Database Search and Reporting Task

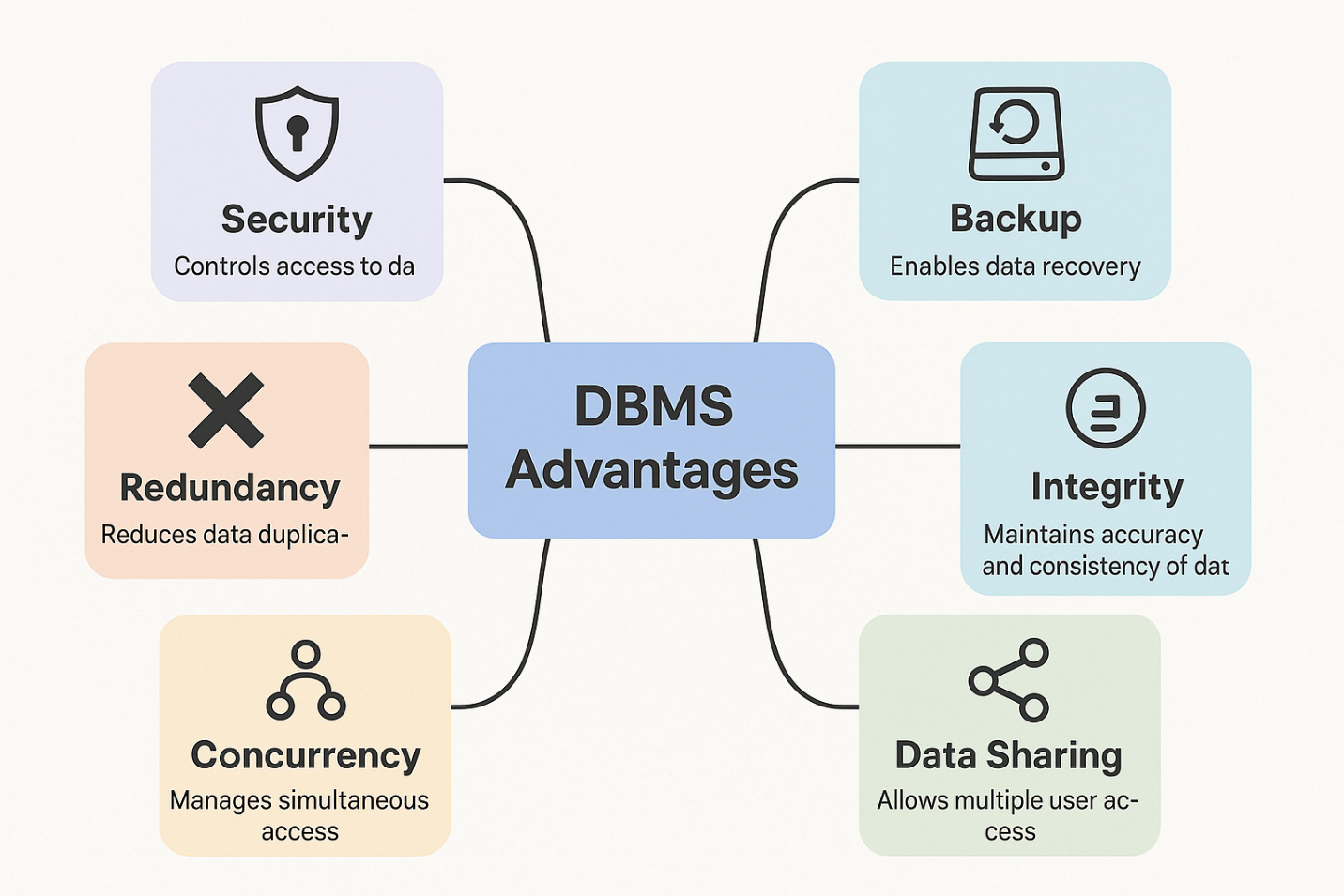
Done By:

Haroon Al Qasmi

# Comparison Assignment

|  |  |  |
| --- | --- | --- |
| Comparison Subjects: | Flat File Systems | Relational Databases |
| Structure | Data is stored in a single table or file, often in plain text or CSV format. No strict schema enforcement. | Data is stored in multiple related tables with well-defined schemas using rows and columns. |
| Data Redundancy | High likelihood of redundancy as all data is stored in one place without normalization. | Low redundancy due to normalization and separation of data into related tables. |
| Relationships | No inherent way to represent relationships between data elements. | Supports complex relationships using primary and foreign keys. |
| Example usage | Simple data storage, configuration files, log files, or small datasets. Example: a CSV file storing customer contact info | Enterprise applications, e-commerce platforms, banking systems. Example: a MySQL or PostgreSQL database for customer orders. |
| Drawbacks | |  | | --- | |  |  |  | | --- | | - Poor scalability and data integrity - No support for complex queries - Difficult to maintain and update | | - Higher complexity and setup time - Requires more system resources - Overhead for small/simple tasks |

# DBMS Advantages Mind Map



# Roles in a Database System

## System Analyst

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

**Key Responsibilities:**

* Understand business processes and user requirements.
* Analyze current systems and identify improvement areas.
* Create functional specifications for system design.

**Goal:** Ensure the final system meets user and business needs effectively.

## Database Designer

**Role:** Designs the structure and schema of databases.

**Key Responsibilities:**

* Define tables, relationships, keys, and constraints.
* Normalize data to eliminate redundancy.
* Create entity-relationship (ER) diagrams and data models.

**Goal:** Ensure an efficient, scalable, and logical database structure.

## Database Developer

**Role:** Builds and maintains the database logic and programs.

**Key Responsibilities:**

* Write SQL queries, stored procedures, and triggers.
* Optimize database performance.
* Implement and test the database components.

**Goal:** Provide functional and optimized data access for applications.

## DBA (Admin)

**Role:** Manages the operational aspects of databases.

**Key Responsibilities:**

* Install, configure, and upgrade database systems.
* Perform backup and recovery tasks.
* Monitor security, availability, and performance.

**Goal:** Ensure the database runs smoothly, securely, and efficiently.

## Application Developer

**Role:** Develops the front-end or back-end application logic that interacts with the database.

**Key Responsibilities:**

* Write code in languages like Java, Python, .NET, etc.
* Implement user interfaces and application features.
* Integrate application logic with the database.

**Goal:** Build user-friendly and functional applications that utilize database data.

## BI Developer

**Role:** Converts raw data into meaningful insights for decision-making.

**Key Responsibilities:**

* Develop dashboards and reports using BI tools (e.g., Power BI, Tableau).
* Design and manage data warehouses or data marts.
* Work with large datasets and perform ETL (Extract, Transform, Load) operations.

**Goal:** Help stakeholders make data-driven decisions.

# Types of Databases

| **Type** | **Description** | **Examples** | **Use Cases** |
| --- | --- | --- | --- |
| **Relational (RDBMS)** | Stores data in structured tables with rows and columns. Supports SQL. | MySQL, PostgreSQL, Oracle, SQL Server | Banking systems, HR systems, ERP, e-commerce |
| **Non-Relational (NoSQL)** | Stores unstructured or semi-structured data. Flexible schema. Includes document, key-value, column, and graph types. | MongoDB (document), Cassandra (column), Redis (key-value) | IoT data, social networks, real-time analytics, content management |

| **Type** | **Description** | **Examples** | **Use Cases** |
| --- | --- | --- | --- |
| **Centralized** | All data is stored in a single location/server. Easy to manage but a single point of failure. | Microsoft Access, single-server MySQL | Small business apps, personal projects |
| **Distributed** | Data is distributed across multiple physical locations or servers. Increases reliability and speed. | Google Spanner, Cassandra, Amazon DynamoDB | Global apps, telecom systems, real-time systems |
| **Cloud** | Hosted and managed on cloud platforms, scalable and accessible remotely. | Amazon RDS, Azure SQL, Google Cloud Firestore | SaaS applications, web/mobile apps, scalable services |

Use Case summary:

* **Relational:** Financial transactions, inventory management, CRM.
* **Non-Relational:** Real-time analytics, messaging apps, personalized content feeds.
* **Centralized:** Local POS systems, internal tools.
* **Distributed:** Multinational platforms, high-availability systems.
* **Cloud:** E-commerce websites, online education platforms, scalable APIs.

# Cloud Storage and Databases

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

**Cloud Storage** is a model where data is stored on remote servers accessed via the internet. These servers are maintained by third-party providers (like AWS, Microsoft Azure, or Google Cloud). It allows users and organizations to store, manage, and retrieve data without having to manage physical infrastructure.

Cloud storage supports the backend of **cloud-based databases** by providing scalable, secure, and resilient infrastructure. A **cloud database** is a database that runs on a cloud computing platform and uses cloud storage for its data files.

Examples include:

* **Amazon RDS (Relational Database Service)**
* **Azure SQL Database**
* **Google Cloud Spanner**

## Advantages and Disadvantages of using cloud-based databases

| **Advantage** | **Explanation** |
| --- | --- |
| **Scalability** | Automatically scales up/down based on usage or storage needs. |
| **Cost Efficiency** | Pay-as-you-go pricing reduces capital expenses and operational overhead. |
| **High Availability** | Built-in redundancy and failover mechanisms ensure minimal downtime. |
| **Automatic Maintenance** | Providers handle updates, backups, and patches. |
| **Global Accessibility** | Data can be accessed from anywhere with an internet connection. |
| **Security** | Robust built-in security features like encryption, access control, and auditing. |

| **Disadvantage** | **Explanation** |
| --- | --- |
| **Internet Dependency** | Requires reliable internet connection for access and operations. |
| **Vendor Lock-In** | Migrating between providers (e.g., from AWS to Azure) can be complex and costly. |
| **Data Privacy Concerns** | Sensitive data in the cloud may raise compliance or legal issues. |
| **Limited Control** | Less customization and control compared to on-premises systems. |
| **Latency** | Some operations may experience delays due to data transmission over the internet. |

# Database Engines and Languages

## What is a Database Engine?

A **Database Engine** is the core service that stores, processes, and secures data. It handles tasks such as:

* Data storage and retrieval
* Transaction management
* Query processing and optimization
* Indexing and locking

It’s the "brain" of a database system that executes SQL commands and manages data access.

## Examples of Database Engines and Languages

| **Database Engine** | **Vendor** | **Primary SQL Language** | **Notes** |
| --- | --- | --- | --- |
| **SQL Server** | Microsoft | **T-SQL (Transact-SQL)** | Extended from ANSI SQL with procedural programming features. |
| **MySQL** | Oracle (formerly Sun) | **SQL (with some extensions)** | Uses standard SQL with vendor-specific extensions. |
| **Oracle Database** | Oracle Corporation | **PL/SQL (Procedural Language/SQL)** | Adds procedural constructs like loops and conditions. |
| **PostgreSQL** | PostgreSQL Global Dev Group | **PL/pgSQL + SQL** | Uses standard SQL and procedural extensions like PL/pgSQL. |

## The relationship between the engine and the language

* Each **database engine** supports **standard SQL** (ANSI SQL), but most also include **proprietary extensions**.
* These extensions are tailored to the engine’s features and performance optimizations.

For example:

* T-SQL is specific to SQL Server, and includes advanced error handling and control-of-flow language.
* PL/SQL is designed for Oracle, enabling stored procedures and complex business logic.

## Can one language work across different engines?

* **ANSI SQL** (the core subset of SQL) **is portable** and works across most engines.
* However, **vendor-specific features and extensions** (like T-SQL or PL/SQL) are **not portable** without modification.

So:

* **Basic SQL queries** (SELECT, INSERT, UPDATE, DELETE) can work across engines.
* **Advanced logic** (stored procedures, triggers, functions) usually requires engine-specific syntax.

# Can We Transfer a Database Between Engines?

**Yes**, but it's not a one-click process. Tools and manual adjustments are often needed to bridge the differences in SQL dialects, features, and storage mechanisms.

Common tools for migration:

* **AWS Schema Conversion Tool**
* **Oracle SQL Developer**
* **Microsoft Data Migration Assistant**
* **pgLoader** (for PostgreSQL)
* **MySQL Workbench Migration Wizard**

## the challenges of engine-to-engine migration

| **Challenge** | **Explanation** |
| --- | --- |
| **SQL Syntax Differences** | Each engine uses different SQL dialects (e.g., T-SQL, PL/SQL). |
| **Data Type Mismatches** | Data types (e.g., NVARCHAR, NUMBER, BOOLEAN) may not have direct equivalents. |
| **Stored Procedures & Triggers** | Custom logic written in proprietary languages may require complete rewriting. |
| **Indexing & Constraints** | Unique rules for foreign keys, indexes, and constraints must be adapted. |
| **Performance Tuning** | Optimizations may not translate directly and may require re-tuning. |
| **Permissions/Security** | Roles and user permissions systems can vary greatly. |
| **Application Dependencies** | Any connected applications may need code changes to work with the new engine. |

## What should we consider before transferring

| **Aspect** | **Considerations** |
| --- | --- |
| **Data Types** | Ensure equivalent or compatible data types exist in the target engine. |
| **Schema Design** | Consider normalization rules and primary/foreign key definitions. |
| **Stored Procedures** | Rewrite using the target engine’s procedural language (e.g., T-SQL → PL/pgSQL). |
| **Triggers & Functions** | Review and rewrite as per target engine’s syntax and features. |
| **Constraints & Indexes** | Rebuild manually if automatic tools do not handle them properly. |
| **Data Integrity** | Validate data accuracy after transfer (row counts, relationships, etc.). |
| **Tool Support** | Choose tools that offer schema conversion and data migration. |