DISTRUBUTED DATABASE MANAGEMENT SYSTEM

UNIT- 1

Features of Distributed versus Centralized Databases Why Distributed Databases- Distributed Database Management Systems (DDBMSs) - Review of Databases - Review of Computer Networks- Levels of Distribution Transparency- Reference Architecture for Distributed Databases-Types of Data Fragmentation- Distribution Transparency for read-only Applications - Distribution transparency for Update Applications - Distributed Database Access Primitives-Integrity Constraints in Distributed Databases

Distributed Database System

A database is an ordered collection of related data that is built for a specific purpose. A database may be organized as a collection of multiple tables, where a table represents a real world element or entity. Each table has several different fields that represent the characteristic features of the entity.

For example, a company database may include tables for projects, employees, departments, products and financial records. The fields in the Employee table may be Name, Company_Id, Date_of_Joining, and so forth.

A database management system is a collection of programs that enables creation and maintenance of a database. DBMS is available as a software package that facilitates definition, construction, manipulation and sharing of data in a database. Definition of a database includes description of the structure of a database. Construction of a database involves actual storing of the data in any storage medium. Manipulation refers to the retrieving information from the database, updating the database and generating reports. Sharing of data facilitates data to be accessed by different users or programs.

A distributed database is basically a database that is not limited to one system, it is spread over different sites, i.e, on multiple computers or over a network of computers. A distributed database system is located on various sites that don't share physical components. This may be required when a particular database needs to be accessed by various users globally. It needs to be managed such that for the users it looks like one single database.

Examples of DBMS Application Areas

Automatic Teller Machines Train Reservation System Employee Management System Student Information System

Examples of DBMS Packages

MySQL

Oracle

SQL Server dBASE

FoxPro PostgreSQL, etc.

Database Schemas

A database schema is a description of the database which is specified during database design and subject to infrequent alterations. It defines the organization of the data, the relationships among them, and the constraints associated with them. Databases are often represented through the three-schema architecture or ANSISPARC architecture. The goal of this architecture is to separate the user application from the physical database. The three levels are

Internal Level having Internal Schema − It describes the physical structure, details of internal storage and access paths for the database.

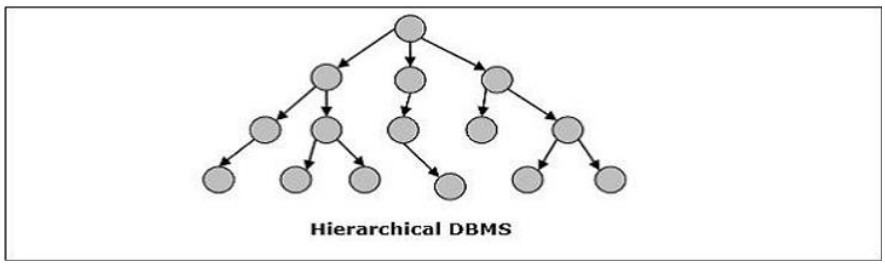
Conceptual Level having Conceptual Schema − It describes the structure of the whole database while hiding the details of physical storage of data. This illustrates the entities, attributes with their data types and constraints, user operations and relationships.

External or View Level having External Schemas or Views − It describes the portion of a database relevant to a particular user or a group of users while hiding the rest of database.

Types of DBMS

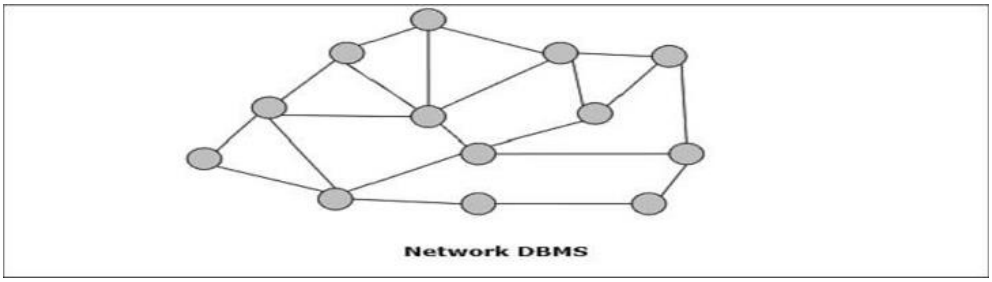
Hierarchical DBMS

In hierarchical DBMS, the relationships among data in the database are established so that one data element exists as a subordinate of another. The data elements have parent-child relationships and are modelled using the “tree” data structure. These are very fast and simple.



Network DBMS Network

DBMS in one where the relationships among data in the database are of type manyto-many in the form of a network. The structure is generally complicated due to the existence of numerous many-to-many relationships. Network DBMS is modelled using “graph” data structure.



Relational DBMS :

In relational databases, the database is represented in the form of relations. Each relation models an entity and is represented as a table of values. In the relation or table, a row is called a tuple and denotes a single record. A column is called a field or an attribute and denotes a characteristic property of the entity. RDBMS is the most popular database management system.

For example − A Student Relation −

Field

Tuple →	S_Id	Name	Year	Stream
	1	Ankit Jha	1	Computer Science
	2	Pushpa Mishra	2	Electronics
	5	Ranjini Iyer	2	Computer Science

Figure 1.3 A Student Relation

Object Oriented DBMS

Object-oriented DBMS is derived from the model of the object-oriented programming paradigm. They are helpful in representing both consistent data as stored in databases, as well as transient data, as found in executing programs. They use small, reusable elements called objects. Each object contains a data part and a set of operations which works upon the data. The object and its attributes are accessed through pointers instead of being stored in relational table models. For example − A simplified Bank Account object-oriented database

For example − A simplified Bank Account object-oriented database −

Field

Tuple →	S_Id	Name	Year	Stream
	1	Ankit Jha	1	Computer Science
	2	Pushpa Mishra	2	Electronics
	5	Ranjini Iyer	2	Computer Science

creditAmount( )
getBalance( )

Figure 1.4 A simplified Bank Account object-oriented database

Distributed DBMS

A distributed database is a set of interconnected databases that is distributed over the computer network or internet. A Distributed Database Management System (DDBMS) manages the distributed database and provides mechanisms so as to make the databases transparent to the users. In these systems, data is intentionally distributed among multiple nodes so that all computing resources of the organization can be optimally used.

A distributed database is a collection of multiple interconnected databases, which are spread physically across various locations that communicate via a computer network.

Features:

• Databases in the collection are logically interrelated with each other. Often they represent a single logical database.

• Data is physically stored across multiple sites. Data in each site can be managed by a DBMS independent of the other sites.

• The processors in the sites are connected via a network. They do not have any multiprocessor configuration.

• A distributed database is not a loosely connected file system.

• A distributed database incorporates transaction processing, but it is not synonymous with a transaction processing system.

Distributed Database Management System

A distributed database management system (DDBMS) is a centralized software system that manages a distributed database in a manner as if it were all stored in a single location.

Features

• It is used to create, retrieve, update and delete distributed databases.

• It synchronizes the database periodically and provides access mechanisms by the virtue of which the distribution becomes transparent to the users.

• It ensures that the data modified at any site is universally updated.

• It is used in application areas where large volumes of data are processed and accessed by numerous users simultaneously.

• It is designed for heterogeneous database platforms.

• It maintains confidentiality and data integrity of the databases.

Applications of Distributed Database:

• **Distributed Nature of Organizational Units** − Most organizations in the current times are subdivided into multiple units that are physically distributed over the globe. Each unit requires its own set of local data. Thus, the overall database of the organization becomes distributed.

• **Need for Sharing of Data** − The multiple organizational units often need to communicate with each other and share their data and resources. This demands common databases or replicated databases that should be used in a synchronized manner.

• **Support for Both OLTP and OLAP** − Online Transaction Processing (OLTP) and Online Analytical Processing (OLAP) work upon diversified systems which may have common data. Distributed database systems aid both these processing by providing synchronized data.

• **Database Recovery** − One of the common techniques used in DDBMS is replication of data across different sites. Replication of data automatically helps in data recovery if database in any site is damaged. Users can access data from other sites while the damaged site is being reconstructed. Thus, database failure may become almost inconspicuous to users.

• **Support for Multiple Application Software** − Most organizations use a variety of application software each with its specific database support. DDBMS provides a uniform functionality for using the same data among different platforms.

Advantages of Distributed Databases

• **Modular Development** − If the system needs to be expanded to new locations or new units, in centralized database systems, the action requires substantial efforts and disruption in the existing functioning. However, in distributed databases, the work simply requires adding new computers and local data to the new site and finally connecting them to the distributed system, with no interruption in current functions
. • **More Reliable** − In case of database failures, the total system of centralized databases comes to a halt. However, in distributed systems, when a component fails, the functioning of the system continues may be at a reduced performance. Hence DDBMS is more reliable.
• **Better Response** − If data is distributed in an efficient manner, then user requests can be met from local data itself, thus providing faster response. On the other hand, in centralized systems, all queries have to pass through the central computer for processing, which increases the response time.
• **Lower Communication Cost** − In distributed database systems, if data is located locally where it is mostly used, then the communication costs for data manipulation can be minimized. This is not feasible in centralized systems.

1) There is fast data processing as several sites participate in request processing.

2) Reliability and availability of this system is high.

3) It possess reduced operating cost.

4) It is easier to expand the system by adding more sites.

5) It has improved sharing ability and local autonomy.

Disadvantages of Distributed Database System :

1) The system becomes complex to manage and control.

2) The security issues must be carefully managed.

3) The system require deadlock handling during the transaction processing otherwise

the entire system may be in inconsistent state.

4) There is need of some standardization for processing of distributed database

system.

Architectures of Distributed Database:

It is used in Corporate Management Information System.

It is used in multimedia applications.

Used in Military's control system, Hotel chains etc.

It is also used in manufacturing control system.

The main advantage of a distributed database system is that it can provide higher availability and reliability than a centralized database system. Because the data is stored across multiple sites, the system can continue to function even if one or more sites fail. In addition, a distributed database system can provide better performance by distributing the data and processing load across multiple sites.

There are several different architectures for distributed database systems,

including:

Client-server architecture: In this architecture, clients connect to a central server, which manages the distributed database system. The server is responsible for coordinating transactions, managing data storage, and providing access control.

Peer-to-peer architecture: In this architecture, each site in the distributed database system is connected to all other sites. Each site is responsible for managing its own data and coordinating transactions with other sites.

Federated architecture: In this architecture, each site in the distributed database system maintains its own independent database, but the databases are integrated through a middleware layer that provides a common interface for accessing and querying the data.

Distributed Database Vs Centralized Database

Centralized DBMS	***Distributed DBMS***
In Centralized DBMS the database are stored in a only one site	In Distributed DBMS the database are stored in different site and help of network it can access it
If the data is stored at a single computer site,which can be used by multiple users	Database and DBMS software distributed over many sites,connected by a computer network
Database is maintained at one site	Database is maintained at a number of different sites

If centralized system fails,entire system is halted	If one system fails,system continues work with other site
It is a less reliable	It is a more reliable

Centralized database

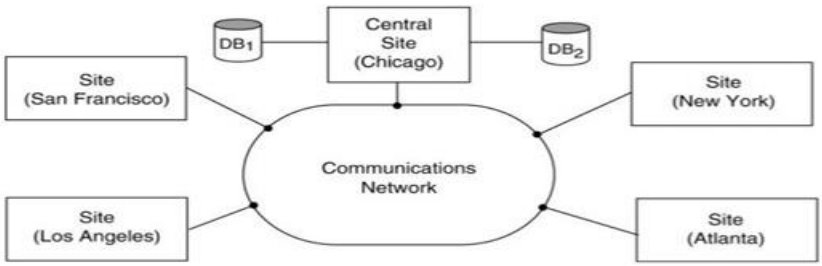


Figure 1.5 Centralized database

Distributed database

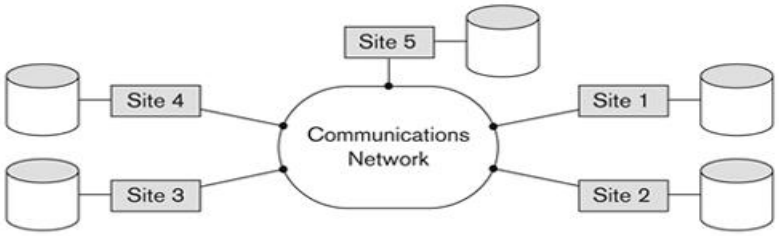


Figure1. 6 Distributed database

Types of Distributed Databases

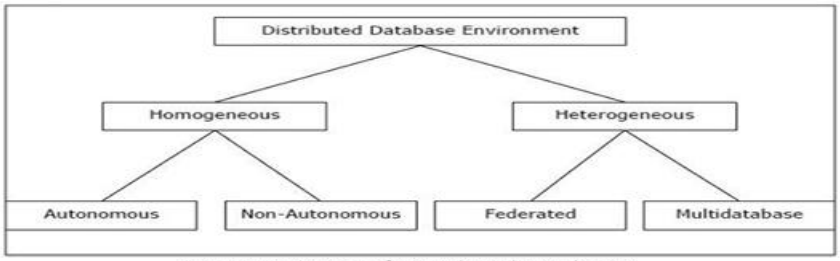


Figure 1.7 Types of Distributed Databases

Distributed database systems can be used in a variety of applications, **including e-commerce, financial services, and telecommunications. However, designing and managing a distributed database system can be complex and requires careful consideration of factors such as data distribution, replication, and consistency.**

Types:

1. Homogeneous Database:

In a homogeneous database, all different sites store database identically. The operating system, database management system, and the data structures used – all are the same at all sites. Hence, they're easy to manage.

Autonomous − Each database is independent that functions on its own. They are integrated by a controlling application and use message passing to share data updates.

Non-autonomous − Data is distributed across the homogeneous nodes and a central or master DBMS co-ordinates data updates across the sites.

2. Heterogeneous Database:

In a heterogeneous distributed database, different sites can use different schema and software that can lead to problems in query processing and transactions. Also, a particular site might be completely unaware of the other sites. Different computers may use a different operating system, different database application. They may even use different data models for the database. Hence, translations are required for different sites to communicate.

Federated − The heterogeneous database systems are independent in nature and integrated together so that they function as a single database system.

Un-federated − The database systems employ a central coordinating module through which the databases are accessed.

Distributed Data Storage:

There are 2 ways in which data can be stored on different sites. These are:

1. Replication –

In this approach, the entire relationship is stored redundantly at 2 or more sites. If the entire database is available at all sites, it is a fully redundant database. Hence, in replication, systems maintain copies of data.

This is **advantageous** as it increases the availability of data at different sites. Also, now query requests can be processed in parallel.

Data needs to be constantly updated. Any change made at one site needs to be recorded at every site that relation is stored or else it may lead to inconsistency. This is a lot of overhead. Also, concurrency control becomes way more complex as concurrent access now needs to be checked over a number of sites.

Advantages of Data Replication:

• **Reliability** − In case of failure of any site, the database system continues to work since a copy is available at another site(s).

• **Reduction in Network Load** − Since local copies of data are available, query processing can be done with reduced network usage, particularly during prime hours. Data updating can be done at non-prime hours.

• **Quicker Response** − Availability of local copies of data ensures quick query processing and consequently quick response time.

• **Simpler Transactions** − Transactions require less number of joins of tables located at different sites and minimal coordination across the network. Thus, they become simpler in nature.

Disadvantages of Data Replication:

• Increased Storage Requirements − Maintaining multiple copies of data is associated with increased storage costs. The storage space required is in multiples of the storage required for a centralized system.

• Increased Cost and Complexity of Data Updating − Each time a data item is updated, the update needs to be reflected in all the copies of the data at the different sites. This requires complex synchronization techniques and protocols.

• Undesirable Application – Database coupling − If complex update mechanisms are not used, removing data inconsistency requires complex co- ordination at application level. This results in undesirable application – database coupling.

Some commonly used replication techniques are

Snapshot replication

Near-real-time replication

Pull replication

2. Fragmentation –

In this approach, the relations are fragmented (i.e., they're divided into smaller parts) and each of the fragments is stored in different sites where they're required. It must be made sure that the fragments are such that they can be used to reconstruct the original relation (i.e, there isn't any loss of data).

Fragmentation is advantageous as it doesn't create copies of data, consistency is not a problem.

Fragmentation of relations can be done in two ways:

Horizontal fragmentation – Splitting by rows –

The relation is fragmented into groups of tuples so that each tuple is assigned to at least one fragment.

Vertical fragmentation – Splitting by columns –

The schema of the relation is divided into smaller schemas. Each fragment must contain a common candidate key so as to ensure a lossless join.

In certain cases, an approach that is hybrid of fragmentation and replication is used.

Distribution 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.

- 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 unfragmented. 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.

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.

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.

Levels of Distributed Transparency.

Outline

- Distribution transparency for read-only application.
- Distribution transparency for update application.

Distribution transparency for
read-only application

Objective

- We analyze with an example the different levels of distribution transparency:
  - Level 1: Fragmentation transparency.
  - Level 2: Location transparency.
  - Level 3: Local mapping transparency.
- For a ***read-only*** application.

Scenario

- Global schema:

  $SUPPLIER\ (SNUM, NAME, CITY)$

- Fragmentation schema:

  $SUPPLIER_1 = SL_{CITY = DHK}(SUPPLIER)$

  $SUPPLIER_2 = SL_{CITY = CTG}(SUPPLIER)$

- Allocation schema:

  $SUPPLIER_1$ @ site 1.

  $SUPPLIER_2$ @ site 2, 3.

Assume, a SUPINQUIRY application –

- Reading a value from terminal and assigning it to a variable:

```
read(terminal, v_SNUM);
```

- Query: Get *NAME* for a given *SNUM*. Example –

```
select NAME into v_NAME
from SUPPLIER[@siteNumber]
where SNUM = v_SNUM;
```

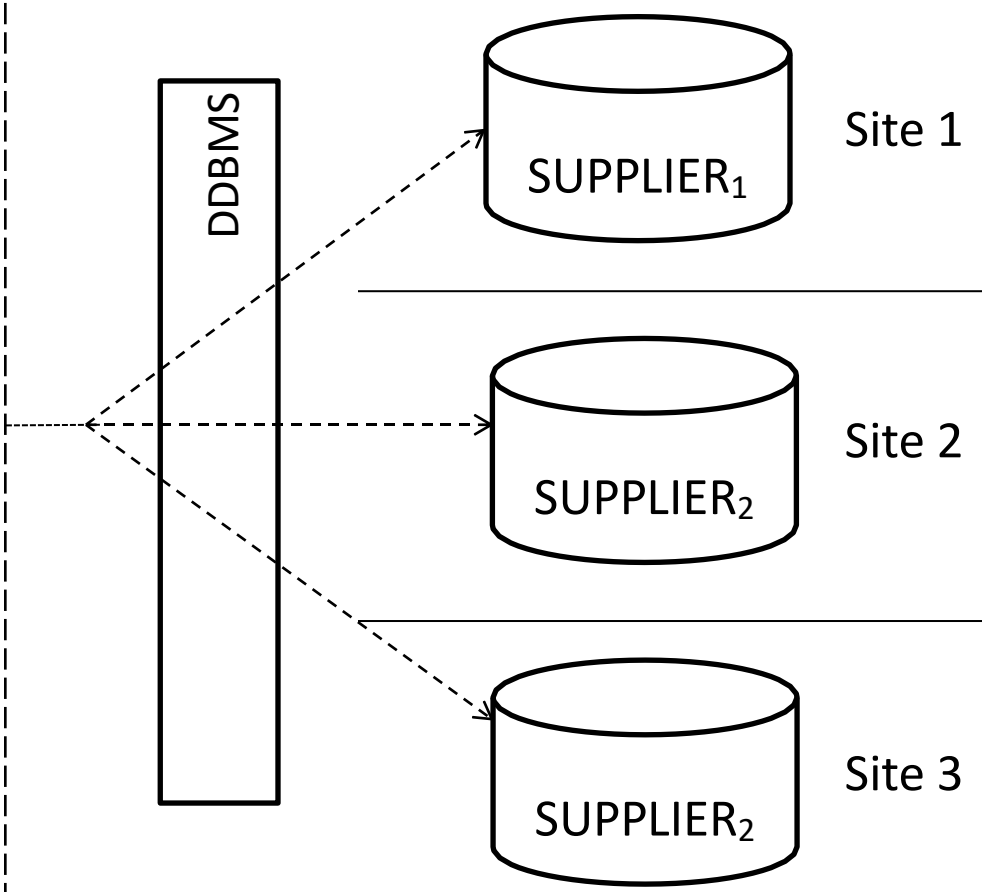
- Writing a value of a variable to terminal:

```
write(terminal, v_NAME);
```

SUPINQUIRY

Hint:

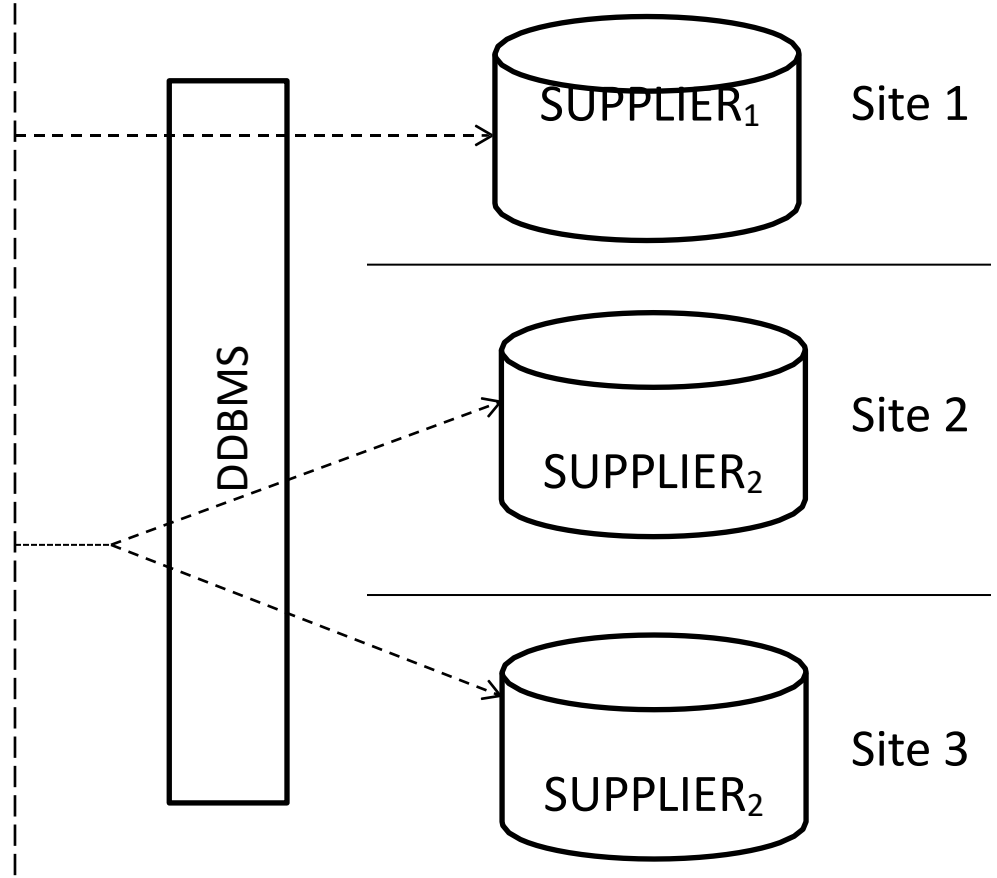
- Use global relation only. Because fragmentation information is hidden.



SUPINQUIRY

Hint:

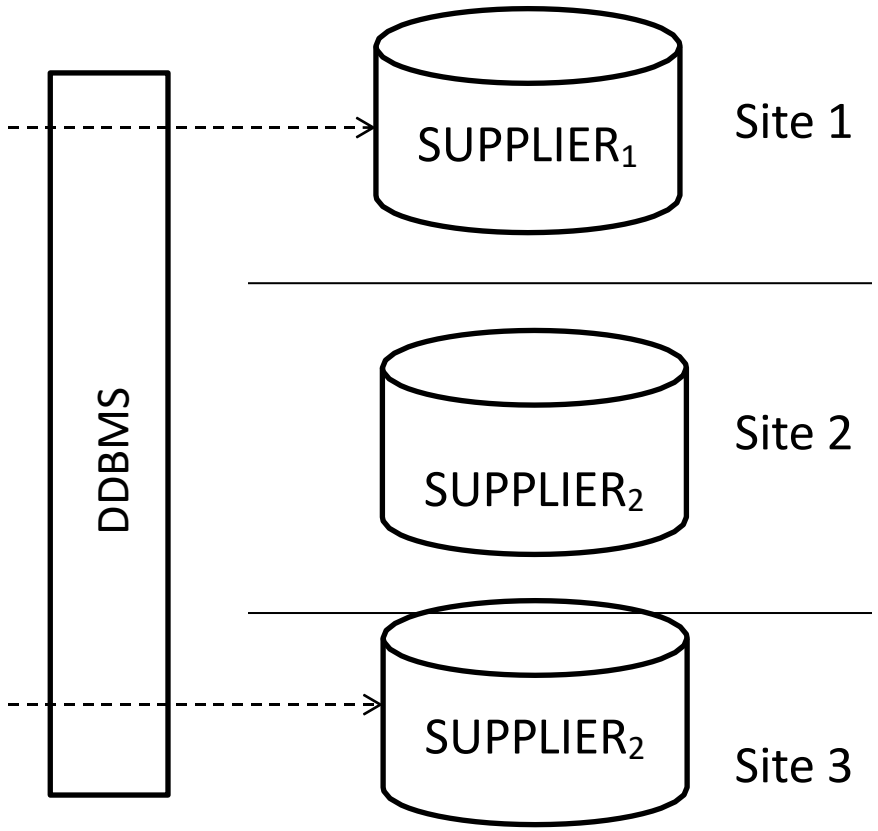
- Use fragmentation information.



SUPINQUIRY

Hint:

- Use fragmentation information + location information (i.e. site numbers).

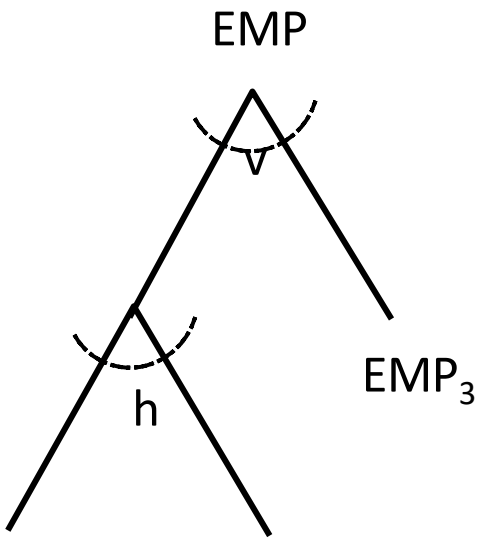


Distribution transparency for update application

Example:

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

Which part of the tree will be effected if *DEPTNUM* is updated?



EMP_1 EMP_2

Objective

- We analyze with an example the different levels of distribution transparency:
  - Level 1: Fragmentation transparency.
  - Level 2: Location transparency.
  - Level 3: Local mapping transparency.
- For an ***update*** application.

- Global schema:

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

- Fragmentation schema:

  $EMP_1 = PJ_{EMPNUM, NAME, SAL, TAX}\ SL_{DEPTNUM \le 10}\ (EMP)$

  $EMP_2 = PJ_{EMPNUM, MGRNUM, DEPTNUM}\ SL_{DEPTNUM \le 10}\ (EMP)$

  $EMP_3 = PJ_{EMPNUM, NAME, DEPTNUM}\ SL_{DEPTNUM > 10}\ (EMP)$

  $EMP_4 = PJ_{EMPNUM, SAL, TAX, MGRNUM}\ SL_{DEPTNUM > 10}\ (EMP)$

- Allocation schema:

  EMP_1 @ site 1, 5; EMP_2 @ site 2, 6

  EMP_3 @ site 3, 7; EMP_4 @ site 4, 8

Assume, a `UPDTEMP` application –

Updating *DEPTNUM* to 15 where *EMPNUM* is *100.*

Example –

```
update EMP [@siteNumber]
set DEPTNUM = 15
where EMPNUM = 100;
```

Hint: Use global relation. No concept of fragments.

Analyzing Level – 2 transparency

Hints: Use fragments.

- Use the concept of *update sub-tree*.
- Follow the *effect of update.*

Effect of Update

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

EMP₁

EMPNUM	*NAME*	*SAL*	*TAX*
100	Smith	10000	1000

EMP₂

EMPNUM	*MGRNUM*	*DEPTNUM*
100	20	3

Effect of updating *DEPTNUM =15* with *EMPNUM = 100*

Effect of Update (cont.)

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

EMP_1

EMPNUM	NAME	SAL	TAX
100	Smith	10000	1000

EMP_2

EMPNUM	MGRNUM	DEPTNUM
100	20	**3**

15 ?

Effect of updating *DEPTNUM =15* with *EMPNUM = 100*

Effect of Update (cont.)

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

EMP_1

EMPNUM	NAME	SAL	TAX
100	Smith	10000	1000

EMP_2

EMPNUM	MGRNUM	DEPTNUM
100	20	3

EMP_3

EMPNUM	NAME	DEPTNUM
		15

EMP_4

EMPNUM	SAL	TAX	MGRNUM

Effect of updating *DEPTNUM =15* with *EMPNUM = 100*

Effect of Update (cont.)

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

EMP_1

EMPNUM	NAME	SAL	TAX
100	Smith	10000	1000

EMP_2

EMPNUM	MGRNUM	DEPTNUM
100	20	3

EMP_3

EMPNUM	NAME	DEPTNUM
100	Smith	**15**

EMP_4

EMPNUM	SAL	TAX	MGRNUM
100	10000	1000	20

Effect of updating *DEPTNUM =15* with *EMPNUM = 100*

Effect of Update (cont.)

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

EMP_1

EMPNUM	NAME	SAL	TAX
~~100~~	~~Smith~~	~~10000~~	~~1000~~

EMP_2

EMPNUM	MGRNUM	DEPTNUM
~~100~~	~~20~~	~~3~~

EMP_3

EMPNUM	NAME	DEPTNUM
100	Smith	15

EMP_4

EMPNUM	SAL	TAX	MGRNUM
100	10000	1000	20

Effect of updating *DEPTNUM =15* with *EMPNUM = 100*

Analyzing Level – 2 transparency (cont.)

Hints: Use fragments. Use the *update sub-tree*. Follow the *effect of update.*

- Store the necessary record from EMP_1 and EMP_2 to temporary variables.
- Insert the records into EMP_3 and EMP_4 from the temporary variables.
- Delete the records from EMP_1 and EMP_2.

Hints: Use fragments + locations. Follow the *effect of update* (like previous level), but this time locations will be considered.

- Store the necessary record from EMP_1 and EMP_2 from any of the *corresponding sites* to temporary variables.
- Insert the records into EMP_3 and EMP_4 at *corresponding sites* from the temporary variables.
- Delete the records from EMP_1 and EMP_2 at *corresponding sites*.

Additional Reading

- Level – 4 transparency.
- Distribution transparency for a *more complex* read- only application.

Practice Problems/ Questions

a) For the example provided in the lecture slides, determine the effect of updating *DEPTNUM = 5* where *EMPNUM = 100 (assume, the record is initially found in* EMP_3 *and* EMP_4 *with DEPTNUM = 19).*

b) Create your own scenario and analyze the different levels of distribution transparency for read-only and update application.

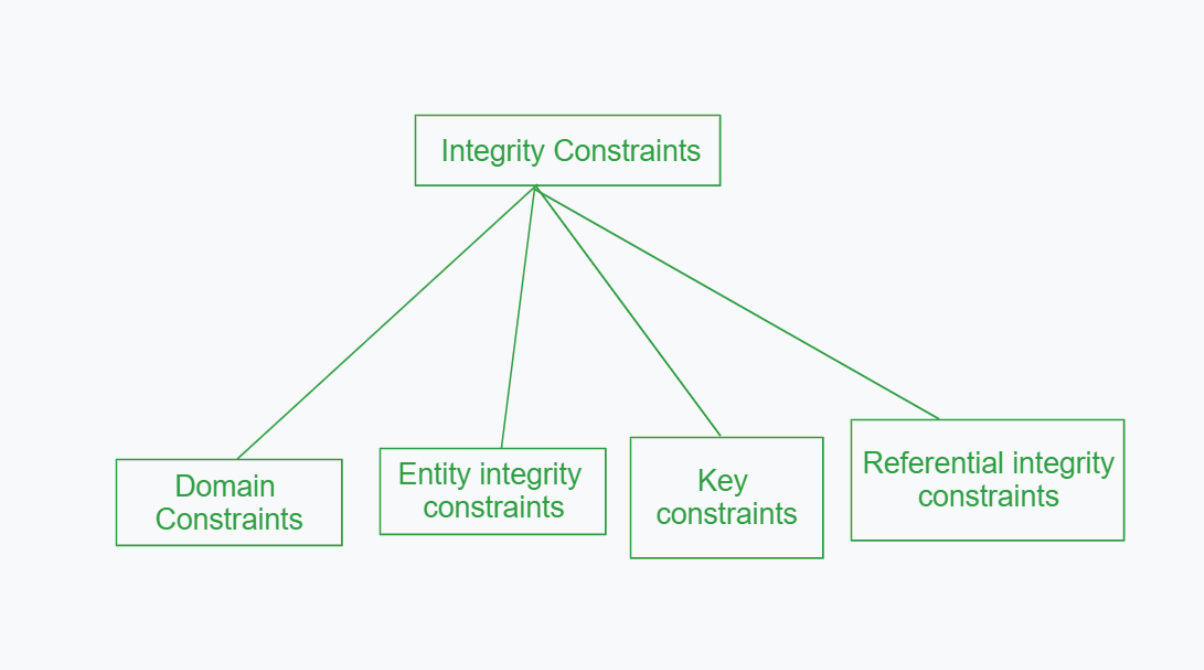
DBMS Integrity Constraints

Integrity constraints are the set of predefined rules that are used to maintain the quality of information. Integrity constraints ensure that the data insertion, data updating, data deleting and other processes have to be performed in such a way that the data integrity is not affected. They act as guidelines ensuring that data in the database remain accurate and consistent. So, integrity constraints are used to protect databases. The various types of integrity constraints are

Types of Integrity Constraints

There are four types of integrity constraints which are:

- Domain Constraints
- Entity integrity Constraints
- Key Constraints
- Referential integrity constraints



Domain Constraints

These are defined as the defination of valid set of values for an atribute. The data type of domain include string, char, time, integer, date, currency etc. The value of the attribute must be avialable in comparable domains.

Example:

Student_Id	Name	Semester	Age
21CSE100	Ramesh	5th	20
21CSE101	Kamlesh	5th	21
21CSE102	Aakash	5th	22
21CSE103	Mukesh	5th	20

Entity Integrity Constraints

Entity integrity constraints state that primary key can never contain null value because primary key is used to determine individual rows in a relation uniquely, if primary key contains null value then we cannot identify those rows. A table can contain null value in it except primary key field.

Example:

It is not allowed because it is containing primary key as NULL value.

Student_id	Name	Semester	Age
21CSE101	Ramesh	5th	20
21CSE102	Kamlesh	5th	21
21CSE103	Aakash	5th	22
	Mukesh	5th	20

Key Constraints

Keys are the entity set that are used to identify an entity within its entity set uniquely. An entity set can contain multiple keys, bit out of them one key will be primary key. A primary key is always unique, it does not contain any null value in table.

Example:

Student_id	Name	Semester	Age
21CSE101	Ramesh	5th	20

Student_id	Name	Semester	Age
21CSE102	Kamlesh	5th	21
21CSE103	Aakash	5th	22
21CSE102	Mukesh	5th	20

It is now acceptable because all rows must be unique.

Referential integrity constraints

It can be specified between two tables. In case of referential integrity constraints, if a Foreign key in Table 1 refers to Primary key of Table 2 then every value of the Foreign key in Table 1 must be null or avialable in Table 2.

Example:

Here, in below example Block_No 22 entry is not allowed because it is not present in 2nd table.

Student_id	Name	Semester	Block_No
22CSE101	Ramesh	5th	20
21CSE105	Kamlesh	6th	21
22CSE102	Aakash	5th	20
23CSE106	Mukesh	2nd	22

Block_No	Block Location
20	Chandigarh
21	Punjab
25	Delhi

Conclusion

Integrity constraints act as the backbone of reliable and robust database. They ensure that the data stored is reliable, worthy, consistent and accurate

within the database. By implement integrity constraints we can improve the quality of the data stored in database. So, as the database continues to grow it will not become inconsistent and inaccurate.