**NOTES : DBMS**

**SEMESTER : 5TH,CSE/ISE**

**PREPARED BY: ASST.PROF KISHOR**

**DEPT OFCSE,**

**BIT, MANGALORE**

**UNIT – 1 : INTRODUCTION**

**INTRODUCTION**

* A **database** is a collection of related data. Here **data** means known facts that can be recorded and that have implicit meaning.
* **For example**, consider the names, telephone numbers, and addresses of the people we know. We may have recorded this data in an indexed address book or you may have stored it on a hard drive, using a personal computer and software such as Microsoft Access or Excel. This collection of related data with an implicit meaning is a database.
* **A database has the following implicit properties**:
* A database represents some aspect of the real world, sometimes called the **miniworld** or the **universe of discourse (UoD)**. Changes to the miniworld are reflected in the database.
* A database is a logically coherent collection of data with some inherent meaning. A random assortment of data cannot correctly be referred to as a database.
* A database 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.
* A **database management system (DBMS)** is a collection of programs that enables users to create and maintain a database. **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 also 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 miniworld, and generating reports from the data.
* **Sharing** a database allows multiple users and programs to access the database simultaneously.
* An **application program** accesses the database by sending queries or requests for data to the DBMS. A **query** typically causes some data to be retrieved. A **transaction** may cause some data to be read and some data to be written into the database.
* Other important functions provided by the DBMS include **protecting**the database and **maintaining**it over a long period of time.

The below figure illustrates the database system. The database and DBMS software together is called as a **database system**.



**AN EXAMPLE DATABASE**

* Consider an UNIVERSITY database for maintaining information concerning students, courses, and grades in a university environment. The below Figure 1.2 shows the database structure and a few sample data for such a database.
* The database is organized as five files, each of which stores **data records** of the same type.
* The STUDENT file stores data on each student, the COURSE file stores data on each course, the SECTION file stores data on each section of a course, the GRADE\_REPORT file stores the grades that students receive in the various sections they have completed, and the PREREQUISITE filestores the prerequisites of each course.
* To **define**this database, we must specify the structure of the records of each file by specifying the different types of **data elements** to be stored in each record.

**For example**: each STUDENT record includes data to represent the student’s Name, Studentnumber, Class and Major(like CS)such as freshman or ‘1’, sophomore or ‘2’, and so forth), and records and have *many relationships* among the records.

* We must also specify a **data type** for each data element within a record.

**For example**, we can specify that Name of STUDENT is a string of alphabetic characters, Studentnumber of STUDENT is an integer, and Grade of GRADE\_REPORT is a single character from the set {‘A’, ‘B’, ‘C’, ‘D’, ‘F’, ‘I’}.

* **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.
* The 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.
* Most medium-size and large databases include many types of records and have *many relationships* among the records.



**Figure-1.2:Adatabase that store student and course information**

* Database **manipulation**involves querying and updating.

Examples of queries are as follows:

* A list of all courses and grades of ‘Smith’ may be retrieved.
* List the prerequisites of the ‘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.

**DIFFERENCE BETWEEN FILE ORIENTED APPROACH & DATABASE APPROACH**:

**FILE ORIENTED APPROACH**

* Data files are created and processed by programs written by programmers or users of files.
* File created by different users of the organization often contain redundant data.
* The application programs depend upon the structural properties of the files.
* The application programs Interact directly with the operating system.

Disadvantages of file oriented approach:

* Files are created and processed by special programs: it is not possible by a program to access freely the other files, but it is necessary in the real world: **Example:** if the manager in an organization has to promote an employee, he must be able to access different files like EMPLOYEE\_PERFORMANCE file,EMPLOYEE\_PROJ file and so on.
* Files developed by different employees in an organization may have redundant data: Uncontrolled redundancy in files increases the cost of the storage and complicates the problem of updates and effects the consistent states of the data.
* Application programs depend upon the structure of the data files they process: Any changes made in the data files structure will require changes to be made in the application programs which are time consuming and expensive.
* Excessive programming effort is required: Application programs directly interact with the operating system. Hence programmers have to do much more in terms of programming to implement DBMS functions. It leads to poor data control.

**DATABASE APPROACH**

The database approach overcomes the drawbacks of file-oriented systems by providing the following features:

* It allows files to be integrated to avoid data redundancies.
* The user can access any file as long as he does not violate security constraints.
* Multiple users can have access to the collection of files.
* Each user may access a part of the database (a view is a subset of data base).
* Database system support centralized data control.
* The security, consistency and accuracy are maintained by DBMS rather than the user programs.

**CHARACTERISTICS OF THE DATABASE APPROACH**

The main characteristics of database approach are as follows:

* **Self-describing nature of a database system.**
* **Insulation between programs and data, and data abstraction.**
* **Support of multiple views of the data.**
* **Sharing of data and multiuser transaction processing.**

**Self-describing nature of a database system**:

* The database system contains not only the database itself but also a complete definition or description of the database structure and constraints.
* The definition is stored in the DBMS catalog, which contains information such as the structure of each file, the type and storage format of each data item, and various constraints on the data. The information stored in the catalog is called **meta-data**, and it describes the structure of the primary database.



* A general-purpose DBMS software package is not written for a specific database application. Therefore, it must refer to the catalog to know the structure of the files in a specific database, such as the type and format of data it will access. The DBMS software must work equally well with any numberofdatabase applications.

**Insulation between programs and data, and data abstraction**:

* In file oriented approach , any changes to the structure of a file requires changing all programsthat access that file. In DBMS access programs do not require such changes as the structure of data files is stored in the DBMS catalog separately from the access programs. This property is called **program-data independence**.
* **For example**, in a DBMS environment If we want to add another piece of data to each STUDENT record, say the Birth\_date, we only need to change the description of STUDENT records in the catalog to reflect the inclusion of the new data item Birth\_date, no programs are changed.
* In some types of database systems, such as object-oriented and object-relational systems users can define operations on data as part of the database definitions. 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 property is termed **program-operation independence**.
* The characteristic that allows program-data independence and program-operation independence is called **data abstraction**.

**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.

**Sharing of Data and Multiuser Transaction Processing**:

* A multiuser DBMS must allow multiple users to access the database at the same time.
* 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.
* A **transaction** is an executing program or processthat includes one or more database accesses, such as reading or updating of database records.
* The **isolation** property ensures that each transaction appears to execute in isolation from other transactions.
* The **atomicity** property ensures that either all the database operations in a transaction are executed or none are.

**ACTORS ON THE SCENE**

The people whose jobs involve the day-to-day use of a large database are called as actors on the scene and are as follows:

* **Database Administrators**
* **Database Designers**
* **End Users**
* **System Analysts and Application Programmers (Software Engineers)**

**Database Administrators**

* In a database environment, the primary resource is the database itself, and the secondary resource is the DBMS and related software. Administering these resources is the responsibility of the **database administrator (DBA)**.
* The DBA is responsible for authorizing access to the database, coordinating and monitoring its use and acquiring software and hardware resources as needed.
* The DBA is accountable for problems such as security breaches and poor system response time.

**Database Designers**

* Database designers are responsible for identifying the data to be stored in the database and for choosing appropriate structures to represent and store this data.
* It is the responsibility of database designers to communicate with all prospective database users in order to understand their requirements and to create a design that meets these requirements. In many cases, the designers are on the staff of the DBA.

**End Users**

* End users are the people whose jobs require access to the database for querying, updating, and generating reports; the database primarily exists for their use.
* The following are the categories of end users:
* **Casual end users:** They occasionally access the database, but they may need different information each time. They use a sophisticated database query language to specify their requests and are typically middle or high-level managers or other occasional browsers. Casual users learn only a few facilities of DBMS that they may use repeatedly.
* **Naive or parametric end users:** They constantly query and update the database, using standard types of queries and updates called canned transactions that have been carefully programmed and tested. Naive end users need to learn very little about the facilities provided by the DBMS. Example: Reservation agents for airlines, hotels, and car rental companies check availability for a given request and make reservations.
* **Sophisticated end users:** Theseinclude engineers, 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.
* **Standalone users:** Theymaintain personal databases by using ready-made program packages that provide easy-to-use menu-based or graphics-based interfaces. For example the user of a tax package that stores a variety of personal financial data for tax purposes.

**System Analysts and Application Programmers (Software Engineers):**

* **System analysts** determine the requirements of end users, especially naive and parametric end users, and develop specifications for standard canned transactions that meet these requirements.
* **Application programmers** implement these specifications as programs, then they test, debug, document, and maintain these canned transactions. Such analysts and programmers are commonly referred to as **software developers** or **software engineers.**

**WORKERS BEHIND THE SCENE**

**The people who are associated with the design, development, and operation of the DBMS software and system environment are called as the** **workers behind the scene**.

The following are categories of workers behind the scene:

* **DBMS system designers and implementers.**
* **Tool developers.**
* **Operators and maintenance personnel.**

**DBMS system designers and implementers**

They 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.

**Tool developers**

They design and implement **tools** which are the software packages that facilitate database modeling and design, database system design, and improved performance. Tools are optional packages that are often purchased separately. They include packages for database design, performance monitoring, natural language or graphical interfaces, prototyping, simulation, and test data generation.

**Operators and maintenance personnel**

They are responsible for the actual running and maintenance of the hardware and software environment for the database system.

**ADVANTAGES OF USING THE DBMS APPROACH**

The following are the advantages of using a DBMS:

* Controlling Redundancy.
* Restricting Unauthorized Access.
* Providing Persistent Storage for Program Objects.
* Providing Storage Structures and Search Techniques for Efficient Query Processing.
* Providing Backup and Recovery.
* Providing Multiple User Interfaces.
* Representing Complex Relationships among Data.
* Enforcing Integrity Constraints.
* Permitting Inferencing and Actions Using Rules.

**Controlling Redundancy**:

* Storing the same data multiple times is called as **redundancy**
* **Redundancy** leads to several problems like:
* Duplication of efforts: A single logical update, such as entering data on a new student need to be made multiple times, once for each file where student data is recorded.
* Storage space is wasted:When the same data is stored repeatedly Storage space is wasted and this problem may be serious for large databases.
* Inconsistency: The file representing the same data may become inconsistent because an update is applied to some of the files but not to others.
* In the database approach, the views of different user groups are integrated during database design. we can have a database design that stores each logical data item such as a student’s name or birth date in only one placein the database. This is known as **data normalization**, and it ensures consistency and saves storage space.
* In some cases it is sometimes necessary to use **controlled redundancy** to improve the performance of queries. By placing all the data together, we do not have to search multiple files to collect this data. This is known as **denormalization**.

**Restricting Unauthorized Access**:

* When multiple users share a large database, some 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.
* Some users may only be permitted to retrieve data, whereas others are allowed to retrieve and update. Hence, users or user groups are given account numbers protected by passwords, which they can use to gain access to the database.
* A DBMS should provide a **security and authorization subsystem**, which the DBA uses to create accounts and to specify account restrictions. Then, the DBMS should enforce these restrictions automatically. For example, only the dba’s staff may be allowed to use certain **privileged software**, such as the software for creating new accounts.
* Parametric users may be allowed to access the database only through the predefined canned transactions developed for their use.

**Providing Persistent Storage for Program Objects**:

* Databases can be used to provide **persistent storage** for program objects and data structures. This is one of the main reasons for **object-oriented database systems**. Object-oriented database systems are compatible with programming languages such as C++ and Java, and the DBMS software automatically performs any necessary conversions. Hence, a complex object in C++ can be stored permanently in an object-oriented DBMS. Such an object is said to be **persistent**, since it survives the termination of program execution and can later be directly retrieved by another C++ program.
* The persistent storage of program objects and data structures is an important function of database systems. Traditional database systems often suffered from the so called **impedance mismatch problem**, since the data structures provided by the DBMS were incompatible with the programming language’s data structures. Object-oriented database systems typically offer data structure **compatibility** with one or more object-oriented programming languages.

**Providing Storage Structures and Search Techniques for Efficient Query Processing**:

* Database systems must provide capabilities for efficiently executing queries and updates*.* Because the database is typically stored on disk, the DBMS must provide specialized data structures and search techniques to speed up disk search for the desired records. Auxiliary files called **indexes** are used for this purpose.
* The DBMS often has a **buffering** or **caching** module that maintains parts of the database in main memory buffers to speed up retrieval of data through querying.
* The **query processing and optimization** module of the DBMS is responsible for choosing an efficient query execution plan for each query based on the existing storage structures.

**Providing Backup and Recovery**:

* A DBMS must provide facilities for recovering from hardware or software failures. 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.

**Providing Multiple User Interfaces**:

* Many types of users with varying levels of technical knowledge use a database, a DBMS should provide a variety of user interfaces.
* Example:
  + 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.
* Both forms-style interfaces and menu-driven interfaces are commonly known as **graphical user interfaces (GUIs)**.

**Representing Complex Relationships among Data:**

A database may include numerous varieties of data that are interrelated in many ways. For example in university database, the record of a student in the STUDENT file is related to records in the GRADE\_REPORT file.

**Enforcing Integrity Constraints**:

* Most database applications have certain **integrity constraints** that must hold for the data. A DBMS should provide capabilities for defining and enforcing these constraints. The simplest type of integrity constraint involves specifying a data type for each data item.
* For example: In university database we can specify that the value of the Class data item

Within each STUDENT record must be a one digit integer and that the value of Name must be a string of no more than 30 alphabetic characters.

**Permitting Inferencing and Actions Using Rules**:

* Some database systems provide capabilities for defining deduction rules for inferencingnew information from the stored database facts. Such systems are called **deductive database systems**.
* For example, there may be complex rules in the miniworld application for determining when a student is on probation. These can be specified *declaratively* as **rules**, which when compiled and maintained by the DBMS can determine all students on probation. In a traditional DBMS, an explicit
* More powerful functionality is provided by **active database systems**, which provide active rules that can automatically initiate actions when certain events and conditions occur.

**ADDITIONAL IMPLICATIONS OF USING THE DATABASE APPROACH**

The following are some additional implications of using the database approach that can benefit most organizations:

* **Potential for Enforcing Standards:**

The database approach permits the DBA to define and enforce standards among database users in a large organization. This facilitates communication and cooperation among various departments, projects, and users within the organization

* **Reduced Application Development Time:**
* A prime selling feature of the database approach is that developing a new application such as the retrieval of certain data from the database for printing a new report takes very little time.
* Designing and implementing a large multiuser database from scratch may take more time than writing a single specialized file application.
* 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 system.
* **Flexibility:**
* It may be necessary to change the structure of a database as requirements change.
* For example, a new user group may emerge that needs information not currently in the database.
* Modern DBMSs allow certain types of evolutionary changes to the structure of the database without affecting the stored data and the existing application programs.
* **Availability of Up-to-Date Information:**

A DBMS makes the database available to all users. As soon as one user’s update is applied to the database, all other users can immediately see this update. This availability of up-to-date information is essential for many transaction-processing applications, such as reservation systems and it is made possible by the concurrency control and recovery subsystems of a DBMS.

* **Economies of Scale:**
* The DBMS approach permits consolidation of data and applications, thus reducing the amount of wasteful overlap between activities of data-processing personnel in different projects or departments as well as redundancies among applications.
* This enables the whole organization to invest in more powerful processors, storage devices, or communication gear, rather than having each department purchase its own (lower performance) equipment. This reduces overall costs of operation and management.

**A BRIEF HISTORY OF DATABASE APPLICATIONS**

* **Early Database Applications Using Hierarchical and Network Systems:**
* Many 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.
* One of the main problems with early database systems was the intermixing of conceptual relationships with the physical storage and placement of records on disk.
* These systems did not provide sufficient *data abstraction* and *program-data independence* capabilities. For example, the grade records of a particular student could be physically stored next to the student record.
* Another shortcoming of early systems was that they 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.
* Relational systems provided flexibility to develop new queries quickly and to reorganize the database as requirements changed. 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:**
* The emergence of object-oriented programming languages in the 1980s and the need to store and share complex, structured objects led to the development of object-oriented databases (OODBs).
* They also incorporated many of the useful object-oriented paradigms, such as abstract data types, encapsulation of operations, inheritance, and object identity.
* They are now mainly used in specialized applications, such as engineering design, multimedia publishing, and manufacturing systems.
* **Interchanging Data on the Web for E-Commerce Using XML:**
* The World Wide Web provides a large network of interconnected computers. In the 1990s, electronic commerce (e-commerce) emerged as a major application on the Web.
* The information on e-commerce Web pages were often dynamically extracted data from DBMSs. A variety of techniques were developed to allow the interchange of data on the Web.
* 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.
* **Some examples** of these applications are S**cientific** applications, Storage and retrieval of **images** and **videos, Data mining** applications **Spatial** applications and **Time series** applications.

**DATABASES VERSUS INFORMATION RETRIEVAL**

* Traditionally, database technology applies to structured and formatted data that arises in routine applications in government, business, and industry.
* 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**

* 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.

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

**DATA MODELS, SCHEMAS, AND INSTANCES**

* **Data abstraction**  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** is a collection of concepts that can be used to describe the structure of a database. structure of a database includes 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.
* The concepts of **dynamic aspect** or **behavior** of a database application can also be included in the data model. 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.

**CATEGORIES OF DATA MODELS**

The data models are categorized according to the types of concepts they use to describe the database structure as follows:

* **High-level** or **conceptual data models**
* **low-level** or **physical data models**
* **Representational** or **implementation** **data models**

**High-Level** Or **Conceptual Data Models:**

* Conceptual data model provide concepts that are close to the way many users perceive data.
* 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 miniworld 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.

**Low-Level** Or **Physical Data Models:**

* Physical data model provide concepts that describe the details of how data is stored on the computer storage media, typically magnetic disks.
* Concepts provided by low-level data models are generally meant for computer specialists, not for end users.
* 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 structure that makes the search for particular database records efficient.
* An **index** is an example of an access path that allows direct access to data using an index term or a keyword.

**Representational** Or **Implementation** **Data Models:**

* Representational data model 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**.
* The **object data model** is 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).

**SCHEMAS, INSTANCES, AND DATABASE STATE**

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



* The Figure 2.1 shows a schema diagram for the University database. The diagram displays the structure of each record type but not the actual instances of records.
* The each object in the schema such as STUDENT or COURSE is called 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 such as the data type of each data item or the relationships among the various files.
* 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.
* Every time a record is inserted or deleted or the value of a data item in a record is changed, one state of the database is changed into another.
* When a new database is **defined**, its schema is specified only to the DBMS. At this point, the corresponding database state is the *empty state* with no data. The database gets its initial state when the database is first **populated** or **loaded** with the initial data. 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. Changing the database schema to represent the changes in the real world is known as **schema evolution**.

**THREE-SCHEMA ARCHITECTURE AND DATA INDEPENDENCE**

**The Three-Schema Architecture**

* The three schema architecture was proposed to help in achieving and visualizing the characteristics of the database approach. The below figure illustrates 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:
* 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.
* 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 data model.
* The three-schema architecture is a convenient tool with which the user can visualize the schema levels in a database system. The processes of transforming requests and results between levels are called **mappings**.

**Data Independence**

* **Data independence** is the capacity to change the schema at one level of a database system without having to change the schema at the next higher level.
* The following are the types of data independence:
* **Logical data independence**
* **Physical data independence**
* **Logical data independence**
* It is the capacity to change the conceptual schema without having to change the external schemas or application programs.
* Conceptual schema may be changed to expand the database, to change constraints, or to reduce the database by removing a record type or data item.
* 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.
* Logical data independence is harder to achieve because it allows structural and constraint changes without affecting application programs.
* **Physical data independence**
* It 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 may be 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, there is no need to change the conceptual schema. For example, providing an access path to improve retrieval speed of section records in university database 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.
* 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.
* In 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.
* The three-schema architecture can make it easier to achieve true data independence, both physical and logical.

**DATABASE LANGUAGES AND INTERFACES**

**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.
* 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 internal and conceptual 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.
* 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.
* For a true three-schema architecture, needs 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 .
* 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.
* 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 other features.
* 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. They are also called **set-at-a-time** or **set-oriented** DMLs. Example SQL. A high-level DML often specifies which data to retrieve rather than how to retrieve it therefore, such languages are also called **declarative**. A high-level DML used in a standalone interactive manner is called a **query language**.
* 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. Example: DL/1
* A query in, 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**.
* 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.

**DBMS INTERFACES**

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

* **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. The query is composed step by step by picking options from a menu that is displayed by the system.
* Pull-down menus are a very popular technique in **Web-based user interfaces**. They are also often used in **browsing interfaces**, which allow a user to look through the contents of a database in an exploratory and unstructured manner.
* **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. Many DBMSs have **forms specification languages**, which are special languages that help programmers, specify such forms.
* **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.
* Most GUIs use a **pointing device**, such as a mouse, to select certain parts of the displayed schema diagram.
* **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*.
* 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.
* **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.
* **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.
* Systems analysts and programmers design and implement a special interface for each known class of naive users. Usually a small set of abbreviated commands is included, with the goal of minimizing the number of keystrokes required for each request. For example, function keys in a terminal can be programmed to initiate various commands. This allows the parametric user to proceed with a minimal number of keystrokes.
* **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.

**THE DATABASE SYSTEM ENVIRONMENT**

**DBMS COMPONENT MODULES**



* The Figure 2.3 above 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 internals of the DBMS responsible for storage of data and processing of transactions.
* 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 this has a considerable effect on performance.
* 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.
* The top part of Figure 2.3 shows interfaces for the DBA staff, casual users application programmers and parametric users.
* The DBA staff works on defining the database and tuning it by making changes to its definition using the DDL and other privileged commands. 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, data types of data items and so on.
* Casual users and persons with occasional need for information from the database interact using some form of interface, which we call the **interactive query** interface. These queries are parsed and validated for correctness by a **query compiler** that compiles them into an internal form. This internal query is subjected to query optimization.
* The **query optimizer** is 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.
* Application programmers write programs in host languages such as Java, C, or C++ that are submitted to a precompiler. The **precompiler** 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.
* In the lower part of Figure 2.3, the **runtime database processor** executes the privileged commands, the executable query plans, and 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 figure 2.3 shows **concurrency control** and **backup and recovery systems** asseparate modules. They are integrated into the working of the runtime database processor for purposes of transaction management.

**DATABASE SYSTEM UTILITIES**

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. Some vendors are offering products that generate the appropriate loading programs, given the existing source and target database storage descriptions (internal schemas). Such tools are also called **conversion tools**. Example: Cincom’s SUPRA Server SQL.

* **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.

* **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.

**TOOLS, APPLICATION ENVIRONMENTS AND COMMUNICATIONS FACILITIES**

* Other tools are often available to database designers, users, and the DBMS.
* CASE tools12 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**.
* **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.
* 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. The integrated DBMS and data communications system is called a **DB/DC** system.
* 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**

Earlier DBMS 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 most users accessed such systems via computer terminals that did not have processing power and only provided display capabilities. Therefore, all processing was performed on a remote on the computer system, and only display information and controls were sent from the remote computer to the display terminals, which were connected to the central computer via various types of communications networks. As prices of hardware declined, terminals were replaced with PCs and workstations but 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 2.4 illustrates the physical components in a centralized architecture.



**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.
* The idea is to define **specialized servers** with specific functionalities. The resources provided by specialized servers can be accessed by many client machines.
* Examples
* **File server** maintains the files of the client machines which are to a number of PCs or small workstations connected to it.
* **Print server** handles all print requests by the client machines connected to it.
* **Web servers** or **e-mail servers** also fall into the specialized server category.
* A **client**  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 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 above figure 2.5 illustrates client/server architecture at the logical level Some machines would be client sites only Other machines would be dedicated servers, and others would have both client and server functionality.
* In general, some machines install only client software, others only server software, and still others may include both client and server software as shown in the below figure 2.6 the physical two-tier client/server architecture.



**TWO-TIER CLIENT/SERVER ARCHITECTURES FOR DBMSS**

* In relational database management systems (RDBMSs), the query and transaction functionality related to SQL processing remained on the server side. In such 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**.
* 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.
* Another 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.
* For example, the **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, and other such functions.
* The **client level** may handle the user interface, data dictionary functions, DBMS interactions with programming language compilers, global query optimization, concurrency control and recovery and other such functions.
* 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.
* In such a client/server architecture, the server has been called a **data server** because it provides data in disk pages to the client. This data can then be structured into objects for the client programs by the client-side DBMS software.
* The architecture is 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.

**THREE-TIER AND N-TIER ARCHITECTURES FOR WEB APPLICATIONS**

* The emergence of the Web changed the roles of clients and servers, leading to the three-tier architecture.
* Many Web applications use the **three-tier architecture**, which adds an intermediate layer between the client and the database server, as illustrated in Figure 2.7(a).
* 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 that are used to access data from the database server.



* Clients contain GUI interfaces and some additional application-specific business rules. The intermediate server accepts requests from the client, processes the request and sends database queries and commands to the database server, and then acts as a conduit for passing (partially) processed data from the database server to the clients, where it may be processed further and presented to users in GUI format. The **user interface, application rules, and data access**act as the three tiers.
* Figure 2.7(b) shows another architecture used by database and other application package vendors. The presentation layer displays information to the user and allows data entry.
* The business logic layer handles intermediate rules and constraints before data is passed up to the user or down to the DBMS.
* The bottom layer includes all data management services.
* 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.
* 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).

**CLASSIFICATION OF DATABASE MANAGEMENT SYSTEMS**

**The DBMS can be classified based on the data model as follows**:

* **Relational DBMS which are base on relational data model**. Example: oracle
* **Object Oriented DBMS which are based on Object data model.**
* **Object-Relational DBMS.**
* **Hierarchical DBMS which are based on hierarchical data model.** Examples: IMS (IBM),System 2K (SAS Inc.) and TDMS.
* **Network DBMS which are based on network data model and** reasonable percentage of Worldwide-computerized data is still in these so-called **legacy database systems**. Examples: IDMS, DMS 1100 VAXDBMS and SUPRA.
* 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.**

**The DBMS can be classified based on the number of users supported by the system as follows:**

* **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.

**The DBMS can be classified based on the number of sites over which the database is distributed as follows:**

* **CENTRALIZED**
* 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.
* **DISTRIBUTED**
* A **Distributed** DBMS (DDBMS) can have the actual database and DBMS software distributed over many sites, connected by a computer network.
* **Homogeneous** DDBMSs use the same DBMS software at all the sites.
* **Heterogeneous** DDBMSs can use different DBMS software at each site.
* **Federated** DBMS or **multidatabase system**, in which the participating DBMSs are loosely coupled and have a degree of local autonomy.

**The DBMS can be classified based on cost as follows:**

* Open source or free DBMS products like MySQL and PostgreSQL that are supported by third-party vendors with additional services.
* Commercial DBMS like oracle or Standalone single user versions of some like Microsoft Access are sold per copy or included in the overall configuration of a desktop or laptop.

**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.**

**The DBMS can be classified based on the Purpose as follows**:

* **General purpose DBMS**
* **Special purpose DBMS**

**THE NETWORK MODEL**

The **network model** represents data as record types and also represents a limited type of 1:N relationship, called a **set type**. Figure 2.8 below shows a network schema diagram for the university database where record types are shown as rectangles and set types are shown as labeled directed arrows. The network model, also known as the CODASYL DBTG model, it has an associated record-at-a-time language that must be embedded in a host programming language. The network DML was proposed in the 1971 Database Task Group (DBTG) Report as an extension of the COBOL language.



**THE HIERARCHICAL MODEL**

The **hierarchical model** represents data as hierarchical tree structures. Each hierarchy represents a number of related records. There is no standard language for the hierarchical model. A popular hierarchical DML is DL/1 of the IMS system. It dominated the DBMS market for over 20 years between 1965 and 1985 and is still a widely used DBMS worldwide, holding a large percentage of data in governmental, health care, and banking and insurance databases.

**QUESTION BANK**

01) Explain the typical components module of a DBMS with neat diagram. (10marks)

02) What are the main characteristics of database approach over the file processing

approach? (08marks)

03) Explain different categories of data models? (06marks)

04) Explain the three-schema architecture? (06marks)

05) What are the responsibilities of DBA and database designers? (04marks)

06) List the advantages and disadvantages of DBMS? Discuss any five advantages by comparing

with file system? (08marks)

07) Define the terms

* 1. Data
  2. Database
  3. Metadata
  4. Datamodel
  5. Canned transactions
  6. Database designers (10marks)

08) With neat diagram explain the operation of two-tier client/server architecture

for the RDBMS? (08marks)

09) What are the data base utilities list few functions that utilities perform. (08marks)

10) Discuss the main characteristics of database approach. How does it

differ from the traditional file system. (10marks)

11) Briefly explain the advantages of object oriented systems. (05marks)

12) What is data independence? Bring out different types of independence. (06marks)

13) Discuss the criteria used to classify DBMS. (06marks)

14) Define the terms: DDL,DML and DCL with examples. (06marks)

15) List the differences between logical and physical data independence (05marks)

16) Define the following and explain with an example each. (05marks)

* 1. Snapshot
  2. Intension
  3. Extension
  4. Schema construct

17) What is meant by “persistent storage for program objects” explain. (05marks)

18) Define schema and instance. (04marks)

19) Explain the advantages of DBMS. (10marks)

20) Define and explain the importance of database catalog and explain the

Internal storage format of a catalog with example. (06marks)

21) What are the problems associated with three schema architecture. (02marks)

22) Why would you choose a database system over flat files. When would make

Sense not to use database system. (10marks)

23) What is a transaction? In what ways it is different from a c program (05marks)

24) Discuss different types of user friendly interfaces and the types of users

Who typically use each? (08marks)

25) With a neat diagram explain the architecture of DBMS and also discuss three

Levels of abstraction? (08marks)

**ASSIGNMENT QUESTIONS**

1. What is program data independence? Why do we need mapping between Schema levels? How do different schema definition languages support this architecture?
2. Discuss the main categories of data models. What are the basic differences between the relational model, the object model, and the XML model?
3. What is the difference between logical data independence and physical data independence? Which one is harder to achieve? Why?
4. What is the difference between procedural and nonprocedural DMLs?
5. With what other computer system software does a DBMS interact?
6. What is the difference between the two-tier and three-tier client/server architectures?
7. What is the additional functionality incorporated in *n*-tier architecture (*n* > 3)?
8. Discuss the differences between database systems and information retrieval systems.
9. Discuss the capabilities that should be provided by a DBMS.
10. What does sharing, protecting and maintaining a database mean.