CHAPTER 5

DATABASE MANAGEMENT SYSTEMS

Data is one of the most valuable assets. At the heart of this lies the **DBMS** — a powerful software tool to handle vast amounts of structured and unstructured data with **precision, security and scalability**. Covers DBMS **components**, the **ACID** properties, **data independence** and the **three-level architecture** (external, conceptual, internal).

CHAPTER ROADMAP

1. Components of a DBMS — 5
2. Database characteristics — ACID
3. Additional characteristics — 6
4. Types of data independence — 2
5. DBMS architecture — 3 levels

DATABASE

An organised collection of data that is stored and accessed electronically.

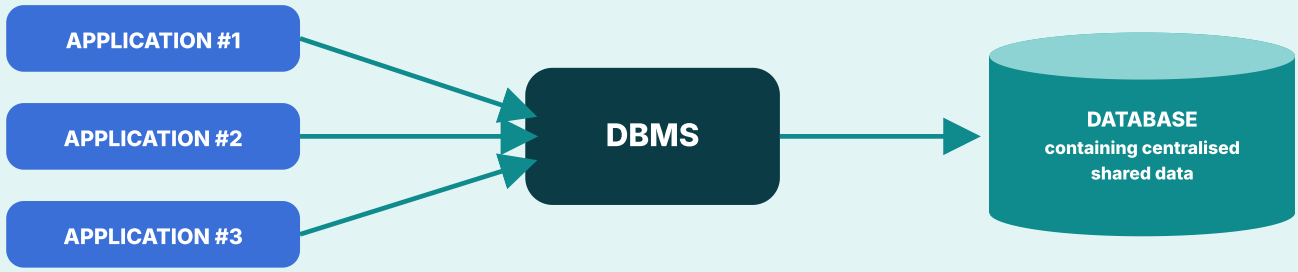
Lets users **store, manage and retrieve** large volumes of structured or unstructured information efficiently; designed so data can be easily **searched, manipulated and updated**.

DBMS

Software that enables the creation, management and use of databases.

Helps **organise, retrieve and secure** data. Like an **operating system manages hardware resources**, a DBMS manages **data resources** — letting **multiple users** interact **securely and concurrently**.

Structure of Database Management



1 COMPONENTS OF A DBMS

work together to manage databases

Database Engine
Storing, retrieving and processing data.

Query Processor
Interprets & executes **SQL** queries, **optimising** them for performance.

Transaction Manager
Processes transactions per **ACID**; protects integrity even during **system failures**.

Storage Manager
Physical storage — allocates disk space, efficient retrieval.

Security Manager
Authentication & authorisation — only authorised users access data.

MEMORY HOOK **"Every Query Takes Some Security"** → **E**ngine · **Q**uery processor · **T**ransaction manager · **S**torage manager · **S**ecurity manager.

2 DATABASE CHARACTERISTICS — THE ACID MODEL

The ACID model defines the **four critical properties of database transactions**. They keep the database in a **consistent state before and after** a transaction — even with **errors, crashes or other failures** — maintaining the **integrity and reliability** of data.

A ATOMICITY
all or nothing

All operations in a transaction succeed — **or none do**. A transaction is an **indivisible unit of work**; if any operation fails, the **entire transaction is rolled back** to the previous state.

Bank transfer $100 A → B
① Deduct $100 from A ② Add $100 to B. If ① succeeds but ② fails (e.g., network issue) → **rolled back**; A is not deducted — **no partial deduction**.

C CONSISTENCY
valid state → valid state

Database moves from **one valid state to another**, maintaining **integrity constraints** (foreign keys, unique keys, business rules). A transaction that violates them is **not committed**.

Rule: balance cannot be negative
Withdraw $500 from an account with $400 → transaction **fails**; database stays consistent; business rule enforced.

I ISOLATION
no interference

Concurrent transactions run **without inconsistencies**; each is isolated until the first completes — behaves **as if it were the only transaction**.

T1: transfer $200 A → B · T2: check B's balance
T2 must **not see the intermediate balance** of B while T1 is running; it sees only the **final** balance after T1 completes.

D DURABILITY
committed = permanent

Once **committed**, effects are **permanently saved** — even after a **system crash, power failure or hardware malfunction**; recoverable if the system fails.

Crash right after commit
The $100 transfer still shows in both A and B when the system is restored.

How durability is achieved

Write-ahead logging
Changes are **first written to a log** before being applied to the database.

Replication
Data is **copied to multiple locations**.

One-line summary

A	all operations or none
C	rules & constraints always hold
I	concurrent transactions don't interfere
D	committed changes survive failures

WHO ENFORCES? The **Transaction Manager** component ensures transactions follow the ACID properties.

3 ADDITIONAL CHARACTERISTICS OF DBMS

beyond ACID

Relation-Based Tables
Data in **relational tables** (rows & columns) → easier to manage & query.

Multiuser Access
Concurrent access by multiple users — collaboration without data conflicts.

Consistency & Integrity
Integrity constraints like **primary keys and foreign keys**.

Security
Role-based access controls and **encryption** safeguard sensitive data.

Query Language
Most DBMS support **SQL** for querying & managing data.

Data Independence
Changes in the **data schema** do not affect the **application layer**.

4 TYPES OF DATA INDEPENDENCE

separate physical storage from logical structure

A fundamental DBMS property: **changes made at one level do not affect other levels → flexibility, efficiency, scalability**; simplifies database management & maintenance by **isolating the effects of changes**.

4.1 Logical Data Independence

change the conceptual schema

Change the **logical (conceptual) schema** — tables, views, relationships, constraints — **without affecting the external schema or application programs**.

Ex 1 · Adding a column
Add **DateOfJoining** to the employee table; an app fetching only EmployeeID & EmployeeName keeps working.

Ex 2 · Splitting a table
Normalise a combined table into **Customers** and **Orders**; **views** maintain backward compatibility.

Flexibility | Reduced costs | Backward compatibility

4.2 Physical Data Independence

change the storage structure

Change the **physical storage structure** — file formats, data structures, **indexing**, storage locations — **without affecting the logical schema or applications**.

Ex 1 · Changing storage location
Move files from local server → cloud, or hard drive → **SSD**; invisible to users.

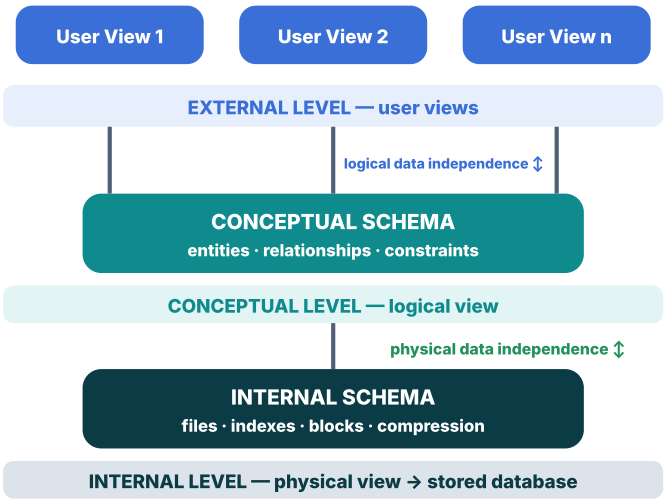
Ex 2 · Adding / modifying indexes
Index on **LastName** → faster searches; same results, no code change.

Performance tuning | Improved scalability | Cost efficiency

Feature	Logical Data Independence	Physical Data Independence
Level	Logical (conceptual schema changes)	Physical (internal storage structure changes)
Impact on apps	None when logical structure is modified	None when physical storage is modified
Example	Adding/removing columns, splitting tables	Changing file location, adding indexes, compressing data
Scope	**Structure** of data (tables, relationships)	**Storage** of data (on disk / in memory)
Use case	Modify schema for new features / better design	Improve performance & storage management

5 DBMS ARCHITECTURE — THREE-LEVEL SCHEMA

ANSI/SPARC · 1970s



Purpose: abstract data at different levels so users interact with data **without knowing how it is physically stored**. Proposed by **ANSI/SPARC** in the **1970s**; now the standard way to describe most modern DBMS.

Keeps the **users' view**, the **logical structure** and the **physical storage** separate → supports **data independence** (changes at one level don't affect others).

External	highest abstraction · user view
Conceptual	logical view · whole database
Internal	lowest level · physical view

5.1 EXTERNAL

User View — **highest** level of abstraction; how data is perceived by individual users / groups.

Each user gets a **customised view** showing only relevant data; **views** are logical representations that can hide parts of the database.

- **Customised views** by role & privileges
- **Security** — hide passwords, salaries
- **Simplification** — no need to know complexity

Same database, two views
HR: name, employee ID, salary.
Finance: revenue, expenditure, profit — **salaries restricted**.

5.2 CONCEPTUAL

Logical View — overall **logical structure of the entire database**; defines the schema.

Defines **entities** (tables), **relationships** (one-to-many, many-to-many) and **constraints** (primary/foreign keys, integrity rules). Hides physical storage.

- **Entity-Relationship model**
- **Data integrity & constraints** — data types, foreign keys, unique
- **Independence from physical storage**

Customer Order System
Entities: Customers, Orders, Products.
• Each Customer can place **multiple Orders**
• Each Order contains **one or more Products**
— defined logically, not how stored.

5.3 INTERNAL

Physical View — **lowest** level; how data is actually stored on disk / devices.

File structures, indexes, data blocks, storage allocation — for **retrieval speed, space and performance**. Hidden from users and the conceptual level.

- **Data storage techniques** — blocks / pages
- **Indexing** — find records without full scan
- **Data compression** · **Physical data access** (buffer mgmt)

Fetching a customer's orders
Data spread in blocks across the disk, managed by **indexes**; the internal level handles how it's accessed & fetched.

Key Benefits of the Three-Level Architecture

① Data Independence
Logical: change tables without affecting users.
Physical: reorganise files / indexes without affecting schema or views.

② Flexibility
Customised views per department — useful in **large organisations**.

③ Security
Access controlled through the **external level** by role & permissions.

④ Simplified Management
DBAs optimise physically **without disrupting** users.

EXAM FOCUS · MCQ TRAPS

commonly confused points

1 Database vs DBMS
Database = the **organised collection of data**. DBMS = the **software** that creates, manages and uses it.

2 Query vs Transaction manager
Query processor = **SQL** interpret, execute, optimise. Transaction manager = **ACID**.

3 Storage vs Security manager
Storage = **disk space & physical storage**. Security = **authentication & authorisation**.

4 Atomicity vs Consistency
Partial transfer rolled back = **Atomicity**. Negative balance blocked by a rule = **Consistency**.

5 Isolation vs Durability
Not seeing an intermediate balance = **Isolation**. Surviving a crash after commit = **Durability**.

6 Write-ahead logging
Technique for **Durability** (with replication) — not atomicity.

7 Logical vs Physical independence
New column / split table = **Logical**. New index / move to SSD or cloud / compression = **Physical**.

8 Highest vs lowest level
External = highest abstraction (user view). **Internal** = lowest (physical). Conceptual sits between.

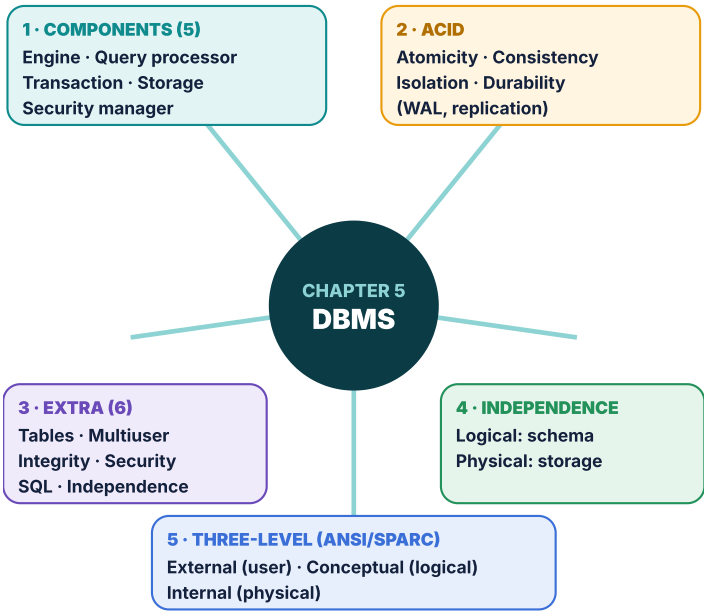
9 Where are views & ER models?
Views → **External**. Entities, relationships, constraints (ER model) → **Conceptual**. Indexing, compression → **Internal**.

10 Origin of the model
Three-level architecture proposed by **ANSI/SPARC** in the **1970s**.

CLASSIFY IT

1	Transfer fails midway → rolled back	Atomicity
2	$500 withdrawal from $400 rejected	Consistency
3	Balance check waits for transfer	Isolation
4	Committed transfer survives crash	Durability
5	Adding DateOfJoining column	Logical DI
6	Index on LastName	Physical DI
7	Finance can't see salaries	External level
8	"Customer places many orders"	Conceptual level
9	Data blocks & compression	Internal level

REVISION MAP



DBMS
Software to create, manage & use databases — like an OS for data.

5 Components
Engine, query, transaction, storage, security.

ACID
All-or-none, valid states, no interference, permanent.

Independence
Logical = schema; physical = storage.

3 Levels
External → Conceptual → Internal.