UNIT-1

SYLLABUS:

Introduction to Databases: Characteristics of the Database Approach, Advantages of using the DBMS Approach, A Brief History of Database Applications.

Overview of Database Languages and Architectures: Data Models, Schemas and Instances, Three-Schema Architecture and Data Independence, Database Languages and Interfaces, Database System environment, Centralized and Client-Server Architecture for DBMSs.

INTRODUCTION TO DATABASES:

Databases and database technology have a major impact on the growing use of computers. It is fair to say that databases play a critical role in almost all areas where computers are used, including business, electronic commerce, engineering, medicine, genetics, law, education, and library science.

DATABASE:

A database is a collection of related data. By data, we mean known facts that can be recorded and that have implicit meaning. For example, consider the names, telephone numbers, and addresses of the people you know.

A database has the following implicit properties:

- It represents some aspect of the real world, sometimes called the mini world or the universe of discourse (UoD). Changes to the mini world are reflected in the database.
- It is a logically coherent collection of data, to which some meaning can be attached.
- It is designed, built, and populated with data for a specific purpose. It has an intended group of users and some preconceived applications in which these users are interested.

To summarize: a database has some source (i.e., the mini world) from which data are derived, some degree of interaction with events in the represented mini world and an audience that is interested in using it.

Size/Complexity: A database can be of any size and complexity. For example, the list of names and addresses referred to earlier may consist of only a few hundred records, each with a simple structure. An example of a large commercial database is Amazon.com. It contains data for over 20 million books, CDs, videos, DVDs, games, electronics, apparel, and other items.

Computerized vs. manual: A database may be generated and maintained manually or it may be computerized. For example, simple database like telephone directory may be created and maintained manually. Huge and complex database may be created and maintained either by a group of application programs written specifically for that task or by a database management system.

Database Management System (DBMS)

A database management system (DBMS) is a collection of programs enabling users to create

and maintain a database. More specifically, the DBMS is a general-purpose software system that facilitates the processes of defining, constructing, manipulating, and sharing databases among various users and applications.

- Defining a database involves specifying the data types, structures, and constraints of the data to be stored in the database. The database definition or descriptive information is stored by the DBMS in the form of a database catalog or dictionary; it is called meta-data.

- Constructing the database is the process of storing the data on some storage medium that is controlled by the DBMS.
- Manipulating a database includes functions such as querying the database to retrieve specific data, updating the database to reflect changes in the mini world, and generating reports from the data.
- Sharing a database allows multiple users and programs to access the database simultaneously.

Other important functions provided by the DBMS include protecting the database and maintaining it over a long period of time.

- ✓ Protection includes system protection against hardware or software malfunction (or crashes) and security protection against unauthorized or malicious access.
- ✓ A typical large database may have a life cycle of many years, so the DBMS must be able to maintain the database system by allowing the system to evolve as requirements change over time.

A database together with the DBMS software is referred to as a database system.

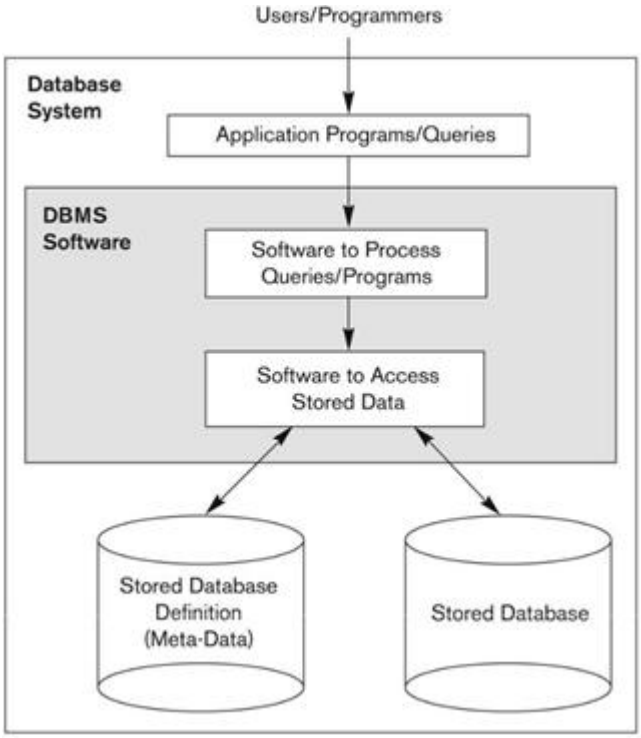


Fig: A simplified database system environment

Example:

Consider a UNIVERSITY database for maintaining information concerning students, courses, and grades in a university environment. The database is organized as five files, each of which stores data records of the same type.

1. STUDENT file: stores data on each student.
2. COURSE file: stores data on each course.
3. SECTION file: stores data on each section of a course.
4. GRADE_REPORT file: stores the grades that students receive in the various sections they have completed.
5. PREREQUISITE file: stores the prerequisites of each course.

STUDENT

Name	Student_number	Class	Major
Smith	17	1	CS
Brown	8	2	CS

COURSE

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	07	King
92	CS1310	Fall	07	Anderson
102	CS3320	Spring	08	Knuth
112	MATH2410	Fall	08	Chang
119	CS1310	Fall	08	Anderson
135	CS3380	Fall	08	Stone

GRADE_REPORT

Student_number	Section_identifier	Grade
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

PREREQUISITE

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

Fig: A database that stores student and course information

Defining a UNIVERSITY database

- Specify the structure of the records of each file - data elements to be stored in each record. For example: each STUDENT record includes data to represent the student's Name, Student_number, Class Major. Similarly each COURSE record includes data to represent the Course_name, Course_number, Credit_hours, and Department.

- Specify a data type for each data element within a record. For example: student's Name is a string of alphabetic characters Student_number is an integer.

Constructing the UNIVERSITY database

- To construct the UNIVERSITY database, we store data to represent each student, course, section, grade report, and prerequisite as a record in the appropriate file.
- Records in the various files may be related. For example, the record for Smith in the STUDENT file is related to two records in the GRADE_REPORT file that specify Smith's grades in two sections. Similarly, each record in the PREREQUISITE file relates two course records: one representing the course and the other representing the prerequisite.

Manipulating a UNIVERSITY database

Database manipulation involves querying and updating.

Examples of queries are as follows:

- ✓ Retrieve the transcript a list of all courses and grades of 'Smith'
- ✓ List the names of students who took the section of 'Database' course offered in fall 2008 and their grades in that section.
- ✓ List the prerequisites of Database course.

Examples of updates include the following:

- ✓ Change the class of 'Smith' to sophomore.
- ✓ Create a new section for the 'Database' course for this semester.
- ✓ Enter a grade of 'A' for 'Smith' in the 'Database' section of last semester.

These informal queries and updates must be specified precisely in the query language of the DBMS before they can be processed.

As with software in general, design of a new application for an existing database or design of a brand new database starts off with a phase called requirements specification and analysis. These requirements are documented in detail and transformed into a conceptual design that can be represented and manipulated using some computerized tools so that it can be easily maintained, modified, and transformed into a database implementation.

The design is then translated to a logical design that can be expressed in a data model implemented in a commercial DBMS. The final stage is physical design, during which further specifications are provided for storing and accessing the database. The database design is implemented, populated with actual data, and continuously maintained to reflect the state of the mini world.

Characteristics of Database Approach:

Database approach vs. File Processing approach

Consider an organization that is organized as a collection of departments/offices. Each department has certain data processing "needs", many of which are unique to it.

In the file processing approach, each department would control a collection of relevant data files and software applications to manipulate that data. For example, one user, the grade reporting office, may keep files on students and grades. Programs to print a student's transcript audio enter new grades are implemented as a part of the application. A second user, the accounting office, may keep track of student files and their payments. Although users are interested in data about students, each

students maintain separate files and programs to manipulate these files because each requires some data from the other user's files. This redundancy in defining and storing data results in wasted storage space and in redundant efforts to maintain common up-to-date data.

In the database approach, a single repository maintains data that is defined once and then accessed by various users. In file systems, each application is free to name data elements independently. In contrast, in a database, the names or labels of data are defined once, and used repeatedly by queries, transactions, and applications.

The main characteristics of the database approach versus the file-processing approach are the following:

1) Self-describing nature of a database system
2) Insulation between programs and data, and data abstraction
3) Support of multiple views of the data
4) Sharing of data and multiuser transaction processing

1) Self-describing nature of a database system:

A fundamental characteristic of the database approach is that the database system contains not only the database itself but also a complete definition or description of the database structure and constraints. This meta-data (i.e., data about data) is stored in the so-called system catalog, which contains a description of the structure of each file, the type and storage format of each field, and the various constraints on the data (i.e., conditions that the data must satisfy).

The system catalog is used not only by users but also by the DBMS software, which certainly needs to "know" how the data is structured/organized in order to interpret it in a manner consistent with that structure.

RELATIONS

Relation_name	No_of_columns
STUDENT	4
COURSE	4
SECTION	5
GRADE_REPORT	3
PREREQUISITE	2

COLUMNS

Column_name	Data_type	Belongs_to_relation
Name	Character (30)	STUDENT
Student_number	Character (4)	STUDENT
Class	Integer (1)	STUDENT
Major	Major_type	STUDENT
Course_name	Character (10)	COURSE
Course_number	XXXXNNNN	COURSE
....	....	.....
....	....	.....
....	....	.....
Prerequisite_number	XXXXNNNN	PREREQUISITE

Fig: An example of a database catalog for the database

2) Insulation between Programs and Data, and Data Abstraction:

Program-Data Independence: In traditional file processing, the structure of the data files accessed by an application is "hard-coded" in its source code. If, for some reason, we decide to change the structure of the data, every application in which a description of that file's structure is hard-coded must be changed!

In contrast, DBMS access programs, in most cases, do not require such changes, because the structure of the data is described separately from the programs that access it and those programs consult the catalog in order to ascertain the structure of the data so that they interpret that data properly.

In other words, the DBMS provides a conceptual or logical view of the data to application programs, so that the underlying implementation may be changed without the programs being modified. (This is referred to as program-data independence.

Program-operation independence: In object-oriented and object-relational systems, users can define operations on data as part of the database definitions. An operation (also called a function or method) is specified in two parts. The interface (or signature) of an operation includes the operation name and the data types of its arguments (or parameters). The implementation (or method) of the operation is specified separately and can be changed without affecting the interface. User application programs can operate on the data by invoking these operations through their names and arguments, regardless of how the operations are implemented. This may be termed program-operation independence.

Data abstraction: The characteristic that allows program-data independence and program-operation independence is called data abstraction. A DBMS provides users with a conceptual representation of data that does not include many of the details of how the data is stored or how the operations are implemented. Informally, a data model is a type of data abstraction that is used to provide this conceptual representation. The data model uses logical concepts, such as objects, their properties, and their inter relationships, that may be easier for most users to understand than computer storage concepts. Hence, the data model hides storage and implementation details that are not of interest to most database users.

3) Support of Multiple Views of the Data

A database typically has many users, each of whom may require a different perspective or view of the database. A view may be a subset of the database or it may contain virtual data that is derived from the database files but is not explicitly stored. A multiuser DBMS whose users have a variety of distinct applications must provide facilities for defining multiple views. For example, one user of the database of Figure 1.2 may be interested only in accessing and printing the transcript of each student; the view for this user is shown in Figure.

TRANSCRIPT

Student_name	Student_transcript				
	Course_number	Grade	Semester	Year	Section_id
Smith	CS1310	C	Fall	08	119
	MATH2410	B	Fall	08	112
Brown	MATH2410	A	Fall	07	85
	CS1310	A	Fall	07	92
	CS3320	B	Spring	08	102
	CS3380	A	Fall	08	135

Fig. View derived from the University database

4) Sharing of Data and Multiuser Transaction Processing:

A multiuser DBMS, as its name implies, must allow multiple users to access the database at the same time. This is essential if data for multiple applications is to be integrated and maintained in a single database. The DBMS must include concurrency control software to ensure that several users trying to update the same data do so in a controlled manner so that the result of the updates is correct. For example, when several reservation agents try to assign a seat on an airline flight, the DBMS should ensure that each seat can be accessed by only one agent at a time for assignment to a passenger. These types of applications are generally called online transaction processing (OLTP) applications. A

fundamental role of multiuser DBMS software is to ensure that concurrent transactions operate correctly and efficiently.

The concept of a transaction has become central to many database applications. A transaction is an executing program or process that includes one or more database accesses, such as reading or updating of database records. The DBMS must enforce several transaction properties. The isolation property ensures that each transaction appears to execute in isolation from other transactions, even though hundreds of transactions may be executing concurrently. The atomicity property ensures that either all the database operations in a transaction are executed or none.

Database Users

Users may be divided into

- Those who actually use and control the database content, and those who design, develop and maintain database applications called **"Actors on the scene".**
- Those who design and develop the DBMS software and related tools, and the computer systems operators called **"Workers behind the scene".**

Actors on the Scene:

1) **Database Administrator (DBA):** chief administrator, who oversees and manages the database system (including the data and software). Duties include authorizing users to access the database, coordinating/monitoring its use, acquiring hardware/software for upgrades, etc. The DBA is accountable for problems such as security breaches and poor system response time. In large organizations, the DBA might have a support staff.
2) **Database Designers:** responsible for identifying the data to be stored and for choosing an appropriate way to organize it. Database designers typically interact with each potential group of users and develop views of the database that meet the data and processing requirements of these groups. The final database design must be capable of supporting the requirements of all user groups.
3) **End Users:** These are persons who access the database for querying, updating, and report generation. There are several categories of end users:
   - ✓ **Casual end users:** use database occasionally, needing different information each time; use query language to specify their requests; typically middle- or high-level managers.
   - ✓ **Naive/Parametric end users:** biggest group of users; frequently query/update the database using standard canned transactions that have been carefully programmed and tested in advance.
     Examples:
     - Bank tellers check account balances, post withdrawals/deposits
     - Reservation clerks for airlines, hotels, etc., check availability of seats/rooms and make reservations.
     - **Sophisticated end users:** Include engineer scientists business analysts and others, who thoroughly familiarize themselves with the facilities of the DBMS in order to implement their own applications to meet their complex requirements.
   - ✓ **Stand-alone users:** maintain personal databases by using ready-made program packages that provide easy-to-use menu-based or graphics-based interfaces.
     Example: user of a tax package that stores a variety of personal financial data for tax purposes
4) **System Analysts and Application Programmers (Software Engineers):**
   - **System Analysts:** determine needs of end users, especially naive and parametric users, and develop specifications for canned transactions that meet these needs.

- **Application Programmers:** Implement, test, document, and maintain programs that satisfy the specifications mentioned above.

Workers behind the Scene

1. **DBMS system designers and implementers**: design and implement the DBMS modules and interfaces as a software package. A DBMS is a very complex software system that consists of many components, or modules, including modules for implementing the catalog, query language processing, interface processing, accessing and buffering data, controlling concurrency, and handling data recovery and security.
2. **Tool developers:** design and implement tools that facilitate database modeling and design, database system design, and improved performance.
3. **Operators and maintenance personnel (system administration personnel):** responsible for the actual running and maintenance of the hardware and software environment for the database system.

Advantages of Using the DBMS Approach:

1) **Controlling Redundancy:** Data redundancy such as tends to occur in the "file processing" approach leads to wasted storage space, duplication of effort and a higher likelihood of the introduction of inconsistency.
   In the database approach, the views of different user groups are integrated during database design. This is known as data normalization, and it ensures consistency and saves storage space. However, it is sometimes necessary to use controlled redundancy to improve the performance of queries. For example, we may store Student_name and Course_number redundantly in a GRADE_REPORT file because whenever we retrieve a GRADE_REPORT record, we want to retrieve the student name and course number along with the grade, student number, and section identifier.
   A DBMS should provide the capability to automatically enforce the rule that no inconsistencies are introduced when data is updated.
2) **Restricting Unauthorized Access:** When multiple users share a large database, it is likely that most users will not be authorized to access all information in the database. For example, financial data is often considered confidential and only authorized persons are allowed to access such data. In addition, some users may only be permitted to retrieve data, whereas others are allowed to retrieve and update. Hence, the type of access operation retrieval or update must also be controlled. A DBMS should provide a security and authorization subsystem, which the DBA uses to create accounts, to specify account restrictions and enforce these restrictions automatically.
3) **Providing Persistent Storage for Program Objects:** The values of program variables or objects are discarded once a program terminates, unless the programmer explicitly stores them in permanent files, which often involves converting these complex structures into a format suitable for file storage. Object-oriented database systems make it easier for complex runtime objects to be saved in secondary storage so as to survive beyond program termination and to be retrievable at a later time.
   Object-oriented database systems are compatible with programming languages such as C++ and Java, and the DBMS software automatically performs any necessary conversions.
4) **Providing Storage Structures and Search Techniques for Efficient Query Processing:** DBMS maintains indexes that are utilized to improve the execution time of queries and updates. DBMS has a buffering or caching module that maintains parts of the database in main memory buffers. The query processing and optimization module is responsible for choosing an efficient query execution plan for each query submitted to the system.
5) **Providing Backup and Recovery**: The backup and recovery subsystem of the DBMS is responsible for recovery. For example, if the computer system fails in the middle of a

complex update transaction, the recovery subsystem is responsible for making sure that the database is restored to the state it was in before the transaction started executing. Disk backup is also necessary in case of a catastrophic disk failure.

6) **Providing Multiple User Interfaces:** Because many types of users with varying levels of technical knowledge use a database, a DBMS should provide a variety of user interfaces. These include
   - Query languages for casual users
   - Programming language interfaces for application programmers
   - Forms and command codes for parametric users
   - Menu-driven interfaces and natural language interfaces for standalone users
7) **Representing Complex Relationships among Data:** A database may include numerous varieties of data that are interrelated in many ways.
   For example each section record is related to one course record and to a number of GRADE_REPORT records one for each student who completed that section. A DBMS must have the capability to represent a variety of complex relationships among the data, to define new relationships as they arise, and to retrieve and update related data easily and efficiently.
8) **Enforcing Integrity Constraints**: Most database applications are such that the semantics of the data require that it satisfy certain restrictions in order to make sense.
   The simplest type of integrity constraint involves specifying a data type for each data item.
   For example, in student table we specified that the value of Name must be a string of no more than 30 alphabetic characters.
   More complex type of constraint is referential integrity involves specifying that a record in one file must be related to records in other files. For example, in university database, we can specify that every section record must be related to a course record.
   Another type of constraint specifies uniqueness on data item values, such as every course record must have a unique value for Course_number. This is known as a key or uniqueness constraint.
9) **Permitting Inferencing and Actions Using Rules:** In a deductive database system, one may specify declarative rules that allow the database to infer new data. For example, figure out which students are on academic probation. Such capabilities would take the place of application programs that would be used to ascertain such information otherwise.
   Active database systems go one step further by allowing "active rules" that can be used to initiate actions automatically.
   In today's relational database system it is possible to associate triggers with tables.
10) **Additional Implications of Using the Database Approach:**
    - **Potential for Enforcing Standards:** database approach permits the DBA to define and enforce standards among database users in a large organization which facilitates communication and cooperation among various departments, projects, and users within the organization.Standards can be defined for names and formats of data elements, display formats, report structures and so on.
    - **Reduced Application Development Time**: once a database is up and running, substantially less time is generally required to create new applications using DBMS facilities. Development time using a DBMS is estimated to be one-sixth to one-fourth of that for a traditional file systems.
    - **Flexibility:** It may be necessary to change the structure of a database as requirements change. DBMSs allow changes to the structure of the database without affecting the stored data and the existing application programs.
    - **Availability of Up-to-Date Information:** DBMS makes the database available to all users. Availability of up-to-date information is essential for many transaction-processing applications, such as reservation systems or banking databases

- **Economies of Scale:** DBMS approach permits consolidation of data and applications, to overlap between activities of data-processing in different projects or departments. This enables the whole organization to invest in more powerful processors, storage devices, or communication gear, rather than having each department purchase its equipment thus reducing overall costs of operation and management.

A Brief History of Database Applications:

- **Early Database Applications Using Hierarchical and Network Systems**

  Early database applications maintained records in large organizations such as corporations, universities, hospitals, and banks. In many of these applications, there were large numbers of records of similar structure. There were also many types of records and many interrelationships among them.

  Problems with the early database systems

  - Lack of data abstraction and program-data independence capabilities
  - Provided only programming language interfaces. This made it time-consuming and expensive to implement new queries and transactions, since new programs had to be written, tested, and debugged.

- **Providing Data Abstraction and Application Flexibility with Relational Databases**

  Relational databases were originally proposed to separate the physical storage of data from its conceptual representation and to provide a mathematical foundation for data representation and querying. The relational data model also introduced high-level query languages that provided an alternative to programming language interfaces, making it much faster to write new queries. Hence, data abstraction and program-data independence were much improved when compared to earlier systems.

- **Object-Oriented Applications and the Need for More Complex Databases**

  Object-oriented databases (OODBs) mainly used in specialized applications, such as engineering design, multimedia publishing, and manufacturing systems. In addition, many object-oriented concepts were incorporated into the newer versions of relational DBMSs, leading to object-relational database management systems, known as ORDBMSs.

- **Interchanging Data on the Web for E-Commerce Using XML**

  The World Wide Web provides a large network of interconnected computers. Users can create documents using a Web publishing language, such as HyperText Markup Language (HTML), and store these documents on Web servers where other users (clients) can access them. Documents can be linked through hyperlinks, which are pointers to other documents

  Currently, eXtended Markup Language (XML) is considered to be the primary standard for interchanging data among various types of databases and Web pages. XML combines concepts from the models used in document systems with database modeling concepts.

- **Extending Database Capabilities for New Applications**

  The success of database systems in traditional applications encouraged developers of other types of applications to attempt to use them. The following are some examples of these applications:

- Scientific applications that store large amounts of data resulting from Scientific experiments in areas such as high-energy physics, the mapping of the human genome, and the discovery of protein structures.
- Storage and retrieval of images, including scanned news or personal photographs, satellite photographic images, and images from medical procedures such as x-rays and MRIs (magnetic resonance imaging).
- Storage and retrieval of videos, such as movies, and video clips from news or personal digital cameras.
- Data mining applications that analyze large amounts of data searching for the occurrences of specific patterns or relationships, and for identifying unusual patterns in areas such as credit card usage.
- Spatial applications that store spatial locations of data, such as weather information, maps used in geographical information systems, and in automobile navigational systems.
- Time series applications that store information such as economic data at regular points in time, such as daily sales and monthly gross national product figures.

- **Databases versus Information Retrieval**

  Database technology is heavily used in manufacturing, retail, banking, insurance, finance, and health care industries, where structured data is collected through forms, such as invoices or patient registration documents. An area related to database technology is Information Retrieval (IR), which deals with books, manuscripts, and various forms of library-based articles. Data is indexed, cataloged, and annotated using keywords. IR is concerned with searching for material based on these keywords, and with the many problems dealing with document processing and free-form text processing.

When Not to Use a DBMS:

- DBMS may involve unnecessary overhead costs that would not be incurred in traditional file processing. The overhead costs of using a DBMS are due to the following:
  - High initial investment in hardware, software, and training.
  - The generality that a DBMS provides for defining and processing data
  - Overhead for providing security, concurrency control, recovery, and integrity functions
- Therefore, it may be more desirable to use regular files under the following circumstances:
  - Simple, well-defined database applications that are not expected to change at all.
  - Stringent, real-time requirements for some application programs that may not be met because of DBMS overhead.
  - Embedded systems with limited storage capacity, where a general-purpose DBMS would not fit.
  - No multiple-user access to data.

Overview of Database Languages and Architectures:

Introduction

The architecture of DBMS packages has evolved from the early monolithic systems, where the whole DBMS software package was one tightly integrated system. Modern DBMS packages are modular in design, with a client/server system architecture. In a basic client/server DBMS architecture, the system functionality is distributed between two types of module. A client module is designed to run on a user workstation or personal computer. The client module handles user interaction and provides the user-friendly interfaces such as forms- or menu-based GUIs. The other kind of module, called a server module handles data storage, access, search, and other functions

Data Models, Schemas and Instances:

Data Model

A data model is a collection of concepts that can be used to describe the structure of a database. 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. Data model provides the necessary means to achieve abstraction.

Categories of Data Models

Data models can be categorized according to the types of concepts they use to describe the database structure.

1) **High-level or conceptual data models**: provide concepts that are close to the way many users perceive data. Conceptual data models use concepts such as entities, attributes, and relationships.
2) **Representational or implementation data models:** 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. 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. Representational data models represent data by using record structures and hence are sometimes called record-based data models.
3) **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. 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.

Database schema

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

Schema diagram

- A displayed schema is called a ***schema diagram.*** 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.

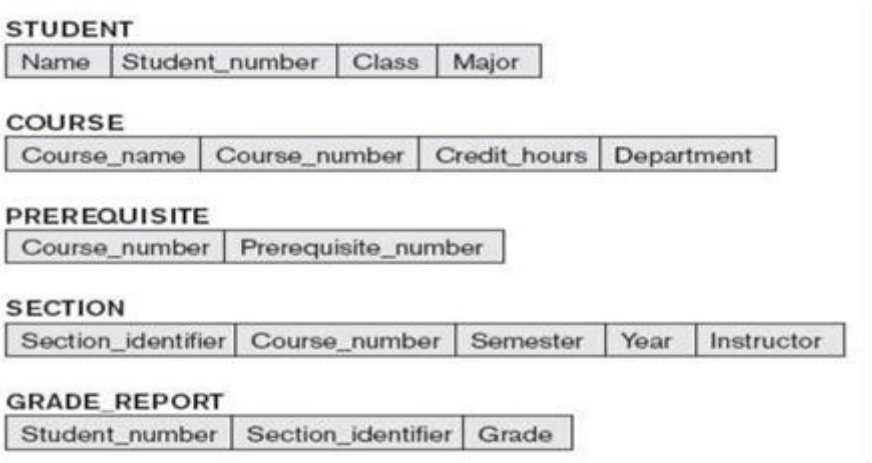


Figure: Schema diagram for the database

Schema construct

Each object in the schema is called schema construct. For example student or course.

Database state or snapshot

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.

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. 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.

Three-Schema Architecture and Data Independence:

The Three-Schema Architecture

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

1) **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.
2) **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.
3) **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. Each external schema is typically implemented using a representational data model, possibly based on an external schema design in a high-level data model.

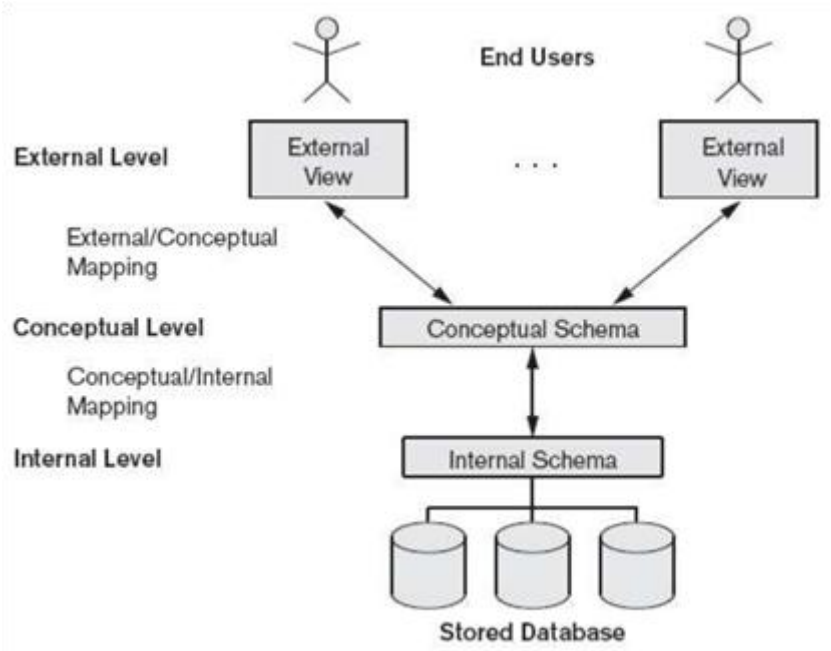


Figure 2.2: The three-schema architecture

In a DBMS based on 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.

Data Independence

Data independence 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** 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, to change constraints, or to reduce the database. Only the view definition and the mappings need to be changed in a DBMS that supports logical data independence.
2) **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.

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.

Database Languages and Interfaces:

The DBMS must provide appropriate languages and interfaces for each category of users.

DBMS Languages

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.

Data Definition Language (DDL)

The data definition language (DDL) is used by the DBA and by database designers to define both schemas when no strict separation of levels is maintained . 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.

Storage Definition Language (SDL)

Storage definition language is used when clear separation is maintained between the conceptual and internal levels, the DDL is used to specify the conceptual schema only.

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.

View Definition Language (VDL),

View definition language is used to specify user views and their mappings to the conceptual schema. In relational DBMSs, SQL is used in the role of VDL to define user or application views as results of predefined queries.

Data Manipulation Language (DML)

Data manipulation languages (DML) are used to perform manipulation operation such as retrieval, insertion, deletion, and modification of the data.

There are two main types of DMLs.:

1) **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 precompiler and processed by the DBMS. 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
2) **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. 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. DL/1, a DML designed for the hierarchical model, is a low-level DML that uses commands such as GET UNIQUE, GET NEXT, or GET NEXT WITHIN PARENT to navigate from record to record within a hierarchy of records in the database.

Host language

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.

A high-level DML used in a standalone interactive manner is called a query language

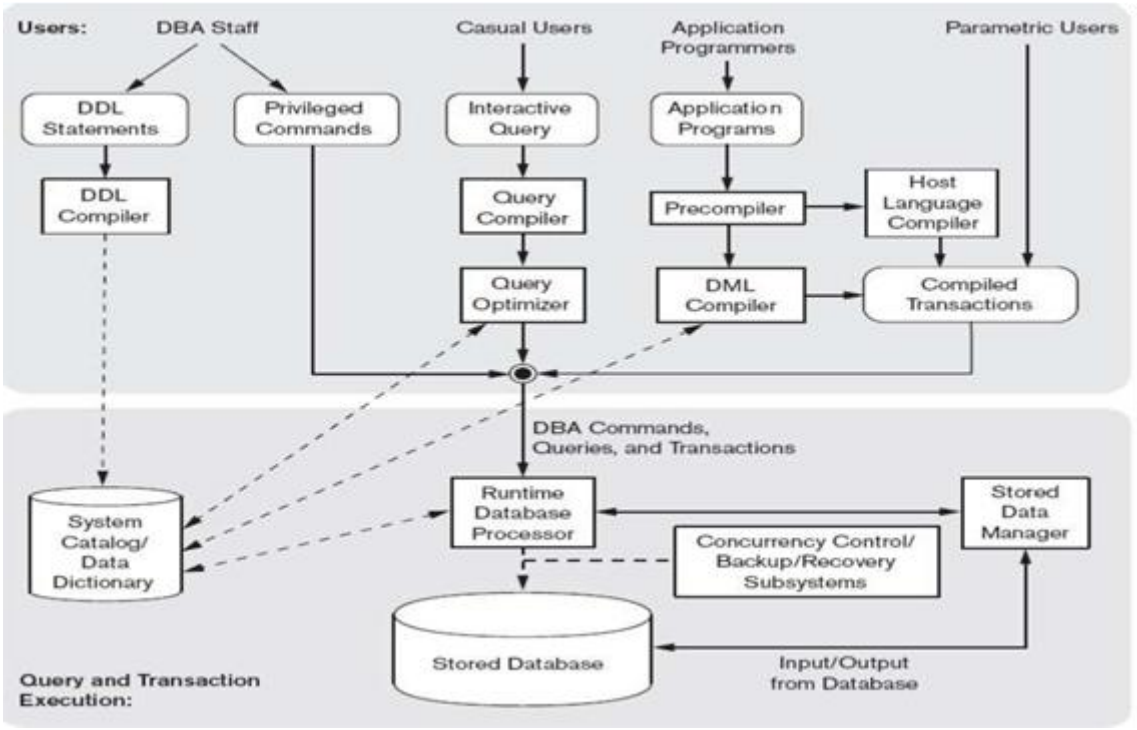
DBMS Interfaces

User-friendly interfaces provided by a DBMS may include the following:

1) **Menu-Based Interfaces for Web Clients or Browsing**: These interfaces present the user with lists of options (called menus) that lead the user through the formulation of a request. There is no need for the user to memorize the specific commands and syntax of a query language. Pull-down menus are a very popular technique in Web-based user interfaces.
2) **Forms-Based Interfaces:** A forms-based interface displays a form to each user. Users can fill out all of the form entries to insert new data, or they can fill out only certain entries, in which case the DBMS will retrieve matching data for the remaining entries. Forms are usually designed and programmed for naive users as interfaces to canned transactions.
3) **Graphical User Interfaces:** A GUI typically displays a schema to the user in diagrammatic form. The user then can specify a query by manipulating the diagram. In many cases, GUIs utilize both menus and forms. Most GUIs use a pointing device, such as a mouse, to select certain parts of the displayed schema diagram.
4) **Natural Language Interfaces:** These interfaces accept requests written in English or some other language and attempt to understand them. A natural language interface usually has its own schema, which is similar to the database conceptual schema, as well as a dictionary of important words. The natural language interface refers to the words in its schema, as well as to the set of standard words in its dictionary, to interpret the request. If the interpretation is successful, the interface generates a high-level query corresponding to the natural language request and submits it to the DBMS for processing; otherwise, a dialogue is started with the user to clarify the request.
5) **Speech Input and Output:** Applications with limited vocabularies such as inquiries for telephone directory, flight arrival/departure, and credit card account information are allowing speech for input and output to enable customers to access this information. The speech input is detected using a library of predefined words and used to set up the parameters that are supplied to the queries. For output, a similar conversion from text or numbers into speech takes place.
6) **Interfaces for Parametric Users**: Parametric users, such as bank tellers, often have a small set of operations that they must perform repeatedly. For example, a teller is able to use single function keys to invoke routine and repetitive transactions such as account deposits or withdrawals, or balance inquiries. Usually a small set of abbreviated commands is included, with the goal of minimizing the number of keystrokes required for each request.
7) **Interfaces for the DBA:** Most database systems contain privileged commands that can be used only by the DBA staff. These include commands for creating accounts, setting system parameters, granting account authorization, changing a schema, and reorganizing the storage structures of a database.

Database System Environment:

DBMS Component Modules



- The top part of the figure refers to the various users of the database environment and their interfaces. The lower part shows the internals of the DBMS responsible for storage of data and processing of transactions.
- DDL compiler-processes schema definitions, specified in the DDL, and stores descriptions of the schemas (meta-data) in the DBMS catalog.
- **Interactive query interface:** interface for Casual users and persons with occasional need for information from the database.
- **Query compiler-** validates for correctness of the query syntax, the names of files and data elements & compiles them into an internal form.
- **Query optimizer** concerned with the rearrangement and possible reordering of operations, elimination of redundancies, and use of correct algorithms and indexes 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.
- **Precompiler -** extracts DML commands from an application program and sends to the DML compiler for compilation into object code for database access.
- **Host language compiler -** 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.
- 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 withstatistics. 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.
- Stored data manager uses basic operating system services for carrying out low-level input/output (read/write) operations between the disk and main memory.
- Concurrency control and backup and recovery systems integrated into the working of the runtime database processor for purposes of transaction management.

Database System Utilities

Database utilities help the DBA to manage the database system.

Common utilities have the following types of functions:

- **Loading:** used to load existing data files such as text files or sequential files into the database.
- **Backup:** 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.
- **Database storage reorganization:** used to reorganize a set of database files into different file organizations, and create new access paths to improve performance.
- **Performance monitoring:** 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

- **Tools**
  - CASE : used in the design phase of database systems
  - Data dictionary: 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.
- **Application Development Environments**
  - **PowerBuilder (Sybase) or JBuilder (Borland):** provide an environment for developing database applications including database design, GUI evelopment, querying and updating, and application program development.
  - **Communications software:** allow users at locations remote from the database system site to access the database through computer terminals, workstations, or

personal computers. Integrated DBMS and data communications system is called a DB/DC system.

Centralized and Client-Server Architecture for DBMSs:

➢ Centralized DBMSs Architecture

All DBMS functionality, application program execution, and user interface processing carried out on one machine.

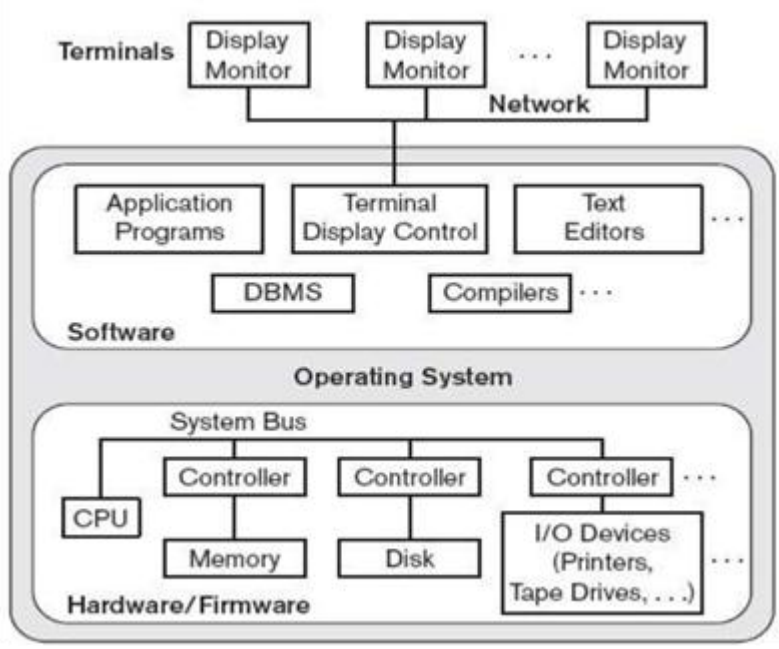


Figure: A physical centralized architecture

❖ **Disadvantages:**

- When the central site computer or database system goes down, then everyone is blocked from using the system.
- Communication costs from the terminals to the central site can expensive.

➢ Basic Client/Server Architectures

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.

- Define specialized servers with specific functionalities.
- For example file server that maintains the files of the client machines.
- The resources provided by specialized servers can be accessed by many client machines.
- The client machines provide the user with the appropriate interfaces to utilize these servers and local processing power to run local applications

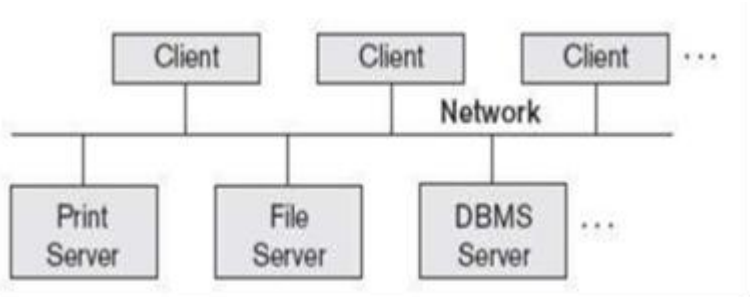


Figure: Logical two-tier client/server architecture

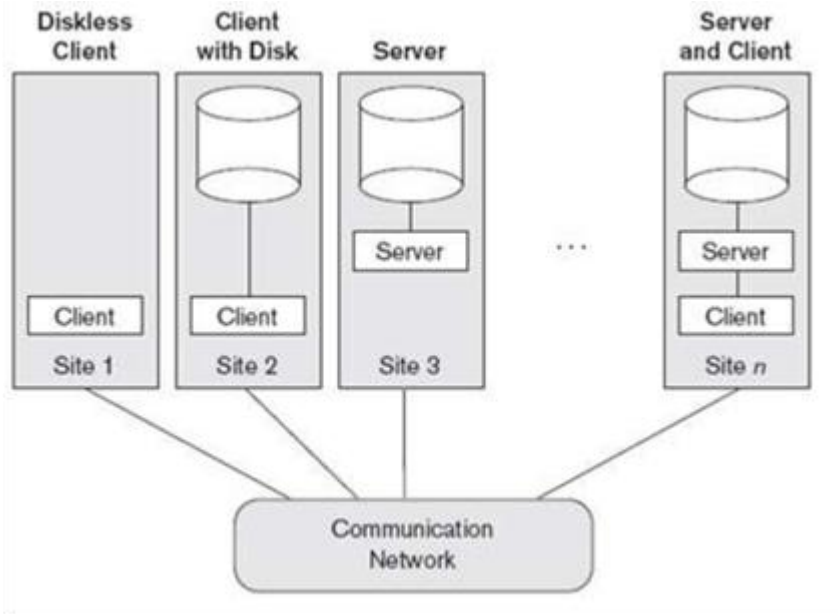


Figure: Physical two-tier client/server architecture

- The concept of client/server architecture assumes an underlying framework that consists of many PCs and workstations as well as a smaller number of mainframe machines, connected via LANs and other types of computer networks.
- A client is 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 that machine, 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.

The software components are distributed over two systems: client and server

✓ **Server handles**

- Query and transaction functionality related to SQL

✓ **Processing Client handles**

- User interface programs and application programs

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 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 to allow Java client programs to access one or more DBMSs through a standard interface.

Object-oriented DBMSs

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.

- **Server level:** May include the part of the DBMS software responsible for handling data storage on disk pages, local concurrency control and recovery, buffering and caching of disk pages.
- **Client level:** May handle the user interface, data dictionary functions, DBMS interactions with programming language compilers, global query optimization, concurrency control, and recovery across multiple servers, structuring of complex objects from the data in the buffers.

In this approach, the client/server interaction is more tightly coupled and is done internally by the DBMS modules some of which reside on the client and some on the server rather than by the users/programmers.

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.

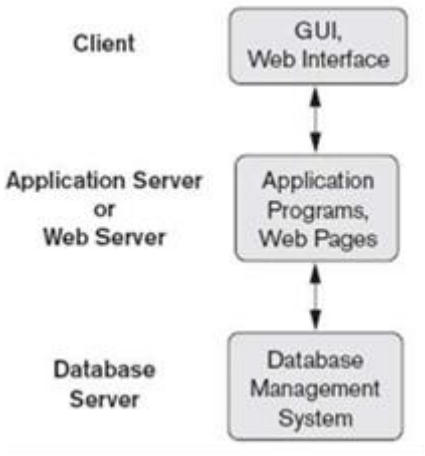
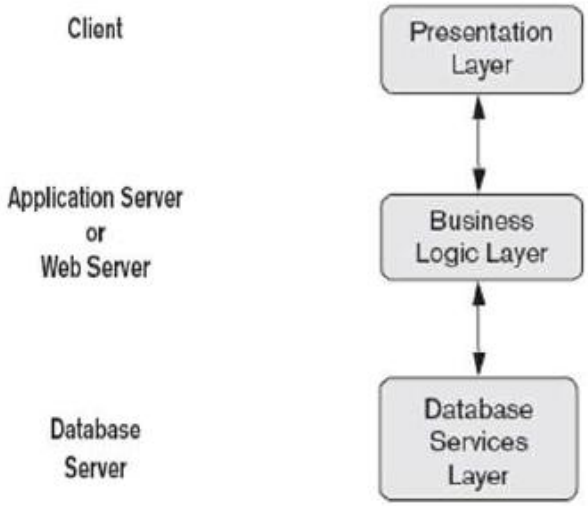


Figure: Logical three-tier client/server architecture

- **Client**
  - Contain GUI interfaces and some additional application-specific business rules.
- **Application Server or the Web Server**
  - Accepts requests from the client, processes the request and sends database queries and commands to the database server, and then passes 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.
  - It can also improve database security by checking client's credentials before forwarding a request from the database server.



The figure shows another architecture used by database and other application package vendors.

- **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.
- 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 bottom Layer:** Includes all the data management services.

➢ **N-tier Architecture:**

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. The business logic layer is divided into multiple layers.

- **Advantage:**
- 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).