CODELINE
by Rihal

Database Search and Reporting

Name: **Maha Al Yahmadi**

Batch: **Data Science**

Flat File Systems vs. Relational Databases

Feature	Flat File System	Relational Database (RDBMS)
1. Structure	Data is stored in a **single file**, like an Excel sheet or a text file. Each line is a record, and columns are fields (e.g., Name, Age, Email).	Data is stored in **multiple tables** that are **linked together**. Each table has rows (records) and columns (fields), and tables relate to each other using **keys**.
Example	A **CSV file** storing student names and grades:	A **database** with two tables:
	Name, Grade	- Students (ID, Name)
	Alice, A	- Grades (ID, StudentID, Grade)
	Bob, B	Here, StudentID in "Grades" links back to "Students."
2. Data Redundancy (Duplication)	**High redundancy** because the same data may be repeated in different files.	**Low redundancy** because data is **split into tables** and linked.
	Example: If a student's name appears in multiple files, updating it requires changing all files.	Example: The student's name is stored **once** on the "Students" table and referenced elsewhere by ID.
3. Relationships	**No relationships** between files. If you need to connect data (e.g., students and grades), you must	**Built-in relationships** using **keys**:
		- **Primary Key (PK)**: Unique ID for each record (e.g., Student ID).

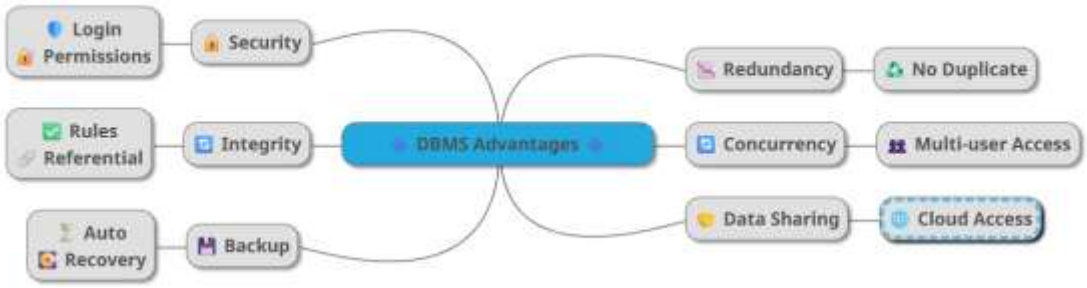
(How Data Connects)	manually match them (slow & error-prone).	- **Foreign Key (FK)**: Links to another table (e.g., Grades.StudentID points to Students.ID).
4. Example Usage	- **Simple lists** (contact lists, grocery lists).	- **Banking systems** (customer accounts, transactions).
	- **Single-user apps** (a personal budget tracker in Excel).	- **E-commerce** (products, orders, customers).
	- **Config files** (settings stored in .txt or .json).	- **School management** (students, courses, grades).
5. Query Capabilities (Searching Data)	**Limited searching** – you can only filter rows (e.g., find all students with Grade "A").	**Powerful querying** with **SQL** (Structured Query Language).
	No way to **combine data** from multiple files easily.	Example: SELECT Name FROM Students WHERE Grade = 'A'
		Can **join tables** (e.g., show students + their grades in one report).

6. Scalability (Handling Growth)	**Works poorly with large data** – slow to open, search, and update.	**Handles large data efficiently** – optimized for speed even with millions of records.
	Example: A 100,000-row Excel file crashes or loads slowly.	Example: Facebook's database stores billions of user posts and comments.
7. Data Integrity (Accuracy & Consistency)	**Low integrity** – no rules to prevent errors.	**High integrity** – enforces rules:
	Example: You can enter "Age: ABC" (text instead of a number).	- Data types (e.g., "Age" must be a number).
		- Constraints (e.g., "Email must be unique").
		- Prevents orphan records (e.g., can't delete a student if grades exist).
8. Security	**Minimal security** – anyone with file access can edit/delete data.	**Strong security** – user logins, permissions (e.g., only admins can delete records).
	Example: A shared Excel file can be accidentally modified.	Example: Banks restrict who can see customer data.

9. Flexibility (Changes & Updates)	**Easy to modify** – just edit the file (add/remove columns).	**Harder to change** – modifying a database **schema** (structure) requires careful planning.
	Example: Adding a "Phone Number" column in Excel.	Example: Adding a "Phone Number" column may require updating queries and apps.
10. Performance (Speed)	**Fast for small data** – quick to open and edit a small file.	**Optimized for speed** – even with huge data, indexes speed up searches.
	Slow for large data – searching 10,000+ records take time.	Example: Finding a single customer in a million-record database is instant.
Best For	✅ **Small, simple datasets** (a to-do list).	✅ **Large, complex data** (banking, e-commerce).
	✅ **Single-user applications**.	✅ **Multi-user systems** (schools, hospitals).
	✅ **Quick, one-time tasks**.	✅ **Apps needing fast, reliable queries**.

Feature	Flat File System	Relational Database (RDBMS)
Structure	Single file (CSV, Excel, text) with rows & columns.	Multiple tables linked via keys (primary/foreign).
Data Redundancy	High (duplicate data across files).	Low (normalization minimizes duplication).
Relationships	No built-in relationships; manual linking.	Supports 1:1, 1:many, many:many relationships.
Example Usage	- Contact lists (CSV).	- Banking systems.
	- Configuration files.	- E-commerce databases.
Query Capabilities	Limited (basic search/filtering).	Advanced (SQL for joins, aggregations, etc.).
Scalability	Poor (slow with large datasets).	Good (handles large, complex data efficiently).
Data Integrity	Low (no constraints; prone to errors).	High (ACID compliance, constraints, validation).
Security	Minimal (file-based access control).	Robust (user permissions, encryption).
Flexibility	High (easy to modify structure).	Low (schema changes require migrations).
Performance	Fast for small, simple reads/writes.	Optimized for complex queries but may need tuning.
Best For	Small datasets, simple apps, one-time use.	Large, structured data with relationships.

DBMS Advantages – Mind Map



Roles in a Database System

Role	What They Do	Example Tasks
1. System Analyst	Acts as a bridge between business needs and technical solutions. Defines what the database should do.	- Interviews users to gather requirements
		- Creates system specifications
		- Defines business rules for data
2. Database Designer	Designs the database structure for efficiency and usability. Focuses on how data will be organized.	- Creates ER (Entity-Relationship) diagrams
		- Normalizes tables to reduce redundancy
		- Defines keys and relationships
3. Database Developer	Implements the database design by writing code (SQL, stored procedures, functions).	- Creates tables, views, and indexes
		- Writes complex queries and triggers

		- Optimizes SQL for performance
4. Database Administrator (DBA)	Manages, secures, and maintains the database system. Ensures it runs smoothly.	- Installs & configures database software
		- Sets up backups and recovery plans
		- Monitors performance and tunes queries
5. Application Developer	Builds software that interacts with the database (e.g., web/mobile apps).	- Connects apps to the database (APIs, ORM)
		- Implements CRUD operations (Create, Read, Update, Delete)
		- Ensures app queries are efficient
6. BI (Business Intelligence) Developer	Uses data to generate reports, dashboards, and insights for decision-making.	- Designs data warehouses - Creates ETL (Extract, Transform, Load) processes - Builds dashboards in tools like Power BI, Tableau

Types of Databases

1. Relational vs. Non-Relational Databases

Feature	Relational (SQL) Databases	Non-Relational (NoSQL) Databases
Structure	Data is stored in **tables** with rows and columns. Tables are linked using keys (e.g., customer ID).	Data is stored in **flexible formats**: - Documents (JSON-like) - Key-value pairs - Graphs - Wide-column stores.
Schema	**Fixed structure** (must define columns beforehand).	**Schema-less** (can add fields on the fly).
Query Language	Uses **SQL** (e.g., SELECT * FROM users).	Uses **database-specific methods** (e.g., MongoDB queries in JSON format).
Scalability	Vertical (upgrade server hardware).	Horizontal (add more servers easily).
Examples	MySQL, PostgreSQL, Oracle.	MongoDB (documents), Cassandra (wide-column), Redis (key-value).

Use Cases	- Banking systems (needs strict consistency).	- Social media (flexible, fast-growing data).
	- Inventory management (structured data).	- IoT sensor data (high volume, varied formats).

2. Centralized vs. Distributed vs. Cloud Databases

Type	Description	Pros	Cons	Use Cases
Centralized	All data is stored in **one location** (single server).	- Simple to manage.	- Single point of failure.	Small businesses, legacy systems.
		- Consistent data.	- Limited scalability.	
Distributed	Data is spread across **multiple servers/locations**.	- Fault-tolerant (no single failure breaks everything).	- Complex to manage.	Global apps (e.g., Netflix, Amazon).
		- Scalable.	- Consistency challenges.	
Cloud	Hosted on **third-party cloud platforms** (AWS, Google Cloud).	- No hardware maintenance.	- Ongoing costs.	Startups, SaaS apps (e.g., Slack, Zoom).
		- Pay-as-you-go scaling.	- Vendor lock-in risk.	

3. Real-World Use Case Examples

Relational Database (SQL)

- **Hospital Patient Records**: Structured data (patient ID, name, medical history) with strict relationships.
- **E-commerce Orders**: Tables for customers, products, and orders linked by IDs.

Non-Relational Database (NoSQL)

- **Twitter Posts**: Millions of tweets with varying data (text, images, hashtags) stored as JSON documents (MongoDB).
- **Netflix Recommendations**: Uses Cassandra to handle massive, unstructured viewing data across regions.

Centralized Database

- **Small Library System**: One server track books, members, and loans.

Distributed Database

- **Cryptocurrency (Bitcoin)**: Blockchain is a distributed ledger across thousands of nodes.

Cloud Database

- **Mobile Apps**: A weather app uses **Google Firebase** (cloud DB) to store user preferences globally.

Cloud Storage vs. Cloud Databases: A Beginner’s Guide

1. What is Cloud Storage?

Cloud storage (e.g., **AWS S3, Google Cloud Storage, Azure Blob**) is like a "digital warehouse" where you store files (documents, images, backups). It **supports databases** by:

- **Storing database backups** (e.g., nightly SQL dumps).
- **Holding large datasets** (e.g., CSV files imported into a database).
- **Serving static files** (e.g., product images linked to a database).

Key Difference:

- **Cloud Storage** = Stores **files** (unstructured data).
- **Cloud Databases** = Stores **structured data** with querying (e.g., customer records).

2. Advantages of Cloud-Based Databases

*(Examples: **Azure SQL, Amazon RDS, Google Cloud Spanner**)*

Advantage	Explanation	Example
1. No Hardware Hassle	No need to buy servers—cloud providers manage everything.	Amazon RDS auto-patches MySQL.
2. Scalability	Instantly add storage/CPU (pay for what you use).	Google Spanner scales globally for apps like Pokémon GO.
3. High Availability	Automatic backups, failover, and uptime guarantees (~99.99%).	Azure SQL keeps running even if a server crashes.
4. Global Access	Data centers worldwide reduce latency.	Netflix uses AWS to stream content globally.
5. Built-in Security	Encryption, firewalls, and compliance (GDPR, HIPAA).	Hospitals use cloud DBs for secure patient data.

3. Disadvantages & Challenges

Challenge	Explanation	Mitigation
1. Cost Overruns	Pay-as-you-go can get expensive if not monitored.	Set budget alerts (e.g., AWS Cost Explorer).
2. Vendor Lock-In	Hard to switch providers (unique tools/data formats).	Use open-source DBs (PostgreSQL) on cloud.
3. Latency Issues	Distance to cloud servers may cause delays.	Use **edge computing** (e.g., Cloudflare).
4. Security Fears	Storing sensitive data on third-party servers.	Encrypt data before uploading (client-side).
5. Internet Dependency	Offline access is limited.	Hybrid setups (cloud + local backup).