Database Systems
(M3020008)

Lecture 2

School of Computer Science and Engineering (SoCSE)

Digital University Kerala ,Thiruvananthapuram

Database System Concepts and Architecture

Introduction

- The architecture of DBMS packages has evolved from the early monolithic systems, where the whole DBMS software package was one tightly integrated system, to the modern DBMS packages that are modular in design, with a client/server system architecture.
- The recent growth in the amount of data requiring storage has led to database systems with distributed architectures comprised of thousands of computers that manage the data stores.
- This evolution mirrors the trends in computing, where large centralized mainframe computers are replaced by hundreds of distributed workstations and personal computers connected via communications networks to various types of server machines—Web servers, database servers, file servers, application servers, and so on.
- The current **cloud computing** environments consist of thousands of large servers managing so-called **big data** for users on the Web.

Client-Server DBMS Architecture

- In a basic client/server DBMS architecture, the system functionality is distributed between two types of modules.
- A **client module** is typically designed so that it will run on a mobile device, user workstation, or personal computer (PC).
- Typically, application programs and user interfaces that access the database run in the client module.
- Hence, the client module handles user interaction and provides the user-friendly interfaces such as apps for mobile devices, or forms- or menu based GUIs (graphical user interfaces) for PCs.
- The other kind of module, called a **server module**, typically handles data storage, access, search, and other functions.
- We discuss client/server architectures in more detail later. First,
- We must study more basic concepts that will give us a better understanding of modern database architectures.

Data Models, Schemas, and Instances

- One fundamental characteristic of the database approach is that it provides some level of data abstraction.
- **Data abstraction** generally refers to the suppression of details of data organization and storage, and the highlighting of the essential features for an improved understanding of data.
- One of the main characteristics of the database approach is to support data abstraction so that different users can perceive data at their preferred level of detail.
- A **data model**—a collection of concepts that can be used to describe the structure of a database—provides the necessary means to achieve this abstraction.
- By *structure of a database* we mean the data types, relationships , and constraints that apply to the data.
- Most data models also include a set of **basic operations** for specifying retrievals and updates on the database.

Contd…

- In addition to the basic operations provided by the data model, it is becoming more common to include concepts in the data model to specify the **dynamic aspect** or **behavior** of a database application.
- This allows the database designer to specify a set of valid user-defined operations that are allowed on the database objects.
- An example of a user-defined operation could be COMPUTE_GPA, which can be applied to a STUDENT object.
- On the other hand, generic operations to insert, delete, modify, or retrieve any kind of object are often included in the *basic data model operations*.
- Concepts to specify behavior are fundamental to object-oriented data models but are also being incorporated in more traditional data models.
- For example, object-relational models extend the basic relational model to include such concepts, among others.
- In the basic relational data model, there is a provision to attach behavior to the relations in the form of persistent stored modules, popularly known as stored procedures.

Categories of Data Models

- Many data models have been proposed, which we can categorize according to the types of concepts they use to describe the database structure.
- **High-level** or **conceptual data models** provide concepts that are close to the way many users perceive data, whereas
- **low-level** or **physical data models** provide concepts that describe the details of how data is stored on the computer storage media, typically magnetic disks.
- Concepts provided by physical data models are generally meant for computer specialists, not for end users.
- Between these two extremes is a class of **representational** (or **implementation**) **data models**, which provide concepts that may be easily understood by end users but that are not too far removed from the way data is organized in computer storage.
- Representational data models hide many details of data storage on disk but can be implemented on a computer system directly.

Conceptual Model

- Conceptual data models use concepts such as **entities**, **attributes**, and **relationships**.
- An **entity** represents a real-world object or concept, such as an employee or a project from the mini world that is described in the database.
- An **attribute** represents some property of interest that further describes an entity, such as the employee’s name or salary.
- A **relationship** among two or more entities represents an association among the entities, for example, a works-on relationship between an employee and a project.
- **Entity–Relationship model**—a popular high-level conceptual data model.

Contd..

- Representational or implementation data models are the models used most frequently in traditional commercial DBMSs.
- These include the widely used **relational data model**, as well as the so-called legacy data models—the **network** and **hierarchical models**—that have been widely used in the past.
- The SQL standard for relational databases will discuss later.
- Representational data models represent data by using record structures and hence are sometimes called **record-based data models**.

Object Data Model

- We can regard the **object data model** as an example of a new family of higher-level implementation data models that are closer to conceptual data models.
- A standard for object databases called the ODMG object model has been proposed by the Object Data Management Group (ODMG).
- We describe the general characteristics of object databases and the object model proposed standard.
- Object data models are also frequently utilized as high-level conceptual models, particularly in the software engineering domain.

Physical Data Models

- Physical data models describe how data is stored as files in the computer by representing information such as record formats, record orderings, and access paths.
- An **access path** is a search structure that makes the search for particular database records efficient, such as indexing or hashing.
- An **index** is an example of an access path that allows direct access to data using an index term or a keyword.
- It is similar to the index at the end of this text, except that it may be organized in a linear, hierarchical (tree-structured), or some other fashion.

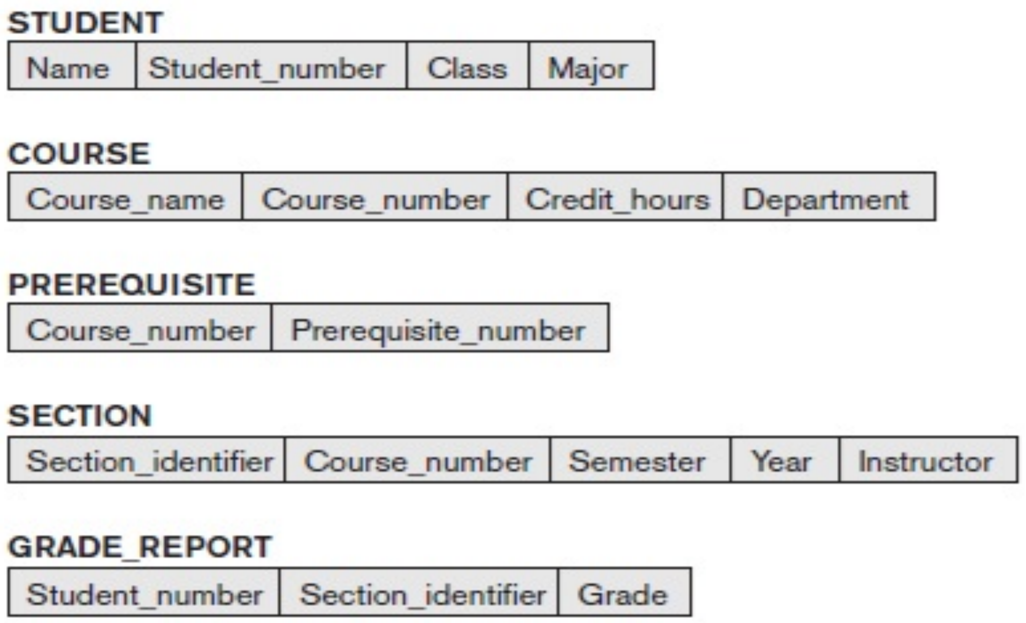
Self-Describing Data Models

- Another class of data models is known as **self-describing data models**.
- The data storage in systems based on these models combines the description of the data with the data values themselves.
- In traditional DBMSs, the description (schema) is separated from the data.
- These models include **XML** as well as many of the **key-value stores** and **NOSQL systems** that were recently created for managing big data.

Schemas, Instances, and Database State

- In a data model, it is important to distinguish between the *description* of the database and the *database itself*.
- The description of a database is called the **database schema**, which is specified during database design and is not expected to change frequently.
- Most data models have certain conventions for displaying schemas as diagrams.
- A displayed schema is called a **schema diagram**.
- Figure 1 shows a schema diagram for the database shown in Figure 2 in previous lecture;
- The diagram displays the structure of each record type but not the actual instances of records.

Schema diagram for the database



STUDENT

Name	Student_number	Class	Major

COURSE

Course_name	Course_number	Credit_hours	Department

PREREQUISITE

Course_number	Prerequisite_number

SECTION

Section_identifier	Course_number	Semester	Year	Instructor

GRADE_REPORT

Student_number	Section_identifier	Grade

Figure 1

Schema Construct

- We call each object in the schema—such as STUDENT or COURSE—a **schema construct**.
- A schema diagram displays only *some aspects* of a schema, such as the names of record types and data items, and some types of constraints.
- Other aspects are not specified in the schema diagram;
- for example, Figure 1 shows neither the data type of each data item nor the relationships among the various files.
- Many types of constraints are not represented in schema diagrams.
- A constraint such as *students majoring in computer science must take CS1310 before the end of their sophomore year* is quite difficult to represent diagrammatically.

Database State/Snapshot

- The actual data in a database may change quite frequently.
- For example, the database shown in Figure 2 in lecture 1 changes every time we add a new student or enter a new grade.
- The data in the database at a particular moment in time is called a **database state** or **snapshot**.
- It is also called the *current* set of **occurrences** or **instances** in the database.
- In a given database state, each schema construct has its own *current set* of instances; for example, the STUDENT construct will contain the set of individual student entities (records) as its instances.
- Many database states can be constructed to correspond to a particular database schema.
- Every time we insert or delete a record or change the value of a data item in a record, we change one state of the database into another state.

Database Schema/Database State

- The distinction between database schema and database state is very important.
- When we **define** a new database, we specify its database schema only to the DBMS.
- At this point, the corresponding database state is the ***empty state*** with no data.
- We get the ***initial state*** of the database when the database is first **populated** or **loaded** with the initial data.
- From then on, every time an update operation is applied to the database, we get another database state.
- At any point in time, the database has a ***current state***.
- The DBMS is partly responsible for ensuring that every state of the database is a **valid state**—that is, a state that satisfies the structure and constraints specified in the schema.
- Hence, specifying a correct schema to the DBMS is extremely important and the schema must be designed with utmost care.
- The DBMS stores the descriptions of the schema constructs and constraints—also called the **meta-data**—in the DBMS catalog so that DBMS software can refer to the schema whenever it needs to.
- The schema is sometimes called the **intension**, and a database state is called an **extension** of the schema.

Schema Evolution

- Although, as mentioned earlier, the schema is not supposed to change frequently, it is not uncommon that changes occasionally need to be applied to the schema as the application requirements change.
- For example, we may decide that another data item needs to be stored for each record in a file,
- such as adding the Date_of_birth to the STUDENT schema in Figure 1.
- This is known as **schema evolution**.
- Most modern DBMSs include some operations for schema evolution that can be applied while the database is operational.

Three-Schema Architecture and Data Independence

- Three of the four important characteristics of the database approach, are

(1) Use of a catalog to store the database description (schema) so as to make it self-describing,

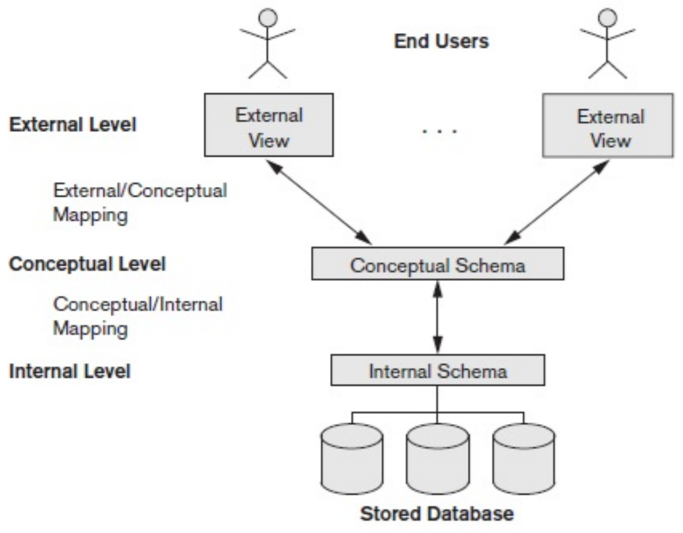
(2) Insulation of programs and data (program-data and program-operation independence), and

(3) Support of multiple user views.

- In this section we specify an architecture for database systems, called the **Three-Schema architecture**, that was proposed to help achieve and visualize these characteristics.
- Then we discuss further the concept of data independence.

The Three-Schema Architecture

- The goal of the three-schema architecture, illustrated in Figure 2, is to separate the user applications from the physical database.
- In this architecture, schemas can be defined at the following three levels:



Contd..

- The **internal level** has an **internal schema**, which describes the physical storage structure of the database.
- The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database.
- The **conceptual level** has a **conceptual schema**, which describes the structure of the whole database for a community of users.
- The conceptual schema hides the details of physical storage structures and concentrates on describing entities , data types, relationships, user operations, and constraints.
- Usually, a representational data model is used to describe the conceptual schema when a database system is implemented.
- This *implementation conceptual schema* is often based on a *conceptual schema design* in a high-level data model.

Contd…

- The **external** or **view level** includes a number of **external schemas** or **user views**.
- Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group.
- As in the previous level, each external schema is typically implemented using a representational data model, possibly based on an external schema design in a high-level conceptual data model.

Contd…

- The three-schema architecture is a convenient tool with which the user can visualize the schema levels in a database system.
- Most DBMSs do not separate the three levels completely and explicitly, but they support the three-schema architecture to some extent.
- Some older DBMSs may include physical-level details in the conceptual schema.
- The three-level ANSI architecture has an important place in database technology development because it clearly separates the users' external level, the

  database's conceptual level, and the internal storage level for designing a database.
- It is very much applicable in the design of DBMSs, even today.
- In most DBMSs that support user views, external schemas are specified in the same data model that describes the conceptual-level information (for example, a relational DBMS like Oracle or SQL Server uses SQL for this).

Contd…

- Notice that the three schemas are only *descriptions* of data; the actual data is stored at the physical level only.
- In the three-schema architecture, each user group refers to its own external schema.
- Hence, the DBMS must transform a request specified on an external schema into a request against the conceptual schema, and then into a request on the internal schema for processing over the stored database.
- If the request is a database retrieval, the data extracted from the stored database must be reformatted to match the user's external view.
- The processes of transforming requests and results between levels are called **mappings**.
- These mappings may be time-consuming, so some DBMSs—especially those that are meant to support small databases—do not support external views.
- Even in such systems, however, it is necessary to transform requests between the conceptual and internal levels.

Data Independence

- The three-schema architecture can be used to further explain the concept of **data independence**, which can be defined as the capacity to change the schema at one level of a database system without having to change the schema at the next higher level.
- We can define two types of data independence:

  1.Logical Data Independence

  2. Physical Data Independence

Logical data independence

- **Logical data independence** is the capacity to change the conceptual schema without having to change external schemas or application programs.
- We may change the conceptual schema to expand the database (by adding a record type or data item), to change constraints, or to reduce the database (by removing a record type or data item).
- In the last case, external schemas that refer only to the remaining data should not be affected.
- For example, the external schema of Figure 1.5(a) should not be affected by changing the GRADE_REPORT file (or record type) shown in Figure 1.2 into the one shown in Figure 1.6(a).
- Only the view definition and the mappings need to be changed in a DBMS that supports logical data independence.
- After the conceptual schema undergoes a logical reorganization, application programs that reference the external schema constructs must work as before.
- Changes to constraints can be applied to the conceptual schema without affecting the external schemas or application programs.

Physical data independence

- **Physical data independence** is the capacity to change the internal schema without having to change the conceptual schema.
- Hence, the external schemas need not be changed as well.
- Changes to the internal schema may be needed because some physical files were reorganized—for example, by creating additional access structures—to improve the performance of retrieval or update.
- If the same data as before remains in the database, we should not have to change the conceptual schema.
- For example, providing an access path to improve retrieval speed of SECTION records (Figure 1.2) by semester and year should not require a query such as *list all sections offered in fall 2008* to be changed, although the query would be executed more efficiently by the DBMS by utilizing the new access path.

Contd…

- Generally, physical data independence exists in most databases and file environments
- Where physical details, such as the exact location of data on disk, and hardware details of storage encoding, placement, compression, splitting, merging of records, and so on are hidden from the user.
- Applications remain unaware of these details.
- On the other hand, logical data independence is harder to achieve because it allows structural and constraint changes without affecting application programs—a much stricter requirement.

Contd…

- Whenever we have a multiple-level DBMS, its catalog must be expanded to include information on how to map requests and data among the various levels.
- The DBMS uses additional software to accomplish these mappings by referring to the mapping information in the catalog.
- Data independence occurs because when the schema is changed at some level, the schema at the next higher level remains unchanged;
- only the *mapping* between the two levels is changed.
- Hence, application programs referring to the higher-level schema need not be changed.

Database Languages and Interfaces

- We have discussed the variety of users supported by a DBMS.
- The DBMS must provide appropriate languages and interfaces for each category of users.
- In this section we discuss the types of languages and interfaces provided by a DBMS and the user categories targeted by each interface.

DBMS Languages - DDL

- Once the design of a database is completed and a DBMS is chosen to implement the database,
- The first step is to specify conceptual and internal schemas for the database and any mappings between the two.
- In many DBMSs where no strict separation of levels is maintained, one language, called the **Data Definition Language** (**DDL**), is used by the DBA and by database designers to define both schemas.
- The DBMS will have a DDL compiler whose function is to process DDL statements in order to identify descriptions of the schema constructs and to store the schema description in the DBMS catalog.

SDL

- In DBMSs where a clear separation is maintained between the conceptual and internal levels, the DDL is used to specify the conceptual schema only.
- Another language, the **Storage Definition Language** (**SDL**), is used to specify the internal schema.
- The mappings between the two schemas may be specified in either one of these languages.
- In most relational DBMSs today, there *is no specific language* that performs the role of SDL.
- Instead, the internal schema is specified by a combination of functions, parameters, and specifications related to storage of files.
- These permit the DBA staff to control indexing choices and mapping of data to storage.

VDL

- For a true three-schema architecture, we would need a third language, the **View Definition Language** (**VDL**), to specify user views and their mappings to the conceptual schema, but in most DBMSs *the DDL is used to define both conceptual and external schemas*.
- In relational DBMSs, SQL is used in the role of VDL to define user or application **Views** as results of predefined queries.

DML

- Once the database schemas are compiled and the database is populated with data, users must have some means to manipulate the database.
- Typical manipulations include retrieval, insertion, deletion, and modification of the data.
- The DBMS provides a set of operations or a language called the **Data Manipulation Language** (**DML**) for these purposes.

SQL

- In current DBMSs, the preceding types of languages are usually *not considered distinct languages*; rather, a comprehensive integrated language is used that includes constructs for conceptual schema definition, view definition, and data manipulation.
- Storage definition is typically kept separate, since it is used for defining physical storage structures to fine-tune the performance of the database system, which is usually done by the DBA staff.
- A typical example of a comprehensive database language is the **SQL Relational Database Language**, which represents a combination of DDL, VDL, and DML, as well as statements for constraint specification, schema evolution, and many other features.
- The SDL was a component in early versions of SQL but has been removed from the language to keep it at the conceptual and external levels only.

Types of DMLs

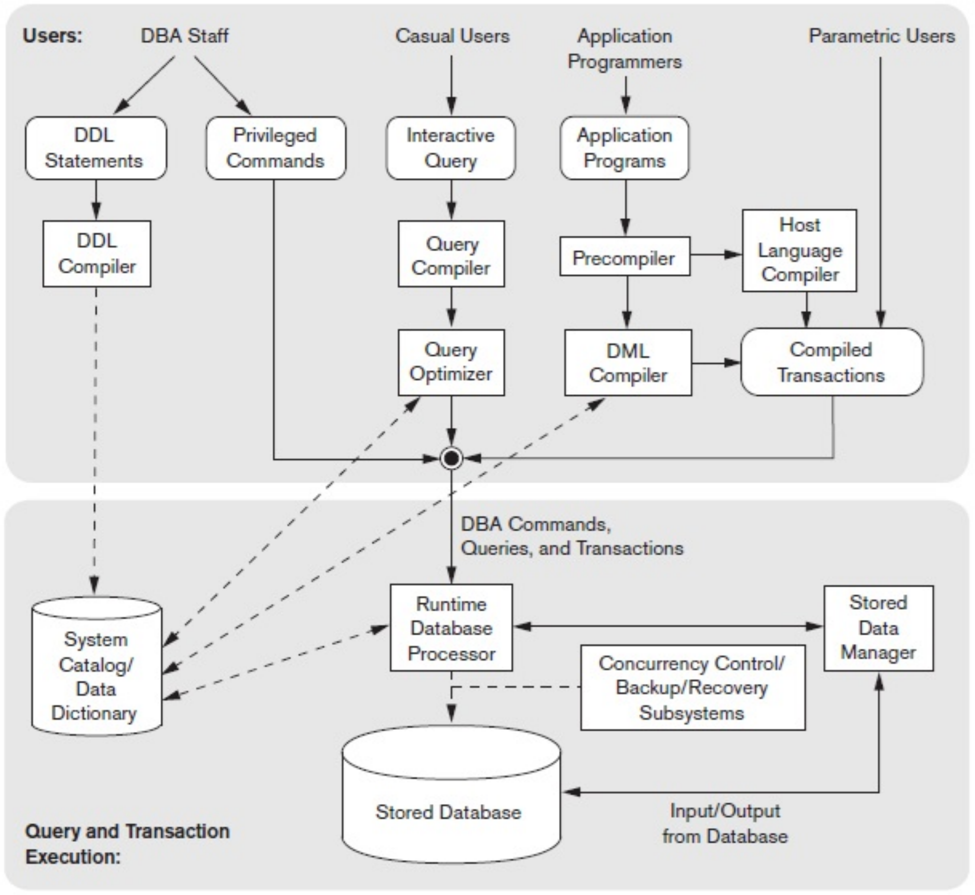
- There are two main types of DMLs.
- A **high-level** or **nonprocedural** DML can be used on its own to specify complex database operations concisely.
- Many DBMSs allow high-level DML statements either to be entered interactively from a display monitor or terminal or to be embedded in a general-purpose programming language.
- In the latter case, DML statements must be identified within the program so that they can be extracted by a pre compiler and processed by the DBMS.
- A **low level** or **procedural** DML *must* be embedded in a general-purpose programming language.
- This type of DML typically retrieves individual records or objects from the database and processes each separately.
- Therefore, it needs to use programming language constructs, such as looping, to retrieve and process each record from a set of records.
- Low-level DMLs are also called **record-at-a-time** DMLs because of this property. High-level DMLs, such as SQL, can specify and retrieve many records in a single DML statement; therefore, they are called **set-at-a-time** or **set-oriented** DMLs.
- A query in a high-level DML often specifies *which* data to retrieve rather than *how* to retrieve it; therefore, such languages are also called **declarative**.

Contd…

- Whenever DML commands, whether high level or low level, are embedded in a general-purpose programming language, that language is called the **host language** and the DML is called the **data sublanguage**.
- On the other hand, a high-level DML used in a standalone interactive manner is called a **query language**.
- In general, both retrieval and update commands of a high-level DML may be used interactively and are hence considered part of the query language.
- Casual end users typically use a high-level query language to specify their requests, whereas programmers use the DML in its embedded form.
- For naive and parametric users, there usually are **user-friendly interfaces** for interacting with the database; these can also be used by casual users or others who do not want to learn the details of a high-level query language.

The Database System Environment

- A DBMS is a complex software system.
- In this section we discuss the types of software components that constitute a DBMS and the types of computer system software with which the DBMS interacts.



DBMS Component Modules

- Figure illustrates, in a simplified form, the typical DBMS components.
- The figure is divided into two parts.
- The top part of the figure refers to the various users of the database environment and their interfaces.
- The lower part shows the internal modules of the DBMS responsible for storage of data and processing of transactions.

Contd…

- The database and the DBMS catalog are usually stored on disk.
- Access to the disk is controlled primarily by the **Operating System** (**OS**), which schedules disk read/write.
- Many DBMSs have their own **buffer management** module to schedule disk read/write, because management of buffer storage has a considerable effect on performance.
- Reducing disk read/write improves performance considerably.
- A higher-level **stored data manager** module of the DBMS controls access to DBMS information that is stored on disk, whether it is part of the database or the catalog.

Contd…

- Let us consider the top part of Figure first.
- It shows **interfaces** for the **DBA staff**, **casual users** who work with interactive interfaces to formulate queries, **application programmers** who create programs using some host programming languages, and **parametric users** who do data entry work by supplying parameters to predefined transactions.
- The DBA staff works on defining the database and tuning it by making changes to its definition using the DDL and other privileged commands.

Contd…

- The DDL compiler processes schema definitions, specified in the DDL, and stores descriptions of the schemas (meta-data) in the DBMS catalog.
- The catalog includes information such as the names and sizes of files, names and data types of data items, storage details of each file, mapping information among schemas, and constraints.
- In addition, the catalog stores many other types of information that are needed by the DBMS modules, which can then look up the catalog information as needed.

Contd…

- **Casual users** and persons with occasional need for information from the database interact using the **interactive query** interface in Figure.
- We have *not explicitly shown* any menu-based or form-based or mobile interactions that are typically used to generate the interactive query automatically or to access canned transactions.
- These queries are parsed and validated for correctness of the query syntax, the names of files and data elements, and so on by a **query compiler** that compiles them into an internal form.
- This internal query is subjected to query optimization.

Query Optimizer

- Among other things, the **query optimizer** is concerned with the rearrangement and possible reordering of operations, elimination of redundancies, and use of efficient search algorithms during execution.
- It consults the system catalog for statistical and other physical information about the stored data and generates executable code that performs the necessary operations for the query and makes calls on the runtime processor.

Pre-compiler

- Application programmers write programs in host languages such as Java, C, or C++ that are submitted to a pre-compiler.
- The **Pre-compiler** extracts DML commands from an application program written in a host programming language.
- These commands are sent to the **DML compiler** for compilation into object code for database access.
- The rest of the program is sent to the host language compiler.
- The object codes for the DML commands and the rest of the program are linked, forming a **canned transaction** whose executable code includes calls to the runtime database processor.
- It is also becoming increasingly common to use scripting languages such as PHP and Python to write database programs.
- Canned transactions are executed repeatedly by parametric users via PCs or mobile apps; these users simply supply the parameters to the transactions.
- Each execution is considered to be a separate transaction.
- An example is a bank payment transaction where the account number, payee, and amount may be supplied as parameters.

Contd…

- In the lower part of Figure, the **runtime database processor** executes (1) the privileged commands, (2) the executable query plans, and (3) the canned transactions with runtime parameters.
- It works with the **system catalog** and may update it with statistics.
- It also works with the **stored data manager**, which in turn uses basic operating system services for carrying out low-level input/output (read/write) operations between the disk and main memory.
- The runtime database processor handles other aspects of data transfer, such as management of buffers in the main memory.
- Some DBMSs have their own buffer management module whereas others depend on the OS for buffer management.
- We have shown **concurrency control** and **backup and recovery systems** separately as a module in this figure.
- They are integrated into the working of the runtime database processor for purposes of transaction management.

Contd…

- It is common to have the **client program** that accesses the DBMS running on a separate computer or device from the computer on which the database resides.
- The former is called the **client computer** running DBMS client software and the latter is called the **database server**.
- In many cases, the client accesses a middle computer, called the **application server**, which in turn accesses the database server.

Contd…

- Figure is not meant to describe a specific DBMS; rather, it illustrates typical DBMS modules.
- The DBMS interacts with the operating system when disk accesses— to the database or to the catalog—are needed.
- If the computer system is shared by many users, the OS will schedule DBMS disk access requests and DBMS processing along with other processes.
- On the other hand, if the computer system is mainly dedicated to running the database server, the DBMS will control main memory buffering of disk pages.
- The DBMS also interfaces with compilers for general purpose host programming languages, and with application servers and client programs running on separate machines through the system network interface.

Database System Utilities

- In addition to possessing the software modules just described, most DBMSs have **database utilities** that help the DBA manage the database system.
- Common utilities have the following types of functions:

Loading

- A **Loading utility** is used to load existing data files—such as text files or sequential files—into the database.
- Usually, the current (source) 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.
- With the proliferation of DBMSs, transferring data from one DBMS to another is becoming common in many organizations.
- Some vendors offer **Conversion tools** that generate the appropriate loading programs, given the existing source and target database storage descriptions (internal schemas).

Backup

- A **Backup utility** creates a backup copy of the database, usually by dumping the entire database onto tape or other mass storage medium.
- The backup copy can be used to restore the database in case of catastrophic disk failure.
- Incremental backups are also often used, where only changes since the previous backup are recorded.
- Incremental backup is more complex, but saves storage space.

Contd…

- **Database storage reorganization.**
  - This utility can be used to reorganize a set of database files into different file organizations and create new access paths 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 or whether to add or drop indexes to improve performance.
  - Other utilities may be available for sorting files, handling data compression, monitoring access by users, interfacing with the network, and performing other functions.

Tools, Application Environments, and Communications Facilities

- Other tools are often available to database designers, users, and the DBMS.
- CASE tools are used in the design phase of database systems.
- Another tool that can be quite useful in large organizations is an expanded **data dictionary** (or **data repository**) **system**.
- In addition to storing catalog information about schemas and constraints, the data dictionary stores other information, such as design decisions, usage standards, application program descriptions, and user information.
- Such a system is also called an **information repository**.
- This information can be accessed *directly* by users or the DBA when needed.
- A data dictionary utility is similar to the DBMS catalog, but it includes a wider variety of information and is accessed mainly by users rather than by the DBMS software.

Application development environments

- **Application development environments**, such as PowerBuilder (Sybase) or JBuilder (Borland), have been quite popular.
- These systems provide an environment for developing database applications and include facilities that help in many facets of database systems, including database design, GUI development, querying and updating, and application program development.

Communication Facilities

- The DBMS also needs to interface with **communications software**, whose function is to allow users at locations remote from the database system site to access the database through computer terminals, workstations, or personal computers.
- These are connected to the database site through data communications hardware such as Internet routers, phone lines, long-haul networks, local networks, or satellite communication devices.
- Many commercial database systems have communication packages that work with the DBMS.
- The integrated DBMS and data communications system is called a **DB/DC** system.
- In addition, some distributed DBMSs are physically distributed over multiple machines.
- In this case, communications networks are needed to connect the machines.
- These are often **local area networks (LANs)**, but they can also be other types of networks.

Centralized and Client/Server Architectures for DBMSs

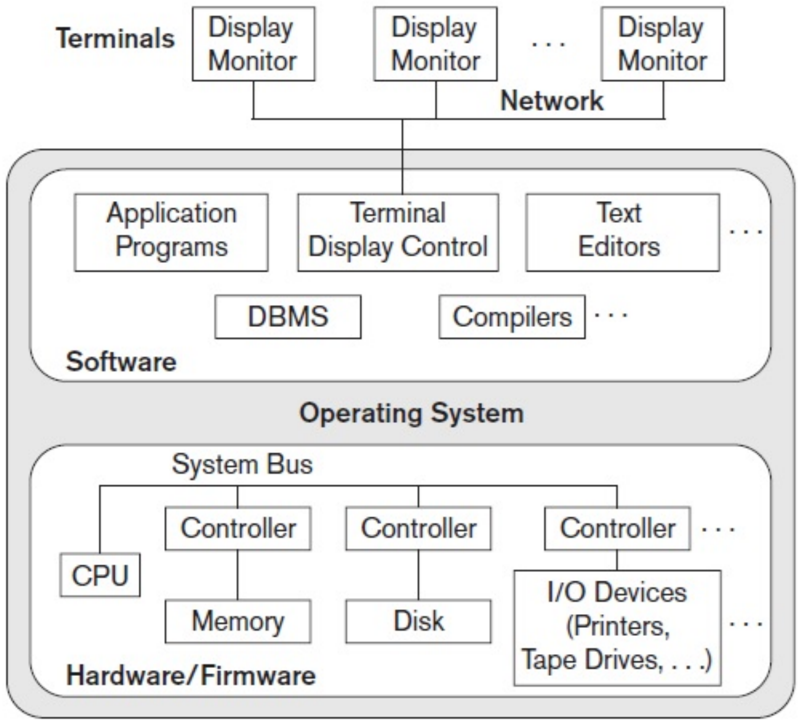
Centralized DBMSs Architecture

- Architectures for DBMSs have followed trends similar to those for general computer system architectures.
- Older 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.
- The reason was that in older systems, most users accessed the DBMS via computer terminals that did not have processing power and only provided display capabilities.
- Therefore, all processing was performed remotely on the computer system housing the DBMS, 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.

Contd…

- As prices of hardware declined, most users replaced their terminals with PCs and workstations, and more recently with mobile devices.
- At first, database systems used these computers similarly to how they had used display terminals,
- so that the DBMS itself was still a **centralized** DBMS in which all the DBMS functionality, application program execution, and user interface processing were carried out on one machine.
- Figure illustrates the physical components in a centralized architecture.
- Gradually, DBMS systems started to exploit the available processing power at the user side, which led to client/server DBMS architectures.

Physical Centralised Architecture

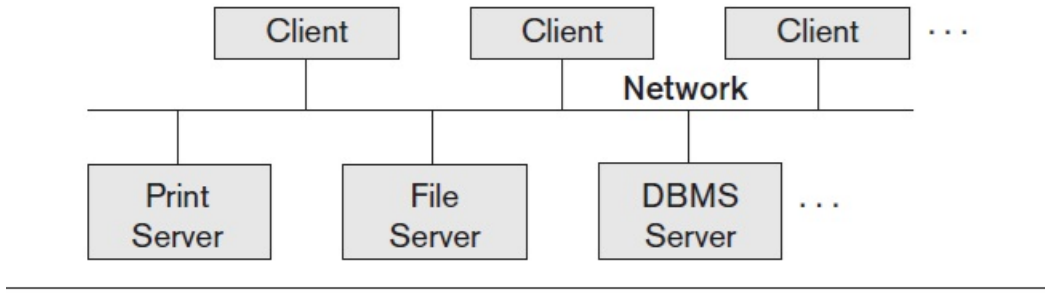


Basic Client/Server Architectures

- First, we discuss client/server architecture in general; then we discuss how it is applied to DBMSs.
- The **client/server architecture** was developed to deal with computing environments in which a large number of PCs, workstations, file servers, printers, database servers, Web servers, e-mail servers, and other software and equipment are connected via a network.
- The idea is to define **specialized servers** with specific functionalities.
- For example, it is possible to connect a number of PCs or small workstations as clients to a **file server** that maintains the files of the client machines.
- Another machine can be designated as a **printer server** by being connected to various printers; all print requests by the clients are forwarded to this machine.
- **Web servers** or **e-mail servers** also fall into the specialized server category.
- The resources provided by specialized servers can be accessed by many client machines.

Logical Level

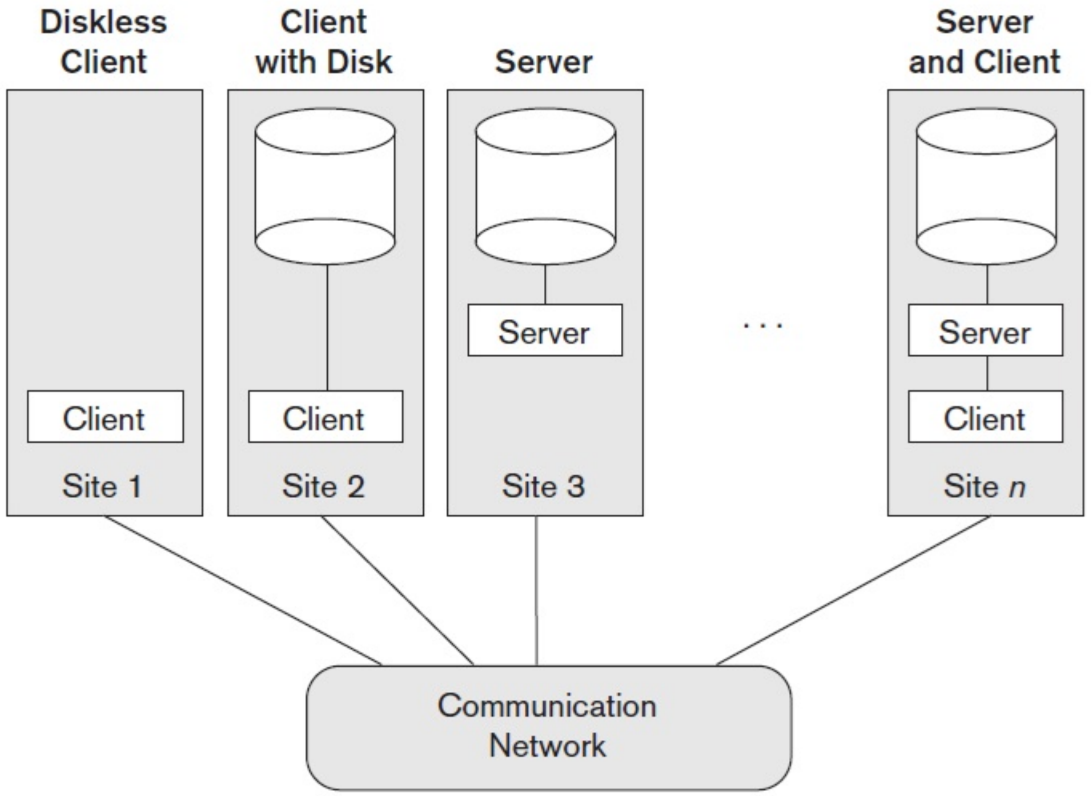
- The **client machines** provide the user with the appropriate interfaces to utilize these servers, as well as with local processing power to run local applications.
- This concept can be carried over to other software packages, with specialized programs— such as a CAD (computer-aided design) package—being stored on specific server machines and being made accessible to multiple clients.
- Figure illustrates client/server architecture at the logical level;



Physical Two-Tier Client/Server architecture

- Figure is a simplified diagram that shows the physical architecture.
- Some machines would be client sites only (for example, mobile devices or workstations/PCs that have only client software installed).
- Other machines would be dedicated servers, and others would have both client and server functionality.

Physical two-tier client/server architecture



Two-tier/Three-tier

- The concept of client/server architecture assumes an underlying framework that consists of many PCs/workstations and mobile devices as well as a smaller number of server machines, connected via wireless networks or LANs and other types of computer networks.
- A **client** in this framework is typically a user machine that provides user interface capabilities and local processing.
- When a client requires access to additional functionality—such as database access—that does not exist at the client, it connects to a server that provides the needed functionality.
- A **server** is a system containing both hardware and software that can provide services to the client machines, such as file access, printing, archiving, or database access.
- In general, some machines install only client software, others only server software, and still others may include both client and server software, as illustrated in Figure.
- However, it is more common that client and server software usually run on separate machines.
- Two main types of basic DBMS architectures were created on this underlying client/server framework: **two-tier** and **three-tier**.

Two-Tier Client/Server Architectures for DBMSs

- In relational database management systems (RDBMSs), many of which started as centralized systems, the system components that were first moved to the client side were the user interface and application programs.
- Because SQL provided a standard language for RDBMSs, this created a logical dividing point between client and server.
- Hence, the query and transaction functionality related to SQL processing remained on the server side. In such an architecture, the server is often called a **query server** or **transaction server** because it provides these two functionalities.
- In an RDBMS, the server is also often called an **SQL server**.

Contd…

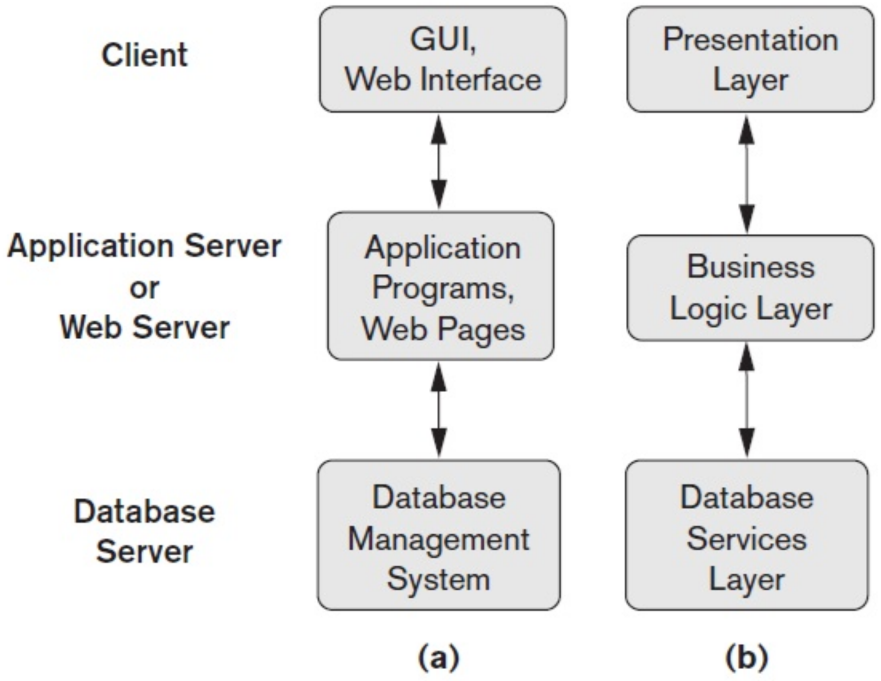
- The user interface programs and application programs can run on the client side.
- When DBMS access is required, the program establishes a connection to the DBMS (which is on the server side); once the connection is created, the client program can communicate with the DBMS.
- A standard called **Open 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.
- Most DBMS vendors provide ODBC drivers for their systems.
- 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.
- A related standard for the Java programming language, called **JDBC**, has also been defined.
- This allows Java client programs to access one or more DBMSs through a standard interface.

Contd…

- The architectures described here are called **two-tier architectures** because the software components are distributed over two systems: client and server.
- 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 and *n*-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 (a).



Contd…

- 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 user interfaces and Web browsers.
- 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 the users.
- Thus, the *user interface, application rules,* and *data access* act as the three tiers.

Contd…

- Figure (b) shows another view of the three-tier 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.
- The client machine is typically a PC or mobile device connected to the Web.

Contd…

- Other architectures have also been proposed.
- It is possible to divide the layers between the user and the stored data further into finer components, thereby giving rise to *n*-tier architectures, where *n* may be four or five tiers.
- Typically, the business logic layer is divided into multiple layers.
- Besides distributing programming and data throughout a network, *n*-tier applications afford the advantage that any one tier can run on an appropriate processor or operating system platform and can be handled independently.
- Vendors of ERP (enterprise resource planning) and CRM (customer relationship management) packages often use a *middleware layer,* which accounts for the front-end modules (clients) communicating with a number of back-end databases (servers).

Contd…

- Advances in encryption and decryption technology make it safer to transfer sensitive data from server to client in encrypted form, where it will be decrypted.
- The latter can be done by the hardware or by advanced software.
- This technology gives higher levels of data security, but the network security issues remain a major concern.
- Various technologies for data compression also help to transfer large amounts of data from servers to clients over wired and wireless networks.

Classification of Database Management Systems

Based on Data Model

- Several criteria can be used to classify DBMSs.
- The first is the **data model** on which the DBMS is based.
- The main data model used in many current commercial DBMSs is the **relational data model**, and the systems based on this model are known as **SQL systems.**
- The **object data model** has been implemented in some commercial systems but has not had widespread use.
- Recently, so-called **big data systems,** also known as **key-value storage systems** and **NOSQL systems,** use various data models: **document-based, graph-based, column-based,** and **key-value data models.**
- Many legacy applications still run on database systems based on the **hierarchical** and **network data models**.

Contd…

- The relational DBMSs are evolving continuously, and, in particular, have been incorporating many of the concepts that were developed in object databases.
- This has led to a new class of DBMSs called **Object-Relational DBMS**s.
- We can categorize DBMSs based on the data model: relational, object, object-relational, NOSQL, key-value, hierarchical, network, and other.
- Some experimental DBMSs are based on the XML (eXtended Markup Language) model, which is a **tree-structured data model.**
- These have been called **native XML DBMSs.**
- Several commercial relational DBMSs have added XML interfaces and storage to their products.

Based on Number of Users

- The second criterion used to classify DBMSs is the **number of users** supported by the system.
- **Single-user systems** support only one user at a time and are mostly used with PCs.
- **Multiuser systems**, which include the majority of DBMSs, support concurrent multiple users.

Based on Number of Sites

- The third criterion is the **number of sites** over which the database is distributed.
- A DBMS is **centralized** if the data is stored at a single computer site.
- A centralized DBMS can support multiple users, but the DBMS and the database reside totally at a single computer site.
- A **distributed** DBMS (DDBMS) can have the actual database and DBMS software distributed over many sites connected by a computer network.
- Big data systems are often massively distributed, with hundreds of sites.
- The data is often replicated on multiple sites so that failure of a site will not make some data unavailable.

Contd…

- **Homogeneous** DDBMSs use the same DBMS software at all the sites, whereas
- **Heterogeneous** DDBMSs can use different DBMS software at each site.
- It is also possible to develop **Middleware Software** to access several autonomous pre existing databases stored under heterogeneous DBMSs.
- This leads to a **Federated** DBMS (or **Multi Database System**), in which the participating DBMSs are loosely coupled and have a degree of local autonomy.
- Many DDBMSs use client-server architecture, as we described previously.

Based on Cost

- The fourth criterion is cost. It is difficult to propose a classification of DBMSs based on cost.
- Today we have open source (free) DBMS products like MySQL and PostgreSQL that are supported by third-party vendors with additional services.
- The main RDBMS products are available as free examination 30-day copy versions as well as personal versions, which may cost under $100 and allow a fair amount of functionality.
- The giant systems are being sold in modular form with components to handle distribution, replication, parallel processing, mobile capability, and so on, and with a large number of parameters that must be defined for the configuration.
- Furthermore, they are sold in the form of licenses—site licenses allow unlimited use of the database system with any number of copies running at the customer site.

Contd….

- Another type of license limits the number of concurrent users or the number of user seats at a location.
- Standalone single-user versions of some systems like Microsoft Access are sold per copy or included in the overall configuration of a desktop or laptop.
- In addition, data warehousing and mining features, as well as support for additional data types, are made available at extra cost.
- It is possible to pay millions of dollars for the installation and maintenance of large database systems annually.

Based on types of Access Path

- We can also classify a DBMS on the basis of the **types of access path** options for storing files.
- One well-known family of DBMSs is based on inverted file structures.
- Finally, a DBMS can be **general purpose** or **special purpose**.
- When performance is a primary consideration, a special-purpose DBMS can be designed and built for a specific application; such a system cannot be used for other applications without major changes.
- Many airline reservations and telephone directory systems developed in the past are special-purpose DBMSs.
- These fall into the category of **Online Transaction Processing** (**OLTP**) systems, which must support a large number of concurrent transactions without imposing excessive delays.