Database Search and Reporting

Comparison Assignment

Flat File Systems vs. Relational Databases

Name: Nabil Khalid AL-Balushi

Flat File Systems vs. Relational Databases

what is flat file system?
A flat file system is a way of storing data in simple format, typically in a plain text file or spreadsheet style format where all the infromation is stored in a single table or file, without any stuctured relationaship between different sets of data.

Characteristics of flat file system:

1. single table format:
   - Data is stored in way of rows and columns similar to Excel or CSV files.
   - Each row represents a recored, each column represens a filed.
2. Plain text or delimited format:
   - Common formats include CSV (Comma-Separated Values), TSV (Tab-Separated Values), or it can be a simple .txt file.
3. No Query Language:
   - SQL cannot run as queries.
   - To search or modify data, you often need to write code or scripts manually.

Where flat file systems are used:

1. Small datasets
2. Exported reports from a database
3. Configuration files in software systems
4. Log files for apps or operating systems
5. simple contact or address lists

Limitations of flat file systems:

Problem	why its matter
Data Redundancy	Same data is repeated many times, wasting space.
No Relationships	Cannot link employees to other tables like departments or projects.
No Validation	Allows incorrect or inconsistent data (like misspellings).
Not Scalable	Slows down with large files. Difficult to maintain and search.

What is relational databases?
A Relational Database is a type of database that stores data in structured tables (also called relations) and uses relationships between those tables to organize and retrieve data efficiently. It is one of the most common and powerful ways to manage large amounts of structured data

Concepts of Relational Databases:

1. Tables (Relations):
   - Data is organized in **tables**, where each **row** is a **record** and each **column** is a **field**.
   - Each table focuses on one type of data (e.g., customers, products, orders).
2. Primary Key:
   - A unique identifier for each record in a table.
3. Relationships:
   - **One-to-One**: One person has one passport.
   - **One-to-Many**: One customer places many orders.
   - **Many-to-Many**: Students enrolled in many courses, and each course has many students.

Benefits of Relational Databases:

Benefit	Description
Structured & Organized	Data is cleanly separated into related tables.
Supports Relationships	You can connect different types of data logically.
Reduces Redundancy	Avoids repeating the same data in multiple places.

1. Structure

Feature	Flat File Systems	Relational Databases
Format	Data stored in plain text or binary files (e.g., CSV, TXT).	Data organized into tables with rows and columns.
Schema	No strict schema enforcement.	Enforces a schema with defined data types and constraints.
Storage	Each file is typically independent.	Centralized database engine stores related data in linked tables.

2. Data Redundancy

Feature	Flat File Systems	Relational Databases
Redundancy	High – same data may be repeated across multiple files	Low – uses normalization to reduce duplication.
Management	Manual updates often lead to inconsistencies.	Handled through keys and relationships, ensuring consistency.

3. Relationships

Feature	Flat File Systems	Relational Databases
Data Linking	No inherent mechanism for linking records between files.	Uses primary and foreign keys to establish relationships.
Integrity	Hard to maintain referential integrity.	Strong support for data integrity through constraints.

4. Example Usage

Feature	Flat File Systems	Relational Databases
Typical Use Cases	Configuration files, simple data export/import, logging.	Enterprise applications, e-commerce platforms, inventory systems.
Tools	Excel, CSV editors, basic file readers.	MySQL, PostgreSQL, Oracle, Microsoft SQL Server.

5. Drawbacks

Feature	Flat File Systems	Relational Databases
Scalability	Poor – not ideal for large or complex datasets.	Scales well with complex, growing datasets.
Performance	Slower with large data due to lack of indexing.	Faster queries and updates due to indexing and optimization.
Security	Minimal – file-level access control only.	Advanced security features like user roles, access control, and encryption.
Data Integrity	Prone to errors and inconsistencies.	Enforced through constraints and rules.

Roles in a Database System:

In a database project, the System Analyst gathers and analyzes business requirements to ensure the database solution meets organizational goals. The Database Designer structures the database by defining tables, relationships, and data flows for optimal performance and organization. The Database Developer implements this design by writing SQL scripts, creating stored procedures, and optimizing database functions. The Database Administrator (DBA) manages the database's performance, security, backups, and user access. The Application Developer builds software that interacts with the database, handling data retrieval and user interaction. Finally, the BI (Business Intelligence) Developer transforms raw data into meaningful insights through dashboards, reports, and analytics, helping organizations make informed decisions.

1. System Analyst

- Role: Bridge between business needs and technical solutions.
- Responsibilities:
    - Gather and analyze user requirements
    - Define system specifications
    - Ensure the database solution aligns with business goals
    - Communicate between stakeholders and the development team

2. Database Designer

- Role: Designs the structure and organization of the database.
- Responsibilities:
    - Create the database schema (tables, relationships, keys)
    - Ensure efficient data organization and normalization
    - Plan data flow and storage strategy
    - Focus on scalability and performance

3. Database Developer

- Role: Implements and builds the database based on the design.
- Responsibilities:
    - Write SQL scripts and stored procedures
    - Develop triggers, views, and functions
    - Optimize queries for performance
    - Work closely with designers and application developers

4. Database Administrator (DBA)

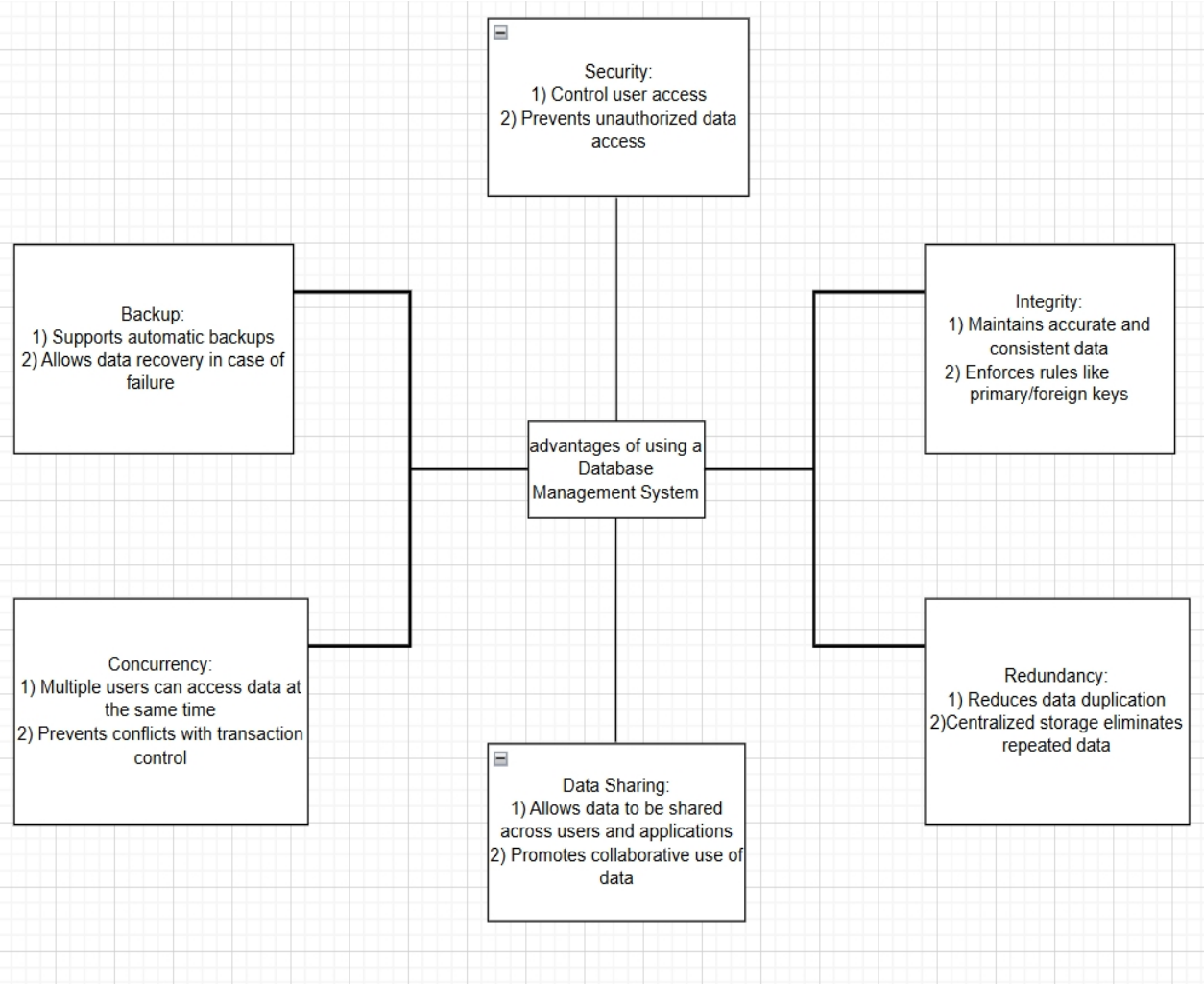
- Role: Manages and maintains the database system.
- Responsibilities:
    - Ensure data security, availability, and backup
    - Monitor performance and troubleshoot issues
    - Manage user access and permissions
    - Perform routine maintenance and updates

5. Application Developer

- Role: Builds software applications that interact with the database.
- Responsibilities:
    - Develop front-end or back-end components
    - Use APIs or SQL queries to access database data
    - Ensure smooth integration with the database
    - Handle user input and data validation

6. BI (Business Intelligence) Developer

- Role: Turns raw data into insights and reports.
- Responsibilities:
    - Design and develop dashboards and data visualizations
    - Write complex queries and data models for analysis
    - Work with data warehouses and reporting tools (e.g., Power BI, Tableau)
    - Help business users make data-driven decisions



Types of Databases

Relational vs. Non-Relational Databases

- Relational Databases (RDBMS)
    - Store data in tables with rows and columns
    - Use SQL for querying
    - Enforce strict schemas and relationships between tables
    - Examples: MySQL, PostgreSQL, Oracle
    - Use Case: Banking systems, ERPs, and applications with structured data
- Non-Relational Databases (NoSQL)
    - Store data in flexible formats: documents, key-value pairs, graphs, or columns
    - Ideal for unstructured or semi-structured data
    - Examples:
        - MongoDB (document-based)
        - Cassandra (wide-column store)
    - Use Case: Real-time analytics, IoT applications, content management systems, big data storage

Centralized vs. Distributed vs. Cloud Databases

- Centralized Database
    - All data stored in a single location/server
    - Easy to manage and secure but may have a single point of failure
    - Use Case: Small businesses, legacy systems, school databases
- Distributed Database

- Data spread across multiple locations or servers
- Offers higher availability and fault tolerance
- Use Case: Global applications like e-commerce platforms or social media

- Cloud Database
  - Hosted on cloud platforms (e.g., AWS, Azure, Google Cloud)
  - Scalable, flexible, and managed by a service provider
  - Use Case: SaaS applications, modern mobile/web apps, startups scaling rapidly

Cloud Storage and Databases

What is Cloud Storage and How Does It Support Databases?

Cloud storage refers to saving data on remote servers accessed over the internet, maintained by third-party providers (like AWS, Azure, or Google Cloud).

In the context of databases, cloud storage:

- Hosts the data files and backups
- Provides scalable and redundant infrastructure
- Supports high availability and fast access for cloud-based databases

It enables databases (e.g., Azure SQL, Amazon RDS) to operate without needing physical on-premise servers, allowing users to store, manage, and retrieve data from anywhere.

Advantages of Cloud-Based Databases

1. Scalability
   - Automatically adjusts resources (storage, compute) based on demand
2. High Availability

    - Redundant systems and failover options ensure uptime
3. Cost Efficiency
    - Pay-as-you-go pricing eliminates upfront hardware costs
4. Automatic Backups and Updates
    - Managed services handle maintenance, patches, and backups
5. Global Access
    - Accessible from anywhere with an internet connection
6. Integration with Cloud Services
    - Easily connects with analytics, AI/ML, and serverless platforms

Disadvantages or Challenges of Cloud-Based Databases

1. Data Security & Privacy Concerns
    - Sensitive data is stored off-site, requiring strong encryption and compliance
2. Internet Dependency
    - Requires stable internet for access and performance
3. Limited Control
    - Less direct control over hardware and some configurations
4. Costs Can Grow Over Time
    - Long-term or high-traffic usage may become expensive
5. Vendor Lock-In
    - Switching providers can be complex and costly