



**Acharya Institute of Technology**

**Bangalore 560107.**

Department of Computer Science and Engineering

**Assignment – 1**

Sub with Code: NOSQL DATABASE (21CS745)

Semester/Section:7 (A, B, C, D)

- 1) What is NoSQL? Explain briefly about aggregate data models with a neat diagram?
- 2) Explain briefly about impedance mismatch, with a neat diagram?
- 3) Write short notes on
  - a) Consequences of Aggregate Orientation
  - b) Key-value data model
  - c) Document data model
  - d) Column family stores
  - e) Graph database
- 4) What are Schemaless databases? Explain in detail?
- 5) What are distribution models? Explain two paths of data distribution?
- 6) Write short notes on
  - a) Single Server
  - b) Combining Sharding and Replication distribution data model.
- 7) Explain about Update and Read Consistency with an example?
- 8) Explain about CAP Theorem

1) What is NoSQL? Explain briefly about aggregate data models with a neat diagram?

**ChatGPT said:**

ChatGPT

**NoSQL** databases are non-relational databases designed to handle large volumes of data, distributed data storage, and high-performance requirements. Unlike traditional relational databases, they don't use SQL for queries and are often schemaless, allowing for flexibility in data storage and retrieval. NoSQL databases are especially popular in applications that require scalability, quick iteration, and efficient handling of big data.

### **Aggregate Data Models in NoSQL**

**Aggregate data models** organize data into "aggregates," which are collections of data treated as single units. Aggregates simplify data distribution across clusters in NoSQL databases, as the data often needs to be moved or processed together. For example, an "Order" in an e-commerce system could include customer details, order items, and payment information within one aggregate.

This model is especially helpful for distributed data systems because each aggregate can be managed as a single unit, which supports easier data partitioning and replication.

**Diagram:**

vbnet

Copy code

```
[Customer Aggregate]
```

```
|
```

```
|
```

```
[Order Aggregate]
```

```
|
```

```
|
```

```
[Order Items, Shipping Address, Payment Details]
```

In this example, each aggregate can be stored and managed independently, making it suitable for large, distributed databases where data consistency within each aggregate is crucial but can be managed at a different pace across aggregates.

4o

**You said:**

2) Explain briefly about impedance mismatch, with a neat diagram?

**ChatGPT said:**

ChatGPT

**Impedance Mismatch** refers to the conflict that arises due to the difference in how data is represented in object-oriented programming (OOP) and relational databases. In OOP, data is

structured as complex objects, often containing nested attributes and hierarchical relationships, while relational databases store data in tables with rows and columns, with each table holding flat, tabular data.

The **impedance mismatch** occurs because:

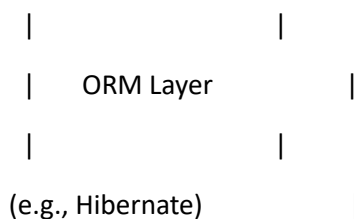
1. Objects in code have attributes that can reference other objects, creating complex nested data structures, but relational databases require data to be normalized and stored across multiple tables.
2. Converting these objects to tables requires an Object-Relational Mapping (ORM) layer, which can lead to performance and consistency issues.
3. Developers often face challenges translating between these two structures, making data access and manipulation more complex.

**Diagram:**

scss

Copy code

[Object-Oriented Structure] <---> [Relational Structure]



In this diagram:

- The **Object-Oriented Structure** represents complex nested relationships, which are more natural in applications.
- The **Relational Structure** represents normalized data in tables, which are better suited for storage in relational databases.
- The **ORM Layer** serves as a bridge, translating between objects in code and rows/columns in the database, although it may introduce overhead and complexity.

4o

**You said:**

3) Write short notes on a) Consequences of Aggregate Orientation b) Key-value data model c) Document data model d) Column family stores e) Graph database

**ChatGPT said:**

ChatGPT

Here are brief explanations for each of the requested NoSQL database models and concepts:

**a) Consequences of Aggregate Orientation**

- **Simplified Distribution:** Aggregates are well-suited for distribution because they are self-contained units of data, making it easier to partition data across nodes in a cluster.
- **Data Duplication:** Storing related data together can lead to data duplication if the same data is part of multiple aggregates, which can increase storage requirements.
- **Consistency Challenges:** Since aggregates are designed to operate as single units, maintaining consistency across aggregates can be challenging in distributed systems, especially when updates need to occur simultaneously across multiple aggregates.
- **Flexibility for Developers:** Aggregate orientation is beneficial for developers as it aligns data storage with the natural structure of applications, grouping data that's frequently accessed together.

#### b) Key-Value Data Model

- **Description:** In the key-value model, data is stored as a collection of key-value pairs, where each key is unique and maps directly to a specific value (the data).
- **Use Cases:** Suitable for applications requiring simple and quick data retrieval, like session storage, caching, or shopping cart data.
- **Advantages:** Fast lookups by key and flexible data storage, as values can be any data type.
- **Example:** Redis, DynamoDB.

#### c) Document Data Model

- **Description:** Document databases store data in documents, typically using formats like JSON, BSON, or XML, which allow for complex nested structures.
- **Use Cases:** Well-suited for content management systems, catalogs, and applications needing flexible schemas.
- **Advantages:** Supports semi-structured and nested data, making it adaptable for changing data requirements.
- **Example:** MongoDB, Couchbase.

#### d) Column Family Stores

- **Description:** Column family stores organize data in columns and rows, similar to relational tables but with more flexible schema definitions. Data is stored in column families, which group related columns together.
- **Use Cases:** Ideal for applications that need fast write and read operations, such as logging systems, real-time analytics, and large-scale data storage.
- **Advantages:** Optimized for queries on specific columns and allows for efficient storage of sparse data.
- **Example:** Cassandra, HBase.

#### e) Graph Database

- **Description:** Graph databases use nodes, edges, and properties to represent and store data, making them ideal for applications with complex relationships.

- **Use Cases:** Perfect for social networks, recommendation engines, and fraud detection, where relationships between data points are critical.
- **Advantages:** Efficiently handles highly connected data and complex queries involving relationships.
- **Example:** Neo4j, Amazon Neptune.

These models each have unique strengths and are chosen based on the requirements of data structure, relationships, and access patterns.

4o

**You said:**

4) What are Schemaless databases? Explain in detail?

**ChatGPT said:**

ChatGPT

**Schemaless Databases**, also known as schema-free or schema-flexible databases, are databases that do not require a predefined schema to define the structure of data. Unlike traditional relational databases, which require a rigid schema with specified tables, columns, and data types, schemaless databases allow for more flexibility in storing data. This flexibility is advantageous for modern applications with dynamic and evolving data requirements.

#### Key Characteristics of Schemaless Databases

1. **Flexible Data Structure:** Schemaless databases allow each record to have a unique structure, meaning new fields can be added to documents without altering existing data or requiring database migrations.
2. **Handling Unstructured and Semi-structured Data:** These databases can store unstructured or semi-structured data, like JSON or XML documents, which are commonly used in applications requiring flexible data models.
3. **Rapid Development and Iteration:** Developers can modify and evolve the database schema directly in the application code, which accelerates development, particularly in Agile environments where requirements change frequently.
4. **Data Variability:** Schemaless databases are ideal for applications where data formats vary across entries. For instance, in an e-commerce application, product data might include different fields depending on the product category (e.g., books vs. electronics).

#### Types of Schemaless Databases

Schemaless databases often fall under the NoSQL category, with popular types including:

- **Document Databases:** Store data in document formats, such as JSON, BSON, or XML. Examples include MongoDB and Couchbase, which store data as flexible documents that can vary in structure.
- **Key-Value Stores:** Store data as a collection of key-value pairs, where each entry can be completely independent of others. Examples include Redis and DynamoDB.

- **Column Family Stores:** Use columns to store data but allow the structure of each row to vary. Examples include Cassandra and HBase.

### Advantages of Schemaless Databases

1. **Adaptability:** As application requirements change, developers can add, remove, or modify fields without restructuring the database, allowing quick adaptation.
2. **Efficiency for Modern Applications:** Many modern applications handle large volumes of data that do not fit well into rigid schemas. Schemaless databases handle such data well, offering flexibility without compromising performance.
3. **Speed of Deployment:** Without the need for database migrations, updates to data models can be deployed faster, reducing downtime and accelerating feature delivery.

### Drawbacks of Schemaless Databases

1. **Data Integrity:** Without a fixed schema, it's challenging to enforce data integrity across the database, which can lead to inconsistencies.
2. **Complex Queries:** Schemaless databases may lack support for complex joins and aggregations, making some query types more difficult or requiring denormalization of data.
3. **Data Validation:** The application must handle data validation since there are fewer in-built constraints in the database.

### Use Cases for Schemaless Databases

- **E-commerce Applications:** Flexible storage for product data where different product types may have varying attributes.
- **Content Management Systems (CMS):** Dynamic content storage, where each document can have unique fields depending on content type.
- **IoT Applications:** IoT devices generate diverse data formats that may change over time, which suits the flexible structure of schemaless databases.

Schemaless databases offer a powerful way to manage dynamic, unstructured, and evolving data in modern applications, providing flexibility but requiring careful handling to maintain data integrity and consistency.

4o

**You said:**

5) What are distribution models? Explain two paths of data distribution?

**ChatGPT said:**

ChatGPT

**Distribution Models** in databases refer to strategies used to distribute data across multiple nodes or servers within a network. Distribution improves performance, scalability, availability, and fault tolerance in large-scale applications. In distributed database systems, data can be managed across several servers, ensuring that if one node fails, others can continue to provide access to data.

### Two Paths of Data Distribution

## 1. Replication:

- **Description:** Replication involves copying data across multiple nodes, ensuring that data is available on more than one server. This approach improves data availability and fault tolerance because if one server fails, the data can still be accessed from another server.
- **Types of Replication:**
  - **Master-Slave Replication:** One node is designated as the master, where all write operations occur. Changes made on the master node are then replicated to the slave nodes, which are typically read-only. This model is effective for read-heavy workloads, where multiple slaves handle read requests while the master manages writes.
  - **Peer-to-Peer Replication:** In this model, all nodes are equal and can perform both read and write operations. Changes are synchronized among nodes. This approach increases resilience, as any node can update data and handle requests, but it introduces complexity in managing data consistency.

## 2. Sharding:

- **Description:** Sharding is a form of horizontal partitioning where data is divided into smaller, distinct parts (called shards), and each shard is stored on a separate server. Each server (node) holds only a subset of the data, rather than a copy of the entire dataset.
- **Benefits:** Sharding allows for distributed write operations, as each shard can handle write requests independently. This helps with scalability because as data grows, new nodes can be added to store additional shards.
- **Challenges:** Managing queries that span multiple shards can be complex, and data retrieval may require multiple nodes to be accessed, which can impact performance.

### Summary of Differences

- **Replication** focuses on **data redundancy** and **availability** by keeping multiple copies of the same data, useful for read-heavy applications.
- **Sharding** focuses on **data partitioning**, distributing distinct data across servers for scalability, making it suitable for applications with high data volume and write operations.

These distribution models can be combined to achieve both scalability and high availability. For example, each shard in a system can be replicated for fault tolerance, balancing both models' strengths in a distributed database system.

1/2

4o

**You said:**

6) Write short notes on a) Single Server b) Combining Sharding and Replication distribution data model.

**ChatGPT said:**

ChatGPT

Here are short notes on the two distribution models you asked about:

**a) Single Server**

- **Description:** In a single-server setup, all data resides on a single machine. This is the simplest database configuration where the server handles all read and write operations.
- **Advantages:** A single-server setup is easy to set up, manage, and reason about, making it ideal for applications with minimal data storage and access needs. It also eliminates the complexities of distributed systems, such as network latency and data consistency.
- **Disadvantages:** Single-server configurations lack scalability and are prone to single points of failure. If the server goes down, the entire database becomes inaccessible, which limits the setup's reliability for critical applications.
- **Use Cases:** Suitable for small applications with minimal traffic or testing environments where high availability is not a priority.

**b) Combining Sharding and Replication Distribution Data Model**

- **Description:** Combining sharding and replication involves partitioning the data into multiple shards and replicating each shard across several nodes. This hybrid approach provides both scalability and fault tolerance by distributing data across nodes and ensuring that each shard has multiple copies for redundancy.
- **Advantages:**
  - **Scalability:** Sharding allows the system to handle large datasets and distribute write operations across nodes, reducing the load on any single node.
  - **Fault Tolerance:** Replication ensures data redundancy, so if one node fails, the data can still be accessed from another replica of the shard.
  - **High Availability:** By replicating shards, read and write operations can continue even if some nodes are down, increasing the overall availability of the database.
- **Challenges:** Managing both sharding and replication adds complexity, as it requires careful configuration to maintain data consistency and balance load across nodes.
- **Use Cases:** This model is widely used in large-scale distributed applications, such as social media platforms and e-commerce sites, where data volumes are high, and uptime is crucial.

This combined model maximizes the strengths of both distribution approaches, making it ideal for applications that demand both high availability and scalability.

4o

**You said:**

7) Explain about Update and Read Consistency with an example?

**ChatGPT said:**



## Update and Read Consistency in Distributed Databases

In distributed databases, **consistency** ensures that data is synchronized across multiple nodes so that users see the same data regardless of where or when they access it. There are two main types of consistency concerns:

### 1. Update Consistency:

- **Description:** Update consistency refers to the guarantee that when data is updated in one location, this update will be propagated consistently across all nodes where the data is replicated. This helps avoid conflicts, such as two users trying to modify the same data at the same time.
- **Example:** Suppose two users, Alice and Bob, try to reserve the last available room in a hotel at the same time. Without update consistency, both users might see the room as available and attempt to book it simultaneously, resulting in a conflict. A consistent update ensures that only one booking is successful, preventing double booking.

### 2. Read Consistency:

- **Description:** Read consistency, also known as read-your-writes consistency, ensures that when a user writes or updates data, they can immediately see the change upon reading it. This prevents scenarios where users read stale data after making updates.
- **Example:** If Alice posts a comment on a blog and then refreshes the page, read consistency ensures she immediately sees her comment, even if it's stored on multiple nodes. Without read consistency, Alice might not see her own comment immediately, which can be confusing.

## Example in Practice

Consider an e-commerce website where users can add items to their cart:

- **Update Consistency** ensures that if Alice adds a product to her cart, this update is propagated across all nodes. If Alice logs in from a different device, she will still see the same items in her cart.
- **Read Consistency** ensures that when Alice adds an item to her cart, she can see it immediately on the same screen without refreshing or waiting, even if there's a delay in synchronizing with other nodes.

## Challenges

Achieving strict update and read consistency can be challenging in distributed systems, especially when using replication across multiple servers. Techniques like quorum-based voting, version stamping, or using distributed locks are often employed to maintain consistency while balancing system performance and availability.

4o

You said:

## 8) Explain about CAP Theorem

**ChatGPT said:**

ChatGPT

### **CAP Theorem**

The **CAP Theorem**, also known as Brewer's Theorem, was formulated by computer scientist Eric Brewer and states that it is impossible for a distributed data system to achieve all three of the following properties simultaneously:

#### **1. Consistency:**

- Consistency in CAP terms means that every read operation returns the most recent write or an error. Essentially, all nodes in the system see the same data at the same time.
- For example, if Alice updates her profile photo in a social network application, a consistent system would ensure that anyone viewing Alice's profile immediately sees the updated photo.

#### **2. Availability:**

- Availability means that every request (read or write) receives a response, even if some nodes are down. In an available system, the system always responds to read and write requests, though it may not reflect the most recent data.
- For example, if a node in a distributed system fails, an available system would route the request to another node, ensuring the user still receives a response.

#### **3. Partition Tolerance:**

- Partition tolerance implies that the system continues to operate, even if there is a loss of communication between nodes (network partitioning). A partition-tolerant system handles scenarios where nodes can't communicate with each other by ensuring that the system continues functioning despite network failures.
- For instance, if there is a temporary network partition between two data centers, the system should continue operating independently on each side of the partition.

### **CAP Trade-offs**

According to the CAP theorem, a distributed system can only satisfy **two out of the three properties** at the same time. Here's how the trade-offs play out:

- **CP (Consistency and Partition Tolerance):**

- A CP system favors consistency and partition tolerance over availability. In the case of a network partition, the system may reject requests to ensure consistency across the partition.
- **Example:** Many traditional relational databases can be configured to prioritize consistency and partition tolerance, so they reject requests if the network partition prevents them from guaranteeing consistency.

- **CA (Consistency and Availability):**
  - A CA system prioritizes consistency and availability but not partition tolerance. This setup is often used in single-node systems where partition tolerance isn't a concern.
  - **Example:** A single-server relational database is an example of a CA system, as there's no partitioning to account for, and it can maintain consistency and availability.
- **AP (Availability and Partition Tolerance):**
  - An AP system maintains availability and partition tolerance but sacrifices strict consistency. In the case of a network partition, different nodes may return different (potentially outdated) values.
  - **Example:** Many NoSQL databases like Cassandra and DynamoDB prioritize availability and partition tolerance, accepting eventual consistency, meaning that the data will eventually become consistent across nodes.

### Practical Application of CAP Theorem

In modern distributed systems, especially for large-scale web applications, **AP** is often favored to ensure high availability and fault tolerance across data centers. Systems designed this way may use **eventual consistency** models, where updates propagate across nodes asynchronously. This is suitable for applications that can tolerate temporary inconsistencies but need to ensure the system remains responsive and resilient.

### Summary

The CAP theorem guides how distributed databases are designed based on the specific needs and tolerance for trade-offs among consistency, availability, and partition tolerance. Most distributed systems lean towards either **AP** or **CP** configurations, balancing application needs with the realities of distributed network environments.