National Technical University “Kharkiv Polytechnic Institute”

Co-funded by the Erasmus+ Programme of the European Union

Distributed Database Systems and Data Warehouses

Dr. Volodymyr Sokol
(vlad.sokol@gmail.com)

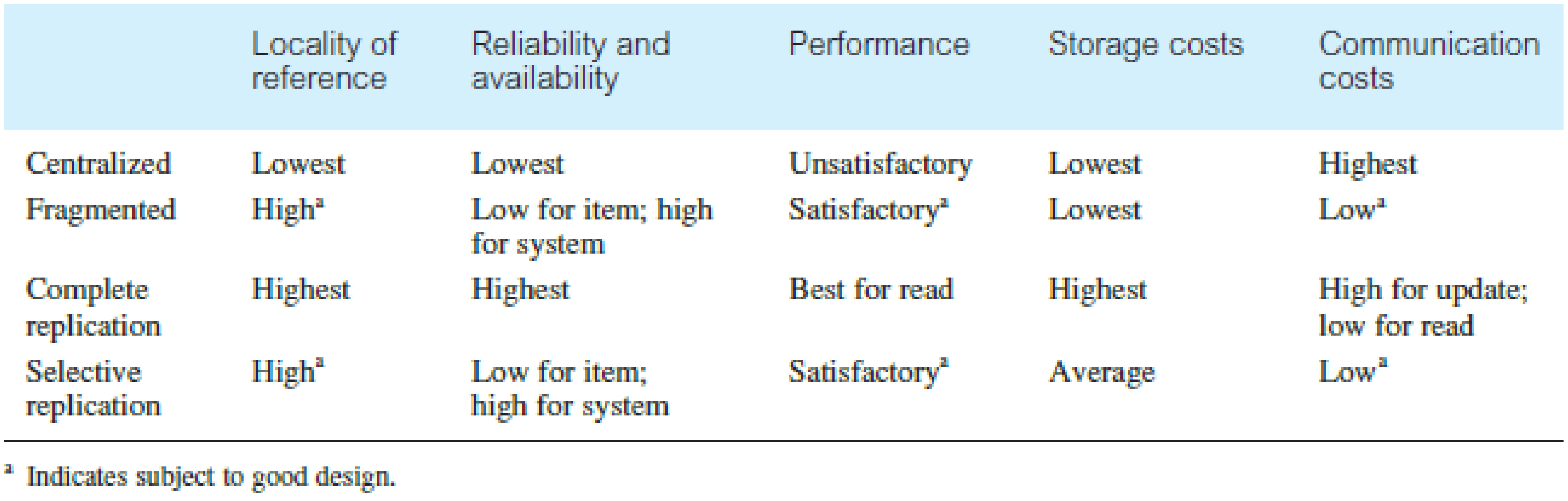
Co-funded by the
Erasmus+ Programme
of the European Union
MASTIS

LECTION 2

Co-funded by the Erasmus+ Programme of the European Union
MASTIS

Data Allocation

- Centralized
- Fragmented (or partitioned)
- Complete replication
- Selective replication



	Locality of reference	Reliability and availability	Performance	Storage costs	Communication costs
Centralized	Lowest	Lowest	Unsatisfactory	Lowest	Highest
Fragmented	High[a]	Low for item; high for system	Satisfactory[a]	Lowest	Low[a]
Complete replication	Highest	Highest	Best for read	Highest	High for update; low for read
Selective replication	High[a]	Low for item; high for system	Satisfactory[a]	Average	Low[a]

[a] Indicates subject to good design.

Fragmentation – Why?

- **Usage.** In general, applications work with views rather than entire relations. Therefore, for data distribution, it seems appropriate to work with subsets of relations as the unit of distribution
- **Efficiency.** Data is stored close to where it is most frequently used. In addition, data that is not needed by local applications is not stored
- **Parallelism.** With fragments as the unit of distribution, a transaction can be divided into several subqueries that operate on fragments. This should increase the degree of concurrency, or parallelism, in the system thereby allowing transactions that can do so safely to execute in parallel
- **Security.** Data not required by local applications is not stored and consequently not available to unauthorized users

Fragmentation - Disadvantages

- **Performance.** The performance of global applications that require data from several fragments located at different sites may be slower
- **Integrity.** Integrity control may be more difficult if data and functional dependencies are fragmented and located at different sites

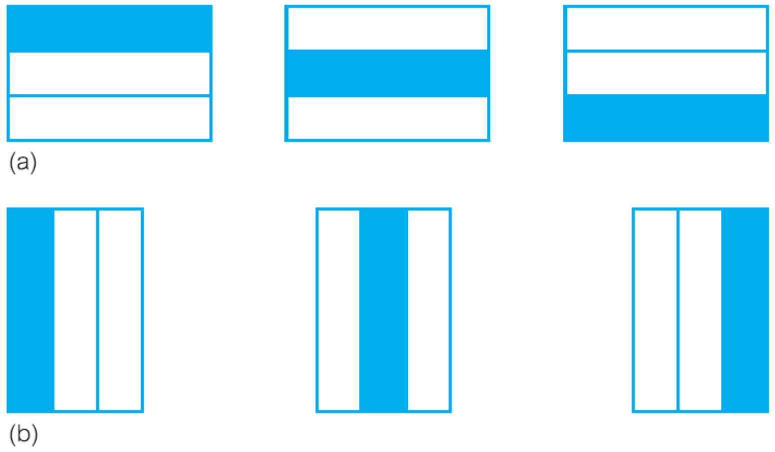
Correctness of fragmentation

There are three rules that must be followed during fragmentation:

- **Completeness.** If a relation instance R is decomposed into fragments R1, R2, . . . , Rn, each data item that can be found in R must appear in at least one fragment. This rule is necessary to ensure that there is no loss of data during fragmentation
- **Reconstruction.** It must be possible to define a relational operation that will reconstruct the relation R from the fragments. This rule ensures that functional dependencies are preserved
- **Disjointness.** If a data item di appears in fragment Ri, then it should not appear in any other fragment. Vertical fragmentation is the exception to this rule, where primary key attributes must be repeated to allow reconstruction. This rule ensures minimal data redundancy

Types of fragmentation

- **Horizontal** - consists of a subset of the tuples of a relation
- **Vertical** - consists of a subset of the attributes of a relation
- **No Fragmenatation**



Horizontal fragmentation

- Horizontal fragmentation groups together the tuples in a relation that are collectively used by the important transactions
- A horizontal fragment is produced by specifying a predicate that performs a restriction on the tuples in the relation

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

Horizontal fragmentation

Fragment P_1

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo	branchNo
PA14	16 Holhead	Aberdeen	AB7 5SU	House	6	650	CO46	SA9	B007
PG21	18 Dale Rd	Glasgow	G12	House	5	600	CO87	SG37	B003

Fragment P_2

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo	branchNo
PL94	6 Argyll St	London	NW2	Flat	4	400	CO87	SL41	B005
PG4	6 Lawrence St	Glasgow	G11 9QX	Flat	3	350	CO40	SG14	B003
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375	CO93	SG37	B003
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	CO93	SG14	B003

Horizontal fragmentation

- *Completeness* Each tuple in the relation appears in either fragment P1 or P2
- *Reconstruction* The PropertyForRent relation can be reconstructed from the fragments using the Union operation, thus:
  P1 ∪ P2 = PropertyForRent
- *Disjointness* The fragments are disjoint; there can be no property type that is both 'House' and 'Flat'

Vertical fragmentation

- Vertical fragmentation groups together the attributes in a relation that are used jointly by the important transactions

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

- *Completeness* Each attribute in the Staff relation appears in either fragment S1 or S2.
- *Reconstruction* The Staff relation can be reconstructed from the fragments using the Natural join operation, thus: $S1 \bowtie S2 = \text{Staff}$
- *Disjointness* The fragments are disjoint except for the primary key, which is necessary for reconstruction

Mixed fragmentation

- **Mixed fragment** - consists of a horizontal fragment that is subsequently vertically fragmented, or a vertical fragment that is then horizontally fragmented

$$\sigma_p(\Pi_{a_1, \ldots, a_n}(R))$$

or

$$\Pi_{a_1, \ldots, a_n}(\sigma_p(R))$$

Co-funded by the
Erasmus+ Programme
of the European Union

Derived horizontal fragmentation

- Some applications may involve a join of two or more relations. If the relations are stored at different locations, there may be a significant overhead in processing the join. In such cases, it may be more appropriate to ensure that the relations, or fragments of relations, are at the same location. We can achieve this using derived horizontal fragmentation
- **Derived fragment** - is a horizontal fragment that is based on the horizontal fragmentation of a parent relation

Transparencies in a DDBMS

Four main types of transparency in a DDBMS:

- distribution transparency
- transaction transparency
- performance transparency
- DBMS transparency

Distribution Transparency

- Distribution transparency allows the user to perceive the database as a single, logical entity
- If a DDBMS exhibits distribution transparency, then the user does not need to know the data is fragmented (**fragmentation transparency**) or the location of data items (**location transparency**)

Distribution Transparency

- **Fragmentation transparency** (highest level) - user does not need to know that the data is fragmented
- **Location transparency** (middle level) - user must know how the data has been fragmented but still does not have to know the location of the data
- **Replication transparency** means that user is unaware of the replication of fragments
- **Local mapping transparency** (lowest level) - the user needs to specify both fragment names and the location of data items, taking into consideration any replication that may exist

Naming transparency

- As in a centralized database, each item in a distributed database must have a unique name. One solution to this problem is to create a central **name server**, which has the responsibility for ensuring uniqueness of all names in the system. However, this approach results in:
  - loss of some local autonomy
  - performance problems, if the central site becomes a bottleneck
  - low availability; if the central site fails, the remaining sites cannot create any new database objects
- An alternative solution is to prefix an object with the identifier of the site that created it. However, this results in loss of distribution transparency.
- An approach that resolves the problems with both these solutions uses **aliases** (sometimes called **synonyms**) for each database object

Transaction Transparency

- Transaction transparency in a DDBMS environment ensures that all distributed transactions maintain the distributed database's integrity and consistency. A **distributed transaction** accesses data stored at more than one location. Each transaction is divided into a number of **subtransactions**, one for each site that has to be accessed; a subtransaction is represented by an **agent**
- We consider two further aspects of transaction transparency:
  - **concurrency transparency**
  - **failure transparency**

Concurrency transparency

- Concurrency transparency is provided by the DDBMS if the results of all concurrent transactions (distributed and non-distributed) execute *independently* and are logically *consistent* with the results that are obtained if the transactions are executed one at a time, in some arbitrary serial order

Failure transparency

In the distributed environment, the DDBMS must also cater for:

- loss of a message
- failure of a communication link
- failure of a site
- network partitioning

Classification of transactions

Distributed Relational DatabaseArchitecture (DRDA) defines four types of transaction (progressive level of complexity):

- remote request
- remote unit of work
- distributed unit of work
- distributed request

Classification of transactions

- **Remote request.** An application at one site can send a request (SQL statement) to some remote site for execution. The request is executed entirely at the remote site and can reference data only at the remote site
- **Remote unit of work.** An application at one (local) site can send all the SQL statements in a unit of work (transaction) to some remote site for execution. All SQL statements are executed entirely at the remote site and can only reference data at the remote site. However, the local site decides whether the transaction is to be committed or rolled back

Classification of transactions

- **Distributed unit of work.** An application at one (local) site can send some of or all the SQL statements in a transaction to one or more remote sites for execution. Each SQL statement is executed entirely at the remote site and can only reference data at the remote site. However, different SQL statements can be executed at different sites. Again, the local site decides whether the transaction is to be committed or rolled back
- **Distributed request.** An application at one (local) site can send some of or all the SQL statements in a transaction to one or more remote sites for execution. However, an SQL statement may require access to data from more than one site (for example, the SQL statement may need to join or union relations/fragments located at different sites)

Performance Transparency

- Performance transparency requires a DDBMS to perform as if it were a centralized DBMS. Distributed query processor (DQP) maps a data request into an ordered sequence of operations on the local databases. It has to decide:
  - which fragment to access
  - which copy of a fragment to use, if the fragment is replicated
  - which location to use