| USN | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|

Internal Assessment Test 1 – September 2025

| Sub: | **NoSQL Databases** | | | | | Sub Code: | BCD515C | Branch: | **CS (DS)** | |
|---|---|---|---|---|---|---|---|---|---|---|
| Date: | 26-09-25 | Duration: | **90 minutes** | Max Marks: | **50** | Sem | **V** | | **OBE** | |

| | | **Answer any FIVE Questions** | **MARKS** | **CO** | **RBT** |
|---|---|---|---|---|---|
| 1 | | Critically evaluate the benefits and drawbacks of using Map-Reduce for large-scale data analytics compared to traditional SQL queries. | [10] | 3 | L5 |
| 2 | | Explain the CAP theorem with real-world scenarios where each trade-off is prioritized | [10] | 2 | L2 |
| 3 | | Consider a large-scale e-commerce application. How would you apply sharding and replication to ensure scalability and reliability? | [10] | 2 | L3 |
| 4 | | Apply the concept of version stamps to explain how conflicts can be resolved in peer-to-peer replication. | [10] | 2 | L3 |
| 5 | | What is a materialized view? How does it help in improving the performance? | [10] | 1 | L1 |

| | | | | | |
|---|---|---|---|---|---|
| 6 | | Compare and contrast traditional databases and schema-less databases in terms of data model, flexibility, and query processing. | [10] | 1 | L4 |

**CI** **CCI** **HOD**

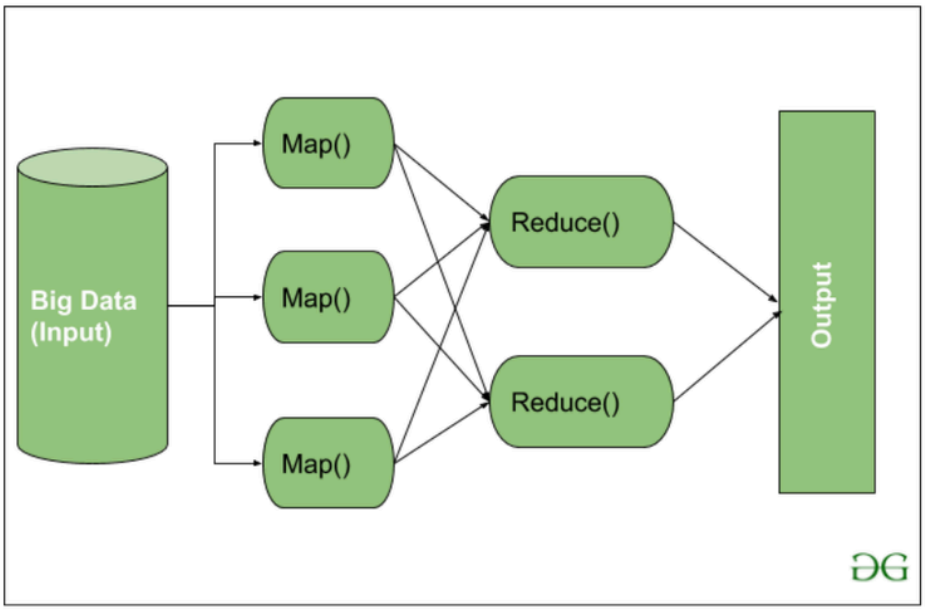
**Answer Key**

## 1. MapReduce vs Traditional SQL Queries

**(CO3 – L5: Evaluate)**

**Answer: Map-Reduce is a processing paradigm where any task is divided into sub-tasks, each sub-tasks is processed parallely and assigned with keys. The key-value pairs are**

**shuffled and sorted and then finally, reduced to an aggregated output.**




- **Benefits of MapReduce:**
  - Designed for **distributed processing** over large datasets.
  - **Scales horizontally** across clusters of commodity hardware.
  - Handles **unstructured/semi-structured data** easily.
  - **Fault-tolerant** – tasks automatically re-executed on failure.
  - Supports **parallel execution**, improving performance on petabyte-scale data.
- **Drawbacks:**
  - **Complex programming model** (requires writing map and reduce functions).
  - **High latency** – unsuitable for real-time queries.
  - **No ad-hoc querying** as in SQL; limited optimization features.
  - Requires **manual tuning** for efficiency.
- **Comparison with SQL:**
  - SQL: Declarative, optimized by query engine, suited for structured data.

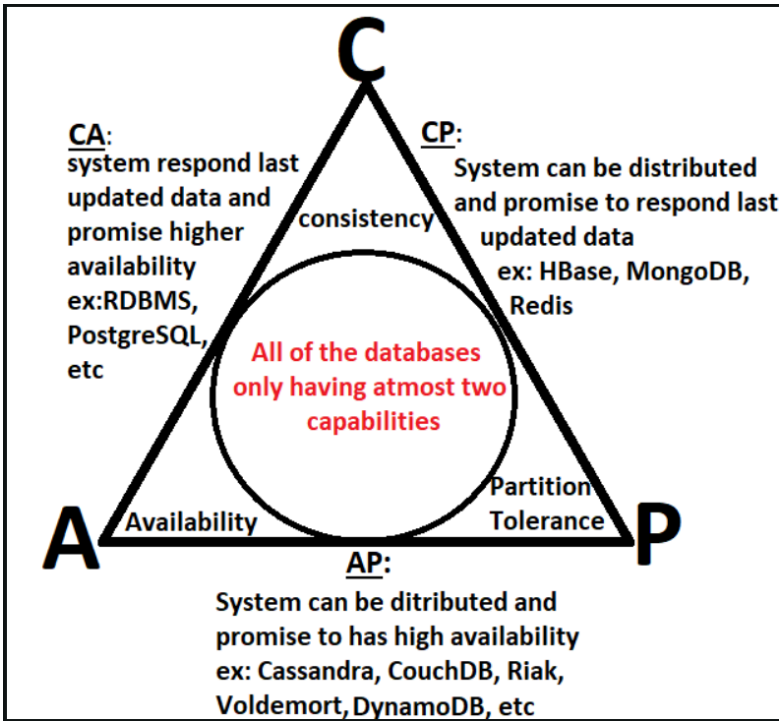
- MapReduce: Procedural, good for batch analytics on massive data.

**Conclusion:** MapReduce is superior for batch analytics on unstructured data, while SQL excels in fast, interactive analysis of structured data.

## 2. CAP Theorem with Real-World Scenarios

**(CO2 – L2: Understand)**
**Answer:**

- **CAP Theorem:** A distributed system can provide only two of the following three:
  - **C – Consistency:** All nodes see the same data.
  - **A – Availability:** Every request receives a response.
  - **P – Partition Tolerance:** System functions despite network failures.




**Scenarios:**

1. **CA (No Partition Tolerance):** Suitable for single server transactions or where ACID property is required.

- Example: Traditional RDBMS like PostgreSQL on a single node.
- Prioritizes data accuracy (Consistency) and response (Availability).

2. **CP (Consistency + Partition Tolerance):**Required for critical transactions where consistency is crucial. Ex. Financial Transactions
   - Example: MongoDB, HBase.
   - During partition, system may reject some requests to maintain consistency.
3. **AP (Availability + Partition Tolerance):** For transactions where availability is required with eventual consistency. For eg. Social Media
   - Example: Cassandra, CouchDB.
   - System remains available, allowing eventual consistency.

**Conclusion:** The choice depends on application needs — banking prefers CP, social media prefers AP.

## 3. Sharding and Replication in E-commerce Application

**(CO2 – L3: Apply)**
**Answer:**

- **Sharding (Horizontal Partitioning):**
  - Distribute user/order/product data across multiple servers.
  - Example:
    - Shard 1: Users A–M
    - Shard 2: Users N–Z
  - Improves **scalability and load distribution**.
- **Replication:**
  - Maintain multiple copies of shards (Master–Slave or Multi-Master).
  - Provides **fault tolerance** and **high availability**.

- **Application:**
    - **Catalog Service:** Sharded by product category.
    - **User Service:** Replicated for faster authentication.
    - **Order Service:** Master–slave setup to ensure durability and quick reads.

**Conclusion:** Combining sharding (for scalability) and replication (for reliability) ensures seamless performance in high-traffic e-commerce systems.

## 4. Version Stamps in Conflict Resolution

**(CO2 – L3: Apply)**
**Answer:**

- **Version Stamp:** Metadata (timestamp or vector clock) attached to each data item to track updates and resolve conflicts in a distributed enviroment.
- **Conflict Scenario:**
    1. Two nodes update the same record concurrently.
- **Resolution Steps:**
    1. Each update tagged with a unique **version vector** (e.g., Node ID + timestamp/ Vector clock etc.).
    2. On synchronization, system compares version stamps:
        - If one version supersedes the other → overwrite.
        - If concurrent → conflict → user-defined merge rule or manual reconciliation.
- **Example (CouchDB, Riak):** Use vector clocks to detect causality among versions.

**Conclusion:** Version stamps ensure eventual consistency by detecting and resolving update conflicts in distributed databases.

## 5. Materialized View and Its Role in Performance

**(CO1 – L1: Remember)**
**Answer:**

- **Definition:**
  A **materialized view** is a precomputed table that stores the result of a query physically, unlike a normal (virtual) view.

- **Purpose:**
  - Reduces computation time for complex joins/aggregations.
  - Avoids repeatedly running expensive queries.

- **Example:**
  Precomputed sales summary by region instead of aggregating daily transactions each time.

- **Performance Benefit:**
  - Faster query response.
  - Reduced load on base tables.
  - Periodically refreshed for updated results.

**Conclusion:** Materialized views improve performance by trading storage for query speed.

## 6. Traditional vs Schema-less Databases

**(CO1 – L4: Analyze)**

| Aspect | Traditional (RDBMS) | Schema-less (NoSQL) |
|---|---|---|
| **Data Model** | Structured tables with predefined schema | Flexible JSON, key-value, column, or graph model |

| | | |
|---|---|---|
| **Schema Flexibility** | Rigid, must alter table to change schema | Dynamic – fields can vary per document |
| **Scalability** | Vertical (single node) | Horizontal (multiple nodes) |
| **Query Language** | SQL (structured queries, joins) | Proprietary (MongoDB query language, CQL, etc.) |
| **Transactions** | ACID (Atomicity, Consistency, Isolation, Durability) | BASE (Basically Available, Soft-state, Eventually consistent) |
| **Use Case** | Banking, ERP | Social media, IoT, analytics |

Schema-less databases offer flexibility and scalability, while traditional DBs ensure strong consistency and transactional integrity.