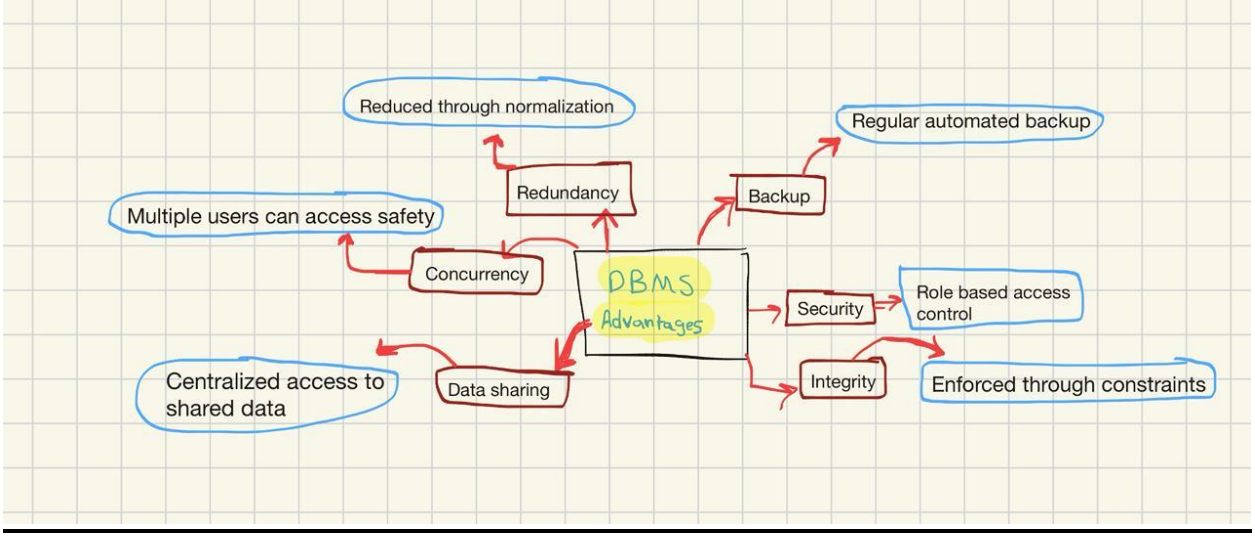
Comparison: Flat File Systems vs. Relational Databases:

Comparison	Flat File Systems	Relational Databases
Structure	Store data in plain text or CSV files where each record is usually independent and lacks a defined schema. There is no enforcement of data types or relationships. [Elsasoft, 2020]	Organize data into structured tables with rows and columns. Each table has a defined schema, and relationships are maintained through keys (primary and foreign keys). [Codd, 1970; Wikipedia, 2024]
Data Redundancy	High data redundancy due to lack of normalization; the same data might be repeated in many places, increasing the chance of inconsistency.	Use normalization techniques to reduce redundancy, ensuring consistent and efficient data storage.
Relationships	Do not inherently support relationships between records; any connection must be manually enforced through program logic or repeated fields.	Support one-to-one, one-to-many, and many-to-many relationships through foreign keys, allowing complex data modeling.
Example Usage	Best for simple, small-scale applications like configuration files, contact lists, or import/export scenarios.	Used in enterprise applications such as ERPs, CRMs, inventory management systems, and banking systems.
Drawbacks	Difficult to scale Prone to data inconsistency Lacks access control and security No built-in query language	Require specialized knowledge to manage and design More resource-intensive May underperform with unstructured or very large datasets

DBMS Advantages:



Roles in a Database System:

1. System Analyst

Gathers and analyzes user requirements to define functional and nonfunctional specifications for database solutions. Typically bridges business needs and technical design.

2. Database Designer (Data Architect / Modeler)

Defines the conceptual, logical, and physical schema using ER diagrams and normalization techniques. Determines data types, relationships, constraints, indexing, partitioning, and availability strategies.

3. Database Developer (Programmer)

Implements database structures in SQL or procedural languages. Designs complex queries, stored procedures, triggers, and performs testing and tuning. Documents database structure and collaborates with application teams to integrate data systems.

4. Database Administrator (DBA)

Responsible for installing, configuring, monitoring, and maintaining the DBMS.

Manages backup and recovery, security and access control, performance tuning, capacity planning, and troubleshooting.

Types include System DBAs, Development DBAs, Application DBAs depending on the focus.

5. Application Developer

Integrates and uses database functionality within applications.

Develops application logic, ensures proper database calls, handles ORM or direct SQL, and maintains interaction patterns.

6. BI (Business Intelligence) Developer / Database Analyst

Designs dashboards, reports, and analytics queries to extract business insights.

Evaluates data quality and structure, provides recommendations for optimizations, and ensures data compliance.

Types of Databases:

Relational Databases (RDBMS):

Data is stored in structured tables (rows and columns) with a defined schema.

Support ACID properties (Atomicity, Consistency, Isolation, Durability) for transactional integrity and strong consistency.

Non-Relational (NoSQL) Databases:

Do not use rigid schemas; support flexible storage models.

Four main types:
- Key-Value
- Document
- Column-Family
- Graph databases

Designed for horizontal scalability, high availability, and big-data use cases, but trade off some consistency (BASE model).

Hierarchical & Network Databases:

Hierarchical: Data is in a tree structure with parent-child relationships (e.g. IBM IMS). Limited flexibility.

Network: More flexible, supports many-to-many relationships; more complex to design.

Object-Oriented Databases:

Store data as objects (combining attributes and behaviors), aligning closely with OOP languages.

Useful for complex data types in multimedia or CAD systems, but less widespread than relational systems.

Cloud Databases:

Hosted and managed on cloud platforms (e.g. AWS, Azure, Google Cloud).

Provide scalability, managed backups, and high availability (DBaaS).

Multi-Model Databases:

Support multiple data models (e.g. relational, document, graph) in one system.

Eliminate the complexity of maintaining separate systems while supporting diverse use cases.

Cloud Storage & Databases:

What is Cloud Storage and its Role in Databases?

Cloud storage refers to remotely hosted storage services that let users store and retrieve files and data over the Internet, often with features like on-demand scaling, redundancy, and cost-effective pay-as-you-go models.
Cloud databases (or DBaaS — Database as a Service) run atop such storage and computing platforms, offering managed instances of relational or NoSQL databases without requiring users to manage the infrastructure

Key Advantages of Cloud-Based Databases:

Scalability & Elasticity:
Compute and storage resources are dynamically scalable to meet changing workloads, supporting features like partitioning, replication, and auto-scaling.

High Availability & Disaster Recovery:
Cloud providers replicate data across regions and automate failover, providing robust availability and disaster recovery capabilities.

Reduced Administrative Overhead:
Routine tasks—such as patching, backups, and capacity planning—are automated by the provider, freeing IT teams to focus on strategic work.

Cost Efficiency & Pay-as-You-Go Model:
Organizations pay based on resource usage rather than upfront infrastructure investment, often leading to lower operational costs.

Challenges and Disadvantages:

Security and Privacy Concerns:
Storing data on third-party infrastructure raises issues around data confidentiality, data breaches, compliance, and user control, with shared infrastructure (multi-tenancy) posing additional risks

Performance Overhead and Latency:
Performance can be impacted by network latency and differences between local SSDs and distributed cloud storage environments, requiring careful optimization.

Vendor Lock-In and Migration Complexity:
Migrating workloads between cloud platforms may be difficult due to proprietary APIs, inconsistent configurations, and issues with portability.

Higher Costs for Certain Workloads:
In some cases, especially for predictable large workloads, costs can exceed those of on-prem systems—particularly when customizing performance or security settings.