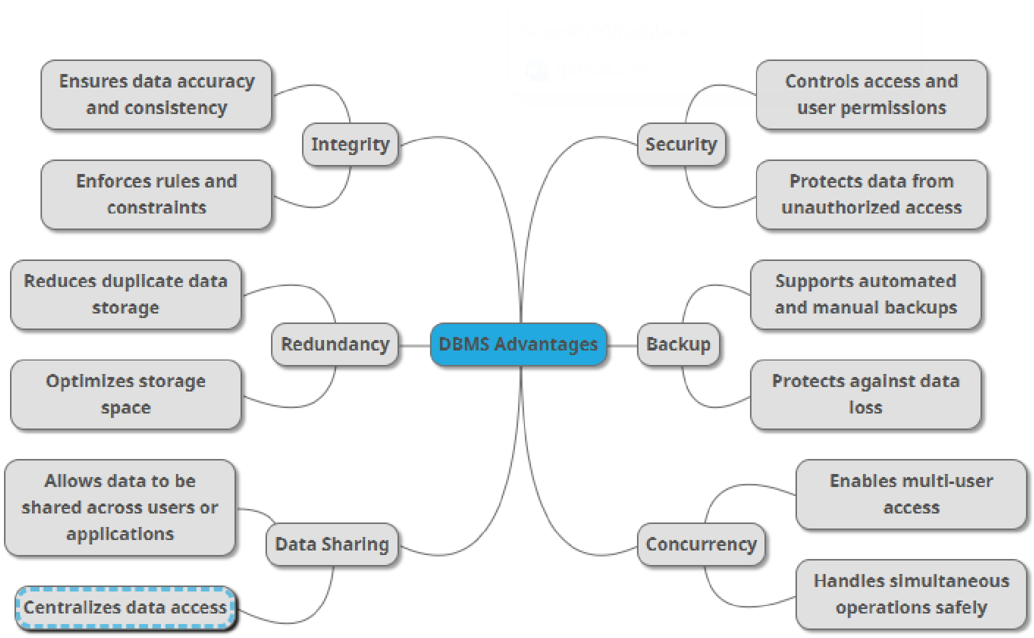
Database Search and Reporting Task

Comparison Assignment

Feature	Flat File System	Relational Database (RDBMS)
Structure	- Stores data in a single table or plain text file (e.g., CSV) - No defined schema enforcement	- Organized in multiple tables with rows and columns - Schema defined and enforced
Data Redundancy	- High redundancy due to lack of normalization	- Reduced redundancy via normalization and foreign key constraints
Relationships	- No native support for relationships	- Tables can be linked using primary and foreign keys
Example Usage	- Simple data storage (e.g., contact lists, logs, config files) - Quick prototyping	- Business applications (e.g., HR systems, finance, inventory) - Websites, CRMs
Drawbacks	- Difficult to manage as data grows - No data validation - Poor performance with large data - Prone to inconsistencies	- More complex to design and maintain - Requires specialized knowledge - Can be overkill for very small/simple datasets

DBMS Advantages Mind Map



Roles in a Database System

🧑‍💻 1. System Analyst

- **Role**: Acts as a bridge between business needs and technical solutions.
- **Responsibilities**:
  - Gathers and analyzes business requirements.
  - Translates user needs into technical specifications.
  - Recommends appropriate database and system architecture.
- **Focus**: Understanding **what** the system must do.

🧱 2. Database Designer

- **Role**: Designs the structure of the database.
- **Responsibilities**:
  - Creates the data model (ERD).

    - Defines tables, relationships, keys, and constraints.
    - Ensures normalization and efficient design.
- **Focus**: **How** the data is organized and stored.

🧑‍🔧 3. Database Developer

- **Role**: Builds the database based on the design.
- **Responsibilities**:
    - Writes SQL scripts (tables, views, stored procedures).
    - Implements triggers, functions, and logic for data processing.
    - Optimizes queries for performance.
- **Focus**: **Creating and programming** the database system.

🛡️ 4. DBA (Database Administrator)

- **Role**: Maintains and secures the database.
- **Responsibilities**:
    - User access control and security.
    - Backup and recovery management.
    - Performance tuning and monitoring.
    - Updates and patching.
- **Focus**: Ensuring the database is **safe, fast, and reliable**.

🧑‍💻 5. Application Developer

- **Role**: Builds applications that use the database.
- **Responsibilities**:
    - Develops front-end and back-end code.
    - Connects the app to the database (using APIs or drivers).
    - Implements business logic based on data interactions.
- **Focus**: Building **user-facing apps** that interact with the database.

📊 6. BI Developer (Business Intelligence Developer)

- **Role**: Turns data into insights for decision-making.
- **Responsibilities**:
    - Creates dashboards and reports.
    - Builds ETL (Extract, Transform, Load) processes.
    - Works with data warehouses and analytics tools (e.g., Power BI, Tableau).
- **Focus**: Making data **understandable and visual** for business users.

Types of Databases

Relational vs Non-Relational Databases

Type	Relational Database	Non-Relational Database
Structure	Tables with rows and columns (schema-based)	Document, key-value, graph, or wide-column stores
Example	MySQL, PostgreSQL, Oracle, SQL Server	MongoDB, Cassandra, Redis, Neo4j
Best For	Structured data, ACID compliance	Flexible schema, high scalability, big data
Query Language	SQL	Various (MongoDB uses BSON + query language)

🧠 **Use Case Example**:

- **Relational**: Banking system with complex transactions
- **Non-Relational**: Social media platform storing posts, comments (MongoDB)

Centralized vs Distributed vs Cloud Databases

Type	Centralized	Distributed	Cloud Database
Location	Single server/location	Multiple servers across locations	Hosted on cloud platforms (AWS, Azure, GCP, etc.)

Access	Local network	Global/local access with replication	Internet-based access
Fault Tolerance	Low (single point of failure)	High (data replicated across nodes)	Very high (managed by provider)
Scalability	Limited	High (horizontal scaling)	Very high (elastic scaling)
Example	Local MySQL server	Cassandra, Google Spanner	Amazon RDS, Firebase, Azure Cosmos DB

🧠 **Use Case Example**:

- **Centralized**: Small local business system
- **Distributed**: Real-time analytics across regions
- **Cloud**: E-commerce website using AWS RDS + S3

Cloud Storage and Databases

☁ What is Cloud Storage?

Cloud storage is a **service that allows data to be stored on remote servers**, accessed over the internet, and managed by a **cloud provider** like Amazon, Microsoft, or Google.

Examples:

- Amazon S3
- Google Cloud Storage
- Microsoft Azure Blob Storage

🗄 How Cloud Storage Relates to Databases

While **cloud storage** is often used for unstructured files (e.g., images, videos, logs), **cloud databases** are specialized services for **structured data** with features like querying, transactions, and indexing.

Cloud Storage	Cloud Database
Stores files, objects	Stores structured/tabular data
Examples: S3, Blob, GCS	Examples: Amazon RDS, Azure SQL, Google Spanner
Accessed via APIs	Accessed via SQL/NoSQL queries
Good for backups, media files	Good for transactional or analytical data

☁ Cloud-Based Databases

Examples include:

- **Amazon RDS** – Managed relational DB (MySQL, PostgreSQL, etc.)
- **Azure SQL Database** – Scalable Microsoft SQL in the cloud
- **Google Cloud Spanner** – Globally distributed SQL with NoSQL scale

☁ Cloud-Based Databases: Pros & Cons

✅ Advantages	💬 Description	⚠️ Disadvantages	💬 Description
Scalability	Easily scale up/down based on demand	**Latency**	Dependent on internet speed and location

Availability	High uptime with automatic failover	**Cost**	Can become expensive at scale or with high query loads
Maintenance-Free	No need to manage hardware, backups, or updates	**Vendor Lock-in**	Hard to migrate between cloud platforms
Global Access	Access data from anywhere via internet	**Limited Customization**	Less control over infrastructure compared to on-prem solutions
Security & Compliance	Encryption, access control, and compliance certifications	**Security Concerns**	Misconfigurations can lead to data leaks or breaches
Disaster Recovery	Built-in backup and restore capabilities		

⚙️ What is a Database Engine?

A **Database Engine** is the **core software component** of a database system that:

- **Stores**, **retrieves**, and **manages data**
- Executes **queries** and handles **transactions**
- Enforces **constraints**, **triggers**, and **procedures**

It's what **powers** the database and interprets **query languages** like SQL.

🔍 Popular Database Engines and Their Languages

🛠️ Database Engine	💬 Main Query Language	🧠 Notes
SQL Server	T-SQL (Transact-SQL)	Microsoft's SQL variant with procedural extensions

MySQL	ANSI SQL + MySQL-specific extensions	Open-source, widely used in web apps
Oracle	PL/SQL (Procedural Language for SQL)	Combines SQL with procedural features like loops, conditions
PostgreSQL	ANSI SQL + PL/pgSQL	Open-source, supports custom functions and advanced types

🔗 Is There a Relationship Between Engine and Language?

Yes ✅:

- Each **engine uses its own dialect** of SQL.
- **Standard SQL (ANSI SQL)** is supported by all engines, but each adds its own **extensions** for:
  - Procedures
  - Functions
  - Error handling
  - Triggers

🔄 Can One Language Work Across Different Engines?

✔️ **Partially**:

- **Basic SQL queries** (e.g., `SELECT`, `INSERT`, `UPDATE`) usually work across engines.
- But:
  - Procedural code (like loops, error handling) is **not portable**.
  - Functions, triggers, and stored procedures need **rewriting** per engine.

Example:

- A `SELECT * FROM Customers` will work on all engines.
- A `TRY...CATCH` block in **T-SQL** won't work in **PL/SQL**.

🔁 Is Migration Possible Between Engines?

From → To	Possible?	Tools Often Used

SQL Server → MySQL	✅	MySQL Workbench, AWS DMS
Oracle → PostgreSQL	✅	Oracle FDW, AWS Schema Conversion Tool (SCT)
MySQL → PostgreSQL	✅	pgLoader, DMS
SQL Server → PostgreSQL	✅	Babelfish, DMS, pgAdmin plugins

🌐 Many migrations are done for cost savings, cloud adoption, or open-source flexibility.

⚠ Challenges in Engine-to-Engine Migration

Challenge	Description
Different SQL Dialects	T-SQL ≠ PL/SQL ≠ MySQL SQL ≠ PL/pgSQL
Data Type Mismatch	`DATETIME` in SQL Server ≠ `TIMESTAMP` in PostgreSQL
Stored Procedures/Functions	Need to rewrite logic in the target engine's language
Triggers & Views	Must be manually reviewed and translated
Constraints & Indexes	Differences in syntax and behavior
Collation & Encoding	Text handling, case sensitivity, character sets can vary
Security/Roles	Users, roles, and permissions might not map directly
Performance Tuning	Indexes, query plans, and optimizers differ between engines

☑ What to Consider Before Transferring a Database

Element	What to Do
Data Types	Map equivalent types (e.g., `VARCHAR(MAX)` to `TEXT`)
Stored Procedures/Functions	Rewrite logic in new dialect
Triggers	Test and adapt triggers in the new engine
Indexes & Keys	Recreate based on performance needs
Foreign Keys & Constraints	Ensure referential integrity rules are preserved

Collation & Charset	Set compatible encoding (e.g., UTF-8)
Access Control	Reassign users/roles/permissions manually
Testing & Validation	Verify data integrity, app compatibility, and query performance

🛠️ Example Tools for Migration

- **MySQL Workbench Migration Wizard**
- **AWS Schema Conversion Tool**
- **pgLoader**
- **Oracle SQL Developer**
- **SQL Server Migration Assistant (SSMA)**

What is a Logical Schema?

A **Logical Schema** defines the **structure of the data** in terms of **entities**, **attributes**, and **relationships** without worrying about how it will be implemented.

- Focuses on **business rules** and **data organization**
- Independent of any database software
- Part of **conceptual/database design phase**

🧠 Think of it as a **blueprint**: What data you need and how it relates.

🖾 What is a Physical Schema?

A **Physical Schema** defines **how the data is stored in the database**—including:

- Tables
- Data types
- Indexes
- Storage format
- Constraints (PKs, FKs)

It's specific to the **DBMS** (like MySQL, PostgreSQL, etc.) and optimized for performance and storage.

🔍 Key Differences

Aspect	Logical Schema	Physical Schema
Focus	What data is stored	How data is stored and accessed
Level	High-level, abstract	Low-level, technical
DBMS Dependency	Independent	DBMS-specific
Includes	Entities, attributes, relationships	Tables, data types, indexes, keys
Audience	Business analysts, designers	Database admins, developers

🎓 Example: Student Entity

Concept	Logical Schema	Physical Schema (e.g., MySQL)
Entity/Table	`Student`	`CREATE TABLE Student (...);`
Attributes	StudentID, Name, DOB, Major	`StudentID INT PRIMARY KEY, Name VARCHAR(100), DOB DATE, Major VARCHAR(50)`
Relationships	One-to-many with `Course`	`FOREIGN KEY (CourseID) REFERENCES Course(CourseID)`

❗ Why Is Understanding Both Important?

- Logical schema helps **model real-world needs** and communication with stakeholders.
- Physical schema helps **implement and optimize** the system in a real DBMS.
- A good logical model ensures **accuracy**, while a good physical model ensures **efficiency**.