Chapter 2: DDBMS Architecture

- Definition of the DDBMS Architecture
- ANSI/SPARC Standard
- Global, Local, External, and Internal Schemas, Example
- DDBMS Architectures
- Components of the DDBMS

Acknowledgements: I am indebted to Arturas Mazeika for providing me his slides of this course.
• **Architecture:** The architecture of a system defines its structure:
  – the components of the system are identified;
  – the function of each component is specified;
  – the interrelationships and interactions among the components are defined.

• Applies both for computer systems as well as for software systems, e.g,
  – division into modules, description of modules, etc.
  – architecture of a computer

• There is a close relationship between the architecture of a system, standardisation efforts, and a reference model.
Motivation for Standardization of DDBMS Architecture

- DDBMS might be implemented as homogeneous or heterogeneous DDBMS

- **Homogeneous** DDBMS
  - All sites use same DBMS product
  - It is much easier to design and manage
  - The approach provides incremental growth and allows increased performance

- **Heterogeneous** DDBMS
  - Sites may run different DBMS products, with possibly different underlying data models
  - This occurs when sites have implemented their own databases first, and integration is considered later
  - Translations are required to allow for different hardware and/or different DBMS products
  - Typical solution is to use gateways

⇒ A common standard to implement DDBMS is needed!
• The standardization efforts in databases developed reference models of DBMS.

• **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.

• Commercial systems might deviate from reference model, still they are useful for the standardization process

• A reference model can be described according to 3 different approaches:
  – component-based
  – function-based
  – data-based
**Components-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 clearly 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
• The interplay among the 3 approaches is important:
  – Need to be used together to define an architectural model
  – Each brings a different point of view and serves to focus on different aspects of the model
• 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
- Conceptual schema: Provides enterprise view of entire database

**RELATION EMP**

- **KEY** = {ENO}
- **ATTRIBUTES** = {
  - ENO : CHARACTER(9)
  - ENAME: CHARACTER(15)
  - TITLE: CHARACTER(10)
}

**RELATION PAY**

- **KEY** = {TITLE}
- **ATTRIBUTES** = {
  - TITLE: CHARACTER(10)
  - SAL : NUMERIC(6)
}

**RELATION PROJ**

- **KEY** = {PNO}
- **ATTRIBUTES** = {
  - PNO : CHARACTER(7)
  - PNAME : CHARACTER(20)
  - BUDGET : NUMERIC(7)
  - LOC : CHARACTER(15)
}

**RELATION ASG**

- **KEY** = {ENO, PNO}
- **ATTRIBUTES** = {
  - ENO : CHARACTER(9)
  - PNO : CHARACTER(7)
  - RESP: CHARACTER(10)
  - DUR : NUMERIC(3)
}
Example ...

- Internal schema: Describes the storage details of the relations.
  - Relation EMP is stored on an indexed file
  - Index is defined on the key attribute ENO and is called EMINX
  - A HEADER field is used that might contain flags (delete, update, etc.)

```
INTERNAL_REL EMP [  
  INDEX ON E# CALL EMINX  
  FIELD =  
    HEADER : BYTE (1)  
    E#      : BYTE (9)  
    ENAME   : BYTE (15)  
    TIT     : BYTE (10)  
]  
```

Conceptual schema:
```
RELATION EMP [  
  KEY = \{ENO\}  
  ATTRIBUTES = \{  
    ENO : CHARACTER (9)  
    ENAME: CHARACTER (15)  
    TITLE: CHARACTER (10)  
  \}  
]  
```
• External view: Specifies the view of different users/applications
  – Application 1: Calculates the payroll payments for engineers

```sql
CREATE VIEW PAYROLL (ENO, ENAME, SAL) AS
SELECT EMP.ENO, EMP.ENAME, PAY.SAL
FROM EMP, PAY
WHERE EMP.TITLE = PAY.TITLE
```

– Application 2: Produces a report on the budget of each project

```sql
CREATE VIEW BUDGET(PNAME, BUD) AS
SELECT PNAME, BUDGET
FROM PROJ
```
Architectural Models for DDBMSs (or more generally for multiple DBMSs) can be classified along three dimensions:

- Autonomy
- Distribution
- Heterogeneity
• **Autonomy**: Refers to the distribution of control (not of data) and indicates the degree to which individual DBMSs can operate independently.
  
  - *Tight integration*: a single-image of the entire database is available to any user who wants to share the information (which may reside in multiple DBs); realized such that one data manager is in control of the processing of each user request.
  
  - *Semiautonomous* systems: individual DBMSs can operate independently, but have decided to participate in a federation to make some of their local data sharable.
  
  - *Total isolation*: the individual systems are stand-alone DBMSs, which know neither of the existence of other DBMSs nor how to communicate with them; there is no global control.

• Autonomy has different dimensions
  
  - *Design autonomy*: each individual DBMS is free to use the data models and transaction management techniques that it prefers.
  
  - *Communication autonomy*: each individual DBMS is free to decide what information to provide to the other DBMSs
  
  - *Execution autonomy*: each individual DBMS can execute the transactions that are submitted to it in any way that it wants to.
• **Distribution:** Refers to the physical distribution of data over multiple sites.
  
  – *No distribution:* No distribution of data at all
  
  – *Client/Server distribution:*
    * Data are concentrated on the server, while clients provide application environment/user interface
    * First attempt to distribution
  
  – *Peer-to-peer distribution* (also called *full distribution*):
    * No distinction between client and server machine
    * Each machine has full DBMS functionality
• **Heterogeneity**: Refers to heterogeneity of the components at various levels
  - hardware
  - communications
  - operating system
  - DB components (e.g., data model, query language, transaction management algorithms)
Client-Server Architecture for DDBMS (Data-based)

- General idea: Divide the functionality into two classes:
  - server functions
    * mainly data management, including query processing, optimization, transaction management, etc.
  - client functions
    * might also include some data management functions (consistency checking, transaction management, etc.) not just user interface

- Provides a two-level architecture
- More efficient division of work
- Different types of client/server architecture
  - Multiple client/single server
  - Multiple client/multiple server
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
Fundamental difference to peer-to-peer DBMS is in the definition of the global conceptual schema (GCS)

- In a MDBMS the GCS represents only the collection of some of the local databases that each local DBMS want to share.

This leads to the question, whether the GCS should even exist in a MDBMS?

Two different architecture models:
- Models with a GCS
- Models without GCS
Multi-DBMS Architecture (Data-based) . . .

- Model with a GCS
  - GCS is the union of parts of the LCSs
  - Local DBMS define their own views on the local DB
Multi-DBMS Architecture (Data-based) . . .

- Model without a GCS
  - The local DBMSs present to the multi-database layer the part of their local DB they are willing to share.
  - External views are defined on top of LCSs
Regular DBMS (Component-based)
General DDBMS (Component-based)
Client-Server Architecture (Component-based)

- One server, many clients

![Diagram of Client-Server Architecture](image)

- Operating System
  - Operating System
  - Communication Software
  - Semantic Data Controller
  - Query Optimizer
  - Transaction Manager
  - Recovery Manager
  - Runtime Support Processor

- SQL queries
- Result relation

- Database

- User Interface
- Application Program
- Client DBMS
- Communication Software
Components of Client-Server Architecture (Component-based)

- Many servers, many clients

- directory
- caching
- query decomposition
- commit protocols

LAN

Applications
Client Services
Communications

Communications
DBMS Services
Database

Communications
DBMS Services
Database
Components of Client-Server Architecture (Component-based) …

- Many servers, many clients

![Diagram](image)

- SQL interface
- Programmatic interface
- Other application support environments

**Components Diagram**

- Applications
- Client Services
- Communications
- LAN
- Database
- DBMS Services
- Communications
Components of Peer-to-Peer Architecture (Component-based)
Components of Multi-DBMS Architecture (Component-based)
• Architecture defines the structure of the system. There are three ways to define the architecture: based on components, functions, or data

• DDBMS might be based on identical components (homogeneous systems) or different components (heterogeneous systems)

• ANSI/SPARC architecture defines external, conceptual, and internal schemas

• There are three orthogonal implementation dimensions for DDBMS: level of distribution, autonomy, and heterogeneity

• Different architectures are discussed:
  – Client-Server Systems
  – Peer-to-Peer Systems
  – Multi-DBMS