CS9152 – DATABASE TECHNOLOGY

UNIT – I

DISTRIBUTED DATABASES

TEXT BOOK

1. Elisa Bertino, Barbara Catania, Gian Piero Zarri, "Intelligent Database Systems", Addison-Wesley, 2001.

REFERENCES

1. Carlo Zaniolo, Stefano Ceri, Christos Faloustsos, R.T.Snodgrass, V.S.Subrahmanian, "Advanced Database Systems", Morgan Kaufman, 1997.
2. N.Tamer Ozsu, Patrick Valduriez, "Principles of Distributed Database Systems", Prentice Hal International Inc. , 1999.
3. C.S.R Prabhu, "Object-Oriented Database Systems", Prentice Hall Of India, 1998.
4. Abdullah Uz Tansel Et Al, "Temporal Databases: Theory, Design And Principles",Benjamin Cummings Publishers , 1993.
5. Raghu Ramakrishnan, Johannes Gehrke, "Database Management Systems", Mcgraw Hill, Third Edition, 2004.
6. Henry F Korth, Abraham Silberschatz, S. Sudharshan, "Database System Concepts", Fourth Ediion, McGraw Hill , 2002.
7. R. Elmasri, S.B. Navathe, "Fundamentals of Database Systems", Pearson Education, 2004.

DISTRIBUTED DATABASES

Syllabus:

UNIT I DISTRIBUTED DATABASES 5
Distributed Databases Vs Conventional Databases – Architecture – Fragmentation – Query Processing – Transaction Processing – Concurrency Control – Recovery.

Table of Contents

Topic – 1: Introduction to Distributed Databases

Distributed Databases- Definition

What is a distributed database?

- "A logically interrelated collection of shared data (and a description of this data), physically distributed over a computer network"

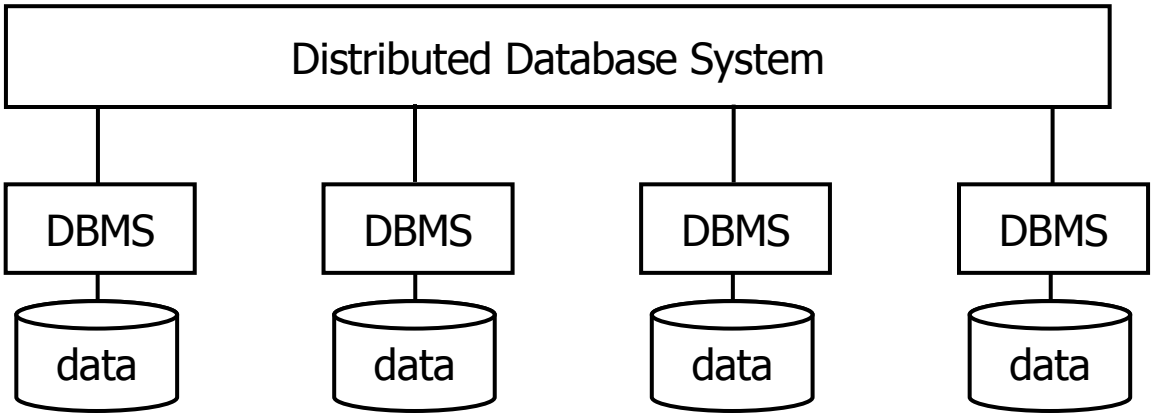
(DDBMS) is the software that manages the DDB and provides an access mechanism that makes this distribution transparent to the users.

A database that consists of two or more data files located at different sites on a computer network. Because the database is distributed, different users can access it without interfering with one another. However, the *DBMS* must periodically synchronize the scattered databases to make sure that they all have consistent data.

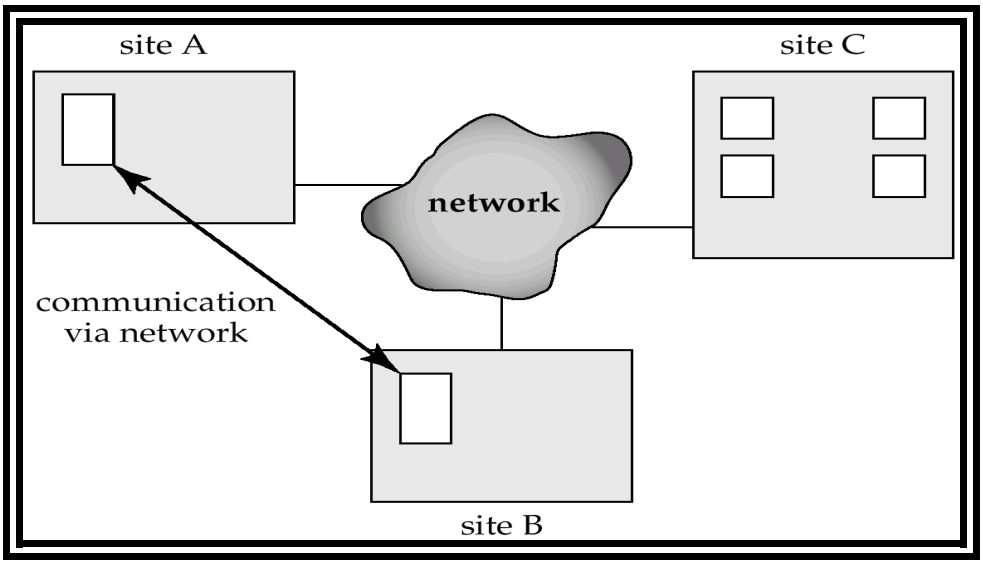
*DDBMS to Avoid `**islands of information'** problem…*

A "Distributed Database": is a logically interrelated collection of shared data (and a description of this data), physically distributed over a computer network.

A "Distributed DBMS" (DDBMS): is a Software system that permits the management of the distributed database and makes the distribution transparent to users.



Fundamental Principle: make distribution transparent to user.

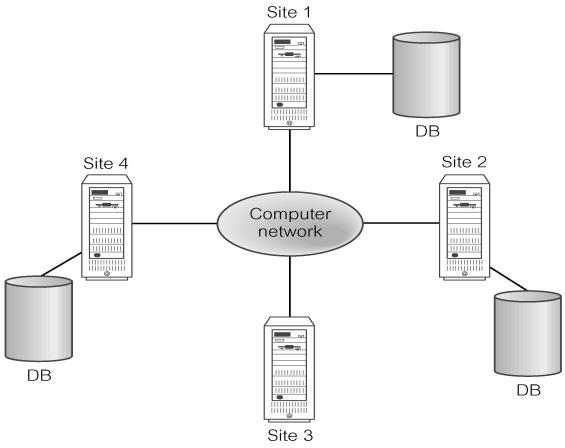


- In distributed databases, data spread over multiple machines (also referred to as sites or nodes.
- Network interconnects the machines
- Data shared by users on multiple machines

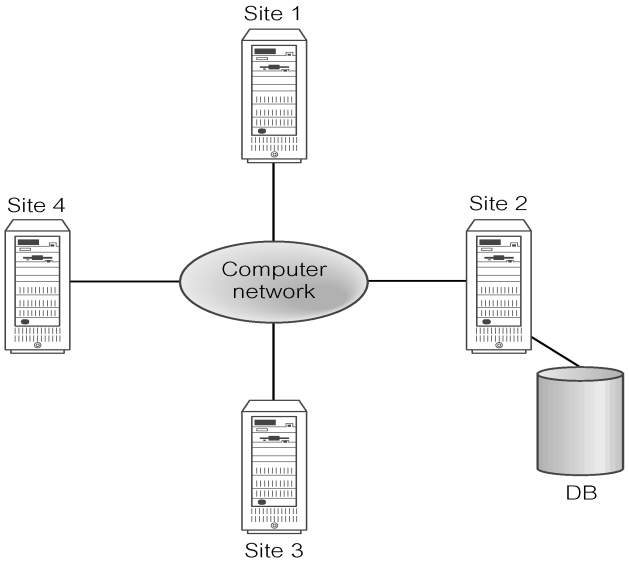
DDBMS has following characteristics:

- Collection of logically-related shared data.
- Data split into fragments.
- Fragments may be replicated.
- Fragments/replicas allocated to sites.
- Sites linked by a communication network.
- Data at each site is under control of a DBMS.
- DBMSs handle local applications autonomously.
- Each DBMS participates in at least one global application.

Important difference between DDBMS and distributed processing



Distributed processing of centralised DBMS



Distributed Processing

- Much more tightly coupled than a DDBMS.
- Database design is same as for standard DBMS
- No attempt to reflect organizational structure
- Much simpler than DDBMS
- More secure than DDBMS
- No local autonomy

Functions of a DDBMS

- Expect DDBMS to have at least the functionality of a DBMS.

Also to have following functionality:

- Extended communication services.
- Extended Data Dictionary.
- Distributed query processing.
- Extended concurrency control.
- Extended recovery services

Additional Functions Provided:

• Ability to access remote sites and transmit queries and data among various sites via a communication network.
• Ability to track of the data distribution and replication in DDBMS catalog.
• Ability to devise execution strategies for queries and transaction that access data from more than one site.
• Ability to decide on which copy of a replicated data item to access.
• Ability to maintain the consistency of copies of a replicated data item.
• Ability to recover from individual site crashes and from new types of failures such as the failure of a communication link.

Hardware Level:

Multiple computers, called sites or nodes.
Communication network.
Local Area Network
Long-haul Network
Network Topologies

Advantages of DDBSs:

• Distributed nature of some database applications
• Local users
• Global users
• Increased reliability and availability
• Reliability - Probability that a system is up at a particular moment.
• Availability – Probability that the system is continuously available during a time interval.
• Improvement is achieved by replicating data and software at more than one site.
• Allowing data sharing while maintaining some measure of local control
• Controlled sharing of data throughout the distributed system.
• Improved Performance
• Smaller database exists at a single site.
• Accesses to more than one site proceed in parallel.
• Reduced response time.

Advantages of distributed databases

- Capacity and incremental growth
- Increase reliability and availability
- Modularity
- Reduced communication overhead
- Protection of valuable data
- Efficiency and Flexibility
- Reflects organizational structure
- Improved shareability and local autonomy
- Improved availability
- Improved reliability
- Improved performance
- Economics
- Modular growth

Disadvantages of distributed databases

- Complexity
- Cost
- Security
- Integrity control more difficult
- Lack of standards
- Lack of experience
- Database design more complex or Increased complexity in system design and implementation.

Applications of DDBMS:

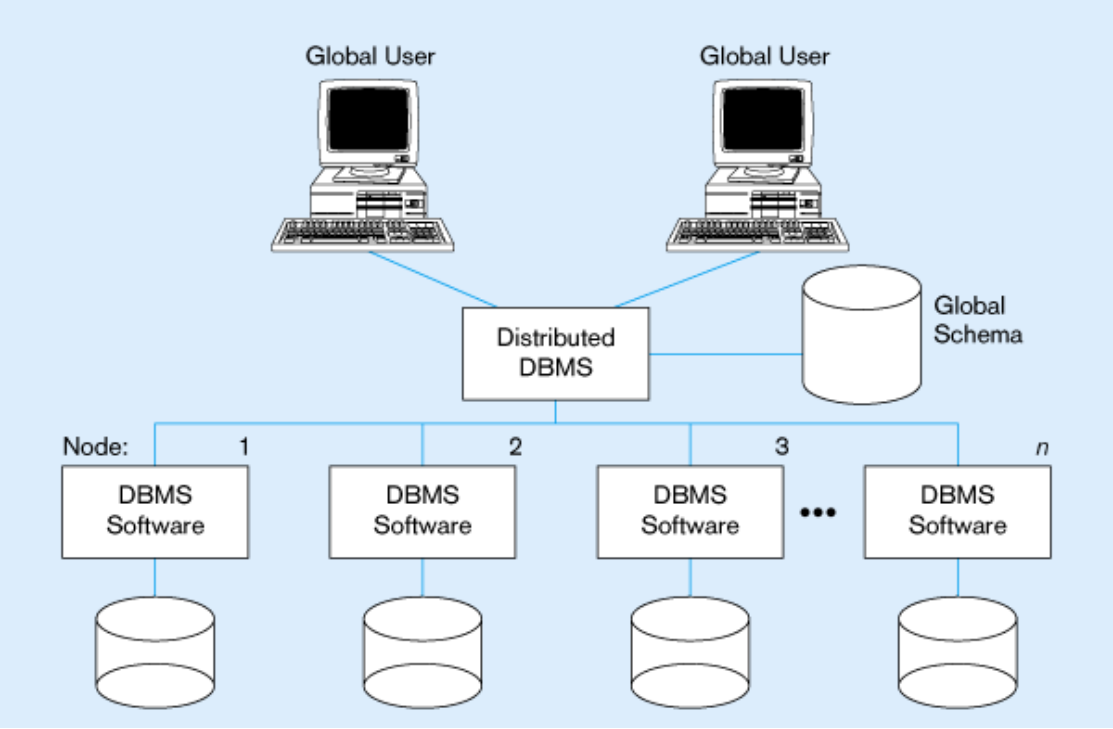
- Manufacturing - especially multi-plant manufacturing
- Military command and control
- Electronic fund transfers and electronic trading
- Corporate MIS
- Airline restrictions
- Hotel chains
- Any organization which has a decentralized organization structure
- User access the distributed data via applications.
  Two types of applications:

  - Logical Applications: Applications that do not required data from other sites.

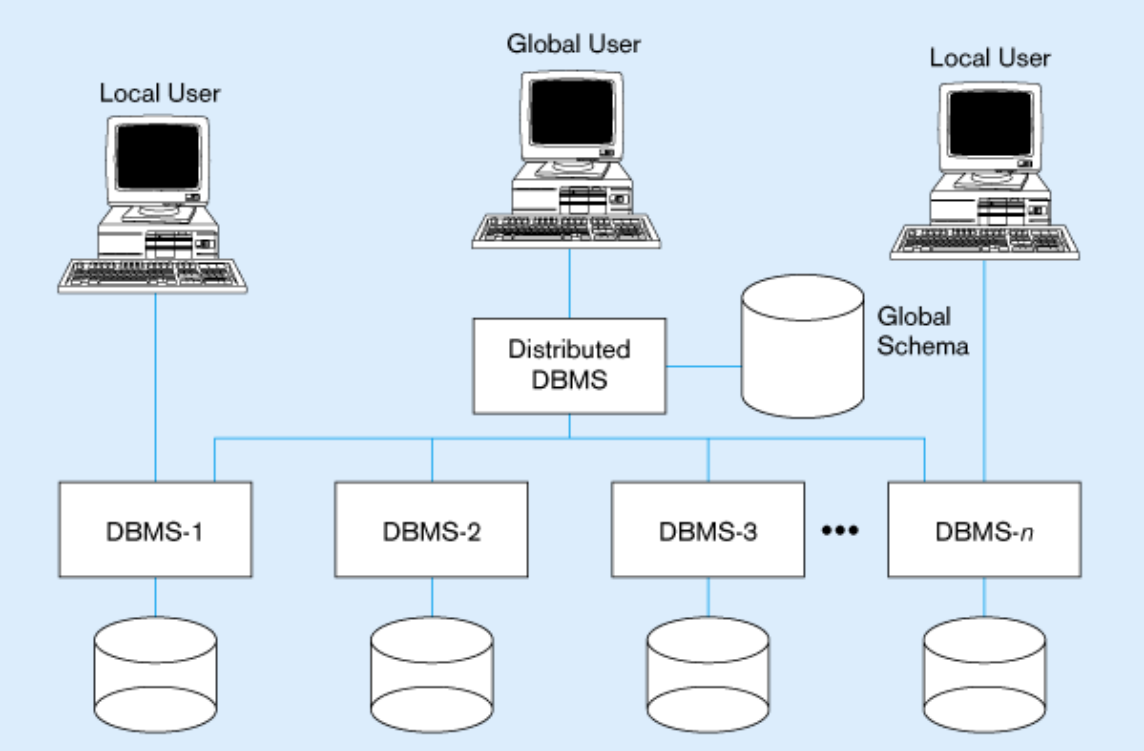
  - Physical Applications: Applications that required data from other sites.

Types of DDBMS

- In a homogeneous distributed database:
  - All sites have identical software.
  - Are aware of each other and agree to cooperate in processing user requests.
  - Each site surrenders part of its autonomy in terms of right to change schemas or software.
  - Appears to user as a single system.



- In a heterogeneous distributed database:
  - Different sites may use different schemas and software.
    - Difference in schema is a major problem for query processing.
    - Difference in software is a major problem for transaction processing.
  - Sites may not be aware of each other and may provide only limited facilities for cooperation in transaction processing.



Two main issues in DDBMS

- Making query from one site to the same or remote site.
- Logical database is partitioned in to different data streams and located at different sites.

Topic – 2: Distributed Databases Vs Conventional Databases

- mimics organisational structure with data
- local access and autonomy without exclusion
- cheaper to create and easier to expand
- improved availability/reliability/performance by removing reliance on a central site
- Reduced communication overhead

- Most data access is local, less expensive and performs better
  - Improved processing power
- Many machines handling the database rather than a single server
  - more complex to implement

- more costly to maintain
- security and integrity control
- standards and experience are lacking
- Design issues are more complex

Centralized Database System:

- All system components (data. DBMS software, secondary storage devices and tapes for backup) reside at a single site.
- Remote access via terminals connected to the site is possible.

• **Distributed Database:**

- Physically spread over the sites of a computer network.
- Communication network
- Distributed Database Systems (DDBSs)
- Distributed Database Management System (DDBMS)

Topic – 3: Distributed Databases Architecture

Overview of Client-Server Architecture
• 3 levels of DDBMS software modules:
• Client - Applicatrion Processor (AP) or front-end machine
• Server – Database Processor (DP) or back-end machine
• Communications software
• Reference to DDBMS catalog by client
• Query processing:
• Client parses a query and decomposes it into a number of independent site queries. Each site query is sent to the appropriate server site.
• Each server processes the local query and sends the resulting relation to the client site.
• Client site combines the results of the subqueries to produce the result of the originally submitted query.

Overview of Distributed Database Architecture

- Location Transparency
  - User does not have to know the location of the data.
  - Data requests automatically forwarded to appropriate sites

- Local Autonomy

- Local site can operate with its database when network connections fail
- Each site controls its own data, security, logging, recovery

- Synchronous Distributed Database
  - All copies of the same data are always identical
  - Data updates are immediately applied to all copies throughout network
  - Good for data integrity
  - High overhead ⟨ slow response times

- Advantages
  - Increased reliability & availability
  - Local control
  - Modular growth
  - Lower communication costs
  - Faster response

- Disadvantages
  - Software cost & complexity
  - Processing overhead
  - Data integrity
  - Slow response

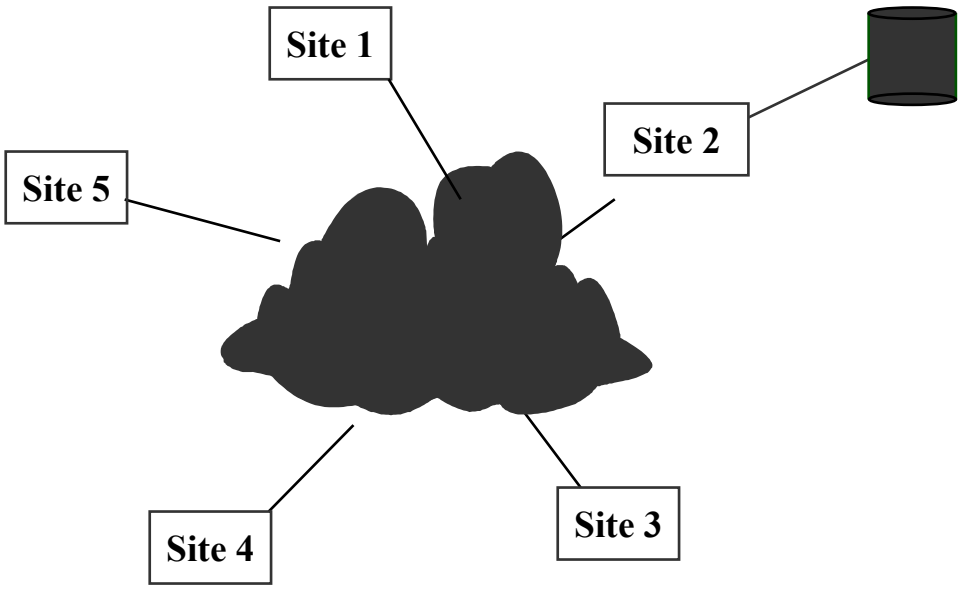
- Asynchronous Distributed Database
  - Some data inconsistency is tolerated
  - Data update propagation is delayed
  - Lower data integrity
  - Less overhead ⟨ faster response time

  - Defines the structure of the system
    - components identified
    - functions of each component defined
    - interrelationships and interactions between components defined

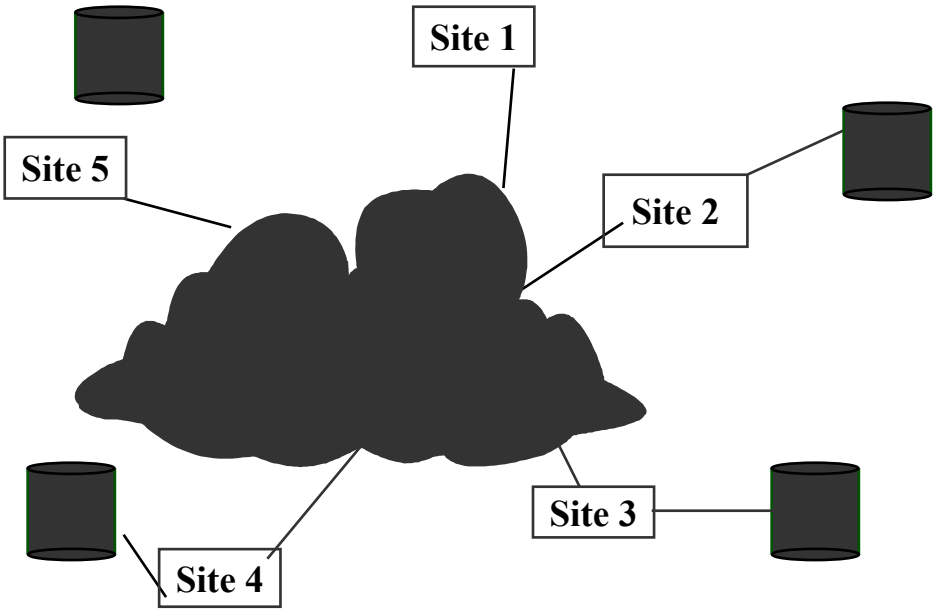
DDBS = DB + Communication

- non-centralised
- DDBMS
  - Motivated by need to integrate operational data and to provide controlled access
  - manages the Distributed database
  - makes the distribution transparent to the user

Centralized DBMS on a Network



Distributed DBMS Environment



Implicit Assumptions

- Data stored at a number of sites | each site *logically* consists of a single processor.
- Processors at different sites are interconnected by a computer network | no multiprocessors
  - parallel database systems
- Distributed database is a database, not a collection of files | data logically related as exhibited in the users' access patterns
  - relational data model
- D-DBMS is a full-fledged DBMS
  - not remote file system, not a TP system

Dimensions of the Problem

- Distribution
  - Whether the components of the system are located on the same machine or not
- Heterogeneity
  - Various levels (hardware, communications, operating system)
  - DBMS important one
    - data model, query language,transaction management algorithms
- Autonomy
  - Not well understood and most troublesome
  - Various versions
    - Design autonomy: Ability of a component DBMS to decide on issues related to its own design.
    - Communication autonomy: Ability of a component DBMS to decide whether and how to communicate with other DBMSs.
    - Execution autonomy: Ability of a component DBMS to execute local operations in any manner it wants to.

Issues of a DDBMS

- Data Allocation
  - Where to locate data and whether to replicate?
- Data Fragmentation
  - Partition the database
- Distributed catalog management
- Distributed transactions

- Distributed Queries
- Making all of the above *transparent* to the user is the key of DDBMS's

Replication or Data Replications in DDBMS

What Is Replication?

Replication is the process of copying and maintaining database objects in multiple databases that make up a distributed database system.

Changes applied at one site are captured and stored locally before being forwarded and applied at each of the remote locations.

Replication provides user with fast, local access to shared data, and protects availability of applications because alternate data access options exist. Even if one site becomes unavailable, users can continue to query or even update the remaining locations.

Replication Objects, Groups, and Sites

The following sections explain the basic components of a replication system, including replication sites, replication groups, and replication objects.

Replication Objects

A replication object is a database object existing on multiple servers in a distributed database system.

Oracle's replication facility enables you to replicate tables and supporting objects such as views, database triggers, packages, indexes, and synonyms. `SCOTT.EMP` and `SCOTT.BONUS` are examples of replication objects.

Replication Groups

- ✓ In a replication environment, Oracle manages replication objects using replication groups.
- ✓ By organizing related database objects within a replication group, it is easier to administer many objects together
- ✓ create and use a replication group to organize the schema objects necessary to support a particular database application. That is not to say that replication groups and schemas must correspond with one another.
- ✓ Objects in a replication group can originate from several database schemas and a schema can contain objects that are members of different replication groups.
- ✓ The restriction is that a replication object can be a member of only one group.

- Database replication is the frequent electronic copying data from a database in one computer or server to a database in another so that all users share the same level of information.
- The result is a distributed database in which users can access data relevant to their tasks without interfering with the work of others.
- If a site (or network path) fails, the data held there is unavailable
- Consider replication (duplication) of data to improve availability
  - ***No replication:***
  - Disjoint fragments
  - ***Partial replication:***
  - Site dependent
  - ***Full replication:***
  - Every site has copy of all data
  - slows down update for consistency
  - expensive

Fragments of relations are placed across the sites multiple times

- ✓ increases reliability - if some sites fail, the data is still available
- ✓ increases locality - the data can be retrieved from the closest or local site
- ✓ increases performance - a certain fragment may be accessed by less users

• but the question
– of mutual consistency
– concurrency control
- transparency must be addressed

• A DB can be
– ***partitioned (no replication)***
– replicated
• fully replicated - the whole DB is copied to each site
• partially replicated

What typical units of data are replicated in the process of data replication in DDBMS?

✓ Data may be replicated row by row, table by table, or database by database, depending on what you need.

If you need to be able to operate independently, then you have a number of replication models to choose from, depending on how much latency is acceptable and how much autonomy is required.

- ✓ Microsoft servers have a replication model that allows complete autonomy... but could theoretically break atomicity.

    - o A distributed database is not stored in its entirety at a single physical location. Instead, it is spread across a network of computers that are geographically dispersed and connected via communications links.

    - o A distributed database allows faster local queries and can reduce network traffic. With these benefits comes the issue of maintaining data integrity.

    - o A key objective for a distributed system is that it looks like a centralized system to the user. The user should not need to know where a piece of data is stored physically.

Comparison of Replication Alternatives:

	Full Replication	**Partial Replication**	**Partitioning**
Query Processing	**Easy**	**Same**	**Difficulty**
Directory Management	**Easy or nonexistent**	**Same**	**Difficulty**
Concurrency Control	**Moderate**	**Difficult**	**Easy**
Reliability	**Very High**	**High**	**Low**
Reality	**Possible Application**	**Realistic**	**Possible application**

Topic – 4: Fragmentation

Concept

- Division of relation r into fragments *r*1, *r*2, …, *rn* which contain sufficient information to reconstruct relation r.
- Horizontal fragmentation: each tuple of *r* is assigned to one or more fragments.
- Vertical fragmentation: the schema for relation *r* is split into several smaller schemas.
    - All schemas must contain a common candidate key (or superkey) to ensure lossless join property.

- A special attribute, the tuple-id attribute may be added to each schema to serve as a candidate key.

- Example : relation account with following schema.
- *Account-schema* = (*branch-name*, *account-number, balance*).

Distribution Design Issues

- Why fragment at all?
- How to fragment?
- How much to fragment?
- How to test correctness?
- How to allocate?
- Information requirements?
- Can't we just distribute relations?
- What is a reasonable unit of distribution?
  - relation
    - views are subsets of relations ê locality
    - extra communication
  - fragments of relations (sub-relations)
    - concurrent execution of a number of transactions that access different portions of a relation
    - views that cannot be defined on a single fragment will require extra processing
    - semantic data control (especially integrity enforcement) more difficult

Why fragment?

Usage:

- Apps work with views rather than entire relations.

Efficiency:

- Data stored close to where most frequently used.
- Data not needed by local applications is not stored.

Security:

- and so not available to unauthorized users.

Parallelism:

- With fragments as unit of distribution, T can be divided into several subqueries that operate on fragments.

Disadvantages: Performance & Integrity.

Types of Fragmentation

a) Horizontal Fragmentation (HF)
   - splitting the database by rows

- e.g. A-J in site 1, K-S in site 2 and T-Z in site 3
  - Primary Horizontal Fragmentation (PHF)
  - Derived Horizontal Fragmentation (DHF)

b) Vertical Fragmentation (VF)
- Splitting database by columns/fields
- e.g. columns/fields 1-3 in site A, 4-6 in site B
- Take the primary key to all sites

c) Hybrid Fragmentation (HF)
-Horizontal and vertical could even be combined

Four types of fragmentation:

1. **Horizontal:** Consists of a subset of the tuples of a relation.

- Defined using *Selection* operation
- Determined by looking at predicates used by Ts.
- Involves finding set of *minimal* (*complete* and *relevant*) predicates.

- Set of predicates is *complete*, iff, any two tuples in same fragment are referenced with same probability by any application.
- Predicate is *relevant* if there is at least one application that accesses fragments differently.

2. Vertical: subset of atts of a relation.
- Defined using *Projection* operation
- Determined by establishing *affinity* of one attribute to another.

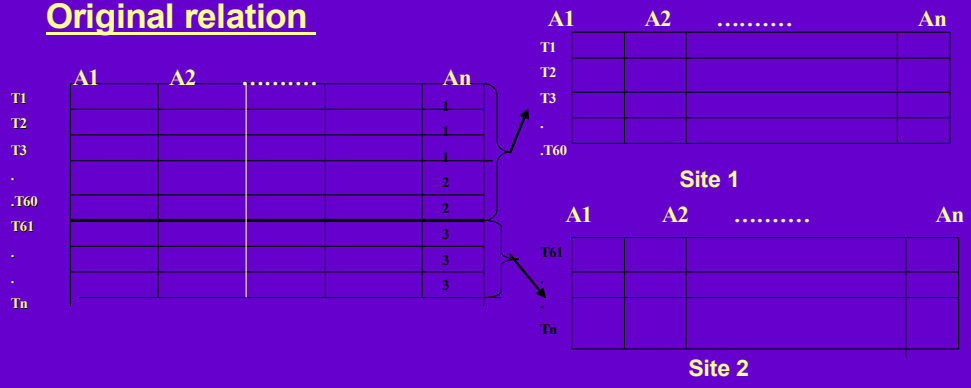
3. Mixed: horizontal fragment that is vertically fragmented, or a vertical fragment that is horizontally fragmented.
- Defined using *Selection* and *Projection* operations

4. Derived: horizontal fragment that is based on horizontal fragmentation of a parent relation.
- Ensures fragments frequently joined together are at same site.
- Defined using *Semijoin* operation

a) **Horizontal Fragmentation (HF)**

HORIZONTAL FRAGMENTATION

Original relation



-Fragments contain subsets of complete tuples (all attributes at all sites)

How to reconstruct R= Rs1 ▪ Rs2 ..... ▪ Rsn

Primary Horizontal Fragmentation

Definition :

$$R_j = \sigma_{F_j}(R), \quad 1 \le j \le w$$

where F_j is a selection formula, which is (preferably) a minterm predicate.

Therefore,

A horizontal fragment R_i of relation R consists of all the tuples of R which satisfy a minterm predicate m_i.

⇓

Given a set of minterm predicates M, there are as many horizontal fragments of relation R as there are minterm predicates.

Set of horizontal fragments also referred to as *minterm fragments*.

PHF – Algorithm

Given: A relation R, the set of simple predicates Pr

Output: The set of fragments of $R = \{R_1, R_2, \ldots, R_w\}$ which obey the fragmentation rules.

Preliminaries :

→ Pr should be *complete*

→ Pr should be *minimal*

PHF – Example

- Two candidate relations : PAY and PROJ.
- Fragmentation of relation PAY
  - → Application: Check the salary info and determine raise.
  - → Employee records kept at two sites ⇒ application run at two sites
  - → Simple predicates

    p_1 : SAL ≤ 30000

    p_2 : SAL > 30000

    $Pr = \{p_1, p_2\}$ which is complete and minimal $Pr' = Pr$
  - → Minterm predicates

    m_1 : (SAL ≤ 30000)

    m_2 : **NOT**(SAL ≤ 30000) = (SAL > 30000)

PHF – Example

PAY_1

TITLE	SAL
Mech. Eng.	27000
Programmer	24000

PAY_2

TITLE	SAL
Elect. Eng.	40000
Syst. Anal.	34000

PHF – Example

- Fragmentation of relation PROJ
  - → Applications:
    - Find the name and budget of projects given their no.
      - Issued at three sites
    - Access project information according to budget
      - one site accesses ≤200000 other accesses >200000
  - → Simple predicates
  - → For application (1)
    - p_1 : LOC = "Montreal"
    - p_2 : LOC = "New York"
    - p_3 : LOC = "Paris"
  - → For application (2)
    - p_4 : BUDGET ≤ 200000
    - p_5 : BUDGET > 200000
  - → $Pr = Pr' = \{p_1, p_2, p_3, p_4, p_5\}$

PHF – Example

PHF – Example

PAY_1

TITLE	SAL
Mech. Eng.	27000
Programmer	24000

PAY_2

TITLE	SAL
Elect. Eng.	40000
Syst. Anal.	34000

- **Fragmentation of relation PROJ**
  - Applications:
    - Find the name and budget of projects given their no.
      - Issued at three sites
    - Access project information according to budget
      - one site accesses ≤200000 other accesses >200000
  - Simple predicates
  - For application (1)
- p_1 : LOC = "Montreal"
- p_2 : LOC = "New York"
- p_3 : LOC = "Paris"
  - For application (2)
- p_4 : BUDGET ≤ 200000
- p_5 : BUDGET > 200000
  - $Pr = Pr' = \{p_1,p_2,p_3,p_4,p_5\}$
  - Minterm fragments left after elimination

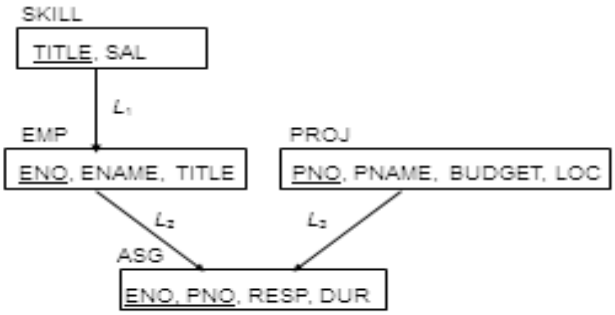
m_1 : (LOC = "Montreal") Ù (BUDGET ≤ 200000)
m_2 : (LOC = "Montreal") Ù (BUDGET > 200000)
m_3 : (LOC = "New York") Ù (BUDGET ≤ 200000)
m_4 : (LOC = "New York") Ù (BUDGET > 200000)
m_5 : (LOC = "Paris") Ù (BUDGET ≤ 200000)
m_6 : (LOC = "Paris") Ù (BUDGET > 200000)

PHF – Correctness

- Completeness
  - Since Pr' is complete and minimal, the selection predicates are complete
- Reconstruction
  - If relation R is fragmented into $F_R = \{R_1,R_2,\ldots,R_r\}$
- $R = È_{"Ri\ îF_R} R_i$
- Disjointness
  - Minterm predicates that form the basis of fragmentation should be mutually exclusive.

Derived Horizontal Fragmentation

- Defined on a member relation of a link according to a selection operation specified on its owner.
  - → Each link is an equijoin.
  - → Equijoin can be implemented by means of semijoins.



Given a link L where *owner*(L)=S and *member*(L)=R, the derived horizontal fragments of R are defined as

$$R_i = R \ltimes_F S_i, \ 1 \le i \le w$$

where w is the maximum number of fragments that will be defined on R and

$S_i = s_{Fi}(S)$

where F_i is the formula according to which the primary horizontal fragment S_i is defined.

DHF – Example

Given link L_1 where owner(L_1)=SKILL and member(L_1)=EMP

$EMP_1 = EMP \ltimes SKILL_1$

$EMP_2 = EMP \ltimes SKILL_2$

where

$SKILL_1 = \sigma_{SAL \le 30000}(SKILL)$

$SKILL_2 = \sigma_{SAL > 30000}(SKILL)$

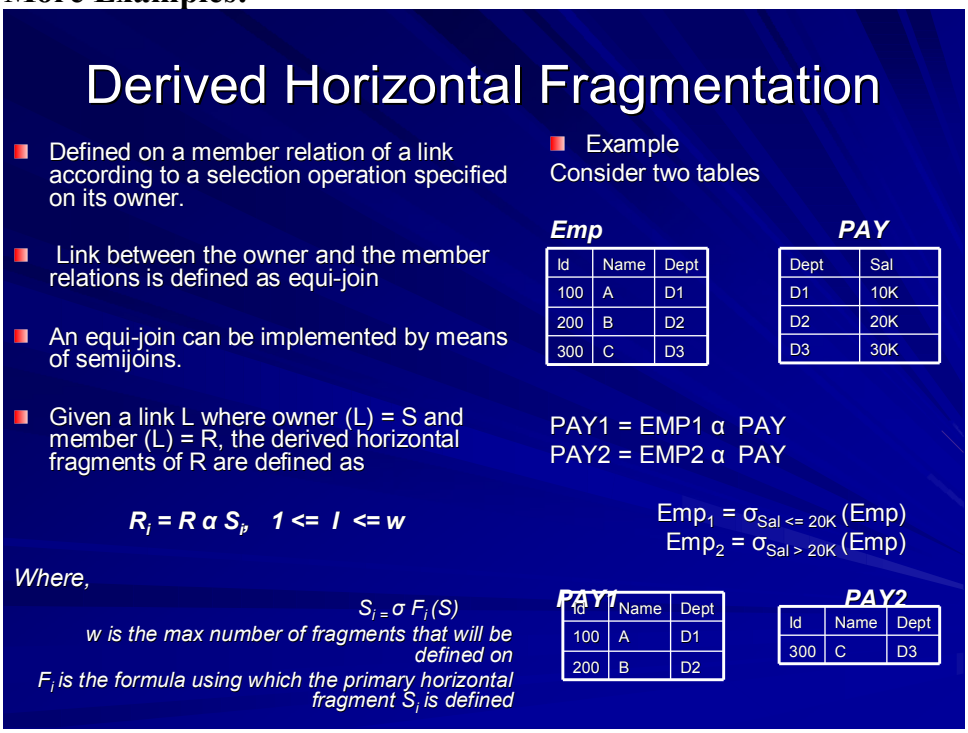
EMP_1

ENO	ENAME	TITLE
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E7	R. Davis	Mech. Eng.

EMP_2

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng.
E2	M. Smith	Syst. Anal.
E5	B. Casey	Syst. Anal.
E6	L. Chu	Elect. Eng.
E8	J. Jones	Syst. Anal.

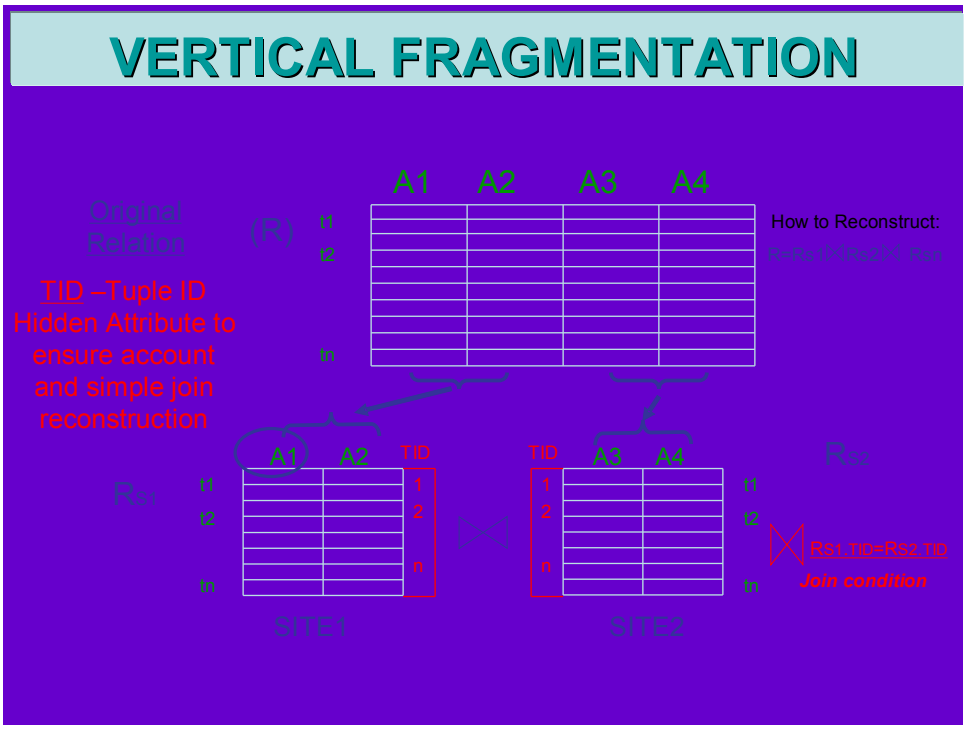
More Examples:



DHF – Correctness

- Completeness
  - Referential integrity
  - Let R be the member relation of a link whose owner is relation S which is fragmented as $F_S = \{S_1, S_2, ..., S_n\}$. Furthermore, let A be the join attribute between R and S. Then, for each tuple t of R, there should be a tuple t' of S such that
- $t[A]=t'[A]$
- Reconstruction
  - Same as primary horizontal fragmentation.
- Disjointness
  - Simple join graphs between the owner and the member fragments.

b) Vertical Fragmentation (VF)



- Has been studied within the centralized context
  - design methodology
  - physical clustering
- More difficult than horizontal, because more alternatives exist.
  - Two approaches :
  - grouping
    - attributes to fragments
  - splitting
    - relation to fragments
- Overlapping fragments
  - grouping
- Non-overlapping fragments
  - splitting

We do not consider the replicated key attributes to be overlapping.

Advantage:

Easier to enforce functional dependencies

(for integrity checking etc.)

VF – Information Requirements

- Application Information
  - Attribute affinities
    - a measure that indicates how closely related the attributes are
    - This is obtained from more primitive usage data
  - Attribute usage values

- Given a set of queries $Q = \{q_1, q_2, \ldots, q_q\}$ that will run on the relation $R[A_1, A_2, \ldots, A_n]$,

$$use(q_i, A_j) = \begin{cases} 1 \text{ if attribute } A_j \text{ is referenced by query } q_i \\ 0 \text{ otherwise} \end{cases}$$

$use(q_i, \bullet)$ can be defined accordingly

Two problems :

- Cluster forming in the middle of the *CA* matrix
  - Shift a row up and a column left and apply the algorithm to find the "best" partitioning point
  - Do this for all possible shifts
  - Cost $O(m^2)$
- More than two clusters
  - m-way partitioning
  - try 1, 2, …, m–1 split points along diagonal and try to find the best point for each of these
  - Cost $O(2^m)$

VF – Correctness

A relation R, defined over attribute set A and key K, generates the vertical partitioning $F_R = \{R_1, R_2, \ldots, R_r\}$.

- Completeness
  - The following should be true for A:

$A = \bigcup A_{Ri}$

- Reconstruction
  - Reconstruction can be achieved by

$R = \bowtie_K R_i \; \forall R_i \in F_R$

- Disjointness
  - TID's are not considered to be overlapping since they are maintained by the system
  - Duplicated keys are not considered to be overlapping

More Examples:

Vertical Fragmentation of *employee-info* Relation

Vertical Fragmentation of *employee-info* Relation

branch-name	customer-name	tuple-id
Hillside	Lowman	1
Hillside	Camp	2
Valleyview	Camp	3
Valleyview	Kahn	4
Hillside	Kahn	5
Valleyview	Kahn	6
Valleyview	Green	7

$deposit_1 = \Pi_{branch\text{-}name,\ customer\text{-}name,\ tuple\text{-}id}(employee\text{-}info)$

account number	balance	tuple-id
A-305	500	1
A-226	336	2
A-177	205	3
A-402	10000	4
A-155	62	5
A-408	1123	6
A-639	750	7

$deposit_2 = \Pi_{account\text{-}number,\ balance,\ tuple\text{-}id}(employee\text{-}info)$

Vertical Fragmentation & Horizontal Fragmentation of Employee Relation

Example Used
Table Name :Employee

Id	Name	Sal	Dept
100	A	10K	D1
200	B	20K	D2
300	C	30K	D3

- *Horizontal Fragmentation*

Rows split : Sal > 20K

Id	Name	Sal	Dept
100	A	10K	D1
200	B	20K	D2

Id	Name	Sal	Dept
300	C	30K	D3

- *Vertical Fragmentation*

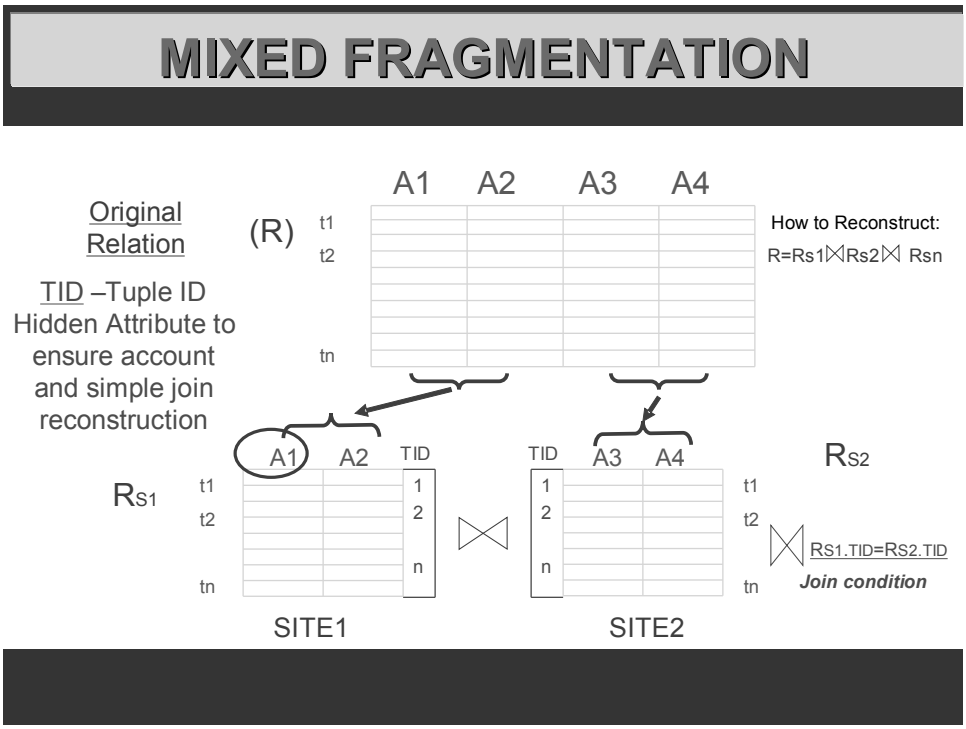
Columns split : Primary Key retained

Id	Name
100	A
200	B
300	C

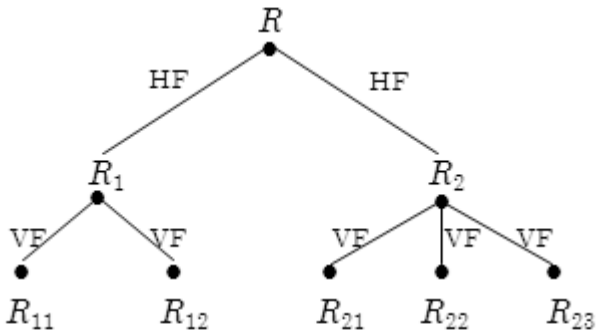
Id	Sal	Dept
100	10K	D1
200	20K	D2
300	30K	D3

c) Hybrid Fragmentation (HF)

- Horizontal or Vertical fragmentation of a Database schema will not be sufficient to satisfy the requirements of user applications.
- In certain cases, a vertical fragmentation may be followed by a horizontal one, or viceversa.
- In case of horizontal fragmentation, one has to stop when each fragment consists of only one tuple, where as the termination part for vertical fragmentation is one attribute per fragment.
- Since two two types of partitioning strategies are applied one after the other, this alternative is called hybrid fragmentation.



Hybrid Fragmentation



Advantages of Fragmentation

- Horizontal:
  - allows parallel processing on fragments of a relation
  - allows a relation to be split so that tuples are located where they are most frequently accessed
- Vertical:
  - allows tuples to be split so that each part of the tuple is stored where it is most frequently accessed
  - tuple-id attribute allows efficient joining of vertical fragments
  - allows parallel processing on a relation
- Vertical and horizontal fragmentation can be mixed.
  - Fragments may be successively fragmented to an arbitrary depth.

Advantages

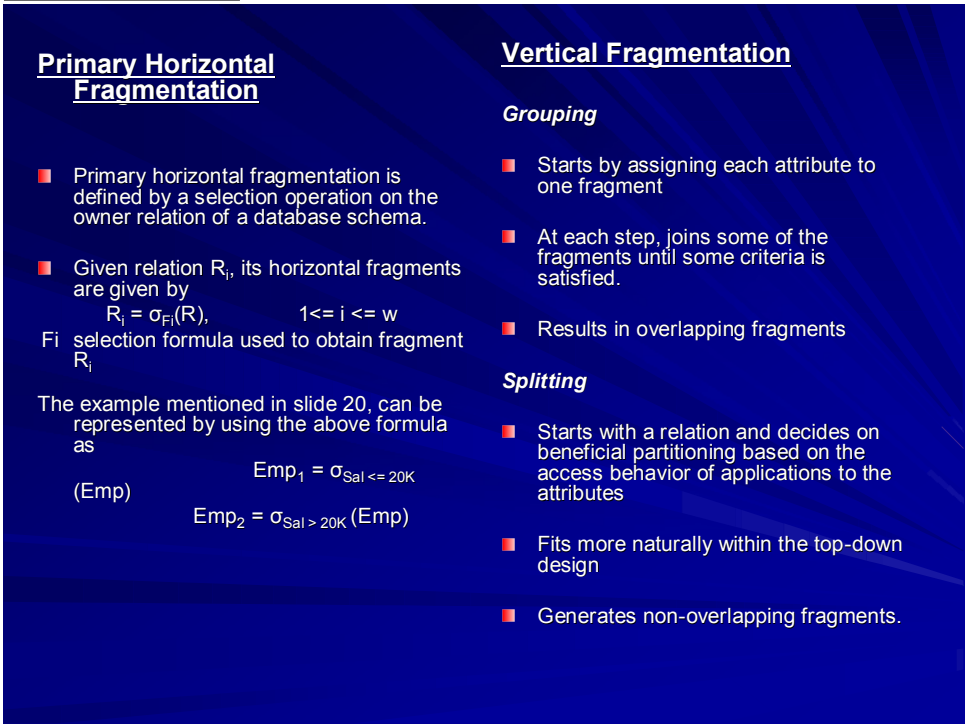
1. Permits a number of transactions to executed concurrently
2. Results in parallel execution of a single query
3. Increases level of concurrency, also referred to as, intra query concurrency
4. Increased System throughput

Disadvantages

1. Applications whose views are defined on more than one fragment may suffer performance degradation, if applications have conflicting requirements.

2. Simple asks like checking for dependencies, would result in chasing after data in a number of sites

PHF Vs VF :



Topic – 5: Query Processing

What is a Query ?
A database query is the vehicle for instructing a DBMS to update or retrieve specific data to/from the physically stored medium.

The actual updating and retrieval of data is performed through various "low-level" operations.
Examples of such operations for a relational DBMS can be relational algebra operations such as project, join, select, Cartesian product, etc.
While the DBMS is designed to process these low-level operations efficiently, it can be quite the burden to a user to submit requests to the DBMS in these formats.

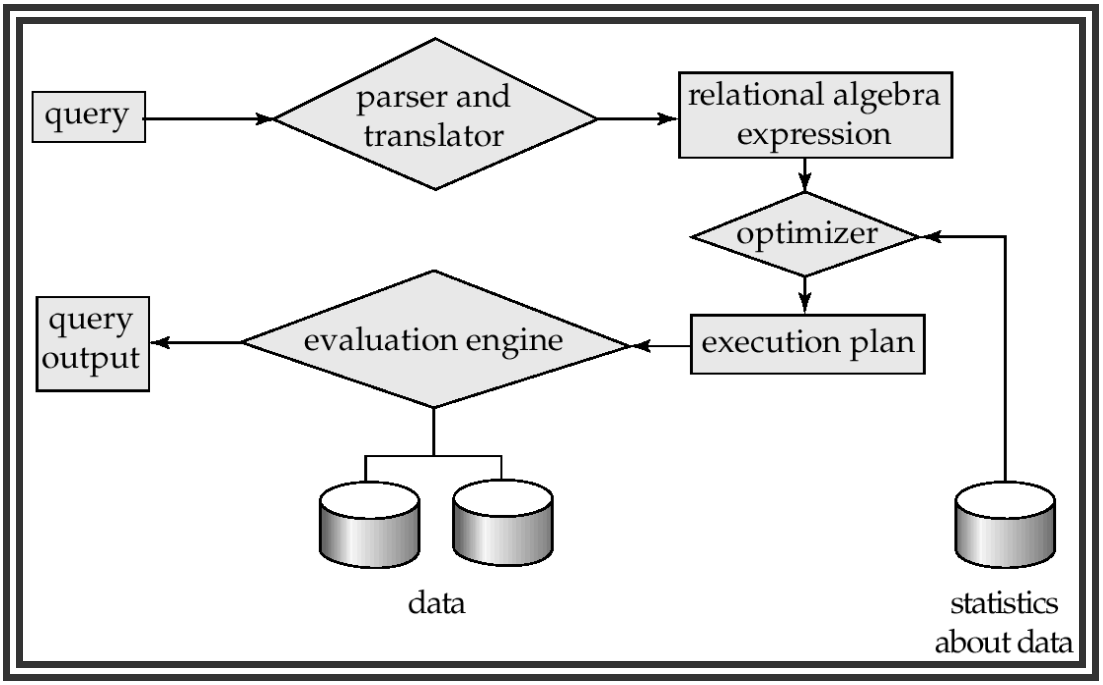
Consider the following request:
"Give me the vehicle ids of all Chevrolet Camaros built in the year 1977."
While this is easily understandable by a human, a DBMS must be presented with a format it can understand, such as this SQL statement:

select *vehicle_id*
from *vehicles*
where *year* = 1977

Note that this SQL statement will still need to be translated further by the DBMS so that the functions/methods within the DBMS program can not only process the request, but do it in a timely manner.

Basic Steps in Query Processing

1. **Parsing and translation**
2. **Optimization**
3. **Evaluation**



- Parsing and translation
  - translate the query into its internal form. This is then translated into relational algebra.
  - Parser checks syntax, verifies relations
- Evaluation
  - The query-execution engine takes a query-evaluation plan, executes that plan, and returns the answers to the query.

Optimization

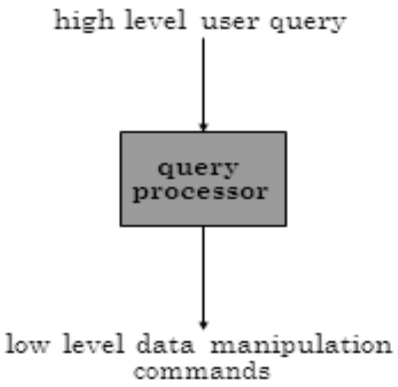
- A relational algebra expression may have many equivalent expressions
  - E.g., $\sigma_{balance<2500}(\Pi_{balance}(account))$ is equivalent to $\Pi_{balance}(\sigma_{balance<2500}(account))$
- Each relational algebra operation can be evaluated using one of several different algorithms
  - Correspondingly, a relational-algebra expression can be evaluated in many ways.
- Annotated expression specifying detailed evaluation strategy is called an evaluation-plan.
  - E.g., can use an index on *balance* to find accounts with balance < 2500,
  - or can perform complete relation scan and discard accounts with balance ≥ 2500

- Query Optimization: Amongst all equivalent evaluation plans choose the one with lowest cost.
  - Cost is estimated using statistical information from the database catalog
    - e.g. number of tuples in each relation, size of tuples, etc.

using client-server architecture

- user creates query
- client parses and sends to server(s) (SQL?)
- servers return appropriate Tables
- client combines into one Table
- Issue of data transfer cost over a network
  - optimise the query to transfer the least amount

Query Processing



Query Processing Components

- Query language that is used
  - SQL: "intergalactic dataspeak"
- Query execution methodology
  - The steps that one goes through in executing high-level (declarative) user queries.
- Query optimization
  - How do we determine the "best" execution plan?

Query Optimization Objectives

- Minimize a cost function
  - I/O cost + CPU cost + communication cost
- These might have different weights in different distributed environments
- Wide area networks
  - communication cost will dominate
    - low bandwidth
    - low speed
    - high protocol overhead
  - most algorithms ignore all other cost components
- Local area networks
  - communication cost not that dominant
  - total cost function should be considered
- Can also maximize throughput

Query Optimization Issues – Types of Optimizers

- Exhaustive search
  - cost-based
  - optimal
  - combinatorial complexity in the number of relations
- Heuristics
  - not optimal
  - regroup common sub-expressions
  - perform selection, projection first
  - replace a join by a series of semijoins
  - reorder operations to reduce intermediate relation size
  - optimize individual operations

Optimization Granularity

- Single query at a time
  - cannot use common intermediate results
- Multiple queries at a time
  - efficient if many similar queries
  - decision space is much larger

Optimization Timing

- Static
  - compilation Þ optimize prior to the execution
  - difficult to estimate the size of the intermediate results Þ error propagation
  - can amortize over many executions
  - R*
- Dynamic
  - run time optimization
  - exact information on the intermediate relation sizes
  - have to reoptimize for multiple executions
  - Distributed INGRES
- Hybrid
  - compile using a static algorithm
  - if the error in estimate sizes > threshold, reoptimize at run time
  - MERMAID

Statistics

- Relation
  - cardinality
  - size of a tuple

- fraction of tuples participating in a join with another relation

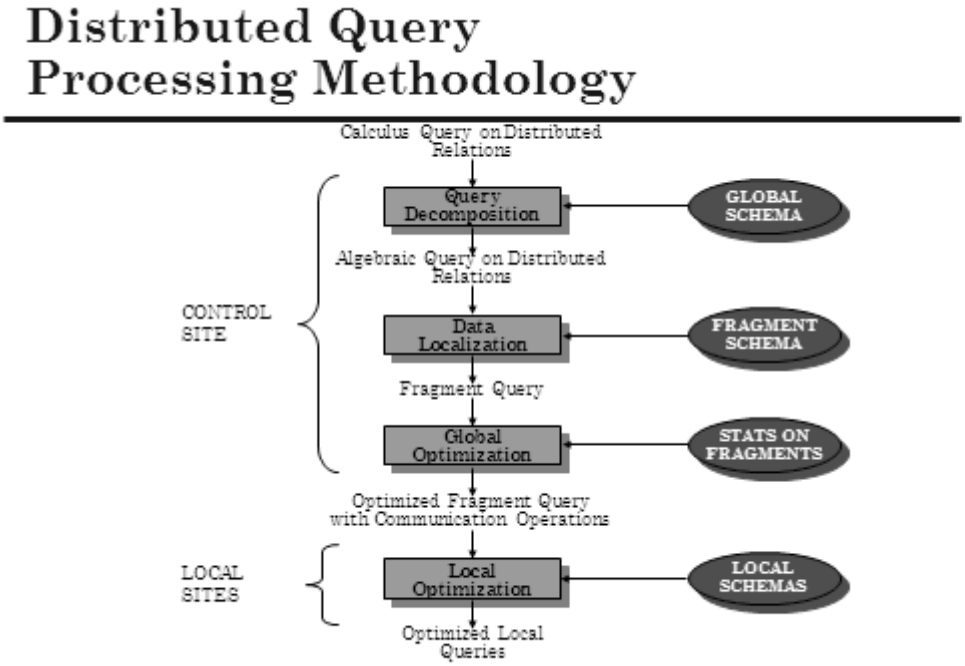
- Attribute
  - cardinality of domain
  - actual number of distinct values
- Common assumptions
  - independence between different attribute values
  - uniform distribution of attribute values within their domain

Decision Sites

- Centralized
  - single site determines the "best" schedule
  - simple
  - need knowledge about the entire distributed database
- Distributed
  - cooperation among sites to determine the schedule
  - need only local information
  - cost of cooperation
- Hybrid
  - one site determines the global schedule
- each site optimizes the local subqueries

Network Topology

- Wide area networks (WAN) – point-to-point
  - characteristics
    - low bandwidth
    - low speed
    - high protocol overhead
  - communication cost will dominate; ignore all other cost factors
  - global schedule to minimize communication cost
  - local schedules according to centralized query optimization
- Local area networks (LAN)
  - communication cost not that dominant
  - total cost function should be considered
  - broadcasting can be exploited (joins)
  - special algorithms exist for star networks



Step 1 – Query Decomposition

- Input : Calculus query on global relations

- Normalization
  - manipulate query quantifiers and qualification
- Analysis
  - detect and reject "incorrect" queries
  - possible for only a subset of relational calculus
- Simplification
  - eliminate redundant predicates
- Restructuring
  - calculus query Þ algebraic query
  - more than one translation is possible
  - use transformation rules

Step 2 – Data Localization

- Input: Algebraic query on distributed relations
- Determine which fragments are involved
- Localization program
  - substitute for each global query its materialization program
  - optimize

Step 3 – Global Query Optimization

- Input: Fragment query
- Find the *best* (not necessarily optimal) global schedule
  - Minimize a cost function
  - Distributed join processing

        - Bushy vs. linear trees
        - Which relation to ship where?
        - Ship-whole vs ship-as-needed
    - Decide on the use of semijoins
        - Semijoin saves on communication at the expense of more local processing.
    - Join methods
        - nested loop vs ordered joins (merge join or hash join)

Centralized Query Optimization

- INGRES
    - dynamic
    - interpretive
- System R
    - static
    - exhaustive search

Topic – 6: Transaction Processing

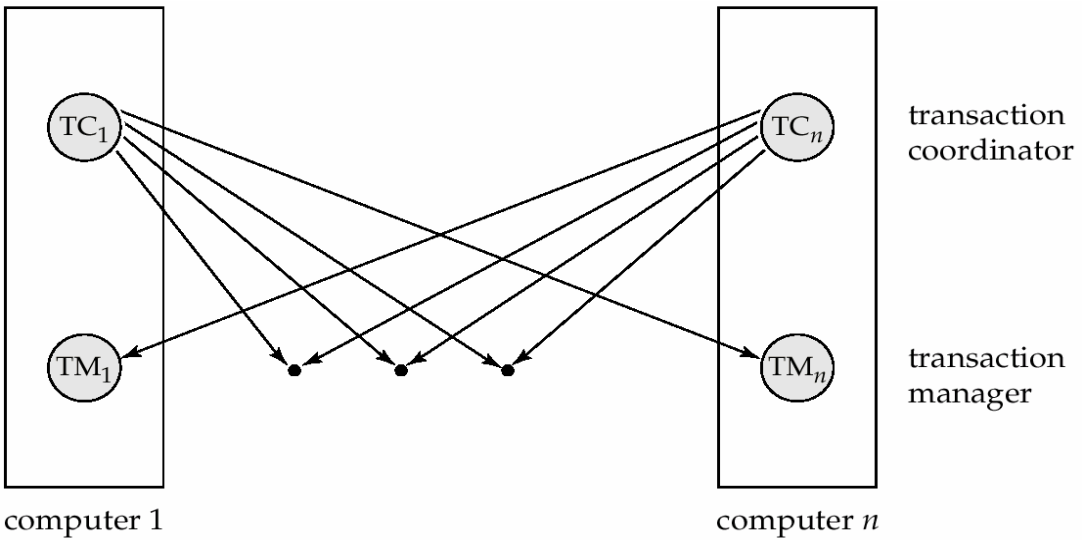
Transaction

A transaction is a collection of actions that make consistent transformations of system states while preserving system consistency.

- concurrency transparency
- failure transparency

☐Transaction may access data at several sites.
☐Each site has a local transaction manager responsible for:
☐Maintaining a log for recovery purposes
☐ Participating in coordinating the concurrent execution of the transactions executing at that site.
☐ Each site has a transaction coordinator, which is responsible for:
☐ Starting the execution of transactions that originate at the site.
☐ Distributing subtransactions at appropriate sites for execution.
☐ Coordinating the termination of each transaction that originates at the site, which may result in the transaction being committed at all sites or aborted at all sites.

Transaction system Architecture



Database in a consistent state ---------->

Database may be temporarily in an inconsistent state during execution
---------->

Database in a consistent state

Transaction Structure

Flat transaction

- Consists of a sequence of primitive operations embraced between a begin and end marks.

Begin_transaction Reservation
.
.
End.

Nested Transaction

-The operations of a transaction may themselves be transactions.

Begin_transaction Reservation

Begin_ transaction Airline .
.
End. Airline
.
Begin_ transaction Hotel .
.
End. Hotel

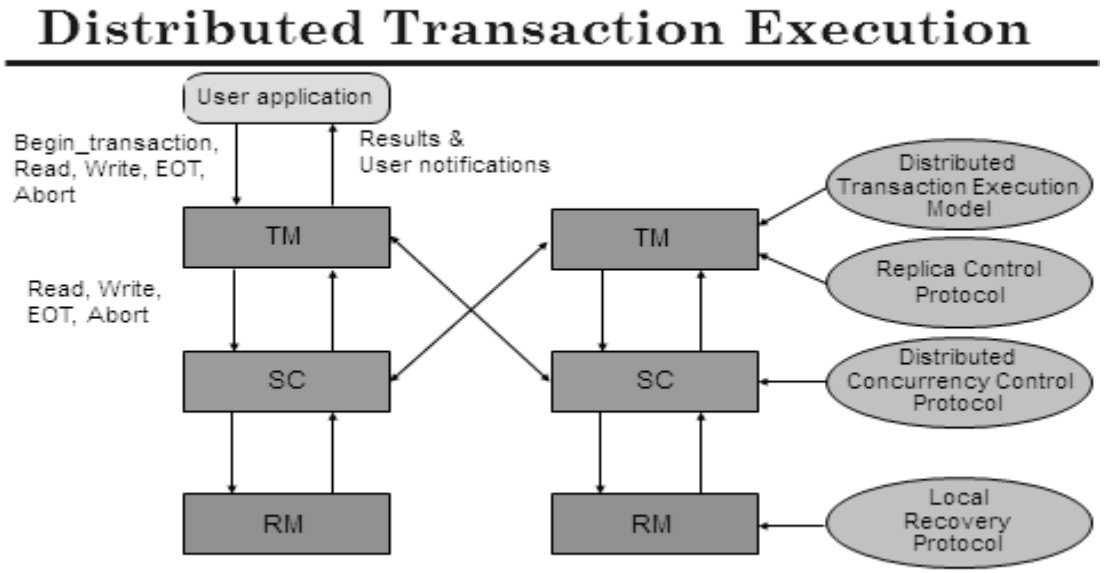
End. Reservation

Properties of Transactions

ACID (Atomicity, Consistency, Isolation, Durability) Property

Atomicity → All or Nothing
Consistency → No violation of integrity constraints
Isolation → Concurrent changes invisible & serialisable
Durability → Committed update persist

-



Transaction Processing Issues

Transaction structure (usually called transaction model)
Flat(simple), nested
Internal database consistency
- Semantic data control (integrity enforcement) algorithms

Reliability Protocols
Atomicity & Durability
Local recovery protocols
Global commit protocols

Concurrency control algorithms

- How to synchronize concurrent transaction executions (correctness criterian)
- Intra-transaction consistency, isolation

Replica control protocols

- How to control the mutual consistency of replicated data
- One copy of equivalence and ROWA

Topic – 7: Concurrency Control

What's concurrency control?
Concurrency control deals with preventing concurrently running processes from improperly inserting, deleting, or updating the same data. Concurrency control is maintained through two mechanisms: Transactions and Locks.

What's transactions?

A transaction is a logical unit of work. It is both the unit of work and the unit of recovery. The statements nested within a transaction must either all happen or none happen.

Transactions are a mandatory facility for maintaining the integrity of a database while running multiple concurrent operations.

Transactions are atomic: there is no such thing as a partial transaction.

A set of transactions is said to be serializable if and only if it produces the same result as some arbi trary serial execution of those same transactions for arbitrary input. A set of transactions can be correct only if it is serializable.

Transactions are a mandatory facility for maintaining the integrity of a database while running multiple concurrent operations. A transaction is a logical unit of work. It is both the unit of work and the unit of recovery.

The statements nested within a transaction must either all happen or none happen.
Transactions are atomic: there is no such thing as a partial transaction.
Concurrency control deals with preventing concurrently running processes from improperly inserting, deleting, or updating the same data.

Two Concurrency control mechanisms:
Concurrency control is maintained through two mechanisms: Transactions and Locks.

What's lock?
A lock is a means of claiming usage rights on some resource.

There can be several different types of resources that can be locked and several different ways of locking those resources.

Most locks used on Teradata resources are locked automatically by default. The Teradata ***lock manager implicitly locks the following objects***:
Database, Table, View and Row hash.

User can apply four ***different levels of locking*** on Teradata resources:
Exclusive, Write, Read and Access.

The Teradata R DBMS applies most of its locks automatically.

☐Modify concurrency control schemes for use in distributed environment.
☐ We assume that each site participates in the execution of a commit protocol to ensure global transaction automicity.
☐ We assume all replicas of any item are updated

- The problem of synchronizing concurrent transactions such that the consistency of the database is maintained while, at the same time, maximum degree of concurrency is achieved.
- Anomalies:
  - Lost updates
    - The effects of some transactions are not reflected on the database.
  - Inconsistent retrievals
    - A transaction, if it reads the same data item more than once, should always read the same value.
- Extends centralised concurrency mechanisms
- Multiple copies of data items
  - maintain consistency
- failures in individual sites/network
  - continue operations, update and rejoin
- distributed commit
  - 2-phase protocol (local and global)
- distributed deadlock
- Global serialisation must occur
  - i.e. serialise local serialisations!
  - Locks and timestamping apply

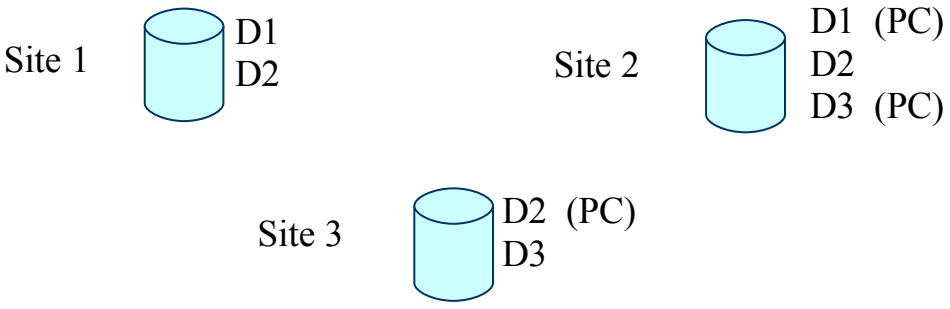
- If database ***not*** replicated and transactions all local or performable at one remote site then:
  - Use centralised concurrency mechanisms
- Otherwise mechanisms need to be extended
  - To deal with replication or transactions involving multiple sites
- Need to consider deadlocks at local and global levels

Distributed Locks

- Just like centralised mechanisms…. But we need to consider locks that manage replication and sub-transactions
- Four modes of management possible:
  - Centralised 2PL
    - Read any copy, update all for updates
    - Single site, bottleneck, failure?
  - Primary Copy 2PL
    - Distributes locks, one copy designated primary, others slaves
    - Only primary copy locked for updates, slaves updated later
  - Distributed 2PL
    - Each site manages its own data locks
    - All copies locked for an update, high cost of comms
  - Majority Locking

Diagrammatic representation

D=Data item (PC=Primary Copy, only for Primary copy 2PL)



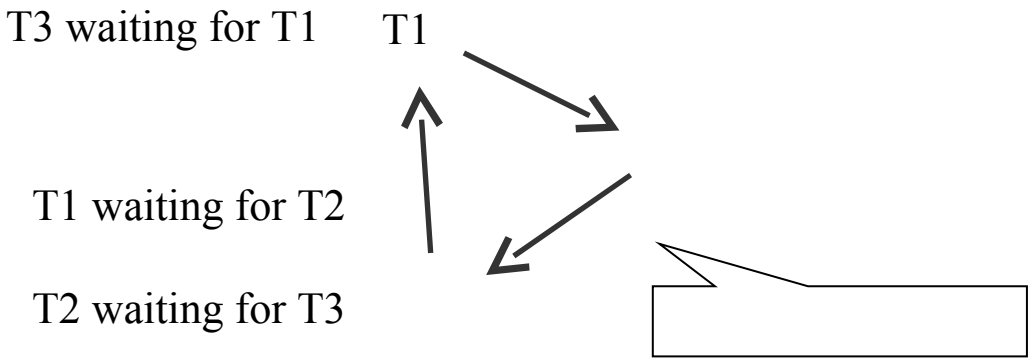
- Centralised: e.g. Site 1 is the only Lock Manager
- Primary Copy: e.g. Site 1 handles locks on D1/D3
  - Site 3 handles locks on D2
  - remember the site does NOT have to hold the PC

- Distributed: All sites lock own data
  (lock all copies for writing)

Majority Locking

- Extension of distributed 2PL
- Doesn't lock all copies before update
- Needs more than half of locks on a copy to proceed
- If so, it informs other sites
- Otherwise it cancels request
- Only one transaction with an exclusive lock
- Many transactions can hold a majority lock on a shared lock

Deadlock



Example

Locally:

Site 1: T3 waiting for T1 $T_{ext} \rightarrow T3 \rightarrow T1 \rightarrow T_{ext}$

$T_{ext} \rightarrow T1 \rightarrow T2 \rightarrow T_{ext}$

Site 3: T2 waiting for T3 $T_{ext} \rightarrow T2 \rightarrow T3 \rightarrow T_{ext}$

Maybe Deadlock?

Site 1 sends WFG to site 2, site 2 combines WFG to

$T_{ext} \rightarrow T3 \rightarrow T1 \rightarrow T2 \rightarrow T_{ext}$

Site 2 sends WFG to site 3, site 3 combines WFG to

$T_{ext} \rightarrow T3 \rightarrow T1 \rightarrow T2 \rightarrow T3 \rightarrow T_{ext}$

Definitely Deadlock!

Distributed Reliability Protocols

- Commit protocols
  - How to execute commit command for distributed transactions.
  - Issue: how to ensure atomicity and durability?
- Termination protocols
  - If a failure occurs, how can the remaining operational sites deal with it.
  - *Non-blocking* : the occurrence of failures should not force the sites to wait until the failure is repaired to terminate the transaction.
- Recovery protocols
  - When a failure occurs, how do the sites where the failure occurred deal with it.
  - *Independent* : a failed site can determine the outcome of a transaction without having to obtain remote information.
- Independent recovery Þ non-blocking termination

Topic – 8: Recovery

Purpose of Database Recovery

- To bring the database into the last consistent state, which existed prior to the failure.

- To preserve transaction properties (Atomicity, Consistency, Isolation and Durability).

Example: If the system crashes before a fund transfer transaction completes its execution, then either one or both accounts may have incorrect value. Thus, the database must be restored to the state before the transaction modified any of the accounts.

Ensures database is fault tolerant, and not corrupted by software, system or media failure

– 7x24 access to mission critical data.

- Failure can occur through
  - Loss of message
    - By network protocol
    - DDBMS deals with it transparently
  - Loss of a communication link
    - Network partitioning (see diagram)
  - Site failure

Types of Failure

The database may become unavailable for use due to

- Transaction failure: Transactions may fail because of incorrect input, deadlock, incorrect synchronization.
- System failure: System may fail because of addressing error, application error, operating system fault, RAM failure, etc.
- Media failure: Disk head crash, power disruption, etc.

Database Recovery Techniques

Transaction Log

For recovery from any type of failure data values prior to modification (BFIM - BeFore Image) and the new value after modification (AFIM – AFter Image) are required. These values and other information is stored in a sequential file called Transaction log.

Data Update

- **Immediate Update**: As soon as a data item is modified in cache, the disk copy is updated.
- **Deferred Update**: All modified data items in the cache is written either after a transaction ends its execution or after a fixed number of transactions have completed their execution.
- **Shadow update**: The modified version of a data item does not overwrite its disk copy but is written at a separate disk location.
- **In-place update**: The disk version of the data item is overwritten by the cache version.

Data Caching

Data items to be modified are first stored into database cache by the Cache Manager (CM) and after modification they are flushed (written) to the disk. The flushing is controlled by **Modified** and **Pin-Unpin** bits.

Pin-Unpin: Instructs the operating system not to flush the data item.
Modified: Indicates the AFIM of the data item

Roll-back: One execution of T1, T2 and T3 as recorded in the log.

Write-Ahead Logging

When **in-place** update (immediate or deferred) is used then log is necessary for recovery and it must be available to recovery manager. This is achieved by **Write-Ahead Logging** (WAL) protocol.

Checkpointing

Time to time (randomly or under some criteria) the database flushes its buffer to database disk to minimize the task of recovery.

Steal/No-Steal and Force/No-Force

Possible ways for flushing database cache to database disk:
Steal: Cache can be flushed before transaction commits.
No-Steal: Cache cannot be flushed before transaction commit.
Force: Cache is immediately flushed (forced) to disk.
No-Force: Cache is deferred until transaction commits.

Recovery Scheme

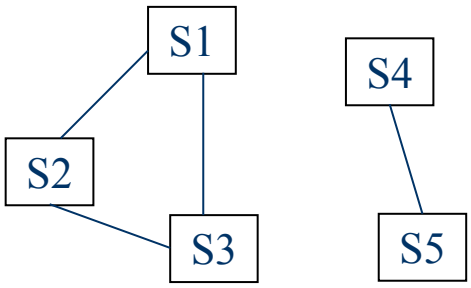
Deferred Update (No Undo/Redo)

Recovery in multidatabase system

- ✓ The multiple nodes agree to commit individually the part of the transaction they were executing. This commit scheme is referred to as "*two-phase commit*" (2PC).

- ✓ If any one of these nodes fails or cannot commit the part of the transaction, then the transaction is aborted. Each node recovers the transaction under its own recovery protocol.

- E.g. S4 out of contact with S1
  - S4 crashed?
  - Link down?
  - Partitioned?
  - S4 busy?



Recovery after failure?

- ➢ Distributed recovery maintains atomicity and durability
- ➢ What happens then?
  - Abort transactions affected by the failure
    - Including all subtransactions
  - Flag the site as failed
  - Check for recovery or wait for message to confirm
  - On restart, abort partial transactions which were active at the time of the failure
  - Perform local recovery
  - Update copy of database to be consistent with remainder of the system

- **Rcovery Protocol**

Protocols at failed site to complete all transactions outstanding at the time of failures.

- Classes of failures
  1. Site failure
  2. Lost messages
  3. Network partitioning
  4. Byzantine failures

- **Effects of failures**
    1. Inconsistent database
    2. Transaction processing is blocked
    3. Failed component unavailable

- **Independent Recovery**

A recovering site makes a transition directly to a final state without communicating with other sites.

- **Lemma**

For a protocol, if a local state's concurrency set contains both an abort and commit, it is not resilient to an arbitrary failure of a single site.

$S_i \rightarrow$ commit because other sites may be in abort

$S_i \rightarrow$ abort because other sites may be in commit

Rule 1: S: Intermediate state

If C(s) contains a commit $\Rightarrow$failure transition from S to commit

Otherwise failure transition from S to abort

Unscheduled restarts occur for one of the following reasons:

- AMP or disk failure
- Software failure
- Parity error

Transaction recovery describes how the Teradata RDBMS restarts itself after a system or media failure.

Two types of automatic recovery of transactions can occur when an unscheduled restart occurs:

- Single transaction recovery
- RDBMS recovery

The following table details when these two automatic recovery mechanisms take place:

This Recovery Type	***Happens When***
Single transaction	The RDBMS aborted a single transaction because of: ▪ Transaction deadlock timeout ▪ User error ▪ User-initiated abort command ▪ An inconsistent data table ▪ Unavailable resources for parsing

	▪ Single transaction recovery uses the transient journal to effect its data restoration
RDBMS	A RDBMS restart is caused by: ▪ Hardware failure ▪ Software failure ▪ User command

Site Failures - 2PC Recovery

- Failure in INITIAL
  - → Unilaterally abort upon recovery
- Failure in READY
  - → The coordinator has been informed about the local decision
  - → Treat as timeout in READY state and invoke the termination protocol
- Failure in ABORT or COMMIT
  - → Nothing special needs to be done

Two-Phase Commit Protocol

Two-phase commit (2PC) is a protocol for assuring concurrency of data in multiple databases in which each participant database manager votes to either commit or abort the changes.

The participants wait before committing the change until it is known that all participants can commit. By voting to commit, the participant guarantees that it can either commit or rollback its part of the transaction, even if it crashes before receiving the result of the vote.

The 2PC protocol allows C ICS and IMS applications to be developed that can update one or more Teradata RDBMS databases and/or databases under some other DBMS in

a synchronized manner. The result is that all updates requested in a defined unit of work will either succeed or fail.

2PC Recovery Protocols – Additional Cases

Arise due to non-atomicity of log and message send actions.

Sample Questions

Topic – 1:

1. What is a distributed database? (2M)
2. Define "Distributed DBMS" (DDBMS) (2M)
3. What are the characteristics of DDBMS ? (2M)
4. What is important difference between DDBMS and distributed processing? (2M)
5. What are the Functions of a DDBMS ? (2M)
6. What are the Advantages and Disadvantages of DDBSs? (2M)

7. What are the Applications of DDBMS? (2M)
8. Explain in detail about the Types of DDBMS. (16M)
9. List the two main issues in DDBMS (2M)

Topic – 2:

1. Explain Distributed Database in detail. (8M)
2. Explain Centralized Database System in detail. (8M).
3. List and differentiate between Distributed Databases Vs Conventional Databases

Topic – 3:

1. Explain the Client-Server Architecture with a neat diagram. (8M)
2. Explain the Distributed Database Architecture with a neat diagram. (8M)
3. What is Synchronous Distributed Database?
4. What is Asynchronous Distributed Database?
5. Explain the major issues a DDBMS in detail. (8M)
6. Explain how Replication or Data Replications used in DDBMS. (8M)

Topic – 4:

1. What is the Concept behind fragmentation ? Give examples. (8M)
2. Why we need fragmentation? (2M)
3. Explain in detail the Types of Fragmentation and give examples for each. (16M)
4. Explain in detail on Horizontal Fragmentation (HF). (8M)
5. Explain in detail on Vertical Fragmentation (VF). (8M)
6. Explain in detail on Hybrid Fragmentation (HF). (8M)
7. Two problems :
8. What are the Advantages and Disadvantages of Fragmentation (2M)
9. Compare and contrast the PHF and VF. (8M)

Topic – 5:

1. What is a Query ? (2M)
2. What are Query Processing? (2M)
3. List and describe the basic Steps in Query Processing. (8M)
4. Explain the concept of Query Optimization. (8M)
5. List and describe the Query Processing Components. (8M)
6. List the Query Optimization Objectives. (4M)
7. Explain the major Query Optimization Issues. (8M)

8. Explain in details on Query processing (8M)
9. What are Centralized Query Optimization? Explain briefly. (8M)

Topic – 6:

1. What is transaction ? Give examples. (2M)
2. Give the local transaction manager responsibilities. (2M)
3. Explain in detail on Transaction system Architecture. Illustrate with a neat diagram (8M)
4. Explain Transaction Structure in detail. (8M)
5. What are the three major Properties of Transactions ? (2M)
6. What is ACID?
7. List and describe the Transaction Processing Issues. (8M).

Topic – 7:

1. What is currency control ? (2M)
2. Explain Concurrency control mechanisms. (2M)
3. What's lock? (2M)
4. What is Distributed Locks ? (2M)
5. Describe the Majority Locking. (2M)
6. Explain in detail on currency control in handled in DDBMS. (8M)

Topic – 8:

1. What is Failure? (2M)
2. What is Recovery? (2M)
3. Is Recovery after failure? Explain. (2M)
4. Explain about Recovery Protocol. (2M)
5. What are the major Effects of failures? (8M)
6. Explain in detail about two automatic recovery mechanisms (8M)
7. Explain in detail on Two-Phase Commit Protocol. (8M)
8. Explain in detail on Recovery in handled in DDBMS. (8M)

University Questions

1. Differentiate homogenous and hetrogenous databases with reference to distributed databases. (2M)
2. Name the fragmentations supported in a distributed system and write examples for each. (2M)
3. Explain how concurrency control and recovery techniques are handled in DDBMS. (16M)

4. Draw simplified physical client Architecture for distributed database systems and discuss in detail (8M)
5. Discuss the techniques of fragmentation, data replication used in distributed database design. (8M)

********** End of Unit – I **********