

## Module-

### 3: NoSQL

3.1 Introduction to NoSQL, NoSQL Business Drivers

3.2 NoSQL Data Architecture Patterns: Key-value stores, Graph stores, Column family (Bigtable)stores, Document stores, Variations of NoSQL architectural patterns, NoSQL Case Study

3.3 NoSQL solution for big data, Understanding the types of big data problems; Analyzing big data with a shared-nothing architecture; Choosing distribution models: master-slave versus peer-to-peer; NoSQL systems to handle big data problems.

#### Practice Questions

1. What is the primary motivation behind adoption of NoSQL databases?

##### 3.1.3 Why NoSQL?

The concept of NoSQL databases became popular with internet giants like Google, Facebook, Amazon etc. Who deals with huge volumes of data the system response time becomes slow when we use RDBMS for massive volumes of data so to resolve this problem, we could scale up our system by upgrading our existing hardware but this process is an expensive. So alternative for this issue is to distribute database load on multiple hosts whenever the load increases this method known as scaling out.

NoSQL databases are non-relational so they scale-out better than relational databases. As they designed with the web applications in mind. Now NoSQL database is exactly type pf database that can handle all sorts of semi structured data, unstructured data, rapidly changing data and bigdata. So, to resolve the problems related to large volume and semi structured data. NoSQL databases have emerged.

ChatGPT:

The primary motivation behind the adoption of NoSQL (Not Only SQL) databases is to address the limitations of traditional relational databases, especially in the context of handling large volumes of unstructured or semi-structured data and providing high scalability and performance in distributed and horizontally scalable environments. Some key motivations include:

1. **Flexible Schema:** NoSQL databases typically have a flexible schema, allowing developers to store and manage data without a predefined schema. This flexibility is beneficial when dealing with dynamic, evolving, or unstructured data, common in modern applications.
2. **Scalability:** NoSQL databases are designed to scale horizontally, meaning they can handle increased load by adding more servers to a distributed database system. This makes them suitable for applications with rapidly growing data and user bases.
3. **High Performance:** NoSQL databases often provide high performance for certain types of queries and operations, especially those involving read and write throughput. They are optimized for specific use cases, such as real-time analytics, which require low-latency responses.
4. **Big Data and Real-Time Applications:** NoSQL databases are well-suited for big data and real-time applications where data needs to be ingested, processed, and analyzed rapidly. They can handle large amounts of data with low-latency requirements, making them suitable for use cases like IoT (Internet of Things) and streaming data processing.
5. **Distributed Architecture:** NoSQL databases are designed to be distributed across multiple nodes or servers. This allows them to provide high availability and fault tolerance. Even if one node fails, the system can continue to operate.

## Module-

### 2. What are the business drivers for NoSQL?

Big Data Analytics (MU-Sem 7-COMP) (NoSQL)....Page no. (3-9)

#### 3.1.8 NoSQL Business Drivers

The scientist-philosopher Thomas Kuhn coined the term paradigm shift to identify a recurring process. He observed that in science, where innovative ideas came in bursts and impacted the world in nonlinear ways. We'll use Kuhn's concept of the paradigm shift as a way to think about and explain the NoSQL movement and the changes in thought patterns, architectures, and methods emerging today.

Many organizations are supporting to the single-CPU relational systems that have fulfilled the needs of their organizations as per the requirements.

Businesses have initiated the value in fast catching and examining huge quantity of adjustable data and making direct changes in their businesses based on the data that they obtain. As all of these drivers applies burden on single-processor relational model, its basis suits less steady and in time no extended encounters the organization's needs.

Fig. 3.1.3 In this, we see how the business drivers volume, speed, variability, and agility create pressure on a single CPU system, resulting in cracks. Volume and velocity state to the capability to handle and manage the big datasets that appears early. Variability states to how various data types do not fit within the structured tables, and agility states to how much fast an organization replies to the business modification.

There are 4 major business drivers for NoSQL as:

(1) Volume      (2) Velocity      (3) Variability      (4) Agility.

**1. Volume**

- Undoubtedly, the main factor forcing organizations to look at alternatives to their existing RDBMS is to investigate big data using clusters of commodity processors.
- Until around 2005, performance problems were eliminated by purchasing faster processors. Over time, the ability to speed up the process is no longer an option.
- As the chip density increases, the heat can no longer dissolve rapidly enough without chip overheating. This phenomenon, known as the power wall, forced system designers to shift their focus from increasing speeds on a single chip to using more processors working together.
- The need to scale out (also known as horizontal scaling), rather than scale up (faster processors), moved organizations from serial to parallel processing where data problems are split into separate paths and sent to separate processors to divide and conquer the work.

(New Syllabus w.e.f academic year 22-23) (M7-80)

Tech-Neo Publications...A SACHIN SHAH Venture

The diagram shows a central oval labeled "Single-node RDBMS". Four arrows point towards it from the sides: "Volume" (top), "Velocity" (left), "Variability" (bottom), and "Agility" (right). Each arrow has a small downward-pointing arrowhead at its tip.

Fig. 3.1.3

## Module-

Big Data Analytics (MU-Sem 7-COMP)

- 2. **Velocity**
  - While large data issues are considered for many organizations moving away from RDBMSs, the ability of a single processor system to read and write data quickly is also important.
  - Many single-processor RDBMS cannot meet the demand for online queries on databases created by real-time insert and public-facing websites.
  - RDBMSs frequently index multiple columns of each new row, a process that reduces system performance.
  - When single-processor RDBMSs are used as a back-end in front of a web store, random outbursts in web traffic reduce the response for everyone, and tuning the system can be expensive when both high read and write throughput is required.
- 3. **Variability**
  - Companies that seek to capture and report exceptional data conflicts when attempting to use the rigorous database schema structure imposed by RDBMSs. For example, if a business unit wants to capture some custom fields for a specific customer, it must store this information even if it does not apply to all customer rows in the database.
  - Adding new columns to RDBMS requires shutting down the system and running the ALTER TABLE command. When the database is large, this process can affect the availability of the system, costing time and money.
- 4. **Agility**
  - Agility is ability to accept change easily and quickly.
  - Among the variety of agility dimensions such as model agility (ease and speed of changing data modules), operational agility (ease models), operational agility (ease and speed of changing operational aspects), and programming agility (ease and speed of application development) – one of the most important is the ability to quickly and seamlessly scale an application to accommodate large amounts of data, users and connections.
  - The most complex part of building applications using RDBMS is the process of putting data into and getting data out of the database.
  - If your data has nested and repeated subgroups of data structures, you need to include an object-relational mapping layer.
  - The responsibility of this layer is to generate the correct combination of Insert, update, delete and select SQL statement to move object data to and from the RDBMS persistence layer.

Big Data Analytics (MU-Sem 7-COMP)

(NoSQL)....Page no. (3-11)

- This process is not simple and is associated with the largest barrier to rapid change when developing new or modifying existing applications.
- Generally, object relational mapping requires experienced software developers.
- Even with experienced staff, small change requests can cause slowdowns in development and testing schedules.
- All these hurdles are best overcome by NOSQL database.
- These databases are schematic and can be scaled down easily.
- They can accommodate application changes easily and can handle any volume of data efficiently.
- This agility has become business driver for NOSQL databases.

3. In what scenarios is NoSQL often preferred over SQL databases?

CHATGPT:

NoSQL databases are often preferred over SQL databases in specific scenarios where their characteristics align with the requirements of the application. Here are some common scenarios where NoSQL databases are often preferred:

### 1. **Big Data and Real-Time Processing:**

- **Scenario:** When dealing with large volumes of rapidly changing, unstructured, or semi-structured data, such as data generated by social media, IoT devices, or real-time analytics.

## **Module-**

- **Reason:** NoSQL databases are designed to handle the high velocity and variety of big data, providing fast and scalable solutions for real-time data processing.

### **2. Flexible Schema and Schema-less Data:**

- **Scenario:** In applications where the data model is evolving, and a flexible or schema-less approach is needed to accommodate diverse and dynamic data structures.
- **Reason:** NoSQL databases allow for a more dynamic schema, enabling developers to insert data without the need for a predefined schema, making them suitable for agile development and scenarios with evolving data requirements.

### **3. Scalability and High Throughput:**

- **Scenario:** Applications that require horizontal scalability to handle increasing amounts of data and traffic.
- **Reason:** NoSQL databases are designed for easy horizontal scaling by adding more nodes to a distributed system. This makes them well-suited for applications with growing user bases or unpredictable workloads.

### **4. Document-Oriented Data:**

- **Scenario:** When dealing with document-oriented data structures, such as JSON or XML documents, commonly used in content management systems, e-commerce platforms, and applications with complex hierarchical data.
- **Reason:** NoSQL document stores, like MongoDB, are specifically designed for efficient storage and retrieval of document-oriented data.

### **5. Key-Value Pair Storage:**

- **Scenario:** Use cases where data is stored in a simple key-value pair format, such as caching systems, session storage, and certain types of distributed systems.
- **Reason:** NoSQL key-value stores, like Redis, are optimized for fast and efficient retrieval of data based on key lookups.

### **4. Describe characteristics of NoSQL database.**

of the same data item in a replicated distributed system. In ACID, it refers to the fact that a transaction will not violate the integrity constraints specified on the database schema

#### **3.1.5 Characteristics / Features of NoSQL**

<b>UQ.</b>	Describe characteristics of a NoSQL database.	MU- Dec 17, 10 Marks
------------	---	----------------------

**1. Non-relational**

- NoSQL databases never follow the relational model

(New Syllabus w.e.f academic year 22-23) (M7-80)

Tech-Neo Publications...A SACHIN SHAH Venture

- Never provide tables with flat fixed-column records
- Work with self-contained aggregates or BLOBs
- Doesn't require object-relational mapping and data normalization
- No complex features like query languages, query planners, referential integrity joins, ACID

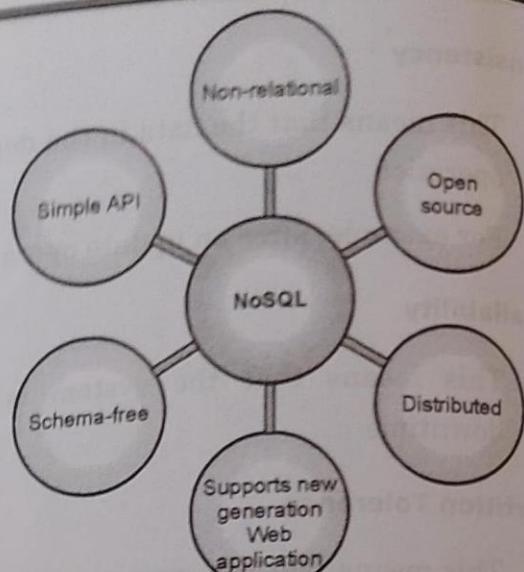


Fig. 3.1.2

#### 2. Open-source

NoSQL databases don't require expensive licensing fees and can run on inexpensive hardware, rendering their deployment cost-effective.

#### 3. Schema-free

- NoSQL databases are either schema-free or have relaxed schemas
- Do not require any sort of definition of the schema of the data
- Offers heterogeneous structures of data in the same domain

#### 4. Simple API

- Offers easy to use interfaces for storage and querying data provided
- APIs allow low-level data manipulation & selection methods
- Text-based protocols mostly used with HTTP REST with JSON
- Mostly used no standard based query language
- Web-enabled databases running as internet-facing services

#### 5. Distributed

- Multiple NoSQL databases can be executed in a distributed fashion
- Offers auto-scaling and fail-over capabilities
- Often ACID concept can be sacrificed for scalability and throughput
- Mostly no synchronous replication between distributed nodes Asynchronous MultiMaster Replication, peer-to-peer, HDFS Replication
- Only providing eventual consistency

5. Elaborate the fundamental differences between NoSQL databases and traditional SQL databases, focusing on key characteristics that make NoSQL unique?

### 3.1.7 Difference between RDBMS and NoSQL

MU- Dec-18, 10 Marks

UQ. Differentiate between a RDBMS and NoSQL database.

Sr. No.	RDBMS	NoSQL
1.	Have fixed or static or predefined schema	Have dynamic schema
2.	Not suited for hierarchical data storage	Best suited for hierarchical data storage
3.	Vertically scalable	Horizontally scalable
4.	Follow ACID property	Follows CAP (consistency, availability, partition tolerance)
5.	Relational Database supports transactions (also complex transactions with joins).	NoSQL databases don't support transactions (support only simple transactions).
6.	Relational database manages only structured data.	NoSQL database can manage structured, unstructured and semi-structured data.
7.	Relational databases have a single point of failure with failover.	NoSQL databases have no single point of failure.
8.	Relational Database supports a powerful query language.	NoSQL Database supports a very simple query language.
9.	It gives only read scalability.	It gives both read and write scalability.
10.	Transactions written in one location.	Transactions written in many locations.
11.	It supports complex transactions.	It supports simple transactions.
12.	It is used to handle data coming in low velocity.	It is used to handle data coming in high velocity.
13.	Examples- MySQL, Oracle, Sqlite, PostgreSQL and MS-SQL etc.	Examples- MongoDB, BigTable, Redis, RavenDB, Cassandra, Hbase, Neo4j, CouchDB etc.

6. How would you articulate the core business drivers that lead organizations to consider adopting NoSQL databases over traditional relational databases?

**Module-**

CHATGPT:

Organizations consider adopting NoSQL databases over traditional relational databases due to several core business drivers that align with the evolving needs of modern applications. These drivers, contributing to the increasing popularity of NoSQL databases, can be summarized as follows:

**1. Agility and Rapid Development :**

- NoSQL databases offer a flexible schema or schema-less design, allowing organizations to adapt quickly to changing data models without the constraints of predefined schemas. This agility is crucial in dynamic business environments where rapid development and iteration are paramount.

**2. Scalability and Performance :**

- NoSQL databases are designed for horizontal scalability, enabling organizations to seamlessly distribute and scale their databases across multiple servers or clusters. This scalability is critical for handling growing datasets, user bases, and unpredictable workloads, ensuring optimal performance and responsiveness.

**3. Handling Big Data and Unstructured Data :**

- NoSQL databases excel in managing large volumes of unstructured or semi-structured data, which is increasingly common in today's data landscape. Organizations dealing with diverse data types, such as social media content, logs, or sensor data, find NoSQL databases more suitable for efficient storage and retrieval.

**4. Real-time Analytics and Processing :**

- NoSQL databases are well-suited for real-time analytics and processing scenarios, providing low-latency access to data. This capability is crucial for applications requiring instantaneous insights, such as real-time monitoring, fraud detection, or personalized user experiences.

**5. Cost-Effectiveness and Open Source Options :**

- NoSQL databases often offer cost-effective solutions, especially when dealing with large datasets and the need for horizontal scaling. Many NoSQL databases are also available as open-source options, reducing licensing costs and fostering a vibrant community of developers and support.

**7. In what scenarios would a company find it most advantageous to switch from a traditional SQL database to a NoSQL solution?**

A company may find it advantageous to switch from a traditional SQL database to a NoSQL solution in various scenarios, depending on its specific needs and the characteristics of its data and application. Here are some scenarios where such a transition could be particularly advantageous:

**Unstructured or Semi-Structured Data (1 mark):**

When the company deals with unstructured or semi-structured data, such as JSON or XML documents, and needs a database that can handle flexible schemas and dynamic data models.

**Scalability Requirements (1 mark):**

When the company anticipates or experiences significant growth in data volume, user base, or transactional load, and requires a database that can scale horizontally to distribute the workload across multiple servers.

**Agile Development and Rapid Iteration (1 mark):**

**Module-**

When the company operates in an environment where rapid development, frequent updates, and the ability to iterate quickly are crucial, and a flexible schema or schema-less design is preferred.

**Real-Time Analytics and Processing (1 mark):**

When the company needs to perform real-time analytics, handle streaming data, or process data with low-latency requirements, and a NoSQL database with efficient support for these use cases is more suitable.

**Varied Data Models (1 mark):**

When the company's application requires support for diverse data models, such as document-oriented, key-value pairs, or graph structures, and a NoSQL solution provides specialized data models for different types of data.

8. How do NoSQL databases align with the needs of businesses that deal with unstructured and rapidly changing data?

NoSQL databases align well with the needs of businesses that deal with unstructured and rapidly changing data due to their inherent characteristics and design principles. Here's how NoSQL databases address the challenges associated with unstructured and dynamic data:

**Flexible Schema:**

Adaptability: NoSQL databases often feature a flexible or schema-less design, allowing businesses to store and manage data without a predefined schema. This adaptability is crucial when dealing with unstructured or evolving data, as the schema can be adjusted on-the-fly to accommodate new data types or changes in the structure.

**Dynamic Data Models:**

Schema Evolution: NoSQL databases support dynamic data models, enabling businesses to evolve their data structures over time. This flexibility is essential in scenarios where the data model is not fixed and needs to accommodate variations or additions without requiring a rigid, predefined schema.

**Document-Oriented Storage:**

Handling Unstructured Data: NoSQL document stores, such as MongoDB, are well-suited for handling unstructured or semi-structured data. Documents can contain nested structures, arrays, and key-value pairs, providing a more natural representation for diverse and dynamic data.

**Horizontal Scalability:**

Scalable Architecture: NoSQL databases are designed for horizontal scalability, allowing businesses to scale out by adding more nodes to a distributed system. This scalability is crucial for accommodating the growing volumes of unstructured data and the increased demands on data processing.

**High Write Throughput:**

Optimized Write Performance: NoSQL databases often offer optimized write performance, making them suitable for scenarios where data is rapidly changing. This is beneficial for applications with high write throughput requirements, such as logging, real-time analytics, and IoT data streams.

**No Joins and Complex Transactions:**

Simplified Operations: NoSQL databases, in some categories like document stores and key-value stores, avoid complex joins and transactions. This simplicity in data operations is advantageous in scenarios where the emphasis is on quick and efficient data access, rather than enforcing complex relational constraints.

9. How NOSQL can be used to handle big data problems.

## 3.7 NOSQL SYSTEMS TO HANDLE BIG DATA PROBLEMS

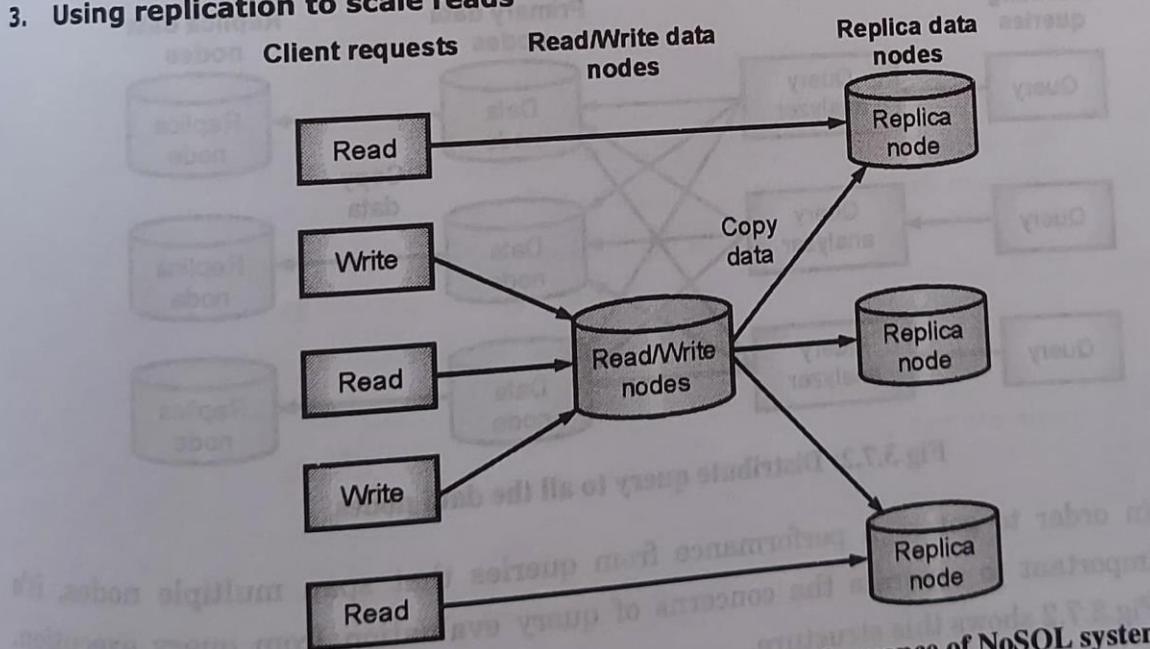
### 1. Moving queries to the data, not data to the queries

- With the exception of large graph databases, most NoSQL systems use commodity processors that each hold a subset of the data on their local shared-nothing drives.
- When a client wants to send a general query to all nodes that hold data, it's more efficient to send the query to each node than it is to transfer large datasets to a central processor. This may seem obvious, but it's amazing how many traditional databases still can't distribute queries and aggregate query results.
- This simple rule helps you understand how NoSQL databases can have dramatic performance advantages over systems that weren't designed to distribute queries to the data nodes.
- Consider an RDBMS that has tables distributed over two different nodes. In order for the SQL query to work, information about rows on one table must all be moved across the network to the other node.
- Larger tables result in more data movement, which results in slower queries. Think of all the steps involved. The tables can be extracted, serialized, sent through the network interface, transmitted over networks, reassembled, and then compared on the server with the SQL query.
- Keeping all the data within each data node in the form of logical documents means that only the query itself and the final result need to be moved over a network. This keeps your big data queries fast.

### 2. Using hash rings to evenly distribute data on a cluster

- One of the most challenging problems with distributed databases is figuring out a consistent way of assigning a document to a processing node.
- Using a hash ring technique to evenly distribute big data loads over many servers with a randomly generated 40-character key is a good way to evenly distribute a network load.
- Hash rings are common in big data solutions because they consistently determine how to assign a piece of data to a specific processor. Hash rings take the leading bits of a document's hash value and use this to determine which node the document should be assigned. This allows any node in a cluster to know what node the data lives on and how to adapt to new assignment methods as your data grows.
- Partitioning keys into ranges and assigning different key ranges to specific nodes is known as keyspace management. Most NoSQL systems, including MapReduce, use keyspace concepts to manage distributed computing problems.
- The concept of a hash ring can also be extended to include the requirement that an item must be stored on multiple nodes.
- When a new item is created, the hash ring rules might indicate both a primary and a secondary copy of where an item is stored. If the node that contains the primary fails, the system can look up the node where these secondary item is stored.

### 3. Using replication to scale reads



**Fig 3.7.1 : Duplication of the Data to increase the performance of NoSQL system**

- The Fig 3.7.1 show how you can replicate data to speed read performance in NoSQL systems. All incoming client requests enter from the left. All reads can be directed to any node, either a primary read/write node or a replica node. All write transactions can be sent to a central read/write node that will update the data and then automatically send the updates to replica nodes. The time between the write to the primary and the time the update arrives on the replica nodes determines how long it takes for reads to return consistent results.
- There are only a few times when you must be concerned about the lag time between a write to the read/write node and a client reading that same record from a replica. One of the most common operations after a write is a read of that same record. If a client does a write and then an immediate read from that same node, there's no problem. The problem occurs if a read occurs from a replica node before the update happens. This is an example of an inconsistent read.
- The best way to avoid this type of problem is to only allow reads to the same write node after a write has been done. This logic can be added to a session or state management system at the application layer. Almost all distributed databases relax database consistency rules when a large number of nodes permit writes. If your application needs fast read/write consistency, you must deal with it at the application layer.

#### 4 Letting the database distribute queries evenly to data nodes

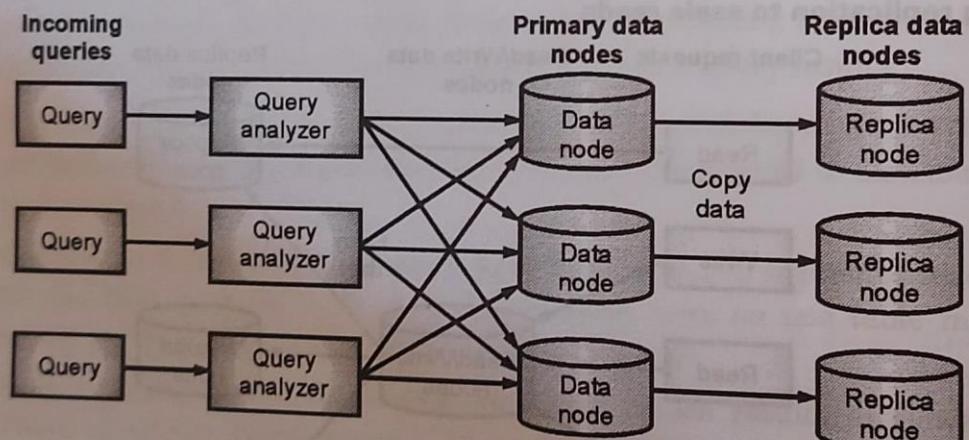


Fig 3.7.2: Distribute query to all the data nodes

- In order to get high performance from queries that span multiple nodes, it's important to separate the concerns of query evaluation from query execution. Fig.3.7.2 shows this structure.

- In this figure NoSQL systems move the query to a data node, but don't move data to a query node. In this example, all incoming queries arrive at query analyzer nodes. These nodes then forward the queries to each data node. If they have matches, the documents are returned to the query node.
- The query won't return until all data nodes (or a response from a replica) have responded to the original query request. If the data node is down, a query can be redirected to a replica of the data node.
- The approach shown in Fig. 3.7.2 is one of moving the query to the data rather than moving the data to the query. This is an important part of NoSQL big data strategies. In this instance, moving the query is handled by the database server, and distribution of the query and waiting for all nodes to respond is the sole responsibility of the database, not the application layer.
- This approach is somewhat similar to the concept of federated search. Federated search takes a single query and distributes it to distinct servers and then combines the results together to give the user the impression they're searching a single system. In some cases, these servers may be in different geographic regions. In this case, sending query to a single cluster that's not only performing search queries on a single local cluster but also performing update and delete operations.

ChapterEnds...

10. Discuss architectural patterns of NoSQL.

### 3.2 NoSQL DATA ARCHITECTURE PATTERNS

**UQ.** What are the different architectural patterns in NoSQL? Explain Graph data store and Column Family Store patterns with relevant examples.  
**(MU - May 19, 10 Marks)**

NoSQL databases were born out of the rigidity of traditional relational or SQL databases, which use tables, columns, and rows to establish relationships across data. Developers welcomed NoSQL databases because they didn't require an upfront schema design; they were able to go straight to development. And it's this flexibility, this "ad-hoc" approach to organizing data, that has arguably been NoSQL's greatest selling point, which continues to appeal to organizations that need to store, retrieve, and analyze either unstructured or rapidly changing data.

The data stored in NoSQL follows any of the four data architecture patterns.

- |                      |                                    |
|----------------------|------------------------------------|
| (A) Key-Value Stores | (B) Column family (Bigtable)Stores |
| (C) Document Stores  | (D) Graph Stores                   |

#### 3.2.1 Key-Value Stores

**UQ.** Explain in detail key-value store NoSQL architectural pattern. Identify two applications that can use this pattern.  
**(MU - May 18, 5 Marks)**

- One of the most basic NoSQL database models is this model. The data is collected in the pattern of Key-Value Pairs, as the name implies. A series of strings, integers or characters is typically the key, but it can also be a more advanced form of data.
- Usually, the value is connected or co-related to the key. The databases for key-value pair storage typically store information as a hash table where each key is unique.

- The value may be of any form (JavaScript Object Notation (JSON), Binary Large Object (BLOB), strings, etc.).
- Application :** This style of architecture is commonly used in shopping websites or e-commerce applications and its important assets is its ability for wide management of data volumes, heavy loads and the ease with which keys are used to retrieve data.

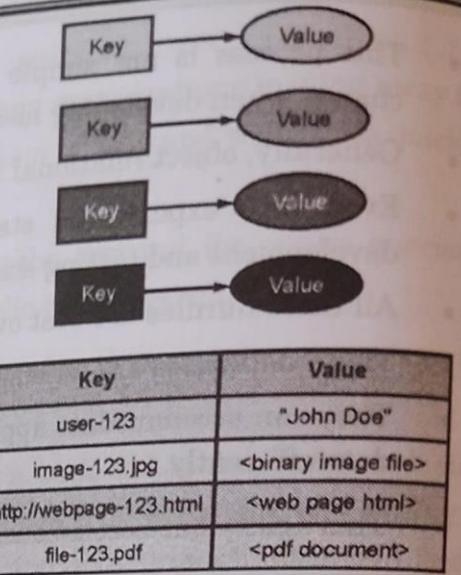


Fig 3.2.1: An example of Key-Value

Keys and values are flexible. Keys can be image names, web page URLs, or file path names that point to values like binary images, HTML web pages, and PDF documents

Constraints associated with the key-value store databases is its complexity in handling queries which will attempt to include many key-value pairs that may delay output and may cause data to clash with many-to-many relationships.

**GQ.** State example of any two key value databases

(2 Marks)

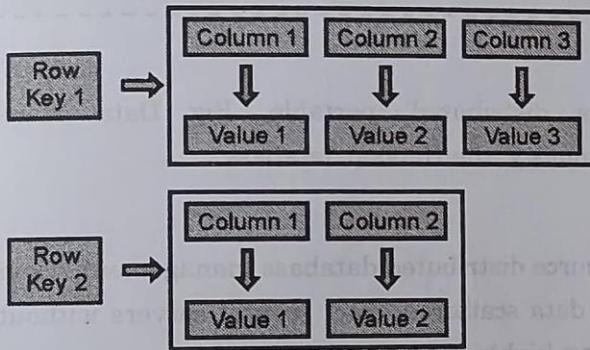
Examples here are:

- DynamoDB (developed by Amazon)
- Berkeley DB (developed by Oracle)
- REDIS:** An advanced open-source key-value store, also referred to as a data structure server because keys can include strings, hashes, lists, sets and sorted sets. This product, written in C/C++, is searingly quick, which makes it perfect for data collection in real time.
- Riak:** An open source that is powerful, distributed database that predictably scales capability and simplifies creation by prototyping, developing, and deploying applications quickly. Written in Erlang and C this technology gives transparent fault-tolerant/fail-over functionality, a comprehensive and versatile API perfect for point-of-sale and factory control systems.
- VoltDB:** scalable database in memory that offers complete transactional ACID consistency and ultra-high throughput, self-referred to as the NewSQL. This technology relies on segmentation and replication to achieve high-availability data snapshots and durable command logging using Java stored processes (for crash recovery), making it ideal for capital markets, digital networks, network services, and for online gaming.

### 3.2.2 Column Store Database

Table 1			
	Column 1	Column 2	Column 3
Row A			

Table 2			
	Column 1	Column 2	Column 3
Row B			



company				super column family	
	name	address		website	
		city	San Francisco	protocol	https
1	DataX	state	California	domain	datax.com
		street num	135	subdomain	www
		street	Kearny St		
		city	Arlington	protocol	https
2	Process-One	state	Virginia	domain	process1.com
		street num	3500	subdomain	www
		street	Wilson St		

row key                          column family

**Fig 3.2.2 : An Example of Column Store**

- This pattern employs data storage in individual cells that is further divided into columns, rather than storing data in relational tuples.
- Databases that are column-oriented operate only on columns. They together store vast quantities of data in columns. The column format and titles will diverge from one row to another.

## Module-

Big Data Analytics (MU-Sem 7-COMP)

- Each column is handled differently, but still, like conventional databases, each individual column will contain several other columns. (Niharika ,2020)
  - Basically, columns are in this sort of storage mode. Data is readily available and it is possible to perform queries such as Number, AVERAGE, COUNT on columns easily.
- The setbacks for this system includes: transactions should be avoided or not supported, queries can decrease high performance with table joins, record updates and deletes reduce storage efficiency, and it can be difficult to design efficient partitioning/indexing schemes.

(2 Marks)

GQ. State example of any two column store databases

Examples here are:

- **HBase:** HBase is a distributed, portable, Big Data Store modelled after Google's BigTable technology, the Hadoop database.
- Google's BigTable
- Cassandra: An open-source distributed database management system built to manage very large volumes of data scattered over several servers without a single point of failure while delivering a highly accessible service.
- Written in Java, this product is best for non-transactional real-time data analysis with linear scalability and proven fault-tolerance combined with column indexes.

### 3.2.3 Document Database

- In the form of key-value pairs, the record database fetches and accumulates information, but here the values are called documents. A complicated data structure can be represented as a text.
- It is hierarchical version of key-value databases.
- The document can be in text form, arrays, strings, JSON (JavaScript Object Notation), XML (Extensible Markup Language) or any other format.
- The use of nested documents is immensely popular. It is highly efficient since most of the generated information is generally in the form of JSONs and is unstructured.

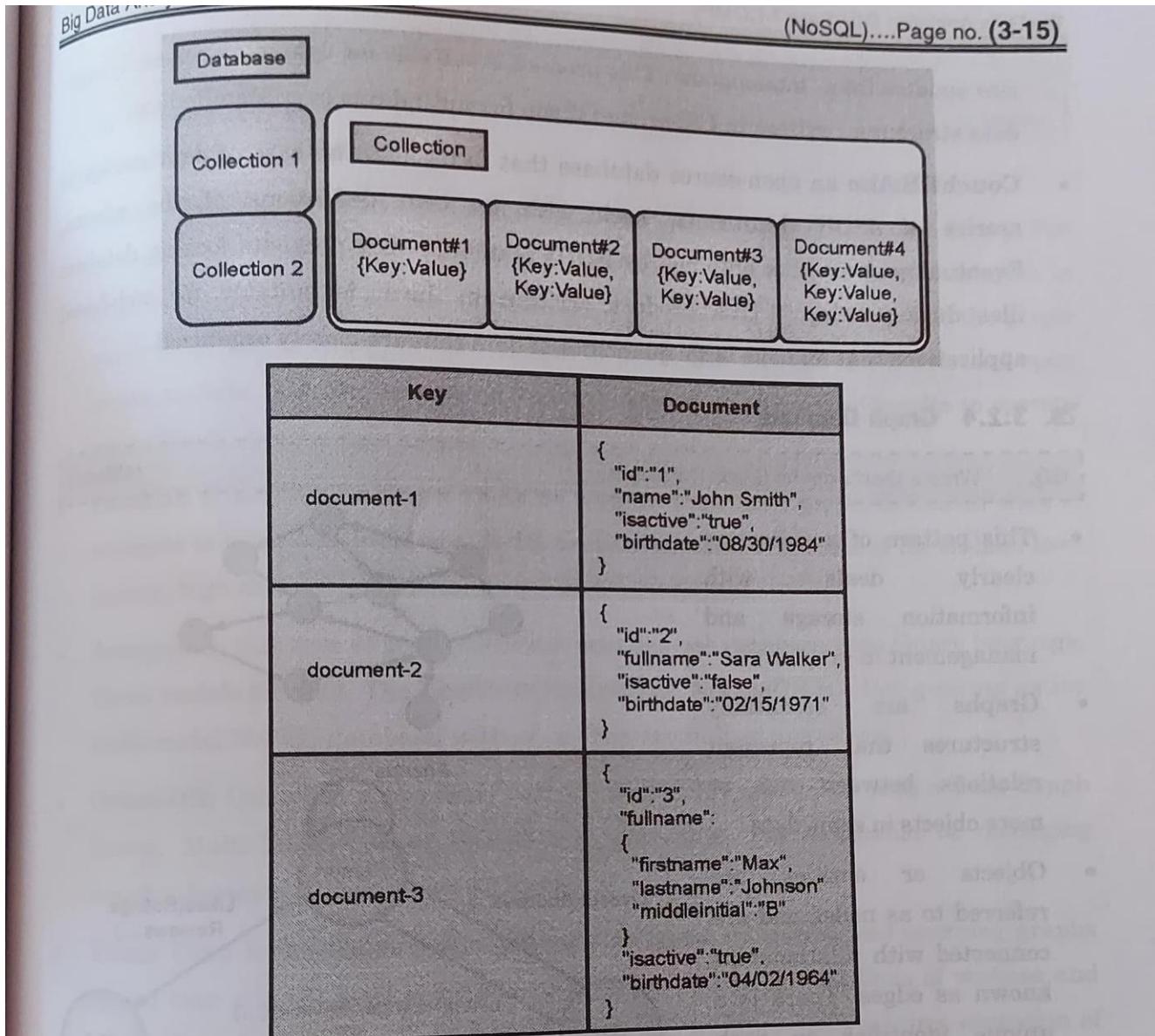


Fig 3.2.3 : An Example of Document

This format is extremely useful and appropriate for semi-structured data, and it is simple to retrieve and handle documents from storage. The drawbacks associated with this system includes the challenging factor of handling multiple documents and the inaccurate working of aggregation operations.

**GQ.** State example of any two document store databases

(2 Marks)

Examples of such databases are:

- **MongoDB:** This scalable, high-performance, open-source NoSQL database features document-oriented (JSON-like) storage, full index support, replication, and fast on-

- **CouchDB:** Also an open-source database that focuses on the ease of data storage in a series of JSON documents, each with its own definitions of the schema. Eventual consistency is enforced by ACID semantics that prevents locking database files during writing. This product, written in Java, is suitable for web-based applications that manage large quantities of data that are loosely organized.

### 3.2.4 Graph Database

**GQ.** Write a short note on Graph Databases. (4Marks)

- This pattern of architecture clearly deals with information storage and management in graphs.
- Graphs are essentially structures that represent relations between two or more objects in some data.
- Objects or entities are referred to as nodes and are connected with relationships known as edges. There is a unique identifier on each edge.

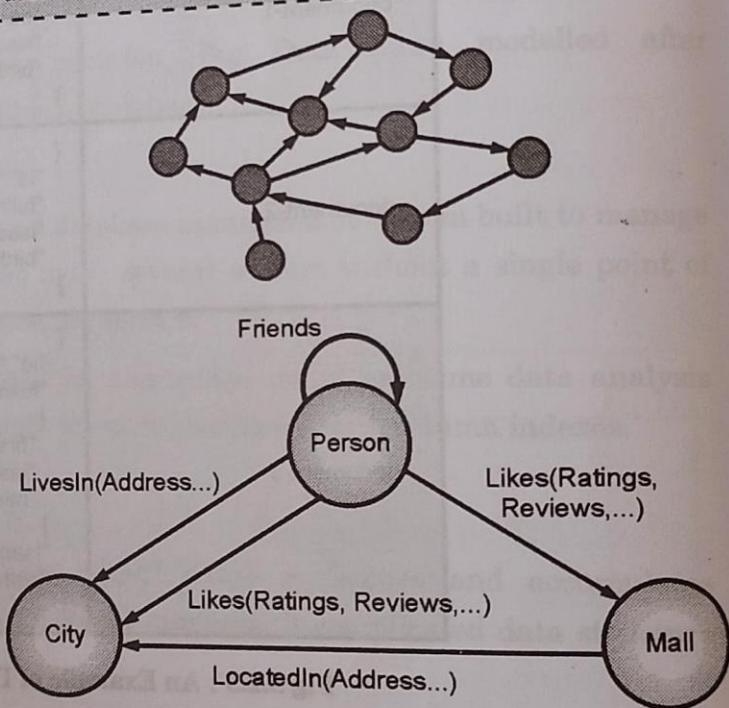


Fig 3.2.4 : An Example of Graph

- For the graph, each node serves as a point of touch. In social networks where there are many and large numbers of entities, this pattern is very widely used and each entity has one or many characteristics that are linked by edges.

There are loosely connected tables in the relational database pattern, whereas graphs are often strong and rigid in nature, have a faster traversal due to connections, and allow spatial data to be easily handled, but incorrect connections can lead to infinite loops. (Ian, 2016)



GQ. State example of any two graph databases

(2 Marks)

Examples of such databases are:

- **Neo4J:** The leading native graph database and graph platform is Neo4J: Neo4j. For enterprise levels of security and high performance and reliability by clustering, it is available both as open source and through a commercial license. Cypher, the graph query language of Neo4j, is very simple to learn and can use newly released open source toolkits, "Cypher on Apache Spark (CApS) and Cypher for Gremlin to operate across Neo4j, Apache Spark and Gremlin-based products."
- **FlockDB (Used by Twitter):** FlockDB is easier than other graph databases since it attempts to solve less problems. It fits horizontally and is optimized for on-line, low-latency, high throughput environments such as websites.
- **ArangoDB:** this type of graph database requires one database, One Query language, Three models for data. The Limitless Possibilities. ArangoDB is a fast-growing native multi-model NoSQL database, with more than one million downloads.
- **OrientDB:** OrientDB is the first Distributed DBMS multi-model with a True Graph Driver. Multi-Model means NoSQL 2nd generation that is capable of managing complex domains with amazing efficiency.
- **Titan:** Titan is a scalable graph database designed for storing and querying graphs spread over a multi-machine cluster comprising hundreds of billions of vertices and edges. Titan is a transactional database that can facilitate the real-time execution of complex graph traversals by thousands of concurrent users.
- **DataStax:** In a rapidly changing environment where aspirations are strong, DataStax helps businesses thrive and new technologies occur daily.
- **Amazon Neptune:** Amazon Neptune with a highly connected datasets to create and run applications is secure, fast and has a fully managed graph database service that is easy. (Niharika, 2020).

11. Describe the key-value store architecture pattern in NoSQL.

ABOVE ANSWER FIRST THREE IMAGES

12. In what situations would you recommend using a key-value store over other NoSQL architecture patterns?

CHATGPT:

## **Module-**

A key-value store is a NoSQL database architecture pattern that is particularly suitable for specific use cases. Here are situations in which you might recommend using a key-value store over other NoSQL architecture patterns:

### **1. Simple Data Model (1 mark):**

- **Situation:** When the data model is simple and can be represented as key-value pairs without the need for complex relationships or nested structures.
- **Reason:** Key-value stores are designed for simplicity, making them efficient for scenarios where the data can be adequately represented using a basic key-value structure.

### **2. High Read and Write Throughput (1 mark):**

- **Situation:** When the application demands high read and write throughput and requires a database that can handle a large volume of simple, frequently accessed data.
- **Reason:** Key-value stores are optimized for performance, especially in read and write operations, making them suitable for scenarios where speed and low-latency access are critical.

### **3. Caching and Session Storage (1 mark):**

- **Situation:** When the primary use case involves caching frequently accessed data or storing session-related information for quick retrieval.
- **Reason:** Key-value stores excel in scenarios where data needs to be rapidly retrieved based on a unique key, making them ideal for caching and session storage use cases.

### **4. Distributed Systems and Partitioning (1 mark):**

- **Situation:** When building distributed systems that require easy partitioning and distribution of data across multiple nodes or servers.
- **Reason:** Key-value stores typically offer straightforward partitioning strategies, making them well-suited for distributed architectures where data needs to be distributed across nodes for horizontal scalability.

### **5. Low Overhead and Simplicity (1 mark):**

- **Situation:** When the goal is to minimize database management overhead and complexity, and a simple data model aligns with the application's requirements.
- **Reason:** Key-value stores often have minimal complexity, with basic operations centered around key lookups. This simplicity can lead to easier maintenance and management.

### **6. Stateless Applications (1 mark):**

- **Situation:** In stateless applications where data is transient, and the primary requirement is rapid and efficient storage and retrieval of information.
- **Reason:** Key-value stores are well-suited for scenarios where the application does not require complex queries or transactions and can benefit from the simplicity and speed of key-based access.

13. Can you describe a use case where a graph store significantly outperforms other NoSQL architectures due to its ability to represent and traverse relationships?

CHATGPT:

One use case where a graph store significantly outperforms other NoSQL architectures is in social network applications. Social networks involve complex relationships between users, and the ability to represent and traverse these relationships efficiently is crucial for delivering optimal performance and user experience. Here's how a graph store excels in this scenario:

## **Module-**

### **Use Case: Social Network Application**

#### **Challenges:**

##### **1. Complex Relationships (1 mark):**

- Social networks consist of intricate and interconnected relationships between users. Users can be friends, followers, or have other types of connections. Representing and navigating these complex relationships can be challenging with other NoSQL architectures.

##### **2. Dynamic Graph Structure (1 mark):**

- The social graph is dynamic, with users forming and breaking connections in real-time. A data model that supports dynamic updates and efficient traversal of the graph is essential to reflect the current state of relationships accurately.

##### **3. Recommendation Systems (1 mark):**

- Many social networks use recommendation systems to suggest new connections based on mutual friends, interests, or other factors. Efficient graph traversal is critical for generating accurate and timely friend recommendations.

#### **How a Graph Store Excels:**

##### **1. Efficient Relationship Representation (1 mark):**

- **Reason:** Graph stores, such as Neo4j, are designed specifically to represent and traverse relationships efficiently. Nodes in the graph can represent users, and edges can represent different types of relationships, making it intuitive to model and query complex social structures.

##### **2. Traversal and Pathfinding (1 mark):**

- **Reason:** Graph databases excel in traversing relationships between nodes. Algorithms for pathfinding, such as finding the shortest path between users or identifying common connections, can be performed efficiently, providing quick responses to user queries.

##### **3. Real-Time Updates (1 mark):**

- **Reason:** Graph stores support real-time updates and queries, allowing the system to reflect changes in the social graph immediately. This is crucial for maintaining an accurate representation of the network as users connect or disconnect.

##### **4. Recommendation Systems (1 mark):**

- **Reason:** Graph databases enable the implementation of sophisticated recommendation systems. Algorithms can traverse the graph to identify potential connections based on shared interests, mutual friends, or other relevant factors, providing personalized and accurate recommendations.

##### **5. Query Expressiveness (1 mark):**

- **Reason:** Graph query languages, such as Cypher for Neo4j, are designed to express complex queries for traversing relationships with ease. This allows developers to write expressive and efficient queries for various use cases within the social network application.

14. Provide an example scenario where a graph store NoSQL database would be more beneficial than other NoSQL architectures?

CHATGPT:

**Scenario: Fraud Detection System**

## **Module-**

**Overview:** Imagine a financial institution deploying a fraud detection system to identify and prevent fraudulent activities in real-time. In this scenario, a graph store NoSQL database, such as Neo4j, would be more beneficial than other NoSQL architectures.

### **Challenges:**

#### **1. Complex Relationships (1 mark):**

- Fraud detection involves analyzing intricate relationships between entities, such as account holders, transactions, IP addresses, and devices. Representing and traversing these complex relationships is crucial for identifying patterns indicative of fraudulent behavior.

#### **2. Dynamic Graph Structure (1 mark):**

- The graph of relationships in a fraud detection system evolves dynamically as new transactions occur and new connections between entities emerge. A data model that accommodates these dynamic changes and allows for efficient traversal is essential.

#### **3. Pattern Recognition (1 mark):**

- Detecting fraud often requires identifying patterns or anomalies within the relationships between seemingly unrelated entities. Traditional tabular or document-based data models may struggle to express and query these complex patterns effectively.

### **Benefits of a Graph Store:**

#### **1. Efficient Relationship Representation (1 mark):**

- **Reason:** A graph database excels at representing complex relationships. Nodes can represent entities like account holders, transactions, and devices, while edges denote relationships such as transactions between accounts or shared IP addresses.

#### **2. Traversal for Pattern Detection (1 mark):**

- **Reason:** Graph databases are designed for efficient traversal of relationships. Algorithms can traverse the graph to identify patterns indicative of fraudulent activities, such as circular transactions, shared IP addresses among seemingly unrelated accounts, or patterns of suspicious behavior.

#### **3. Real-Time Updates (1 mark):**

- **Reason:** Fraud detection systems require real-time updates to adapt to emerging threats. A graph store allows for immediate updates to the graph as new transactions occur or as relationships change, ensuring the system reflects the most current state of potential fraud.

#### **4. Connectivity Analysis (1 mark):**

- **Reason:** Graph databases excel in connectivity analysis. Detecting fraud often involves identifying clusters of interconnected entities engaging in suspicious activities. Graph algorithms can efficiently analyze these patterns, aiding in the identification of fraudulent networks.

#### **5. Query Expressiveness (1 mark):**

- **Reason:** Graph query languages, such as Cypher for Neo4j, offer expressive syntax for querying complex relationships. This allows fraud detection analysts to formulate sophisticated queries to uncover intricate patterns and anomalies within the graph.

15. Provide examples of industries or applications where a column family store is particularly well-suited and explain why.

## **Module-**

A column-family store is particularly well-suited for certain industries and applications that involve handling and analyzing large volumes of data with a focus on read and write performance. Here are examples of industries and applications where a column-family store, such as Apache Cassandra, is commonly employed and the reasons for its suitability:

### **1. Big Data Analytics (1 mark):**

- **Industry/Application:** Big data analytics platforms and frameworks.
- **Reason:** Column-family stores excel in handling large-scale analytics workloads. The columnar storage allows for efficient read and write operations on massive datasets, making them well-suited for applications where rapid data analysis and retrieval are crucial.

### **2. Content Management Systems (1 mark):**

- **Industry/Application:** Content management and digital publishing platforms.
- **Reason:** Column-family stores are well-suited for managing and retrieving large volumes of content-related data, such as articles, images, and metadata. The schema flexibility and ability to add new columns dynamically make them adaptable to the evolving content structure.

### **3. User Profile and Personalization (1 mark):**

- **Industry/Application:** E-commerce platforms, social media networks, and recommendation engines.
- **Reason:** Column-family stores can efficiently store and retrieve user-related data, such as preferences, purchase history, and social connections. The ability to add new attributes to user profiles without modifying the entire schema makes them suitable for dynamic and personalized applications.

### **4. Log and Event Data (1 mark):**

- **Industry/Application:** Logging systems, cybersecurity platforms, and event tracking.
- **Reason:** Column-family stores are well-suited for handling log and event data, where each column family can represent a different type of event or log entry. This allows for quick retrieval and analysis of specific types of events without scanning the entire dataset.

### **5. Data Warehousing (1 mark):**

- **Industry/Application:** Data warehousing and business intelligence.
- **Reason:** Column-family stores can efficiently store and retrieve large datasets used in data warehousing scenarios. The columnar storage format is conducive to analytics queries, where typically only a subset of columns is needed for analysis.

### **Inventory Management and Supply Chain (1 mark):**

**Industry/Application:** Supply chain management systems, inventory tracking, and order processing.

**Reason:** Column-family stores can efficiently handle the complex and dynamic data associated with inventory management, such as tracking product quantities, order histories, and supplier information. The ability to add new attributes dynamically is advantageous in evolving supply chain scenarios.

## **16. Discuss the querying capabilities of document stores. How do they handle complex queries and aggregations?**

**CHATGPT:**

Document stores, a type of NoSQL database, are designed to store and retrieve data in flexible, semi-structured, or structured documents (such as JSON or BSON). Querying capabilities in document stores vary between databases, but some common features and approaches are shared across many of them. Let's discuss how document stores handle complex queries and aggregations:

## **Module-**

### **1. Document-Oriented Query Language:**

- Document stores often come with a document-oriented query language that allows developers to express queries using the structure of the stored documents. Examples include MongoDB's query language and CouchDB's MapReduce queries.

### **2. Flexible Schema and Nested Documents:**

- Document stores support a flexible schema, allowing documents within the same collection to have different fields. This flexibility accommodates variations in data structure. Queries can target specific fields, and nested documents can be queried to access and filter data within substructures.

### **3. Indexing:**

- Document stores typically provide indexing mechanisms to improve query performance. By creating indexes on specific fields, queries can efficiently retrieve documents based on the indexed criteria. Indexing is crucial for handling complex queries in large datasets.

### **4. Filtering and Projection:**

- Document stores support filtering and projection to specify criteria for selecting documents and controlling which fields to include or exclude in query results. This allows for the retrieval of specific subsets of data and improves query efficiency.

### **5. Aggregation Frameworks:**

- Many document stores offer aggregation frameworks that enable the processing and transformation of documents to produce aggregated results. For example, MongoDB's Aggregation Pipeline allows for complex transformations, grouping, sorting, and other operations to be performed on the data.

### **6. MapReduce:**

- Some document stores, such as CouchDB and MongoDB, support MapReduce for handling complex queries and aggregations. MapReduce is a programming model that allows developers to define custom functions for mapping, grouping, and reducing data.

## Module-

17. What does the CAP theorem state?

### 3.1.4 CAP Theorem

GQ. What is CAP Theorem? How it is applicable to NOSQL systems? (4 Marks)

It plays important role in NoSQL databases. CAP theorem is also called brewer's theorem which states that it is impossible for a distributed data store to offer more than two out of three guarantees:

So basically, some NoSQL databases offer consistency and partition tolerance. While some offer availability and partition tolerance. But partition tolerance is common as NoSQL databases are distributed in nature so based on requirement, we can choose NoSQL database has to be used. Different types of NoSQL databases are available based on data models.

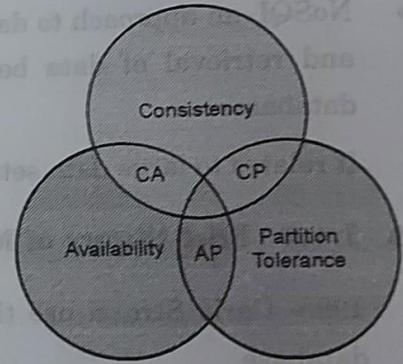


Fig 3.1.1 :CAP Property

**Consistency**

- This means that the data in the database remains consistent after the execution of an operation.
- For example, after an update operation all clients see the same data.

**Availability**

- This means that the system is always on (service guarantee availability), no downtime.

**Partition Tolerance**

- This means that the system continues to function even the communication among the servers is unreliable, i.e., the servers may be partitioned into multiple groups that cannot communicate with one another.

In theoretically it is impossible to fulfil all 3 requirements. CAP provides the basic requirements for a distributed system to follow 2 of the 3 requirements. Therefore, all the current NoSQL database follow the different combinations of the C, A, P from the CAP theorem.

Here is the brief description of **three combinations CA, CP, AP** :

- CA - Single site cluster, therefore all nodes are always in contact. When a partition occurs, the system blocks.
- CP - Some data may not be accessible, but the rest is still consistent/accurate.
- AP - System is still available under partitioning, but some of the data returned may be inaccurate.

The use of the word consistency in CAP and its use in ACID do not refer to the same identical concept.

In CAP, the term consistency refers to the consistency of the values in different copies of the same data item in a replicated distributed system. In ACID, it refers to the fact that a transaction will not violate the integrity constraints specified on the database schema

**3.1.5 Characteristics / Features of NoSQL**

**UQ.** Describe characteristics of a NoSQL database.

**MU- Dec 17, 10 Marks**

**1. Non-relational**

- NoSQL databases never follow the relational model



18. Define each of the three terms in the CAP theorem: Consistency, Availability, and Partition Tolerance. How are these terms specifically defined in the context of distributed systems?

## **Module-**

ABOVE IMAGES.

19. A startup is considering different database options for their new project. They are curious about NoSQL databases but unsure about how they differ from traditional SQL databases. Provide them with a comprehensive overview of NoSQL, highlighting the fundamental differences and advantages of using NoSQL in certain scenarios.

CHATGPT:

### **1. Data Model:**

- **SQL Databases:**

- Follow a structured, tabular data model with predefined schemas.
  - Use SQL (Structured Query Language) for querying and defining the schema.
  - Well-suited for applications with well-defined and stable data structures.
- **NoSQL Databases:**
- Embrace flexible or schema-less data models.
  - Can handle unstructured, semi-structured, or structured data.
  - Well-suited for dynamic, evolving, or diverse data structures.

### **2. Scalability:**

- **SQL Databases:**

- Typically scale vertically by adding more resources to a single server.
- Vertical scaling has limitations in handling large volumes of data and traffic.

- **NoSQL Databases:**

- Designed for horizontal scalability, allowing data distribution across multiple servers or nodes.
- Ideal for applications with rapidly growing data and user bases.

### **3. Query Language:**

- **SQL Databases:**

- Utilize SQL for querying and manipulating data.
- Support complex queries, joins, and transactions.

- **NoSQL Databases:**

- Use various query languages specific to the database type (e.g., MongoDB's query language or Cassandra Query Language).
- May not support complex joins but excel in specific query patterns.

### **4. Schema Design:**

- **SQL Databases:**

- Require a predefined schema with a fixed structure.
- Changes to the schema can be challenging and may require downtime.

- **NoSQL Databases:**

- Support dynamic and flexible schemas.
- Easily accommodate changes to the data model without downtime.

### **5. Use Cases:**

## **Module-**

- **SQL Databases:**
  - Well-suited for applications with well-defined relationships and transactions, such as financial systems and traditional business applications.
- **NoSQL Databases:**
  - Ideal for scenarios involving large-scale data, real-time analytics, content management, social networks, IoT, and applications with dynamic or evolving data structures.

## **6. Consistency and Transactions:**

- **SQL Databases:**
  - Emphasize ACID properties (Atomicity, Consistency, Isolation, Durability).
  - Strong consistency and support for complex transactions.
- **NoSQL Databases:**
  - Embrace the CAP theorem (Consistency, Availability, Partition Tolerance).
  - Prioritize availability and partition tolerance over strict consistency, making them suitable for distributed and fault-tolerant systems.

## **7. Advantages of NoSQL:**

- **Scalability:** NoSQL databases can easily scale horizontally to handle increased load.
- **Flexibility:** They accommodate changes in data structure without downtime.
- **Performance:** NoSQL databases often provide high performance for specific use cases, such as read and write-intensive operations.
- **Diverse Data Models:** Different NoSQL types (document stores, key-value stores, graph databases) allow choosing the most suitable model for specific data needs.
- **Cost-Effective Scaling:** Horizontal scaling with commodity hardware is often more cost-effective than vertical scaling.

20. A company has been using traditional relational databases for several years. They are facing challenges in handling the increasing volume of data and need to scale their system. Explain to them the key concepts of NoSQL databases and the business drivers that make NoSQL a suitable choice for their evolving needs.

### **Key Concepts of NoSQL Databases:**

1. **Flexible Schema:**
  - NoSQL databases offer a flexible schema, allowing the storage of data without a predefined structure. This flexibility is particularly beneficial when dealing with diverse and evolving data models.
2. **Scalability:**
  - NoSQL databases are designed for horizontal scalability, enabling companies to scale out by adding more servers or nodes to a distributed system. This approach is well-suited for handling the increasing volume of data and traffic.
3. **Diverse Data Models:**
  - NoSQL databases come in various types, including document stores, key-value stores, column-family stores, and graph databases. Each type is optimized for specific use cases, allowing companies to choose the most suitable data model for their application.
4. **CAP Theorem:**

## **Module-**

- NoSQL databases operate based on the CAP theorem, emphasizing trade-offs between Consistency, Availability, and Partition Tolerance. NoSQL databases often prioritize Availability and Partition Tolerance, making them suitable for distributed and fault-tolerant systems.

### **5. Polyglot Persistence:**

- Companies can adopt a polyglot persistence strategy, using multiple NoSQL databases alongside traditional relational databases to address different aspects of their data storage needs.

### **Business Drivers for NoSQL Adoption:**

#### **1. Scalability Requirements (1 mark):**

- **Challenge:** Traditional relational databases may face challenges in scaling horizontally to handle growing data volumes and increased user activity.
- **NoSQL Solution:** NoSQL databases are designed for easy horizontal scalability, allowing companies to distribute data across multiple servers to meet scalability requirements effectively.

#### **2. Flexible Data Models (1 mark):**

- **Challenge:** Evolving and diverse data structures may be difficult to accommodate in a rigid relational schema.
- **NoSQL Solution:** NoSQL databases support flexible and dynamic schemas, enabling companies to adapt to changing data structures without the need for extensive schema modifications.

#### **3. High Write Throughput (1 mark):**

- **Challenge:** Traditional databases may face performance issues when dealing with high write-intensive workloads.
- **NoSQL Solution:** NoSQL databases, particularly key-value stores and column-family stores, are optimized for high write throughput, making them suitable for applications with frequent data updates.

#### **4. Unstructured and Semi-Structured Data (1 mark):**

- **Challenge:** Handling unstructured or semi-structured data efficiently can be challenging for traditional relational databases.
- **NoSQL Solution:** NoSQL databases, especially document stores and key-value stores, excel in managing diverse data types, such as JSON or XML documents, providing efficient storage and retrieval.

#### **5. Real-Time Analytics (1 mark):**

- **Challenge:** Traditional databases may struggle to provide real-time analytics on large datasets.
- **NoSQL Solution:** NoSQL databases are often optimized for real-time analytics and processing, making them suitable for applications requiring immediate insights from rapidly changing data.

21. Consider an e-commerce platform where users can make purchases and view product availability. In the event of a network partition, how would you ensure that users can still browse available products while maintaining data consistency?

#### **CHATGPT:**

In the context of an e-commerce platform where users can make purchases and view product availability, ensuring a seamless user experience during a network partition is crucial. One approach to handle this situation involves leveraging concepts from the CAP theorem (Consistency, Availability, and Partition

## **Module-**

Tolerance) and adopting strategies that prioritize availability and partition tolerance while maintaining eventual consistency. Here are some techniques to address this scenario:

### **1. Read Availability with Eventual Consistency:**

- Design the system to prioritize read availability, even in the presence of a network partition. Allow users to browse and view product availability without being blocked by consistency constraints.

### **2. Replication and Data Partitioning:**

- Implement data replication across multiple nodes and employ a distributed database architecture. Use techniques like sharding to partition data, ensuring that product information is distributed across nodes to enhance availability and fault tolerance.

### **3. Use of Caching:**

- Employ caching mechanisms strategically to store frequently accessed product information. This can help mitigate the impact of network partitions on read operations, allowing users to access product details from local caches even if direct database access is temporarily disrupted.

### **4. Asynchronous Updates:**

- Allow the e-commerce platform to accept user requests for product views and purchases even during a network partition. Queue updates and transactions asynchronously, and apply them to the database once network connectivity is restored. This approach prioritizes availability while ensuring eventual consistency.

### **5. Conflict Resolution Strategies:**

- Implement conflict resolution strategies for scenarios where updates occur independently on different nodes during a partition. Techniques such as last-write-wins or manual conflict resolution can be employed to reconcile conflicting changes when the network is restored.

### **6. Offline Mode:**

- Provide users with an offline mode that allows them to continue browsing products and viewing availability information locally on their devices. When network connectivity is restored, synchronize local changes with the central database.

### **7. Graceful Degradation:**

- Implement a graceful degradation strategy where certain non-critical functionalities may be temporarily disabled during a network partition, ensuring that core features like product browsing and availability checks remain accessible.

### **8. Load Balancing and Failover:**

- Implement load balancing and failover mechanisms to redirect user requests to available and responsive nodes. This helps maintain a level of service even when some parts of the network are temporarily unreachable.

## **Module-**

### **4: Mining Data Streams**

4.1 The Stream Data Model: A Data-Stream-Management System, Examples of Stream Sources, Stream Queries, Issues in Stream Processing.

4.2 Sampling Data techniques in a Stream

4.3 Filtering Streams: Bloom Filter with Analysis.

4.4 Counting Distinct Elements in a Stream, Count Distinct Problem, Flajolet-Martin Algorithm, Combining Estimates, Space Requirements

4.5 Counting Ones in a Window: The Cost of Exact Counts, The Datar-Gionis-Indyk-Motwani Algorithm, Query Answering in the DGIM Algorithm, Decaying Windows.

### **Practice Questions**

1. What is a Data-Stream-Management System (DSMS)?

#### **4.2.1 Data Stream Management System (DSMS)**

**UQ.** Explain with block diagram architecture of Data stream Management System.

**(MU - Dec. 19, 10 Marks)**

**UQ.** Explain abstract architecture of Data Stream Management System (DSMS).

**(MU - Dec. 16, 10 Marks)**

**UQ.** What is Data Stream Management System? Explain with block diagram.

**(MU - May 17, 10 Marks)**

- We can view a stream processor as a kind of data-management system, the high-level organization which is shown in Fig. 4.2.1.
- Any number of streams can enter the system. Each stream can provide elements at its own schedule; they need not have the same data rates or data types, and the time between elements of one stream need not be uniform.
- The fact that the rate of arrival of stream elements is not under the control of the system distinguishes stream processing from the processing of data that goes on within a database-management system. The latter system controls the rate at which data is read from the disk, and therefore never has to worry about data getting lost as it attempts to execute queries.
- Streams may be archived in a large archival store, but we assume it is not possible to answer queries from the archival store.

It could be examined only under special circumstances using time-consuming retrieval processes.

There is also a working store, into which summaries or parts of streams may be placed, and which can be used for answering queries.

The working store might be disk, or it might be main memory, depending on how fast we need to process queries. But either way, it is of sufficiently limited capacity that it cannot store all the data from all the streams.

A Data Stream Management System (DSMS) is a computer software system to manage continuous **Data Streams**. It is similar to a **Database Management System** (DBMS), which is, however, designed for static data in conventional **Databases**.

A DBMS also offers a flexible query processing so that the information needed can be expressed using queries. However, in contrast to a DBMS, a DSMS executes a continuous query that is not only performed once, but is permanently installed. Therefore, the query is continuously executed until it is explicitly uninstalled. Since most DSMS are data-driven, a continuous query produces new results as long as new data arrive at the system. This basic concept is similar to **Complex Event Processing** so that both technologies are partially coalescing.

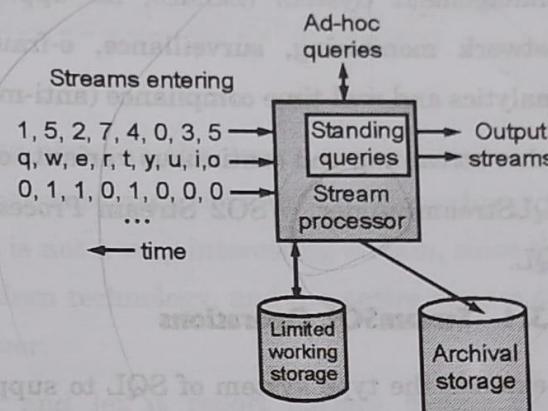


Fig. 4.2.1 : A data-stream-management system

2. Explain with Block Diagram of Data-Stream-Management System.

ABOVE IMAGES:

3. Discuss the components of DSMS (Data Stream Management System) architecture.

CHATGPT:

A Data Stream Management System (DSMS) is designed to handle continuous streams of data in real-time, processing, analyzing, and responding to the data as it is generated. DSMS architecture consists of various components that work together to manage and process data streams efficiently. Here are the key components of a Data Stream Management System architecture:

#### 1. Data Stream Sources:

- These are the origins of the continuous data streams. Sources can include sensors, IoT devices, social media feeds, log files, financial transactions, or any other system generating real-time data. DSMS is designed to ingest and process data from diverse sources.

#### 2. Stream Processing Engine:

**Module-**

- The core of a DSMS is the stream processing engine. This engine is responsible for processing the incoming data streams in real-time. It performs operations such as filtering, aggregation, transformation, and analysis on the data as it flows through the system.

**3. Query Language:**

- DSMS architectures typically include a query language or query interface that allows users to define and express operations on the streaming data. This can include SQL-like languages or domain-specific languages designed for stream processing.

**4. Query Processor:**

- The query processor interprets and executes the queries written by users. It translates the high-level queries into low-level operations that can be efficiently applied to the streaming data. Optimization techniques may be applied to improve the performance of query execution.

**5. Operators and Functions:**

- DSMS includes a set of operators and functions that define the processing logic to be applied to the data streams. These include operations like filtering, mapping, aggregation, pattern matching, and windowing. Operators are fundamental building blocks for constructing streaming queries

4. Provide an example of a stream source.

### 4.3.2 Examples of Stream Sources

GQ. Explain different types of Stream sources.

#### Sensor Data

- Imagine a temperature sensor bobbing about in the ocean, sending back to a base station a reading of the surface temperature each hour. The data produced by this sensor is a stream of real numbers. It is not a very interesting stream, since the data rate is so low. It would not stress modern technology, and the entire stream could be kept in main memory, essentially forever.
- Now, give the sensor a GPS unit, and let it report surface height instead of temperature. The surface height varies quite rapidly compared with temperature, so we might have the sensor send back a reading every tenth of a second. If it sends a 4-byte real number each time, then it produces 3.5 megabytes per day. It will still take some time to fill up main memory, let alone a single disk.
- But one sensor might not be that interesting. To learn something about ocean behavior, we might want to deploy a million sensors, each sending back a stream, at the rate of ten per second. A million sensors isn't very many; there would be one for every 150 square miles of ocean. Now we have 3.5 terabytes arriving every day, and we definitely need to think about what can be kept in working storage and what can only be archived.

#### Image Data

- Satellites often send down to earth streams consisting of many terabytes of images per day. Surveillance cameras produce images with lower resolution than satellites, but there can be many of them, each producing a stream of images at intervals like one second.
- London is said to have six million such cameras, each producing a stream.

#### Internet and Web Traffic

- A switching node in the middle of the Internet receives streams of IP packets from many inputs and routes them to its outputs. Normally, the job of the switch is to transmit data and not to retain it or query it. But there is a tendency to put more capability into the switch, e.g., the ability to detect denial-of-service attacks or the ability to reroute packets based on information about congestion in the network.

## **Module-**

5. Identify two major issues in stream processing.

The issues in stream processing mainly arise because of the following two basic reasons :

1. The rapid rate of arrival of stream data, and 2
2. . The huge size of the data when all of the input streams are considered.

Because of the rapid arrival of stream data, the processing speed also must match the arrival speed of the data. To achieve this, the entire stream processing algorithm must reside in main memory and there should be minimal secondary memory accesses. This is because the secondary memory accesses are many times slower than the main memory accesses.

In the case of a slow stream it may be possible to process the data in that stream using a small portion of main memory. But if the number of such slow streams becomes large then again, the problem of shortage of main memory arises. – Thus, one way to solve the issues related to stream processing is by having a large amount of main memory.

But in real life compute nodes the amount of main memory available is usually limited which makes it unfeasible. So, we have to resort to some other way for solving the issues. One such way is to use the technique of sampling.

6. What are the challenges of querying on large data stream?

**4.4 KEY ISSUES IN BIG DATA STREAM ANALYSIS**

**UQ.** What are the challenges of querying on large data stream? **(MU - May 18, 5 Marks)**

**Scalability**

- One of the main challenges in big data streaming analysis is the issue of scalability. The big data stream Is experiencing exponential growth in a way much faster than computer resources.
- The processors follow Moore's law, but the size of data is exploding. Therefore, research efforts should be geared towards developing scalable frameworks and algorithms that will accommodate data stream computing mode, effective resource allocation strategy and parallelization issues to cope with the ever-growing size and complexity of data.

(New Syllabus w.e.f academic year 22-23) (M7-80)

 Tech-Neo Publications...A SACHIN SHAH Venture

**Integration**

- Building a distributed system where each node has a view of the data flow, that is, every node performing analysis with a small number of sources, then aggregating these views to build a global view is non-trivial.
- An integration technique should be designed to enable efficient operations across different datasets.

**Fault-tolerance**

High fault-tolerance is required in life-critical systems.

- As data is real-time and infinite in big data stream computing environments, a good scalable high fault-tolerance strategy is required that allows an application to continue working despite component failure without interruption.

**Timeliness**

- Time is of the essence for time-sensitive processes such as mitigating security threats, thwarting fraud, or responding to a natural disaster.
- There is a need for scalable architectures or platforms that will enable continuous processing of data streams which can be used to maximize the timeliness of data.
- The main challenge is implementing a distributed architecture that will aggregate local views of data into global view with minimal latency between communicating nodes.

**Consistency**

- Achieving high consistency (i.e. stability) in big data stream computing environments is non-trivial as it is difficult to determine which data are needed and which nodes should be consistent.
- Hence a good system structure is required.

**Heterogeneity and incompleteness**

- Big data streams are heterogeneous in structure, organisations, semantics, accessibility and granularity. The challenge here is how to handle an always ever-increasing data, extract meaningful content out of it, aggregate and correlate streaming data from multiple sources in real-time.
- A competent data presentation should be designed to reflect the structure, diversity and hierarchy of the streaming data.

## Module-

- How stream processing is different from batch processing?

### Batch processing v/s Streaming processing

Aspect	Batch Processing	Stream Processing
Data Flow	Data collected and processed in fixed-size batches	Continuous and real-time data flow in small units (events)
Real-Time Processing	Typically performs processing offline or on a scheduled basis, resulting in higher latency	Processes data as it arrives, aiming for real-time or near-real-time analysis
Latency	Higher latency due to data waiting for batch accumulation before processing	Low-latency processing, minimizing the time between data generation and processing
Event-Driven	Typically, not event-driven; processing occurs at scheduled intervals	Event-driven; actions triggered based on incoming events or patterns
Data Size	Processes fixed-size batches, often with a substantial volume of data	Handles data streams of varying volumes and rates
Scalability	Scalability is limited to batch size and may require additional resources for larger batches	Highly scalable to accommodate dynamic data sources and fluctuations in data rates
Immediate Insights	Results are available after batch processing is complete	Provides immediate insights and enables real-time decision-making
Use Cases	Data warehousing, ETL (Extract, Transform, Load), historical analytics	Real-time monitoring, fraud detection, recommendation engines, IoT applications
Examples	Monthly financial reports, daily website analytics, quarterly sales summaries	Social media updates, sensor data, financial transactions, log streams

## Module-

8. Name a common technique for sampling in stream processing.

## ► 4.5 SAMPLING TECHNIQUES FOR EFFICIENT STREAM PROCESSING

**UQ.** Describe any two sampling techniques for big data with the help of examples.

(MU - May 16, 10 Marks)

### ➤ 4.5.1 Sliding Window

- This is the simplest and most straightforward method. A first-in, first-out (FIFO) queue with size  $n$  and a skip / sub-sampling factor  $k \geq 1$  is maintained.

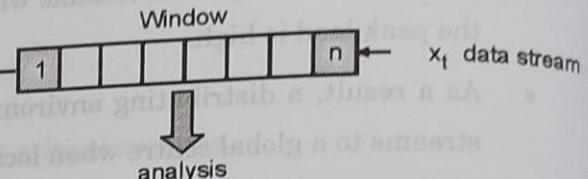


Fig. 4.5.1 : Sliding Window

- In addition to that, a stride factor  $s \geq 1$  describes by how many time-steps the window is shifted before analyzing it.

#### ➤ Advantage

- Simple to implement.
- Deterministic — reservoir can be filled very fast from the beginning.

#### ➤ Drawbacks

The time history represented by the reservoir  $R$  is short; long-term concept drifts cannot be detected easily — outliers can create noisy analyses.

### ➤ 4.5.2 Unbiased Reservoir Sampling

- A reservoir  $R$  is maintained such that at time  $t > n$  the probability of accepting point  $x(t)$  in the reservoir is equal to  $n/t$ .
- The algorithm [1] is as follows:
  - Fill the reservoir  $R$  with the first  $n$  points of the stream.
  - At time  $t > n$  replace a randomly chosen (equal probability) entry in the reservoir  $R$  with acceptance probability  $n/t$ .
- This leads to a reservoir  $R(t)$  such that each point  $x(1) \dots x(t)$  is contained in  $R(t)$  with equal property  $n/t$ .

#### ➤ Advantages

- The reservoir contains data points from all history of the stream with equal probability.

**Module-**

9. Why might Bloom Filters produce false positives?

CHATGPT:

Bloom Filters may produce false positives due to the nature of their probabilistic approach to representing set membership. A Bloom Filter is a space-efficient data structure used to test whether a given element is a member of a set. However, it comes with the trade-off of allowing for false positives. Here's why false positives can occur in Bloom Filters:

**1. Hash Collisions:**

- Bloom Filters use multiple hash functions to map an element to multiple positions in the filter's array. The hash functions distribute elements across the array positions. However, due to the finite size of the array and the potential for different elements to hash to the same positions (collisions), false positives may occur.

**2. Overlapping Hash Positions:**

- Different elements may produce the same set of positions in the array, creating overlapping patterns. When checking for set membership, if a query element hashes to positions that have been set by other elements in the filter, it may incorrectly appear as if it is a member of the set.

**3. Bit Flipping by Other Elements:**

- As elements are added to the Bloom Filter, they set bits in the array to 1. When a new element is added, it may hash to positions already set by other elements. This can cause bits to be flipped from 0 to 1, potentially affecting the filter's representation of other elements and leading to false positives.

**4. Limited Size and Capacity:**

- Bloom Filters have a fixed-size array, which means that as more elements are added, the probability of hash collisions increases. If the filter becomes saturated with elements, it becomes more likely to produce false positives as the available positions in the array are more likely to be set.

**5. Optimizations for Space Efficiency:**

- Bloom Filters are designed to be space-efficient by using a relatively small array of bits. This optimization increases the likelihood of false positives as more elements are added to the filter. The trade-off for space efficiency is an increased chance of collisions.

10. How bloom filter is useful for big data analytics? Explain with one example.

### Bloom Filter with Analysis

- A simple space-efficient data structure introduced by Burton Howard Bloom in 1970.
- The filter matches the membership of an element in a dataset.
- The filter is basically a bit vector of length  $m$  that represent a set  $S = \{x_1, x_2, \dots, x_m\}$  of  $m$  elements,
- Initially all bits 0. Then, define  $k$  independent hash functions,  $h_1, h_2, \dots, h_k$ .

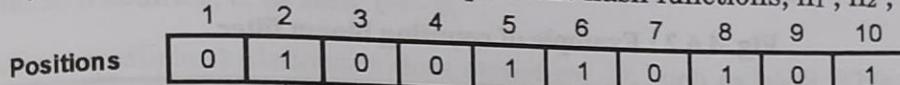
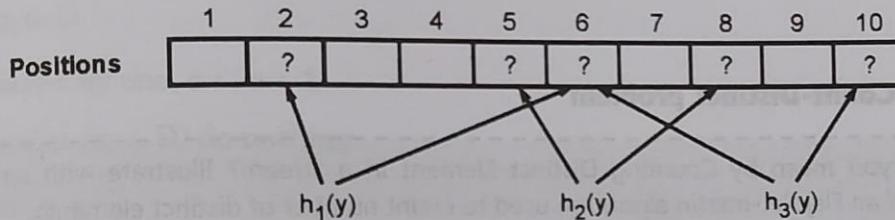
(a) Inserting an element  $x$  in bit vector (filter) of length  $m = 10$ (b) Finding element  $y$  is the Filter(a)Inserting an element  $x$  in bit vector(filter) of length  $m = 10$ ,(b) finding an element  $y$  in an example of Bloom filter

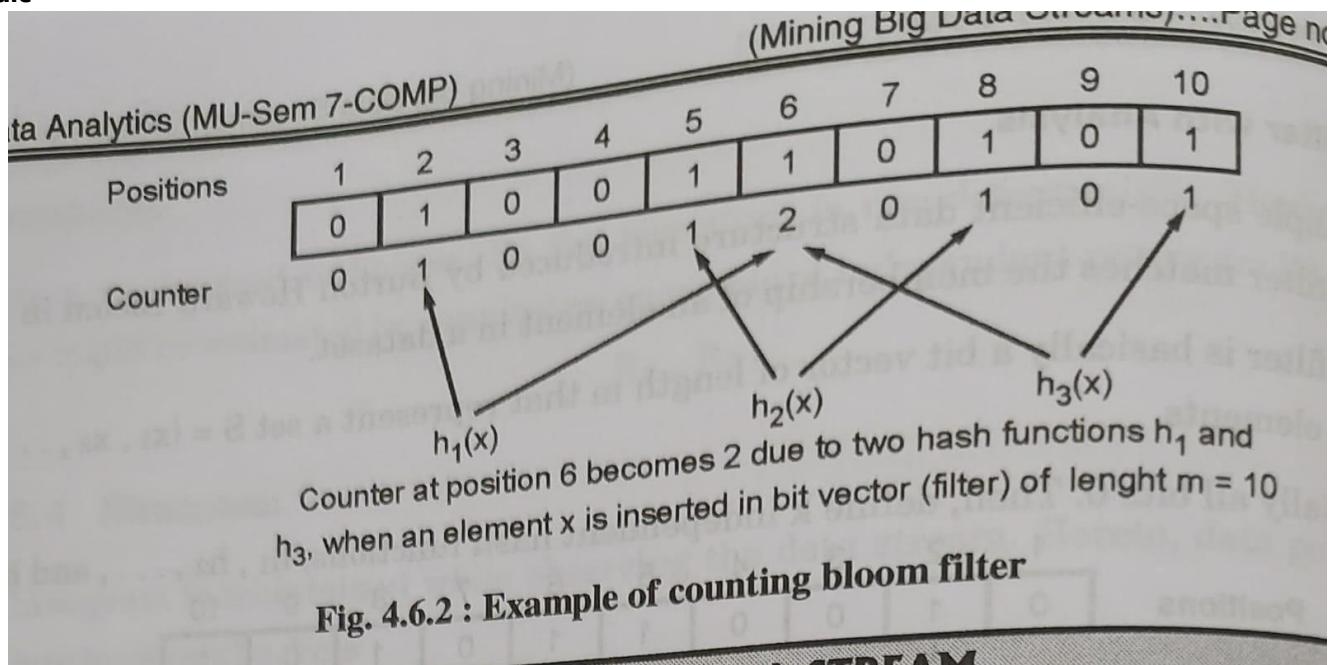
Fig. 4.6.1

- Each of which maps (hashes) some element  $x$  in set  $S$  to one of the  $m$  array positions with a uniform random distribution.
- Number  $k$  is constant, and much smaller than  $m$ . That is, for each element  $x \in S$ , the bits  $h_i(x)$  are set to 1 for  $1 \leq i \leq k$ . [ $\in$  is symbol in set theory for 'contained in'.]

### Counting Bloom Filter: A Variant of Bloom Filter

- It maintains a counter for each bit in the Bloom filter. The counters corresponding to the  $k$  hash values increment or decrement, whenever an element in the filter is added or deleted, respectively.
- As soon as a counter changes from 0 to 1, the corresponding bit in the bit vector is set to 1. When a counter changes from 1 to 0, the corresponding bit in the bit vector is set to 0. The counter basically maintains the number of elements that hashed

## Module-



11. Justify "Bloom Filter has false positives but no false negatives".

The statement "Bloom Filter has false positives but no false negatives" reflects a fundamental characteristic of Bloom Filters, which are probabilistic data structures designed for efficient membership tests. Let's justify this statement:

### Probabilistic Nature:

Bloom Filters use a probabilistic approach to determine set membership. Instead of providing definite answers, they offer a probability of membership. This probabilistic nature arises from the use of hash functions to map elements to positions in the filter's array.

### Hash Functions and Bit Array:

Bloom Filters employ multiple hash functions to generate positions in a bit array. When adding an element to the filter, these hash functions set corresponding bits to 1 in the array. The presence of a 1 in a bit position indicates that an element might be in the set, but it doesn't guarantee it.

### False Positives:

The probability of a false positive in a Bloom Filter increases as more elements are added. This is because different elements may hash to the same positions, resulting in overlapping patterns of set bits. Consequently, when checking for membership, an element may erroneously be considered part of the set (false positive) due to collisions.

### No False Negatives:

Bloom Filters, by design, do not produce false negatives. If the filter indicates that an element is not in the set, it is definitive. This property is a consequence of the non-invertible nature of the hash functions and bit-setting operations. Once a bit is set to 1, it cannot be uniquely traced back to a single element, preventing false negatives.

### Efficiency and Space Optimization:

Bloom Filters are chosen for scenarios where space efficiency is critical. By using a relatively small bit array and a limited number of hash functions, they provide a compact representation of set membership information. This optimization, however, introduces the trade-off of potential false positives.

**Module-**

12. What is the Count Distinct Problem in stream processing?

## 4.7 COUNTING DISTINCT ELEMENTS IN A STREAM

It relates to finding the number of dissimilar elements in a data stream. The stream of data contains repeated elements. This is a well-known problem in networking and databases.

### 4.7.1 The Count-Distinct problem

**UQ.** What do you mean by Counting Distinct Element in a stream? Illustrate with an example working of an Flajolet-martin algorithm used to count number of distinct elements.

(MU - Dec. 19, 10 Marks)

- The count-distinct problem[1] (also known in applied mathematics as the cardinality estimation problem) is the problem of finding the number of distinct elements in a data stream with repeated elements.
- This is a well-known problem with numerous applications.
- The elements might represent IP addresses of packets passing through a router, unique visitors to a web site, elements in a large database, motifs in a DNA sequence, or elements of RFID/sensor networks.

**Formal definition****Instance**

A stream of elements  $x_1, x_2, \dots, x_n$  with repetitions, and an integer m. Let n be the number of distinct elements, namely  $n = \{x_1, x_2, \dots, x_n\}$  and let these elements by  $\{e_1, e_2, \dots, e_n\}$ .

**Objective**

Find an estimate n of n using only m storage units, where  $m \ll n$



An example of an instance for the cardinality estimation problem is the stream  $a, b, a, c, d, b, d$ . For this instance,

$$n = |\{a, b, c, d\}| = 4$$

#### Naive solution [edit]

The naive solution to the problem is as follows :

- Initialize a counter,  $c$ , to zero,  $c \leftarrow 0$ ,
- Initialize an efficient dictionary data structure,  $D$ , such as hash table or search tree in which insertion and membership can be performed quickly.

For each element  $x_i$ , a membership query is issued.

If  $x_i$  is not a member of  $D$  ( $x_i \notin D$ )

Add  $x_i$  to  $D$

Increase  $c$  by one,  $c \leftarrow c + 1$

Otherwise ( $x_i \in D$ ) do nothing.

Output  $n = c$ .

13. How does the DGIM Algorithm handle query answering?

#### 4.9.2 Query Answering in the DGIM Algorithm

- Suppose we are asked how many 1's there are in the last  $k$  bits of the window, for some  $1 \leq k \leq N$ . Find the bucket  $b$  with the earliest timestamp that includes at least some of the  $k$  most recent bits. Estimate the number of 1's to be the sum of the sizes of all the buckets to the right (more recent) than bucket  $b$ , plus half the size of  $b$  itself
- **Example :** Suppose the stream is that of Fig. 4.9.1, and  $k = 10$ . Then the query asks for the number of 1's in the ten rightmost bits, which happen to be 0110010110. Let the current timestamp (time of the rightmost bit) be  $t$ . Then the two buckets with one 1, having timestamps  $t - 1$  and  $t - 2$  are completely included in the answer.

The bucket of size 2, with timestamp  $t - 4$ , is also completely included. However, the rightmost bucket of size 4, with timestamp  $t - 8$  is only partly included. We know it is the last bucket to contribute to the answer, because the next bucket to its left has timestamp less than  $t - 9$  and thus is completely out of the window.

On the other hand, we know the buckets to its right are completely inside the range of the query because of the existence of a bucket to their left with timestamp  $t - 9$  or greater.

- Our estimate of the number of 1's in the last ten positions is thus 6. This number is the two buckets of size 1, the bucket of size 2, and half the bucket of size 4 that is partially within range. Of course the correct answer is 5. 2 Suppose the above estimate of the answer to a query involves a bucket  $b$  of size  $2^j$  that is partially within the range of the query.
- Let us consider how far from the correct answer  $c$  our estimate could be. There are two cases: the estimate could be larger or smaller than  $c$ .
- **Case 1:** The estimate is less than  $c$ . In the worst case, all the 1's of  $b$  are actually within the range of the query, so the estimate misses half bucket  $b$ , or  $2^{j-1}$  1's. But in this case,  $c$  is at least  $2^j$ ; in fact it is at least  $2^{j+1} - 1$ , since there is at least one bucket of each of the sizes  $2^j - 1, 2^j - 2, \dots, 1$ . We conclude that our estimate is at least 50% of  $c$ .
- **Case 2:** The estimate is greater than  $c$ . In the worst case, only the rightmost bit of bucket  $b$  is within range, and there is only one bucket of each of the sizes smaller than  $b$ . Then  $c = 1 + 2^j - 1 + 2^j - 2 + \dots + 1 = 2^j$  and the estimate we give is  $2^{j-1} + 2^{j-1} + 2^{j-2} + \dots + 1 = 2^j + 2^{j-1} - 1$ . We see that the estimate is no more than 50% greater than  $c$ .

14. Explain the concept of decaying windows in stream processing.

### 4.9.3 Decaying Windows

**GQ.** Explain Decaying windows method.

It is useful in applications which need identification of most common elements. Decaying window concept assigns more weight to recent elements. The technique computes a smooth aggregation of all the 1's ever seen in the stream, with decaying weights. When an element further appears in the stream, less weight is given. The effect of exponentially decaying weights is to spread out the weights of the stream elements as far back in time as the stream flows.

**The Problem of Most-Common Elements**

- Suppose we have a stream whose elements are the movie tickets purchased all over the world, with the name of the movie as part of the element. We want to keep a summary of the stream that is the most popular movies "currently." While the notion of "currently" is imprecise, intuitively, we want to discount the popularity of a movie like Star Wars-Episode 4, which sold many tickets, but most of these were sold decades ago.
- On the other hand, a movie that sold  $n$  tickets in each of the last 10 weeks is probably more popular than a movie that sold  $2n$  tickets last week but nothing in previous weeks. One solution would be to imagine a bit stream for each movie.
- The  $i^{\text{th}}$  bit has value 1 if the  $i^{\text{th}}$  ticket is for that movie, and 0 otherwise. Pick a window size  $N$ , which is the number of most recent tickets that would be considered in evaluating popularity. Then, use the method of Section 4.6 to estimate the number of tickets for each movie, and rank movies by their estimated counts.
- This technique might work for movies, because there are only thousands of movies, but it would fail if we were instead recording the popularity of items sold at Amazon, or the rate at which different Twitter-users tweet, because there are too many Amazon products and too many tweeters. Further, it only offers approximate answers

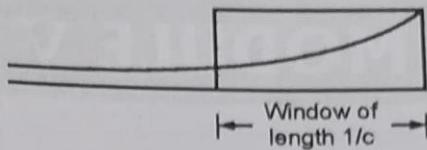
**Definition of the Decaying Window**

- An alternative approach is to redefine the question so that we are not asking for a count of 1's in a window. Rather, let us compute a smooth aggregation of all the 1's ever seen in the stream, with decaying weights, so the further back in the stream, the less weight is given.
- Formally, let a stream currently consist of the elements  $a_1, a_2, \dots, a_t$ , where  $a_1$  is the first element to arrive and  $a_t$  is the current element. Let  $c$  be a small constant, such as  $10^{-6}$  or  $10^{-9}$ . Define the exponentially decaying window for this stream to be the sum

$$\sum_{i=0}^{t-1} a_{t-i} (1 - c)^i$$

- The effect of this definition is to spread out the weights of the stream elements as far back in time as the stream goes.

- In contrast, a fixed window with the same sum of the weights,  $1/c$ , would put equal weight 1 on each of the most recent  $1/c$  elements to arrive and weight 0 on all previous elements. The distinction is suggested by Fig. 4.9.2.



**Fig. 4.9.2 : A decaying window and a fixed-length window of equal weight**

It is much easier to adjust the sum in an exponentially decaying window than in a sliding window of fixed length. In the sliding window, we have to worry about the element that falls out of the window each time a new element arrives. That forces us to keep the exact elements along with the sum, or to use an approximation scheme such as DGIM. However, when a new element  $a_{t+1}$  arrives at the stream input, all we need to do is:

1. Multiply the current sum by  $1 - c$ .
2. Add  $a_{t+1}$ .

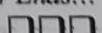
The reason this method works is that each of the previous elements has now moved one position further from the current element, so its weight is multiplied by  $1 - c$ . Further, the weight on the current element is  $(1 - c)^0 = 1$ , so adding  $a_{t+1}$  is the correct way to include the new element's contribution.

#### Finding the Most Popular Elements

- The problem of finding the most popular movies in a stream of ticket sales. We shall use an exponentially decaying window with a constant  $c$ , which you might think of as  $10^{-9}$ . That is, we approximate a sliding window holding the last one billion ticket sales.
- For each movie, we imagine a separate stream with a 1 each time a ticket for that movie appears in the stream, and a 0 each time a ticket for some other movie arrives. The decaying sum of the 1's measures the current popularity of the movie.
- We imagine that the number of possible movies in the stream is huge, so we do not want to record values for the unpopular movies. Therefore, we establish a threshold, say  $1/2$ , so that if the popularity score for a movie goes below this number, its score is dropped from the counting. For reasons that will become obvious, the threshold must be less than 1, although it can be any number less than 1.

When a new ticket arrives on the stream, do the following:

1. For each movie whose score we are currently maintaining, multiply its score by  $(1 - c)$ .
2. Suppose the new ticket is for movie M. If there is currently a score for M, add 1 to that score. If there is no score for M, create one and initialize it to 1.
3. If any score is below the threshold  $1/2$ , drop that score.



## Module-

15. What is Sliding window sampling technique for big data.

## 4.5 SAMPLING TECHNIQUES FOR EFFICIENT STREAM PROCESSING

UQ. Describe any two sampling techniques for big data with the help of examples.

(MU - May 16, 10 Marks)

### 4.5.1 Sliding Window

- This is the simplest and most straightforward method. A first-in, first-out (FIFO) queue with size  $n$  and a skip / sub-sampling factor  $k \geq 1$  is maintained.
- In addition to that, a stride factor  $s \geq 1$  describes by how many time-steps the window is shifted before analyzing it.

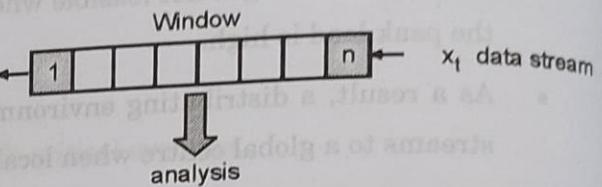


Fig. 4.5.1 : Sliding Window

#### Advantage

- Simple to implement.
- Deterministic — reservoir can be filled very fast from the beginning.

#### Drawbacks

The time history represented by the reservoir  $R$  is short; long-term concept drifts cannot be detected easily — outliers can create noisy analyses.

16. Consider size of Bloom filter  $m = 10$  and two hash functions:  $h_1(x) = x \bmod 10$  and  $h_2(x) = (3x+1) \bmod 10$ . Insert the element 5, 7, 12 and 13. Check, are elements 11, 15 and 8 presents? Which elements return false positive result? How we can reduce it.

CHATGPT:

Let's walk through the steps of inserting elements into a Bloom filter with size  $m=10$  using two hash functions:  $h_1(x)=x \bmod 10$  and  $h_2(x)=(3x+1) \bmod 10$ . We'll then check whether certain elements are present and identify any false positives. Finally, we'll discuss how to reduce false positives.

#### Step 1: Insert Elements (5, 7, 12, 13)

Using the hash functions, we calculate the positions for each element in the Bloom filter:

1. Insert 5:

- $h_1(5)=5 \bmod 10=5$
- $h_2(5)=(3 \times 5+1) \bmod 10=6$
- Bloom filter: 0000000000
- Set positions 5 and 6 to 1: 0000011000

## Module-

### 2. Insert 7:

- $h_1(7) = 7 \bmod 10 = 7$   $h_1(7) = 7 \bmod 10 = 7$
- $h_2(7) = (3 \times 7 + 1) \bmod 10 = 2$   $h_2(7) = (3 \times 7 + 1) \bmod 10 = 2$
- Bloom filter: 00000110000000011000
- Set positions 7 and 2 to 1: 01000110000100011000

### 3. Insert 12:

- $h_1(12) = 12 \bmod 10 = 2$   $h_1(12) = 12 \bmod 10 = 2$
- $h_2(12) = (3 \times 12 + 1) \bmod 10 = 7$   $h_2(12) = (3 \times 12 + 1) \bmod 10 = 7$
- Bloom filter: 01000110000100011000
- Set positions 2 and 7 to 1: 01000111000100011100

### 4. Insert 13:

- $h_1(13) = 13 \bmod 10 = 3$   $h_1(13) = 13 \bmod 10 = 3$
- $h_2(13) = (3 \times 13 + 1) \bmod 10 = 0$   $h_2(13) = (3 \times 13 + 1) \bmod 10 = 0$
- Bloom filter: 01000111000100011100
- Set positions 3 and 0 to 1: 110001110001100011100

## Step 2: Check for Elements (11, 15, 8)

Now, let's check whether elements 11, 15, and 8 are present in the Bloom filter:

### 1. Check 11:

- $h_1(11) = 11 \bmod 10 = 1$   $h_1(11) = 11 \bmod 10 = 1$
- $h_2(11) = (3 \times 11 + 1) \bmod 10 = 9$   $h_2(11) = (3 \times 11 + 1) \bmod 10 = 9$
- Both positions 1 and 9 are 0: **Not present**

### 2. Check 15:

- $h_1(15) = 15 \bmod 10 = 5$   $h_1(15) = 15 \bmod 10 = 5$
- $h_2(15) = (3 \times 15 + 1) \bmod 10 = 6$   $h_2(15) = (3 \times 15 + 1) \bmod 10 = 6$
- Both positions 5 and 6 are 1: **Possibly present (false positive)**

### 3. Check 8:

- $h_1(8) = 8 \bmod 10 = 8$   $h_1(8) = 8 \bmod 10 = 8$
- $h_2(8) = (3 \times 8 + 1) \bmod 10 = 5$   $h_2(8) = (3 \times 8 + 1) \bmod 10 = 5$
- Both positions 8 and 5 are 1: **Possibly present (false positive)**

### Identifying False Positives:

- Elements 15 and 8 are false positives because both hash functions indicate their presence, but they were not inserted.

### Reducing False Positives:

- To reduce false positives, you can consider increasing the size of the Bloom filter ( $m$ ) and/or using additional hash functions. A larger filter and more hash functions generally decrease the probability of collisions and false positives but come at the cost of increased memory usage and computational overhead.

In this case, with a small filter size ( $m = 10$ ) and only two hash functions, false positives are more likely. Increasing the filter size or adding more hash functions can help mitigate this issue.

## **Module-**

17. What is Flajolet-Martin (FM) algorithm. Using FM algorithm, compute number of distinct elements in the set  $S = \{1, 3, 2, 1, 2, 3, 4, 3, 1, 2, 3, 1\}$  with hash function  $h(x)=(6x+1) \bmod 5$ .

CHATGPT:

The Flajolet-Martin algorithm is a probabilistic algorithm used for estimating the number of distinct elements in a set. It's commonly employed in the context of streaming data, where it provides a memory-efficient way to estimate cardinality without storing the entire set. The algorithm uses hash functions to convert elements into binary representations and then analyzes the rightmost zeros in these binary representations.

Set S: {1, 3, 2, 1, 2, 3, 4, 3, 1, 2, 3, 1}

Hash Function:  $h(x) = (6x + 1) \bmod 5$

Binary Representations:

1: 110 (rightmost zero count = 1)

3: 101 (rightmost zero count = 0)

2: 010 (rightmost zero count = 1)

1: 110 (rightmost zero count = 1)

2: 010 (rightmost zero count = 1)

3: 101 (rightmost zero count = 0)

4: 011 (rightmost zero count = 0)

3: 101 (rightmost zero count = 0)

1: 110 (rightmost zero count = 1)

2: 010 (rightmost zero count = 1)

3: 101 (rightmost zero count = 0)

1: 110 (rightmost zero count = 1)

$$\text{Average Count of Rightmost Zeros} = (1 + 0 + 1 + 1 + 1 + 0 + 0 + 0 + 1 + 1 + 0 + 1) / 12 \approx 0.5833$$

$$\text{Estimated Cardinality} = 2^{(0.5833)} \approx 1.512$$

18. In the context of network security, you need to identify malicious IP addresses from a stream of incoming data. Explain how a Bloom Filter can assist in quickly identifying whether an IP address is part of a known blacklist. Discuss the implications of false positives and false negatives in this security application.

CHATGPT:

## Module-

In the context of network security, using a Bloom Filter to identify malicious IP addresses from a stream of incoming data is a common and efficient approach. Here's how a Bloom Filter can assist and the implications of false positives and false negatives in this security application:

### How a Bloom Filter Assists:

#### 1. Space-Efficient Storage:

- Bloom Filters provide a space-efficient representation of a set of elements. In the case of a blacklist of malicious IP addresses, storing the entire list in memory can be resource-intensive. A Bloom Filter allows for compact storage and efficient use of memory.

#### 2. Fast Membership Testing:

- Bloom Filters enable fast membership tests. Given an incoming IP address, the Bloom Filter can quickly determine whether it is likely to be in the blacklist or not. If the filter indicates the presence of the IP address, further, more resource-intensive checks can be performed.

#### 3. Reduced False Negatives:

- While Bloom Filters may produce false positives, they do not produce false negatives. If the filter indicates that an IP address is not in the blacklist, it is definitive. This property ensures that no malicious IP address is mistakenly considered safe.

#### 4. Scalability:

- Bloom Filters are scalable, making them suitable for large and dynamically changing blacklists. As the blacklist grows or changes, the Bloom Filter can adapt without a significant increase in memory requirements.

### Implications of False Positives and False Negatives:

#### 1. False Positives:

- *Definition:* A false positive occurs when the Bloom Filter indicates that an IP address is in the blacklist, but it is not.
- *Implications:* False positives may lead to legitimate traffic being mistakenly flagged as malicious. This can result in the blocking of benign IP addresses, causing service disruptions and affecting user experience.

#### 2. False Negatives:

- *Definition:* In the context of Bloom Filters, false negatives do not occur. If the filter indicates that an IP address is not in the blacklist, it is guaranteed to be correct.
- *Implications:* The absence of false negatives ensures that known malicious IP addresses are not overlooked. If an IP address is in the blacklist, the Bloom Filter will correctly identify it.

### Mitigating False Positives:

#### 1. Adjusting Filter Parameters:

- Modifying the parameters of the Bloom Filter, such as the size of the filter and the number of hash functions, can impact the rate of false positives. Tuning these parameters to balance false positives and memory usage is crucial.

#### 2. Combining with Additional Checks:

## **Module-**

- While the Bloom Filter provides a quick preliminary check, it should be used in conjunction with more robust methods for confirming the malicious nature of an IP address. Combining it with other techniques, such as deep packet inspection or behavioral analysis, can help reduce false positives.

### **3. Regularly Updating Blacklist:**

- Keeping the blacklist up-to-date is important. Regularly updating the Bloom Filter with the latest information ensures that it accurately reflects the current state of known malicious IP addresses.

19. You are working on a social media analytics platform that needs to estimate the number of unique hashtags in a real-time stream. How could the Flajolet-Martin Algorithm be applied in this scenario? Discuss the considerations for combining estimates from multiple streams to improve accuracy.

CHATGPT:

#### **Applying Flajolet-Martin Algorithm in Social Media Analytics:**

*Estimating Unique Hashtags in Real-Time:*

##### **1. Hash Function Selection:**

- Choose a hash function that maps hashtags to integers. For example,  $h(\text{tag}) = \text{hash code of tag}$  and  $h(x) = \text{hash code of } x$ .

##### **2. Binary Representation:**

- Convert each hash value into its binary representation. Count the rightmost consecutive zeros.

##### **3. Average Count Calculation:**

- Calculate the average count of rightmost zeros across all hash values.

##### **4. Estimate Unique Hashtags:**

- Use the formula  $\sqrt{2} \times \text{average count of rightmost zeros}$  to estimate the number of unique hashtags in the real-time stream.

#### **Considerations for Combining Estimates from Multiple Streams:**

*Improving Accuracy Through Aggregation:*

##### **1. Independent Streams:**

- Ensure that the streams are independent to avoid correlated errors. If multiple streams share common elements, the estimates may be less accurate.

##### **2. Weighted Averaging:**

- Assign weights to each stream based on its reliability or importance. Use a weighted average of the estimates to give more influence to trustworthy streams.

##### **3. Consistency Checks:**

- Implement consistency checks to identify and handle streams with significantly deviant estimates. This helps filter out unreliable sources.

## **Module-**

### **4. Dynamic Weight Adjustments:**

- Dynamically adjust weights based on the historical performance of each stream. This adaptive approach accounts for changing reliability over time.

### **5. Thresholds for Confidence:**

- Set confidence thresholds to identify situations where estimates from multiple streams significantly differ. In such cases, additional verification or human intervention may be necessary.

### **6. Error Bounds:**

- Establish error bounds for each stream to quantify the expected level of inaccuracy. Consider the overlap of error bounds when combining estimates.

### **7. Continuous Monitoring and Calibration:**

- Continuously monitor the performance of each stream and calibrate the system as needed. Regularly update weights and thresholds based on real-time feedback.

#### *Benefits of Combining Estimates:*

- **Increased Robustness:** Combining estimates from multiple streams reduces the impact of errors from individual sources.
- **Enhanced Accuracy:** Aggregating information from diverse streams provides a more accurate and reliable estimation of unique hashtags.
- **Adaptability:** Dynamic adjustments enable the system to adapt to changing conditions and maintain accuracy over time.

20. Imagine you are developing a system for monitoring power consumption in a smart city. Explain how the Datar-Gionis-Indyk-Motwani (DGIM) Algorithm could be applied to estimate the number of occurrences of power spikes within a sliding window.

#### **Applying DGIM Algorithm for Power Spike Monitoring:**

##### **1. Overview:**

- The Datar-Gionis-Indyk-Motwani (DGIM) Algorithm is a probabilistic algorithm used for estimating the number of occurrences of events in a sliding window. In the context of monitoring power consumption in a smart city, the DGIM Algorithm can be applied to estimate the number of power spikes within a defined sliding window.

##### **2. Steps for Application:**

###### **a. Data Representation:**

- Represent the stream of power consumption events as a sequence of timestamped occurrences, where each occurrence is a power spike event.

###### **b. Sliding Window:**

- Define a sliding window that spans a specific time duration. This window moves over the stream, and its size determines the time range for which the estimation is made.

###### **c. Bucket Structure:**

## **Module-**

- Divide the sliding window into buckets, where each bucket represents a time interval. The bucket sizes follow a binary progression (1, 2, 4, 8, ...), forming a logarithmic structure.

### **d. Counter in Buckets:**

- Assign a counter to each bucket, counting the number of occurrences within that specific time interval.

### **e. Bucket Consolidation:**

- Periodically consolidate buckets to maintain a logarithmic structure. Merge pairs of buckets with the same counter size, discarding the oldest bucket.

### **f. Estimation:**

- Use the counters in the remaining buckets to estimate the total number of power spike occurrences within the sliding window. The algorithm provides an approximate count with a controlled error rate.

## **3. Considerations for Power Spike Monitoring:**

### **a. Error Rate and Accuracy:**

- Adjust the parameters of the DGIM Algorithm to control the error rate and accuracy of the estimation. Balancing these factors is essential for reliable power spike monitoring.

### **b. Time Granularity:**

- Choose an appropriate time granularity for the buckets to ensure that the algorithm captures and estimates power spikes accurately within the desired time resolution.

### **c. Window Size:**

- Set the size of the sliding window based on the monitoring requirements. A larger window accommodates a longer time span for estimation but may require more memory.

### **d. Dynamic Adjustment:**

- Consider dynamically adjusting parameters based on the characteristics of the power consumption data. The system may adapt to varying patterns and intensities of power spikes.

## **4. Benefits of DGIM for Power Spike Monitoring:**

### **a. Memory Efficiency:**

- DGIM Algorithm provides an efficient way to estimate the number of occurrences within a sliding window using limited memory.

### **b. Real-Time Estimation:**

- The algorithm allows for real-time estimation of power spikes, making it suitable for monitoring and responding to dynamic changes in power consumption.

### **c. Scalability:**

- DGIM is scalable and capable of handling large streams of data, making it applicable to smart city-scale power consumption monitoring.

## **5. Use Case:**

- For example, in a smart city context, the DGIM Algorithm could be applied to estimate the number of power spikes in a sliding window of the past hour. This estimation can assist in identifying unusual patterns and optimizing the city's power distribution and management systems.

**Module-****5: Real-Time Big Data Models**

5.1 A Model for Recommendation Systems, Content-Based Recommendations, Collaborative Filtering

5.2 Case Study: Product Recommendation

5.3 Social Networks as Graphs, Clustering of Social-Network Graphs, Direct Discovery of Communities in a social graph

**Practice Questions**

1. What is social Network?

TECHKNOWLEDGE:

Social network idea came into theory and research in 1980s by Ferdinand Tonnis and Emile Durkheim. Social network is bind with domain like social links, social group.

Social network deals with large-scale data. After analyzing large data a huge set of information can be achieved.

LinkedIn, Facebook are vast widely used and very popular examples for social network. As we can find friends over the network with 1st, 2nd, 3rd connection or mutual friends (i.e. friends of friend) in LinkedIn and Facebook respectively.

Google+ is one of social network which gives link nodes in groups categories like Friends, Family, Acquaintances following Featured on Google+ etc.

Social network is huge platform to analyze data and obtain information. Further will see efficient algorithm to discover different graphs properties.

2. In a social network graph, what do nodes and edges represent?

TECHKNOWLEDGE:

In general a graph is collection of set of edges (e) and set of vertices (V). If there is an edge exists between any two nodes of graph then that node relates with each other.

Graphs are categories by many parameters like ordered pairs of nodes, unordered pairs of nodes.

Some edge has direction, weight. Relationship among graph is explained with help of an adjacency matrix. Small network can be easily managed to construct a graph, it is quite impossible with huge/wide network. Summary statistics and performance metrics are useful for design of graph for a large network.

Network and graphs can be elaborate with the help of few parameters like diameter i.e. largest distance between any two nodes, centrality degree distribution.

Social website like Facebook uses undirected social graph for friends while directed graph used in social website like Twitter, Google+ (plus). Twitter gives connection like 1st , 2nd , 3rd and Google classify linked connection in friends, family, Acquaintances, Following etc.

3. Discuss the concept of clustering in the context of social-network graphs.

**Other Examples of Social Graphs**

- Many other phenomena give rise to graphs that look something like social graphs, especially exhibiting locality.
- Examples include: information networks(documents, web graphs, patents), infrastructure networks (roads, planes, water pipes, power grids), biological networks (genes, proteins, food-webs of animal seating each other), as well as other types, like product co-purchasing networks(e.g., Group on).

**5.3.2 Clustering of Social-Network Graphs**

**GQ.** Explain clustering of Social-Network Graphs with example.

(10 Marks)

An important aspect of social networks is that they contain communities of entities that are connected by many edges. These typically correspond to groups of friends at school or groups of researchers interested in the same topic, for example.

**Distance Measures for Social-Network Graphs**

- When the edges of the graph have labels, these labels might be usable as a distance measure, depending on what they represented. But when the edges are unlabeled, as in a “friends” graph, there is not much we can do to define a suitable distance.
- Our first instinct is to assume that nodes are close if they have an edge between them and distant if not. Thus, we could say that the distance  $d(x, y)$  is 0 if there is an edge  $(x, y)$  and 1 if there is no such edge. We could use any other two values, such as 1 and  $\infty$ , as long as the distance is closer when there is an edge.
- Neither of these two-valued “distance measures” – 0 and 1 or 1 and  $\infty$  – is a true distance measure.

**Applying Standard Clustering Methods**

- In particular, suppose we use as the inter cluster distance the minimum distance between nodes of the two clusters. Hierarchical clustering of a social-network graph starts by combining some two nodes that are connected by an edge.
- Successively, edges that are not between two nodes of the same cluster would be chosen randomly to combine the clusters to which their two nodes belong. The choices would be random, because all distances represented by an edge are the same.

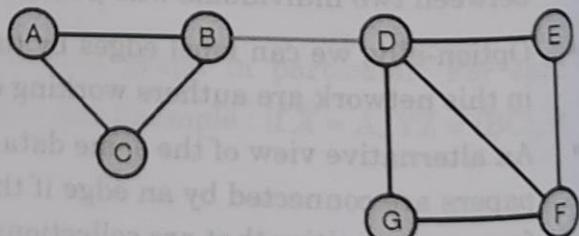


Fig. 5.3.1

**Point-assignment approach**

- in k-means randomly picked nodes might be in the same cluster
- randomly chosen and another as far away as possible doesn't do much better (i.e. E and G)
- even choosing B and F – problem with assignment of D

**Betweenness**

This method based on finding the edges that are **least likely to be inside a community**. Betweenness of an edge  $(a, b)$  is the number of pairs of nodes  $x$  and  $y$  such that the edge  $(a, b)$  lies on the shortest path between  $x$  and  $y$ . High value suggests that  $(a, b)$  runs between two different communities –  $a$  and  $b$  do not belong to the same community

4. How do content-based filtering and collaborative filtering differ when employed in recommendation systems?

CHATGPT (ANY 5 POINTS):

**Content-Based Filtering vs. Collaborative Filtering in Recommendation Systems:****1. Definition:**

- **Content-Based Filtering:**

- Recommends items based on the attributes and characteristics of the items themselves, as well as the user's preferences. It relies on a profile of the user's interests and the features of items.

- **Collaborative Filtering:**

- Recommends items based on the preferences and behavior of other users. It identifies patterns by considering the actions and preferences of a community of users.

**2. Data Requirement:**

- **Content-Based Filtering:**

- Requires detailed information about the attributes and features of items. It builds a user profile based on the content and characteristics of items.

- **Collaborative Filtering:**

- Requires user-item interaction data, such as user ratings or purchase history. It identifies user preferences based on the behavior and feedback of a user community.

**3. User Profile:**

- **Content-Based Filtering:**

- Builds a user profile by analyzing the features and content of items that the user has liked or interacted with. The focus is on the user's explicit preferences.

- **Collaborative Filtering:**

## **Module-**

- Utilizes the preferences and behaviors of a group of users to recommend items to an individual user. It relies on the implicit or explicit feedback of the user community.

### **4. Item Representation:**

- **Content-Based Filtering:**
  - Represents items using their features and attributes. It uses a content vector to describe the characteristics of each item.
- **Collaborative Filtering:**
  - Represents items based on user interactions. It may use user-item interaction matrices to capture the preferences and relationships between users and items.

### **5. Dependency on User History:**

- **Content-Based Filtering:**
  - Recommendations are based on the user's individual history and preferences. It is less reliant on the behavior of other users.
- **Collaborative Filtering:**
  - Relies on the collective behavior and preferences of a group of users. It assumes that users with similar tastes in the past will continue to have similar tastes.

### **6. Cold Start Problem:**

- **Content-Based Filtering:**
  - Handles the cold start problem well, as it can make recommendations based on the features of items, even for new users with limited interaction history.
- **Collaborative Filtering:**
  - Faces challenges in the cold start problem, especially for new users with little to no interaction history. It relies heavily on historical user data.

### **7. Diversity of Recommendations:**

- **Content-Based Filtering:**
  - Tends to provide recommendations similar to what the user has liked in the past, potentially limiting diversity.
- **Collaborative Filtering:**
  - Can offer diverse recommendations by identifying patterns and preferences that may not be immediately apparent from the user's history.

### **8. Scalability:**

- **Content-Based Filtering:**
  - Scales well as it primarily depends on the user's own preferences and the features of items.
- **Collaborative Filtering:**
  - May face scalability challenges with a large user base as it requires analyzing the interactions and preferences of all users.

## **Module-**

5. What role does community detection play in enhancing user experience on social media platforms?

CHATGPT:

### **Role of Community Detection in Enhancing User Experience on Social Media Platforms:**

#### **1. Content Personalization:**

- **Role:** Community detection helps identify groups of users with similar interests and interactions.
- **Enhancement:** Enables personalized content recommendations based on the preferences and behaviors of users within specific communities. Users receive content that aligns more closely with their interests.

#### **2. Friendship Recommendations:**

- **Role:** Community detection identifies clusters of users who frequently interact or share common interests.
- **Enhancement:** Facilitates more accurate and relevant friend recommendations. Users are suggested connections within their communities, fostering meaningful interactions and connections.

#### **3. Targeted Advertising:**

- **Role:** Identifying user communities allows for the segmentation of the audience based on shared characteristics.
- **Enhancement:** Advertisers can target specific communities with advertisements that are more likely to resonate with their members. This enhances the relevance of ads and improves the overall advertising experience.

#### **4. Content Discovery:**

- **Role:** Community detection reveals subgroups of users with distinct preferences and trends.
- **Enhancement:** Users can discover new content and trends within their specific communities, leading to a more tailored and satisfying content discovery experience.

#### **5. Filtering and Moderation:**

- **Role:** Detecting communities helps identify potentially harmful or inappropriate content and behaviors.
- **Enhancement:** Enables platform moderators to apply more targeted content filtering and moderation measures within specific communities, ensuring a safer and more enjoyable environment for users.

#### **6. Enhanced Engagement Metrics:**

Role: Understanding community structures helps measure engagement within and across user groups.

Enhancement: Platforms can analyze community-specific engagement metrics, allowing them to tailor features and algorithms to enhance user engagement within different communities.

#### **7. User Empowerment:**

Role: Community detection empowers users by highlighting their belongingness to specific interest groups or communities.

## **Module-**

Enhancement: Users can actively participate in and contribute to communities aligned with their interests, fostering a sense of belonging and enhancing their overall social media experience.

6. In the context of a product recommendation, what factors might be considered in a content-based recommendation system?

CHATGPT:

In a content-based recommendation system, the system suggests items to users based on the characteristics and features of the items themselves, as well as the preferences expressed by the user. The factors considered in a content-based recommendation system are primarily related to the attributes and properties of the items. Here are key factors considered:

### **1. Item Features:**

- **Description:** The textual description of the item, which can be analyzed for keywords, sentiments, and relevance to user preferences.
- **Attributes:** Specific attributes or features of the item, such as size, color, brand, genre, author, or any relevant metadata.

### **2. User Preferences:**

- **User Profile:** The historical preferences and interactions of the user with similar items. This includes the user's past ratings, likes, or purchase history.
- **Explicit Feedback:** Any explicit feedback provided by the user, such as ratings, reviews, or preferences explicitly stated in their profile.

### **3. Text Analysis:**

- **Natural Language Processing (NLP):** Analyzing textual content associated with the item, such as reviews or product descriptions, to understand sentiments, topics, and user preferences.

### **4. Image Analysis:**

- **Visual Features:** Analyzing visual content, such as images or videos associated with the item, to extract visual features that might influence user preferences.

### **5. Metadata:**

- **Genre or Category:** Assigning items to specific genres or categories, which can be used to recommend items similar to those the user has previously shown interest in.
- **Tags and Labels:** Adding tags or labels to items based on their characteristics, facilitating more granular content-based recommendations.

### **6. Collaborative Filtering Incorporation:**

- **Hybrid Approaches:** Combining content-based recommendation with collaborative filtering to leverage both item features and user behavior. This hybrid model can provide more accurate and diverse recommendations.

### **7. Temporal Factors:**

- **Time Sensitivity:** Considering temporal factors, such as the release date of an item, to recommend newer items or account for changing user preferences over time.

### **8. Location-Based Features:**

- **Geographical Relevance:** For certain products or services, considering the geographical location of the user or the item may be relevant in making personalized recommendations.

## Module-

### 9. Popularity and Trends:

- **Popularity Scores:** Incorporating popularity scores or trends associated with items to balance personalized recommendations with what is currently popular among a broader audience.

### 10. Diversity of Recommendations:

- **Diverse Attributes:** Ensuring diversity in recommendations by considering a mix of item attributes to avoid repetitiveness and broaden the user's exploration.

### 11. Domain-Specific Factors:

- **Domain Knowledge:** Incorporating domain-specific knowledge or rules to enhance the relevance of recommendations based on the unique characteristics of the items.

### 12. Scalability:

- **Efficient Feature Representation:** Ensuring that the representation of item features is scalable for a large number of items, allowing the system to handle a diverse and extensive catalog.

7. What challenges might arise in implementing a product recommendation system for a large e-commerce platform?

CHATGPT:

Implementing a product recommendation system for a large e-commerce platform comes with several challenges, considering the scale, diversity of products, and user behaviors. Here are some challenges that may arise:

#### 1. Scalability:

- **Challenge:** Managing and processing large volumes of data, including user interactions, product catalog updates, and recommendations, in real-time for a massive user base.
- **Solution:** Employ scalable distributed computing frameworks and optimize algorithms for efficient processing.

#### 2. Data Sparsity:

- **Challenge:** Sparse user-item interaction data where users may only interact with a small subset of available products, making it challenging to capture user preferences accurately.
- **Solution:** Implement techniques like matrix factorization, implicit feedback handling, or hybrid models to address data sparsity.

#### 3. Diversity of Products:

- **Challenge:** Handling a diverse product catalog with various categories, attributes, and characteristics, which requires capturing nuanced user preferences.
- **Solution:** Use content-based methods, incorporate rich item metadata, and apply feature engineering to represent diverse product attributes.

#### 4. Cold Start Problem:

- **Challenge:** Recommending items for new users or new products with limited interaction history.
- **Solution:** Utilize hybrid recommendation systems, leverage demographic information, and employ content-based approaches to address the cold start problem.

## **Module-**

### **5. Real-time Recommendations:**

- **Challenge:** Providing real-time recommendations while considering the dynamic nature of user behavior and product availability.
- **Solution:** Implement streaming recommendation algorithms and employ caching strategies to reduce latency.

### **6. Privacy Concerns:**

- **Challenge:** Balancing the need for personalized recommendations with user privacy concerns, especially when dealing with sensitive data.
- **Solution:** Implement privacy-preserving recommendation techniques, anonymize data, and ensure compliance with data protection regulations.

### **7. Algorithmic Bias:**

- **Challenge:** Addressing biases in recommendation algorithms that may result in unfair or discriminatory recommendations.
- **Solution:** Regularly audit algorithms for biases, employ diversity-aware training, and incorporate fairness-aware metrics in model evaluation.

## **8. Can collaborative filtering work well when there are new items in the system?**

### **CHATGPT:**

Collaborative filtering methods, especially traditional user-item collaborative filtering, may face challenges when dealing with new items in the system. This challenge is often referred to as the "cold start" problem. Here are some reasons why collaborative filtering may struggle with new items:

#### **1. Lack of User Interactions:**

- Collaborative filtering relies on historical user interactions to identify patterns and make recommendations. When a new item is introduced, there are no or very few historical interactions to use for similarity calculations.

#### **2. Sparsity of User-Item Matrix:**

- The user-item interaction matrix used in collaborative filtering is often sparse, and the sparsity becomes more pronounced for new items. Since users haven't interacted with the new items, the collaborative signal is weak or absent.

#### **3. Similarity Calculation Challenges:**

- Collaborative filtering calculates item similarity based on user behavior. With new items lacking user interactions, it's challenging to establish meaningful similarity measures, leading to less accurate recommendations.

#### **4. Cold Start for Users:**

- The cold start problem is not only about new items but also about new users. If a new user joins the system, collaborative filtering may struggle to provide accurate recommendations due to a lack of user history.

### **Strategies to Address the Cold Start Problem for New Items:**

#### **1. Content-Based Methods:**

## **Module-**

- Use content-based methods to make recommendations for new items. Content-based filtering relies on item attributes and features, which can be used to recommend items even without historical user interactions.

### **2. Hybrid Models:**

- Combine collaborative filtering with content-based filtering in hybrid models. This allows the system to leverage the strengths of both approaches and mitigate the limitations of each.

### **3. Popularity-Based Recommendations:**

- Initially, recommend popular items that have received interactions from a significant number of users. This helps in providing some level of personalization even for new items.

### **4. Item Seeding:**

- Seed new items with initial interactions to jumpstart the collaborative filtering process. This can involve manually assigning some initial interactions or using implicit feedback mechanisms.

### **5. Exploration Strategies:**

- Implement exploration strategies to encourage users to interact with new items. Techniques like novelty-based recommendations or recommendation diversity can be employed to attract user attention to new items.

### **6. Temporal Considerations:**

- Consider temporal factors when applying collaborative filtering. Gradually integrate new items into the collaborative filtering process as they accumulate interactions over time.

### **7. Hybrid Models with Embeddings:**

- Explore advanced collaborative filtering models with embeddings that can capture latent features even with limited interactions. Techniques like matrix factorization with embeddings can provide better representations for both new items and users.

9. Can you provide examples of real-world applications where social network graphs are used?

CHATGPT:

### **1. Social Media Recommendations:**

- **Application:** Social media platforms use social network graphs to recommend friends, connections, and content to users based on their social interactions, interests, and the activities of their network.

### **2. Friendship Suggestions:**

- **Application:** Social networking sites utilize social graphs to suggest potential connections or friends based on mutual connections, shared interests, or similar profiles.

### **3. Influencer Marketing:**

- **Application:** Brands and marketers analyze social network graphs to identify influencers within a network who can effectively promote their products or services to a specific target audience.

### **4. Community Detection:**

**Module-**

- **Application:** Identifying communities or groups within a social network helps understand the structure of relationships. This is applied in sociology, anthropology, and marketing to study group dynamics.

**5. Personalized Content Recommendations:**

- **Application:** Social network graphs assist in providing personalized content recommendations by considering the preferences and interactions of individuals within a user's social network.

**6. Employment and Recruitment:**

- **Application:** Professional social networks use graph analysis to suggest potential job opportunities, identify candidates, and recommend professional connections based on skills, experiences, and network connections.

**7. Fraud Detection:**

- **Application:** Financial institutions and online platforms utilize social network graphs to detect fraudulent activities by identifying unusual patterns, connections, or collaborations within a network.

10. In the context of a social network graph, what is a community?

**5.3.3 Direct Discovery of Communities in a social graph**

<b>UQ.</b>	Explain the Clique Percolation Method (CPM) used in direct discovery of communities in a social graph with example.	(MU - Dec. 18, 10 Marks)
<b>UQ.</b>	What is a "Community" in a social network Graph?	(MU - May 18, 5 Marks)
<b>UQ.</b>	What is a "Community" in a social Network Graph? Explain any one algorithm for finding communities in a social graph.	(MU - Dec. 17, 10 Marks)

Discovering communities directly by looking for subsets of the nodes that have a relatively large number of edges among them. Interestingly, the technique for doing this search on a large graph involves finding large frequent item sets

(New Syllabus w.e.f academic year 22-23) (M7-80)

**Finding Cliques**

- We want to discover communities directly by looking for subsets of the nodes that have a relatively large number of edges among them
- first thought-finding a large clique
- NP-complete problem, even approximating the maximal clique is hard
- It is possible to have a set of nodes with almost all edges between them, yet with relatively small cliques

**Complete Bipartite Graphs**

- Graph that consists of 's' nodes on one side and 't' nodes on the other side with all 'st' possible edges between them
- It is possible to guarantee that a bipartite graph with many edges has a large complete bipartite sub graph (unlike cliques)
- It might be regarded as the nucleus of community

**Finding Complete Bipartite Sub graphs**

we are given a large bipartite graph  $G$ , and we want to find instances of  $K_{s,t}$  within it. It is similar to finding frequent item sets. Items" on the left side ( $K_{s,t} - t$  nodes there) assumption that  $t \leq s$  The baskets" on the right side. member of the basket are the nodes from left side connected to that node. support threshold  $s$  frequent item set of size  $t$  and  $s$  of the baskets, in which all those items appear, form an instance of  $K_{s,t}$

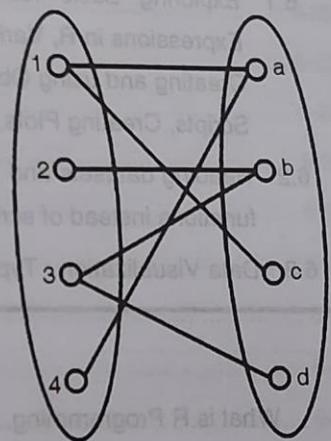


Fig. 5.3.7 : The bipartite graph

**Why Complete Bipartite Graphs Must Exist**

- If there is a community with  $n$  nodes and average degree  $d$ , then this community is guaranteed to have a complete bipartite sub graph  $K_{s,t}$  when:

$$n \frac{\binom{d}{t}}{\binom{n}{t}} \geq s$$

which approximately is :

$$n \left( \frac{d}{n} \right)^t \geq s$$

## Module-

### Betweenness

This method based on finding the edges that are **least likely to be inside a community**. Betweenness of an edge  $(a, b)$  is the number of pairs of nodes  $x$  and  $y$  such that the edge  $(a, b)$  lies on the shortest path between  $x$  and  $y$ . High value suggests that  $(a, b)$  runs between two different communities –  $a$  and  $b$  do not belong to the same community.

### Using Betweenness to Find Communities

- to complete betweenness calculation:
  - repeat these steps for every node as the root and sum the contributions
  - divide by 2, because every shortest path will be discovered twice, once for each endpoints
- betweenness may behave like a distance measure, but is not exactly a distance measure
- ordering edges by betweenness and removing / adding nodes from graph

## 12. How can the concept of communities be applied to enhance the analysis of social networks?

The concept of communities, or groups of nodes with higher internal connectivity compared to external connectivity, can significantly enhance the analysis of social networks. Applying community detection methods can provide valuable insights into the structure, dynamics, and functionality of social networks. Here's how the concept of communities can be applied to enhance social network analysis:

### Identifying Subgroups:

**Application:** Community detection helps identify natural subgroups within a social network, revealing clusters of individuals who share common interests, interactions, or affiliations.

**Benefits:** Enhances understanding of the network's structure, allowing for more targeted analyses of specific subpopulations.

### Understanding Social Structures:

**Application:** Communities provide a lens through which to understand the social structures within a network, including cliques, clusters, and tightly-knit groups.

**Benefits:** Facilitates the identification of influential individuals, key connectors, and the overall hierarchy or organization of relationships.

### Improving Recommendation Systems:

**Application:** Communities can be leveraged to enhance recommendation systems by recommending connections, content, or activities based on the preferences and behaviors of individuals within the same community.

**Benefits:** Increases the relevance and effectiveness of recommendations by considering the preferences of individuals with similar social contexts.

### Enhancing Virality Analysis:

**Module-**

Application: Community structures impact the virality of information or trends. Analyzing communities helps understand how information spreads within and across groups.

Benefits: Aids in predicting and optimizing the virality of content by targeting influential communities and individuals.

**Detecting Anomalies or Outliers:**

Application: Communities can be used to identify anomalous or outlier behavior within a social network. Deviations from the expected community structure may indicate fraudulent activities or unusual patterns.

Benefits: Enhances the detection of suspicious behavior, contributing to security and fraud prevention efforts.

**Improving Targeted Advertising:**

Application: Advertisers can target specific communities with advertisements that are more likely to resonate with the interests and preferences of community members.

Benefits: Increases the effectiveness of targeted advertising by aligning campaigns with the characteristics of distinct social communities.

**Facilitating Content Moderation:**

Application: Communities provide a basis for content moderation strategies. Analyzing community structures helps identify and address issues such as the spread of misinformation or toxic behavior within specific groups.

Benefits: Supports the development of targeted moderation policies to maintain a positive user experience.

13. Outline the main steps of the Girvan-Newman algorithm for detecting communities in a social network graph.

## Module-

### The Girvan-Newman Algorithm

To exploit betweenness, we need to calculate the number of shortest paths going through each edge

Girvan-Newman Algorithm visits each node X once and computes the number of shortest paths from X to other nodes through each of the edges.

#### ► Step I :

**BFS** : Start at a node X, perform a BFS with X as root

**Observe** : level of node Y = length of shortest path from X to Y

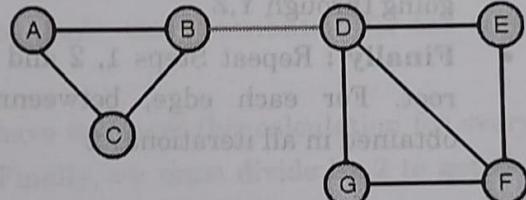


Fig. 5.3.2

Edges between level are called "DAG" edges

Each DAG edge is part of at least one shortest path from X

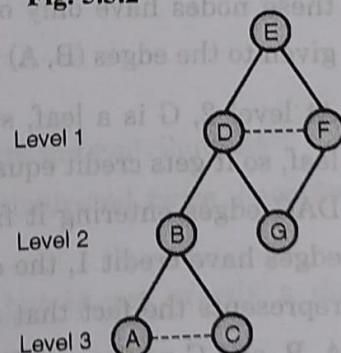


Fig. 5.3.3 : Newman Algorithm

## ► Step II :

Label each node by the number of shortest paths that reach it from the root the sum of the labels of its parents

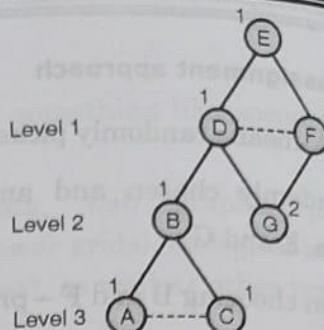


Fig. 5.3.4 : Newman Algorithm

## ► Step III :

- do the calculation starting from the bottom according to rules:
- leaf gets a credit of 1 node that is not a leaf gets a credit equal to 1 plus the sum of the credits of the edges to the level below edge entering node from the level above is given a proportional share of that node

Credit of DAG edges : Let  $Y_i$  ( $i = 1, \dots, k$ ) be parents of  $Z$ ,  $p_i = \text{label}(Y_i)$

$$\text{credit}(Y, Z) = \frac{\text{credit}(Z) \times p_i}{(p_1 + \dots + p_k)}$$

- Intuition :** a DAG edge  $Y, Z$  gets the share of credit of  $Z$  proportional to the # of shortest paths from  $X$  to  $Z$  going through  $Y, Z$
- Finally :** Repeat Steps 1, 2 and 2 with each node as root. For each edge, betweenness = sum credits obtained in all iterations / 2

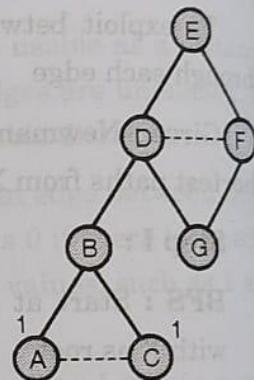


Fig. 5.3.5 : Newman Algorithm

- First, A and C, being leaves, get credit 1. Each of these nodes have only one parent, so their credit is given to the edges (B, A) and (B, C), respectively.
- At level 2, G is a leaf, so it gets credit 1. B is not a leaf, so it gets credit equal to 1 plus the credits on the DAG edges entering it from below. Since both these edges have credit 1, the credit of B is 3. Intuitively 3 represents the fact that all shortest paths from E to A, B, and C go through B. Fig. 5.3.6 shows the credits assigned so far.

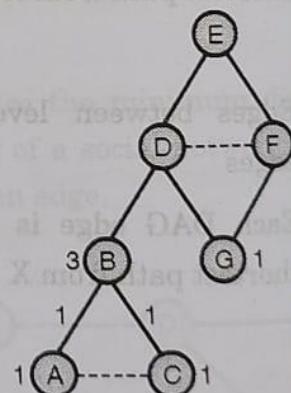


Fig. 5.3.6 : Newman Algorithm – levels 3 and 2

Now, let us proceed to level 1. B has only one parent, D, so the edge (D,B) gets the entire credit of B, which is 3. However, G has two parents, D and F. We therefore need to divide the credit of 1 that G has between the edges (D, G) and (F, G). In what proportion do we divide? If you examine the labels of Fig. 5.3.4, you see that both D and F have label 1, representing the fact that there is one shortest path from E to each of these nodes.

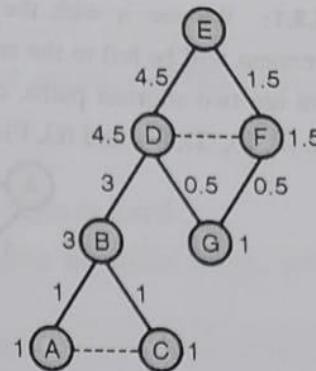


Fig. 5.3.7 : Newman Algorithm

- Thus, we give half the credit of G to each of these edges; i.e., their credit is each  $1/(1+1) = 0.5$ .

- Had the labels of D and F in Fig. 5.3.7 been 5 and 3, meaning there were five shortest paths to D and only three to F, then the credit of edge (D, G) would have been  $5/8$  and the credit of edge (F, G) would have been  $3/8$ .

- Now, we can assign credits to the nodes at level 1. D gets 1 plus the credits of the edges entering it from below, which are 3 and 0.5. That is, the credit of D is 4.5. The credit of F is 1 plus the credit of the edge (F, G), or 1.5. Finally, the edges (E, D) and (E, F) receive the credit of D and F, respectively, since each of these nodes has only one parent. These credits are all shown in Fig. 5.3.7.

- The credit on each of the edges in Fig. 5.3.7 is the contribution to the betweenness of that edge due to shortest paths from E. For example, this contribution for the edge (E, D) is 4.5.
- To complete the betweenness calculation, we have to repeat this calculation for every node as the root and sum the contributions. Finally, we must divide by 2 to get the true betweenness, since every shortest path will be discovered twice, once for each of its endpoints.

### Using Betweenness to Find Communities

- to complete betweenness calculation:
  - repeat these steps for every node as the root and sum the contributions
  - divide by 2, because every shortest path will be discovered twice, once for each endpoints
- betweenness may behave like a distance measure, but is not exactly a distance measure
- ordering edges by betweenness and removing / adding nodes from graph

**Module-**

14. A new e-commerce website is looking to implement a recommendation system to enhance user experience. Compare and contrast content-based recommendations and collaborative filtering, highlighting the advantages and potential challenges of each approach for this business.

**Content-Based Recommendations:**

- **Advantages:**
  - Personalized based on item features.
  - Addresses the cold start problem.
  - Transparent recommendations.
- **Challenges:**
  - Limited in capturing diverse preferences.
  - Depends on item features.

**Collaborative Filtering:**

- **Advantages:**
  - Captures complex user preferences.
  - Effective for diverse products.
- **Challenges:**
  - Cold start problem for new users/items.
  - Sensitive to data sparsity.

**Comparison:**

- **Personalization:**
  - Content-Based: Personalized based on features.
  - Collaborative Filtering: Personalized within user communities.
- **Cold Start:**
  - Content-Based: Effective solution.
  - Collaborative Filtering: Challenges for new items/users.
- **Diversity:**
  - Content-Based: Limited diversity.
  - Collaborative Filtering: Offers diverse recommendations.

**Conclusion for the Business:**

- **New Platform:**
  - Content-based for cold start.
  - Recommending based on item features for personalized onboarding.
- **Established Platform:**
  - Collaborative filtering for community-driven recommendations.

## **Module-**

- Leveraging user behavior for nuanced suggestions.

### **Hybrid Approaches:**

- Consider a hybrid model for a robust recommendation system.
- Combining both approaches can mitigate individual limitations.
- Achieve a balance between personalization and community-driven recommendations.

15. Imagine you are building a recommendation system for an online streaming platform. Explain how contentbased recommendations work and provide an example of how this approach could be applied to suggest movies or TV shows to users with specific preferences.

### **Content-Based Recommendations for an Online Streaming Platform:**

**How Content-Based Recommendations Work:** Content-based recommendation systems analyze the characteristics of items and users' preferences to make personalized suggestions. For an online streaming platform, this involves examining the features of movies or TV shows and recommending content that aligns with the user's past preferences. The system assesses the content itself, such as genres, actors, directors, and plot keywords, to generate recommendations.

#### **Example Application:**

**Scenario:** Let's consider a user who has previously enjoyed science fiction movies and TV shows. The content-based recommendation system would leverage this user's viewing history and preferences to suggest new content.

#### **Steps:**

##### **1. User Profile:**

- The system maintains a user profile based on the user's historical interactions, capturing their preference for science fiction.

##### **2. Feature Extraction:**

- Movies and TV shows in the platform's catalog are analyzed for various features, such as genre, actors, directors, and plot keywords.

##### **3. Content Similarity:**

- The system calculates the similarity between the user's profile and the features of available content, prioritizing items with characteristics similar to the user's preferences.

##### **4. Recommendation Generation:**

- Based on the content similarity scores, the system generates a list of personalized recommendations for the user.

**Example Recommendations:** Considering the user's preference for science fiction, the system might recommend the following movies and TV shows:

##### **1. Interstellar (Movie):**

- *Genre:* Science Fiction
- *Director:* Christopher Nolan
- *Stars:* Matthew McConaughey, Anne Hathaway

## Module-

### 2. Stranger Things (TV Show):

- *Genre:* Science Fiction, Horror
- *Stars:* Winona Ryder, Millie Bobby Brown

### 3. The Martian (Movie):

- *Genre:* Science Fiction
- *Director:* Ridley Scott
- *Stars:* Matt Damon, Jessica Chastain

**Explanation:** The recommended items share key features with the user's previously enjoyed content. For instance, the system suggests "Interstellar" and "The Martian" because they are science fiction movies directed by Christopher Nolan and Ridley Scott, respectively. "Stranger Things" is recommended as it aligns with the user's preference for science fiction and horror genres.

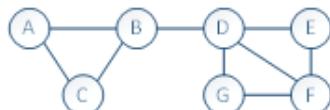
#### Advantages:

- **Personalization:** Tailored recommendations based on individual preferences.
- **Transparency:** Recommendations are explained by the features of the content.
- **Cold Start:** Effective for new users with limited interaction history.

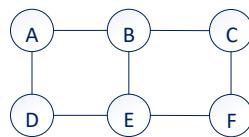
#### Challenges:

- **Limited Diversity:** Recommendations may align closely with the user's existing preferences.
- **Dependency on Features:** The quality of recommendations is highly dependent on the richness and accuracy of item features.

16. What you mean communities in social network. Detect communities in the below given social graph using Girvan-Newman (GN) algorithm.



17. Calculate edge betweenness for every edge in the below social graph. Also detect communities using Girvan-Newman (GN) algorithm.



## 6: Data Analytics with R

6.1 Exploring Basic features of R, Exploring RGUI, Exploring Rstudio, Handling Basic Expressions in R, Variables in R, Working with Vectors, Storing and Calculating Values in R, Creating and using Objects, interacting with users, Handling data in R workspace, Executing Scripts, Creating Plots, Accessing help and documentation in R

6.2 Reading datasets and Exporting data from R, Manipulating and Processing Data in R, Using functions instead of script, built-in functions in R

6.3 Data Visualization: Types, Applications

**Module-****Important Questions**

1. List the basic features of R.

(Data Analytics with R)....Page no. (6-4)

### W 6.3 FEATURES OF R PROGRAMMING LANGUAGE

- R programming is used as a leading tool for machine learning, statistics, and data analysis. Objects, functions, and packages can easily be created by R.
- It's a platform-independent language. That means it can be applied to all operating system.
- It's an open-source free language. That means anyone can install it in any organization without purchasing a license.
- R programming language is not only a statistic package but also allows us to integrate with other languages (C, C++). Thus, you can easily interact with many data sources and statistical packages.
- R is a well-developed, simple and effective programming language which includes conditionals, loops, user defined recursive functions and input and output facilities.
- R has an effective data handling and storage facility,
- R provides a suite of operators for calculations on arrays, lists, vectors and matrices.
- R is currently one of the most requested programming languages in the Data Science job market that makes it the hottest trend nowadays.
- R provides a large, coherent and integrated collection of tools for data analysis.
- R provides graphical facilities for data analysis and display either directly at the computer or printing at the papers.

2. What are the different data structures in R. Explain with example

## 6.12 CREATING AND USING OBJECTS (R DATA STRUCTURES)

- Data structures are very important to understand. Data structure are the objects which we will manipulate in our day-to-day basis in R.
- We can say that everything in R is an object.
- R has many data structures, which include :
 

1. Vector	2. List	3. Array
4. Matrices	5. Data Frame	6. Factors

**GQ.** What is vector? Explain vector creation?

### 1. Vectors

- A vector is the basic data structure in R, or we can say vectors are the most basic R data objects.
- “A vector is a collection of elements which is most commonly of mode character, integer, logical or numeric” When you want to create vector with more than one element, you should use c() function which means to combine the elements into a vector. There are six types of atomic vectors. They are logical, integer, double, complex, character and raw.

#### 1.1 Vector Creation

##### Single Element Vector

In R when we write just one value, it becomes a vector of length 1 and belongs to one of the above vector types.

##### Example of vector creation

```
# Atomic vector of type character.
print("xyz");
# Atomic vector of type double.
print(19.5)
# Atomic vector of type integer.
print(69L)
# Atomic vector of type logical.
print(TRUE)
# Atomic vector of type complex.
print(2+4i)
# Atomic vector of type raw.
print(charToRaw('hello'))
```

**Result**

```
[1] "xyz"
[1] 19.5
[1] 69
[1] TRUE
[1] 2+4i
[1] 68 65 6c 6c 6f
```

**1.2 Using the c() function**

The non-character values are coerced to character type if one of the elements is a character.

**Example of R Programming Using the c() function**

```
# The logical and numeric values are converted to characters.
s <- c('apple','red',5,TRUE)
print(s)
```

**result**

```
[1] "apple" "red" "5" "TRUE"
```

**1.4 Accessing Vector Elements**

- Elements of a Vector are accessed using indexing. The [ ] brackets are used for indexing.
- Indexing starts with position 1. Giving a negative value in the index drops that element from result. **TRUE**, **FALSE** or **0** and **1** can also be used for indexing.

**Example of R Programming Accessing Vector Elements**

```
# Accessing vector elements using position.
t <- c("Sun","Mon","Tue","Wed","Thurs","Fri","Sat")
u <- t[c(2,3,6)]
print(u)

# Accessing vector elements using logical indexing.
v <- t[c(TRUE,FALSE,FALSE, FALSE, FALSE, TRUE, FALSE)]
print(v)

# Accessing vector elements using negative indexing.
x <- t[c(-2,-5)]
print(x)

# Accessing vector elements using 0/1 indexing.
y <- t[c(0,0,0,0,0,0,1)]
print(y)
```



### **Result**

```
[1] "Mon" "Tue" "Fri"  
[1] "Sun" "Fri"  
[1] "Sun" "Tue" "Wed" "Fri" "Sat"  
[1] "Sun"
```

### **2. Lists**

- A list in R is a generic object consisting of an ordered collection of objects. Lists are one-dimensional, heterogeneous data structures. The list can be a list of vectors, a list of matrices, a list of characters and a list of functions, and so on.
- A list is a vector but with heterogeneous data elements. A list in R is created with the use of **list()** function. R allows accessing elements of a list with the use of the index value. In R, the indexing of a list starts with 1 instead of 0 like other programming languages.
- To create a List in R you need to use the function called "**list()**". In other words, a list is a generic vector containing other objects.
- We want to build a list of employees with the details. So for this, we want attributes such as ID, employee name, and the number of employees.

```
e.empList = list(empld, empName, numberofEmp)
```

### **3. Arrays**

- While matrices are confined to two dimensions, arrays can be of any number of dimensions. The **array** function takes a **dim** attribute which creates the required number of dimension.
- In R, an array is created with the help of **array()** function. This function takes a vector as an input and uses the value in the **dim** parameter to create an array.

```
e.g. a <- array(c('green','yellow'),dim = c(3,3,2))
```

### **4. Matrices**

- A matrix is an R object in which the elements are arranged in a two-dimensional rectangular layout.
- In the matrix, elements of the same atomic types are contained. For mathematical calculation, this can use a matrix containing the numeric element. A matrix is created with the help of the **matrix()** function in R.

**Syntax**

The basic syntax of creating a matrix is as follows :

```
1. matrix(data, nrow, ncol, by_row, dim_name)
```

**data:** input vector which becomes data element of matrix

**nrow and ncol:** number of rows and columns

**by\_row:** logical value. If true then input vector elements arranged by rows else by columns.

**Dim\_name:** name assigned to rows and columns.

e.g. M = matrix(c('a','a','b','c','b','a'), nrow = 2, ncol = 3, byrow = TRUE)

**5. Factors**

- These are also data objects that are used to categorize the data and store it as levels. Factors can store both strings and integers.
- Columns have a limited number of unique values so that factors are very useful in columns. It is very useful in data analysis for statistical modeling.
- Factors are created with the help of **factor()** function by taking a vector as an input parameter.

**E.g.**

```
# Create a vector.  
apple_colors<- c('green','green','yellow','red','red','red','green')  
  
# Create a factor object.  
factor_apple<- factor(apple_colors)
```

**6. Data Frames**

- Data frames are tabular data objects. Unlike a matrix in data frame each column can contain different modes of data.
- The first column can be numeric while the second column can be character and third column can be logical. It is a list of vectors of equal length.
- Data Frames are created using the **data.frame()** function.

**E.g.**

```
# Create the data frame.  
BMI <- data.frame(  
  gender = c("Male", "Male","Female"),  
  height = c(152, 171.5, 165),  
  weight = c(81,93, 78),  
  Age = c(42,38,26)  
)
```

(New Syllabus w.e.f academic year 22-23) (M7-80)



Tech-Neo Publications...A SACHIN SHAH Venture

3. Explain the concept of vectors in R. How are they defined, and what operations can be performed on them?

FIRST DS IN ABOVE IMAGES/

## **Module-**

4. Discuss the importance of objects in R. Provide examples of creating objects and illustrate how they are used in R programming.  
Same as data structure.
5. How can R interact with users during script execution? Describe methods for taking user input and displaying output within R scripts.

In R, interaction with users during script execution is facilitated through methods for taking user input and displaying output. Here are some common ways to achieve user interaction in R scripts:

### **Taking User Input:**

1. **readline() Function:**

- **Description:** Reads a line of text from the console, allowing users to input data.
- **Example:**

```
# Taking user input for their name  
  
user_name <- readline(prompt = "Enter your name: ")  
  
# Displaying a greeting using the input  
  
cat("Hello,", user_name, "\n")
```

2. **scan() Function:**

- **Description:** Reads data from the console or a file. Users can input numeric or character values interactively.
- **Example:**

```
# Taking user input for numeric values  
  
user_data <- scan(n = 2, prompt = "Enter two numeric values: ")  
  
# Displaying the sum of the input values  
  
cat("Sum of the input values:", sum(user_data), "\n")
```

### **Displaying Output:**

1. **cat() Function:**

- **Description:** Concatenates and prints the specified arguments.
- **Example:**

```
# Displaying a message  
  
cat("Welcome to R scripting!\n")
```

### **print() Function:**

- **Description:** Prints the specified objects to the console.

## Module-

- Example:

```
# Displaying the result of a calculation  
result <- 5 + 3  
print(result)
```

6. Discuss the capabilities of R for data visualization. Provide examples of creating basic plots using R's plotting functions.

### 6.18 DATA VISUALIZATION IN R

GQ. List and explain different types of Data visualization techniques ?

GQ. What are the different application areas of data visualization ?

**Data visualization** is the technique used to deliver insights in data using visual cues such as graphs, charts, maps, and many others. This is useful as it helps in intuitive and easy understanding of the large quantities of data and thereby make better decisions regarding it.

#### Data Visualization in R Programming Language

- The popular data visualization tools that are available are Tableau, Plotly, R, Google Charts, Infogram, and Kibana.
- The various data visualization platforms have different capabilities, functionality, and use cases. They also require a different skill set. R is a language that is designed for statistical computing, graphical data analysis, and scientific research. It is usually preferred for data visualization as it offers flexibility and minimum required coding through its packages.
- Consider the following *airquality* data set for visualization in R:

Ozone	Solar.R.	Wind	Temp	Month	Day
41	190	7.4	67	5	1
36	118	8.0	72	5	2
12	149	12.6	74	5	3
18	313	11.5	62	5	4
NA	NA	14.3	56	5	5
28	NA	14.9	66	5	6

- Highlighting is accomplished similar to what you may be familiar with in word processor applications.
- You click your left mouse button and the beginning of the text you want to highlight, you hold the mouse button and drag the cursor to the end of the text and release the button. With those four lines of code highlighted, click the editor **Run** button.
- All four lines of code are executed in the command console. That is all it takes to run script code in RStudio.

## 6.16 CREATING PLOTS IN R

### 6.16.1 About Plot Function

- The most used plotting function in R programming is the **plot()** function. It is a generic function, meaning, it has many methods which are called according to the type of object passed to **plot()**.
- In the simplest case, we can pass in a vector and we will get a scatter plot of magnitude vs index. But generally, we pass in two vectors and a scatter plot of these points are plotted.
- For example, the command **plot(c(1,2), c(3,5))** would plot the points (1,3) and (2,5).
- Here is a more concrete example where we plot a sine function from range -pi to pi.

```
x <- seq(-pi,pi,0.1)
plot(x, sin(x))
```

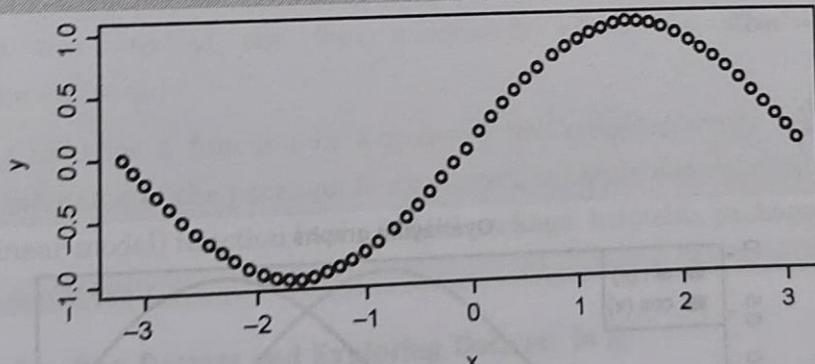


Fig. 6.16.1

### 6.16.2 Adding Titles and Labeling Axes

We can add a title to our plot with the parameter **main**. Similarly, **xlab** and **ylab** can be used to label the x-axis and y-axis respectively.



```
plot(x, sin(x),
main = "The Sine Function",
ylab = "sin(x)")
```

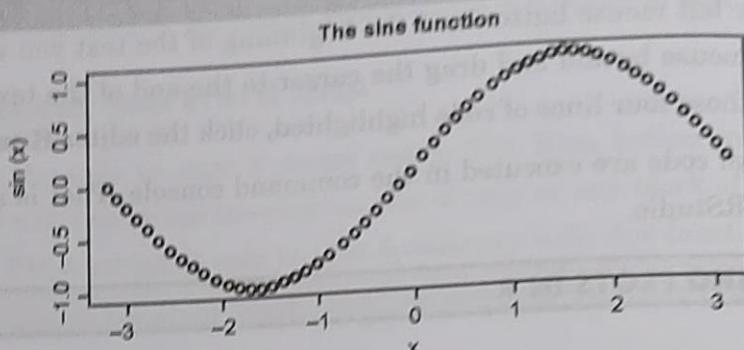


Fig. 6.16.2

### 6.16.3 Overlaying Plots Using legend() function

- Calling `plot()` multiple times will have the effect of plotting the current graph on the same window replacing the previous one.
- However, sometimes we wish to overlay the plots in order to compare the results.
- This is made possible with the functions `lines()` and `points()` to add lines and points respectively, to the existing plot.

```
plot(x, sin(x),
main = "Overlaying Graphs",
ylab = "",
type = "l",
col = "blue")
lines(x, cos(x), col = "red")
legend("topleft",
c("sin(x)", "cos(x")),
fill = c("blue", "red"))
)
```

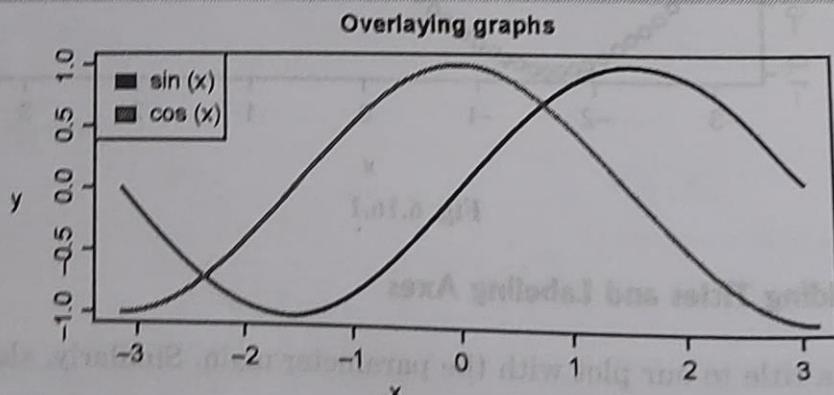


Fig. 6.16.3

## **Module-**

7. Explore some of the essential built-in functions in R. Provide examples of functions for basic statistical calculations and data summarization.

CHATGPT:

### **Basic Statistical Calculations:**

#### **1. mean() Function:**

- **Description:** Calculates the mean (average) of numeric values.
- **Example:**

```
# Calculate the mean of a numeric vector  
  
data <- c(10, 15, 20, 25, 30)  
  
mean_value <- mean(data)  
  
print(mean_value)
```

#### **2. median() Function:**

- **Description:** Calculates the median of numeric values.
- **Example:**

```
# Calculate the median of a numeric vector  
  
data <- c(12, 18, 22, 25, 30)  
  
median_value <- median(data)  
  
print(median_value)
```

#### **1. sd() Function:**

- **Description:** Calculates the standard deviation of numeric values.
- **Example:**

```
# Calculate the standard deviation of a numeric vector  
  
data <- c(5, 8, 12, 15, 20)  
  
sd_value <- sd(data)  
  
print(sd_value)
```

#### **2. var() Function:**

- **Description:** Calculates the variance of numeric values.
- **Example:**

```
# Calculate the variance of a numeric vector  
  
data <- c(8, 10, 12, 14, 16)
```

## **Module-**

```
var_value <- var(data)
```

```
print(var_value)
```

### **3. summary() Function:**

- **Description:** Provides summary statistics (including mean, median, min, max, and quartiles) for numeric vectors.
- **Example:**

```
# Generate summary statistics for a numeric vector
```

```
data <- c(15, 22, 28, 35, 40)
```

```
summary_stats <- summary(data)
```

```
print(summary_stats)
```

## **Data Summarization:**

### **1. table() Function:**

- **Description:** Generates a frequency table for categorical data.
- **Example:**

```
# Create a frequency table for categorical data
```

```
categories <- c("A", "B", "A", "C", "B", "A", "A", "C")
```

```
frequency_table <- table(categories)
```

```
print(frequency_table)
```

### **2. table() with Two Variables:**

- **Description:** Generates a contingency table for two categorical variables.
- **Example:**

```
# Create a contingency table for two categorical variables
```

```
var1 <- c("Yes", "No", "Yes", "No", "Yes")
```

```
var2 <- c("A", "B", "A", "B", "A")
```

```
contingency_table <- table(var1, var2)
```

```
print(contingency_table)
```

### **3. str() Function:**

- **Description:** Displays the structure of an R object, providing information about its type and content.
- **Example:**

```
# Display the structure of a data frame df <- data.frame(Name = c("Alice", "Bob", "Charlie"), Age = c(25, 30, 22)) str(df)
```

## Module-

8. Define a vector in R. How can you create a numeric vector with values 1, 2, 3, 4, and 5?

In R, a vector is a one-dimensional array that can hold elements of the same data type. Vectors can be created using the `c()` function, which stands for "combine" or "concatenate." To create a numeric vector with values 1, 2, 3, 4, and 5, you can use the following syntax:

```
# Creating a numeric vector with values 1, 2, 3, 4, and 5
```

```
numeric_vector <- c(1, 2, 3, 4, 5)
```

```
# Printing the created vector
```

```
print(numeric_vector)
```

In this example:

`c()` is the combine function, and it's used to concatenate values into a vector.

The values 1, 2, 3, 4, and 5 are enclosed within the parentheses, forming a numeric vector.

After running this code, the variable `numeric_vector` will contain the numeric vector with the specified values. The `print()` function is used to display the contents of the vector:

```
[1] 1 2 3 4 5
```

Here, [1] indicates the first element of the vector, followed by the values 1, 2, 3, 4, and 5.

9. Perform element-wise multiplication using R on two vectors: `vec1 = c(2, 4, 6)` and `vec2 = c(1, 2, 3)`.

CHATGPT:

In R, you can perform element-wise multiplication on two vectors using the `*` operator. Here's how you can perform element-wise multiplication on the given vectors `vec1` and `vec2`:

```
# Define the vectors
```

```
vec1 <- c(2, 4, 6)
```

```
vec2 <- c(1, 2, 3)
```

```
# Perform element-wise multiplication
```

```
result_vector <- vec1 * vec2
```

```
# Print the result
```

```
print(result_vector)
```

When you run this code, the `result_vector` will contain the element-wise multiplication of the corresponding elements from `vec1` and `vec2`. The output will be:

```
[1] 2 8 18
```

## Module-

Explanation:

The element-wise multiplication multiplies the corresponding elements of vec1 and vec2.

2 \* 1 results in 2, 4 \* 2 results in 8, and 6 \* 3 results in 18.

10. Define a matrix in R. Create a 3x3 matrix with values 1 to 9.

CHATGPT:

In R, a matrix is a two-dimensional array that can store elements of the same data type. You can create a matrix using the **matrix()** function. Here's how you can define a 3x3 matrix with values 1 to 9:

```
# Create a 3x3 matrix with values 1 to 9  
  
my_matrix <- matrix(1:9, nrow = 3, ncol = 3)  
  
# Print the matrix  
  
print(my_matrix)
```

In this example:

- **1:9** generates a sequence of numbers from 1 to 9.
- The **matrix()** function reshapes the sequence into a 3x3 matrix with 3 rows (**nrow = 3**) and 3 columns (**ncol = 3**).

The output will be:

```
[,1] [,2] [,3]  
  
[1] 1 4 7  
[2] 2 5 8  
[3] 3 6 9
```

This matrix has values ranging from 1 to 9 arranged in a 3x3 grid. The first column contains values 1, 2, 3; the second column contains values 4, 5, 6; and the third column contains values 7, 8, 9.

11. Define a factor in R. Create a factor vector with levels "Low," "Medium," and "High."

CHATGPT:

In R, a factor is a categorical variable that can take on a limited and fixed set of values known as levels. You can create a factor using the **factor()** function. Here's how you can define a factor vector with levels "Low," "Medium," and "High":

```
# Create a factor vector with levels "Low," "Medium," and "High"  
  
factor_vector <- factor(c("Low", "Medium", "High", "Medium", "Low"))  
  
# Print the factor vector  
  
print(factor_vector)
```

In this example:

**Module-**

- The c() function is used to create a character vector with values "Low," "Medium," "High," "Medium," and "Low."
- The factor() function is then applied to convert the character vector into a factor.
- By default, levels are assigned based on the unique values present in the vector.

**The output will be:**

[1] Low Medium High Medium Low

Levels: High Low Medium

This factor vector has levels "Low," "Medium," and "High," and each element in the vector is associated with its corresponding level. The levels are automatically determined based on the unique values present in the original character vector.

12. Your team is preparing a presentation for a client, and visual representation of data is crucial. Discuss the different types of data visualizations available in R and provide examples of when to use scatter plots, bar charts, and line graphs based on the nature of the data.

**6.18 DATA VISUALIZATION IN R**

GQ. List and explain different types of Data visualization techniques ?

GQ. What are the different application areas of data visualization ?

**Data visualization** is the technique used to deliver insights in data using visual cues such as graphs, charts, maps, and many others. This is useful as it helps in intuitive and easy understanding of the large quantities of data and thereby make better decisions regarding it.

**Data Visualization in R Programming Language**

- The popular data visualization tools that are available are Tableau, Plotly, R, Google Charts, Infogram, and Kibana.
- The various data visualization platforms have different capabilities, functionality, and use cases. They also require a different skill set. R is a language that is designed for statistical computing, graphical data analysis, and scientific research. It is usually preferred for data visualization as it offers flexibility and minimum required coding through its packages.
- Consider the following *airquality* data set for visualization in R:

Ozone	Solar.R.	Wind	Temp	Month	Day
41	190	7.4	67	5	1
36	118	8.0	72	5	2
12	149	12.6	74	5	3
18	313	11.5	62	5	4
NA	NA	14.3	56	5	5
28	NA	14.9	66	5	6

### 6.18.1 Types of Data Visualizations

Some of the types of visualizations offered by R are as follow :

#### 1. Bar Plot

- There are two types of bar plots- horizontal and vertical which represent data points as horizontal or vertical bars of certain lengths proportional to the value of the data item.
- They are generally used for continuous and categorical variable plotting. By setting the **horiz** parameter to true and false, we can get horizontal and vertical bar plots respectively.

```
barplot(airquality$Ozone,  
       main = 'Ozone Concentration in air',  
       xlab = 'ozone levels', horiz = TRUE)
```

#### Result

Ozone concentration in air

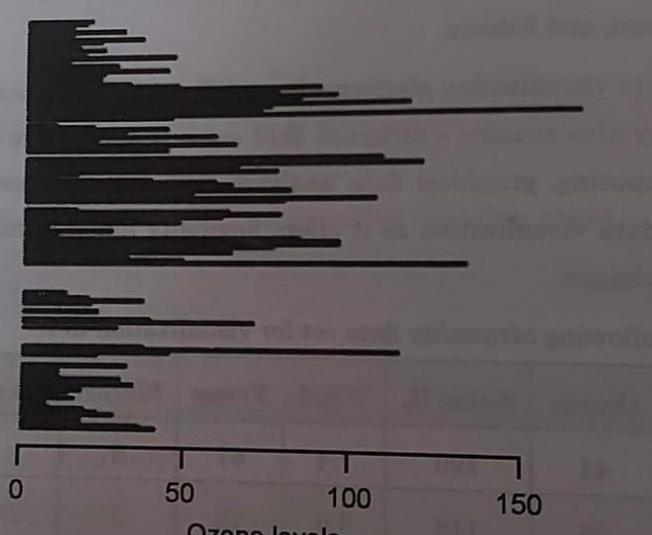


Fig. 6.18.1

Bar plots are used for the following scenarios :

- To perform a comparative study between the various data categories in the data set.
- To analyze the change of a variable over time in months or years.

**Histograms are used in the following scenarios**

- To verify an equal and symmetric distribution of the data.
- To identify deviations from expected values.

### 3. Box Plot

- The statistical summary of the given data is presented graphically using a boxplot.
- A boxplot depicts information like the minimum and maximum data point, the median value, first and third quartile, and interquartile range.

**Example :**

```
data(airquality)
boxplot(airquality$Wind, main = "Average wind speed
at La Guardia Airport",
xlab = "Miles per hour", ylab = "Wind",
col = "orange", border = "brown",
horizontal = TRUE, notch = TRUE)
```

**Output**

Average wind speed at La Guardia Airport

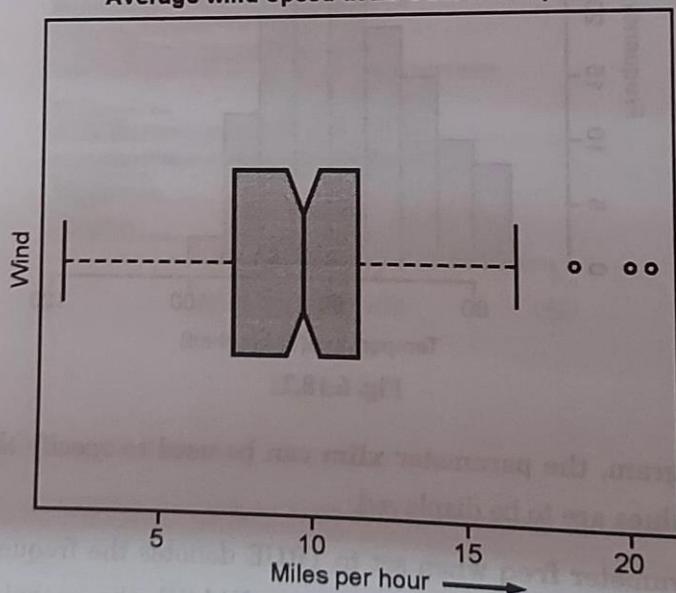


Fig. 6.18.3

## Module-

Multiple box plots can also be generated at once through the following code

Example :

```
R# Multiple Box plots, each representing
# an Air Quality Parameter
boxplot(airquality[, 0:4],
        main ='Box Plots for Air Quality Parameters')
```

### Output

Output: Box plots for air quality parameters

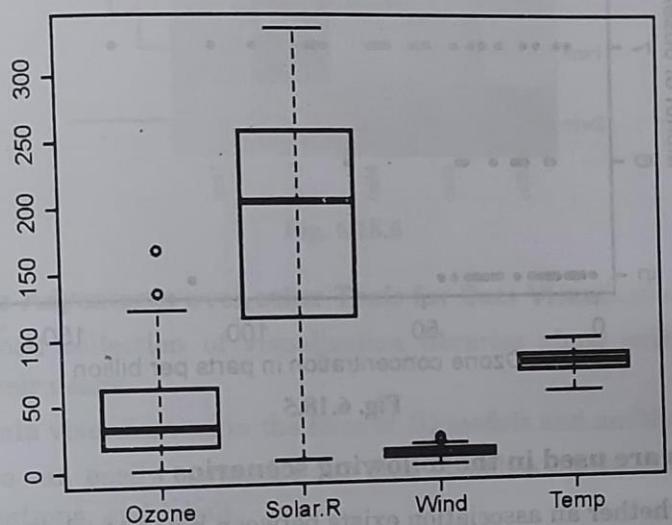


Fig. 6.18.4

### Box Plots are used for :

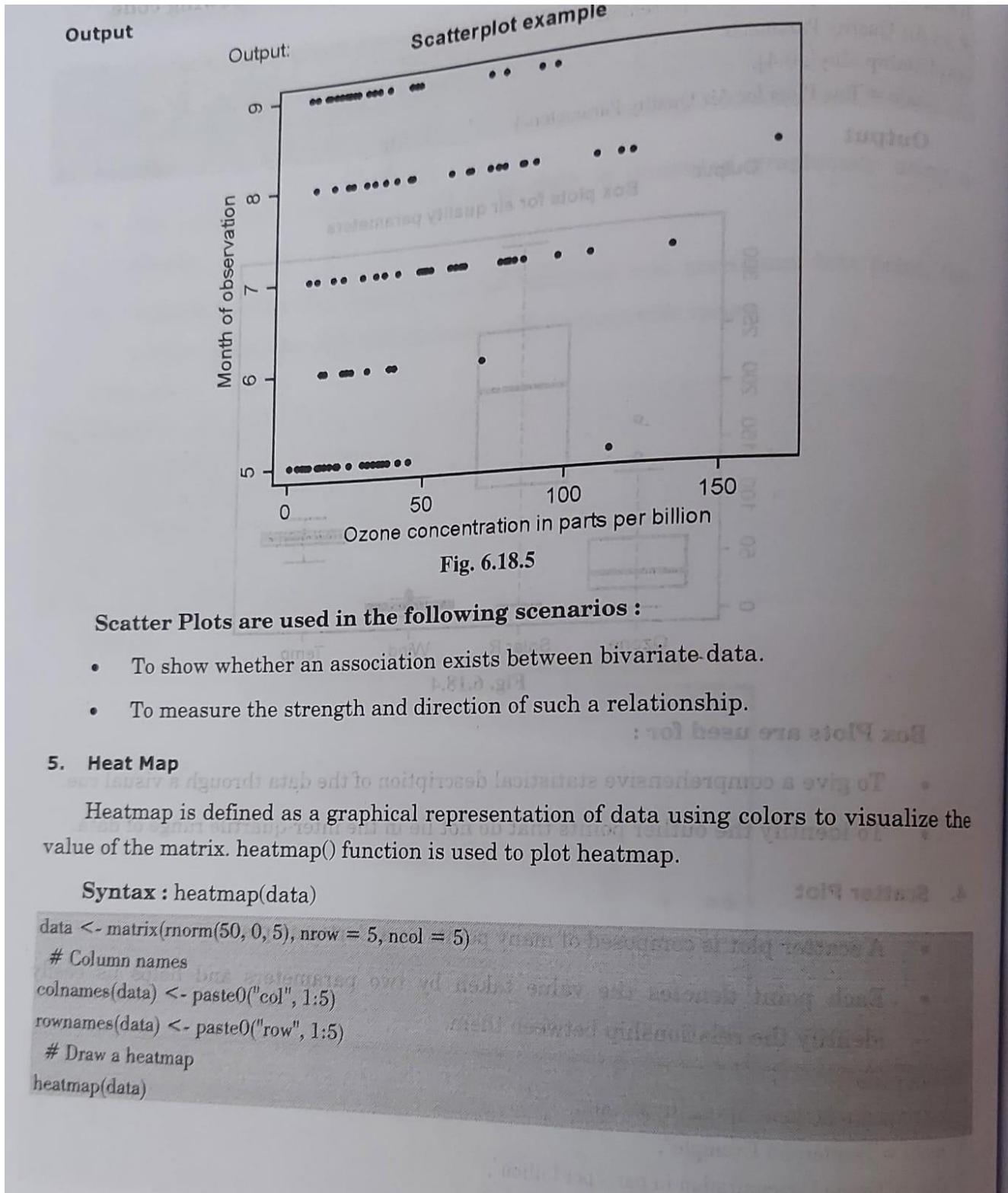
- To give a comprehensive statistical description of the data through a visual cue.
- To identify the outlier points that do not lie in the inter-quartile range of data.

### 4. Scatter Plot

- A scatter plot is composed of many points on a Cartesian plane.
- Each point denotes the value taken by two parameters and helps us easily identify the relationship between them.

```
data(airquality)
plot(airquality$Ozone, airquality$Month,
      main ="Scatterplot Example",
      xlab ="Ozone Concentration in parts per billion",
      ylab ="Month of observation ", pch = 19)
```

## Module-



13. The marketing department wants to convey sales trends over the past year to stakeholders. Illustrate how you would choose and create an appropriate data visualization in R to effectively communicate this information. Emphasize the importance of selecting the right visualization type for the intended message.

CHATGPT:

To effectively communicate sales trends over the past year to stakeholders, you need to choose a data visualization that best highlights the patterns and insights in the sales data. Here's a step-by-step guide on how to select and create an appropriate data visualization in R for this purpose:

## **Module-**

### 1. Examine the Data:

- Understand the characteristics of the sales data, such as the time period covered, the frequency of data points (daily, monthly), and any notable trends or fluctuations.

### 2. Select an Appropriate Visualization:

- Based on the nature of the data and the story you want to convey, consider the following options:
  - Line Chart: Suitable for showing trends and variations over time.
  - Bar Chart: Effective for comparing sales figures for different periods.
  - Area Chart: Similar to a line chart but emphasizes the area under the line, useful for showing cumulative sales over time.

### 3. Prepare the Data:

- Organize the sales data with columns for the time period (e.g., months) and corresponding sales values.

### 4. Create the Visualization:

- Example: Line Chart

```
# Sample data (replace with your actual sales data)
time_period <- seq(as.Date("2022-01-01"), as.Date("2022-12-31"), by = "months")
sales_data <- c(10000, 12000, 15000, 18000, 20000, 22000, 21000, 19000, 17000, 16000, 18000, 20000)

# Create a data frame
sales_df <- data.frame(Time_Period = time_period, Sales = sales_data)

# Plotting a line chart
plot(sales_df$Time_Period, sales_df$Sales, type = "l", col = "blue", lwd = 2,
      main = "Sales Trends Over the Past Year", xlab = "Time Period", ylab = "Sales Amount")

# Add labels and grid lines for better readability
title(main = "Sales Trends Over the Past Year", sub = "Monthly Sales Data", col.main = "blue", font.main = 4)
```

**Module-**

```
abline(h = 0, col = "gray", lty = 2
```

## 5. Enhance Readability:

- Add labels, titles, and annotations to provide context and highlight key insights.
- Consider using colors, legends, or shading to distinguish different categories or trends.

## 6. Interpret the Visualization:

- When presenting the visualization, provide context and interpretations of notable points, trends, or changes.
- Highlight key insights that stakeholders should focus on.

## 7. Iterative Process:

- Don't hesitate to iterate and refine the visualization based on feedback or changing requirements.

## 8. Consider Interactive Visualizations:

- For presentations or reports, consider creating interactive visualizations using tools like Shiny or other R packages that allow stakeholders to explore the data dynamically.

**Key Points:**

- Choose a Visualization That Tells a Story: The selected visualization should effectively tell the story of sales trends over the past year. Consider what message you want stakeholders to take away from the data.
- Clarity is Key: Keep the visualization simple, clear, and easy to interpret. Avoid clutter and unnecessary details.
- Align with Stakeholder Preferences: Consider the preferences and familiarity of stakeholders with different visualization types.

By selecting the appropriate visualization type and creating a clear and informative plot, you can effectively convey sales trends to stakeholders, facilitating better decision-making and understanding of the data.

14. Your manager has requested a summary report of specific metrics from a large dataset. Explain how you would manipulate and process data in R to extract the necessary information efficiently. Highlight the use of functions to streamline the analysis process.

**CHATGPT:**

To efficiently extract specific metrics from a large dataset in R, you can follow a systematic approach involving data manipulation and processing. Here's a step-by-step guide, emphasizing the use of functions to streamline the analysis process:

## 1. Import the Data:

- Use functions like `read.csv()`, `read.table()`, or others depending on the data format to import the dataset into R.

```
# Example: Reading a CSV file
```

## **Module-**

```
data <- read.csv("your_dataset.csv")
```

### 2. Explore and Understand the Data:

- Utilize functions like `head()`, `summary()`, and `str()` to get an overview of the dataset, understand variable types, and identify potential issues.

```
# Display the first few rows
```

```
head(data)
```

```
# Summary statistics
```

```
summary(data)
```

```
# Structure of the dataset
```

```
str(data)
```

### 3. Data Cleaning:

- Use functions to handle missing values (`na.omit()`, `complete.cases()`), filter rows (`subset()`), and address any other data quality issues.

```
# Remove rows with missing values
```

```
clean_data <- na.omit(data)
```

```
# Filter data based on a condition
```

```
filtered_data <- subset(clean_data, column_name > threshold)
```

### 4. Subset the Data:

- Create a subset of the data containing only the relevant columns or rows using functions like `subset()` or indexing.

```
# Subset data based on specific columns
```

```
relevant_data <- data[, c("column1", "column2")]
```

```
# Subset data based on a condition
```

```
filtered_data <- data[data$column_name > threshold, ]
```

### 5. Calculate Metrics:

Utilize functions to calculate the necessary metrics. For example, use `mean()`, `sum()`, or custom functions for more complex calculations.

```
# Calculate mean of a column
```

```
mean_value <- mean(data$column_name)
```

```
# Calculate sum of a column
```

```
total_sum <- sum(data$numeric_column)
```

### 6. Aggregation and Grouping:

## **Module-**

- Use functions like `aggregate()`, `tapply()`, or `dplyr` functions (`group\_by()`, `summarize()`) for aggregating data based on specific categories.

```
# Aggregate data by a categorical variable  
aggregated_data <- aggregate(numeric_column ~ category_column, data, mean)
```

### 7. Data Visualization (Optional):

- If needed, use functions like `plot()`, `ggplot2`, or other visualization packages to create informative plots summarizing the data.

```
# Example: Creating a histogram  
hist(data$numeric_column, main = "Histogram of Numeric Column", xlab = "Values", col = "lightblue")
```

### 8. Create Summary Report:

- Use functions like `write.csv()`, `write.table()`, or `knitr` to export the results to a summary report in a suitable format.

```
# Example: Write summary data to a CSV file  
write.csv(aggregated_data, "summary_report.csv", row.names = FALSE)
```

### 9. Document Your Analysis:

- Provide comments and documentation within your R script or notebook to explain the steps taken, rationale behind decisions, and any assumptions made during the analysis.

By systematically applying these steps and leveraging relevant functions, you can efficiently manipulate and process large datasets in R, extracting the necessary information to generate a concise summary report for your manager. The use of functions enhances code readability, maintainability, and reproducibility.