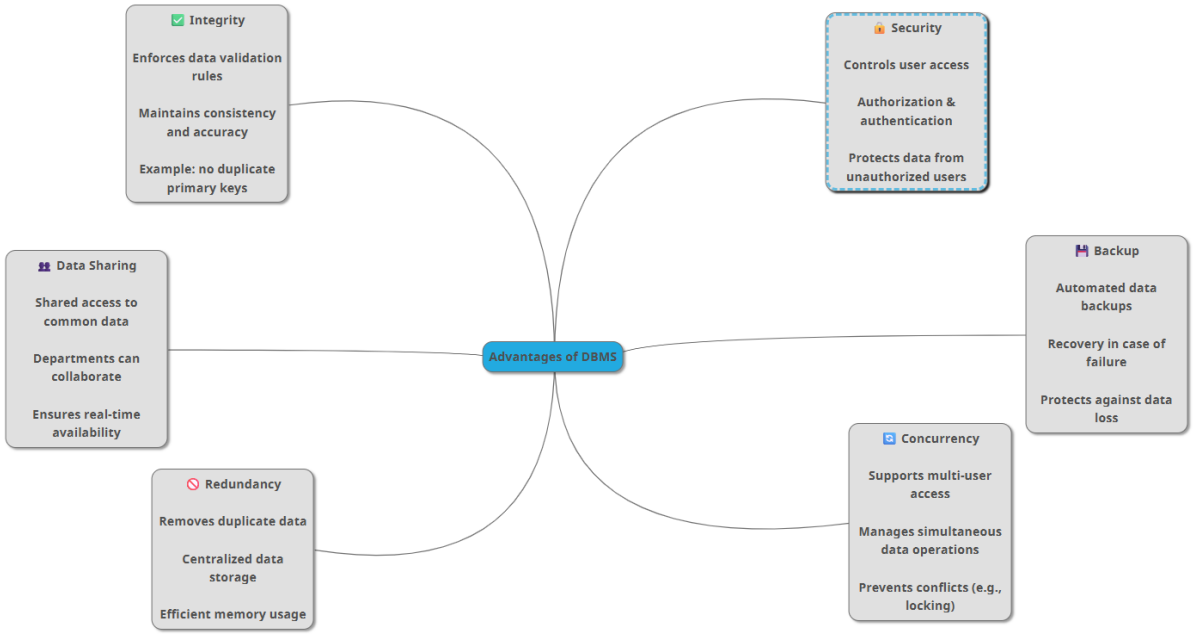
- **Comparison Assessment: Flat File Systems vs. Relational Databases**

Feature	Flat File Systems	Relational Databases (RDBMS)
Structure	Data stored in plain text or binary files (e.g., CSV, TXT). Each file contains a single table with no strict schema.	Data is organized into multiple related tables with a defined schema (columns, types, constraints).
Data Redundancy	High — same data may appear in multiple files, leading to duplication.	Low — uses relationships and normalization to avoid redundancy.
Relationships	all relations must be manually managed through file logic.	Strong — tables are linked via foreign keys, enabling complex relationships.
Example Usage	Simple configurations Log files Small-scale applications	Enterprise applications- E-commerce systems Financial records
Drawbacks	Harder to manage relationships Poor scalability Limited querying and security	Requires a database management system (DBMS) More complex to set up and maintain

- **DBMS Advantages – Mind Map**



- **Roles in a Database System**

📌 1. System Analyst

Role: Acts as a bridge between users and the technical team.
Responsibilities:

A. Gathers and analyzes business requirements.
B. Defines the scope of the database system.
C. Creates specifications and system models.
D. Ensures that the system meets user needs.

🧩 2. Database Designer

Role: Designs the structure and schema of the database
Responsibilities:

A. Defines tables, fields, data types, and relationships.
B. Applies normalization to avoid redundancy.
C. Creates ER diagrams (Entity-Relationship diagrams).
D. Ensures that the design supports performance and scalability.

💻 3. Database Developer

Role: Implements the design by writing database code.
Responsibilities:

A. Creates SQL queries, stored procedures, and triggers.
B. Builds views, indexes, and joins for efficiency.
C. Optimizes database queries and data access logic.
D. Works closely with the application developers.

🛠️ 4. Database Administrator (DBA)

Role: Manages the database system daily.
Responsibilities:

A. Installs and configures the DBMS.
B. Performs backups, restores, and updates.
C. Monitors performance and manages security.
D. Handles user permissions and ensures data integrity.

📱 5. Application Developer

Role: Builds the front-end or business application that uses the database.
Responsibilities:

A. Connects apps to the database using APIs or frameworks.
B. Handles CRUD operations (Create, Read, Update, Delete).
C. Ensures seamless interaction between UI and data.
D. Works with the database developer to test queries.

📊 6. BI (Business Intelligence) Developer

Role: Transforms data into insights for decision-makers
Responsibilities:

A. Designs dashboards and reports using tools like Power BI, Tableau.
B. Builds data pipelines and integrates multiple data sources.
C. Writes complex queries to extract meaningful trends.
D. Supports data-driven decision-making.

📂 Relational vs. Non-Relational Databases

Feature	Relational Database	Non-Relational Database
Structure	Stores data in tables (rows & columns)	Stores data in documents, key-value pairs, graphs, or wide-column stores
Schema	Fixed schema (predefined structure)	Flexible schema (dynamic structure)
Relationships	Uses foreign keys to link tables	Usually, no strict relationships
Query Language	SQL (Structured Query Language)	NoSQL (varies: MongoDB uses queries in JSON-like syntax)
Examples	MySQL, PostgreSQL, Oracle	MongoDB (document), Cassandra (wide-column), Redis (key-value)

🔧 Use Cases:

- **Relational DBs:** Banking systems, ERP, inventory management — where consistency and structured data matter.
- **Non-Relational DBs:** Real-time analytics, social media apps, IoT — where flexibility and scalability are critical.

🏢 2. Centralized vs. Distributed vs. Cloud Databases

Type	Description	Example	Use Case Example
Centralized	All data stored in one single location/server	MS Access, SQLite	Small businesses, local inventory systems
Distributed	Data is split/stored across multiple locations	Apache Cassandra, Google Spanner	Global apps, content delivery networks (CDNs)
Cloud	Hosted on cloud platforms with web access	Amazon RDS, Google BigQuery, Azure SQL	Web apps, SaaS products, scalable e-commerce

🔧 Use Cases:

Centralized Databases: Small businesses, single-location inventory systems — where simplicity and local access are preferred.

Distributed Databases: Global applications, content delivery networks (CDNs), large-scale e-commerce — where high availability and performance are essential.

Cloud Databases: Web apps, SaaS platforms, scalable enterprise systems — where flexibility, remote access, and auto-scaling are needed.

☁️ What is Cloud Storage and How Does it Support Databases?

Cloud Storage refers to storing data on remote servers accessed via the internet. These servers are managed by cloud providers like **Amazon (AWS)**, **Microsoft (Azure)**, and **Google Cloud**.

How it supports databases:

- Cloud storage provides the infrastructure (disk space, network access, backups) that databases use.
- Cloud-based databases are built on top of cloud storage, benefiting from its scalability, reliability, and remote access.
- Databases hosted in the cloud (like Amazon RDS or Azure SQL) automatically use cloud storage to save, replicate, and back up data.

✅ Advantages of Cloud-Based Databases

1. **Scalability**
   - Easily scale up/down based on demand.
2. **Cost Efficiency**
   - Pay-as-you-go pricing, no need to invest in physical hardware.
3. **Accessibility**
   - Access your database from anywhere with an internet connection.
4. **Automatic Backups**
   - Built-in tools for regular backup and disaster recovery.
5. **Maintenance-Free**
   - Cloud providers handle updates, patches, and monitoring.
6. **High Availability**
   - Redundant data storage across regions for failover protection.
7. **Security Features**
   - Encryption, role-based access control, and auditing provided by the platform.

⚠️ Disadvantages / Challenges of Cloud-Based Databases

1. **Internet Dependency**
   - You need a reliable internet connection to access the database.
2. **Latency Issues**
   - Applications with high-speed requirements may experience delays.
3. **Data Privacy & Compliance**
   - Sensitive data may raise concerns with regulations like GDPR or HIPAA.
4. **Vendor Lock-In**
   - Switching from one provider to another can be difficult and costly.
5. **Limited Control**
   - Less low-level access to hardware and configuration compared to on-premises setups.