MODULE - 1

1. Why organizations may choose to use a NoSQL database?

Flexible data model: NoSQL databases offer a flexible data model, allowing organizations to store and retrieve data without adhering to a predefined schema. This flexibility is particularly beneficial when dealing with unstructured or semi-structured data, such as social media posts, sensor data, or user-generated content.

Scalability: NoSQL databases are designed to scale horizontally, meaning they can handle large amounts of data and high traffic loads by distributing data across multiple servers. This scalability enables organizations to accommodate growing data volumes and handle increasing user demands without experiencing performance degradation.

High performance: NoSQL databases are optimized for specific use cases, such as read-heavy or write-heavy workloads. By focusing on specific performance requirements, NoSQL databases can provide faster data access and processing compared to traditional relational databases, which are typically designed for complex transactions.

Big data and real-time analytics: With the rise of big data, organizations need databases that can efficiently process and analyze vast amounts of data in real-time. NoSQL databases excel in this area by offering features like sharding, in-memory processing, and distributed computing, enabling organizations to extract valuable insights from their data in a timely manner.

Cost-effectiveness: NoSQL databases can be more cost-effective for certain use cases compared to traditional relational databases. They often run on commodity hardware and can be deployed on cloud platforms, allowing organizations to scale their infrastructure as needed and pay only for the resources they consume, reducing upfront costs.

2. Explain the value of Relational Database.

Structured data organization: Relational databases store data in a structured manner, using tables with rows and columns. This structure allows for efficient storage and retrieval of data, ensuring data integrity and consistency.

Data integrity and consistency: Relational databases enforce data integrity through the use of constraints, such as primary keys and foreign keys. These constraints ensure that data is accurately and consistently represented, preventing inconsistencies and maintaining data quality.

Powerful querying capabilities: Relational databases provide a powerful query language, typically SQL (Structured Query Language), which allows organizations to retrieve and manipulate data in a structured and efficient manner. SQL provides a standardized and intuitive way to perform complex operations on the data stored in the database.

Relationships and data integrity: Relational databases excel at managing relationships between different sets of data. By using primary keys and foreign keys, organizations can establish relationships between tables, ensuring data integrity and enabling complex data queries that involve multiple related tables.

ACID properties: Relational databases adhere to the ACID (Atomicity, Consistency, Isolation, Durability) properties, which ensure data reliability and transactional consistency. ACID properties guarantee that database operations are executed reliably, even in the presence of failures, and maintain data integrity throughout the transaction process.

3. Write a note on getting at Persistent data.

Persistent data refers to data that is stored for a long time, even after a computer or application is shut down. There are several ways to get at persistent data, depending on where and how it is stored.

1. File System: If the persistent data is stored in a file system, you can access it using file I/O operations. For example, in Python, you can use the built-in `open()` function

to open a file and read or write data.

2. Databases: If the persistent data is stored in a database, you can use SQL or other database-specific query languages to access it. For example, in MySQL, you can use the `SELECT` statement to retrieve data from a table.

3. APIs: If the persistent data is stored in a remote server, you can use APIs to retrieve it. APIs (Application Programming Interfaces) are interfaces that allow different software systems to communicate with each other. For example, you can use the Twitter API to retrieve tweets from a user's timeline.

4. Cloud Storage: If the persistent data is stored in a cloud storage service like Amazon S3 or Google Cloud Storage, you can use APIs or SDKs provided by the service to access the data. For example, Amazon S3 provides a REST API that allows you to interact with objects stored in S3 buckets.

4. Explain Concurrency and Integration in NoSql.

Concurrency:

Simultaneous Operations: NoSQL databases can handle multiple users or processes accessing and modifying data at the same time without causing conflicts.

Conflict Resolution: When conflicts occur, the database resolves them by comparing changes and applying appropriate strategies, like choosing the most recent update or using conflict resolution algorithms.

Distributed Locking: Some NoSQL databases use techniques to ensure that only one user or process can modify a specific piece of data at a time, maintaining data integrity.

Integration:

APIs: NoSQL databases provide interfaces that allow developers to easily integrate the database with other parts of their application by following a set of predefined rules.

Connectors: NoSQL databases offer ready-made tools or extensions that simplify the process of connecting the database to different programming languages, frameworks, and tools.

Compatibility: NoSQL databases are designed to work well with various data processing and analytics tools, allowing seamless integration and interoperability with the existing technology ecosystem.

5. Explain the Standard Model.

While there is no universal "Standard Model" in NoSQL databases, some databases have become popular and are widely used, and they share some common characteristics. These databases have been referred to as the "Big Four" NoSQL databases: MongoDB, Cassandra, Redis, and Couchbase. While each of these databases has its own unique features, they share some common characteristics that have become somewhat of a standard in the NoSQL world.

1. Document-oriented: This means that data is stored in documents, which can be thought of as key-value pairs, but with more structured data in the value. Each document can have a different structure, and the schema is flexible.
2. Column-family: This means that data is stored in columns, which are grouped into column families. Each column can have a different data type, and the schema is flexible.
3. Key-value: This means that data is stored as key-value pairs, with no structure enforced on the value.
4. Distributed: All of these databases are designed to be highly scalable and can be run on distributed systems.
5. Open-source: All of these databases are open-source and have active communities contributing to their development and support.

6. Explain Application database and Integration database.

Application Database:

Purpose: An application database is like a storage space specifically designed for one particular application or system. It holds the data that the application needs to function properly.

Structure: The database has a predefined structure with tables and columns that match the data model of the application. It makes sure the data is organized and consistent.

Transactions: The database supports transactions, which means it can handle operations that should happen together as a single unit. This ensures that changes to the data are reliable and consistent.

Integration Database:

Purpose: An integration database serves as a central hub where data from different sources or systems is brought together. It helps combine and harmonize data from various places.

Data Integration: The database collects data from different sources, like application databases or external systems, and transforms it into a common format. This makes it easier to analyze and understand the data.

Reporting and Analysis: The integration database is optimized for generating reports and performing data analysis. It provides tools and functions that help extract valuable insights from the combined data.

7. Write a note on Emergence of NoSql.

NoSQL (Not Only SQL) databases emerged as an alternative to traditional relational databases in the late 2000s, and have gained popularity due to their ability to handle large volumes of unstructured and semi-structured data, their scalability, and their ability to provide high performance and availability.

The emergence of NoSQL was largely driven by the need to handle the massive amounts of data generated by the growth of the internet and the rise of social media, mobile devices, and IoT (Internet of Things) devices.

NoSQL databases, on the other hand, were designed to handle these types of data by providing flexible schemas, horizontal scalability, and distributed architectures that can handle massive volumes of data across many nodes or clusters.

Some of the key features of NoSQL databases include:

1. Flexible schemas: Unlike relational databases, which require a fixed schema, NoSQL databases can handle dynamic and evolving data structures, making them well-suited for applications that require frequent schema changes.
2. Horizontal scalability: NoSQL databases can easily scale horizontally by adding more nodes or clusters, which makes them well-suited for handling large volumes of data and high-traffic applications.
3. Distributed architectures: NoSQL databases are designed to run across multiple nodes or clusters, which provides high availability and fault tolerance.
4. High performance: NoSQL databases are optimized for high-speed data access and can provide sub-millisecond response times for read and write operations.

8. Explain the types of data model.

Hierarchical Data Model:

- ✓ In the hierarchical data model, data is organized in a tree-like structure with parent-child relationships.
- ✓ Each parent can have multiple children, but each child can only have one parent.

Network Data Model:

- ✓ The network data model is an extension of the hierarchical model.
- ✓ It allows for more complex relationships by allowing each child to have multiple parents, forming a network-like structure.
- ✓ It is suitable for representing many-to-many relationships and provides more flexibility in data organization.

Relational Data Model:

- ✓ The relational data model is based on tables, with each table representing an entity and each column representing an attribute of that entity.
- ✓ Relationships between entities are established through keys, such as primary keys and foreign keys.

Object-Oriented Data Model:

- ✓ The object-oriented data model represents data as objects, similar to how objects are represented in object-oriented programming.
- ✓ Each object has properties (attributes) and behaviors (methods) associated with it.
- ✓ It supports encapsulation, inheritance, and polymorphism, allowing for more complex data modeling and representation.

Document Data Model:

- ✓ The document data model is used by NoSQL databases like MongoDB.
- ✓ It stores data in flexible, self-describing document formats like JSON or BSON.

9. Write a note on Matrialized View.

Purpose: A materialized view is a database object that stores the results of a pre-computed query as a physical table.

Derived from Queries: It is derived from one or more queries, typically complex or time-consuming ones, executed against the underlying data in the database.

Data Storage: Unlike a regular view that simply provides a logical representation of data, a materialized view stores the actual data in a physical table, which is updated periodically or on-demand.

Improved Query Performance: By pre-computing and storing the results of expensive queries, materialized views enhance query performance. When a query is executed against a materialized view, it retrieves the pre-computed data instead of re-computing it from scratch, reducing execution time and resource consumption.

Data Aggregation and Transformation: Materialized views can be used to aggregate data, perform calculations, or apply transformations to the underlying data. This

allows for faster retrieval of summarized or transformed information, saving processing time and improving query response times.

10. Explain the Schemaless Database

Definition: A schemaless database, also known as a schema-free database, is a type of database that does not require a predefined structure or schema for storing data.

Flexible Data Model: In a schemaless database, data can be stored in a flexible and dynamic manner without adhering to a fixed schema. Each record or document can have its own unique structure, allowing for variations in data attributes and types.

No Fixed Schema: Unlike traditional relational databases where a fixed schema is defined in advance, schemaless databases do not enforce a rigid structure for data. This means that fields or attributes can be added, modified, or removed from records without requiring any modifications to the database schema.

Easy Adaptability: Schemaless databases are highly adaptable and flexible. They can accommodate evolving data requirements and allow for seamless integration of new data attributes or changes to existing ones without requiring schema modifications or data migrations.

Agility and Speed: The schemaless approach offers agility and speed in development and data management. It allows developers to iterate quickly and make changes to the data model without the constraints of a fixed schema. This can lead to faster development cycles and easier data manipulation.