Chapter 2 – Database System Concepts and Architecture

Data Models, Schemas, and Instances

A **data model** --- a collection of concepts that can be used to describe the conceptual/logical structure of a database

Categories of Data Models (based on degree of abstractness):

- **High-Level/Conceptual**: (e.g., ER model of Chapter 3) provides a view close to the way users would perceive data; uses concepts such as
    - **entity**: real-world object or concept (e.g., student, employee, course, department, event)
    - **attribute**: some property of interest describing an entity (e.g., height, age, color)
    - **relationship**: an interaction among entities (e.g., works-on relationship between an employee and a project)
- **Representational/Implementational**: intermediate level of abstractness; example is relational data model (or the network model alluded to earlier). Also called **record-based** model.
- **Low-Level/Physical**: gives details as to how data is stored in computer system, such as record formats, orderings of records, access paths (indexes).

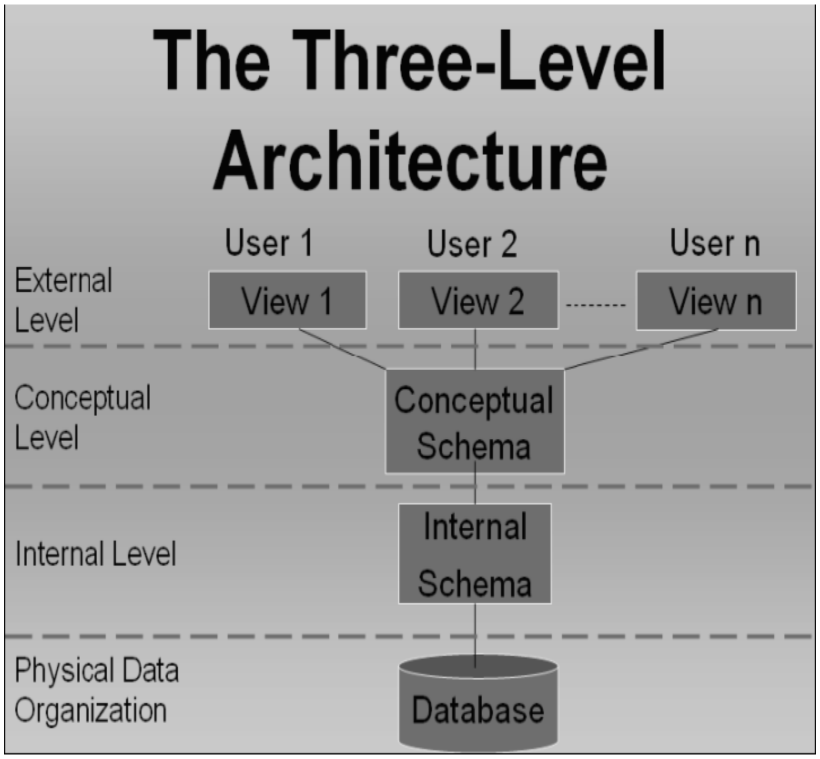
Schemas, Instances, and Database State

The *description* of a database is called the **database schema**, which is specified during database design and is not expected to change frequently.

The actual data stored in the database probably changes often. The data in the database at a particular time is called the **state of the database**, or a **snapshot**.

DBMS Architecture and Data Independence

Three-Schema Architecture: This idea was first described by the ANSI/SPARC committee in late 1970's. The goal is to separate (i.e., insert layers of "insulation" between) user applications and the physical database.



- **Internal/physical schema:** describes the physical storage structure (using a low-level data model)
- **Conceptual schema:** describes the (logical) structure of the whole database for a community of users. Hides physical storage details, concentrating upon describing entities, data types, relationships, user operations, and constraints. Can be described using either high-level or implementational data model.
- **External schema (or user views):** Each such schema describes part of the database that a particular category of users is interested in, hiding rest of database. Can be described using either high-level or implementational data model.

Users (including application programs) submit queries that are expressed with respect to the external level. It is the responsibility of the DBMS to **transform** such a query into one that is expressed with respect to the internal level (and to transform the result, which is at the internal level, into its equivalent at the external level).

Example: Select students with GPA > 3.5.

Q: How is this accomplished?
A: By virtue of **mappings** between the levels:

- **external/conceptual** mapping (providing **logical** data independence)
- **conceptual/internal** mapping (providing **physical** data independence)

Data independence

Data independence is the capacity to change the schema at one level of the architecture without having to change the schema at the next higher level.

Logical Data Independence:

The capacity to change the conceptual schema without having to change the external schemas and their associated application programs.

Physical Data Independence:

The capacity to change the internal schema without having to change the conceptual schema.

For an example of physical data independence, suppose that the internal schema is modified (because we decide to add a new index, or change the encoding scheme used in representing some field's value, or stipulate that some previously unordered file must be ordered by a particular field). Then we can change the mapping between the conceptual and internal schemas in order to avoid changing the conceptual schema itself.

Database Languages and Interfaces

DBMS Languages

DDL: Data definition language (conceptual schema, possibly internal/external))

- Used by the DBA and database designers to specify the conceptual schema of a database.
- In many DBMSs, the DDL is also used to define internal and external schemas (views).
- In some DBMSs, separate **storage definition language (SDL)** and **view definition language (VDL)** are used to define internal and external schemas.

SDL: Storage definition language (internal schema)

- SDL is typically realized via DBMS commands provided to the DBA and database designers

VDL: View definition language (external schema)

DML: Data manipulation language (retrieval, update)

- High-Level or Non-procedural Languages: These include the relational language SQL

        - May be used in a standalone way or may be embedded in a programming language
    - Low Level or Procedural Languages:
        - These must be embedded in a programming language

DBMS Interfaces

Menu-based: popular for browsing on the web

Forms-based: designed for naïve users

GUI-based: (Point and Click, Drag and Drop, etc.)

Natural language: requests in written English

Special purpose for parametric users: Bank clerks uses interfaces which require minimum key presses.

Interface for DBA: Facilitate to

- Creating user accounts, granting authorizations
- Setting system parameters
- Changing schemas or access paths

The Database System Environment

DBMS Component Modules

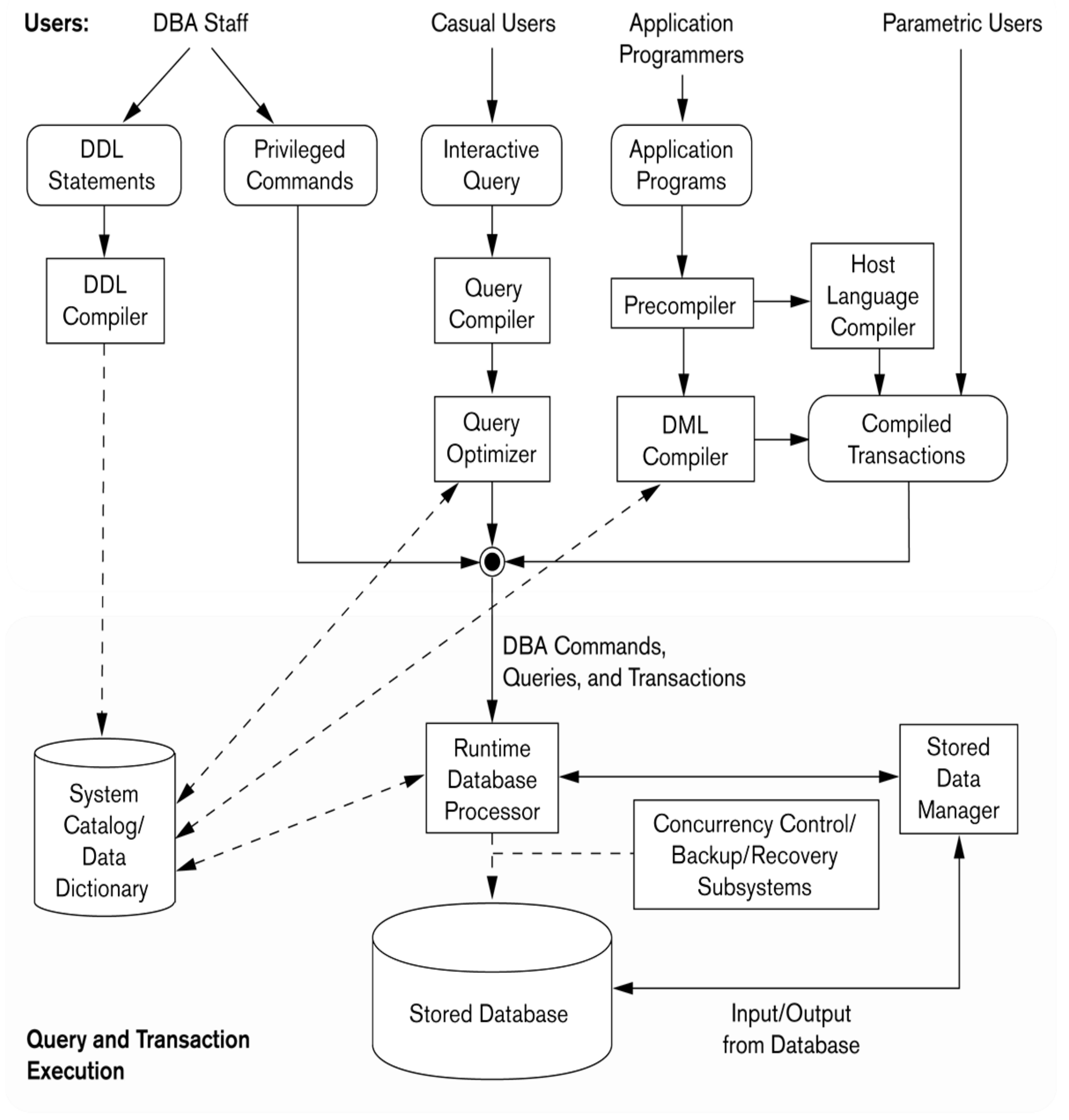


Figure 2.3
Component modules of a DBMS and their interactions.

Stored data manager: this module of DBMS controls access to DBMS information that is stored on disk, whether it is part of the database or the catalog. The dotted lines illustrate the accesses.

DDL compiler: it processes schema definitions, specified in the DDL, and stores description of schemas in the DBMS catalog.

DBMS catalog: includes information such as the names and data types of data item, storage details of each file, mapping information among schemas, and constraints. These are accessed by various modules of DBMS as shown by dotted lines.

Run time database processor: handles database accesses at runtime; it receives retrieval or update operations and carries them out on the database.

Query compiler: handles high level queries that are entered interactively. It parses, analyzes and compiles or interprets a query by creating database access code and then generates calls to the runtime database processor.

Pre compiler: Extracts DML commands from an application program written in a host language. These are sent to DML compiler for further processing.

DBMS Utilities

Most DBMSs have database utilities that help the DBA in managing the database system. Common utilities have the following functions:

- **Loading:** A loading utility is used to load existing data files into the database. Usually, the current format of the data file and the desired (target) database file structure are specified to the utility, which then automatically reformats the data and stores it in the database.
- **Backup:** A backup utility creates a backup copy of the database. This backup copy can be used to restore the database in case of catastrophic failure.
- **File reorganization:** This utility can be used to reorganize a database file into a different file organization to improve performance.(For example, we decide to add a new index, or change the encoding scheme used in representing some field's value, or stipulate that some previously unordered file must be ordered by a particular field to improve performance)
- **Performance Monitoring:** Such a utility monitors database usage and provides statistics to the DBA. The DBA uses the statistics in making decisions such as whether or not to reorganize files to improve performance.

Tools, Application Environments, and communications Facilities

The following other tools are often available to database designers, users, and DBAs

- CASE tools are used in the design phase of database systems.
- In addition to storing catalog information, the Data Dictionary stores other information such as design decisions, usage standards, application program descriptions, and user information. This information can be directly accessed by users or the DBA when needed.
- Application development environments such as Power Builder system provide an environment for developing database applications

Many commercial database systems have communication packages that work with the DBMS.

Classification of DBMS's

Normally the following criteria are used to classify **DBMS's.**

Based upon

- underlying data model (e.g., relational, object, object-relational, network)
- multi-user vs. single-user
- centralized vs. distributed
- cost
- general-purpose vs. special-purpose
- types of **access path** options

Centralized and Client/Server Architectures for DBMSs

Centralized Architecture

- Centralized architectures used mainframe computers to provide the main processing for all system functions, including user application programs and user interface programs, as well as all the DBMS functionality. So, all processing was performed remotely on the computer system, and only display information and controls were sent from the computer to the display terminals, which were connected to the central computer via various types of communications networks.

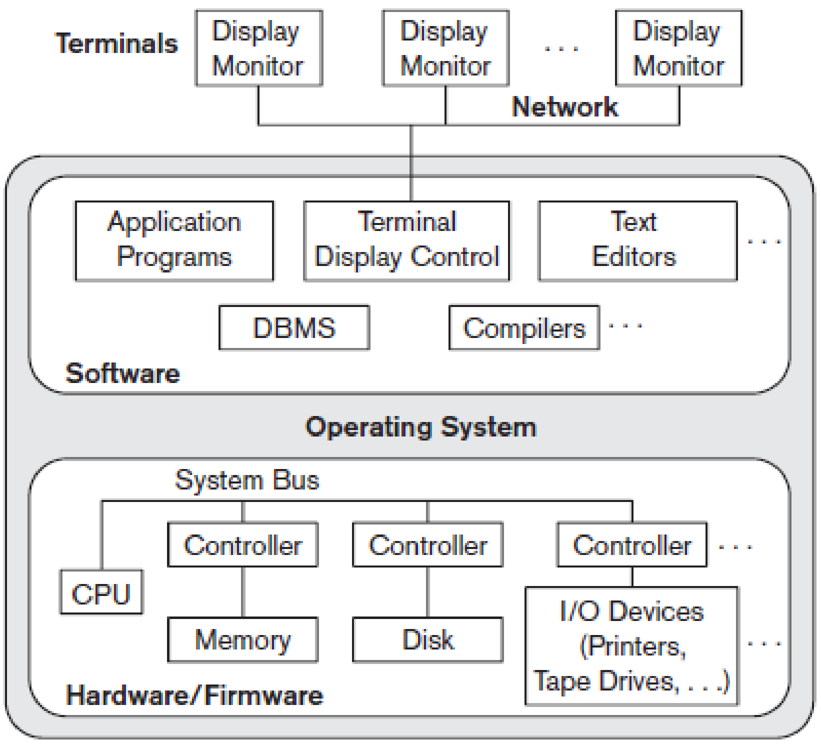


Figure 2.4
A physical centralized architecture.

Two-Tier Client/Server Architectures for DBMSs

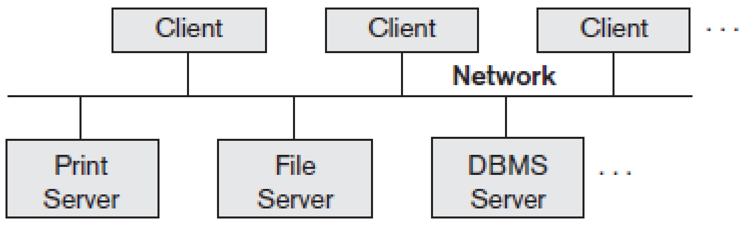


Figure 2.5
Logical two-tier client/server architecture.

- In one model of client server architecture, the user interface programs and application programs were run on the client side and the query and transaction functionality related to SQL processing remained on the server side.
- When DBMS access is required, the program establishes a connection to the DBMS (which is on the server side); once the connection is created, the client program can communicate with the DBMS. A standard called Open Database Connectivity (ODBC) provides an application programming interface (API), which allows client-side programs to call the DBMS, as long as both client and server machines have the necessary software installed.
- A client program can actually connect to several RDBMSs and send query and transaction requests using the ODBC API, which are then processed at the server sites. Any query results are sent back to the client program, which can process and display the results as needed.

- The different approach to two-tier client/server architecture was taken by some object- oriented DBMSs, where the software modules of the DBMS were divided between client and server in a more integrated way.
- The advantages of this architecture are its simplicity and seamless compatibility with existing systems. The emergence of the Web changed the roles of clients and servers, leading to the three-tier architecture.

Three-Tier Architectures for Web Applications

- Many Web applications use an architecture called the three-tier architecture, which adds an intermediate layer between the client and the database server, as illustrated in Figure 2.7(a).

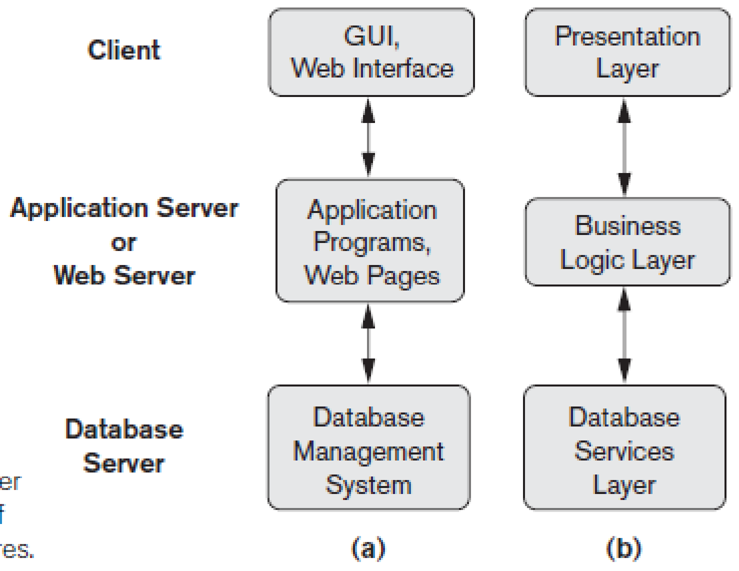


Figure 2.7
Logical three-tier client/server architecture, with a couple of commonly used nomenclatures.

- This intermediate layer or middle tier is called the application server or the Web server, depending on the application. This server plays an intermediary role by running application programs and storing business rules (procedures or constraints) that are used to access data from the database server. It can also improve database security by checking a client's credentials before forwarding a request to the database server.
- Clients contain GUI interfaces and some additional application-specific business rules. The intermediate server accepts requests from the client, processes the request and sends database queries and commands to the database server, and then acts as a conduit for passing (partially) processed data from the database server to the clients, where it may be processed further and filtered to be presented to users in GUI format. Thus, the user interface, application rules, and data access act as the three tiers.
- Figure 2.7(b) shows another architecture used by database and other application package vendors. The presentation layer displays information to the user and allows data entry. The business logic layer handles intermediate rules and constraints before data is passed up to the user or down to the DBMS. The bottom layer includes all data management services. The middle layer can also act as a Web server, which retrieves query results from the database server and formats

them into dynamic Web pages that are viewed by the Web browser at the client side.