**Task 1 ( Report )**

**Database Search and Reporting Task**

**Objective:**

To develop research and analytical reporting skills by exploring key database concepts

**Make a new github repo ( Database Course documentation ) and add the following requirements to a one report and upload it**

**1. Comparison Assignment**

Create a comparison between **Flat File Systems** and **Relational Databases covering:**

|  | **Flat File systems** | **Relational Databases** |
| --- | --- | --- |
| **Structure** | Stores data in a plain text or spreadsheet format (e.g., .txt, .csv). Each file is independent. | **Data is stored in structured tables with rows and columns. Tables are linked by keys** |
| **Data Redundancy** | High – Same data might be repeated in multiple files. | **Low – Normalization reduces data duplication by separating data into related tables.** |
| **RelationShips** | No built-in support for relationships between files. Relationships must be managed manually. | **Supports complex relationships (e.g., one-to-many, many-to-many) using foreign keys.** |
| **Example usage** | Simple log files, contact lists in .csv, small configurations. | | **Business systems,**  **inventory systems,**  **student databases,**  **banking apps.** | | --- | |
| Drawbacks | - No data validation |  |

**2. DBMS Advantages Mind Map**

Draw a mind map (or use online tools like MindMup) illustrating the **advantages of using a DBMS**. Include short descriptions or icons for:

• Security

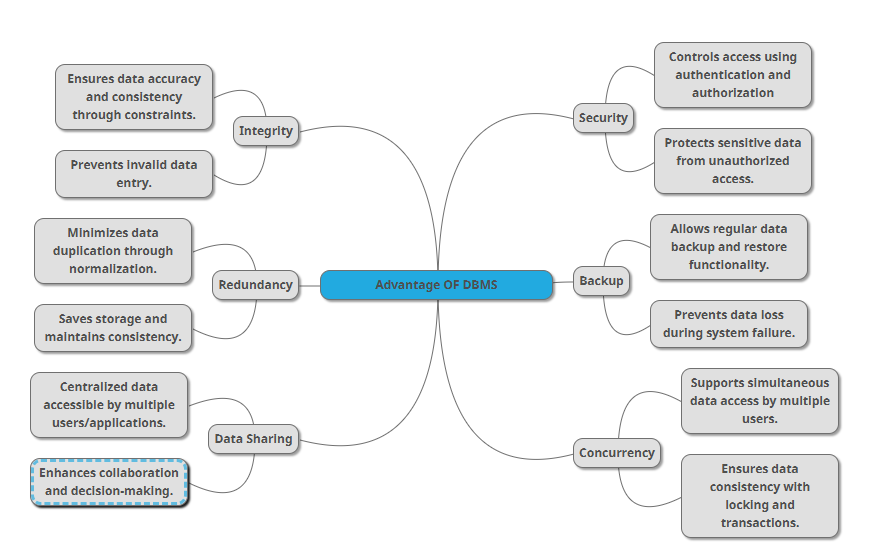
• Integrity

• Backup

• Redundancy

• Concurrency

• Data sharing



**3. Roles in a Database System**

Explaining each of the following roles:

**System Analyst:**

**Role**: Acts as a bridge between business requirements and technical solutions.  
**Responsibilities**:

* Gathers and analyzes user needs.
* Translates business processes into technical specifications.
* Works closely with database designers and developers to ensure alignment with business goals.

**Database Designer:**

**Role:** Designs the structure of the database.

**Responsibilities:**

* Defines the schema (tables, relationships, constraints).
* Ensures data normalization and integrity.
* Creates ER diagrams to visually represent data models.

**Database Developer**:

**Role**: Builds and maintains the database structure.

**Responsibilities**:

* Implements the database schema using SQL/DDL.
* Writes stored procedures, functions, and triggers.
* Optimizes queries for better performance.

**DBA (Admin):**

**Role:** Manages and maintains the database system.

**Responsibilities:**

* Controls access, user roles, and permissions.
* Performs backup, recovery, and performance tuning.
* Ensures database security, availability, and reliability.

**Application Developer:**

**Role:** Develops applications that interact with the database.

**Responsibilities:**

* Writes application code that reads from/writes to the database.
* Designs user interfaces (UI) and business logic.
* Works with APIs or ORM tools to access the database securely and efficiently.

**BI Developer:**

**Role**: Converts data into insights for business decisions.

**Responsibilities**:

* Creates dashboards, reports, and visualizations.
* Writes complex queries to extract and aggregate data.
* Works with tools like Power BI, Tableau, or SQL-based analytics.

**Additional Research Topics to Include in the Report:**

These topics are often overlooked but important for understanding the ecosystem of databases in modern applications.

**Types of Databases**

Search and briefly describe:

• **Relational** vs **Non-Relational** (e.g., MongoDB, Cassandra)

|  | Description | Examples | Use Cases |
| --- | --- | --- | --- |
| Relational (RDBMS) | Stores structured data in tables with fixed schemas. Uses SQL for queries. Ensures ACID properties. | MySQL, PostgreSQL, Oracle, SQL Server | Banking systems, payroll software, ERP systems |
| Non-Relational (NoSQL) | Stores unstructured or semi-structured data. Supports flexible schemas. Highly scalable. | MongoDB (Document), Cassandra (Wide-column) | Social networks, IoT data, real-time analytics, e-commerce catalogues |

• Centralized vs Distributed vs Cloud Databases

|  | Description | Examples | Use Cases |
| --- | --- | --- | --- |
| Centralized | All data stored on a single server/database. Easy to manage but less scalable and fault-tolerant. | On-premise MySQL, Oracle DB | Internal tools, school systems, desktop software |
| Distributed | Data is stored across multiple physical locations/nodes. Enhances reliability and scalability. | Cassandra, Google Spanner, CockroachDB | Global apps, streaming services, multi-region platforms |
| Cloud | Hosted in the cloud with managed services, auto-scaling, and backups. Accessible via the internet. | Amazon RDS, Azure SQL, Google Cloud Firestore | SaaS, mobile apps, startups, real-time systems |

• Use case examples

**Cloud Storage and Databases**

• What is **Cloud Storage** and how does it relate to databases?

**Cloud storage** refers to a service where data is stored on remote servers accessed via the internet. These servers are managed by cloud providers like Amazon Web Services (AWS), Google Cloud Platform (GCP), or Microsoft Azure.

• Advantages and Disadvantages of using cloud-based databases (e.g., Azure SQL, Amazon RDS, Google Cloud Spanner)

### ✅ Advantages of Cloud-Based Databases:

* **Scalability**: Automatically adjusts resources based on demand.
* **High Availability**: Built-in backups, failover, and disaster recovery.
* **Low Maintenance**: No need to manage hardware or software updates.
* **Global Deployment**: Easily accessible from different regions.
* **Strong Security**: Includes encryption, access control, and compliance support.

### ❌ Disadvantages of Cloud-Based Databases:

* **Cost**: Can become expensive with high data usage or traffic.
* **Vendor Lock-In**: Difficult to switch providers due to platform-specific features.
* **Limited Control**: Less access to underlying system configurations.
* **Latency**: Delays may occur if services are in different geographic locations.
* **Migration Complexity**: Transferring between engines is time-consuming and requires careful planning.

**Database Engines and Languages**

Search about:

• What is a **Database Engine**?

the core software component that stores, retrieves, and manages data in a database. It handles tasks like executing queries, enforcing rules, managing transactions, and ensuring data consistency and integrity.

• Examples: **SQL Server**, **MySQL**, **Oracle**, **PostgreSQL**

• What languages do they use? (e.g., T-SQL, PL/SQL, ANSI SQL)

**SQL Server** → Uses **T-SQL (Transact-SQL)**

**MySQL** → Uses **ANSI SQL** with MySQL

**Oracle** → Uses **PL/SQL (Procedural Language/SQL)**

**PostgreSQL** → Uses **ANSI SQL + PL/pgSQL**

• Is there a relationship between the **engine** and the **language**?

Yes, each engine uses **its own SQL dialect**, which is an extension or variation of standard **ANSI SQL**. These dialects include custom features and syntax specific to each engine.

• Can one language work across different engines?

**Basic ANSI SQL** (e.g., SELECT, INSERT, UPDATE) is mostly compatible across all engines.

**Advanced features** (e.g., stored procedures, triggers, error handling) vary, and code written for one engine’s dialect (like T-SQL or PL/SQL) **may not run on another without modification**.

**Can We Transfer a Database Between Engines?**

Yes, it is **possible to migrate** a database from one engine to another—such as from **SQL Server to MySQL** or **Oracle to PostgreSQL**—but it requires careful planning and execution due to differences between engines.

• Is it possible to migrate a database from **SQL Server to MySQL**, or **Oracle to PostgreSQL**? • What are the challenges of engine-to-engine migration?

**Different SQL Dialects**: T-SQL (SQL Server), PL/SQL (Oracle), and others use different syntax and logic.  
**Data Type Incompatibility**: Some data types don’t have direct equivalents (e.g., NUMBER in Oracle vs. INT in PostgreSQL).

**Stored Procedures and Functions**: These often need to be rewritten due to language differences.

**Triggers and Views**: Logic and performance behavior may differ.  
**Constraints and Indexes**: Need to be manually reviewed and adjusted.

• What should we consider before transferring (data types, triggers, stored procedures, etc.)?

**Logical vs. Physical Schema**

Data Types: Ensure type mapping between source and target engines.

Triggers & Constraints: Redesign if syntax or behavior differs.

Stored Procedures & Functions: Rewrite using the target engine’s supported language.

Character Sets & Collation: Match encoding to avoid data corruption.

Tools: Use migration tools like AWS DMS, Oracle SQL Developer, or pgLoader.

Testing: Perform data validation and performance testing after migration.

• What is the **Logical Schema** in database design?

A logical schema is the high-level structure of the database, describing entities, their attributes, and relationships without focusing on how they are physically implemented. It’s used during the design phase to plan the database’s structure conceptually.

• What is the **Physical Schema**?

A physical schema defines the actual implementation of the database on hardware/software. It includes table structures, data types, indexes, storage locations, and performance tuning settings.

• What’s the difference between them?

• Why is it important to understand both?

• Example: Show how one entity (e.g., Student) would appear in both logical and physical schemas.

--------------------------------------------------------------------------------------------------------------------------------------

**Evaluation Criteria:**

**Criteria Marks**

Content Accuracy and Depth 10

Research Effort 10

Organization & Clarity 5

Visuals and Mind Map 5

GitHub Submission 5

**Total 35 marks**