Database system concepts & architecture

Data Model

Data abstraction generally refers to the suppression of details of data organization and storage, and the highlighting of the essential features for an improved understanding of data. One of the main characteristics of the database approach is to support data abstraction so that different users can perceive data at their preferred level of detail. A data model provides necessary means to achieve data abstraction.

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.

There exists various data models namely, entity-relationship model (ER model), relational model, object-oriented model, object-relational model, hierarchical model and network model (HM and NM are called legacy model). Most data models also include a set of basic operations for specifying retrievals and updates on the database.

Categories of Data Models

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

*Conceptual data model*s use concepts such as entities, attributes, and relationships. An entity represents a real-world object or concept, such as an employee or a project from the mini world that is described in the database. An attribute represents some property of interest that further describes an entity, such as the employee’s name or salary. A relationship among two or more entities represents an association among the entities. A popular high-level conceptual data model is Entity-Relationship model (ER model). The Enhanced Entity-Relationship model (EER model) describes additional abstractions used for advanced modeling, such as generalization, specialization, and categories (union types).

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

We can regard the object data model as an example of a new family of higher level implementation data models that are closer to conceptual data models.

*Physical data model*s 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.

Schemas 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**. Each object in a schema diagram is called a schema construct.

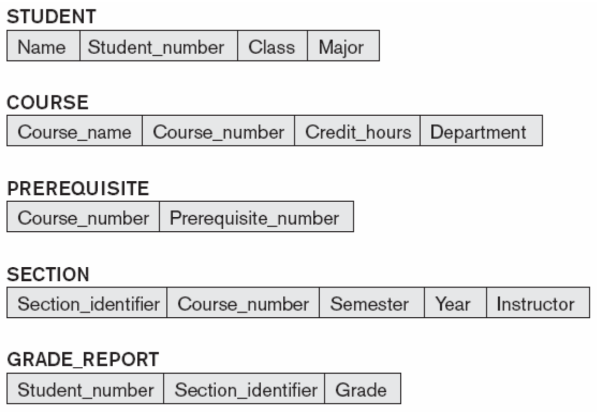
STUDENT
Name | Student_number | Class | Major
COURSE
Course_name | Course_number | Credit_hours | Department
PREREQUISITE
Course_number | Prerequisite_number
SECTION
Section_identifier | Course_number | Semester | Year | Instructor
GRADE_REPORT
Student_number | Section_identifier | Grade

Figure 2.1
Schema diagram for the database in Figure 1.2.

The data in the database at a particular moment in time is called a **database state** or **snapshot** or **database instance**. It is also called the current set of occurrences or instances in the database. 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. 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.
Though the schema is not supposed to change frequently, it is not uncommon that changes occasionally need to be applied to the schema as the application requirements change. This is known as schema evolution. Most modern DBMSs include some operations for schema evolution that can be applied while the database is operational.

Three-Schema Architecture

The goal of the three-schema architecture, illustrated in Figure 2.2, 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, 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.
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. 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.

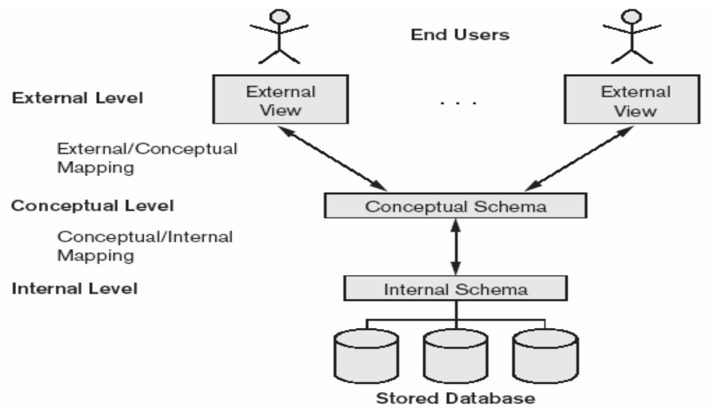

Figure 2.2
The three-schema architecture.

The three schemas are only descriptions of data; the stored data that actually exists is at the physical level only. 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 database retrieval, the data extracted from the stored database must be reformatted to match the user's external view. The processes of transforming requests and results between levels are called mappings. These mappings may be time consuming, so some DBMSs—especially those that are meant to support small databases—do not support external views. Even in such systems, however, a certain amount of mapping is necessary to transform requests between the conceptual and internal levels.

Data Independence

The concept of data independence is 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:

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. In the last case, external schemas that refer only to the remaining data should not be affected.

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.

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

Whenever we have a multiple-level DBMS, its catalog must be expanded to include information on how to map requests and data among the various levels. The DBMS uses additional software to accomplish these mappings by referring to the mapping information in the catalog. Data independence occurs because when the schema is changed at some level, the schema at the next higher level remains unchanged; only the mapping between the two levels is changed. Hence, application programs refer-ring 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. However, the two levels of mappings create an overhead during compilation or execution of a query or program, leading to inefficiencies in the DBMS. Because of this, few DBMSs have implemented the full three-schema architecture.

A database language facilitates creation and manipulation of a database. A database language can be categorized as data definition language (DDL) and data manipulation language (DML). A **DDL** is used by the DBA and by database designers to define physical and representational schemas respectively. 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. The language that describes the external schema is called view definition language (VDL). The data manipulation language (DML) provides retrieval, insertion, deletion, and modification of the data. The SQL DML is called a high-level DML because it can retrieve and manipulate multiple records with a single DML statement. A DML is called a low-level DML if it can retrieve and manipulate only one record with a single DML statement. Another language, the storage definition language (SDL), is used to specify the internal schema. 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. The SDL was a component in early versions of SQL but has been removed from the language to keep it at the conceptual and external levels only. Whenever DML commands are embedded in a general purpose programming language, that language is called the *host language* and the DML is called the *data sublanguage*.

DBMS interfaces

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

i) 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. Menus do away with the need to memorize the specific commands and syntax of a query language; rather, 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. SQL*Forms is a form-based language that specifies queries using a form designed in conjunction with the relational database schema. Oracle Forms is a component of the Oracle product suite that provides an extensive set of features to design and build applications using forms. Some systems have utilities that define a form by letting the end user interactively construct a sample form on the screen.

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.

Natural language interfaces

These interfaces accept requests written in English or some other language and attempt to understand them. The natural language interface tries 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. The capabilities of natural language interfaces have not advanced rapidly. Today, we see search engines that accept strings of natural language (like English or Spanish) words and match them with documents at specific sites (for local search engines) or Web pages on the Web at large (for engines like Google or Ask).They use predefined indexes on words and use ranking functions to retrieve and present resulting documents in a decreasing degree of match. Such “free form” textual query interfaces are not yet common in structured relational or legacy model databases, although a research area called keyword-based querying has emerged recently for relational databases.

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.

Database System Environment

A DBMS is a complex software system consisting of a large number of modules (sub system) and interacts with operating system and compiler of programming language.

DBMS component modules

The database and the DBMS catalog are usually stored on disk. A higher-level **stored data manager** module of the DBMS controls access to DBMS information (data and meta-data) that is stored on disk. Let us consider the top part of Figure 2.3 first. It shows interfaces for the DBA staff, casual users who work with interactive interfaces to formulate queries, application programmers who create programs using some host programming languages and parametric users who do data entry work by supplying parameters to predefined transactions. The DBA staff works on defining the database and tuning it by making changes to its definition using the DDL and other privileged commands.

The DDL compiler processes schema definitions, specified in the DDL, and stores descriptions of the schemas (meta-data) in the DBMS catalog. The catalog includes information such as the names and sizes of files, names and data types of data items, storage details of each file, mapping information among schemas, and constraints. In addition, the catalog stores many other types of information that are needed by the DBMS modules, which can then look up the catalog information as needed. 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 (menu-based or forms-based interface used to generate the interactive query automatically). These queries are parsed and validated for correctness of the query syntax, the names of files and data elements, and so on by a query compiler that compiles them into an internal form. This internal query is subjected to query optimization. Among other things, the query optimizer is concerned with there arrangement 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 pre-compiler. The pre-compiler extracts DML commands from an application program written in a host programming language. These commands are sent to the DML compiler for compilation into object code for database access. The rest of the program is sent to the host language compiler. The object codes for the DML commands and the rest of the program are linked, forming a canned transaction whose executable code includes calls to the runtime database processor. Canned transactions are executed repeatedly by parametric users, who simply supply the parameters to the transactions. Each execution is considered to be a separate transaction. An example is a bank withdrawal transaction where the account number and the amount may be supplied as parameters.

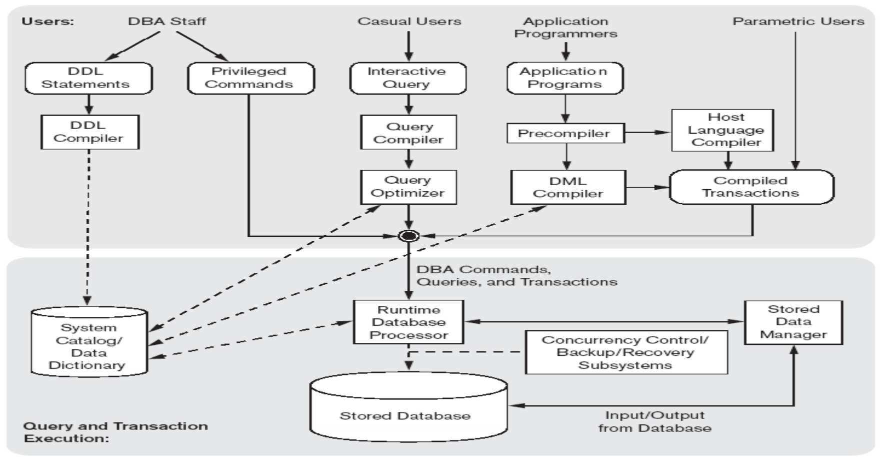

Figure 2.3
Component modules of a DBMS and their interactions.

In the lower part of Figure 2.3, the runtime database processor executes (1) the privileged commands, (2) the executable query plans, and (3) the canned transactions with runtime parameters. It works with the system catalog and may update it with statistics. It also works with the stored data manager, which in turn uses basic operating system services for carrying out low-level input/output (read/write) operations between the disk and main memory. The runtime database processor handles other aspects of data transfer, such as management of buffers in the main memory. Some DBMSs have their own buffer management module while others depend on the OS for buffer management. We have shown concurrency control and backup and recovery systems separately as a module in this figure. They are integrated into the working of the runtime database processor for purposes of transaction management.

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

Figure 2.3 is not meant to describe a specific DBMS; rather, it illustrates typical DBMS modules. The DBMS interacts with the operating system when disk accesses—to the database or to the catalog—are needed. If the computer system is shared by many users, the OS will schedule DBMS disk access requests and DBMS processing along with other processes. On the other hand, if the computer system is mainly dedicated to running the database server, the DBMS will control main memory buffering of disk pages. The DBMS also interfaces with compilers for general-purpose host programming languages, and with application servers and client programs running on separate machines through the system network interface.

Database System Utilities

In addition to possessing the software modules, 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. You will also find *conversion* tools that convert database from legacy model to relational model.

Backup

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

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.

Centralized DBMS architecture

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.

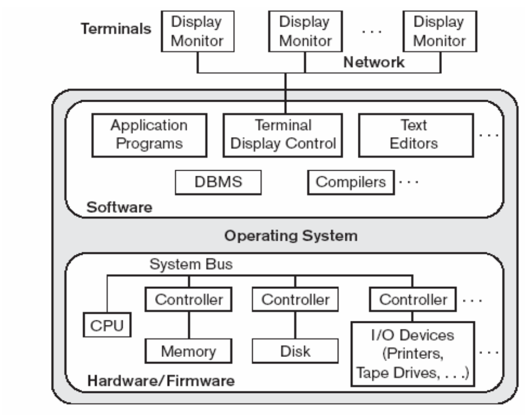

Figure 2.4
A physical centralized architecture.

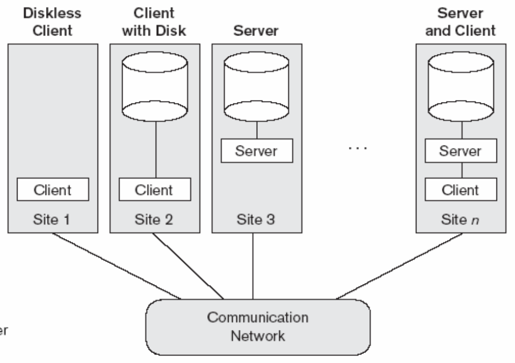

Figure 2.6
Physical two-tier client/server architecture.

Basic Client/Server architectures

The idea is to define specialized servers (database servers, Web servers, e-mail servers, file server, print server) with specific functionalities. The client machines provide the user with the appropriate interfaces to utilize these servers, as well as with local processing power to run local applications and other software and equipment are connected via a network.

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 in this framework is typically a user machine that provides user interface capabilities and local processing. When a client requires access to additional functionality—such as database access—that does not exist at 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. In general, some machines install only client software, others only server software, and still others may include both client and server software, as illustrated in Figure 2.6. However, it is more common that client and server software usually run on separate machines. Two main types of basic DBMS architectures were created on this underlying client/server framework: two-tier and three-tier.

Two-Tier Client/Server Architectures for DBMSs

Two-tier architectures because the software components are distributed over two systems: client and server. The advantages of this architecture are its simplicity and seamless compatibility with existing systems. The emergence of the Web changed the roles of clients and servers, leading to the three-tier architecture.

Many Web applications use an architecture called the three-tier architecture, which adds an intermediate layer between the client and the database server. The user interface, application rules, and data access act as the three tiers. The presentation layer displays information to the user and allows data entry. The business logic layer handles intermediate rules and constraints before data is passed up to the user or down to the DBMS. The bottom layer includes all data management services. The middle layer can also act as a Web server, which retrieves query results from the database server and formats them into dynamic Web pages that are viewed by the Web browser at the client side.

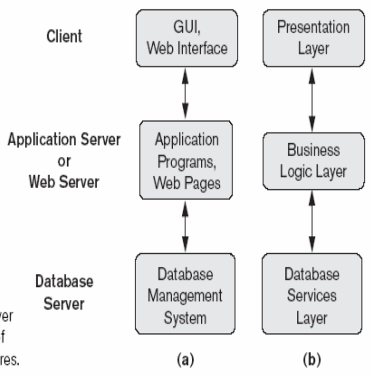

Figure 2.7
Logical three-tier client/server architecture, with a couple of commonly used nomenclatures.

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 System

Several criteria are normally used to classify DBMS. The first is the data model on which the DBMS is based. The main data model used in many current commercial DBMSs is the relational data model. The object data model has been implemented in some commercial systems but has not had widespread use. Many legacy applications still run on database systems based on the hierarchical and network data models.

Legacy application software is the one that had been superseded but is difficult to replace because of its wide use.

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

The third criterion is the number of sites over which the database is distributed. A DBMS is centralized if the data is stored at a single computer site. A centralized DBMS can support multiple users, but the DBMS and the database reside totally at a single computer site. A distributed DBMS (DDBMS) can have the actual database and DBMS software distributed over many sites, connected by a computer network. Homogeneous DDBMSs use the same DBMS software at all the sites, whereas heterogeneous DDBMSs can use different DBMS software at each site. It is also possible to develop middleware software to access several autonomous preexisting databases stored under heterogeneous DBMSs. This leads to a federated DBMS (or multi database system), in which the participating DBMSs are loosely coupled and have a degree of local autonomy. Many DDBMSs use client-server architecture.

We can also classify a DBMS on the basis of the types of access path options for storing files. One well-known family of DBMSs is based on inverted file structures.

Finally, a DBMS can be general purpose or special purpose. When performance is a primary consideration, a special-purpose DBMS can be designed and built for a specific application; such a system cannot be used for other applications without major changes. Many airline reservations and telephone directory systems developed in the past are special-purpose DBMSs. These fall into the category of online transaction processing (OLTP) systems, which must support a large number of concurrent transactions without imposing excessive delays.

About data model

The **relational data model** represents a database as a collection of tables, where each table can be stored as a separate file. Most relational databases use the high-level query language called SQL and support a limited form of user views.

The **object data model** defines a database in terms of objects, their properties, and their operations. Objects with the same structure and behavior belong to a class, and classes are organized into hierarchies (or acyclic graphs). The operations of each class are specified in terms of predefined procedures called methods. Relational DBMS have been extending their models to incorporate object database concepts and other capabilities; these systems are referred to as object-relational or extended relational systems.

The **XML model** has emerged as a standard for exchanging data over the Web, and has been used as a basis for implementing several prototype native XML systems. XML uses hierarchical tree structures. It combines database concepts with concepts from document representation models. Data is represented as elements; with the use of tags, data can be nested to create complex hierarchical structures. This model conceptually resembles the object model but uses different terminology. XML capabilities have been added to many commercial DBMS products.

Two older, historically important data models, now known as legacy data models, are the network and hierarchical models. The **network model** represents data as record types organized as a graph. The **hierarchical model** represents data record types organized in a tree structures. Each hierarchy represents a number of related records.

Review Questions

1. Give the definition of data model, database schema and database state. (6)

2. Describe the three-schema architecture. What is data dependence? Distinguish between logical data independence and physical data independence. (4 + 2 + 2)

3. Describe the concept of internal schema, conceptual schema and external schema. It is more difficult to achieve logical data independence than physical data independence – justify. (3 + 2)

4. What is a database language? What is meant by data definition language and data manipulation language? What s a query language? (2 + 4 + 2)

5. What is client-server architecture? What do you mean by three-tier and four-tier client-server architecture? (2 + 3)

6. What is the difference between procedural and nonprocedural data manipulation languages? (3)

7. Discuss the different types of user-friendly interfaces and the types of users who typically use each. (5)

8. With what other computer system software does a DBMS interact? (3)

9. What is the difference between the two-tier and three-tier client/server architectures? (3)

10. Describe how DBMS software may be classified. (5)

11. Describe different modules of a DBMS software. (6)