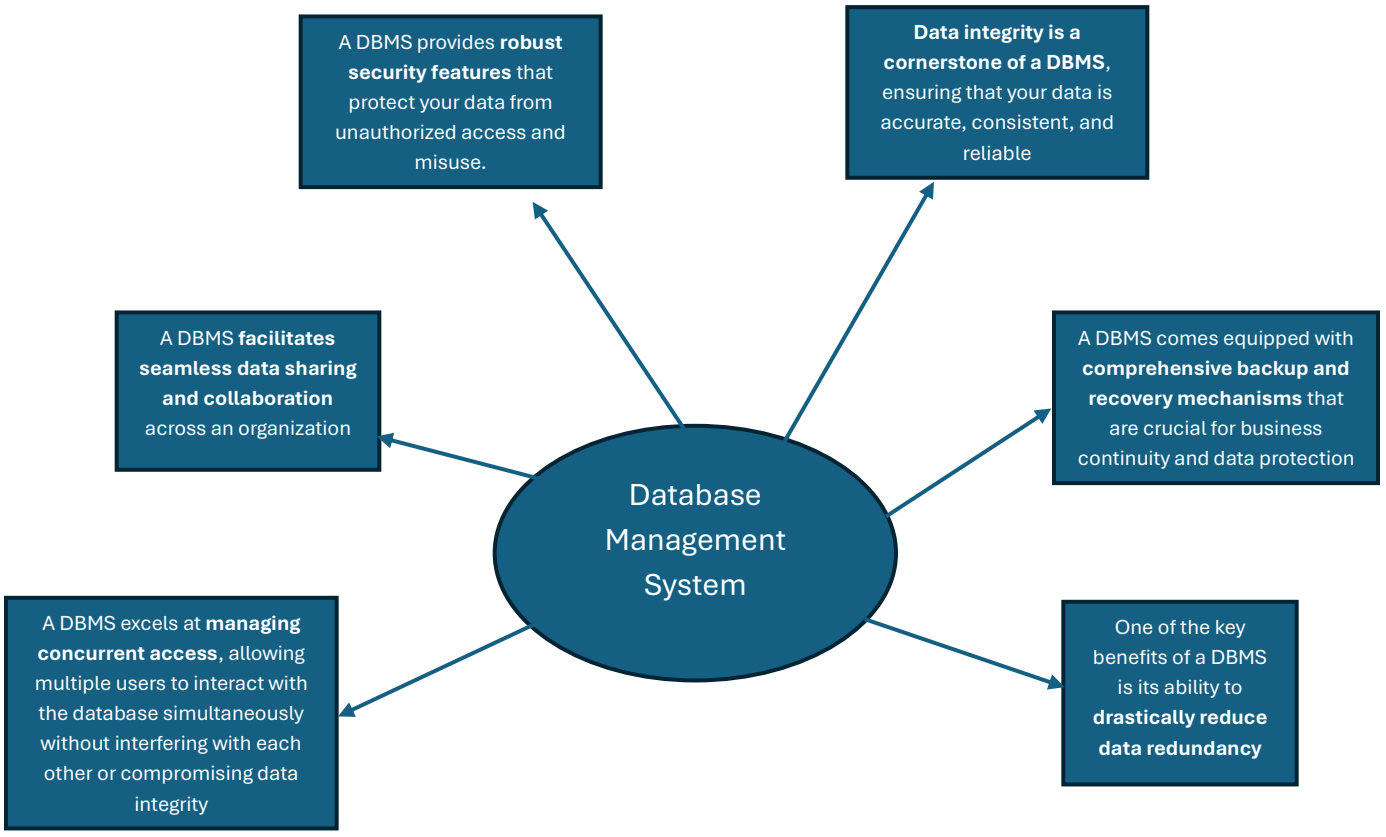
Task 1

Titles	Flat File Systems	Relational Databases
Structure	there are no structures for indexing or recognizing relationships between records	It uses tables, rows, and columns, to create relationships between the tables through keys. This allows for efficient storage and retrieval of data
Data Redundancy	Data is stored in a single table, leading to Data Redundancy(repetition of data)	It reduces, but doesn't eliminate Data redundancy
Relationships	No data relationships in Flat File Systems	Tables(entities) are logically related to each other, allowing data to be stored efficiently and retrieved accurately
Example Usage	A database application for an online video game	calculator
Drawbacks	1. scalability challenges, 2. schema rigidity, 3. difficulties handling unstructured data	as the more data grows the more difficult it is to Manage and manipulate data

Task 2



Task 3

Roles in a Database System:

System Analyst

The System Analyst acts as the bridge between the business stakeholders and the technical team. Their primary role is to understand the organization's needs, problems, and processes, and then translate these into detailed requirements for the database system. They conduct thorough research, gather user requirements through interviews, workshops, and documentation review, and analyze existing systems. Based on this analysis, they propose solutions, define the scope of the project, and create functional specifications that guide the database designer and developers. They often play a crucial role in ensuring the final system aligns with business objectives and user expectations.

Database Designer

The Database Designer is responsible for creating the logical and physical structure of the database. Based on the requirements gathered by the System Analyst, they develop the conceptual data model (e.g., Entity-Relationship Diagrams - ERDs), which defines the entities, attributes, and relationships within the data. They then translate this into a logical data model, and finally into a physical data model, specifying tables, columns, data types, indexes, and constraints for a particular DBMS. Their goal is to create an efficient, scalable, and well-normalized database schema that supports the application's needs while ensuring data integrity and performance.

Database Developer

The Database Developer is primarily responsible for implementing and maintaining the database components. They translate the database design into actual database objects, writing code for stored procedures, functions, triggers, and views using SQL and other database-specific languages (e.g., PL/SQL for Oracle, T-SQL for SQL Server). They are involved in data migration, performance tuning of queries, and troubleshooting database-related issues. They work closely with application developers to ensure the application efficiently interacts with the database, often optimizing queries and ensuring data access methods are secure and performant.

Database Administrator (DBA)

The Database Administrator (DBA) is responsible for the overall health, performance, security, and availability of the database system. Their duties are broad and include installing and configuring database software, managing user accounts and security

permissions, monitoring database performance, and optimizing queries. They are also responsible for performing regular backups and implementing disaster recovery plans, ensuring data integrity, and troubleshooting database issues. DBAs are critical for maintaining the operational efficiency and reliability of the database environment.

Application Developer

The Application Developer focuses on building the software applications that interact with the database to provide functionality to end-users. While they might have some database knowledge, their primary expertise lies in programming languages (e.g., Java, Python, C#, JavaScript) and frameworks used to create user interfaces and business logic. They write code to retrieve, insert, update, and delete data from the database, often utilizing APIs, ORMs (Object-Relational Mappers), or direct SQL queries. They work closely with Database Developers to understand the database schema and optimize their application's data access patterns for efficiency.

BI (Business Intelligence) Developer

The BI (Business Intelligence) Developer specializes in transforming raw data into meaningful insights for business decision-making. They design, develop, and maintain data warehouses and data marts, often using Extract, Transform, Load (ETL) processes to collect data from various sources, clean it, and load it into a structured format optimized for analytical queries. They create dashboards, reports, and data visualizations using BI tools (e.g., Tableau, Power BI, QlikView) to present complex data in an understandable way to business users, helping them identify trends, patterns, and opportunities. They work closely with business stakeholders to understand their analytical needs and provide the necessary data-driven intelligence.

Task 3

Relational vs. Non-Relational Databases

Relational Databases Relational databases, often referred to as SQL databases, store data in structured tables with predefined schemas. Data is organized into rows and columns, and relationships between tables are established using primary and foreign keys. This strict schema ensures data integrity and consistency, making them ideal for applications requiring ACID (Atomicity, Consistency, Isolation, Durability) properties.

- **Use Case Example:** Online transaction processing (OLTP) systems, financial systems, e-commerce platforms, and applications where data integrity and complex querying are paramount (e.g., banking systems managing customer accounts and transactions).

Non-Relational Databases (NoSQL) Non-relational databases, or NoSQL databases, offer more flexible data models than traditional relational databases. They do not typically use a fixed, tabular schema, allowing for varied data structures like documents, key-value pairs, wide-column stores, or graphs. This flexibility makes them highly scalable and suitable for handling large volumes of unstructured or semi-structured data, often at the expense of strict ACID compliance.

- **Example Non-Relational DBs:**
    - **MongoDB:** A document-oriented database that stores data in flexible, JSON-like documents.
        - **Use Case Example:** Content management systems, mobile applications, and real-time analytics where data schema can evolve rapidly or data is semi-structured (e.g., storing user profiles with varying attributes).
    - **Cassandra:** A wide-column store designed for handling massive amounts of data across many servers, providing high availability with no single point of failure.
        - **Use Case Example:** Large-scale distributed applications requiring high write throughput and continuous availability, such as IoT data ingestion, real-time messaging, and fraud detection (e.g., storing sensor data from millions of devices).

Centralized vs. Distributed vs. Cloud Databases

Centralized Databases A centralized database system stores all its data on a single computer system or server at a single location. All users and applications connect directly to this single server to access and manipulate data. This architecture simplifies management, backup, and security but can become a bottleneck in terms of performance and scalability as the volume of data or number of users grows.

- **Use Case Example:** Small to medium-sized businesses with limited data volume and user concurrency, where administrative simplicity and cost-effectiveness are priorities (e.g., a local library's catalog system, a small business's internal accounting database).

Distributed Databases A distributed database system has its data stored across multiple physical locations, machines, or nodes, which are connected by a network. Data can be replicated or partitioned across these nodes. This architecture offers high availability, fault tolerance, and scalability, as the workload can be distributed among different servers.

However, it introduces complexity in terms of data consistency, synchronization, and distributed transaction management.

- **Use Case Example:** Large enterprises or global applications requiring high availability, fault tolerance, and the ability to handle massive datasets and user loads across geographical regions (e.g., global e-commerce platforms, social media networks like Facebook or X (formerly Twitter), where user data is spread across data centers worldwide).

Cloud Databases Cloud databases are database services built and accessed through a cloud computing platform (e.g., AWS, Azure, Google Cloud). Instead of managing hardware and software on-premises, users provision and scale database instances as a service. Cloud databases can be relational or non-relational and offer benefits like scalability, elasticity (ability to easily scale up or down), high availability, disaster recovery, and reduced operational overhead, as the cloud provider handles much of the underlying infrastructure management.

- **Use Case Example:** Startups and enterprises looking for managed services to avoid infrastructure setup and maintenance, applications with variable workloads, or those needing global deployment capabilities. This includes almost any modern web application, data analytics platforms, and microservices architectures that benefit from the flexibility and scalability of the cloud (e.g., Netflix using AWS databases for its streaming service, a startup building a new SaaS product).

Task 4

What is Cloud Storage and how does it support database functionality?

Cloud Storage refers to a service where digital data is stored in logical pools across multiple distributed servers, rather than on a single local server or personal computer. These servers are typically managed by a third-party cloud provider (like Amazon Web Services, Microsoft Azure, Google Cloud). Cloud storage services abstract away the underlying infrastructure, allowing users to store and retrieve data on demand without worrying about hardware procurement, maintenance, or scaling.

Cloud storage supports database functionality in several critical ways:

1. **Persistence and Durability for Database Files:** Databases, whether relational or NoSQL, inherently store their data, logs, and configuration files. Cloud storage services (like Amazon S3, Azure Blob Storage, or Google Cloud Storage) provide highly durable, available, and scalable storage for these underlying database files. This means that even if a virtual machine hosting a database instance fails, the

actual database files persist safely in cloud storage and can be quickly attached to a new instance.

2. **Backup and Recovery:** Cloud storage is fundamental for robust database backup and recovery strategies. Database backups (full, incremental, transaction logs) can be easily stored in highly durable and geographically redundant cloud storage. This ensures that even in the event of a regional disaster, database backups are safe and can be restored, significantly enhancing disaster recovery capabilities.
3. **Scalability and Elasticity:** As databases grow, they require more storage. Cloud storage can scale almost infinitely and on-demand, without requiring manual provisioning or upgrades. This elasticity directly supports databases by providing the necessary storage capacity as data volumes increase, without performance bottlenecks related to storage limitations.
4. **Cost-Effectiveness:** Cloud storage often operates on a pay-as-you-go model, meaning you only pay for the storage you consume. This is more cost-effective than investing in and maintaining expensive on-premises storage infrastructure, especially for databases with fluctuating storage needs.
5. **Integration with Cloud Database Services:** Cloud providers integrate their storage services directly with their managed database offerings (e.g., Amazon RDS using EBS and S3 under the hood). This tight integration allows for seamless provisioning, scaling, and backup of databases, with the storage layer managed entirely by the cloud provider.

Task 5

Advantages of using Cloud-Based Databases (e.g., Azure SQL, Amazon RDS, Google Cloud Spanner)

1. **Scalability and Elasticity:** Cloud databases can be easily scaled up (more CPU/RAM) or scaled out (more instances/shards) with minimal downtime, allowing applications to handle fluctuating workloads and growing data volumes without re-architecting. You can pay for only what you need, scaling resources up or down as demand changes.
2. **High Availability and Durability:** Cloud providers build their database services with high availability in mind, often offering features like automatic failover to replica instances, multi-AZ (Availability Zone) deployments, and robust data replication, ensuring minimal downtime even during hardware failures or maintenance. Data is typically stored with very high durability rates (e.g., 99.999999999% durability).

3. **Reduced Operational Overhead:** The cloud provider manages much of the undifferentiated heavy lifting, including hardware provisioning, software patching, backups, disaster recovery, and often performance tuning. This frees up database administrators and development teams to focus on higher-value tasks.
4. **Cost-Effectiveness:** While large, consistent workloads might make on-premises competitive, for many scenarios, the pay-as-you-go model, elimination of upfront capital expenditure, and reduced operational costs make cloud databases highly cost-effective.
5. **Global Reach and Low Latency:** Cloud databases can be deployed in various geographical regions and availability zones, bringing data closer to users worldwide, which reduces latency and improves application performance for a global user base.
6. **Built-in Security and Compliance:** Cloud providers invest heavily in security, offering advanced features like encryption at rest and in transit, network isolation, identity and access management (IAM), and compliance certifications (e.g., ISO, SOC 2, HIPAA), which can be challenging and expensive to achieve on-premises.

Disadvantages or Challenges with Cloud-Based Databases

1. **Vendor Lock-in:** Relying heavily on a specific cloud provider's proprietary database services can make it difficult and costly to migrate to a different provider or back to an on-premises environment in the future.
2. **Cost Management Complexity:** While often cost-effective, managing cloud costs can be complex. Without careful monitoring and optimization, unexpected charges can accumulate, especially with scaling, data transfer (egress fees), and advanced features.
3. **Performance Tuning and Optimization Limitations:** While cloud providers offer some optimization tools, the "managed" nature means less direct control over the underlying operating system, hardware, and certain database configurations. This can sometimes limit deep-level performance tuning options available to on-premises DBAs.
4. **Security and Compliance Responsibility Sharing:** While cloud providers manage the "security of the cloud," customers are responsible for "security in the cloud." This shared responsibility model means customers must properly configure security settings, manage access controls, and ensure their applications are secure, which can be a source of misconfigurations if not handled carefully.

5. **Latency for On-Premises Integration:** If your application components are split between on-premises and the cloud, network latency between them can impact database performance.
6. **Data Egress Costs:** While data ingress is often free, transferring large amounts of data out of the cloud (egress) can incur significant costs, which needs to be factored into the total cost of ownership.
7. **Internet Dependency:** Access to cloud databases relies entirely on a stable internet connection. Any network disruptions can lead to application downtime.