Chapter 24

Distributed DBMSs – Concepts and Design

Chapter 24 - Objectives

- Concepts.
- Advantages and disadvantages of distributed databases.
- Functions and architecture for a DDBMS.
- Distributed database design.
- Levels of transparency.
- Comparison criteria for DDBMSs.

Concepts

Distributed Database

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

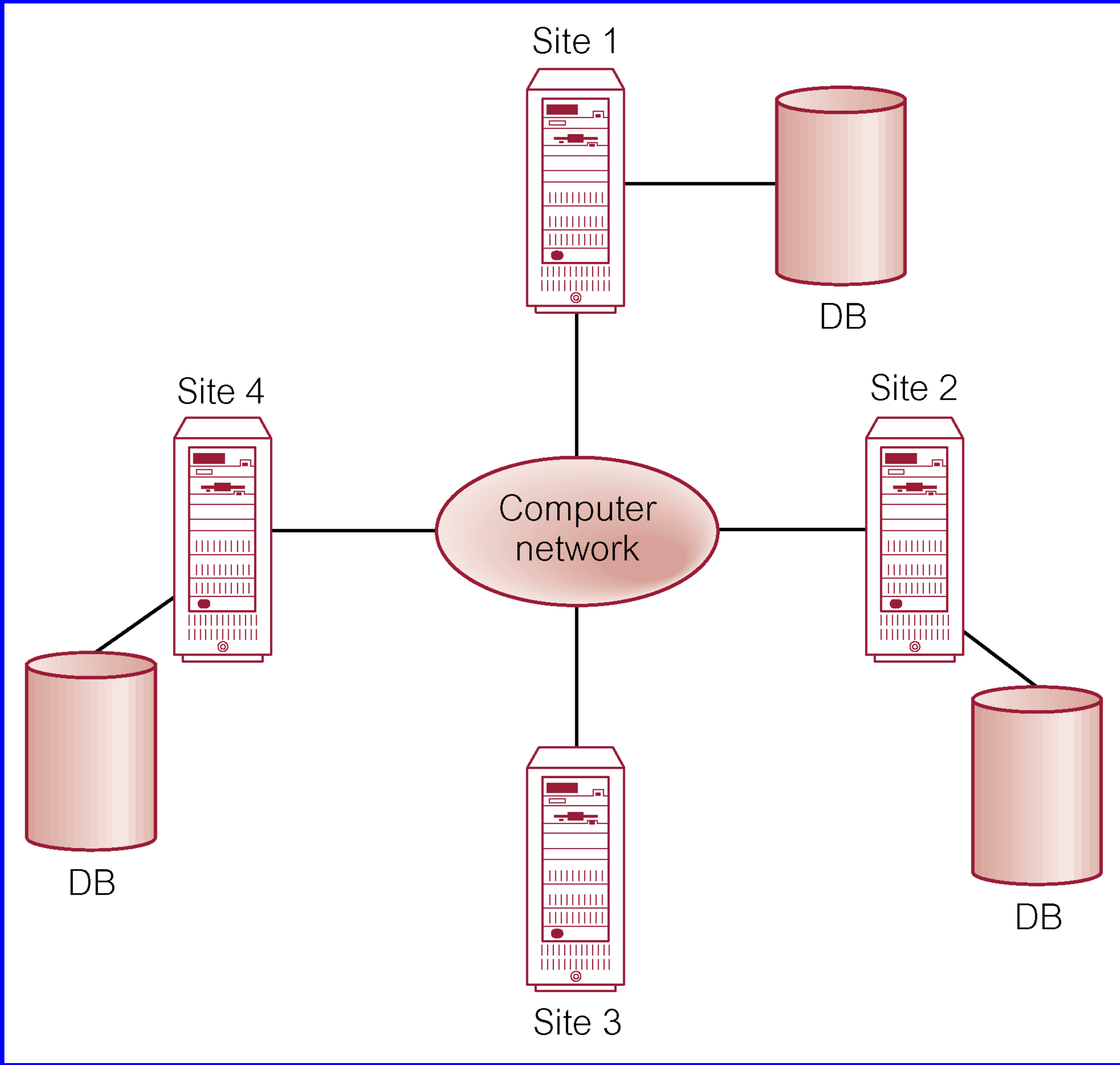
Distributed DBMS

Software system that permits the management of the distributed database and makes the distribution transparent to users.

Concepts

- Collection of logically-related shared data.
- Data split into fragments.
- Fragments may be replicated.
- Fragments/replicas allocated to sites.
- Sites linked by a communications 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.

Distributed DBMS



Types of DDBMS

- Homogeneous DDBMS
- Heterogeneous DDBMS

Homogeneous DDBMS

- All sites use same DBMS product.
- Much easier to design and manage.
- Approach provides incremental growth and allows increased performance.

Heterogeneous DDBMS

- Sites may run different DBMS products, with possibly different underlying data models.
- Occurs when sites have implemented their own databases and integration is considered later.
- Translations required to allow for:
  - Different hardware.
  - Different DBMS products.
  - Different hardware and different DBMS products.
- Typical solution is to use *gateways*.

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.

Distributed Database Design

- Three key issues:
  - Fragmentation,
  - Allocation,
  - Replication.

Distributed Database Design

Fragmentation

Relation may be divided into a number of sub-relations, which are then distributed.

Allocation

Each fragment is stored at site with "optimal" distribution.

Replication

Copy of fragment may be maintained at several sites.

Fragmentation

- Definition and allocation of fragments carried out strategically to achieve:
  - Locality of Reference.
  - Improved Reliability and Availability.
  - Improved Performance.
  - Balanced Storage Capacities and Costs.
  - Minimal Communication Costs.
- Involves analyzing most important applications, based on quantitative/qualitative information.

Fragmentation

- Quantitative information may include:
  - frequency with which an application is run;
  - site from which an application is run;
  - performance criteria for transactions and applications.

- Qualitative information may include transactions that are executed by application, type of access (read or write), and predicates of read operations.

Data Allocation

- Four alternative strategies regarding placement of data:
  - Centralized,
  - Partitioned (or Fragmented),
  - Complete Replication,
  - Selective Replication.

Data Allocation

Centralized: Consists of single database and DBMS stored at one site with users distributed across the network.

Partitioned: Database partitioned into disjoint fragments, each fragment assigned to one site.

Complete Replication: Consists of maintaining complete copy of database at each site.

Selective Replication: Combination of partitioning, replication, and centralization.

Why Fragment?

- Usage
  - Applications work with views rather than entire relations.
- Efficiency
  - Data is stored close to where it is most frequently used.
  - Data that is not needed by local applications is not stored.

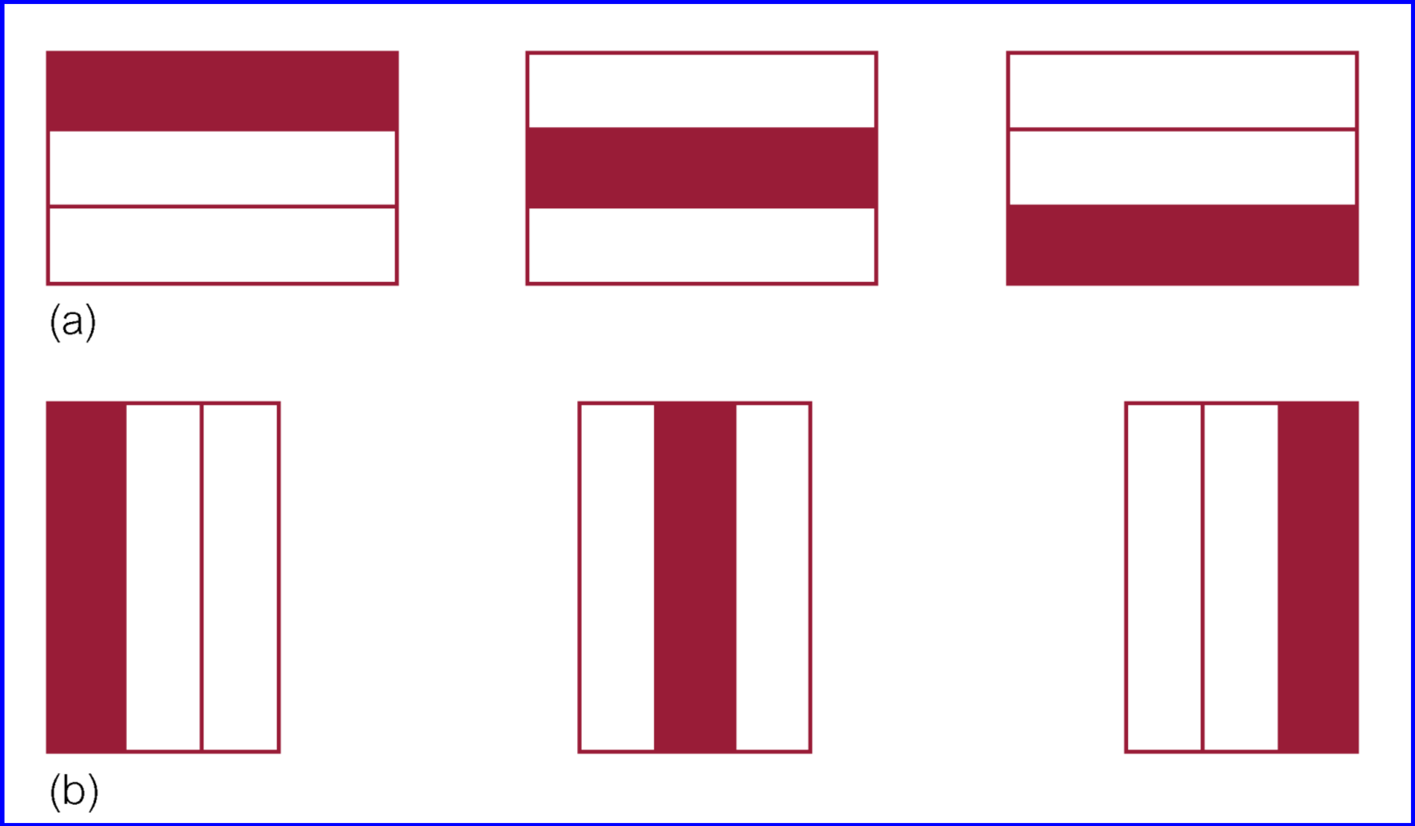
Why Fragment?

- **Parallelism**
  - With fragments as unit of distribution, transaction can be divided into several subqueries that operate on fragments.
- **Security**
  - Data not required by local applications is not stored and so not available to unauthorized users.

Why Fragment?

- **Disadvantages**
  - Performance,
  - Integrity.

Horizontal and Vertical Fragmentation



Correctness of Fragmentation

- Three correctness rules:
  - Completeness,
  - Reconstruction,
  - Disjointness.

Correctness of Fragmentation

Completeness

If relation *R* is decomposed into fragments R_1, R_2, ... R_n, each data item that can be found in *R* must appear in at least one fragment.

Reconstruction

- Must be possible to define a relational operation that will reconstruct *R* from the fragments.
- Reconstruction for horizontal fragmentation is Union operation and Join for vertical .

Correctness of Fragmentation

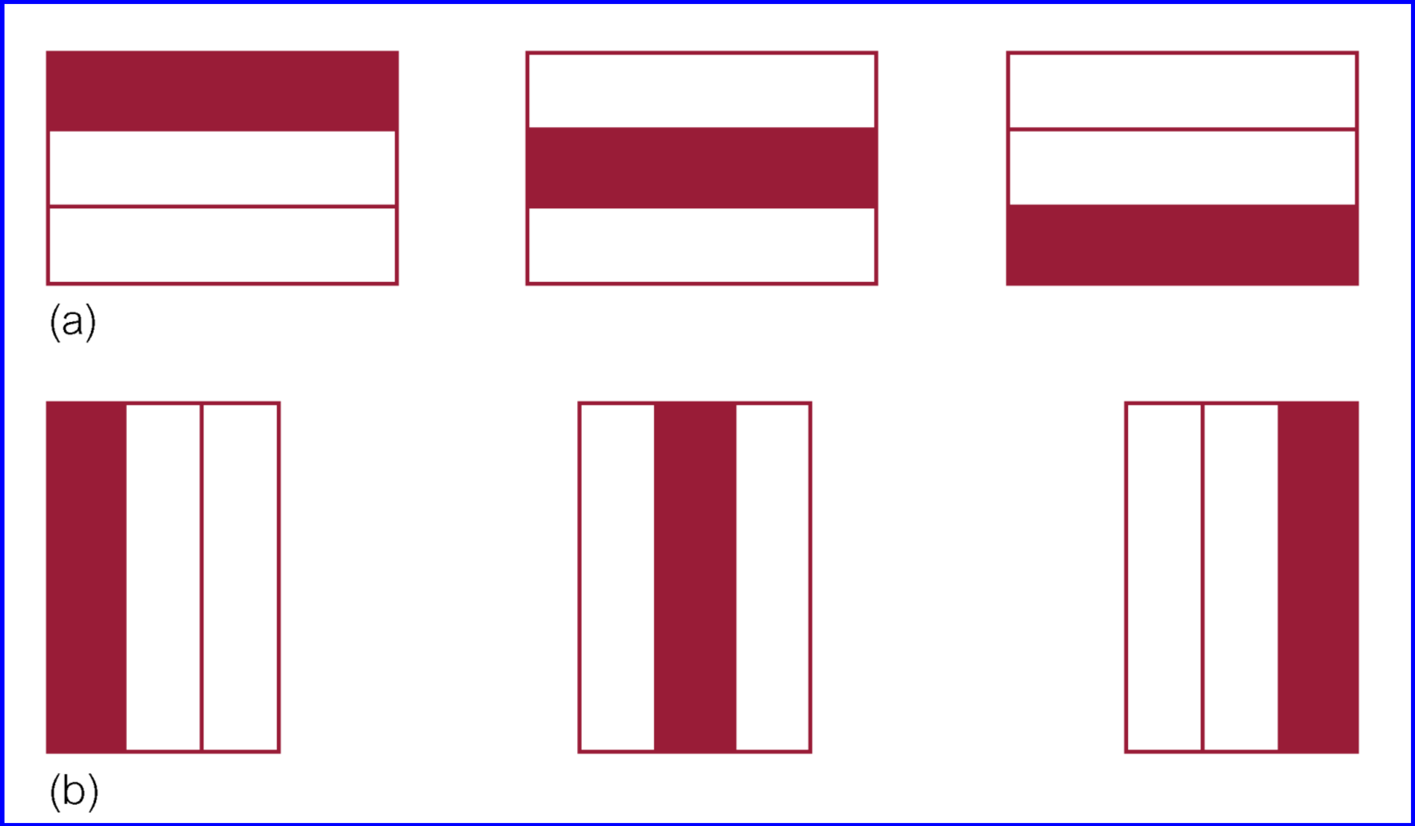
Disjointness

- If data item d_i appears in fragment R_i, then it should not appear in any other fragment.
- Exception: vertical fragmentation, where primary key attributes must be repeated to allow reconstruction.
- For horizontal fragmentation, data item is a tuple.
- For vertical fragmentation, data item is an attribute.

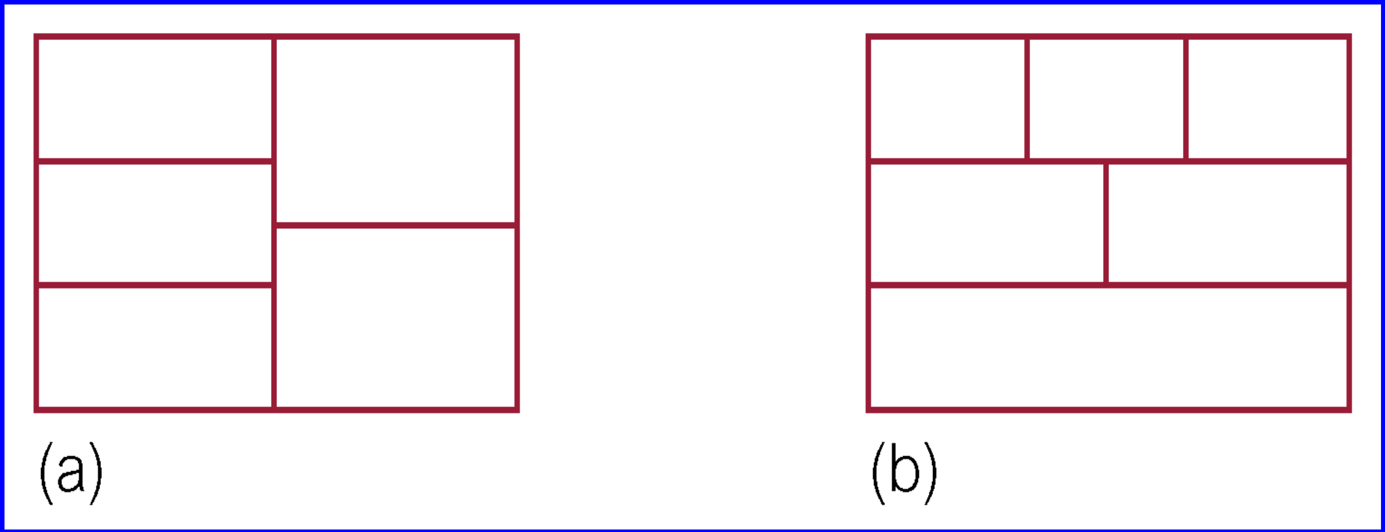
Types of Fragmentation

- Four types of fragmentation:
  - Horizontal,
  - Vertical,
  - Mixed,
  - Derived.
- Other possibility is no fragmentation:
  - If relation is small and not updated frequently, may be better not to fragment relation.

Horizontal and Vertical Fragmentation



Mixed Fragmentation



Horizontal Fragmentation

- Consists of a subset of the tuples of a relation.
- Defined using *Selection* operation of relational algebra:

$$\sigma_p(R)$$

- For example:

$$P_1 = \sigma_{\text{type='House'}}(\text{PropertyForRent})$$
$$P_2 = \sigma_{\text{type='Flat'}}(\text{PropertyForRent})$$

Vertical Fragmentation

- Consists of a subset of attributes of a relation.
- Defined using *Projection* operation of relational algebra:

$$\Pi_{a1, \ldots ,an}(R)$$

- For example:

$$S_1 = \Pi_{\text{staffNo, position, sex, DOB, salary}}(\text{Staff})$$
$$S_2 = \Pi_{\text{staffNo, fName, lName, branchNo}}(\text{Staff})$$

- Determined by establishing *affinity* of one attribute to another.

Distributed Database Design Methodology

1. Use normal methodology to produce a design for the global relations.
2. Examine topology of system to determine where databases will be located.
3. Analyze most important transactions and identify appropriateness of horizontal/vertical fragmentation.
4. Decide which relations are not to be fragmented.

Transparencies in a DDBMS

- Distribution Transparency
  - Fragmentation Transparency
  - Location Transparency
  - Replication Transparency
  - Local Mapping Transparency

Transparencies in a DDBMS

- Transaction Transparency
  - Concurrency Transparency
  - Failure Transparency
- DBMS Transparency

Distribution Transparency

- Distribution transparency allows user to perceive database as single, logical entity.
- If DDBMS exhibits distribution transparency, user does not need to know:
  - data is fragmented (fragmentation transparency),
  - location of data items (location transparency),
  - otherwise call this local mapping transparency.
- With replication transparency, user is unaware of replication of fragments .

Naming Transparency

- Each item in a DDB must have a unique name.
- DDBMS must ensure that no two sites create a database object with same name.
- One solution is to create central name server. However, this results in:
  - loss of some local autonomy;
  - central site may become a bottleneck;
  - low availability; if the central site fails, remaining sites cannot create any new objects.

Transaction Transparency

- Ensures that all distributed transactions maintain distributed database's integrity and consistency.
- Distributed transaction accesses data stored at more than one location.
- Each transaction is divided into number of subtransactions, one for each site that has to be accessed.
- DDBMS must ensure the indivisibility of both the global transaction and each of the subtransactions.

Concurrency Transparency

- All transactions must execute independently and be logically consistent with results obtained if transactions executed one at a time, in some arbitrary serial order.
- Same fundamental principles as for centralized DBMS.
- DDBMS must ensure both global and local transactions do not interfere with each other.
- Similarly, DDBMS must ensure consistency of all subtransactions of global transaction.

Concurrency Transparency

- Results of concurrent transactions same as some serial order of transaction execution.
- Replication makes concurrency more complex.
- If a copy of a replicated data item is updated, update must be propagated to all copies.
- Could propagate changes as part of original transaction, making it an atomic operation.
- However, if one site holding copy is not reachable, then transaction is delayed until site is reachable.

Concurrency Transparency

- Could limit update propagation to only those sites currently available. Remaining sites updated when they become available again.
- Could allow updates to copies to happen asynchronously, sometime after the original update. Delay in regaining consistency may range from a few seconds to several hours.

Failure Transparency

- DDBMS must ensure atomicity and durability of global transaction.
- Means ensuring that subtransactions of global transaction either all commit or all abort.
- Thus, DDBMS must synchronize global transaction to ensure that all subtransactions have completed successfully before recording a final COMMIT for global transaction.
- Must do this in presence of site and network failures.