CSE-4845 || **Distributed Database**|| Mid Term Note
by **Sorowar Mahabub**, C201032

Seg:1

“The standardization efforts in databases developed reference models of DBMS”- in this point of view, what is the meaning of Reference Model? What are the different approaches for the A reference Model?

Reference Model: A conceptual framework whose purpose is to divide standardization work into manageable pieces and to show at a general level how these pieces are related to each other. A reference model can be thought of as an idealized architectural model of the system.

A reference model can be described according to 3 different approaches:

– **component-based:** Components of the system are defined together with the interrelationships between the components → Good for design and implementation of the system → It might be difficult to determine the functionality of the system from its components.

– **function-based:** Classes of users are identified together with the functionality that the system will provide for each class → Typically a hierarchical system with clearly defined interfaces between different layers – The objectives of the system are identified. → Not clear how to achieve the objectives → Example: ISO/OSI architecture of computer networks.

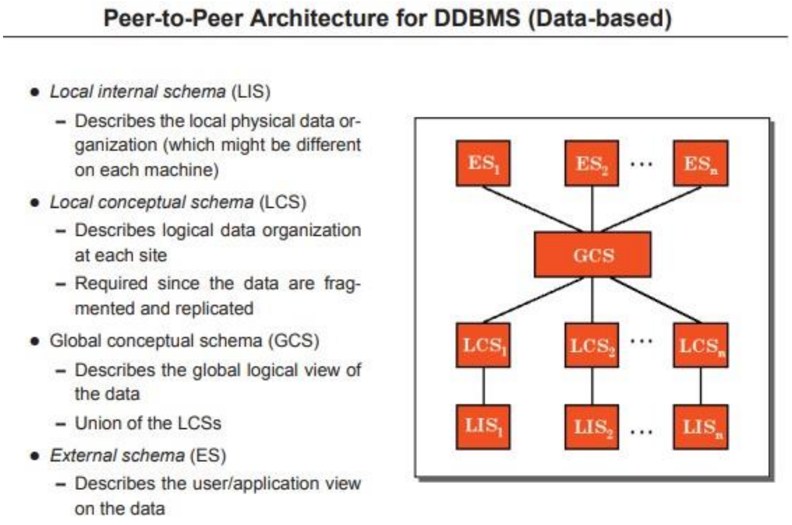
– **data-based:** Identify the different types of the data and specify the functional units that will realize and/or use data according to these views → Gives central importance to data (which is also the central resource of any DBMS) → Claimed to be the preferable choice for standardization of DBMS → The full architecture of the system is not clear without the description of functional modules. → Example: ANSI/SPARC architecture of DBMS.

----**Write down the Peer-to-Peer Architecture for DDBMS according to the function-based approach.**

---pic

.

----**Write down the Peer-to-Peer Architecture for DDBMS according to the data-based approach.**



Peer-to-Peer Architecture for DDBMS (Data-based)

- *Local internal schema* (LIS)
  - Describes the local physical data organization (which might be different on each machine)
- *Local conceptual schema* (LCS)
  - Describes logical data organization at each site
  - Required since the data are fragmented and replicated
- Global conceptual schema (GCS)
  - Describes the global logical view of the data
  - Union of the LCSs
- *External schema* (ES)
  - Describes the user/application view on the data

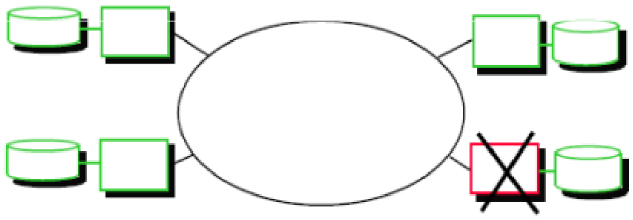
.

----**Write down the function goals of Distributed database management system.**

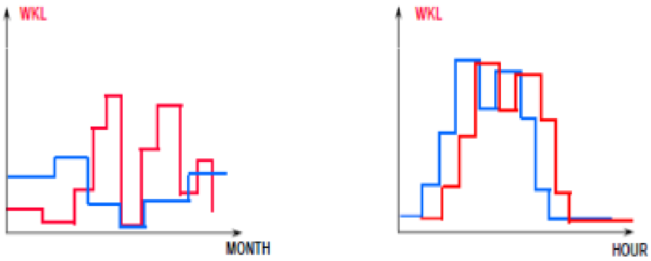
- Availability

- Load Sharing

- REDUNDANT HW/SW RESOURCES CAN BE USED TO OBTAIN AN OVERALL SYSTEM HIGHER AVAILABILITY
  - FAULT TOLERANT SYSTEMS
  - SOFT DEGRADATION SYSTEMS



IT ALLOWS A BALANCED RESOURCES DEVELOPMENT



- Quality of Service to the User

- **RESOURCE SHARING**

  SPECIALIZED OR UNIQUE RESOURCES CAN BE SHARED AT WHICHEVER NODE

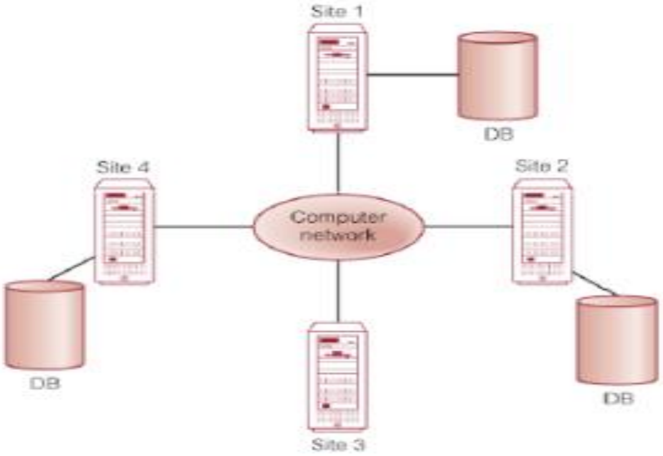
- **QUALITY OF SERVICE IMPROVEMENT**
  - LOCAL PROCESSING CAPABILITIES
  - RESPONSE TIME REDUCTION
  - USER FRIENDLY INTERFACE

- Resource Sharing

.

----**What are the characteristics of Distributed database management system? An Example is DD.**

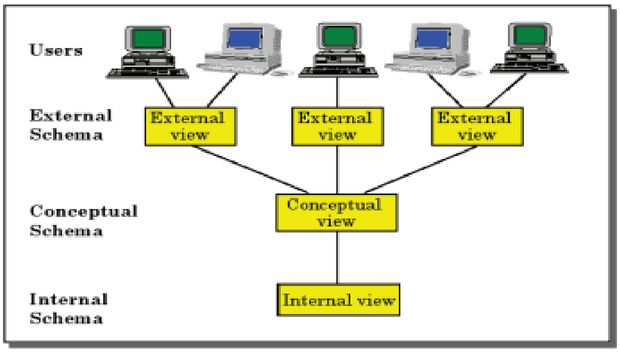
- Collection of logically-related shared data.
- Data split into fragments.
- Fragments may be replicated.
- Fragments/replicas allocated to sites.
- Sites linked by a communications network.
- Data at each site is under control of a DBMS.
- DBMSs handle local applications autonomousl
- Each DBMS participates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



.

----**What do you know about the ANSI/SPARC Architecture of DBMS**? Describe it.

- ANSI/SPARC architecture is based on data
- 3 views of data: external view, conceptual view, internal view
- Defines a total of 43 interfaces between these views



Seg-2

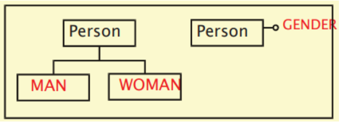
----**Write down the steps of view integration for a unique DB**.

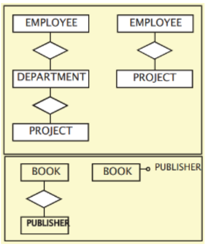
(Mixed strategy)

1. Subsystem (functionality) identification 2. Design of the skeleton schema 3. Subschema (view) conceptual design 4. View integration and restructuring 5. Conceptual to logical translation (of the global schema, of the single subschema) 6. Reconciliation of the global logical schema with the single schemata (logical view definition

----**Conflict analysis**: Conflict analysis in distributed databases refers to the process of identifying and resolving conflicts that may arise when multiple transactions are executed concurrently on different nodes of a distributed database system.

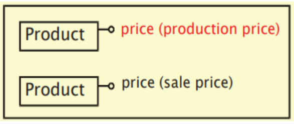
----**Structure Conflicts**: In the context of distributed databases, a structural conflict occurs when related real-world concepts are modeled using different constructs in different schemas. This can happen when integrating multiple schemas into a global schema that describes all the data in existing databases participating in a distributed or federated database management system.



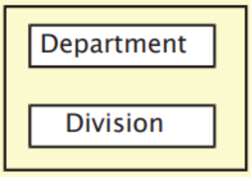


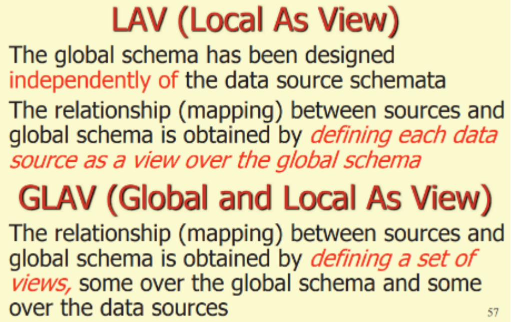
----**Name Conflicts**: In the context of distributed databases, a name conflict can occur when two or more data sources have the same name for different objects, such as tables, columns, or schemas. This can cause problems when integrating data from multiple sources into a single global schema, as it can lead to ambiguity and confusion. There are several approaches to resolving these conflicts, including schema integration and data transformation techniques.

1. **HOMONYMS**: In the context of distributed databases, a homonym name conflict occurs when two or more data sources use the same name to refer to different objects, such as tables, columns, or schemas. This can cause problems when integrating data from multiple sources into a single global schema, as it can lead to ambiguity and confusion. To resolve this homonym name conflict, schema integration techniques can be used to merge the different schemas into a single, consistent global schema.



2. **SYNONIMS**: A synonym name conflict occurs when two or more data sources use different names to refer to the same object, such as tables, columns, or schemas. This can cause problems when integrating data from multiple sources into a single global schema, as it can lead to redundancy and inconsistency. There are several approaches to resolving these conflicts, including schema integration techniques that involve renaming objects or defining mappings between objects in different schemas.





LAV (Local As View)

The global schema has been designed *independently of* the data source schemata

The relationship (mapping) between sources and global schema is obtained by *defining each data source as a view over the global schema*

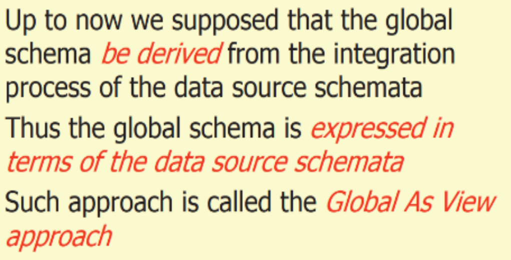
GLAV (Global and Local As View)

The relationship (mapping) between sources and global schema is obtained by *defining a set of views,* some over the global schema and some over the data sources

57

Dependency Conflicts: See at last page.

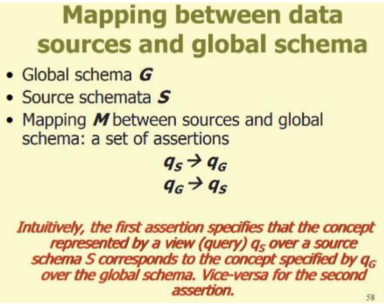
GAV & LAV: ---**GAV** stands for Global as View, which is one of the mediator types of view-based data integration in distributed databases. In GAV, the global schema acts as a view over the source schema, meaning that the mediator schema is described in terms of the local schema. When a query is made over the global schema, the mediator will follow existing rules and templates to convert the query into source-specific queries, which are then sent to wrappers for execution.



Up to now we supposed that the global schema *be derived* from the integration process of the data source schemata

Thus the global schema is *expressed in terms of the data source schemata*

Such approach is called the *Global As View approach*



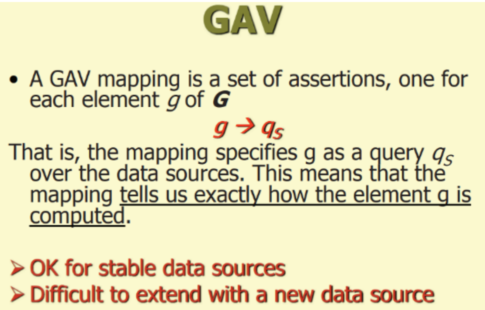
Mapping between data sources and global schema

- Global schema ***G***
- Source schemata ***S***
- Mapping ***M*** between sources and global schema: a set of assertions

$$q_S \rightarrow q_G$$
$$q_G \rightarrow q_S$$

Intuitively, the first assertion specifies that the concept represented by a view (query) q_S over a source schema S corresponds to the concept specified by q_G over the global schema. Vice-versa for the second assertion.

58



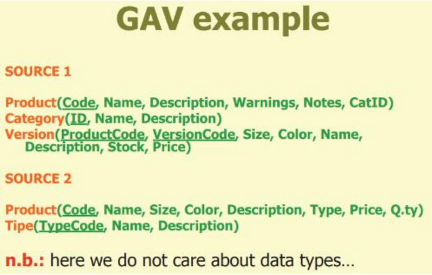
GAV

- A GAV mapping is a set of assertions, one for each element *g* of ***G***

$$g \rightarrow q_S$$

That is, the mapping specifies g as a query q_S over the data sources. This means that the mapping tells us exactly how the element g is computed.

➢ OK for stable data sources
➢ Difficult to extend with a new data source



GAV example

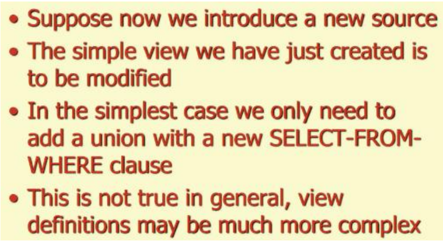
SOURCE 1

Product(Code, Name, Description, Warnings, Notes, CatID)
Category(ID, Name, Description)
Version(ProductCode, VersionCode, Size, Color, Name, Description, Stock, Price)

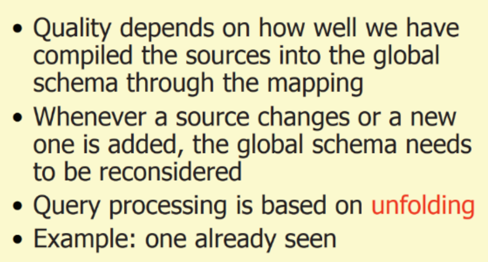
SOURCE 2

Product(Code, Name, Size, Color, Description, Type, Price, Q.ty)
Tipe(TypeCode, Name, Description)

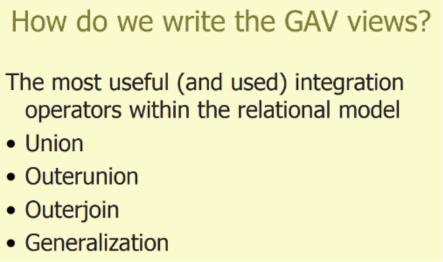
n.b.: here we do not care about data types…



- Suppose now we introduce a new source
- The simple view we have just created is to be modified
- In the simplest case we only need to add a union with a new SELECT-FROM-WHERE clause
- This is not true in general, view definitions may be much more complex



- Quality depends on how well we have compiled the sources into the global schema through the mapping
- Whenever a source changes or a new one is added, the global schema needs to be reconciled
- Query processing is based on unfolding
- Example: one already seen

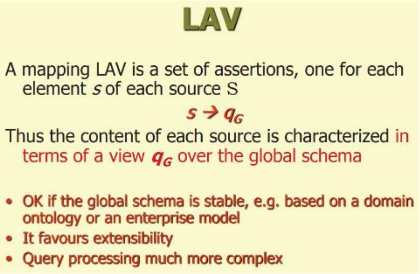


How do we write the GAV views?

The most useful (and used) integration operators within the relational model

- Union
- Outerunion
- Outerjoin
- Generalization

---**LAV** stands for Local as View, which is one of the mediator types of view-based data integration in distributed databases. In LAV, the source schema acts as a view over the global schema, meaning that the local schema is described in terms of the mediator schema. When a query is made ov er the global schema, the mediator will follow existing rules and templates to convert the query into source-specific queries, which are then sent to wrappers for execution.



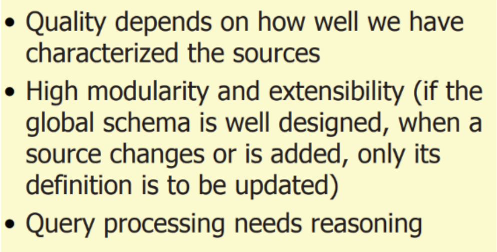
LAV

A mapping LAV is a set of assertions, one for each element *s* of each source S

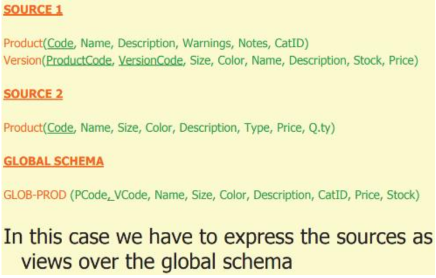
$s \rightarrow q_G$

Thus the content of each source is characterized in terms of a view q_G over the global schema

- OK if the global schema is stable, e.g. based on a domain ontology or an enterprise model
- It favours extensibility
- Query processing much more complex



- Quality depends on how well we have characterized the sources
- High modularity and extensibility (if the global schema is well designed, when a source changes or is added, only its definition is to be updated)
- Query processing needs reasoning



SOURCE 1

Product(Code, Name, Description, Warnings, Notes, CatID)
Version(ProductCode, VersionCode, Size, Color, Name, Description, Stock, Price)

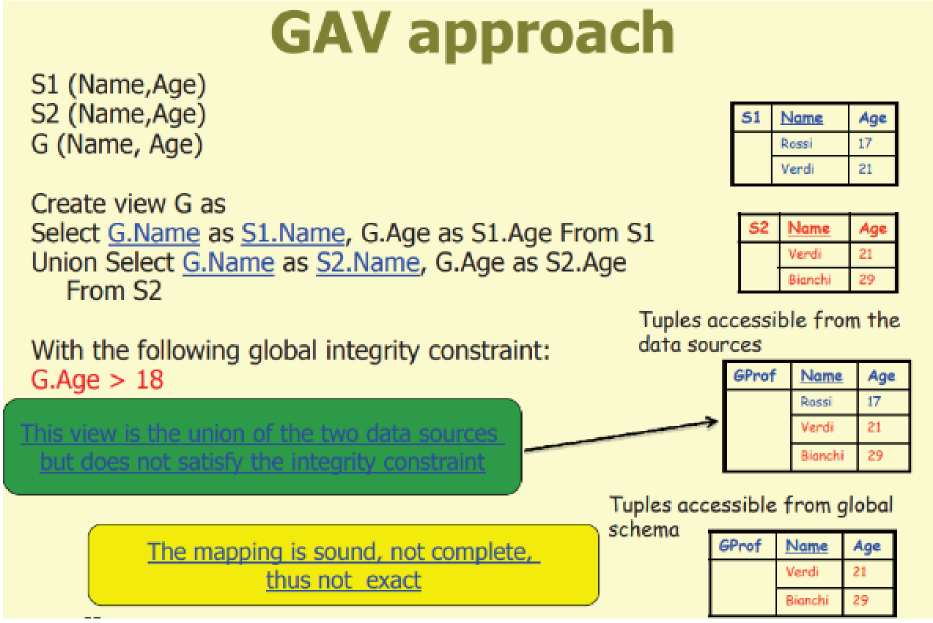
SOURCE 2

Product(Code, Name, Size, Color, Description, Type, Price, Q.ty)

GLOBAL SCHEMA

GLOB-PROD (PCode, VCode, Name, Size, Color, Description, CatID, Price, Stock)

In this case we have to express the sources as views over the global schema



GAV approach

S1 (Name,Age)
S2 (Name,Age)
G (Name, Age)

Create view G as
Select G.Name as S1.Name, G.Age as S1.Age From S1
Union Select G.Name as S2.Name, G.Age as S2.Age
From S2

With the following global integrity constraint:
G.Age > 18

S1	Name	Age
	Rossi	17
	Verdi	21

S2	Name	Age
	Verdi	21
	Bianchi	29

Tuples accessible from the data sources

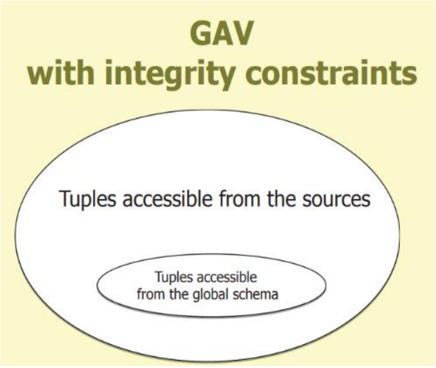
GProf	Name	Age
	Rossi	17
	Verdi	21
	Bianchi	29

This view is the union of the two data sources but does not satisfy the integrity constraint

Tuples accessible from global schema

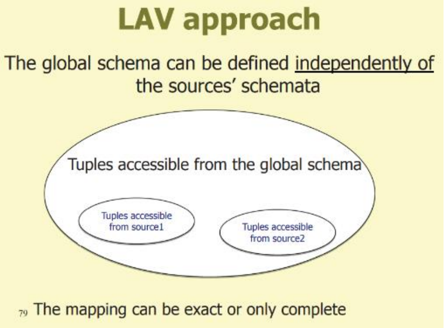
GProf	Name	Age
	Verdi	21
	Bianchi	29

The mapping is sound, not complete, thus not exact



GAV with integrity constraints

Tuples accessible from the sources
Tuples accessible from the global schema



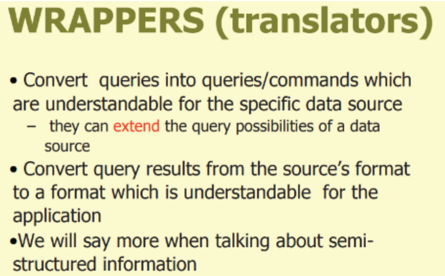
LAV approach

The global schema can be defined independently of the sources' schemata

Tuples accessible from the global schema
Tuples accessible from source1
Tuples accessible from source2

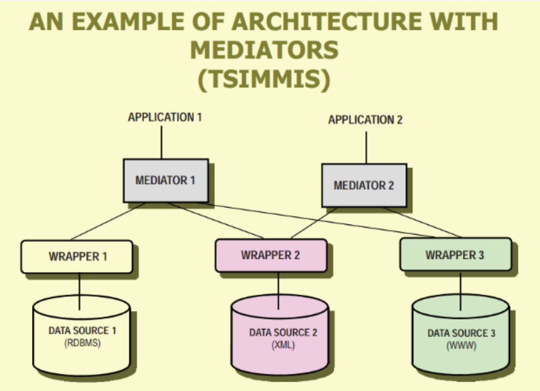
79 The mapping can be exact or only complete

Wrappers: In the context of distributed databases, a wrapper is a software component that provides a uniform interface to another software component that would otherwise be incompatible. Wrappers are used to bridge the gap between different software components, allowing them to work together seamlessly. Wrappers are the components of a data integration system that communicates with the data sources. The wrapper's task involves sending queries from the higher levels of the data integration system to the sources and then converting the replies to a format that can be manipulated by the query processor. The complexity of the wrapper depends on the nature of the data source.



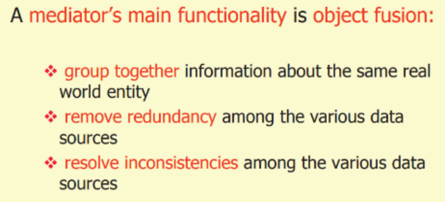
WRAPPERS (translators)

- Convert queries into queries/commands which are understandable for the specific data source
  - they can extend the query possibilities of a data source
- Convert query results from the source's format to a format which is understandable for the application
- We will say more when talking about semi-structured information



AN EXAMPLE OF ARCHITECTURE WITH MEDIATORS (TSIMMIS)

Mediators: In the context of distributed databases, a mediator is a software component that facilitates communication and coordination between multiple other software components. Mediators are used in situations where multiple components need to work together but cannot do so directly due to differences in their interfaces or communication protocols. The mediator acts as an intermediary, translating messages and coordinating interactions between the different components. Mediators can be used to map between data sources and a global schema, allowing them to work together seamlessly.



A mediator's main functionality is object fusion:

- ❖ group together information about the same real world entity
- ❖ remove redundancy among the various data sources
- ❖ resolve inconsistencies among the various data sources

--Q1: What is Wrappers? Define Mediators along with their functionality. How could we map between Data sources and Global Schema? 3Consider the following Data sources: -
SOURCE 1: Product (Code, Name, Description, Warnings, Notes, CatID)
Version (ProductCode, VersionCode, Size, Color, Name, Description, Stock, Price)
SOURCE 2: Product (Code, Name, Size, Color, Description, Type, Price, Q.ty) [AU-21]

----ANS: --A wrapper is a piece of software that provides a simplified interface to another software component or library. It can be used to encapsulate and simplify the use of complex APIs, or to provide compatibility between different software components.

--A mediator is a design pattern that promotes loose coupling between objects by providing a centralized communication channel between them. The mediator pattern is used to reduce the complexity of communication between objects by encapsulating the interactions between them in a separate mediator object. This allows objects to communicate with each other indirectly, through the mediator, rather than directly with each other.

--In data integration, mapping between data sources and a global schema is achieved using schema mappings. Schema mappings establish semantic connections between schemas and are used to describe the relationship between the schemas and their instances.

--In the example, we have two data sources with different schemas for the Product entity. To map between these data sources and a global schema, we will need to define schema mappings that specify how the attributes in each source schema correspond to the attributes in the global schema. For example, we may specify that the Code attribute in SOURCE 1 corresponds to the Code attribute in SOURCE 2 and in the global schema, while the Name attribute in SOURCE 1 corresponds to the Name attribute in SOURCE 2 and in the global schema. Similarly, we will need to define mappings for all other attributes to fully integrate these data sources into a global schema.

--Q2: According to the given data sources you need to create the Global as Views (GAV). [AU-21]

----ANS: --Global as View (GAV) is a type of view-based data integration where the global schema acts as a view over the source schemas. In other words, the mediator schema is described in terms of the local schemas. Given a query over the global schema, the mediator will follow existing rules and templates to convert the query into source-specific queries.

--In the example, we have two data sources with different schemas for the Product entity. To create a GAV for these data sources, we will need to define a global schema that acts as a view over the source schemas. This global schema would describe the Product entity in terms of the attributes present in the source schemas. For example, we might define a global schema for the Product entity as follows:

```
--Product (Code, Name, Description, Warnings, Notes, CatID, Size, Color, Type,
Price, Q.ty)
```

--This global schema includes all of the attributes present in the source schemas for the Product entity. We would then need to define rules and templates that specify how queries over this global schema should be converted into queries over the source schemas. For example, we might specify that a query for all products with a certain Code value should be translated into two queries: one for SOURCE 1 and one for SOURCE 2. These queries would retrieve all products with the specified Code value from each source and combine them into a single result set.

--Similarly, we will need to define rules and templates for all other types of queries that could be issued over the global schema to fully integrate these data sources using GAV.

----**Challenges of Data Integration**:

Data integration in distributed databases (DDB) can be challenging due to several reasons. Some of the common challenges include:

Lack of Planning: Data is only as useful as the operations it is being used for. Without proper planning, data integration can become a complex and time-consuming process.

Using Manual Data Integration: Manual data integration can be error-prone and time-consuming. It is important to use automated tools and processes to ensure data accuracy and consistency.

Lack of Scalability: As the amount of data and the number of data sources grow, it is important to have a scalable data integration solution that can handle the increased volume and complexity.

Low-Quality Data: Low-quality data can negatively impact the accuracy and reliability of integrated data. It is important to have processes in place to ensure data quality.

Duplicated Data: Duplicated data can lead to inconsistencies and inaccuracies in the integrated data. It is important to have processes in place to identify and eliminate duplicate data.

Data in the Wrong Format: Data may be stored in different formats in different data sources. It is important to have processes in place to transform data into a common format for integration.

Data Not Available When Needed: Data may not always be available when it is needed for integration. It is important to have processes in place to ensure that data is available when it is needed.

----**Design Problems in Distributed Systems**:

Designing distributed systems comes with several challenges:

Communication: Managing communication between nodes efficiently while considering latency, bandwidth, and reliability.

Consistency: Ensuring data consistency across distributed nodes despite failures and delays.

Fault Tolerance: Designing for system resilience to node failures without compromising overall functionality.

Scalability: Making the system easily expandable as the load and user base grow.

Concurrency Control: Handling simultaneous access to shared resources without conflicts.

Data Distribution: Deciding how data is distributed, replicated, and allocated across nodes.

Security: Ensuring data security and access control across distributed nodes.

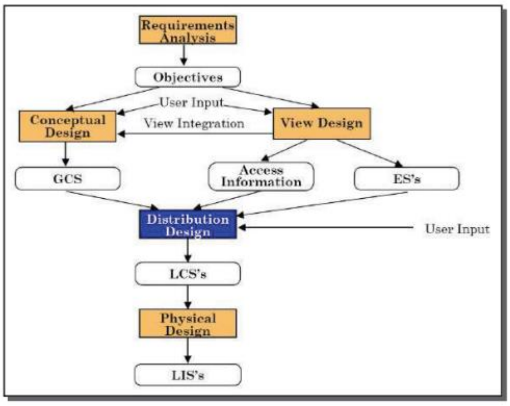
Synchronization: Managing synchronization of processes and data across nodes.

Seg-3

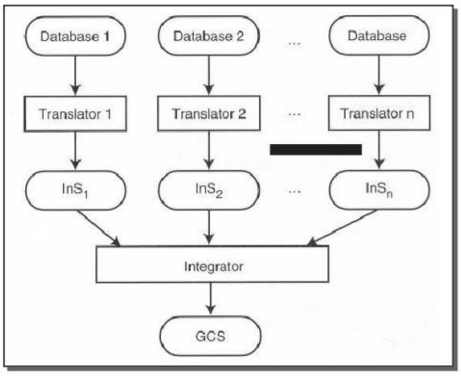
Distributed Design Strategies

Top-down approach	**Bottom-up approach**
1. Designing systems from scratch 2. Homogeneous systems	1. The databases already exist at a number of sites. 2. The databases should be connected to solve common tasks

- **Top-down design strategy**



- **Bottom-up design strategy**



[redacted]

[Data Fragmentation, Replication, and Allocation Techniques for Distributed Database Design (brainkart.com)]

Fragmentation

Fragmentation is a process of dividing the whole or full database into various sub-tables or sub-relations so that data can be stored in different systems. The small pieces or sub-relations or sub-tables are called *fragments*.

Q: What is Replication and Allocation? What is a reasonable unit of distribution? Relation or fragment of relation? SP-22 5

Data Replication:

Data replication is the process of storing separate copies of the database at two or more sites. It is a popular fault tolerance technique for distributed databases.

Data Allocation:

Data allocation refers to the process of deciding where to store data within a distributed database system. This involves determining which data fragments or replicas should be placed on which nodes or servers. The goal is to optimize factors such as access performance, data distribution, load balancing, and fault tolerance.

Relations as unit of distribution:

1. If the relation is not replicated, we get a high volume of remote data accesses.
2. If the relation is replicated, we get unnecessary replications, which cause problems in executing updates and waste disk space.
3. Might be an Ok solution, if queries need all the data in the relation and data stays at the only sites that uses the data.

Fragments of relationas as unit of distribution:

1. Application views are usually subsets of relations
2. Thus, the locality of accesses of applications is defined on subsets of relations.
3. Permits a number of transactions to execute concurrently, since they will access different portions of a relation.
4. Parallel execution of a single query (intra-query concurrency)
5. However, semantic data control (especially integrity enforcement) is more difficult.

Fragments of relations are (usually) the appropriate unit of distribution.

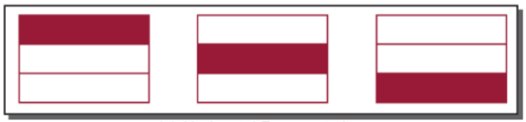
Q: What are the design problems of distributed system? What is the Fragment allocation? How can we define Horizontal and Vertical Fragmentation?

Design problem of distributed systems: Making decisions about the placement of data and programs across the sites of a computer network as well as possibly designing the network itself.

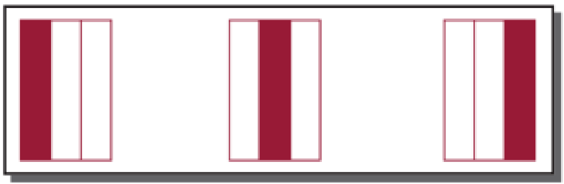
Fragment allocation refers to the process of determining how to divide and distribute the data of a database across multiple nodes or servers in a distributed system.

Types of Fragmentation

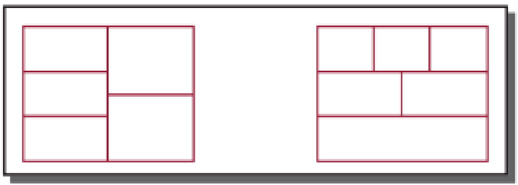
1. **Horizontal**: partitions a relation along its tuples
2. **Vertical**: partitions a relation along its attributes
3. **Mixed/hybrid**: a combination of horizontal and vertical fragmentation



(a) Horizontal Fragmentation



(b) Vertical Fragmentation



(c) Mixed Fragmentation

Horizontal Fragmentation:

Horizontal fragmentation is a database design technique in which a table is divided into smaller subsets or fragments based on the values of certain attributes. Each fragment contains a subset of the rows from the original table that satisfy a specific condition or range of values. This technique is often used to distribute related data across multiple nodes in a distributed database system, allowing for improved parallelism and performance. Horizontal fragmentation is particularly useful when data can be naturally partitioned, such as geographically or by time periods.

Vertical Fragmentation:

Vertical fragmentation is a database design technique where a table is divided into smaller subsets or fragments based on the columns of the table. Each fragment contains a subset of the columns for each row, typically including a common primary key or identifier column. Vertical fragmentation is used when different subsets of data need to be accessed independently, or when some columns are accessed more frequently than others. This technique can help optimize data storage and retrieval, as well as reduce data transfer overhead in distributed database systems.

Q: Write down the steps of view integrations for unique database? Discuss about the different types of conflict analysis? SP-22?

View Integration Steps for a Unique Database:

Understanding: Grasp database structure and user requirements.

Design: Develop SQL queries for intended views.

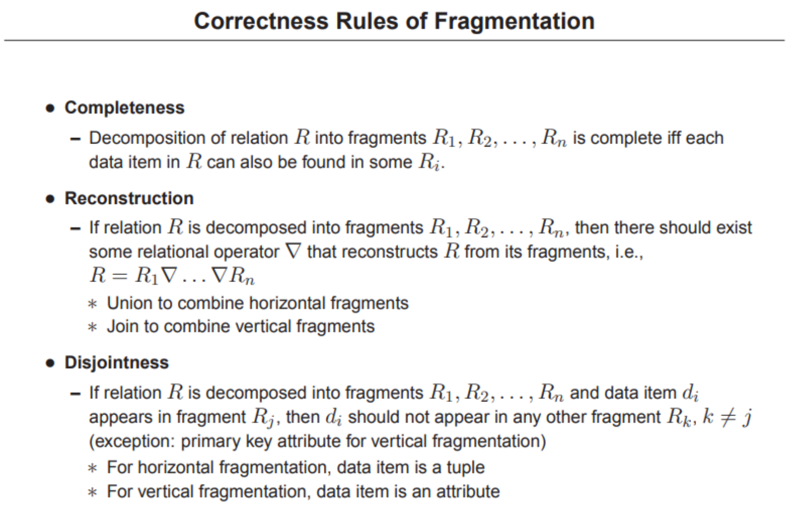
Implementation: Utilize CREATE VIEW to define views.

Security: Apply access controls and permissions.

Testing: Verify views for accurate data presentation.
Optimization: Enhance performance as necessary.
Documentation: Provide clear guidance for users.
Maintenance: Update views according to changes.
Education: Train users on utilizing the views.
Monitoring: Regularly assess views for issues and feedback.

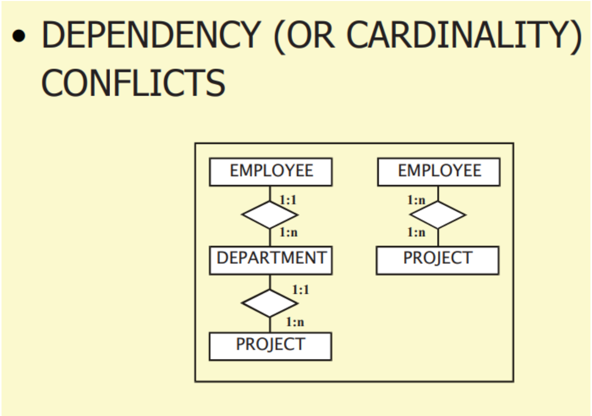
Types of Conflict Analysis:

Naming Conflicts: Different views use the same names for attributes or views, causing confusion.
Data Type Conflicts: Views interpret data in underlying tables differently, leading to inconsistent data types.
Aggregation Conflicts: Views calculate aggregate functions differently, resulting in varying summary values.
Filtering Conflicts: Views apply different filters to the same data, causing data discrepancies.
Joining Conflicts: Views combine data from tables differently, affecting data accuracy.
Sorting Conflicts: Views present the same data in different orders, leading to inconsistency.
Duplication Conflicts: The same data appears redundantly in different views.
Inconsistency Conflicts: Views contradict each other due to varying transformation logic.
Temporal Conflicts: Views present data at different times, causing inconsistencies.
Normalization Conflicts: Different views interpret data normalization levels diversely.
Semantic Conflicts: Views interpret data meaning differently, leading to misunderstandings.



Correctness Rules of Fragmentation

- **Completeness**
  - Decomposition of relation R into fragments $R_1, R_2, \ldots, R_n$ is complete iff each data item in R can also be found in some R_i.
- **Reconstruction**
  - If relation R is decomposed into fragments $R_1, R_2, \ldots, R_n$, then there should exist some relational operator ∇ that reconstructs R from its fragments, i.e., $R = R_1 \nabla \ldots \nabla R_n$
    - * Union to combine horizontal fragments
    - * Join to combine vertical fragments
- **Disjointness**
  - If relation R is decomposed into fragments $R_1, R_2, \ldots, R_n$ and data item d_i appears in fragment R_j, then d_i should not appear in any other fragment R_k, $k \neq j$ (exception: primary key attribute for vertical fragmentation)
    - * For horizontal fragmentation, data item is a tuple
    - * For vertical fragmentation, data item is an attribute



- DEPENDENCY (OR CARDINALITY) CONFLICTS

Dependency Conflict:

In a distributed database system, dependency conflicts can arise when different components or parts of the system require different versions of the same library, framework, or other software dependency. These conflicts can lead to various issues, including system instability, unexpected behavior, or even failures. Here's a more detailed explanation of how dependency conflicts can occur in a distributed database context:

Distributed Nature: A distributed database system consists of multiple nodes, each with its own software stack. These nodes communicate and collaborate to provide a unified database service. Each node might have its own set of dependencies.

Shared Dependencies: Some dependencies might be shared across multiple nodes or components of the distributed database system. These dependencies can include database drivers, communication libraries, data serialization libraries, and more.

Dependency Versions: Different components or nodes might require specific versions of shared dependencies to function properly. For example, Node A might need Dependency X version 1.0, while Node B might need Dependency X version 2.0 due to API changes or bug fixes.

Prepared By:
Sorowar, Shahin, Emdadul, Asif.

The Data Integration problem

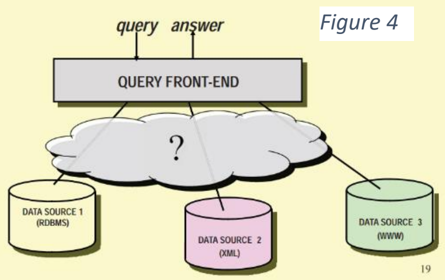


Figure 4

- Autonomy:
  - Design, or representation, autonomy: which data, and how
  - Communication autonomy: which services should be provided to the users or to the other DB systems
  - Execution autonomy: which algorithms for query processing and in general for data access

  which causes
- Heterogeneity:
  - Different platforms
  - Different data models
  - Different query languages
  - Different data schemas, i.e., modeling styles (*conflicts*…)
  - Different values for the same info (*inconsistency*)

Figure 2

The possible situations

- Also in a unique, centralized DB, there is a problem of integration
- Distributed or federated DB
  - Homogeneous data: ***same*** **data model**
  - Heterogeneous data: ***different*** **data models**
  - Semi-structured data
- The extreme case: data integration for transient, initially unknown data sources

Figure 3

An orthogonal classification

- Centralized architecture: the traditional architecture for centralized, virtual or materialized data integration
- Data exchange: pairwise exchange of data between two data sources
- Peer-to-peer: decentralized, dynamic, data-centric coordination between autonomous organizations

Figure 4

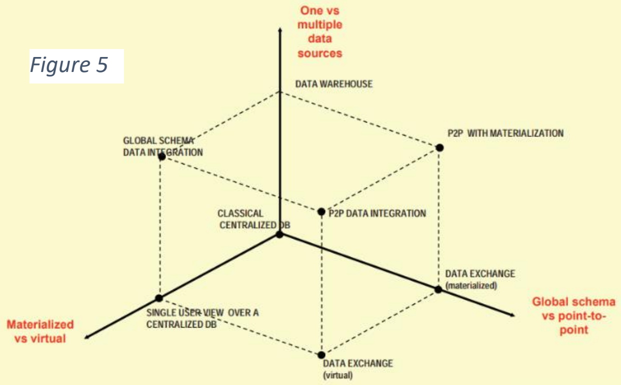
The typical problem of data design

Figure 7

- Given a data model
- Organizing data according to the chosen data model in order to:
  - Avoid inconsistencies
  - Allow *query optimization*
- In a unique DB:
  - only *schema heterogeity*

DATA INTEGRATION

Figure 5


One vs multiple data sources
DATA WAREHOUSE
GLOBAL SCHEMA DATA INTEGRATION
P2P WITH MATERIALIZATION
CLASSICAL CENTRALIZED DB
P2P DATA INTEGRATION
DATA EXCHANGE (materialized)
Global schema vs point-to-point
Materialized vs virtual
SINGLE USER VIEW OVER A CENTRALIZED DB
DATA EXCHANGE (virtual)

The possible solutions

Figure 6

- ➢ Data integration problems arise even in the simplest situation: unique, centralized DB…

…and it becomes more and more complex

…up to the extreme case of transient, dynamic, initially unknown data sources…

- ➢ We propose techniques and methods which should be applied, adding complication as the situation becomes more complex

Figure 8

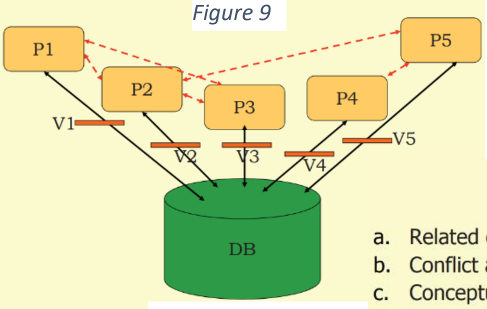
UNIQUE DB

Each datum, whatever application uses it, only appears once. This ELIMINATES useless REDOUNDANCIES, which would cause:

- Inconsistencies
- Useless memory occupation

An integrated DB

Figure 9



View Integration

a. Related concept identification
b. Conflict analysis and resolution
c. Conceptual Schema integration

Figure 30

→ Recall: in a unique DB we can only have *schema heterogeity*

→ This same operation is carried out for each of the integration problems, from the simplest (this one) to the most complex

Figure 12

Exercise

We want to define the database of a ski school having different sites in Italy. Each site has a name, a location, a phone number and an e-mail address. We want to store information about customers, employees and teachers (SSN, name, surname, birth date, address, and phone). For each teacher we store also the technique (cross-country, downhill, snow board). The personnel office has a view over the personnel data, that is, ski teachers and employees.

The school organizes courses in the different locations of the school. The course organization office has a view over these courses. The courses have a code (unique for each site), the starting date, the day of the week in which the course takes place, the time, and the kind of course (cross-country, downhill, snow board), the level, the number of lessons, the cost, the minimal age for participant. For each course, a unique teacher is associated, and the participants.

4.c Schema Integration

Figure 21

- Conflict resolution
- Production of a new conceptual schema which expresses (as much as possible) the same semantics as the schemata we wanted to integrate
- Production of the transformations between the original schemata and the integrated one: $V_1(DB)$, $V_2(DB)$,… $V_3(DB)$