Name- Suraj Kr. Vishwakarma
Course - MCA 1st Year
Sub - ADBMS
QID - 24510031
College - Quantum University

INDEX

NAME Su

SUBJECT ADBMS

STD.

SEC.

ROLL NO :

SCHOOL/ COLLEGE

S. No.	Date	Title	Page No.	Teacher Sign.

Date
Page

Tutorial - 1

Q1. Explain the term algorithm. What are the properties of an algorithm? List any five application of it.

Algorithm:

An algorithm is a finite set of well-defined instructions or a step-by-step procedure for solving a problem or performing a specific task. It takes an input, processes it according to the steps, and produces an output. Algorithms are fundamental to computing, as they form the basis for software development, data processing, and automated decision making.

Properties of an Algorithm :-

1. Finiteness - An algorithm must always terminate after a finite number of steps. It should not run indefinitely.

2. Definiteness - Each steps of an algorithm must be clear and unambigious, meaning it should be precisely defined so there is no confusion in its execution.

3. Input - An algorithm should take zero or more inputs from a specified set of data. It operates on this input to produce the output.

4. Output - An algorithm should produce one ore more outputs. These outputs are the results or solution to the given problem.

5. Effectiveness - Each step of the algorithm must be simple and doable, whether by a person or a computer.

Applications of Algorithms :-

1. Search Engines - Algorithms are used to fetch, index, and rank web pages based on user queries, providing the most relevant search results.

2. Cryptography - Algorithms like RSA or AES are used to encrypt and decrypt data, ensuring communication over the internet.

3. Machine Learning - Algorithms such as decision trees, neutral networks, and support vector machines help computers learn from

data and make predictions or decisions.

4. Routing Algorithms - Algorithms like Dijkstra's and A* help in finding the shortest path between two points in a map for navigation puposes.

5. Sorting and Searching in Databases - Algorith like quicksort, mergesort, and binary Search are used to efficiently retrieve and organize data in databases.

Q2. Define terms with example :—
(a) Time Complexity Analysis
(b) Space Complexity Analysis

(a) Time Complexity Analysis

Time Complexity Analysis refers to the process of determining how the time required by an algorithm grows with respect to the size of the input. Usually expressed in terms of "Big O notation" eg- O(n), O(log n).

Example: Linear Search(O(n))

```
def linear_search (arr, target);
 for i in range (len(arr)):
  if arr[i] == target:
      return i
  return -1
```

Time Complexity - O(n)
In the worst case, you might have to look at all 'n' elements, so the time complexity grows linearly with the input size.

(b) Space Complexity Analysis

Space Complexity Analysis refers to the process of determining how much memory or space an algorithm requires relative to the size of its input. It measures the total amount of memory needed to run the algorithm, including variables, data structures, and any auxiliary space used during execution.

Example: Linear Search (O(1))

```
def Linear_Search (arr, target):
    for i in range (len (arr)):
        if arr[i] == target:
            return i
    return -1
```

Space Complexity - O(1)
We only need space for variables like 'i' and 'target', so the space usage does not depend on the 'input size.

Q3. Differentiate between primitive and Non-primitive data types with an example.

Primitive Data Types:
- These are the most basic data types in Java that predefined by the language. They store simple values and do not have any additional methods or properties associated with them.
- Store the actual value in the memory where the variables is allocated.
- Fixed size depending on the type (e.g., "int" is always 4 bytes).
- Hold the actual values and operate directly on them.

Examples - int, char, float, double, boolean, byte, short, long

Non-Primitive Data Types:
- Also known as reference types, these data types are derived from primitive types and other classes. They refer to objects and have methods or properties associated with them.
- Store a reference where actual object is stored.

- Can have variable size depending on the object's structure and content.
- Hold references to the objects, and any modification to the object affects the original value.

Example - Arrays, Classes, Interfaces, Strings

Code :-

```
public class Data_Type_Example {
    public static void main (String[] args){
        int primitiveInt = 10;
        char primitiveChar = 'A';
        boolean primitiveBool = true;

        String nonPrimitiveStr = "Hello World";
        int[] nonPrimitiveArray = {1,2,3,4,5};
    }
}
```

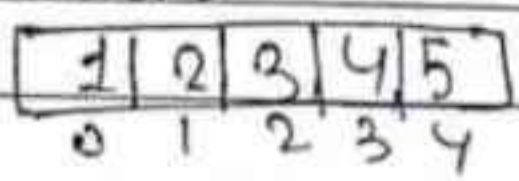
Q4. Discuss the concepts of Linear and Non-linear data structure through appropriate examples.

Linear Data Structures :—

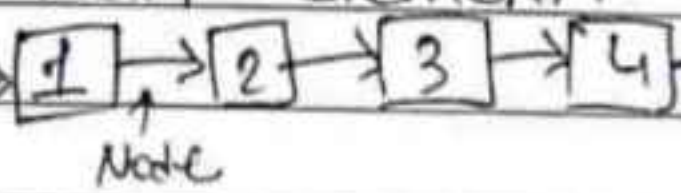
Linear Data Structures are organized in a sequential manner, where each element is connected to its previous and next element. Each element is accessed in a linear sequence, one after the other.

Examples of Linear Data Structures :—

1. Arrays - A collection of elements of the same data type stored in contiguous memory locations.

Eg- 

2. Linked Lists - A dynamic collection of elements, where each element points to the next element.

Eg - 

3. Stacks - A Last-In-First-Out (LIFO) data structure, where elements are added and removed from the top

4. Queues - A First-In-First-Out (FIFO) data structure, where elements are added to the end and removed from the top.

Non-Linear Data Structures :-

Non-linear data structures are organized in a hierarchical or complex manner, where elements are connected to multiple other elements.

Examples of Non-Linear Data Structures :-

1. Trees - A hierarchical data structure, where each node has a value and child nodes. Eg - A family Tree

2. Graphs - A network of nodes connected by edges, where each node can have multiple connections. Eg - Social media connec

3. Hash Tables - A data structure that maps keys to values using a hash function. Eg - A phone book.

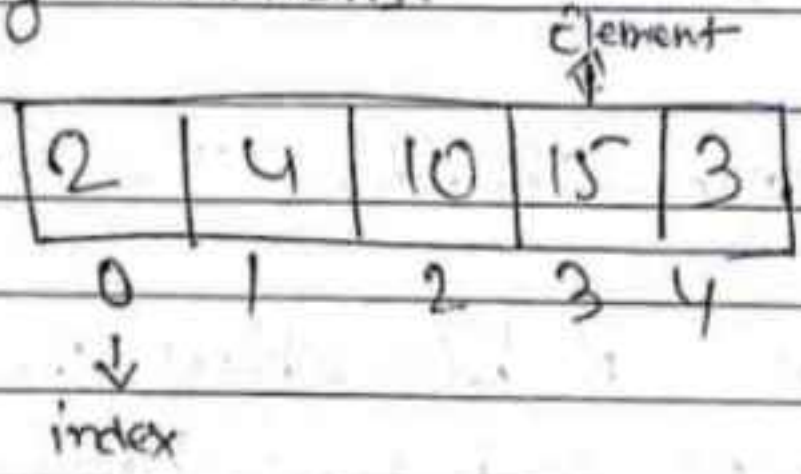
Q5. Define following terms :—

(a) Array (c) Stack

(b) Linked list (d) Queues

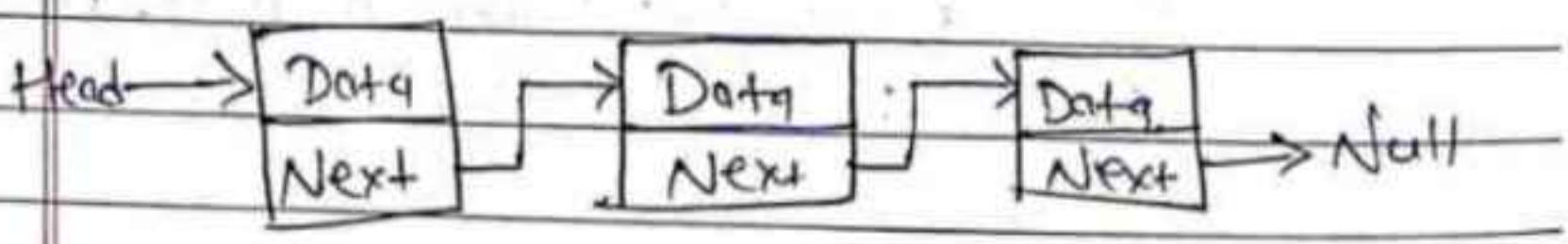
(a) Array

An array is a linear data structure that stores a collection of elements of the same data type in contiguous memory locations.



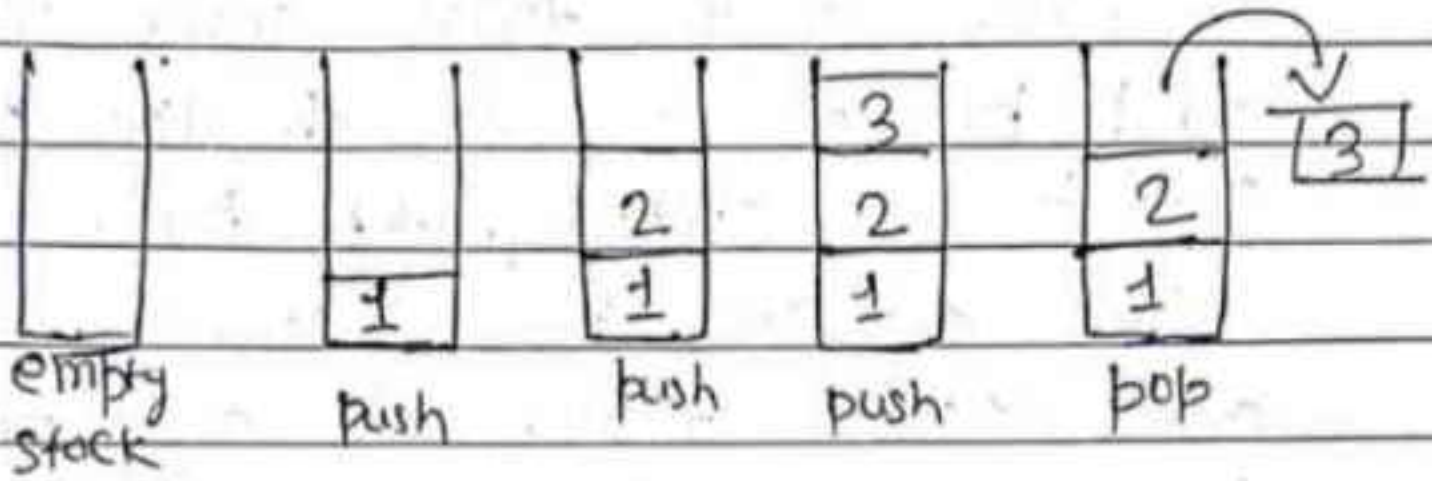
(b) Linked List

A Linked list is a dynamic data structure consisting of nodes, where each node contains a value and a reference to the next node



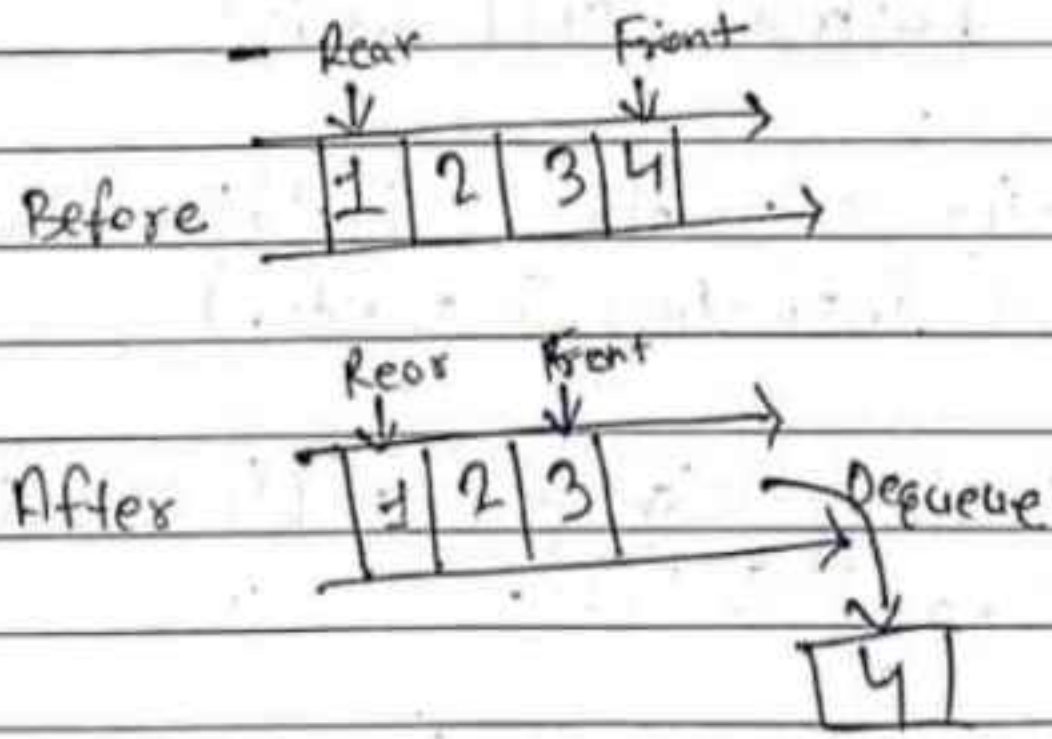
(C) Stack

A stack is a linear data structure that follows the Last-In-First-Out (LIFO) principle, where elements are added and removed from the top.



(d) Queues

Queue is a linear data structure that follows First-In-First-Out (FIFO) principle where elements are added to the end and removed from the front.



Date
Page

Tutorial-2

Q1. Explain the concept of an algorithm and state the criteria that an algorithm should meet.

An algorithm is a well-defined procedure that takes some input and produces a corresponding output. It is a step-by-step process that solves a specific problem or performs a particular task. An algorithm can be expressed in various forms, such as natural language, flowcharts, pseudocode, or programming languages.

Types of Algorithms :-

Algorithm can be categorized based on their purpose, such as :-

1. Sorting algorithms (Bubble sort, Quick Sort, Insertion Sort etc.)

2. Searching algorithms (Linear Search, Binary Search)

3. Graph algorithms (Dijkstra's Algorithm

, Bellman - Ford algorithms)

4. Dynamic Programming (Fibonacci Series)

Criteria for an Algorithm :-

1. Efficiency - Minimize computational resources (time, space, etc.).

2. Correctness - Produce accurate results.

3. Robustness - Handle invalid or unexpected input.

4. Scalibility - Perform well with large inputs.

5. Simplicity - Easy to understand and implement.

Q2. Define the term asymptotic notation in the study of algorithms.

Asymptotic notation :-

Asymptotic notation is a mathematical framework used to describe the growth rate or complexity of an algorithm's time or space complexity as the input size increases. It provides a way to analyze and compare the efficiency of algorithms.

Types of Asymptotic Notation :-

There are three main types of asymptotic notation :-

1. Big O Notation (Upper Bound) -
Represent the worst-case scenario, providing an upper bound on the algorithm's complexity. In formally, it say's "the algorithm takes no more than " a certain amount of time/ space.
Eg:- $O(n^2)$ means the algorithm's running time grows quadratically with the size of

the input.

2. Omega Notation (Big Ω: Lower Bound) -
Represent the best-case scenario, providing a lower bound on the algorithm's complexity. Informally, it says "the algorithm takes at least" a certain amount of time/space.
Eg:- Ω(n) means the algorithm's running time grows at least linearly with the size of the input.

3. Theta Notation (Big Θ: Tight Bound) -
Represent both the upper and lower bounds, providing an exact characterization of algorithm's complexity. Informally, it says "the algorithm takes exactly" a certain amount of time/space.
Eg:- Θ(n log n) means the algorithm's running time grows exactly like the product of the input size and it algorithm.

Q3. Define terms :-

(a) Worst- Case Analysis
(b) Average- Case Aalysis
(c) Best- Case Aalysis

(a) Worst - Case Analysis

Worst-case analysis is a method of analyzing an algorithm's performance by considering the maximum amount of time or resources it requires to complete, given the worst possible input. The analysis provides and upper bound on the algorithm's complexity.

Key characteristics :-

i. Considers the most difficult or largest possible input.
ii. Provides an upper bound on the algo-rithm's runnig time or space complexity.
iii. Helps ensure the algorithm can handle extreme cases.

(b) Average - Case Analysis

Average-Case analysis is a methed of analyzing an algorithm's performance by considering the expected or average time

Date
Page

or resources it requires to complete, given a typical or random input. This analysis provides a realistic estimate of the algorithm's complexity.

Key characteristics :-

i. Considers the easiest or smallest possible input.
ii. Provides a lower bound on the algorithm's running time or space complexity.
iii. Helps identify optimal scenarios.

(c) Best Case Analysis :

Best-case analysis is a method of analyzing an algorithm's performance by considering the minimum amount of time or resources it requires to complete given the best possible input. This analysis provides a lower bound on the algorithm's complexity.

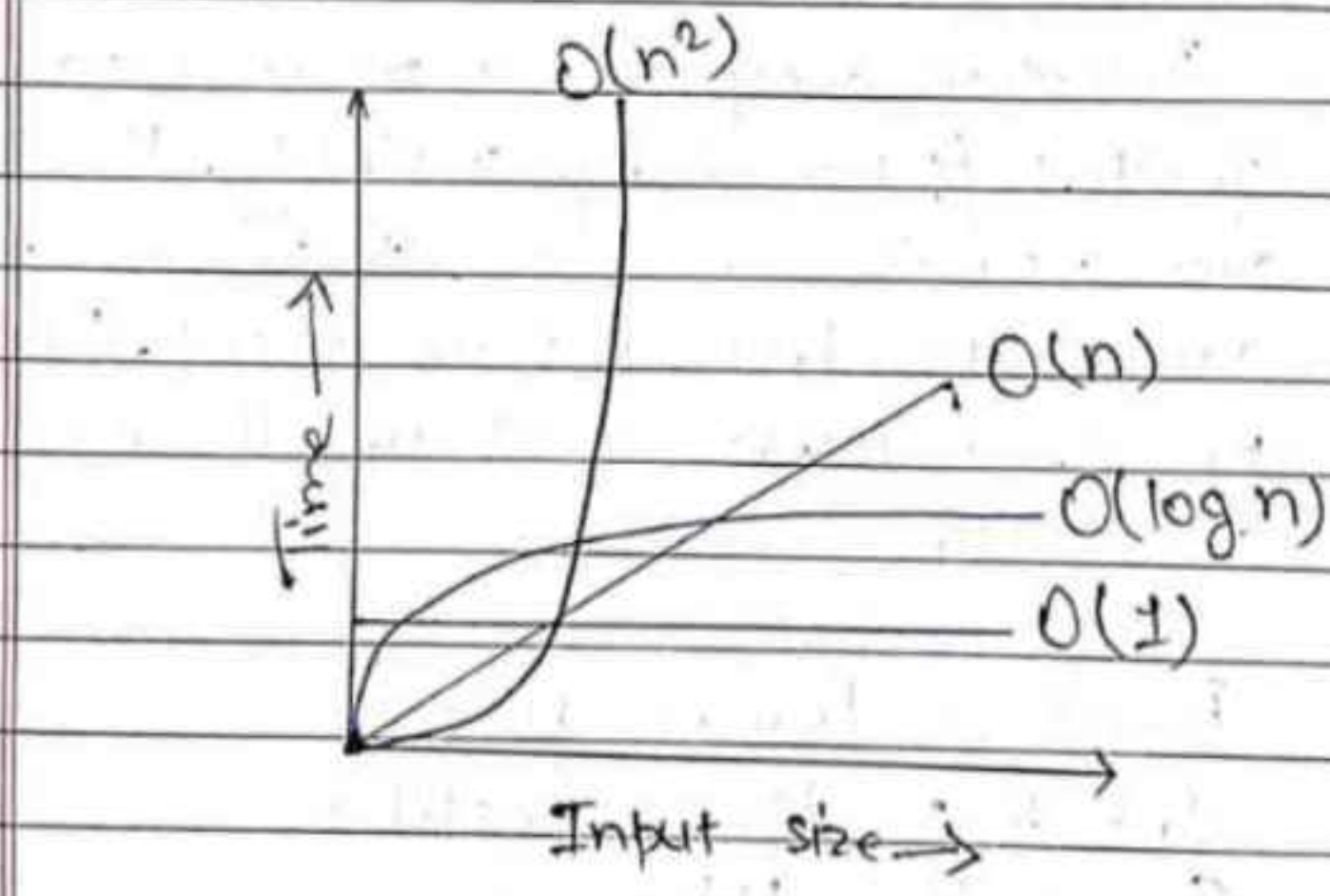
Focus - Lower bound.
Input - Best possible
Complexity - Minimum

Eg:- In a sorting algorithm, the best case scenario might be when the input is already sorted.

Q4. Design the graphical representation of big "oh", "omega" and "theta" notations to analyze algorithms. Give at least one example for each.

Here an graphical representation of Big O, Ω, Θ notations, along with examples for each :—

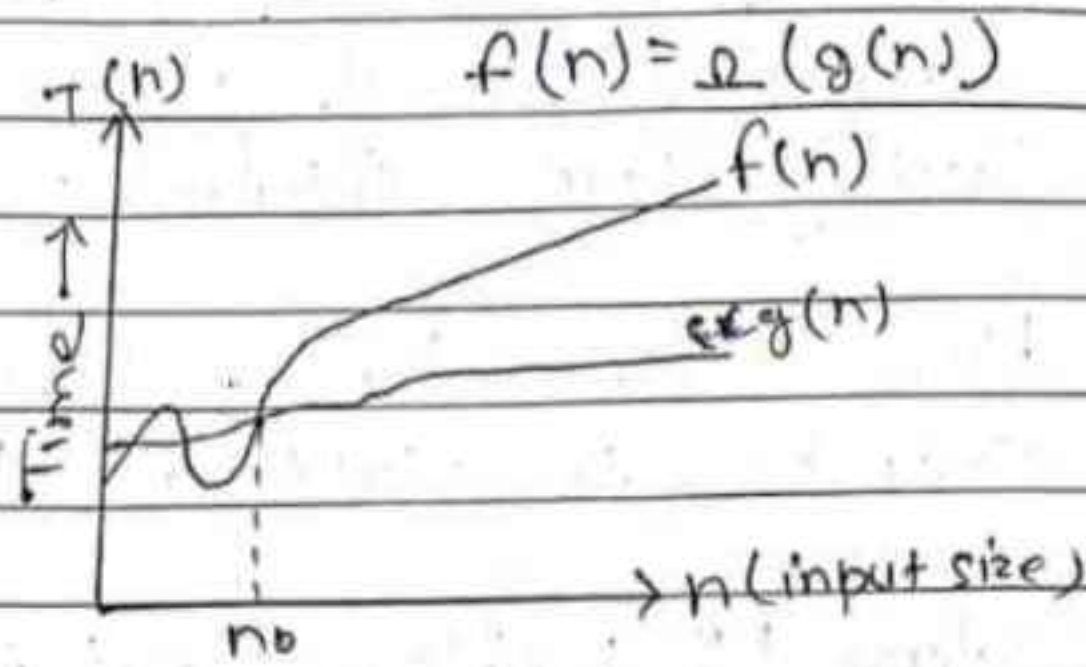
Big O (Upper Bound) :—



Example — Bubble Sort algorithm has a time complexity of $O(n^2)$, meaning its performance is bounded above by a quadratic function.

Date
Page

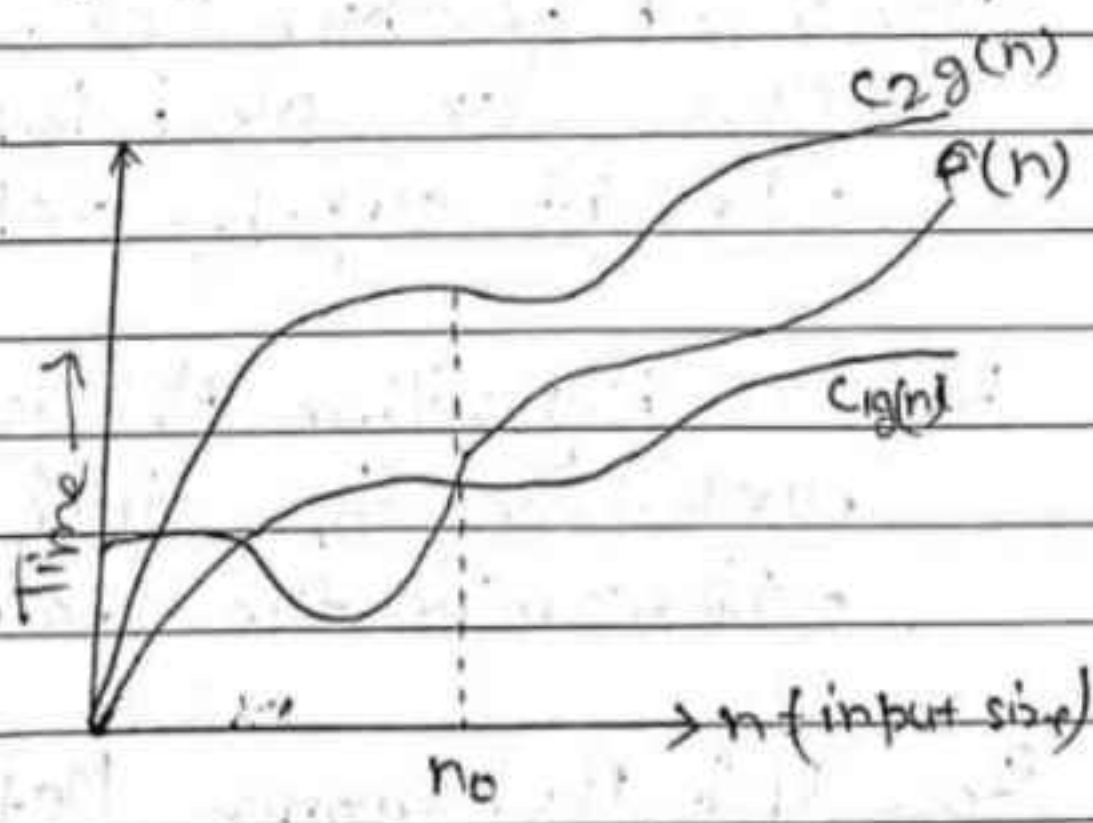
Big Ω (Lower Bound) :—



f(n) is bounded below by g(n) asymptotically.

Example - Binary search algorithm has a time complexity of Ω(log n).

Big Θ (Tight Bound) -



$f(n) = \Theta(g(n))$

Example - Merge sort algorithm has a time complexity of Θ(n log n).

Q5: Explain the various stages of the algorithm to design a performance analysis for any given problem. Justify this statement with an example.

Here are the stages and an example to illustrate them:-

Stage 1: Problem Definition
- Clearly articulate the problem statement.
- Identify the goals and objectives
- Determine the constraints.

Stage 2: Algorithm Design
- Choose an algorithmic approach.
- Develop step-by-step solution.

Stage 3: Algorithm Analysis
- Determine the time complexity.
- Determine the space complexity.

Stage 4: Performance Metrics Selection
- Choose relevant metrics.
- Establish baseline values or thresholds.

Stage 5: Experimental Design

- Decide on input data and scenarios.
- Plan experiments to collect performance data.

Stage 6 : Implementation and Testing
- Implement the algorithm.
- Test with selected inputs and scenarios.

Stage 7 : Data Collection and Analysis
- Collect performance data.
- Analyse result using statistical method

Stage 8 : Optimization and Refining
- Identify bottlenecks.
- Refine the algorithm for improved performance.

Stage 9 : Validation and Verification
- Confirm results meet expectations
- Validate against known solutions or benchmarks.

Example : Sorting Algorithm Performance Analysis

Problem :- Sort an array of integers in

ascending order.

Stage 1 : Problem Definition
- Minimize execution time.
- Array size (n) varies from 100 to 10,000.

Stage 2 : Algorithm Design
- Choose Quicksort algorithm
- Partition the array around the pivot

Stage 3 : Algorithm Analysis
- Quicksort O(nlogn) average-case, $O(n^2)$ worst-case.

Stage 4 : Performance Metrics Selection
- Executio time
- Memory usage

Stage 5 : Experimental Design
- Randomly generated arrays of sizes 10^3, 10^4, 10^5.
- Intel Core i7, 16GB RAM.

Stage 6 : Implementation and Testing.
code: . . . .

ascending order.

Stage 1 : Problem Definition
- Minimize execution time.
- Array size (n) varies from 100 to 10,000.

Stage 2 : Algorithm Design
- Choose Quicksort algorithm
- Partition the array around the pivot

Stage 3 : Algorithm Analysis
- Quicksort $O(n \log n)$ average-case, $O(n^2)$ worst-case.

Stage 4 : Performance Metrics Selection
- Executio time
- Memory usage

Stage 5 : Experimental Design
- Randomly generated arrays of sizes 10^3, 10^4, 10^5.
- Intel Core i7, 16GB RAM.

Stage 6 : Implementation and Testing
code! . . . .

Stage 7 : Data Collection and Analysis
- Run experiments, collect execution time data

Stage 8 : Validation and Verification
Verify correctness, Validate performance results.

Tutorial - 3

Q.1. Explain the detail concepts of distribution database over a centralised database with architecture and diswss the term of data replica and fragmentation.

Centralized Database :-
A centralized database is a traditional database architecture where all data is stored in a single location, typically on a single server or mainframe.

Distributed Database :-
A distributed database is a database that stores data across multiple physical locations, connected by communication links.

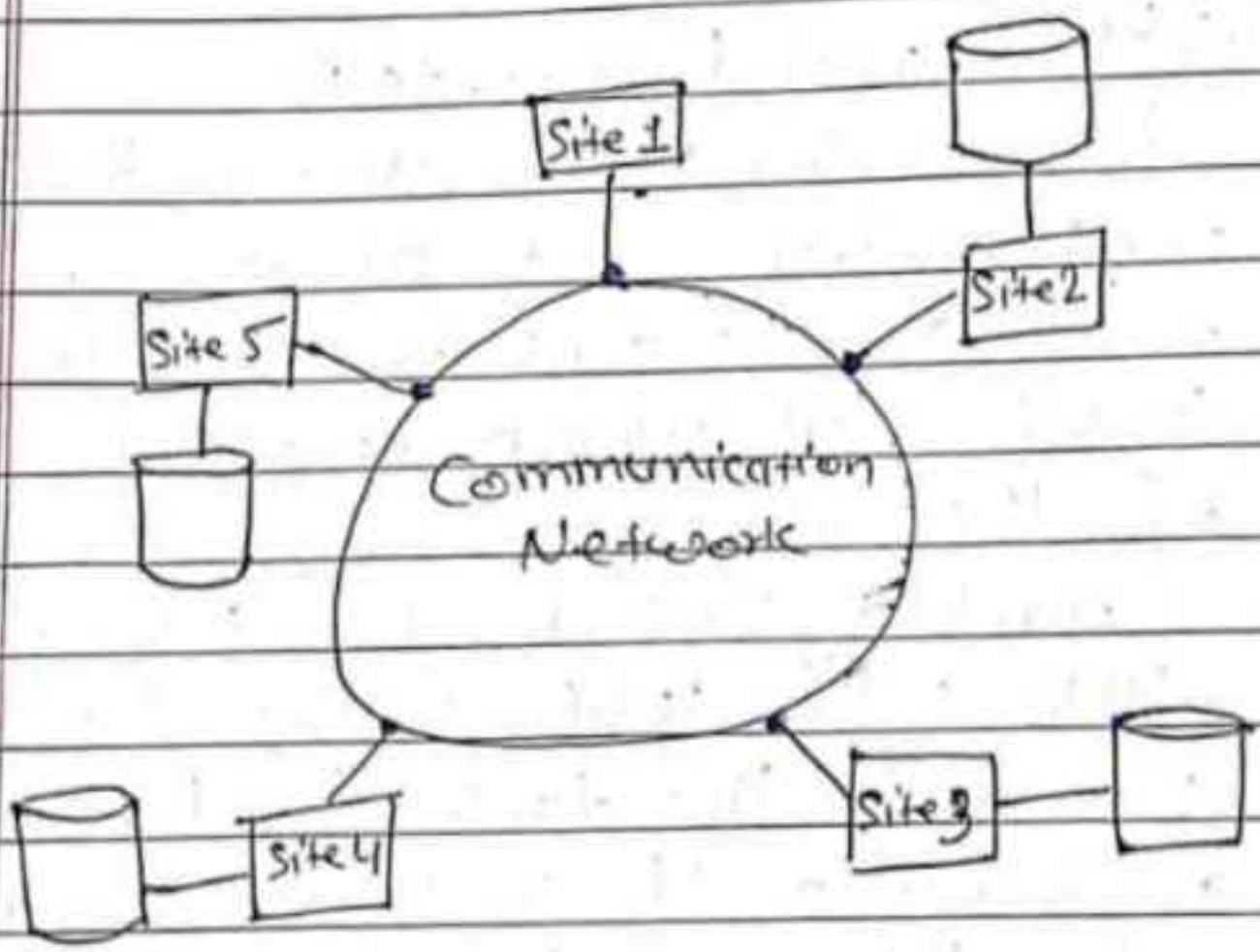
Architecture :-
- Multiple database servers (nodes).
- Data fragmented and distributed accross nodes.
- Each node may store a portion of the total data.
- Nodes communicate through a network

Data Replica Fragmentation :-
Data replica fragmentation occurs when

Date
Page

replicated data becomes inconsistent or outdated accross different nodes in a distributed database.

Architecture of distributed database:-



Q2. Define terms with example &—
a. Horizontal fragmentation
b. Vertical fragmentation
c. Daiven fragmentation
d. Reconstruction fragmentation

(a) Horizontal fragmentation
Horizontal fragmentation divides a relation into subsets of rows based on a specific condition. Each fragment contains all columns but only a subset of the rows.
Example - Consider a "Customers" table with 1000 entries. If we fragment this table by location, one fragment could contain all customers from "New York" while another could include customers from "Los Angles".

(b) Vertical Fragmentation
It divides a relation into subsets of columns with each fragment containing all rows but only a subset of colomn.
Example - For a "Employees" table containing columns like "Employid", "name", "address" and "salary", We might

two fragments. One with "Emp-id", "name" and another with "Emp-id", "salary".

(c) Driven Fragmentation
It is based on the accessed patterns or usage of data, meaning fragments are created based on how frequently specific data is accessed.
Example - If a sales application frequently access "orders" and "customers" together, a driven fragmentation that includes both table with only the necessary columns.

(d) Reconstruction Fragmentation
It is the process of combining fragments back into a complete relation. This is necessary when querying data has been fragmented.
Example - If the "customers" table was horizontally fragmented into two fragments reconstruction would involve combining these two fragments back to form the complete "customers" table when a query requests all customers data.

Q3. Explain the role of the concurrency control and 2 phase commit protocol in distributed transactions and its limitations.

Concurrency Control

Concurrency control ensures that database transactions operate concurrently without leading to inconsistencies. In distributed systems where transactions may involve multiple trasaction may involve to manage simultaneous operations to maintain data integrity. Techniques include locking, timestamp ordering and optimistic con-currency control.

2-Phase Commit Protocol:-

The 2-phase commit protocol is a mechanism for achieving consensus in distributed transactions. It ensures that all participating nodes either commit to the transaction or abort it, maintaining atomicity. The protocol consists of two phases:-

i. Preparing Phase - The co-ordinator sends a "prepare" request to all participants. Each participation votes to either commit

os about based on whether they can successfully complete the transaction.

ii. Commit Phase - If all participants vote to commit, the co-ordinate sends a "commit message". If any participants vote to commit, the co-ordinator sends a "abort message".

Limitations of 2-PC:

1. Performance overhead - Additional network communication and locking mechanism introduce latency

2. Single Point of failure - Co-ordination failure can block the entire transaction.

3. Locks - Prolonged locking can lead to resource unavailability and decreased concurrency.

4. Blocking - Transaction may wait indefinited for other transactions to release resource

Q4. Illustrate the role of the con-currency control and 2. Phase Commit protocol in distributed transactions and its limitations.

Role of the Con-Currency Control in Distributed Transactions :-

1. Manage simultaneous transaction requests.
2. Prevent data inconsistencies and conflicts.
3. Ensure serializability of transactions.
4. Lock resources exclusively
5. Check transaction timestamps.
6. Maintain multiple data versions.

Role of the 2- Phase Commit Protocol in Distributed Transactions :-

1. Co-ordinate transaction commitment.
2. Ensure all nodes agree on transaction outcome
3. Maintain transaction atomicity.
4. Nodes prepare to commit.
5. Nodes roll back.

Limitations of Concurrency Control and 2-Phase Commit :-

1. Scalibility - Complexity arise with increased nodes.

2. Network reliance - Performance impacted by network issues.

3. Resource locking - Potential for deadlocks.

To overcome these limitations, researchers and developers explore alternative protocols and optimizations, focusing on performance, scalibility and fault tolerance in distributed systems.

Q5. Construct an ER Diagram for a university registrar's office. The office maintains data about each data class, including the instructor the enrolment, and the time and place of the class meetings. For each student-class pair, a grade is recorded. Document all assumption that you make about the mapping constraints.

ER Diagram Components :

Entities :-

1. Student —
   - Student_Id (Primary key)
   - First_Name
   - Last_ Name
   - Email

2. Class Attributes —
   - Class_ID (Primary key)
   - Class_Name
   - Class_Time
   - Class_Location

3. Instructor Attributes —

- Instructor_Id (Primary key)
- First_Name
- Last_Name
- Email
- Department

4. Email Attributes —
   - Enrollment_Id (Primary key)
   - Grade

Assumptions :—

1. Instructor and class Relationship —
   - Each class is taught by exactly one instructor.
   - An instructor can teach multiple classes.

2. Student and Class Relationship —
   - A student can enroll in multiple classes.
   - Each classes can have multiple stedent enrolled.

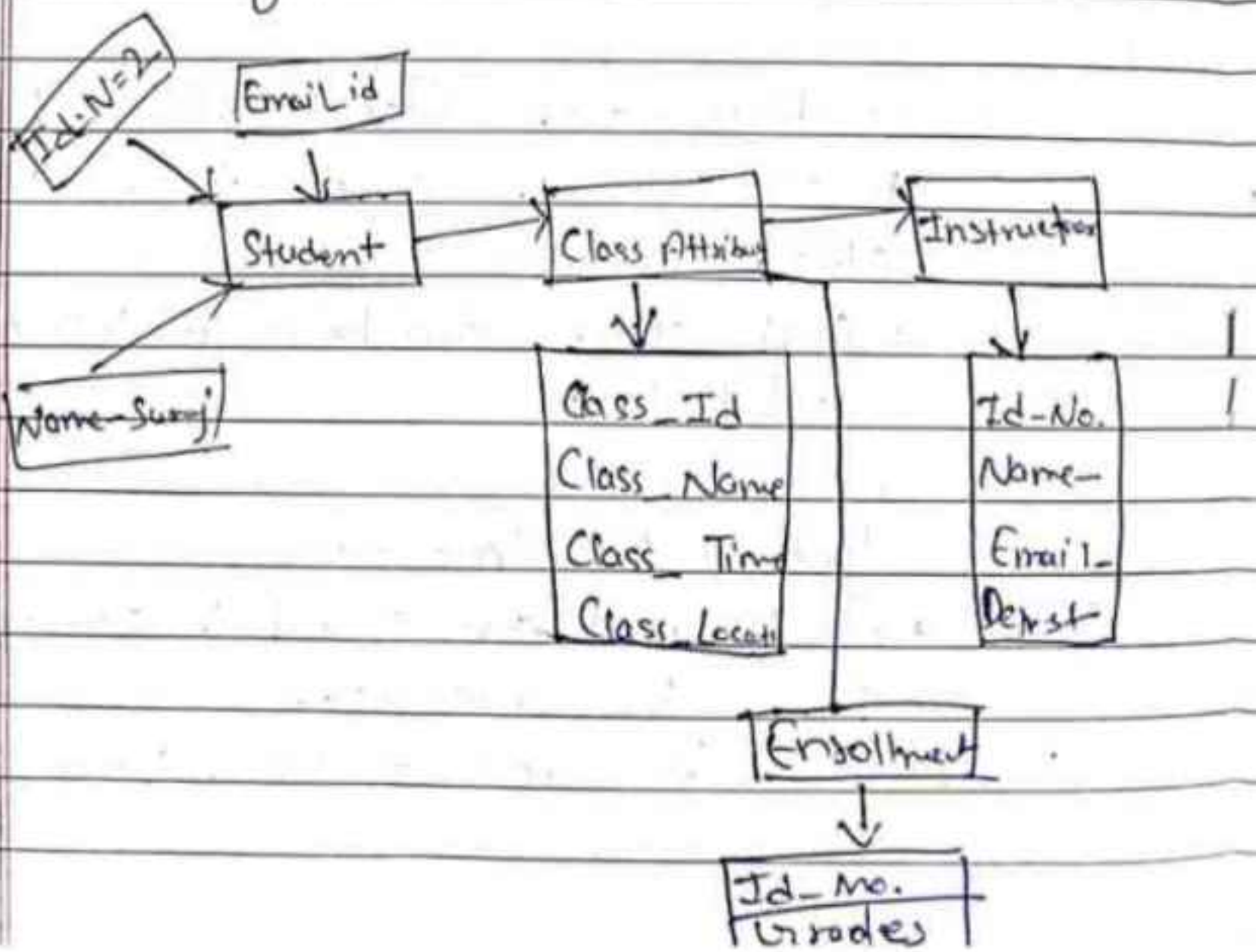
3. Student Enrollment —
   - A student can enroll in multiple classes per semester.
   - Each enrollment record includes a

grade specific to the student class pair.

4. Class Details –
- Each class has a fixed time and place is taught by one instructor at a time.
- Class can have different grades based on their unique enrollment.

5. No Duplicate Enrollments –
- A student cannot enroll in the same class more than once in a given semester

ER Diagram :-



Date
Page

Tutorial-4

Q1. Differences between homogeneus and hetrogeneus distributed database systems. Provide example of each.

Homogenous :

- In a homogenous DDB, all are the sites used same DBMS and operating system.
- The side used identical DBMS from the same users.
- Each site is aware of all other sites and cooperate with other site to cooperates user request.
- The DB is access a single interface as if 0 did repeated.

Example -

1. Oracle RAC - Multiple oracle DB work together seamlessly as a single database.
2. Postgre SQL with Citus - Distributes data across multiple Post SQL instance allowing for scaling and parallel proces

Hetrogenous :

- Its different sites have a different operating system.

- The system may be a composed a variety of DBMS — Types - Heirarchal DBMS, Relational DBMS, N/W DBMS.
- The hetrogenous DDB query processing is complex due to desimilar TMA translation processing is complex due to dismiliar software.
- A site main of beware of other sites and so, there is a limited co-operation in processing user request.

Example -

1. A System combining MySQL, MongoDB, Oracle — Different applications may use different databases based on their requirement, necessitating integration.

2. Federated Database System — Systems where multiple autonomous databases are accessed through a verified Interfied.

Q2. What are some common distributed database management system? Compare their features and use cases.

Here some common distributed database management systems along with a comparison of their features and uses :—

1. Cassandra :—

Features —

- Highly scalable with no single point of failure.
- Offers a decentralised architecture.
- Supports wide-column storage.
- Tunable consistency levels.

Use Cases —

- Ideal for applications requiring high write and read throughput.
- Suitable for real-time analytics and big data applications.

2. Mongo DB

Features —

- Document-oriented No SQL database
- Flexible schema design (JSON-like documents)
- Built-in sharing for horizontal scalability.

- Strong querying capabilities with aggregation framework

Use case —
- Good for application needing flexing data models
- Frequently used in web applications and mobile apps.

3. Cockroach DB -

Features —
- SQL complicated with a strong focus on resilience.
- Automatically replicates data across multiple nodes.
- Supports ACID transactions.
- Geo-distributions for low-latency access.

Use Cases —
- Suitable for financial services, ecommerce and applications, requiring high availability and strong consistency.

Q3. Define terms —
(a) Consistency.
(b) Site - distributes database
(C) Tokens
(d) Scanner
(e) Parser

(a) Consistency -
In the context of databases, consistency refers to the property that ensures any transaction property that ensures any transaction will bring the database from one valid state to another maintaining predefined rules and constraints.

(b) Site - distributes database -
A site distributed database is a type of database that is spread across multiple physical location on sites. Each site can operate independentity while still maintains some degree of co-ordination and data sychronigation with other sites.

(C) Tokens -

In Computer Science, tokens are symbolic representation used to denote specific types of information such as keywords, data identifiers or operators in programming language.

(d) Scanner –

A scanner is a component of a compiler or interpreter that reads source code and converts it into a sequence of tokens. It performs lexical analysis by identifying the basic elements of the code such as keywords, operators etc.

(e) Parser –

A parser is a component that analyses the structure of sequence of tokens generated by a scanner. It checks the syntax of the token sequence against the graduatically rules of the language.

Q4. Discuss the primary advantages of using a distributed database over a centralised database with an example.

1. Scalability :-
Distributed databases can easily scale horizontally by adding more nodes to handle increasing loads. For instance, if an applications like a social media platform experiences rapid user growth additional server can be integrated to manage data.

2. Fault Tolerance and Availability :-
With data spread across multiple across locations, distributed databases can continue operating even if one or more nodes fail. For example, in an e-commerce applications.

3. Geographical Distribution :-
Data can be stored closer to users in different geographical locations, reducing latency and improving access speeds. A global service like Netflix benefits from this by serving content from locate near users.

4. Data Redundancy and Backup :-
Data is often replicated across multiple nodes, providing redundancy that enhances data safety. In case of failure, backups can quickly restore data

Example - Consider a multinational retail chaining distribute databases. Each store can maintain a local database that syncs with a central system. This set up allows each store to operate independently, offering quick access to inventory data.

Q5. Evaluate the future trends in distributed databases. What emerging technologies or practicies are shapping the evolution, of distributed database system.

Some future trends :-

1. Multi-Cloud Strategies :-
Organizations are increasingly adopting multi-cloud environments, distributing their databases across various cloud provides. This enhances resilience and prevents vendor lock-in, allowing for greater flexibility.

2. Severless Architecture :-
Severless Computing is gaining tracien, enabling databases to scale auto-matically based on demond without the need for manual programming.

3. Distributed SQL Database :-
Emerging distributed SQL databases, like Cockroach DB and Yugabyte DB Combine the benefits of SQL and distributed systems.

4. Edge Computing :-
As. Iot devices proliferate data processing is shifting closer to the source. Distribute database that support edge computing architecture ~~tht~~ null enable real time data process and analytics.

5. Machine learning Integration :-
In co-operation machine learning directly into distributed database can enhance data processing capabilities. Automated optimization techniques can improve query performance and resource allocation.

Date
Page

Tutorial-5

Q1. Explain the key difference between SQL and NoSQL databases in terms of schema design and scalibility with suitable example.

SQL Databases :-

1. Schema Design :-

- SQL databases use a structured schema design where the database schema is predefined.
- Data is stored in tables with row and columns, and the structure schema where the database must be defined before adding any data.
- Examples of SQL databases – MySQL, PostgreSQL, Microsoft SQL Server.

Example –

```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    NAME VARCHAR(100),
    Department VARCHAR(50),
    Salary DECIMAL(10,2)
);
```

2. Scalibility :-

- SQL databases typically rely on vertical setting., meaning you add more resources to a single seever to handle more dataand transaction.
- This can become expensive and may have limitations as the data size grows significantly.
- SQL databases are ideal for compiex queries and transaction where data integrity is crucial.

NoSQL Databases :

1. Schema Design :-

- NoSQL databases usea flexible schema or sometimes no schema at all.
- Data can be stored in various forms like documents, key-value pairs, wide-column stores, or graphs.
- No strict schema is required, meaning data structures can be changed dynamically

Example :-

```
{
    "EmployeeeID" : 1,
    "Nome" : "John Doe",
    "Department" : "Scales",
    "Salary" : 5000
}
```

2. Scalibility :-

- NoSQL databases are designed for horizontal scaling, meaning you can distribute the data across multiple servers as dataset grows.
- Horizontal scaling allows NoSQL data-bases to handle massive amounts of unstructured data efficiently across distributed systems.
- No SQL databases are ideal for applications that require high availability, big hata handling, and rapid scalibility.

Q2. Explain following terms:
i. Horizontal fragmentation
ii. Vertical fragmentation
iii. MongoDB
iv. Primary key
v. Rollback
vi. Isolation
vii. Commit
viii. Deriven fragmentation
ix. Reconstruction
x. Data types

i. Horizontal fragmentation — Dividing rows of a table across different locations based on a condition. Each fragment contains all columns but only part of the rows.

ii. Vertical fragmentation - Dividing a table's of columns into smaller fragments. Each fragment has fewer columns and is distributed across locations.

iii. MongoDB - A NoSQL database that stores flexible, JSON-like data without a fixed schema, designed for scalibility and high availability.

iv. Primary Key - A unique identifier for each record in a table, ensuring no two rows have the same key and that it cannot be "NULL".

v. Rollback - Reverts a transaction to its previous state when an error occurs, undoing any partial changes.

vi. Isolation - Ensures that each transaction is independent and that changes are not visible to others until the transaction is complete.

vii. Commit - Finalizes a transaction, making its changes permanent.

viii. Derived Fragmentation - Fragmentation of one table based on the fragmentation of another related table.

ix. Reconstruction - Reassembling fragmented data from different locations or fragments to process a query.

x. Data Types - Define the kind of data a column can store, like Integer, VARCHAR, Date, etc.

Q3: Discuss the CAP theorem and how it impacts and operation NoSQL database. Provide examples of NoSQL databases that emphasize different aspects of the CAP theorem.

CAP Theorem Overview :-
The CAP Theorem states that a distributed system can only guarantee two out of three properties :-

1. Consistency - All nodes see the same data at the same time.
2. Availability - Every request gets a response, but not necessarily the latest data.
3. Partition Tolerance - The system continues to operates despite network issues.

Impact on NoSQL Databases :-

1. CP (Consistency & Partition Tolerance) :-
   - Focuses on consistent data across all nodes, but may sacrifice availibility.
   Example - HBase prioritizes consistency, ideal for financial systems.

2. AP (Availability & Partition Tolerance) :-

- Paioritizes availibility even during network issues but offers eventual consistency. Example- Cassandra and Riak ensure availability but data may be temporarily inconsis

3. CA (Consitency & Availability) :-
   - Can be achieved in single-node systems bu not in distributed one with network partitions.

Examples :
- Cassandra (AP) - High availibility with eventual consistency, used in real-time analytics.
- HBase (CP) - Ensures strict consistency, used in financial applications.
- MongoDB - Can switch between prioritizing consistency or availability.

Q4. What are the primary advantages of using a graph database over the NoSQL database types for managing Complex relationships? Illustrate with an example use case.

Advantages of Graph Databases for Managing Complex Relationships :

1. Efficient Relationship Mangement :-
Graph databases store relationships as first-class entities, making it easier to model and query connections directly.

2. Flexible Schema :-
Easily add or modify nodes and relationships without reconstructing the entire database.

3. Optimized for Relationship Queries :-
Native graph traversal allows efficient querying of complex relationships, like many-to-many connections.

4. Simplifies Complex Queries :-
Handles queries involving deep or wide relationships without complex joins,

unlike relational databases.

5. Better Performance for Connected Data :-
Optimized for handling and scaling complex, interconnected data.

Example Use Case : Social Network :
In a social network, graph databases efficiently manage users, friendships, and follows, quickly answering queries like "friends of friends" or "suggest connections". "Which would be slow in relational or other NoSQL databases.

Conclusion :
Graph databases are ideal for applications involving complex relationships, such as social networks or recommendation systems due to their efficient relationship handling and high performance.

Q5. Compare and contrast key-value stores and column-family stores in NoSQL databases. What are the strengths and limitations of each type?

Comparison of key-value Stores and Column-Family Stores in NoSQL Databases:

Both key-value stores and column-family stores are types of NoSQL databases, but they differ in structure, use cases, and functionality.

Key-Value Stores:

1. Structure:-
   - Data is stored as simple pairs: a key and a value.
   - The value can be any type of data.
2. Strengths:-
   - Very easy to understand and implement. Works well for simple data storage and retrieval.
   - Optimize for fast reads and writes, especially for cases where specific keys are used to retrieve values.

Limitations:

- Queries are based on keys only: you can't query by the value or perform complex queries.
- Locks support for managing relationships between data or querying across multiple key-value pairs.

Use Case:

Caching, session storage, or any application requiring simple, fast retrieval of data based on unique identifiers.

Column-Family Stores:

1. Structure:—
   - Data is stored in tables but with a flexible schema where rows contain multiple columns grouped into families.
   - Different rows can have different sets of columns.
2. Strengths:—
   - Ideal for storing wide ~~eff~~ datasets where not all columns are always populated.
   - Allows querying based on column names and values within column families, making it more versatile than key-value stores.
   - Designed for high write throughput, making

it suitable for write-heavy applications

Limitations:
- More complex to manage and under-stand compared to key-value stores.
- While more flexible than key-value stores, it's still limited in comparison to relational databases for complex querying.

Use Case:
- System that require handling large volumes of data which flexible columns, like time-series data, event logging., or large-scale analytics.

Date
Page

Tutorial-6

Q.1. Analyze the concept of eventual consistency in NoSQL databases. How does it differ from strong consistency, and what are the trade-offs involved?

Eventual Consistency in NoSQL Databases :-
Eventual consistency is a model where updates to data propagate to all nodes overtime, ensuring that all nodes will eventually coverage to the same state. However, there is no guarantee that reads will return the latest data immediately after a write.

Key Concepts :-
- Reads may return stale data temporarily.
- All nodes will become consistent eventually.
- Systems prioritize responding to requests, even during failures.

Eventual Consistency Vs. Strong Consistency :-

1. Strong Consistency :-
- Guarantees that all reads returns the latest data after a write.
- Requires coordination, which can slow

response times and affect availability during network partitions.

2. Eventual Consistency :-
- Allows temporary discrepancies, ensuring high availability and faster responses.
- Suitable for applications where stale data is acceptable, such as social media and e-commerce.

Trade-offs :
- Eventual consistency offers higher availibility and performance, while strong consistency ensures accuracy but may reduce availability.
- Eventual consistency has lower latency, while strong consistency incurs higher overhead.

Q2. Explain how sharding work in NoSQL databases and why it is important for scalability. What are the challenges associated with sharding?

Sharding in NoSQL Databases :
Sharding is a partitioning method that distributes data across multiple servers or nodes, improving scalability, performance, and availability. Each partition, or shard, contains a subset of the dataset.

How Sharding Works :
- The dataset is divided based on a sharding key, with each shard stored on a different node.
- A router directs request to the appropriate shard based on the sharding key.
- New shards can be added as data volume increases, allowing for easy scaling.

Importance of Scalability :
- Reduce load on individual servers, leading to faster operations.
- Enables horizontal scaling to accommodate growing datasets.
- A failure in one shard affects only a portion

of the data.

Challenges Associated with Sharding:
- Increases management complexity and data integrity maintanance.
- Quesies involving multiple shards can be et inefficient.
- Poor sharding key choices lead to data distribution and hot spots.
- Redistributing data among shards as the datasets changes can be challenging.
- Maintaining consistency across shards, especially with frequent updates, is complex.

Q3. Discuss the role and functionality of indexes in NoSQL databases. How do indexing strategies vary between document NoSQL db, key-value NoSQL db, graph db and column-family NoSQL db?

Role and Functionality of Indexes in NoSQL Databases:

Indexes in NoSQL databases serve to enhance query performance by providing a mechanism for efficient data retrieval. They allow the databases to locate and access data without scanning the entire dataset, improving response times for read operations.

Functionality of Indexes:

- Indexes enable quick lookups of data, reducing the time required to fulfill queries.
- They can support various types of queries, including filtering, sorting, and aggregation, depending on the database's capabilities.
- Indexes help in accessing frequently queried fields, enhancing overall application performance.

Indexing Strategies:

1. Documen NoSQL Databases:—

- Index on individual fields for faster queries.
- Index on multiple fields for efficient querying.
- Support full-text search.
- Optimize location-based queries.
- Distribute data evenly for efficient querying.

Key-Value NoSQL DB :—
- Automatically created on the primary key.
- Optional, created on non-primary key fields.
- Efficient for equality queries.

Graph DB :—
- Index nodes for faster traversal.
- Index relationships between nodes.
- Index nodes based on labels.
- Index node/relationship properties.

Column-Family NoSQL :—
- Automatically created on the primary key.
- Optional, created on non-primary key columns
- Index on multiple columns.

Q4. Evalute the impact of data replication on consistency and availability in NoSQL databases. How do different NoSQL databases handle replication and what are the implications for system performance?

Impact of Data Replication on Consistency and Availability in NoSQL Databases:
Data replication involves storing copies of data across multiple nodes, enhancing availability, fault tolerance, and performance. However, it also impacts consistency.

Impact on Availability and Consistency:
1. Availability:—
- Replication ensures continued access if a nodes fails, enhancing system resilience.
- Distributes read request across replicas, improving response times.

2. Consistency:—
- Increased availability can compliate consistency, leading to scenarios where replicas may have different data versions.
- NoSQL databases adopt various consistency models to balance availability and consistency.

Replication Handling in Different NoSQL Databases

1. Cassandra :—
   - Allow users to set replication factors and Consistency levels.
   - Flexible but can complicate data consistency during writes.

2. MongoDB :—
   - Uses a primary node for writes and secondary nodes for reads, with automatic failover.
   - Distributes read loads but may result in stale reads due to replication delays.

3. DynamoDB :—
   - Replicates data across regions for high availability.
   - Defaults to eventual consistency, with options for strong consistency, balancing availability and accuracy.

4. Neo4j :—
   - Handles replication with a leader node for writes, ensuring strong consistency.
   - Can introduce latency due to synchronization among nodes.

5. HBase :-

- Data is stored across servers with HDFS providing replication for fault tolerance.
- Ensures high availability but increases complexity for maintaining consistency.

Implications for System Performance :

- Replication may introduce delays due to the time needed to synchronize changes.
- While read throughput can improve, write throughput may decrease due to the need for coordination among replicas.
- Managing replication and consistency add complexity, requiring careful architectural design.

Q5. Provide a case study of a NoSQL database implementation in a real world scenario. Discuss the problem it addressed, the NoSQL database used, and how it benefited from the choice of NoSQL over traditional SQL databases.

Case Study :- Netflix and Cassandra

Netflix a leading streaming service, faced significant callenges with its traditional SQL databases —

- Scalibility — The demand for video content urged, requiring the ability to store and manage vast amounts of data, including user preferences.

- Performance — As user interactions increased the need for quick data retrieval seamless streaming experiences.

- High Availability — With a global audience netflix needs a system that could provide high availability and redundancy to minimize down time and maintain service reliability.

Benefits of NoSQL over Traditional SQL databases :-

1. Scalibility :-
   - Horizontal Scaling - Cassandra allows Netfli to add more servers to handle increasing data loads without significant reconfigurat
   - Distributed Architecture - Data is spread across multiple nodes, enabling Netflix to scale geographically.

2. Performance :-
   - Fast Writes and Reads - Cassandra is optimized for high write throughput, allowing Netflix to quickly log user activities and preferences.
   - Data Model Flexibility - The schema less nature of Cassandra allows Netflix to adapt to changing data require -ments without extensive downtime

3. High Availability :-
   - Replication and fault Tolerance - Cassandra's application features ensure that data is copied accross multiple nodes.
   - Tunable Consistency - Netflix can confi -gure the consistency level based

on the specific requirements of differents of different operations.

4. Global Distribution :-

- Multi-Data Centre Support - With users spread around the globe Cassandra's ability to operate accross multiple data centres allo netflix to provide low latency acces to data and enhance disaster recovery capabilities.