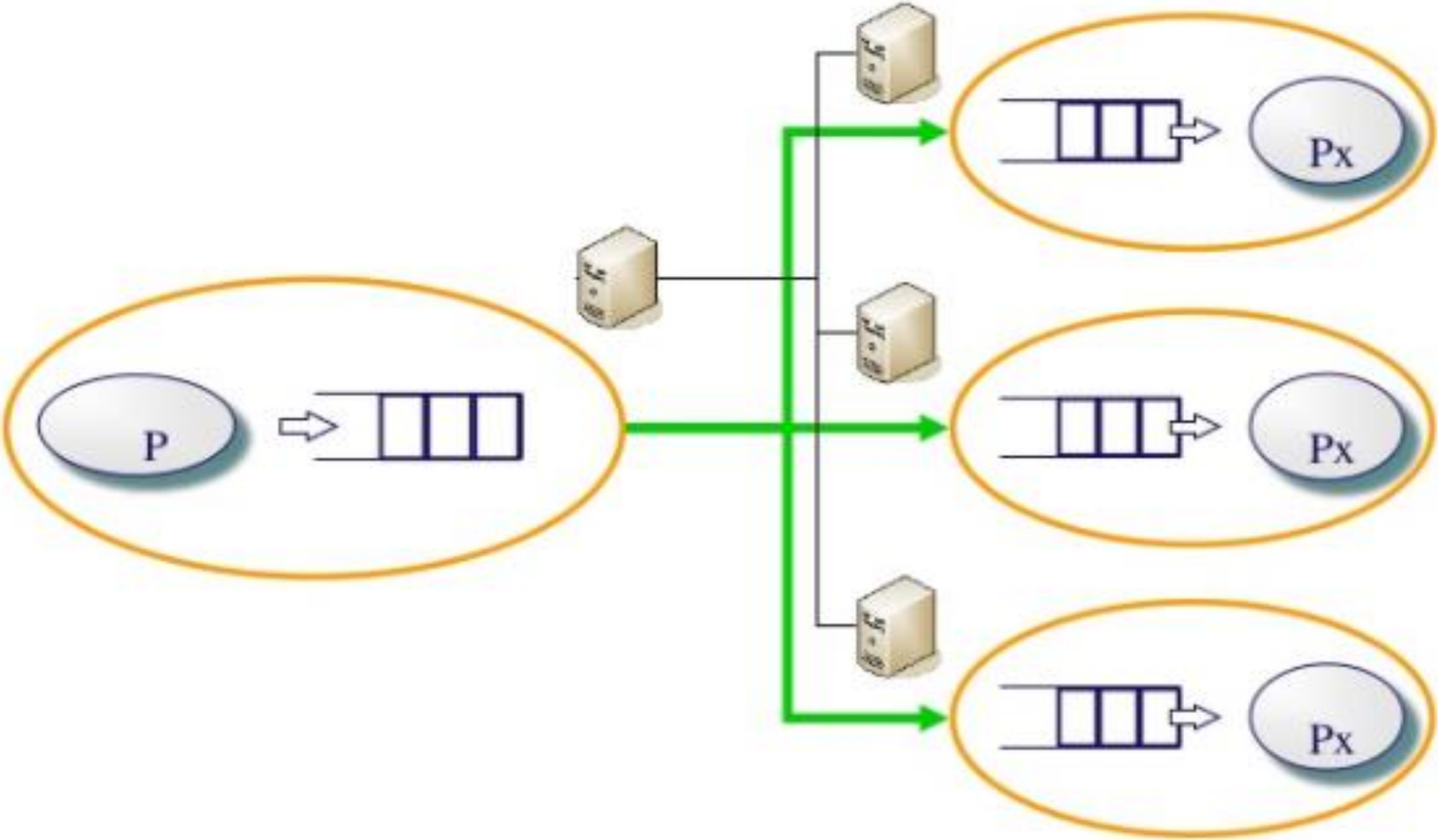


Replication Transparency

- Replication transparency ensures that replication of databases are hidden from the users. It enables users to query upon a table as if only a single copy of the table exists.
- Replication transparency is associated with concurrency transparency and failure transparency. Whenever a user updates a data item, the update is reflected in all the copies of the table.
- However, this operation should not be known to the user. This is concurrency transparency.
- Also, in case of failure of a site, the user can still proceed with his queries using replicated copies without any knowledge of failure. This is failure transparency

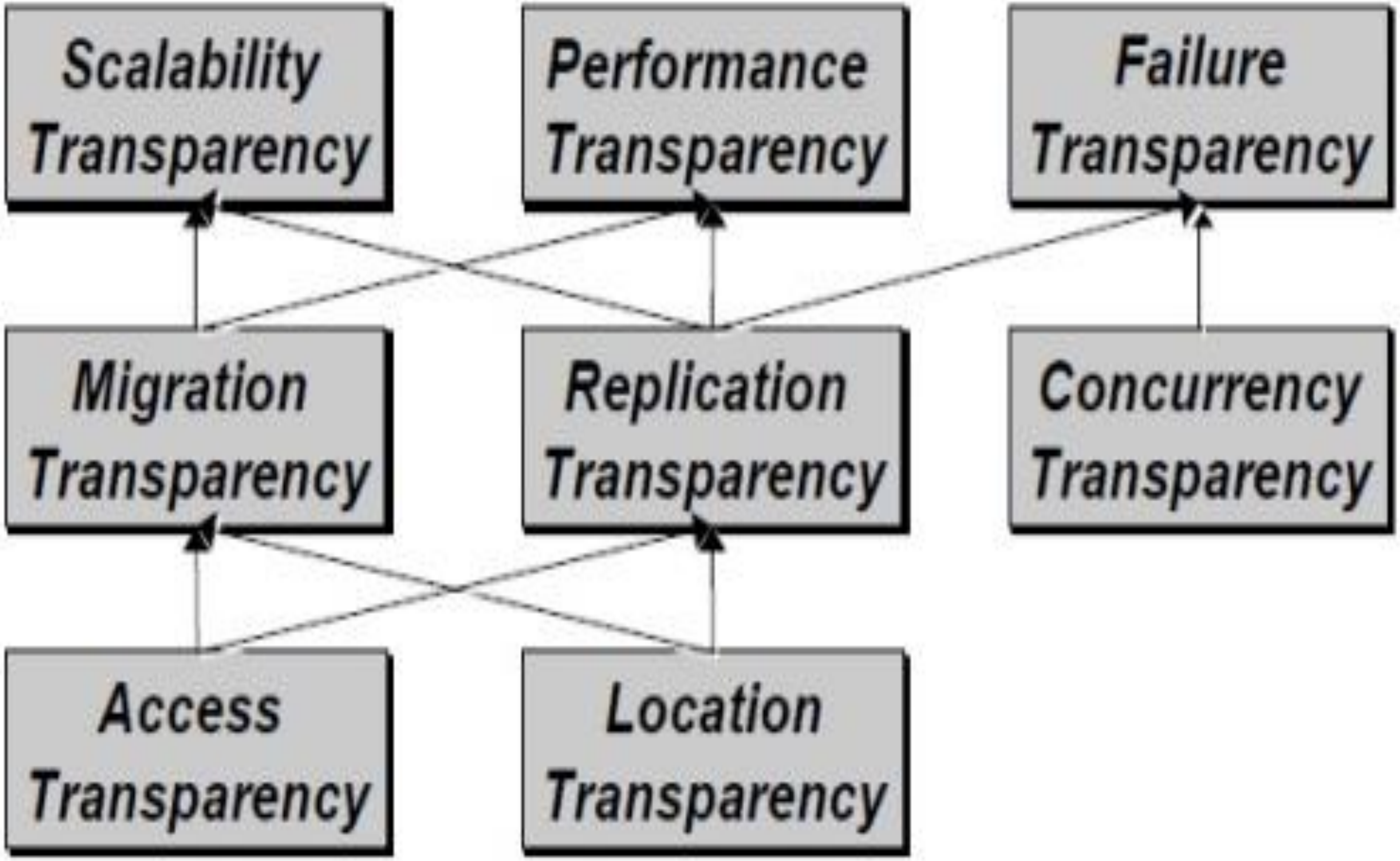
. REPLICATION TANSPARENCY

Message Queue Replication Transparency



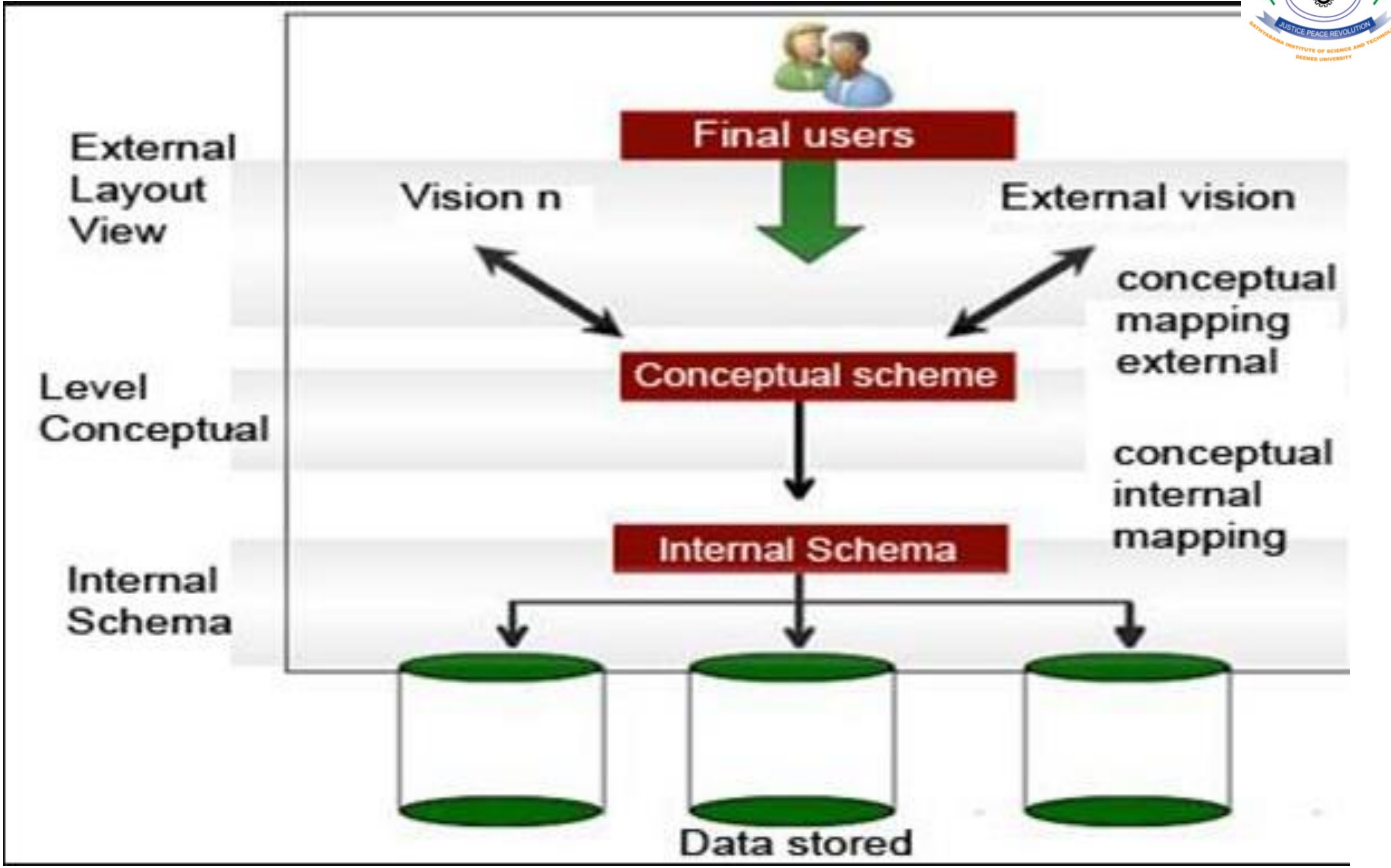
Combination of Transparencies

- In any distributed database system, the designer should ensure that all the stated transparencies are maintained to a considerable extent.
- The designer may choose to fragment tables, replicate them and store them at different sites; all oblivious to the end user.
- However, complete distribution transparency is a tough task and requires considerable design efforts


Scalability Transparency
Performance Transparency
Failure Transparency
Migration Transparency
Replication Transparency
Concurrency Transparency
Access Transparency
Location Transparency

Reference Architecture

- Data in distributed system are usually fragmented and replicated.
- Considering this fragmentation and replication issue. The reference architecture of DBMS consist of the following schemas:-
  - A set of global external schema.
  - A global conceptual schema.
  - A fragmentation schema and allocation schema.
  - A set of schemas for each local DBMS.


External
Layout
View
Final users
Vision n
External vision
conceptual
mapping
external
Level
Conceptual
Conceptual scheme
conceptual
internal
mapping
Internal Schema
Internal
Schema
Data stored

Global external schema

- In a distributed system, user applications and user access to the distributed database are represented by a number of global external schemas.
- This is the topmost level in the reference architecture of DBMS.
- This level describes the part of the distributed database that is relevant to different users.

Global conceptual schema

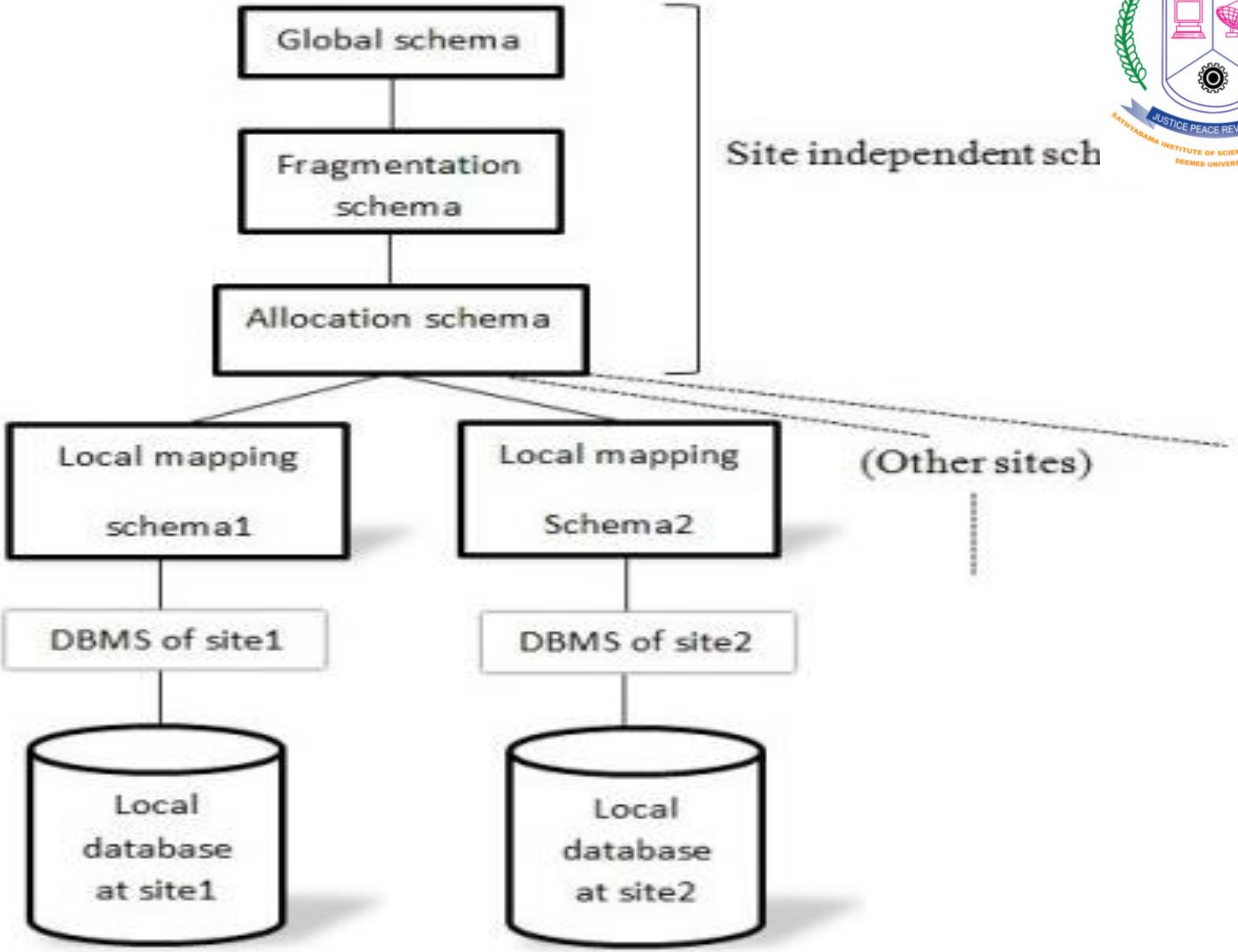
- The GCS represents the logical description of entire database as if it is not distributed.
- This level contains definitions of all entities, relationships among entities and security and integrity information of whole databases stored at all sites in a distributed system.

Fragmentation schema and allocation schema

- The fragmentation schema describes how the data is to be logically partitioned in a distributed database.
- The GCS consists of a set of global relations and the mapping between the global relations and fragments is defined in the fragmentation schema.
- The allocation schema is a description of where the data(fragments)are to be located, taking account of any replication.
- The type of mapping in the allocation schema determined whether the distributed database is redundant or non redundant.
- In case of redundant data distribution the mapping is one to many, whereas in case of non redundant data distribution is one to one.

Local schemas

- In a distributed database system, the physical data organization at each machine is probably different, and therefore it requires an individual internal schema definition at each site, called **local internal schema.**
- To handle fragmentation and replication issues, the logical organization of data at each sites is described by a third layer in the architecture called **local conceptual schema.**



Global schema

Fragmentation schema

Allocation schema

Site independent sch

Local mapping schema1

Local mapping Schema2

(Other sites)

DBMS of site1

DBMS of site2

Local database at site1

Local database at site2

Fragmentation

- Fragmentation is the task of dividing a table into a set of smaller tables. The subsets of the table are called **fragments**.
- Fragmentation can be of three types:
  1. Horizontal
  2. vertical,
  3. hybrid (combination of horizontal and vertical).
- Fragmentation should be done in a way so that the original table can be reconstructed from the fragments. This is needed so that the original table can be reconstructed from the fragments whenever required. This requirement is called “reconstructiveness.”

Advantages of Fragmentation

- Since data is stored close to the site of usage, efficiency of the database system is increased.
- Local query optimization techniques are sufficient for most queries since data is locally available.
- Since irrelevant data is not available at the sites, security and privacy of the database system can be maintained.

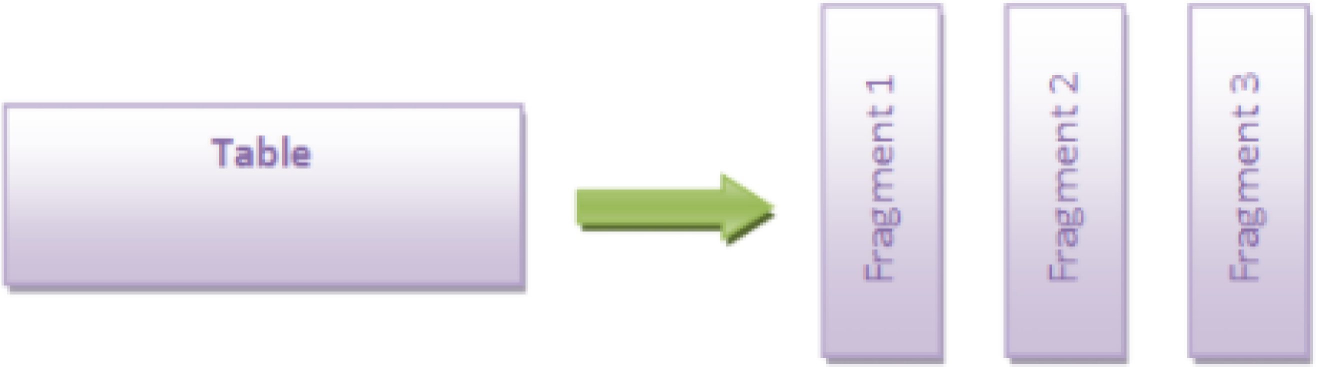
Disadvantages of Fragmentatio

- When data from different fragments are required, the access speeds need to be very high.
- In case of recursive fragmentations, the job of reconstruction will need expensive techniques.
- Lack of back-up copies of data in different sites may render the database ineffective in case of failure of a site.

Vertical Fragmentation

- In vertical fragmentation, the fields or columns of a table are grouped into fragments.
- In order to maintain reconstructiveness, each fragment should contain the primary key field(s) of the table.
- Vertical fragmentation can be used to enforce privacy of data.

Vertical Fragmentation



Vertical Fragmentation

- For example, let us consider that a University database keeps records of all registered students in a Student table having the following schema

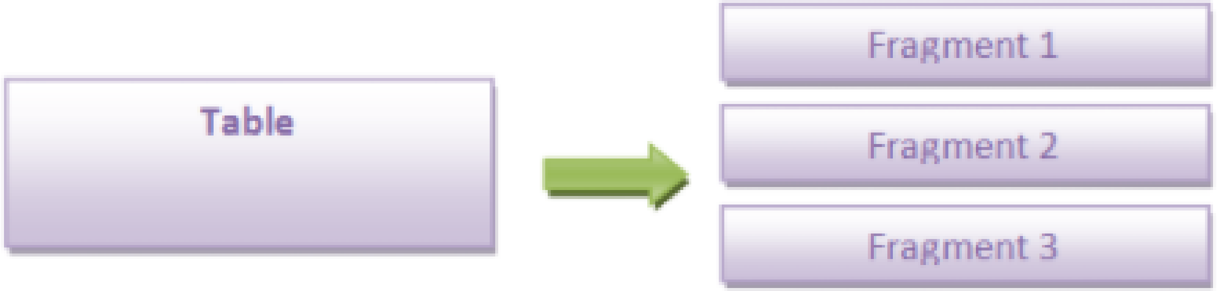
Regd_No	Name	Course	Address	Semester	Fees	Marks

- Now, the fees details are maintained in the accounts section. In this case, the designer will fragment the database as follows –

- CREATE TABLE STD_FEES AS SELECT Regd_No, Fees FROM STUDENT;

Horizontal fragmentation

- Horizontal fragmentation groups the tuples of a table in accordance to values of one or more fields. Horizontal fragmentation should also confirm to the rule of reconstructiveness. Each horizontal fragment must have all columns of the original base table.



Horizontal fragmentation

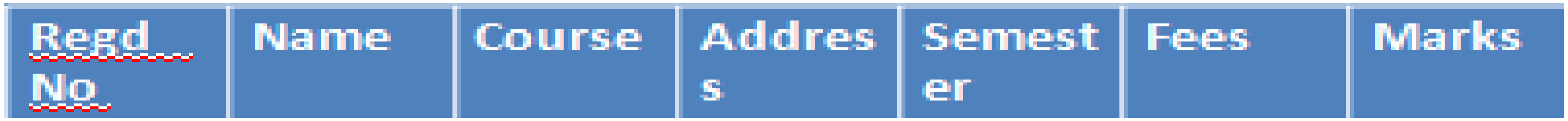
- Horizontal fragmentation can further be classified into two techniques:

1.primary horizontal fragmentation

2.derived horizontal fragmentation

Horizontal fragmentation

- For example, in the student schema, if the details of all students of Computer Science Course needs to be maintained at the School of Computer Science, then the designer will horizontally fragment the database as follows



Regd No	Name	Course	Address	Semester	Fees	Marks

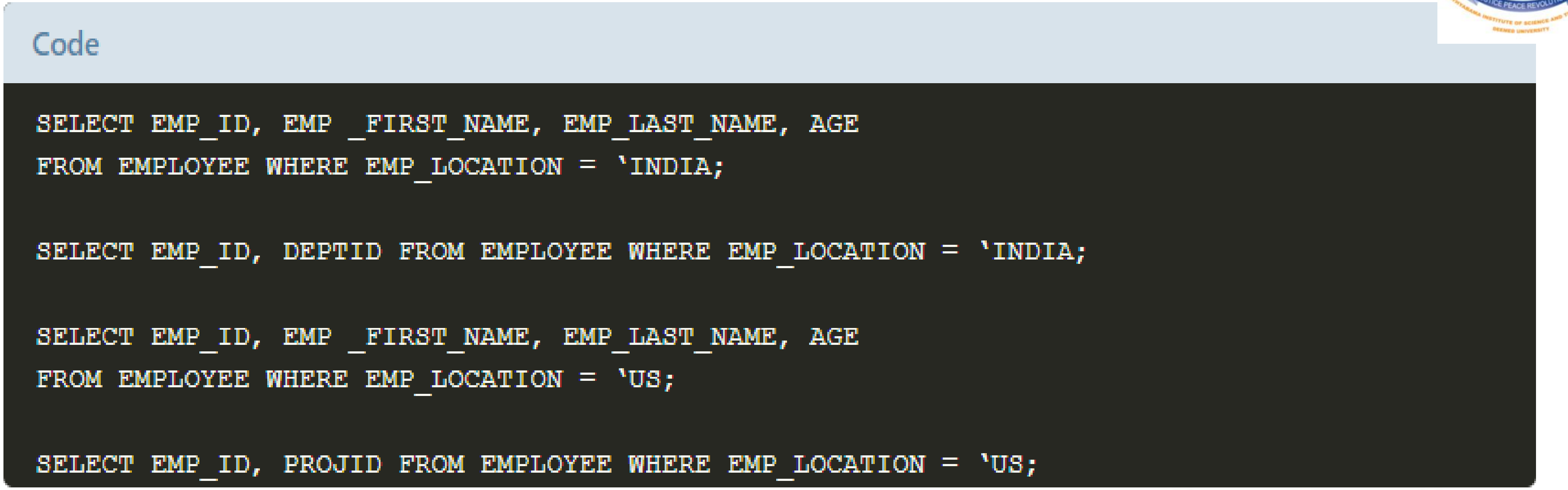
- CREATE COMP_STD AS SELECT * FROM STUDENT WHERE COURSE = "Computer Science“;

Hybrid Fragmentation

- In hybrid fragmentation, a combination of horizontal and vertical fragmentation techniques are used. This is the most flexible fragmentation technique since it generates fragments with minimal extraneous information. However, reconstruction of the original table is often an expensive task.
- Hybrid fragmentation can be done in two alternative ways −

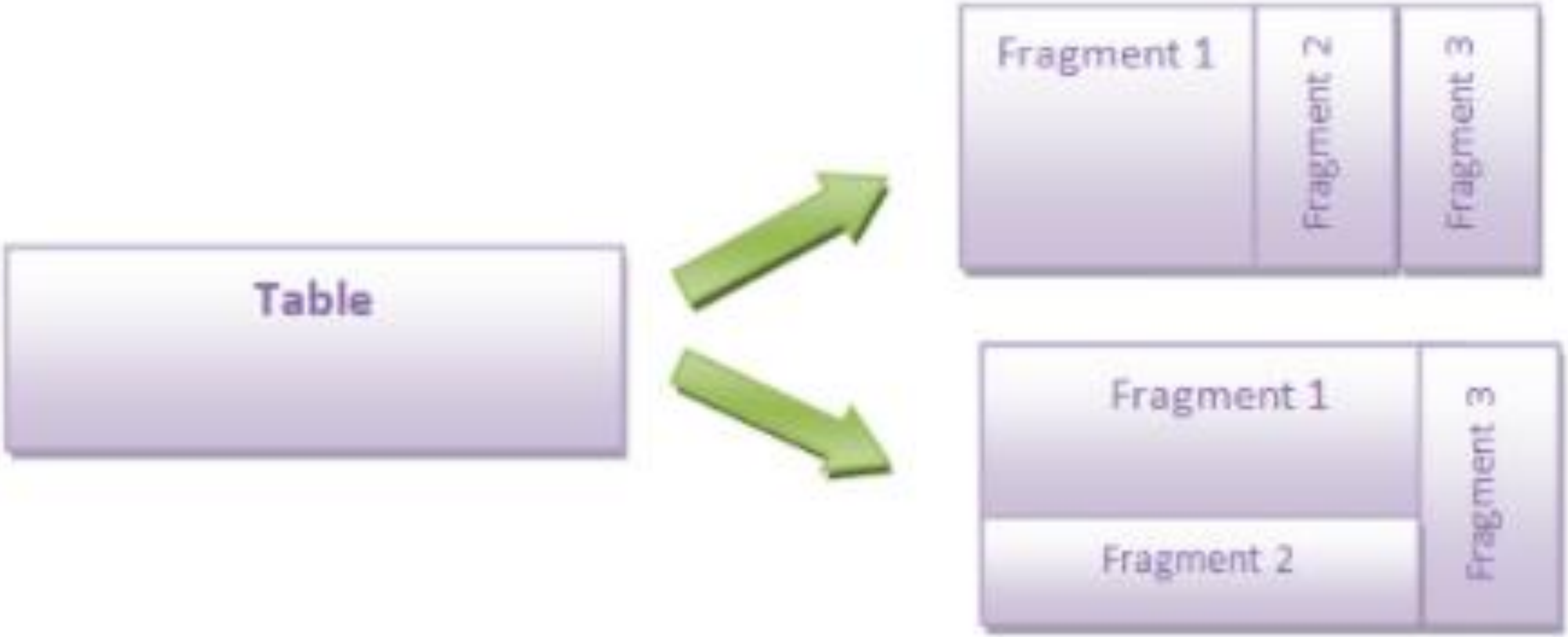
1. At first, generate a set of horizontal fragments; then generate vertical fragments from one or more of the horizontal fragments.
2. At first, generate a set of vertical fragments; then generate horizontal fragments from one or more of the vertical fragments.

Hybrid Fragmentation



Code

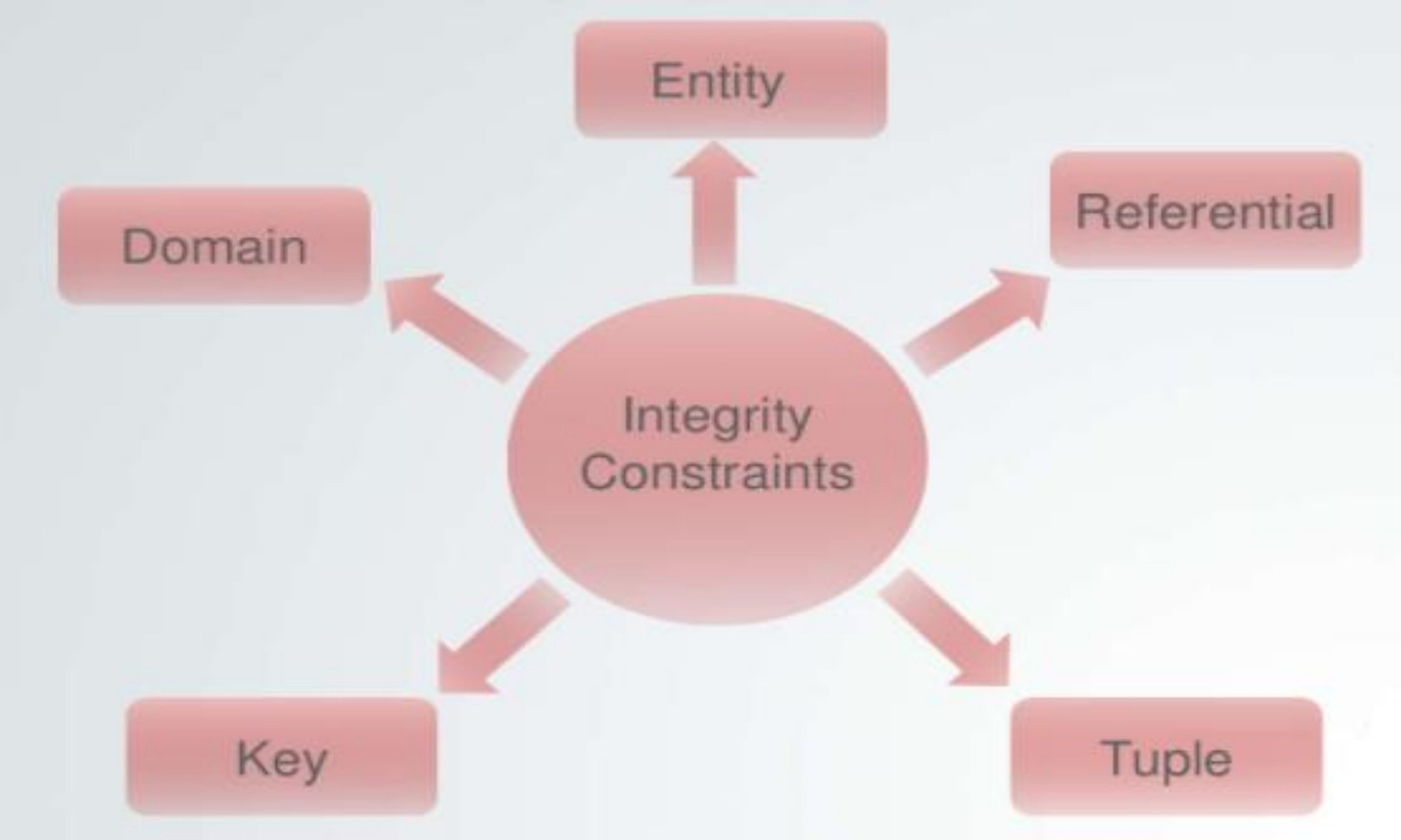
```
SELECT EMP_ID, EMP _FIRST_NAME, EMP_LAST_NAME, AGE
FROM EMPLOYEE WHERE EMP_LOCATION = ‘INDIA;

SELECT EMP_ID, DEPTID FROM EMPLOYEE WHERE EMP_LOCATION = ‘INDIA;

SELECT EMP_ID, EMP _FIRST_NAME, EMP_LAST_NAME, AGE
FROM EMPLOYEE WHERE EMP_LOCATION = ‘US;

SELECT EMP_ID, PROJID FROM EMPLOYEE WHERE EMP_LOCATION = ‘US;
```



Integrity Constraints in Distributed Databases

- Integrity constraints are a set of rules. It is used to maintain the quality of information.
- Integrity constraints ensure that the data insertion, updating, and other processes have to be performed in such a way that data integrity is not affected.
- Thus, integrity constraint is used to guard against accidental damage to the database.

Types of Integrity Constraints



Domain Constraints

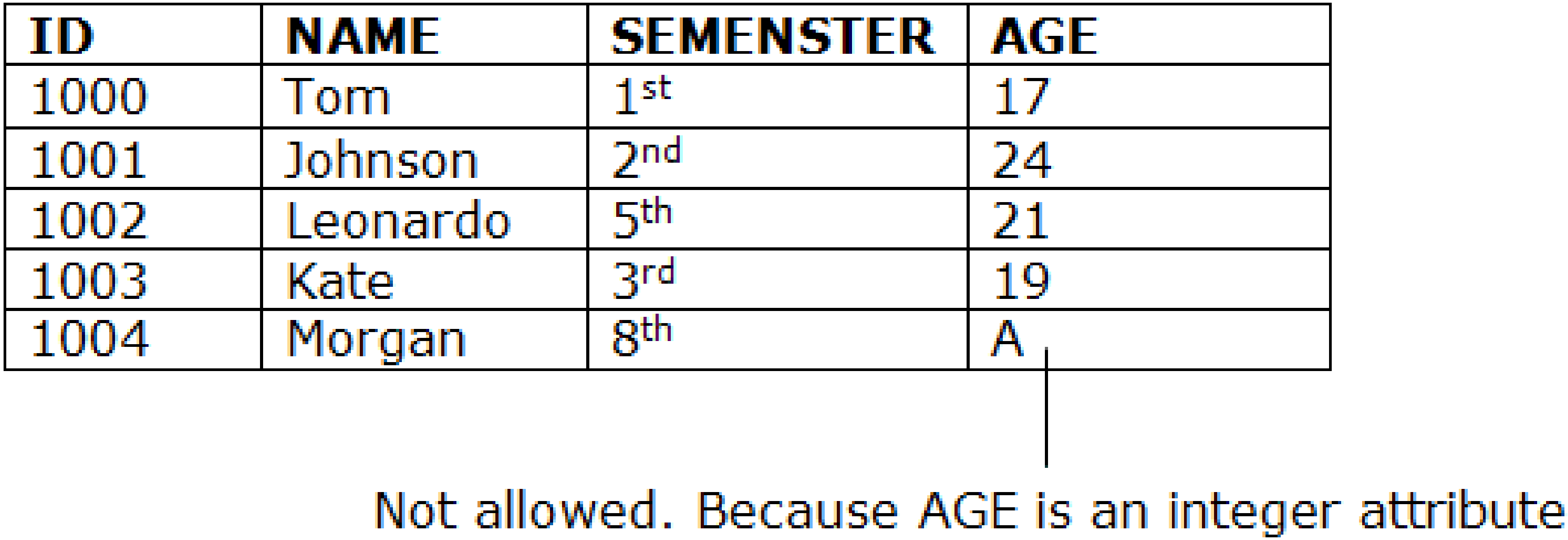
- Domain constraints can be defined as the definition of a valid set of values for an attribute.
- The data type of domain includes string, character, integer, time, date, currency, etc. The value of the attribute must be available in the corresponding domain.
- A data type constraint restricts the range of values and the type of operations that can be applied to the field with the specified data type.

Domain constraints Example

- For example, let us consider that a table "HOSTEL" has three fields - the hostel number, hostel name and capacity. The hostel number should start with capital letter "H" and cannot be NULL, and the capacity should not be more than 150. The following SQL command can be used for data definition –

```
CREATE TABLE HOSTEL ( H_NO VARCHAR2(5) NOT NULL,
 H_NAME VARCHAR2(15),
CAPACITY INTEGER,
CHECK ( H_NO LIKE 'H%'),
CHECK ( CAPACITY <= 150) );
```

Domain constraints Example



ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1004	Morgan	8th	A

Not allowed. Because AGE is an integer attribute

Entity Integrity Control

- Entity integrity control enforces the rules so that each tuple can be uniquely identified from other tuples.
- For this a primary key is defined.
- A primary key is a set of minimal fields that can uniquely identify a tuple.
- Entity integrity constraint states that no two tuples in a table can have identical values for primary keys.

Entity Integrity Control

- The entity integrity constraint states that no field which is a part of the primary key can have NULL value.
- This is because the primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows.
- A table can contain a null value other than the primary key field.

Entity Integrity Control

- For example, in the above hostel table, the hostel number can be assigned as the primary key through the following SQL statement (ignoring the checks) –

```
CREATE TABLE HOSTEL
( H_NO VARCHAR2(5) PRIMARY KEY,
H_NAME VARCHAR2(15),
CAPACITY INTEGER );
```

Entity Integrity Control

Example:

EMPLOYEE

EMP_ID	EMP_NAME	SALARY
123	Jack	30000
142	Harry	60000
164	John	20000
	Jackson	27000

Not allowed as primary key can't contain a NULL value

Referential integrity

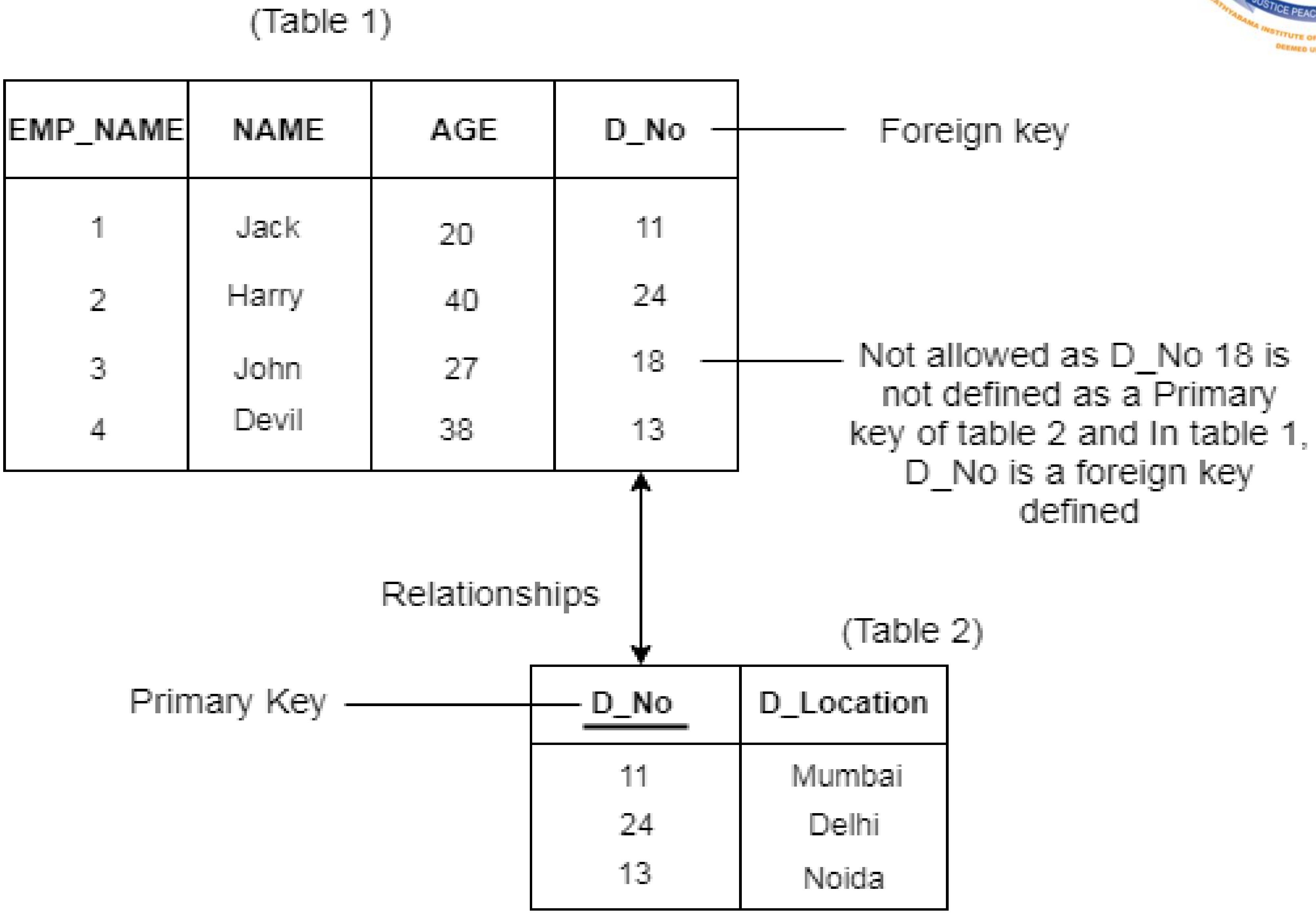
- A referential integrity constraint is specified between two tables.
- Referential integrity constraint lays down the rules of foreign keys.
- A foreign key is a field in a data table that is the primary key of a related table.
- The referential integrity constraint lays down the rule that the value of the foreign key field should either be among the values of the primary key of the referenced table or be entirely NULL.

Referential integrity

- For example, let us consider a student table where a student may opt to live in a hostel. To include this, the primary key of hostel table should be included as a foreign key in the student table. The following SQL statement incorporates this

-
  ```
  CREATE TABLE STUDENT
  ( S_ROLL INTEGER PRIMARY KEY,
  S_NAME VARCHAR2(25) NOT NULL,
  S_COURSE VARCHAR2(10),
  S_HOSTEL VARCHAR2(5) REFERENCES HOSTEL );
  ```

Referential integrity



(Table 1)

EMP_NAME	NAME	AGE	D_No
1	Jack	20	11
2	Harry	40	24
3	John	27	18
4	Devil	38	13

Foreign key

Not allowed as D_No 18 is not defined as a Primary key of table 2 and In table 1, D_No is a foreign key defined

Relationships

(Table 2)

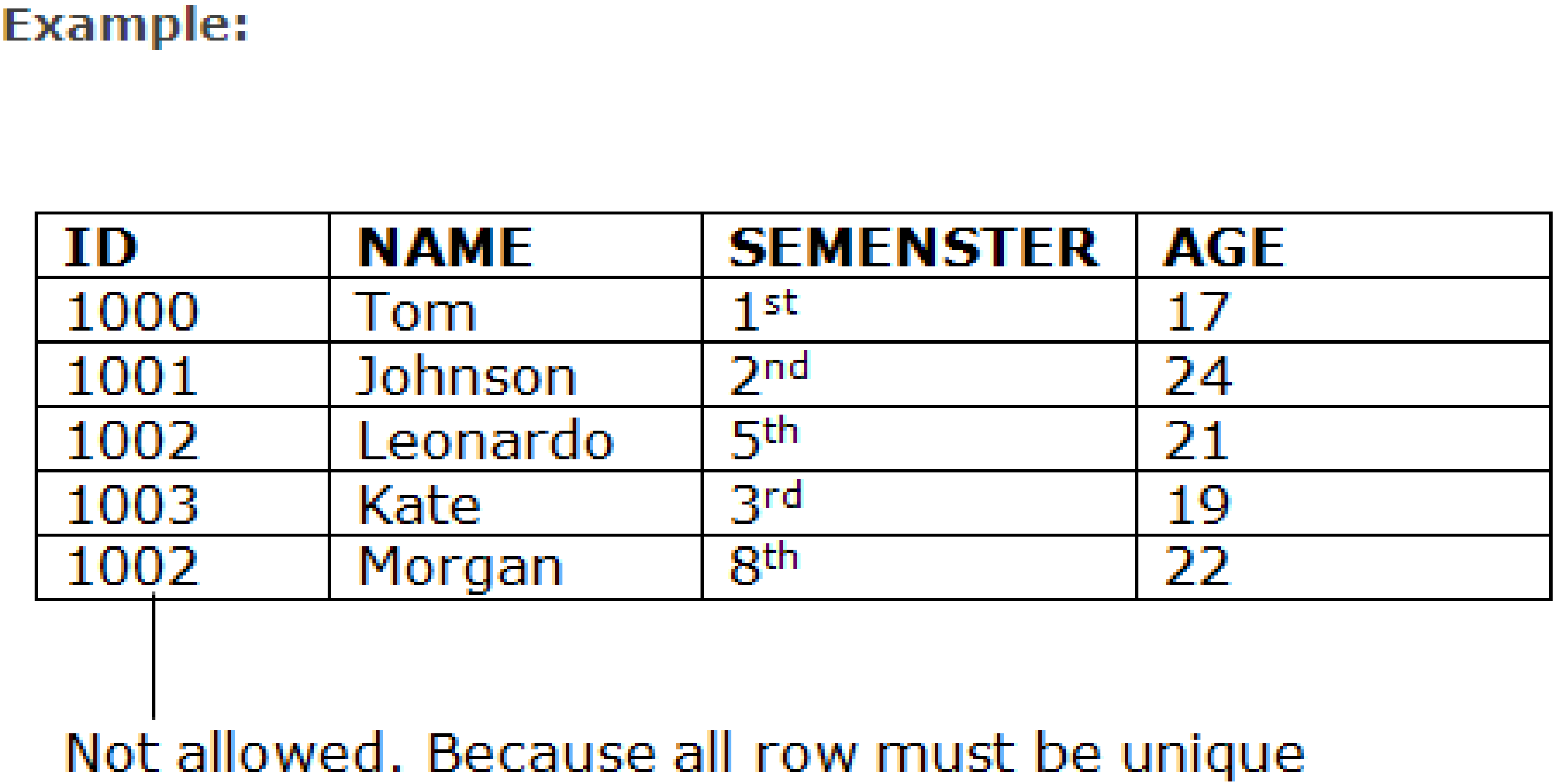
D_No	D_Location
11	Mumbai
24	Delhi
13	Noida

Primary Key

Key constraints

- Keys are the entity set that is used to identify an entity within its entity set uniquely.
- An entity set can have multiple keys, but out of which one key will be the primary key.
- A primary key can contain a unique and null value in the relational table.

Key constraints



Example:

ID	**NAME**	**SEMENSTER**	**AGE**
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1002	Morgan	8th	22

Not allowed. Because all row must be unique

Foreign Key integrity constraints

- There are two types of foreign key integrity constraints

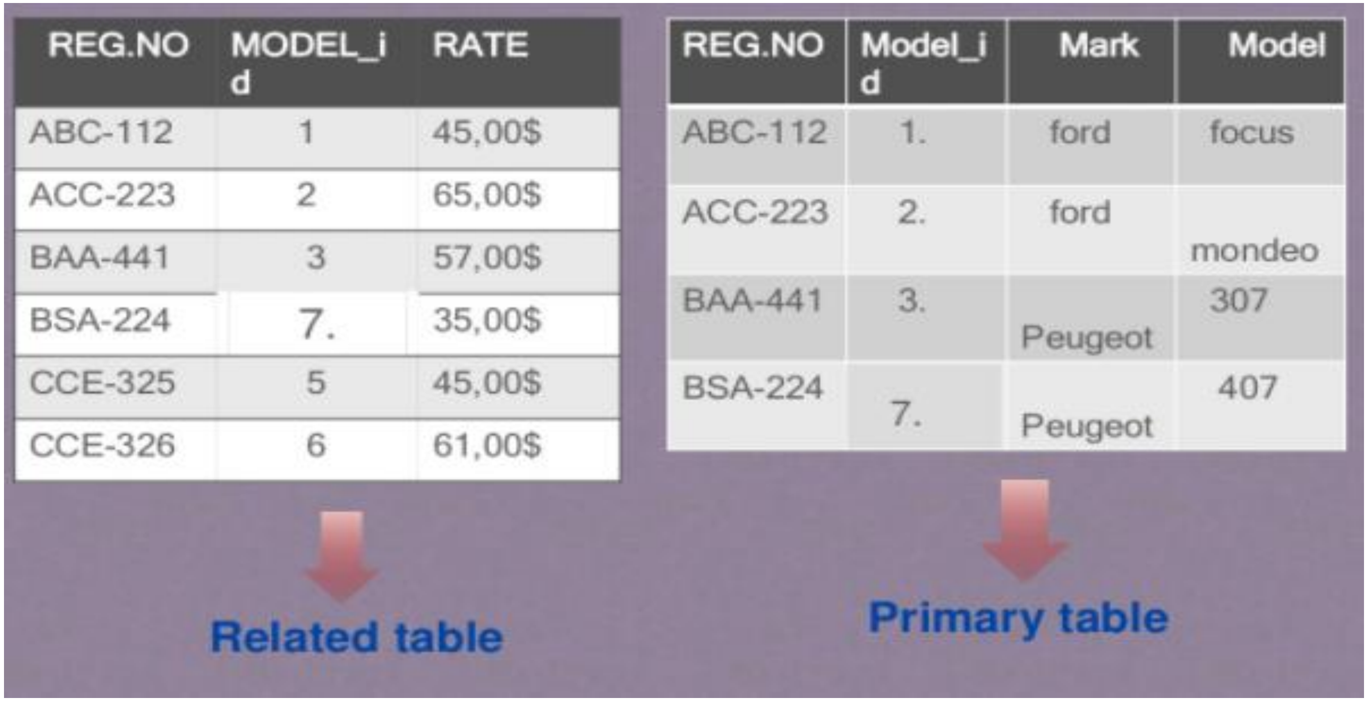
  1.Cascade update related fields

  2.Cascade delete related rows

- These constraints affect referential integrity constraints

Cascade update related fields

- Any time you change the primary table the foreign key values are updated in the matching rows in the related table



REG.NO	MODEL_id	RATE
ABC-112	1	45,00$
ACC-223	2	65,00$
BAA-441	3	57,00$
BSA-224	7.	35,00$
CCE-325	5	45,00$
CCE-326	6	61,00$

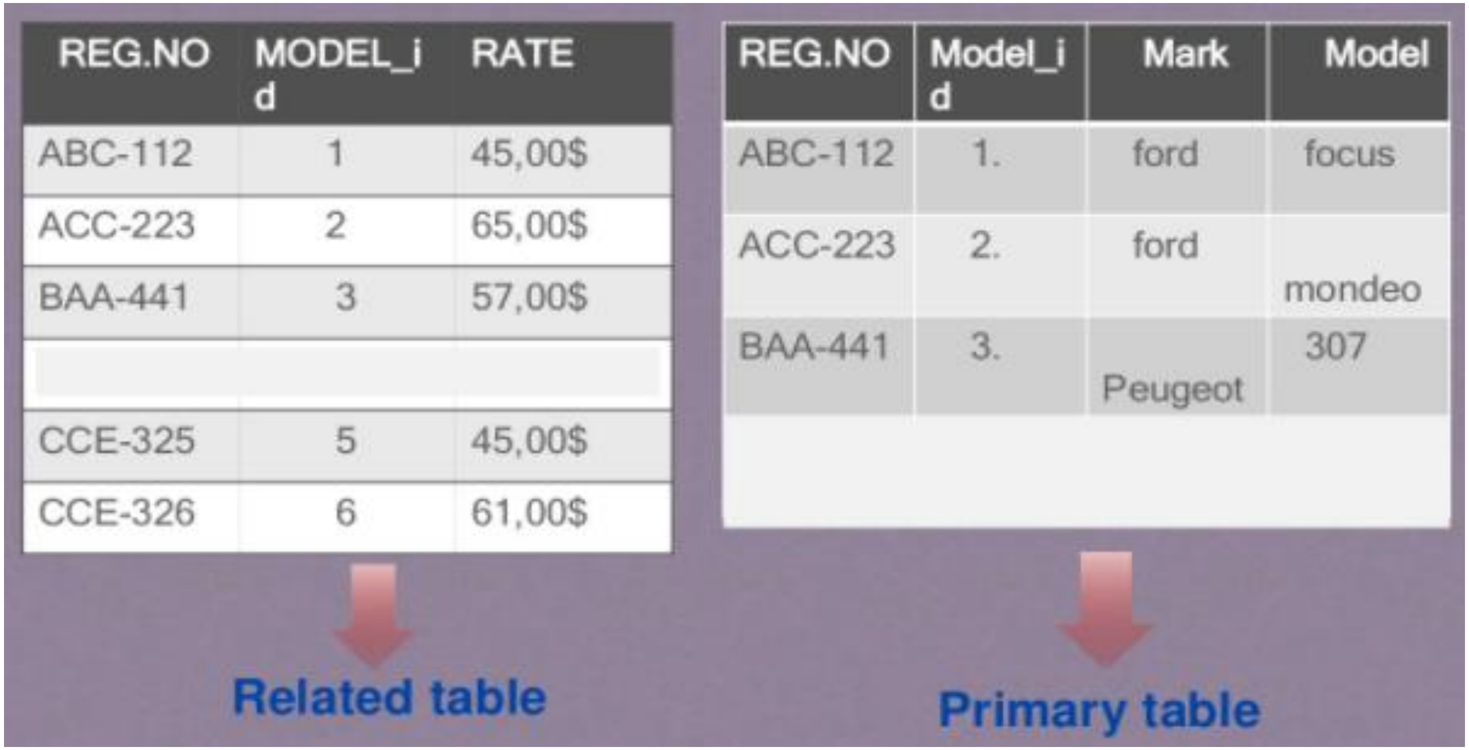
Related table

REG.NO	Model_id	Mark	Model
ABC-112	1.	ford	focus
ACC-223	2.	ford	mondeo
BAA-441	3.	Peugeot	307
BSA-224	7.	Peugeot	407

Primary table

Cascade delete related rows

- Any time you delete a row in the primary table the matching rows are automatically deleted in the related table



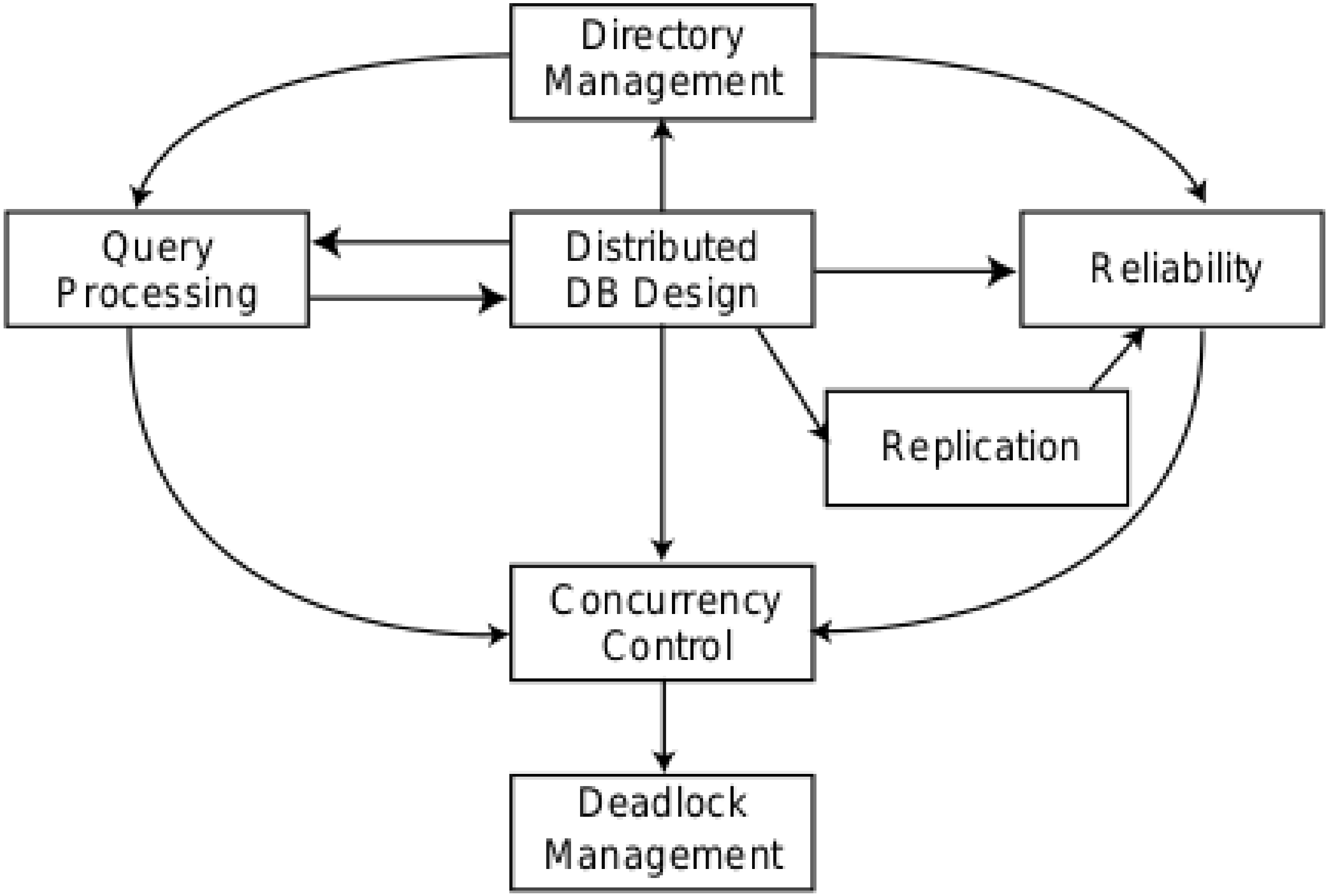
REG.NO	MODEL_id	RATE
ABC-112	1	45,00$
ACC-223	2	65,00$
BAA-441	3	57,00$
CCE-325	5	45,00$
CCE-326	6	61,00$

Related table

REG.NO	Model_id	Mark	Model
ABC-112	1.	ford	focus
ACC-223	2.	ford	mondeo
BAA-441	3.	Peugeot	307

Primary table

Architectural Issues



Distributed Database Design

- One of the main questions that is being addressed is how database and the applications that run against it should be placed across the sites.
- There are two basic alternatives to placing data:

  1. partitioned (or no-replicated) and
  2. replicated.

- In the partitioned scheme the database is divided into a number of disjoint partitions each of which is placed at different site.

Distributed Database Design

- Replicated designs can be either fully replicated (also called fully duplicated) where entire database is stored at each site, or partially replicated (or partially duplicated) where each partition of the database is stored at more than one site, but not at all the sites.
- The two fundamental design issues are fragmentation, the separation of the database into partitions called fragments, and distribution, the optimum distribution of fragments

Distributed Directory Management

- A directory contains information (such as descriptions and locations) about data items in the database.
- Problems related to directory management are similar in nature to the database placement problem discussed in the preceding section.
- A directory may be global to the entire DDBS or local to each site; it can be centralized at one site or distributed over several sites; there can be a single copy or multiple copies.

Distributed Query Processing

- Query processing deals with designing algorithms that analyze queries and convert them into a series of data manipulation operations.
- The problem is how to decide on a strategy for executing each query over the network in the most cost effective way, however cost is defined.
- The factors to be considered are the distribution of data, communication cost, and lack of sufficient locally-available information.
- The objective is to optimize where the inherent parallelism is used to improve the performance of executing the transaction, subject to the abovementioned constraints.

Distributed Concurrency Control

- Concurrency control involves the synchronization of access to the distributed database, such that the integrity of the database is maintained.
- One not only has to worry about the integrity of a single database, but also about the consistency of multiple copies of the database.
- The condition that requires all values of multiple copies of every data item to converge to the same value is called mutual consistency.

Distributed Concurrency Control

- The two general classes are

  1. Pessimistic, synchronizing the execution of the user request before the execution starts,

  2. Optimistic, executing requests and then checking if the execution has compromised the consistency of the database.

- Two fundamental primitives that can be used with both approaches are locking, which is based on the

  1. Mutual exclusion of access to data items,

  2. Time stamping, where transactions executions are ordered based on timestamps.

Distributed Deadlock Management

- The deadlock problem in DDBSs is similar in nature to that encountered in operating systems.
- The competition among users for access to a set of resources (data, in this case) can result in a deadlock if the synchronization mechanism is based on locking.
- The well-known alternatives of prevention, avoidance, and detection/recovery also apply to DDBSs.

Reliability of Distributed DBMS

- It is important that mechanisms be provided to ensure the consistency of the database as well as to detect failures and recover from them.
- The implication for DDBSs is that when a failure occurs and various sites become either inoperable or inaccessible, the databases at the operational sites remain consistent and up to date.
- Furthermore, when the computer system or network recovers from the failure, the DDBSs should be able to recover and bring the databases at the failed sites up-to date.
- This may be especially difficult in the case of network partitioning, where the sites are divided into two or more groups with no communication among them.

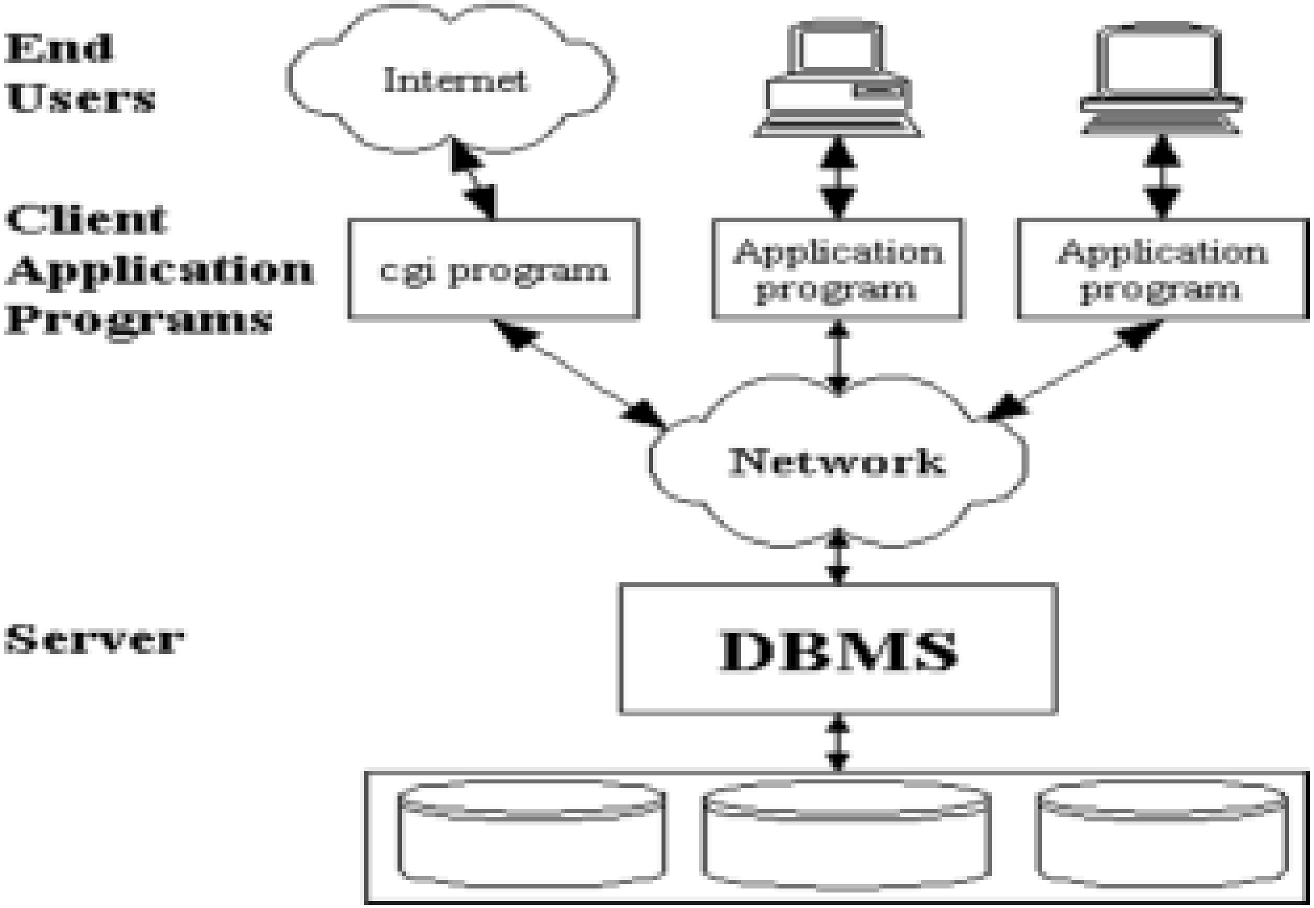
Replication

- If the distributed database is (partially or fully) replicated, it is necessary to implement protocols that ensure the consistency of the replicas, i.e. copies of the same data item have the same value.
- These protocols can be eager in that they force the updates to be applied to all the replicas before the transactions completes, or they may be lazy so that the transactions updates one copy (called the master) from which updates are propagated to the others after the transaction completes.

Client - Server Architecture for DDBMS

- This is a two-level architecture where the functionality is divided into servers and clients.
- The server functions primarily encompass data management, query processing, optimization and transaction management.
- Client functions include mainly user interface. However, they have some functions like consistency checking and transaction management.

Client/Server Architecture



End Users

Client Application Programs

Server

Internet

cgi program

Application program

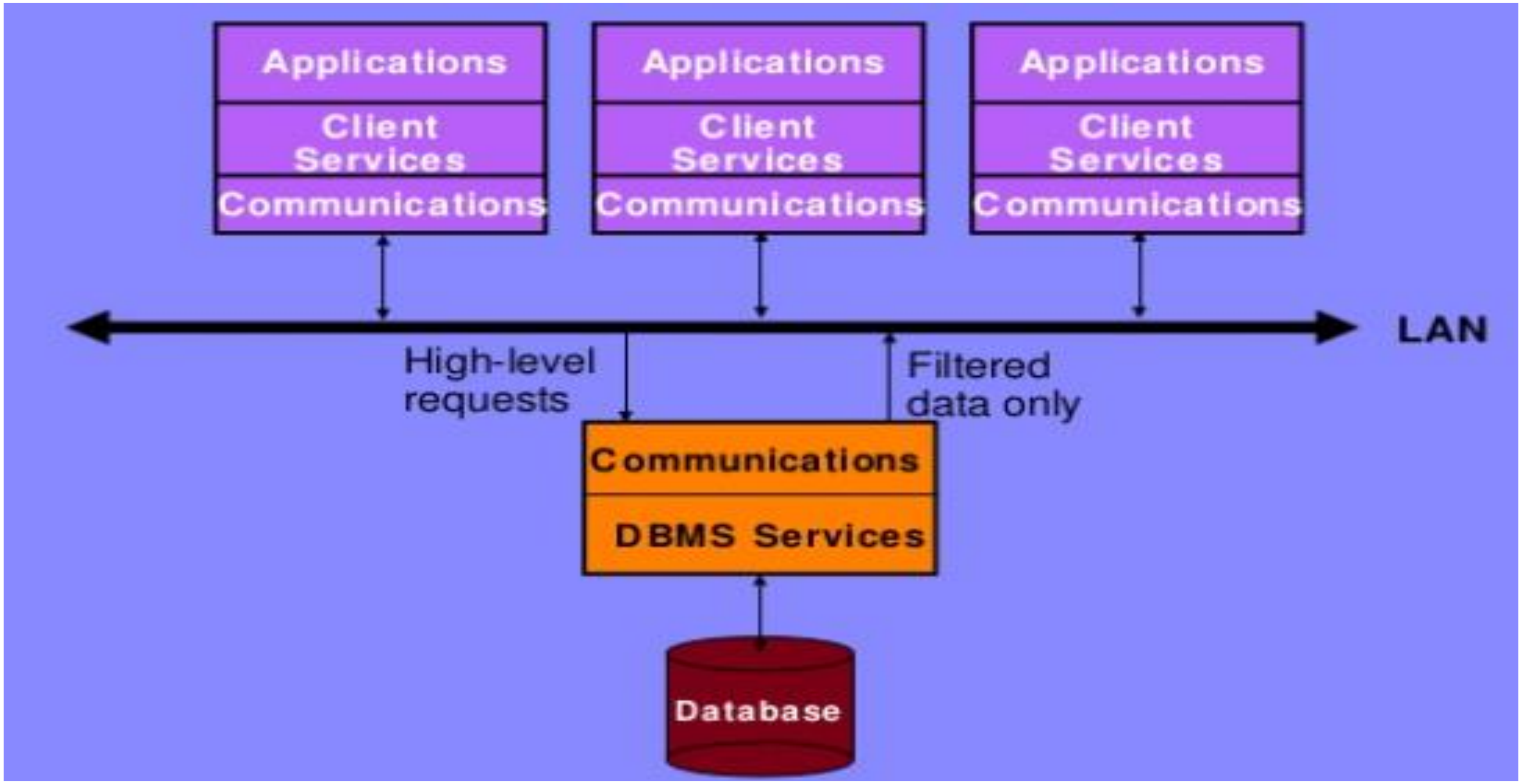
Application program

Network

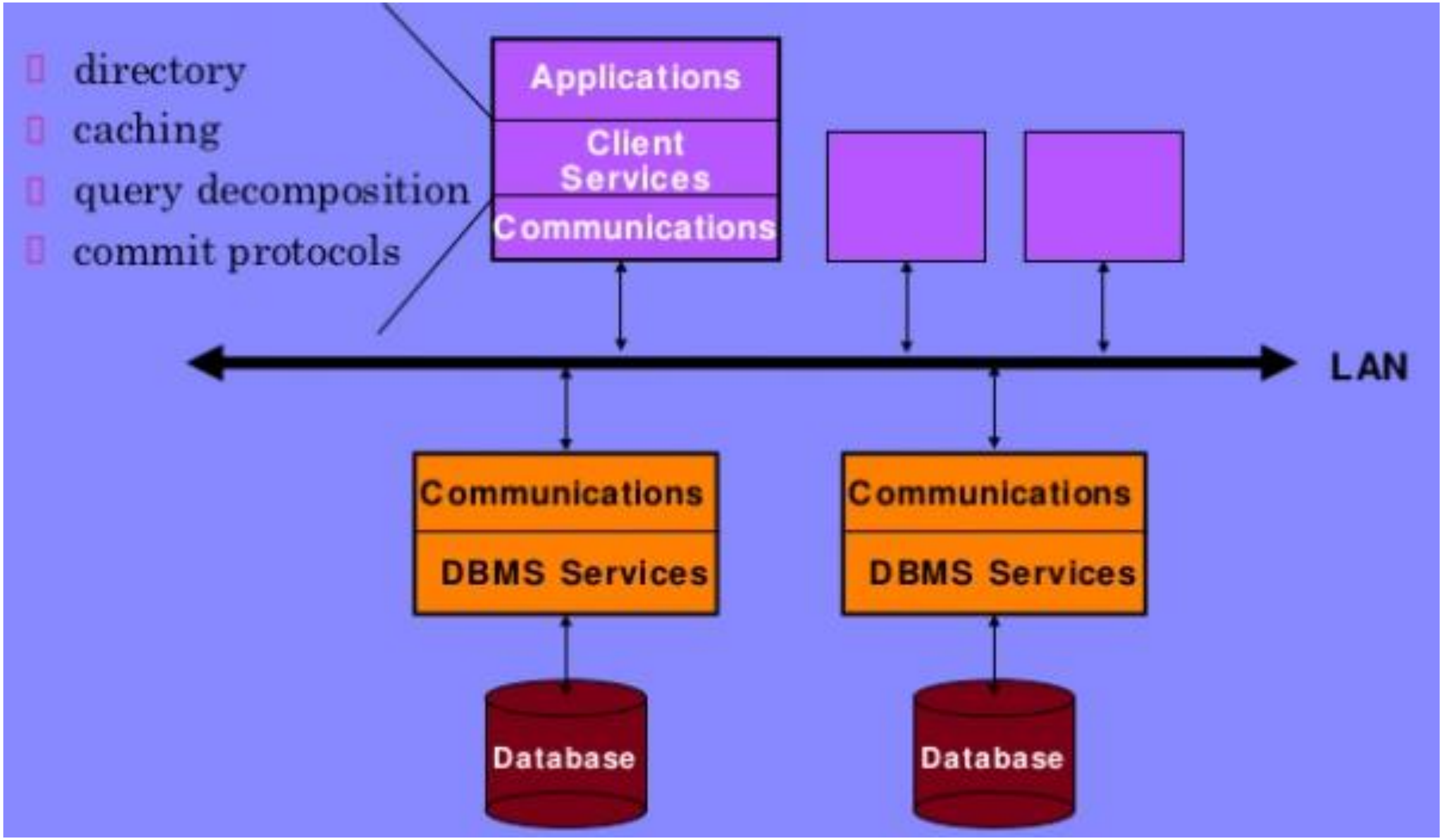
DBMS

- The two different client - server architecture are –
  - ✓Single Server Multiple Client
  - ✓Multiple Server Multiple Client

Single Server Multiple Client

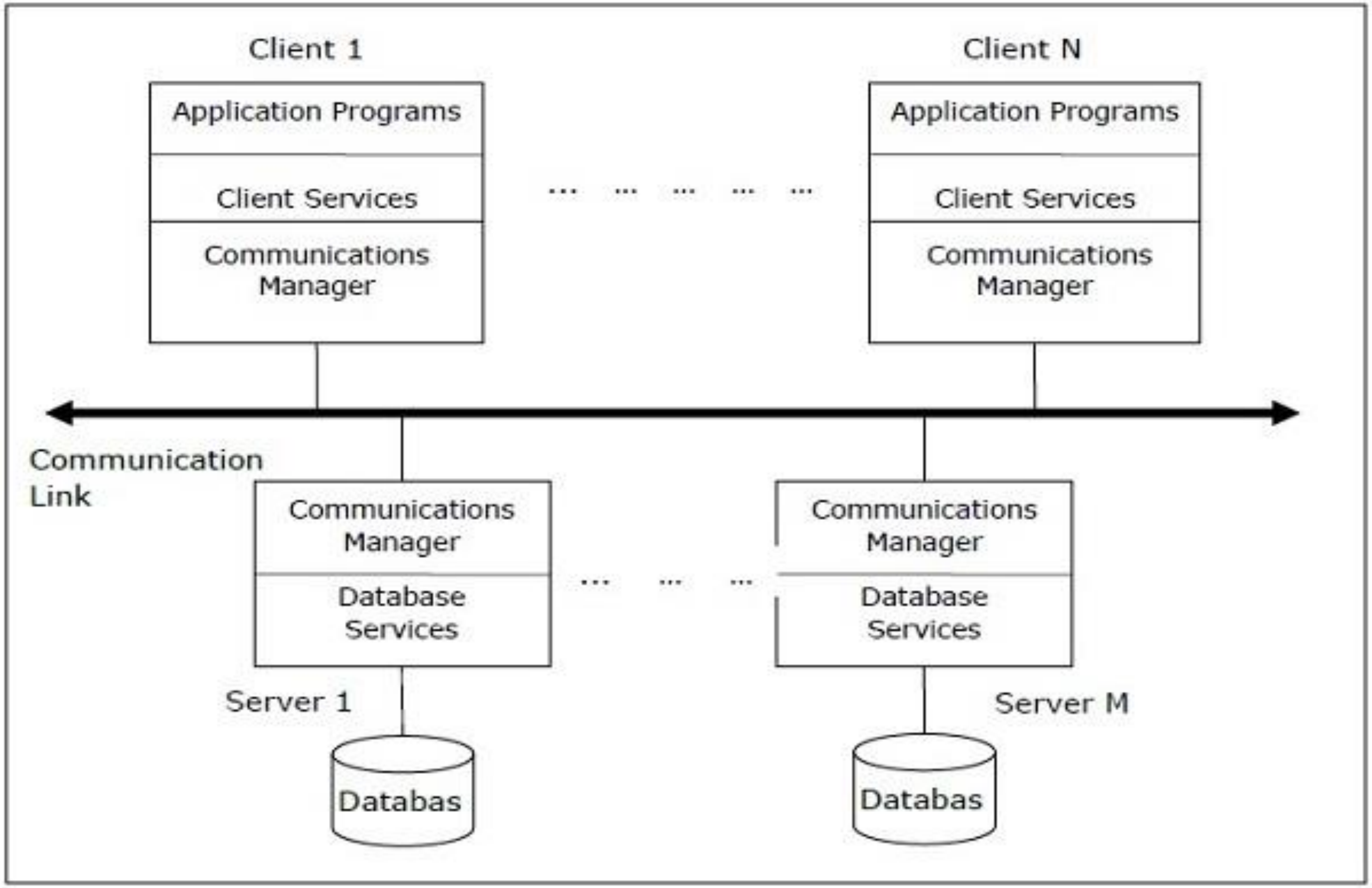


Multiple Server Multiple Client



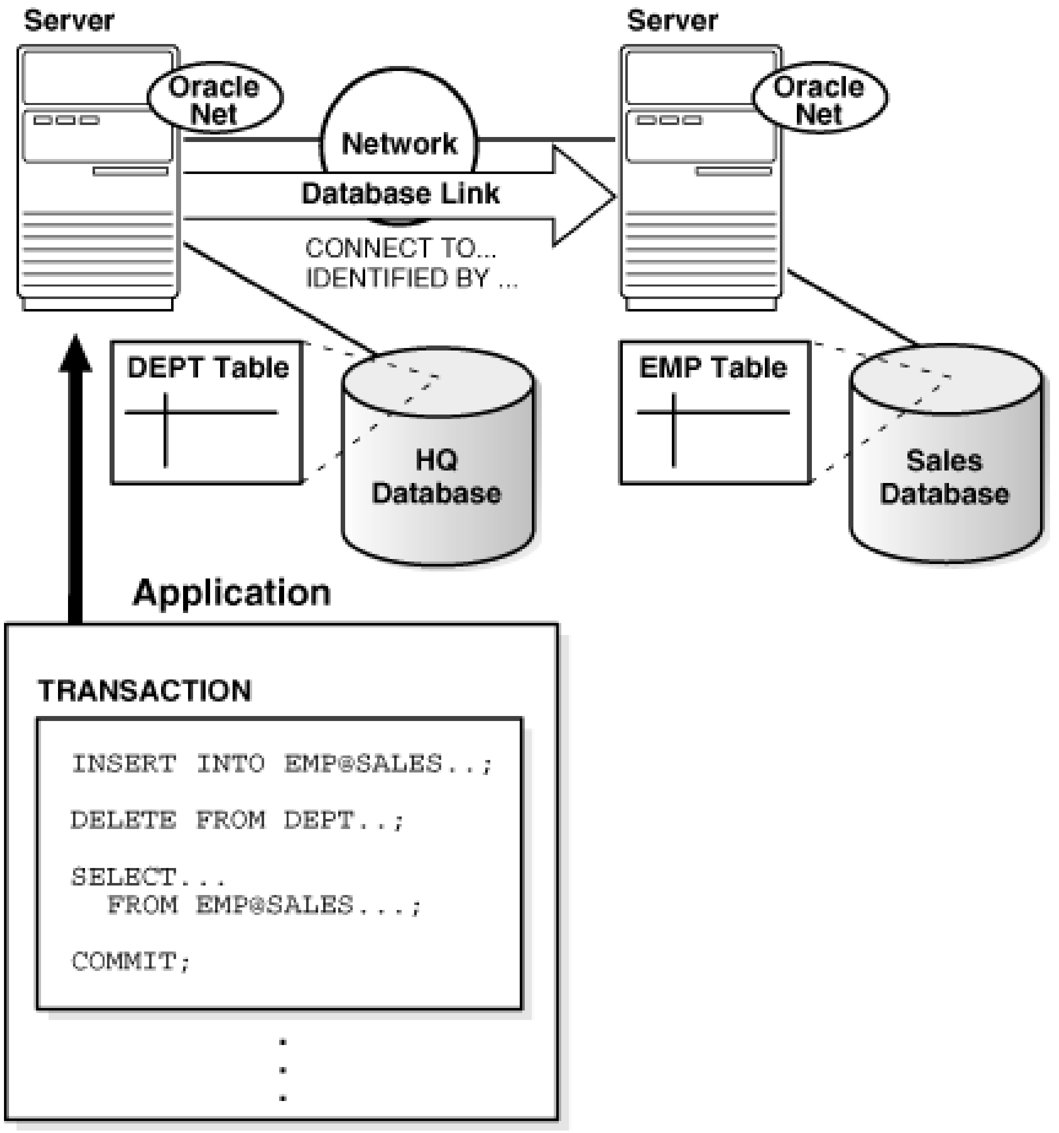
- directory
- caching
- query decomposition
- commit protocols

Multiple Server Multiple Client



Case study

- A database server is the Oracle software managing a database, and a client is an application that requests information from a server.
- Each computer in a network is a node that can host one or more databases. Each node in a distributed database system can act as a client, a server, or both, depending on the situation.
- In the figure, the host for the hq database is acting as a database server when a statement is issued against its local data (for example, the second statement in each transaction issues a statement against the local dept table), but is acting as a client when it issues a statement against remote data (for example, the first statement in each transaction is issued against the remote table emp in the sales database).



```
INSERT INTO EMP@SALES..;

DELETE FROM DEPT..;

SELECT...
  FROM EMP@SALES...;

COMMIT;
```

- A client can connect **directly** or **indirectly** to a database server. A direct connection occurs when a client connects to a server and accesses information from a database contained on that server. For example, if you connect to the hq database and access the dept table on this database as in the figure, you can issue the following:
- SELECT * FROM dept; This query is direct because you are not accessing an object on a remote database.
- In contrast, an indirect connection occurs when a client connects to a server and then accesses information contained in a database on a different server. For example, if you connect to the hq database but access the emp table on the remote sales database as in the figure, you can issue the following:
- SELECT * FROM emp@sales; This query is indirect because the object you are accessing is not on the database to which you are directly connected.