ENSEM IMP DATABASE MANAGMENT SYSTEM UNIT – 5

Q.1] BASE Transactions ensures the properties like Basically Available, Soft State, Eventual Consistency. What is soft state of any system, how it is depend on Eventual consistency property?

ANS:

Soft State in a System:

1. **Flexible State Representation:**
   - **Soft state refers to a system's flexible representation of data, where the system may allow temporary inconsistency or imprecision in the data. It's a design choice that accepts temporary inconsistencies for performance or availability reasons.**
2. **Permissible Inconsistencies:**
   - **It allows data to be temporarily inconsistent or imprecise, accepting that the data may not be up-to-date at all times. This can be due to factors like network delays, system loads, or replication latencies in a distributed system.**
3. **Transient Nature:**
   - **Soft state recognizes that the data inconsistencies are temporary and expected to converge to a consistent state over time as the system processes, synchronizes, or updates the data.**

Dependence of Soft State on Eventual Consistency:

1. **Eventual Consistency:**
   - **Eventual consistency is a property of distributed systems that ensures that given enough time and no further updates, all replicas of the data in a distributed system will converge to the same value or a consistent state.**
2. **Relation to Soft State:**
   - **Soft state is closely related to the principle of eventual consistency. Soft state systems often accept temporary data inconsistencies, knowing that over time, due to the eventual consistency property, the data will converge to a consistent state across the distributed system.**
3. **Balancing Trade-offs:**
   - **Soft state acknowledges that achieving immediate consistency might incur heavy costs in terms of performance or availability. Therefore, it tolerates temporary inconsistencies, relying on the eventual consistency property to rectify these temporary variations and converge to a consistent state over time.**
4. **Trade-off between Consistency and Performance:**
   - **Soft state is about balancing the trade-off between consistency and performance, recognizing that immediate consistency might hinder system performance or availability. Instead, it allows transient inconsistencies with the expectation that eventual consistency will resolve them.**

Q.2] Enlist the different types of NOSQL databases and explain with suitable examples

ANS:

1. **Document Stores:**
    - **These databases store and retrieve data in the form of documents, typically in JSON or BSON formats. Each document is a self-descriptive unit containing key-value pairs.**

Example: MongoDB

- **MongoDB stores data in a format similar to JSON, allowing flexibility and scalability. For instance, in MongoDB, data for a user might be stored as a document containing various fields like name, age, address, and contact details.**

2. **Key-Value Stores:**
    - **Key-Value stores are the simplest NoSQL databases, storing data in a schema-less way with a unique key attached to the data.**

Example: Redis

- **Redis is an in-memory key-value store often used for caching and real-time analytics. It can store various data types associated with unique keys, such as strings, lists, sets, and hashes.**

3. **Column-Family Stores:**
    - **These databases organize data in columns rather than rows, well-suited for analyzing and managing large amounts of data.**

Example: Apache Cassandra

- **Cassandra is designed for handling large amounts of structured data across multiple commodity servers. It allows for the efficient storage and retrieval of columns associated with rows.**

4. **Graph Databases:**
    - **Graph databases are used to store data in terms of entities (nodes), relationships (edges), and properties. They're ideal for relationship-centric data.**

Example: Neo4j

- **Neo4j is a graph database that excels in handling complex relationships. It represents data as nodes and edges, making it efficient for scenarios like social networks, recommendation systems, and network analysis.**

5. **Wide-Column Stores:**
    - **Similar to column-family stores, wide-column stores organize data in columns, offering flexibility in terms of adding columns dynamically to a row.**

Example: Apache HBase

- **HBase, part of the Apache Hadoop project, provides a fault-tolerant, scalable, distributed, and consistent database suited for storing sparse data in a wide-column format.**

Q.3] What is structured and unstructured data. Explain with example.
ANS: Structured and unstructured data represent different formats and organization of information, impacting how it's stored, processed, and analyzed.
Structured Data:

- **Definition: Structured data is highly organized and formatted in a way that's easily processed by machines. It fits neatly into predefined schemas, typically stored in relational databases, and is easy to query using SQL.**
- **Example: Consider a table in a database storing employee information:**

Employee ID	Name	Age	Department
001	John	30	Engineering
002	Sarah	28	Marketing
003	Mike	35	Sales

- **Characteristics:**
  - **Organized into rows and columns.**
  - **Follows a specific schema or structure.**
  - **Can be queried using standardized methods like SQL.**
  - **Well-suited for analysis, reporting, and easy to process by machines.**

Unstructured Data:

- **Definition: Unstructured data lacks a predefined structure and organization, making it more challenging to process and analyze using traditional methods. It can be in the form of text, images, videos, audio, social media posts, etc.**
- **Example: Social media posts, like tweets, are unstructured data. For instance:**

***Tweet*: "Enjoying a fantastic cup of coffee at the local cafe ☕ #coffeeholic #relaxation"**

- **Characteristics:**
  - **Doesn't conform to a fixed schema.**
  - **Varied formats such as text, images, videos, audio, etc.**
  - **Requires specialized tools for processing and analysis.**
  - **Contains valuable insights but may require advanced techniques like natural language processing, image recognition, or sentiment analysis.**

Key Differences:

1. **Organization: Structured data is highly organized into rows and columns, while unstructured data lacks this specific organization.**
2. **Processing: Structured data is easy to process using traditional methods (SQL), while unstructured data often requires specialized tools and techniques for analysis.**
3. **Source: Structured data often comes from databases and spreadsheets, while unstructured data can be from social media, multimedia, sensor data, etc.**
4. **Ease of Analysis: Structured data is easily analyzed and queried, while extracting insights from unstructured data might require more complex algorithms and tools.**

Q.4] Explain how NOSQL databases are different than relational databases? Describe in detail the key value store NOSQL data model with example.

ANS:

Differences between NoSQL and Relational Databases:

1. **Data Model:**
   - **Relational Databases: These use a structured data model with tables, rows, and columns, adhering to a fixed schema.**
   - **NoSQL Databases: NoSQL databases offer different data models like key-value stores, document stores, column-family stores, graph databases, etc., allowing more flexibility and scalability.**
2. **Schema Flexibility:**
   - **Relational Databases: Relational databases have a fixed schema that requires predefined table structures.**
   - **NoSQL Databases: NoSQL databases offer schema flexibility, enabling the addition or modification of data without altering the entire database schema.**
3. **Scalability:**
   - **Relational Databases: Traditionally, relational databases have faced challenges in scaling horizontally (across multiple servers) due to their rigid structure.**
   - **NoSQL Databases: NoSQL databases are often designed to be horizontally scalable, making them suitable for handling large amounts of distributed data across multiple servers.**
4. **Consistency and ACID Properties:**
   - **Relational Databases: Relational databases prioritize consistency and transactional integrity, adhering to ACID properties (Atomicity, Consistency, Isolation, Durability).**
   - **NoSQL Databases: NoSQL databases often prioritize availability and partition tolerance (the CAP theorem) over strict consistency, offering BASE properties (Basically Available, Soft state, Eventually consistent).**

Key-Value Store in NoSQL:

The key-value store is one of the fundamental NoSQL database models:

- **Data Model: Key-value stores represent data as a collection of key-value pairs, where each unique key is associated with a value. It's a simple and efficient way to store and retrieve data.**
- **Example: Redis**

Data Structure: Consider a basic representation in Redis where keys are associated with values:

mathematicaCopy code

Key: "user:1" Value: "{ name: 'John', age: 30, city: 'New York' }" Key: "product:101" Value: "{ name: 'Mobile Phone', price: 500, brand: 'Apple' }"

- **Characteristics:**
  - **Simplicity: It offers a straightforward way to store and retrieve data with a unique key.**
  - **Schema-less: The data model doesn't enforce any fixed schema, allowing flexibility in the structure of stored values.**
  - **High Performance: Key-value stores are optimized for fast retrieval based on keys.**

- **Use Cases:**
  - **Caching: Storing frequently accessed data for quicker access.**
  - **Session Management: Keeping session-related information.**
  - **Metadata Storage: Storing metadata in applications or web services.**

Q.5] Explain BASE properties with its significance. How soft state of system is depending on Eventual consistency property?

ANS:

BASE Properties:

BASE properties are an alternative approach to database consistency, in contrast to the ACID (Atomicity, Consistency, Isolation, Durability) properties followed by traditional relational databases. BASE stands for:

1. **Basically Available: The system remains operational and available for use despite potential failures or inconsistencies. It might sacrifice immediate consistency for availability.**
2. **Soft State: The data within the system might be temporarily inconsistent due to factors like network latency or replication delays, but it's expected to converge to consistency eventually.**
3. **Eventual Consistency: The system, given enough time and no further updates, will converge to a consistent state across all replicas or nodes in a distributed system.**

Significance of BASE:

- **High Availability: BASE prioritizes availability, ensuring that the system remains accessible even in the face of partial failures or temporary inconsistencies. This is crucial for systems that can't afford downtime or need to be constantly available.**
- **Scalability: BASE properties align well with the needs of distributed and scalable systems. By allowing temporary inconsistencies, it facilitates better scalability and performance.**
- **Flexibility: BASE acknowledges that achieving strict consistency in a distributed system might hinder performance and scalability. Therefore, it allows for more flexible trade-offs between consistency and availability.**

Soft State and Eventual Consistency:

- **Soft State: Soft state refers to the system's tolerance for temporary inconsistencies. It accepts that data may be temporarily inconsistent due to various factors like network latency or replication delays.**
- **Eventual Consistency: Eventual consistency ensures that, given enough time and no further updates, all replicas of the data across a distributed system will converge to a consistent state.**

Dependence of Soft State on Eventual Consistency:

- **Soft state relies on the principle of eventual consistency to resolve temporary data inconsistencies over time. While the system allows for temporary inconsistencies, it's expected that, given enough time, these inconsistencies will be resolved, and the data will converge to a consistent state across the distributed system.**

Q.6] Explain the CAP theorem referred during the development of any distributed application.

ANS: The CAP theorem, also known as Brewer's theorem, is a fundamental concept in distributed systems, describing the trade-offs between three key properties: Consistency, Availability, and Partition Tolerance. It states that in a distributed system, it's impossible to guarantee all three of these properties simultaneously.

1. **Consistency (C):**
    - **Every read from the database gets the most recent write or an error. In a consistent system, all nodes see the same data at the same time.**
2. **Availability (A):**
    - **Every request made to the database gets a response, without guaranteeing the most recent data. An available system will always respond to a request, even if the data might be slightly outdated or inconsistent.**
3. **Partition Tolerance (P):**
    - **The system continues to operate despite network partitions (communication failures) that cause certain nodes to be unreachable by others. In a partition-tolerant system, nodes can still operate and process requests independently when communication breaks down.**

The CAP theorem states that in a distributed system, you can only have two out of the three properties—Consistency, Availability, and Partition Tolerance—at any given time.

For example:

- **In a scenario where a network experiences a partition (P), the system must choose between maintaining Consistency (C) or Availability (A). If it chooses to remain consistent across all nodes (C), it might sacrifice availability by not responding to requests until the network issue is resolved.**
- **Conversely, if it prioritizes Availability (A) during a partition, the system might continue responding to requests but might provide slightly outdated or inconsistent data due to the partition.**

Significance in Distributed Application Development:

Understanding the CAP theorem is crucial when designing and developing distributed systems:

- **Architectural Decisions: It influences the design choices in distributed systems, guiding developers to make decisions based on the specific needs and priorities of the application.**
- **Trade-offs: It helps in recognizing and managing the trade-offs between consistency, availability, and partition tolerance. Developers need to make informed decisions about which trade-off best suits their system's requirements.**
- **System Design: It's fundamental in designing systems to ensure that the architecture aligns with the desired balance between the three properties based on the application's specific use case and needs.**

Q.7] Analyze the use of NOSQL databases in current social networking environment also explain need of NOSQL databases in social networking environment over RDBMS

ANS:

Use of NoSQL Databases in Social Networking:

NoSQL databases have found extensive use in the social networking environment due to their ability to handle vast amounts of unstructured and semi-structured data, support high scalability, and accommodate the dynamic nature of social media platforms.

Reasons for NoSQL Use in Social Networking:

1. **Scalability: Social networking platforms experience huge volumes of data with a rapidly increasing user base. NoSQL databases, with their ability to scale horizontally, can easily handle the high volume and unpredictable growth of data, accommodating the dynamic nature of social media traffic.**
2. **Unstructured Data: Social networking platforms deal with diverse types of data such as text, images, videos, and user interactions. NoSQL databases, particularly document stores and key-value stores, are well-suited for handling these diverse and unstructured data types efficiently.**
3. **Flexible Schema: NoSQL databases provide schema flexibility, allowing for easy accommodation of changes in data structure. Social media data is constantly evolving, and NoSQL's schema-less approach allows for easy updates without disrupting the system.**
4. **High Performance: For social media platforms that require rapid access to specific elements of data, key-value stores offer high performance in retrieving and storing individual data points associated with unique keys, aiding in rapid access and updates.**
5. **Geographically Distributed Data: Social networks often have a global user base. NoSQL databases are adept at managing and replicating data across multiple geographies, supporting local access and ensuring low-latency responses.**

Need for NoSQL Databases over RDBMS in Social Networking:

1. **Schema Flexibility: Social networking platforms witness a constant evolution in data structure. NoSQL's flexible schema allows seamless integration of new data types and attributes without requiring complex schema modifications, unlike RDBMS.**
2. **Handling Unstructured Data: Social networking platforms deal with a myriad of unstructured and semi-structured data types. NoSQL's ability to handle these data types efficiently makes it a more suitable choice compared to RDBMS designed for structured data.**
3. **Scalability and Performance: The dynamic and unpredictable nature of social media demands high scalability and performance. NoSQL databases, designed for horizontal scaling and optimized for high performance, provide an edge over the rigid scaling limits of traditional RDBMS.**
4. **Elasticity and Adaptability: NoSQL databases are more adaptable to changes in traffic, user interactions, and data types. They can scale seamlessly and handle fluctuating loads, offering the elasticity needed for social networking platforms.**

Q.8] Explain the difference between SQL and NOSQL database.

ANS: SQL (Structured Query Language) and NoSQL (Not Only SQL) databases differ in various aspects, from data models to query languages and use cases. Here are the key differences:

Data Model:

- **SQL (Relational Databases): SQL databases use a structured data model based on tables with rows and columns. They enforce a rigid schema and adhere to ACID (Atomicity, Consistency, Isolation, Durability) properties.**
- **NoSQL Databases: NoSQL databases offer various data models such as document stores, key-value stores, column-family stores, and graph databases. They provide more flexibility in data structures, allowing for schema-less or flexible schemas.**

Query Language:

- **SQL (Relational Databases): SQL databases use the SQL query language, a standardized language for interacting with relational databases. SQL is based on a declarative approach, allowing users to specify what data they want without needing to define how to retrieve it.**
- **NoSQL Databases: NoSQL databases use various query languages or interfaces. While some NoSQL databases have their query languages, others might have simpler APIs or different query mechanisms based on the database type.**

Scaling and Performance:

- **SQL (Relational Databases): Traditional SQL databases have faced challenges in scaling horizontally, particularly in large-scale distributed systems. They are designed for consistency and offer ACID properties, prioritizing strict data consistency over scaling.**
- **NoSQL Databases: NoSQL databases are often designed for horizontal scaling. They prioritize availability and partition tolerance, focusing on BASE properties (Basically Available, Soft state, Eventually consistent) over strict consistency, which allows for better scalability and performance.**

Use Cases:

- **SQL (Relational Databases): Well-suited for applications requiring strict consistency, integrity, and complex transactions. Commonly used in traditional enterprise systems, financial applications, and scenarios where strict data relations are crucial.**
- **NoSQL Databases: NoSQL databases are preferred for applications that demand high scalability, handling large volumes of unstructured or semi-structured data, such as social media, e-commerce, real-time analytics, IoT, and content management systems.**

Schema:

- **SQL (Relational Databases): Relational databases enforce a fixed schema. Any changes to the schema often require alterations to the entire database, which can be complex and time-consuming.**
- **NoSQL Databases: NoSQL databases often provide schema flexibility. They allow for easy additions or modifications to data without requiring changes to the entire database schema.**

Q.9] List the different NOSQL data models. Explain document store NOSQL data model with example.

ANS: NoSQL databases offer various data models to store and manage data. The major types of NoSQL data models include:

1. **Document Stores:**
   - **Document stores organize data into documents, usually in JSON, XML, or BSON formats, where each document contains key-value pairs.**
2. **Key-Value Stores:**
   - **Key-value stores manage data as unique keys associated with values, which can be simple values, objects, or complex data structures.**
3. **Column-Family Stores:**
   - **Column-family stores organize data into columns, akin to tables in relational databases, grouping related data together.**
4. **Graph Databases:**
   - **Graph databases manage data in nodes (entities) and edges (relationships), allowing for efficient handling of complex relationships between data points.**

Document Store NoSQL Data Model:

- **Overview: Document stores structure data in flexible and self-describing documents. Each document, often in JSON or BSON format, contains key-value pairs and represents a distinct entity or record.**
- **Example: MongoDB**

Data Structure: Consider a basic representation of a document in MongoDB:

```
{
  "_id": 1,
  "name": "John Doe",
  "age": 30,
  "email": "john@example.com",
  "address": {
    "street": "123 Main St",
    "city": "Anytown",
    "zipcode": "12345"
  },
  "interests": ["hiking", "photography", "reading"]
}
```

- **Characteristics:**
  - **Schema-less: Documents within a collection can have different structures. The model allows for schema flexibility, enabling each document to have different fields or structures.**
  - **Self-Describing: Each document holds its schema within itself, providing a self-contained representation of the data.**
  - **Nesting and Arrays: Documents can have nested structures and arrays, enabling complex data representation.**
  - **Efficient Retrieval: Well-suited for applications needing quick access to complete entities, allowing retrieval of the entire document in a single operation.**

Q.10] State and explain the concept of CAP theorem and BASE properties with example.

ANS: The CAP theorem and BASE properties are fundamental concepts in distributed systems that address the trade-offs between consistency, availability, and partition tolerance.

CAP Theorem:

- **Consistency (C): Ensures that all nodes in a distributed system have the same data at the same time.**
- **Availability (A): Guarantees that every request receives a response, even if it's an outdated or inconsistent one.**
- **Partition Tolerance (P): Allows the system to continue operating despite network partitions or communication failures.**

The CAP theorem states that in a distributed system, it's impossible to simultaneously achieve all three properties (C, A, and P). When a partition occurs, a system has to choose between maintaining consistency (C) or ensuring availability (A).

Example: CAP Theorem in Distributed Databases

Consider a distributed database with multiple nodes across different locations. If a network partition occurs (P), the system has to decide:

- **Consistency over Availability (CP): If the system prioritizes consistency, it might restrict access to some data until consistency is restored across all nodes. Users might experience unavailability or delays until the partition is resolved, ensuring that all nodes have consistent data.**
- **Availability over Consistency (AP): If the system prioritizes availability, it will continue to serve requests even during a partition. However, this might result in users accessing slightly outdated or inconsistent data on different nodes until the partition is resolved.**

BASE Properties:

BASE properties are an alternative to ACID properties in databases, focusing on trade-offs in distributed systems:

- **Basically Available: The system remains available and operational despite failures, sacrificing immediate consistency.**
- **Soft State: Acknowledges that the system may temporarily exhibit inconsistencies due to factors like replication delays or network latency.**
- **Eventual Consistency: Over time, given no further updates and a return to a stable state, all replicas or nodes in the system will converge to a consistent state.**

Example: BASE in a Distributed System

Consider a distributed system for a social media platform:

- **Basically Available: Even if certain nodes or parts of the system face issues, the platform remains accessible to users. It might provide slightly outdated information or experience inconsistencies but will continue to operate.**
- **Soft State: Due to network delays or replication latencies, some nodes might exhibit temporary inconsistencies. Users might see different post updates in different parts of the system for a brief time.**
- **Eventual Consistency: Given enough time and no further updates, the data across all nodes will eventually converge to a consistent state, ensuring that all users eventually see the same information.**

Q.11] Explain following NOSQL database types with examples and also state the scenario where it is useful

i) Column-oriented

ii) Graph

iii) Document -oriented

ANS: here's a breakdown of the different types of NoSQL databases along with examples and scenarios where they are useful:

i) Column-oriented:

- **Examples: Apache Cassandra, HBase**
- **Scenario: Ideal for applications requiring fast read and write operations on large amounts of data, especially when dealing with time-series data, analytics, and data warehousing. For example, Cassandra is commonly used in applications like social media analytics, IoT data management, and real-time recommendation systems.**

ii) Graph:

- **Examples: Neo4j, Amazon Neptune**
- **Scenario: Suitable for scenarios where relationships between data points are complex and need to be represented efficiently. Graph databases excel in applications like social networks, recommendation engines, fraud detection, and network and IT operations. For instance, Neo4j is used in social networking platforms for managing friend connections and in logistics for route optimization.**

iii) Document-oriented:

- **Examples: MongoDB, Couchbase**
- **Scenario: Great for storing and managing semi-structured or unstructured data, such as documents, JSON, XML, etc. Document databases are versatile and can be used in various scenarios like content management systems, e-commerce platforms, mobile app development, and real-time analytics. For example, MongoDB is commonly used in blogging platforms for content storage, in e-commerce for product catalogs, and in IoT applications for sensor data storage.**

Q.12] Describe distributed database. Explain System architecture of distributed transaction.
ANS: here's a simplified explanation of distributed databases and the system architecture of distributed transactions:

Distributed Database:

1. **Definition: A distributed database is a collection of multiple interconnected databases spread across different locations but logically connected to function as a single unit.**
2. **Data Distribution: Data is distributed across multiple nodes or servers in the network, allowing for improved scalability and fault tolerance.**
3. **Replication: Some or all data may be replicated across multiple nodes to ensure redundancy and fault tolerance.**
4. **Data Access: Users can access and query the distributed database as if it were a single database, hiding the complexities of data distribution and replication.**
5. **Coordination: Mechanisms for coordinating data access, consistency, and transactions are crucial for maintaining integrity and reliability in distributed environments.**

System Architecture of Distributed Transactions:

1. **Transaction Management: Distributed transactions involve multiple operations that need to be executed as a single unit to maintain data consistency.**
2. **Participants: Each node participating in the distributed transaction is called a participant. These participants may be spread across different locations and may run different database management systems.**
3. **Transaction Coordinator: A central entity, known as the transaction coordinator, coordinates the execution of the distributed transaction.**
4. **Two-Phase Commit Protocol (2PC):**
    - **Phase 1 (Voting): The transaction coordinator asks each participant if they are ready to commit. Participants respond with a vote indicating whether they can commit or abort.**
    - **Phase 2 (Decision): Based on the votes received, the transaction coordinator decides whether to commit or abort the transaction. If all participants vote to commit, the coordinator sends a commit message to all participants. Otherwise, it sends an abort message.**
5. **Atomicity and Durability: The 2PC protocol ensures that either all participants commit or none commit, maintaining atomicity. Additionally, committed transactions are durably stored on disk to ensure durability.**
6. **Fault Tolerance: The distributed transaction system must be resilient to failures, such as participant crashes or network partitions. Mechanisms like timeout mechanisms and participant recovery protocols help ensure fault tolerance.**

Q.13] Explain following types of data with example
i) Structured
ii) Semi-structured
iii) Unstructured
ANS: here's a simple and easy explanation of each type of data with examples:
i) Structured Data:

1. **Definition: Data that is organized into a specific format with a well-defined schema.**
2. **Example:**
    - **A spreadsheet containing columns for "Name," "Age," "Gender," and "City."**
    - **Relational databases with tables and predefined relationships between them.**
    - **Data stored in tables in SQL databases.**
3. **Characteristics:**
    - **Easy to search, organize, and analyze.**
    - **Follows a clear and consistent structure.**
    - **Typically found in traditional databases.**

ii) Semi-structured Data:

1. **Definition: Data that does not have a rigid structure like structured data but still contains some organizational properties.**
2. **Example:**
    - **JSON (JavaScript Object Notation) files.**
    - **XML (Extensible Markup Language) documents.**
    - **CSV (Comma-Separated Values) files with varying column lengths.**
3. **Characteristics:**
    - **Contains tags, markers, or keys that provide some structure.**
    - **Can be partially organized but may lack a strict schema.**
    - **More flexible than structured data but less so than unstructured data.**

iii) Unstructured Data:

1. **Definition: Data that lacks a predefined structure and organization.**
2. **Example:**
    - **Text documents such as emails, social media posts, and articles.**
    - **Images, videos, and audio recordings.**
    - **Sensor data, like raw output from IoT devices.**
3. **Characteristics:**
    - **No predefined schema or organization.**
    - **Often in human-readable format but lacks clear structure for machines.**
    - **Requires advanced techniques like natural language processing (NLP) and machine learning for analysis and understanding.**

Q.14] Explain BASE Properties of NOSQL Database.
ANS: BASE properties are key characteristics of NoSQL databases, which stand in contrast to the ACID properties of traditional relational databases. BASE stands for:

1. **Basically Available: This means that the system remains operational even in the face of partial failures. Instead of guaranteeing immediate consistency, BASE allows for temporary inconsistency. It prioritizes availability over consistency, ensuring that users can still access the system and perform operations even if certain parts of the system are unavailable or experiencing issues.**
2. **Soft state: NoSQL databases don't require all data to be in a consistent state at all times. This allows for flexibility in handling data, with the system accommodating varying states of consistency as needed. Soft state implies that the system's state can change over time as updates are made, and there might be temporary inconsistencies between different replicas or shards of the data.**
3. **Eventual consistency: Unlike the immediate consistency provided by ACID databases, NoSQL databases offer eventual consistency. This means that the system will eventually become consistent once all updates cease, without requiring immediate synchronization of all data across all nodes. It allows for asynchronous replication and propagation of updates, enabling high availability and scalability while tolerating temporary inconsistencies.**

Let's break down each point further:

- **Basically Available:**
    - **NoSQL databases prioritize availability, ensuring that users can access the system and perform operations even under adverse conditions such as network partitions or hardware failures.**
    - **It allows for decentralized architectures where different parts of the system can continue operating independently, ensuring that the system as a whole remains functional.**
    - **Trade-offs may include potential data inconsistencies or the need for conflict resolution mechanisms.**
- **Soft State:**
    - **Soft state acknowledges that data consistency can vary over time, allowing for transient inconsistencies during updates or replication processes.**
    - **This flexibility enables systems to handle concurrent updates and distributed transactions more efficiently, without enforcing strict consistency requirements at all times.**

- **Soft state encourages the use of conflict resolution strategies and eventual reconciliation to ensure eventual consistency across the distributed system.**

- **Eventual Consistency:**
  - **Eventual consistency guarantees that, given a period of time with no updates, all replicas of data will converge to a consistent state.**
  - **Instead of enforcing immediate consistency across all nodes, NoSQL databases allow for temporary inconsistencies between replicas, prioritizing availability and partition tolerance.**
  - **This approach facilitates horizontal scalability and fault tolerance, as data updates can be propagated asynchronously across distributed nodes without blocking operations or requiring synchronous coordination.**

Q.15] Explain Document Based and Key value data model of NOSQL Database.

ANS: here's a simple breakdown of the Document-Based and Key-Value data models in NoSQL databases:

Document-Based Model:

1. **Structure: Data is stored in flexible, schema-less documents, typically using formats like JSON or BSON.**
2. **Document: Each record in the database is a document, which can contain nested structures, arrays, or other key-value pairs.**
3. **Example: Think of a document as a file containing all the information about a single entity, like a user profile or a product listing.**
4. **Flexibility: Allows for easy storage of heterogeneous data without predefined schemas, making it suitable for applications with evolving data requirements.**
5. **Querying: Queries are often performed using document-oriented query languages like MongoDB's query language, which can handle complex nested structures and search criteria.**

Key-Value Model:

1. **Basic Structure: Data is stored as a collection of key-value pairs.**
2. **Keys: Each piece of data is identified by a unique key.**
3. **Values: Corresponding to each key, there is a value, which can be any data type (string, number, JSON object, etc.).**
4. **Example: Imagine it like a dictionary where you look up a word (key) to find its definition (value).**
5. **Scalability: Provides excellent scalability as it's simple to distribute data across multiple nodes.**
6. **Performance: Offers fast read and write operations since data access is direct based on the key.**
7. **Use Cases: Often used for caching, session management, and simple data retrieval tasks where the structure of the data is straightforward.**

Q.16] Explain the CRUD operations used in MongoDB with example.
ANS: Sure, I'll break it down for you:

1. **Create (C):**
    - **This operation is used to insert new documents into a MongoDB collection.**
    - **Example: Let's say we have a collection named "users". To create a new user document, you can execute a MongoDB insert operation like this:**

```
db.users.insertOne({ name: "John", age: 30, email: "john@example.com" })
```

2. **Read (R):**

- **This operation is used to retrieve documents from a MongoDB collection.**
- **Example: To read all user documents from the "users" collection, you can use the find() method:**

```
db.users.find()
```

3. **Update (U):**

- **This operation is used to modify existing documents in a MongoDB collection.**
- **Example: Let's say we want to update John's age to 31. We can use the updateOne() method like this:**

```
db.users.updateOne({ name: "John" }, { $set: { age: 31 } })
```

4. **Delete (D):**

- **This operation is used to remove documents from a MongoDB collection.**
- **Example: If we want to delete the user document with the name "John", we can use the deleteOne() method:**

```
db.users.deleteOne({ name: "John" })
```

These are the basic CRUD operations used in MongoDB to Create, Read, Update, and Delete documents within collections.

Q.17] Explain Map Reduce with example.

ANS: Here's a simple explanation of MapReduce with an example:

1. **Map:**
    - **Breaks down a problem into smaller tasks and assigns them to different worker nodes.**
    - **Each worker node processes its assigned task independently.**
    - **Example: Let's say you have a list of numbers and you want to square each number. The "map" step would assign each number to a different worker node, and each node would square its assigned number.**
2. **Shuffle and Sort:**
    - **Groups and sorts the output of the Map step based on a key.**
    - **Example: If multiple worker nodes square numbers, the shuffle and sort step will organize the squared numbers based on their original keys (the input numbers).**
3. **Reduce:**
    - **Aggregates and processes the output of the Shuffle and Sort step.**
    - **Combines the results to produce a final output.**
    - **Example: In our case, the "reduce" step would sum up all the squared numbers produced by the map step to get the final result.**

Example: Let's say we have a list of numbers: [1, 2, 3, 4, 5] We want to square each number and then sum the squares.

- **Map Step:**
    - **Each number is sent to a worker node.**
    - **Worker nodes square each number.**
    - **Output: [(1, 1), (2, 4), (3, 9), (4, 16), (5, 25)]**
- **Shuffle and Sort Step:**
    - **Groups and sorts the output based on keys.**
    - **Output: [(1, [1]), (2, [4]), (3, [9]), (4, [16]), (5, [25])]**
- **Reduce Step:**
    - **Takes each group and sums the squares.**
    - **Output: [(1, 1), (2, 4), (3, 9), (4, 16), (5, 25)]**
    - **Final result: Sum of squares = 1 + 4 + 9 + 16 + 25 = 55**