Database Management System (CSC260)

B.Sc. CSIT, IV Semester

Prepared By:

Ms. Sarita Neupane

Senior Lecturer

Kathford International College of Engineering and Management

Email: sarita.neupane@kathford.edu.np

Data

Data is a collection of raw, unorganized facts or values that do not carry any specific meaning by themselves. It could be numbers, characters, symbols, or images that are yet to be processed.

Example:

- ✓ `85`, `92`, `78`, `90` — just a list of numbers
- ✓ `"Ram"`, `"CS260"`, `"A+"` — individual values

These do not tell us much **until they are given context**.

Information

Information is processed or organized data that is meaningful and useful for decision-making. When data is interpreted, it becomes information.

Example:

- ✓ "Ram scored 85 in the DBMS midterm exam."
- ✓ "The average score of students in course CS260 is 86.25."

Here, the **context** and **meaning** of the data make it informative.

Data and Information

Aspect	Data	Information
Nature	Raw facts	Processed, meaningful data
Meaning	Alone has no meaning	Carries meaning
Example	`78`, `92`, `"DBMS"`	`"DBMS average score is 85"`
Usage	Input for processing	Output of processing
Dependence	Independent of context	Dependent on context and relevance

---

Database

A **database** is an organized collection of interrelated data that can be easily accessed, managed, and updated.

Example:

- ✓ A **library database** stores details about books, authors, and borrowers.
  - **Books Table**: BookID, Title, Author, ISBN, Availability
  - **Borrowers Table**: UserID, Name, BookBorrowed, DueDate

- ✓ A **Kathford College database** stores student records, Faculty records etc:
  - **Students Table**: StudentID, Name, Program, GPA
  - **Faculty Table:** FacultyID, Name, Salary, Address etc.

Database Management System (DBMS)

A **Database Management System (DBMS)** is software that interacts with users, applications, and the database itself to define, store, retrieve, and manage data in a database.

In other word, it is the combination of Database and application program to manage those data.

Key Functions of a DBMS

1. **Data Definition**: Create/modify database structures (tables, relationships).
2. **Data Manipulation**: Insert, update, delete, and query data (e.g., using SQL).
3. **Data Security**: Control user access (e.g., admin, user roles/permissions).
4. **Data Integrity**: Enforce rules (e.g., "No duplicate student IDs").
5. **Concurrency Control**: Manage simultaneous access (e.g., two users booking the same flight seat).
6. **Backup & Recovery**: Restore data after failures (e.g., server crash/system failure).

Examples:

1. Relational Database Management System (RDMS):

Data stored in tables (rows/columns) with relationships.

MYSQL – Most of the social media sites, YouTube and other web applications

Oracle – Most of Bank

PostgreSQL etc.

Example of RDBMS Query:

```
SELECT * FROM Students WHERE grade = 'A';
```

2. NoSQL:
   Handles unstructured/semi-structured data (JSON, documents).

   MangoDB

   Example of MangoDB:

   ```
   db.users.find({ age: { $gt: 30 } });
   ```
3. Graph DBMS:
   Stores data as nodes/edges (e.g., social networks).

   Neo4j: LinkedIn uses it for connection recommendations.

Imagine a **database** as a **library's collection of books** (data), while the **DBMS** is the **librarian and software system** that:

- Organizes the books (data),
- Helps you find a book (query),
- Ensures the books are borrowed and returned properly (transaction),
- Keeps unauthorized people out of the restricted section (security).

Traditional File Processing System

A **Traditional File Processing System** is an early method of storing and managing data where each application uses its own set of files, leading to **data redundancy**, **inconsistency**, and **limited sharing**. Unlike a DBMS, it lacks centralized control, efficient querying, and security features.

Key Characteristics

1. **Decentralized Data**: Each department/application maintains separate files.
2. **No Relationships**: Files are independent (unlike tables in a DBMS).
3. **Program-Dependent**: Changing file structure requires rewriting programs.
4. **Limited Security**: Access control is file-based, not user-based.

Examples of File Processing Systems

1. University Management (Pre-DBMS Era)

- **Files**:
  - `students.txt` (Student IDs, names)
  - `courses.txt` (Course codes, titles)
  - `grades.txt` (Student IDs, course codes, grades)
- **Problems**:
  - **Redundancy**: Student names appear in both `students.txt` and `grades.txt`.
  - **Inconsistency**: If a student's name changes, it must be updated in multiple files.
  - **No Querying**: To find "students with grade A in Math," a custom program must be written.

2. Hospital Records

- **Files**:
  - `patients.txt` (Patient ID, name, diagnosis)
  - `billing.txt` (Patient ID, invoice amount)
- **Problems**:
  - **Data Isolation**: Billing staff can't easily access diagnosis data.

- **Integrity Issues**: Deleting a patient record might leave orphaned billing entries.

3. Library System

- **Files**:
- books.txt (Book ID, title, author)
- borrow.txt (Book ID, borrower name, due date)

Limitation of Traditional File System

Problem	Description
Data Redundancy	Same data stored in multiple files (e.g., student address in both admin and library files).
Data Inconsistency	Changes in one file are not reflected in others.
Limited Data Sharing	Files are application-specific and not accessible by others.
Lack of Data Security	No central access control.
Difficult Data Access	Requires writing custom programs for every query.
No Concurrency Control	Cannot handle multiple users editing data simultaneously.

Why DBMS Evolved?

The **Database Management System (DBMS)** emerged to overcome the critical limitations of **Traditional File Processing Systems**.

OR

The **Database Management System (DBMS)** evolved as a solution to the **limitations of traditional file processing systems**. As data became more complex and the need for sharing, security, and consistency increased, a better approach was required to manage data efficiently and effectively.

For Example:

Before DBMS:

- Every department in a university maintains its own student file.
- Student "A" updates their phone number in the library, but it's outdated in the exam department.

After DBMS:

- A centralized system holds all student information.
- Any update is reflected everywhere instantly and securely.

Characteristics of DBMS:

1. Self-Describing Nature

- **What it means**: The DBMS stores **metadata** (data about data), such as table schemas, constraints, and indexes.
- **Example**:
  - In MySQL, the `INFORMATION_SCHEMA` database stores metadata about tables, columns, and permissions.
  - When you run `DESCRIBE Students;` the DBMS retrieves metadata (column names, data types) about the `Students` table.

2. Program-Data Independence

- **What it means**: Changes in data structure (e.g., adding a column) don't require rewriting application code.
- **Example**:
  - A banking app uses a `Customers` table. If the bank adds a new column (`credit_score`), the app still works without modification.
  - In a file system, changing a file format (e.g., CSV → JSON) breaks existing programs.

3. Support for Multiple Views

- **What it means**: Different users see customized data without accessing the entire database.
- **Example**:
  - In a **hospital DBMS**:
    - **Doctors** see patient medical history.
    - **Billing staff** see insurance details but not diagnoses.

4. Data Sharing & Multi-User Access

- **What it means**: Many users/apps can access data simultaneously without conflicts.
- **Example**:
  - **Airline Reservation System**:
    - Thousands of users book flights concurrently.
    - The DBMS uses **locking** to prevent double-booking the same seat.

5. Controlled Redundancy

- **What it means**: Minimizes duplicate data (unlike file systems) but allows redundancy for performance.
- **Example**:
  - A **university DBMS** stores `student_id` in both `Enrollments` and `Grades` tables for fast joins but doesn't duplicate `student_name`.
  - In a file system, `student_name` might appear in 10+ files.

6. Data Integrity

- **What it means**: Enforces rules to keep data accurate (e.g., "Age cannot be negative").
- **Example**:

- **SQL Constraints**:

```
CREATE TABLE Employees (
    id INT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    age INT CHECK (age >= 18)
);
```

- Rejects invalid data (e.g., `age = -5`).

7. Backup & Recovery

- **What it means**: Automatically logs changes and restores data after failures.
- **Example**:
  - A **bank DBMS** uses transaction logs to undo a failed transfer after a power outage.
  - In a file system, a corrupted `transactions.txt` could mean permanent data loss.

8. Security & Access Control

- **What it means**: Restricts data access based on user roles.
- **Example**:
  - **SQL Permissions**:

```
GRANT SELECT ON Customers TO sales_team;
DENY DELETE ON Employees TO interns;
```

  - In a file system, anyone with file access can edit/delete data.

9. Efficient Query Processing

- **What it means**: Optimizes queries for speed using indexes and execution plans.
- **Example**:
  - A query like:

```
SELECT * FROM Orders WHERE customer_id = 101;
```

    uses an **index** on `customer_id` to fetch results instantly, even with millions of orders.

10. ACID Transactions

- **What it means**: Ensures transactions are **Atomic, Consistent, Isolated, and Durable**.
- **Example**:
  - **Bank Transfer**:
1. Deduct Rs.100 from Account A.
2. Add Rs.100 to Account B.
   - If step 2 fails, the DBMS **rolls back** step 1 (Atomicity).

Advantages of DBMS

1. Reduced Data Redundancy

- **Problem in File Systems**: Same data is stored in multiple files (e.g., `students.txt`, `grades.txt`), leading to wasted space and inconsistency.
- **DBMS Solution**: Stores data in a **centralized** and **normalized** manner.
  - **Example**:
    - In a **hospital DBMS**, a patient's name is stored **once** in the `Patients` table and linked via `patient_id` in other tables (e.g., `Appointments`, `Bills`).

---

2. Improved Data Consistency

- **Problem in File Systems**: Updating data in one file may leave others outdated.
- **DBMS Solution**: Changes propagate automatically.
  - **Example**:
    - In a **banking DBMS**, if a customer updates their phone number in the `Customers` table, all linked tables (e.g., `Accounts`, `Loans`) reflect the change instantly.

---

3. Enhanced Data Security

- **Problem in File Systems**: Limited access control (e.g., anyone with file access can edit data).
- **DBMS Solution**: Role-based permissions (e.g., read-only, read-write).
  - **Example**:

```
GRANT SELECT ON Employees TO hr_team;  -- HR can only view data
DENY DELETE ON Patients TO nurses;     -- Nurses cannot delete records
```

---

4. Efficient Data Retrieval

- **Problem in File Systems**: Slow manual searches (e.g., scanning `inventory.txt` line by line).
- **DBMS Solution**: **SQL queries** and **indexes** for fast searches.
  - **Example**:

```
-- Finds all products under $10 in milliseconds
SELECT * FROM Products WHERE price < 10;
```

---

5. Concurrent Access Control

- **Problem in File Systems**: Only one user can edit a file at a time.
- **DBMS Solution**: **Multi-user support** with transaction locking.
  - **Example**:
    - **Airline Booking**: Thousands of users can book flights simultaneously without double-booking seats.

---

6. Data Integrity Enforcement

- **Problem in File Systems**: No rules to prevent invalid data (e.g., `age = -5`).
- **DBMS Solution**: **Constraints** (e.g., `PRIMARY KEY`, `CHECK`).
  - **Example**:

```
CREATE TABLE Employees (
    id INT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    age INT CHECK (age >= 18)  -- Rejects underage entries
);
```

---

7. Backup & Recovery

- **Problem in File Systems**: Manual backups risk data loss.
- **DBMS Solution**: **Automated logs** and **point-in-time recovery**.
  - **Example**:
    - A **bank DBMS** restores transactions after a server crash using transaction logs.

---

8. Scalability

- **Problem in File Systems**: Struggles with large datasets (e.g., 1M+ records).
- **DBMS Solution**: Handles **big data** via indexing, partitioning, and cloud scaling.
  - **Example**:
    - **Amazon** uses distributed DBMS (e.g., DynamoDB) to manage billions of product listings.

---

9. Reduced Application Development Time

- **Problem in File Systems**: Developers write custom code for data access.

- **DBMS Solution**: **Standardized SQL** simplifies app integration.
  - **Example**:
    - A **food delivery app** uses the same SQL queries (`INSERT`, `SELECT`) across Android/iOS/web.

---

10. Centralized Management

- **Problem in File Systems**: Decentralized data leads to chaos (e.g., finance vs. HR files).
- **DBMS Solution**: **Single point of control** via a database administrator (DBA).
  - **Example**:
    - A **university DBA** manages student data for admissions, exams, and fees in one system.

Actors on the Scene & Workers Behind the Scene in DBMS

A **Database Management System (DBMS)** involves two groups of people:

1. **Actors on the Scene** – Those who directly interact with the database.
2. **Workers Behind the Scene** – Those who maintain and develop the DBMS but rarely interact with end users.

1. Actors on the Scene

These are the **primary users** who interact with the DBMS daily.

A. Database Administrators (DBA)

- **Role**: Manages the DBMS environment.
- **Responsibilities**:
  - Install, upgrade, and configure the DBMS.
  - Ensure **security** (user access, encryption).
  - Optimize performance (indexing, query tuning).
  - Backup and recovery.

- **Example**:
  - A **bank DBA** ensures transaction data is secure and available 24/7.

B. Database Designers

- **Role**: Designs the database structure.
- **Responsibilities**:
  - Create **ER diagrams** and relational schemas.
  - Define tables, relationships, and constraints.
- **Example**:
  - A **hospital database designer** models tables for Patients, Doctors, and Appointments.

C. End Users

- **Role**: Use the database for day-to-day operations.

  Types:
  - ✓ **Casual Users**: Run queries (e.g., managers generating reports).
  - ✓ **Naïve Users**: Use apps (e.g., bank tellers entering transactions).
  - ✓ **Sophisticated Users**: Write complex queries (e.g., data analysts).

- **Example**:
  - A **sales executive** queries the CRM database to check customer orders.

D. System Analysts & Application Programmers

- **Role**: Develop apps that interact with the DBMS.
- **Responsibilities**:
  - Write code (e.g., Python, Java) to **insert/update/delete** data.
  - Optimize SQL queries for performance.
- **Example**:
  - A **Java developer** builds an e-commerce app that fetches product data from MySQL.

Types of Database Users

1. Naive Users (End Users / Parametric Users)

- Use **predefined applications** (forms, GUIs).
- **Do not write queries** or scripts.
- Interact through buttons, drop-downs, and forms.

Examples:

- Bank customers using ATMs
- Students checking results online
- Cashiers at billing counters

---

2. Casual Users

- Use the database **occasionally**.
- Run **ad-hoc queries** to get specific information.
- Some familiarity with **SQL** or reporting tools.

Examples:

- A manager generating monthly sales reports
- A teacher querying students enrolled in a course

---

3. Sophisticated Users

- Write **complex queries**, scripts, or programs.
- Use **advanced database tools** and analytics.
- Have **deep technical knowledge** of DBMS.

Examples:

- Data analysts using SQL for trend analysis
- Engineers writing reports using data mining tools

---

4. Standalone Users

- Use **desktop-based DBMS software** for personal or small-scale use.
- Manage the entire database themselves (design, input, query).
- Not connected to a central server or multi-user system.

Examples:

- A shop owner using MS Access to track inventory
- A student maintaining a personal movie collection in SQLite

Database Administrator (DBA): Roles & Responsibilities

A **Database Administrator (DBA)** is a critical IT professional responsible for managing, securing, and optimizing database systems to ensure high availability, performance, and integrity of organizational data.

A DBA ensures that:

✔ Databases run efficiently and securely.

✔ Data is available when needed.

✔ Database systems comply with business requirements and regulations.

Types of DBAs

Type	Focus Area
System DBA	Manages DBMS installation, upgrades, and server performance.
Application DBA	Optimizes databases for specific apps (e.g., ERP, CRM).
Cloud DBA	Manages cloud-based databases (AWS RDS, Azure SQL).
Analytics DBA	Supports data warehouses & BI tools (e.g., Snowflake, Redshift).

Key Responsibilities of a Database Administrator (DBA)

1. **Database Design & Implementation**
   - Designs schemas, tables, and relationships while optimizing storage and indexing.
2. **Database Security Management**
   - Controls user access, encrypts data, and audits logs to prevent breaches.
3. **Performance Tuning & Optimization**
   - Identifies slow queries, optimizes SQL, and adjusts DB configurations for speed.
4. **Backup & Disaster Recovery**
   - Implements automated backups, replication, and recovery plans to prevent data loss.
5. **Database Maintenance**
   - Upgrades DBMS software, rebuilds indexes, and archives old data.
6. **Troubleshooting & Issue Resolution**
   - Fixes crashes, deadlocks, and corruption using logs and recovery tools.
7. **High Availability & Scalability**
   - Configures clustering, load balancing, and cloud scaling for uninterrupted access.
8. **Compliance & Documentation**
   - Ensures databases meet legal (GDPR, HIPAA) and business policies; maintains documentation.

9. **Monitoring & Alerts**
   - Uses tools (Prometheus, Grafana) to track performance and set up proactive alerts.
10. **Collaboration with Developers**

- Advises on efficient SQL queries, schema changes, and application-DB integration.