Serving Education for Last 40 Years

NEERAJ®

MCS-207

Database Management Systems

Chapter Wise Reference Book
Including Many Solved Sample Papers

Based on

I.G.N.O.U.

& Various Central, State & Other Open Universities

By: *Anand Prakash Srivastava*

NEERAJ PUBLICATIONS
(Publishers of Educational Books)

Mob.: 8510009872, 8510009878   E-mail: info@neerajbooks.com
Website: www.neerajbooks.com

MRP ₹ 280/-

Content

DATABASE MANAGEMENT SYSTEMS

QUESTION PAPER

June – 2024

(Solved)

DATABASE MANAGEMENT SYSTEMS **MCS-207**

Time: 3 Hours] *[Maximum Marks : 100*
Weightage : 70%

Note: *Question No. 1 is compulsory. Attempt any three questions from Question No. 2 to Question No. 5.*

Q. 1. ***(a)*** **Compare and contrast the traditional file based system with database approach.**

Ans. Ref.: See Chapter-1, Page No. 7, Q. No. 3.

(b) **Explain the desirable properties of decomposition of a relation with the help of an example.**

Ans. Ref.: See Chapter-5, Page No. 45, 'Desirable Properties of Decomposition'.

(c) **What is query processing ? List the basic steps in query processing.**

Ans. Ref.: See Chapter-12, Page No. 97-98, 'Query Processing: An Introduction' and Page No. 102-103, Q. No. 1.

(d) **What are the limitations of Relational Databases? Explain the need for Object Oriented Databases.**

Ans. Ref.: See Chapter-13, Page No. 109, 'Limitations of Relational Databases' and 'The Need for Object-Oriented Databases'.

(e) **Explain the following basic relational operations with the help of an appropriate examples for each:**

(i) **SELECTION**

(ii) **PROJECTION**

(iii) **CARTESIAN PRODUCT**

(iv) **JOIN**

Ans. Ref.: See Chapter-2, Page No. 14, 'Relational Algebra'.

(f) **Design an ER-diagram for the specifications to maintain any *Hospital*. Clearly indicate the entities, attributes, primary key, constraints, relationships and cardinality.**

Ans. Hospital Management System ER Diagram Design

Entities and Attributes

1. Patient
- *Attributes*: Patient_ID (PK), Name, Gender, Date_of_Birth, Contact_Info, Address, Blood_Type, Emergency_Contact

2. Doctor
- *Attributes*: Doctor_ID (PK), Name, Specialty, Phone, E-mail, Department_ID (FK)

3. Department
- *Attributes*: Department_ID (PK), Name, Location, Head_Doctor_ID (FK)

4. Appointment
- *Attributes*: Appointment_ID (PK), Patient_ID (FK), Doctor_ID (FK), Appointment_Date, Appointment_Time, Reason, Status

5. Medical_Record
- *Attributes*: Record_ID (PK), Patient_ID (FK), Doctor_ID (FK), Diagnosis, Treatment, Record_Date, Notes

6. Medication
- *Attributes*: Medication_ID (PK), Name, Dosage, Frequency, Side_Effects

7. Prescription
- *Attributes*: Prescription_ID (PK), Record_ID (FK), Medication_ID (FK), Quantity, Instructions

8. Room
- *Attributes*: Room_ID (PK), Room_Number, Type (e.g., ICU, General), Capacity, Availability_Status

9. Admission
- *Attributes*: Admission_ID (PK), Patient_ID (FK), Room_ID (FK), Admission_Date, Discharge_Date, Reason

10. Billing
- *Attributes*: Bill_ID (PK), Patient_ID (FK), Admission_ID (FK), Total_Amount, Payment_Status, Billing_Date

11. Staff
- *Attributes*: Staff_ID (PK), Name, Role (e.g., Nurse, Technician), Department_ID (FK), Contact_Info

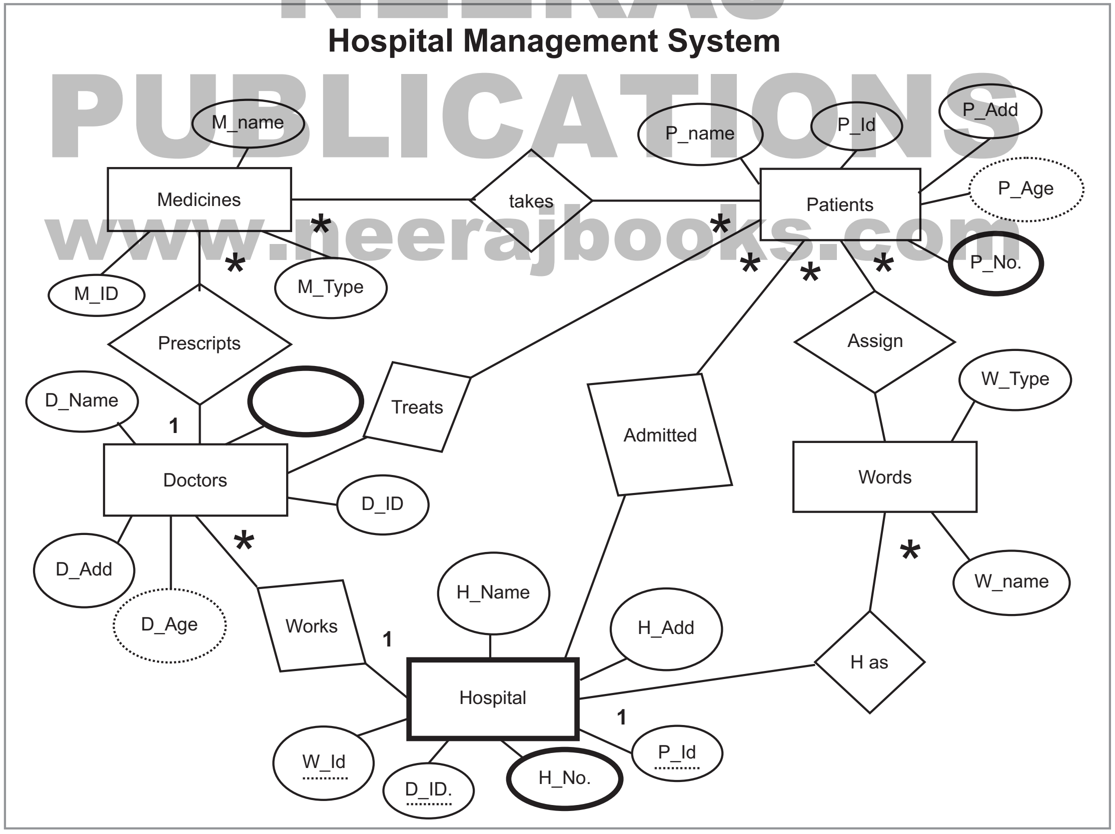
Relationships and Cardinalities

- **Patient ↔ Appointment**
- A patient can have multiple appointments.
- An appointment is associated with one patient.
- *Cardinality*: Patient (1) — (0..*) Appointment
- **Doctor ↔ Appointment**
- A doctor can have multiple appointments.
- An appointment is with one doctor.
- *Cardinality*: Doctor (1) — (0..*) Appointment
- **Patient ↔ Medical_Record**
- A patient can have multiple medical records.
- Each medical record pertains to one patient.
- *Cardinality*: Patient (1) — (0..*) Medical_Record
- **Doctor ↔ Medical_Record**
- A doctor can create multiple medical records.
- Each medical record is created by one doctor.

- *Cardinality*: Doctor (1) — (0..*) Medical_Record
- **Medical_Record ↔ Prescription**
- A medical record can have multiple prescriptions.
- Each prescription is linked to one medical record.
- *Cardinality*: Medical_Record (1) — (0..*) Prescription
- **Prescription ↔ Medication**
- A prescription includes one medication.
- A medication can be prescribed in multiple prescriptions.
- *Cardinality*: Medication (1) — (0..*) Prescription
- **Patient ↔ Admission**
- A patient can have multiple admissions.
- Each admission is for one patient.
- *Cardinality*: Patient (1) — (0..*) Admission
- **Room ↔ Admission**
- A room can be assigned to multiple admissions over time.
- Each admission is assigned to one room.
- *Cardinality*: Room (1) — (0..*) Admission
- **Admission ↔ Billing**
- Each admission has one billing record.
- Each billing record corresponds to one admission.
- *Cardinality*: Admission (1) — (1) Billing
- **Department ↔ Doctor**
- A department can have multiple doctors.
- Each doctor belongs to one department.
- *Cardinality*: Department (1) — (0..*) Doctor
- **Department ↔ Staff**
- A department can have multiple staff members.
- Each staff member belongs to one department.
- *Cardinality*: Department (1) — (0..*) Staff

Constraints

- **Primary Keys (PK):** Uniquely identify each record within an entity.
- **Foreign Keys (FK):** Ensure referential integrity between related entities.
- **Unique Constraints:** Attributes like Room_ Number should be unique within the Room entity.
- **Not Null Constraints:** Essential attributes such as Patient_ID, Doctor_ID, and Admission_Date should not be null.
- **Check Constraints:** For attributes like Availability_Status in Room, ensure values are within a predefined set (e.g., 'Available', 'Occupied').



Q. 2. ***(a)*** **Explain the physical DBMS architecture with the help of a neat diagram.**

Ans. Ref.: See Chapter-1, Page No. 3, 'Physical DBMS Structure'.

DATABASE MANAGEMENT SYSTEMS

Database Management System – An Introduction

1

INTRODUCTION

Now-a-days, most of our online activities involve interacting with a database system that functions as the backend of an application. Whether you're shopping on an e-commerce website, making a transaction at a bank, booking a hotel, flight, or train ticket, accessing a digital library, subscribing to a magazine, or using smartphone apps to buy products – all these actions involve database access. These systems are commonly known as **Traditional Database Applications**. Typically, they handle and store data in the form of text or numbers.

However, with technological advancements over the past few decades, newer and more advanced database models have emerged.

CHAPTER AT A GLANCE

NEED FOR A DATABASE MANAGEMENT SYSTEM

A database is an organized, persistent collection of an organization's data, managed by a Database Management System (DBMS). Before databases, file-based systems were commonly used. To understand the advantages of DBMS, it's important to first look at the limitations of file-based systems, which we'll discuss further.

File Based System

Computerized file-based systems digitize traditional filing methods for storing records like projects or bank statements. While effective for storing large amounts of data, they struggle with processing and cross-referencing. For instance, in a university setup, answering complex queries – such as total fees by Computer Science students, bus facility usage, faculty-student teaching records, year-wise pass-out comparisons, or cross-department course enrollment-becomes difficult and inefficient.

Limitations of File Based System

File-based systems require opening multiple files for a single application. These files often contain duplicate data, leading to various drawbacks. Some of the key shortcomings are discussed below:

Data Isolation: File systems store data in separate files for different applications, making it hard to access or share data across systems. With many files, finding relevant data becomes difficult.

Data Duplication: Different applications may store similar data, leading to duplication. This wastes storage and can cause inconsistencies, such as storing student addresses in both student and bus list files.

Inconsistent Data: When multiple people update data independently or enter incorrect data, inconsistencies arise. For example, a student's new address updated only in one file leads to mismatched records.

Data Dependence: File systems require predefined data structures in application code. Any change in structure means modifying and retesting all related programs, making updates complex and time-consuming.

Incompatible File Formats: Files created by different programming languages (e.g., COBOL *vs.* C) have incompatible formats. Converting these files for joint processing is costly and time-consuming.

Fixed Queries: File-based systems rely on application-specific programs. New queries or reports require writing new programs, making it difficult to adapt to changing needs.

The Database Approach

To overcome the weaknesses of file-based systems, the **database approach** was introduced. It separates data from application programs and allows integrated, secure data sharing through a **Database Management System (DBMS)**. Key features include:

- Centralised or distributed data storage
- Managed by DBMS
- Metadata stored in a data dictionary

- Integrated and securely shared data
- Supports operations like database creation, data insertion/editing, enforcing security, and controlled access

Reduction of Redundancies: In database systems, data is integrated and stored centrally or with controlled redundancy. This removes duplicate data, reducing inconsistencies that were common in file-based systems with multiple copies of the same data.

Sharing of Data: Databases store integrated data for the entire organisation. DBMS controls secure data sharing, allowing authorised users or applications to access only relevant parts. New applications can also be added and share data as needed.

Data Independence: Unlike file-based systems, DBMS separates data descriptions from application programs. This means changes in data structure do not affect programs, offering data independence and a consistent user interface.

Improved Integrity: DBMS enforces rules and constraints to ensure data is valid and consistent. For example, an age field can be restricted between 18 and 70.

Efficient Data Access: DBMS uses advanced techniques to store and retrieve data efficiently, even for unexpected queries, ensuring quick and smooth data access.

User Interfaces as per Users' Technical Knowledge: DBMS provides different interfaces based on user expertise, such as:

- Menu-driven Forms and Report Interfaces
- Application Programming Interface (API)
- Query Language Interface
- Natural Language Interface

Representing Relationship among Data: DBMS maintains relationships among related data objects within integrated organizational data, making it easy to access connected information.

Improved Security: DBMS ensures only authorised users can access specific data. A Database Administrator (DBA) manages access rights, limiting operations like view, update, or delete based on user roles, enhancing data protection.

Improved Backup and Recovery: Unlike file systems, DBMS supports automatic backup and recovery mechanisms to protect data from system or hardware failures.

Support for Concurrent Transactions: DBMS allows multiple transactions to occur simultaneously without interference. For example, multiple banking operations can run at once while maintaining data accuracy and integrity.

THE LOGICAL DBMS ARCHITECTURE

Most advantages of database systems come from the DBMS software, which is complex due to the need to manage large volumes of data reliably. This section introduces **DBMS Architecture**, explaining how it functions. Two types of architectures are discussed:

- **Logical Architecture:** Focuses on how data is organised and accessed at different logical levels.
- **Physical Architecture:** Describes the various components of the DBMS software that manage data operations.

Three Level Database Architecture

The **Three-level Database Architecture** defines different levels of data abstraction for various users. Standardised by ANSI, it is also known as the **ANSI/SPARC architecture**. This model visualises the database schema at three levels, as shown in Figure 3. These levels are explained below.

The External or View Level: This level defines how users view the data based on their access rights. Different users can have different views, hiding the overall database structure. It also maps external records to the conceptual view, ensuring user-specific data access.

The Conceptual Level: This level defines the structure, relationships, and constraints of the database. It includes data objects, their relationships, and access controls. Defined using Data Definition Language (DDL), the conceptual schema is usually created by the database or system administrator.

The Internal or Physical Level: This level manages how data is physically stored, including data files, metadata files, and access structure files. It defines storage structures and access methods, using DDL. All files are controlled by the DBMS and the database administrator to ensure performance and security.

Mappings Between the Three Levels and its Relationship with Data Independence

The three-level architecture maps data across different levels, enabling **data independence**, which was lacking in file-based systems.

The **first mapping** is between the **conceptual and external levels**. External views are derived from the conceptual schema, so changes in the conceptual level (e.g., adding a *middlename* field) do not affect user programs – only the mapping needs updating. This is known as **logical data independence**.

The **second mapping** is between the **conceptual and physical levels**, allowing changes in storage structure or indexing (e.g., changing file organisation) without affecting the conceptual schema. This is called **physical data independence**.

Objective of The Three-Level Architecture

The **Three-level Architecture** separates the data seen by users from how it is physically stored in the database. Its main objectives are:

- **Support for multiple user views:** Each user can have a different view of the data. When an application changes, only the related views need to be updated. This ensures **data and program independence**.
- **Higher Data Abstraction:** Since application programs don't interact with the physical file organisation, they operate at a higher abstraction level. Users and applications don't need to manage the **Conceptual or Physical Schemas** – this is handled by the **Database Administrator (DBA)** using Data Definition Language (DDL).

PHYSICAL DBMS STRUCTURE

A **Database Management System (DBMS)** is a complex software that handles various critical aspects of database management, including query processing, data integrity, security, file access, and performance optimization. **Figure** illustrates the key software components that are logically present in most modern DBMSs.

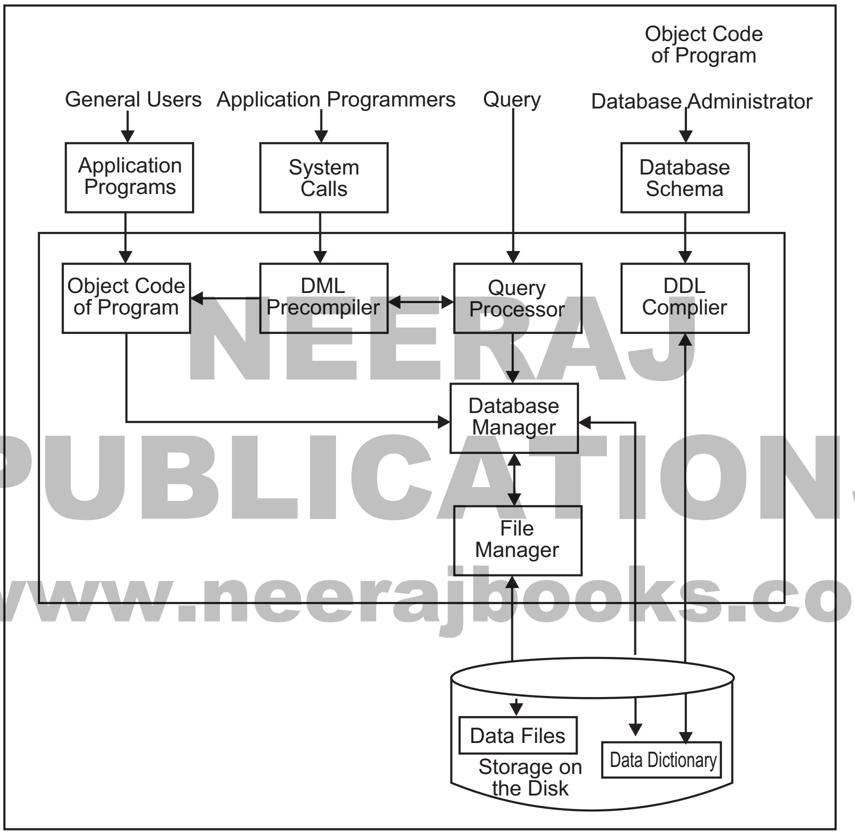


Fig.: DBMS Structure

DML Precompiler

A **DBMS** includes several languages for defining and manipulating data. **Data Definition Language (DDL)** defines data structures, types, constraints, and access rights. **Data Manipulation Language (DML)** is used for inserting, modifying, and retrieving data. A **DML precompiler** translates DML commands in application programs into executable procedure calls and works with the **query processor** to handle user queries.

DDL Compiler

A **DDL compiler** translates DDL statements into metadata tables that define data structures, constraints, and access rights. These metadata tables are stored in the **data dictionary** or **system catalog** and are used by other DBMS components to manage, protect, and ensure data integrity.

File Manager

All database tables and metadata are stored as files managed by the operating system. However, since OS file systems have limited features for DBMS needs, the **File Manager** in the DBMS tracks all data, index, and related files. Actual input/output operations are handled by the operating system.

Database Manager

The Database Manager is a key DBMS component that handles user and application requests for data access or manipulation. It works with the **data dictionary** and **file manager**, enforces **security and integrity constraints**, manages **memory access**, supports **concurrent transactions**, and ensures **data recovery** in case of failure.

Key Roles of Database Manager:

Collaborating with File Manager: Controls file operations in coordination with the OS through the file manager.

Integrity Enforcement: Enforces data correctness using constraints stored in the data dictionary.

Security Enforcement: Restricts data access to authorised users only, based on permissions.

Backup and Recovery: Protects data during failures like hardware or software issues, ensuring data remains consistent.

Concurrency Control: Maintains data consistency during simultaneous transactions.

Software Components of Database Manager:

Authorisation Control: Verifies user credentials for access.

Command Processor: Converts user commands into executable code.

Integrity Checker: Ensures inserted or modified data follows defined constraints.

Query Optimizer: Enhances query processing efficiency.

Transaction Manager: Validates transaction permissions.

Scheduler: Manages order of concurrent transactions to maintain consistency.

Recovery Manager: Restores database to a consistent state after failure.

Buffer Manager: Manages data transfer between disk and main memory efficiently (also called cache manager).

Query Processor

Every DBMS includes a **query language** for database creation, manipulation, retrieval, and control. Typically a high-level 4th generation language, it is translated and optimized for efficient execution.

The **query language processor** has *two* main components:

Parser: Checks query syntax, converts it into tasks.

Query Optimiser: Generates multiple query execution plans, estimates response times (data access, execution, communication), and selects the most efficient plan for execution.

Database Administrator

The **Database Administrator (DBA)** manages all aspects of the database, including its structure, access, performance, security, and recovery.

Key functions of a DBA:

- Defines database schema at all three levels using DDL
- Specifies file organisation and access methods (e.g., indexes)
- Updates schema, file structure, or indexes as needed
- Manages user authorisations for data access and modification
- Sets and enforces integrity constraints on the database

Data Files, Indices and Data Dictionary

Data is stored in data files, which may be standard or encrypted and are usually not directly accessible to users. Index files store indices that allow fast data retrieval. For example, a book database may be ordered by accession number but indexed by author name or title for quicker searches.

Data Dictionary: The data dictionary stores **metadata –** information about the structure and properties of the database. It helps guide users in accessing data efficiently and includes:

- Schema definitions at internal, conceptual, and external levels
  - Table definitions, attributes, data types, keys, and constraints
  - Relationships (e.g., foreign keys), and access authorisations
- User and application permissions
- Database statistics (e.g., record counts, value frequencies)

DATABASE SYSTEM ARCHITECTURES

Initially, databases were **centralised**, stored on a single machine for one or multiple users. Today, most systems are **client-server** based, where powerful **backend servers** manage data, security, and access, and **frontend applications** allow user interaction. Frontends may be developed by vendors or third parties using APIs.

Interfaces

- **Command Line Interface:** Allows expert users (e.g., DBAs) to write DDL, DML, and other commands.
- **Graphical User Interface (GUI):** Menu-driven, user-friendly interface for easier interaction, widely used today.

Utilities

- **Backup/Restore:** Periodically saves the database state for recovery after failures.
- **Load/Unload of Data:** Moves data between systems during hardware changes, upgrades, or data migration.
- **Reporting/Analysis:** Generates reports and visual data analysis to support decision-making in organisations.