Assignment No.2

1. Explain the architecture of Data Stream Management system (DSMS) and give any two applications of DSMS.

Ans: A Data Stream Management System (DSMS) is a specialized system designed to process continuous, high-velocity data streams in real-time. Unlike a traditional Database Management System (DBMS) that stores static data for later querying, a DSMS must handle transient data that is too large to store in its entirety.

DSMS Architecture

A generic DSMS is architected to ingest, process, and output data from one or more continuous streams. The flow of data is managed by several key components.

The primary components and their functions are:

- Input Monitor/Regulator: This is the entry point for data streams. It regulates the rate of incoming data, which can be unpredictable and variable. If the arrival rate is too high for the system to handle, the regulator may need to drop data packets to manage the load.
- Storage: Since storing the entire stream is infeasible, a DSMS uses a multi-layered storage approach:
    - Working Storage: A limited, temporary memory buffer used to hold recent portions of a stream (e.g., a "sliding window") for active query processing.
    - Summary Storage: This holds compact summaries, or synopses, of the data stream. These summaries (like sketches or histograms) allow the system to provide approximate answers to queries about historical data without storing every element.
    - Static Storage: This repository holds metadata about the streams, such as their sources, data types, or user-defined schemas. • Query Processor: This is the engine of the DSMS. It executes

continuous queries that are long-running and provide incrementally updated results as new data arrives. It interacts with all storage layers to process these queries and can re-optimize query plans based on changing stream conditions.

- Query Repository: This component stores the specifications for continuous queries that users have registered. This allows the DSMS to manage multiple queries simultaneously and identify opportunities for shared processing if queries have common subexpressions.
- User Interface: This interface allows users to submit queries and receive results. Output can be streamed back to the user or temporarily buffered for retrieval.

Applications of DSMS

DSMS are crucial in domains where real-time analysis of continuous data is required. Two key applications are:

1. Sensor Networks: Modern environments are filled with sensors that generate continuous data streams, such as temperature readings at a weather station or power usage statistics from a smart grid. A DSMS can process these streams to perform realtime monitoring and alerting. For example, a DSMS could execute a continuous query that joins temperature and ocean current streams to provide early warnings for disasters like cyclones.
2. Network Traffic Analysis: Internet service providers and network administrators use DSMS to analyze high-volume network traffic streams. This allows them to detect anomalies that could indicate fraudulent activity, denial-of-service attacks, or potential network congestion. A typical query might be to continuously check if a stream of network actions within a given time window matches the pattern of a known intrusion.

2. Describe:

a. **Continuous Queries:** A Continuous Query is a long-running or "standing" query evaluated continuously as a data stream arrives, with its answer being produced over time to always reflect the most current data.

Unlike a traditional database query that runs once and terminates, a continuous query is registered with a Data Stream Management System (DSMS) and remains active, constantly processing new data elements as they arrive. This makes them ideal for realtime monitoring and analytics applications. Key Characteristics:

- Persistent Nature: They are also known as persistent or standing queries because they are stored in the system's query repository and run indefinitely.
- Dynamic Output: The answer to a continuous query is not a single, static result. Instead, the result is updated over time as new data is seen. The output can either be stored and updated (e.g., the current maximum value) or produced as a new data stream itself.
- Pre-defined Execution: Continuous queries are typically pre-defined, meaning they are supplied to the DSMS before the relevant data has arrived. This allows the system to optimize their execution, often by sharing computations among multiple similar queries.

Example

A common example is monitoring stock prices. A user might register a continuous query such as: "Alert me when the price of stock XYZ crosses a particular price point." The DSMS will continuously monitor the stream of stock trades and only produce an output when the condition is met. Another example is calculating a moving average, where the system continuously updates the average price of a stock over all time by adjusting its calculation each time a new reading arrives.

b. **Adhoc Queries:** An ad-hoc query is a "one-time" query that a user issues online, after a data stream has already begun, to ask a question about the current state of that stream.Unlike pre-defined, continuous queries, ad-hoc queries are not known to the Data Stream Management System (DSMS) in advance. This spontaneity makes them more challenging to process for several reasons. Key Characteristics:

- One-Time Execution: Ad-hoc queries are typically executed once to get a snapshot of the stream at a particular moment. For example, a user might ask, "What was the average temperature from sensor X over the last minute?"
- Unpredictability: Since they are not known in advance, the DSMS cannot perform preoptimization or set up specialized data structures to answer them efficiently.
- Reliance on Historical Data: The most significant challenge is that an ad-hoc query may require information about data elements that have already passed through the stream and have been discarded due to memory limitations.

How DSMS Handle Ad-hoc Queries

Because it's impossible to store the entire stream, a DSMS cannot answer arbitrary adhoc queries with perfect accuracy. Instead, the system prepares for anticipated *types* of queries by storing compact summaries or synopses of the stream data. When an adhoc query arrives, it is executed against these summaries to provide a highquality approximate answer rather than an exact one.

3. Explain DGIM Algorithm. Consider one data stream and for given k , count no. of 1. The Datar-Gionis-Indyk-Motwani (DGIM) algorithm is a space-efficient method for estimating the number of 1s in a sliding window over a binary stream. It provides an approximate count with a guaranteed error of no more than 50% while using significantly less memory than storing the entire window.

How the DGIM Algorithm Works

The algorithm's core idea is to summarize the stream's window by grouping 1s into buckets. Instead of storing the position of every 1, it only stores metadata about these buckets.

Bucket Representation

Each bucket in the DGIM algorithm has two main properties:

1. Timestamp: This is the position of the bucket's most recent (rightmost) 1 within the window of size N. Positions are numbered from right to left, with the newest bit at position 1.
2. Size: This is the number of 1s the bucket represents.

The structure is maintained by a strict set of rules:

- Power-of-2 Sizes: The number of 1s in any bucket must be a power of 2 (e.g., 1, 2, 4, 8, ...).
- One or Two Buckets per Size: For each size (like 1, 2, 4, etc.), there can be either one or two buckets.
- Non-decreasing Size: As you go back in time (from newer to older buckets), the bucket sizes cannot decrease.

- Merging Buckets: When a new 1 arrives, a new bucket of size 1 is created. If this results in three buckets of the same size, the two oldest are merged into a single bucket of double the size. This merge may cascade, forcing subsequent merges of larger-sized buckets to maintain the "one or two per size" rule.

This structure ensures that for a window of size N, the stream is represented by only buckets, leading to a total space requirement of bits.

Example: Counting 1s in a Stream

Let's consider a binary stream and use the DGIM algorithm to estimate the number of 1s in the last k=14 bits.

Stream Window (last 19 bits shown): ...0101101011001011001 Query: How many 1s are in the last 14 bits? Actual Window: 101011001011001 Actual Count: There are 8 ones in this window.

Assume the DGIM algorithm has summarized the stream with the following buckets (ordered from newest to oldest):

- Bucket 1: Timestamp 1, Size 1
- Bucket 2: Timestamp 5, Size 1
- Bucket 3: Timestamp 7, Size 2
- Bucket 4: Timestamp 11, Size 2
- Bucket 5: Timestamp 18, Size 4

Answering the Query

To estimate the number of 1s in the last 14 bits, we perform the following steps:

1. Sum sizes of all buckets fully within the window. Buckets 1, 2, 3, and 4 have timestamps (1, 5, 7, 11) that are all . We add their sizes:

1 + 1 + 2 + 2 = 6.

2. Identify the last (oldest) bucket that is partially in the window. Bucket 5 has a timestamp of 18, which is older than our window size of 14. This means some of its 1s are inside the window and some are outside.
3. Estimate the contribution of the partial bucket. For this last bucket, we add half of its size to our sum.

Contribution = Size of Bucket 5 / 2 = 4 / 2 = 2.

4. Calculate the final estimate. We add the sum from the full buckets and the estimate from the partial bucket.

Final Estimate = 6 + 2 = 8.

In this case, the estimated count is 8, which matches the actual count. The DGIM algorithm's design ensures that this estimate is always within 50% of the true value.

4. What are the Issues in Data Stream Query Processing? Explain in detail.

Query processing in the data stream model of computation presents several unique challenges because data arrives continuously, rapidly, and possibly without bound. Unlike traditional databases, data streams cannot be stored in their entirety, and queries often need to be answered in real time.The main issues in data stream query processing are as follows:

1. Unbounded Memory Requirements

- Data streams are potentially infinite in size, so the amount of storage needed to compute an exact query answer may also grow indefinitely.

- Algorithms using external memory (like disk storage) are too slow for real-time processing and do not support continuous queries efficiently.
- Therefore, algorithms for stream processing must work within main memory and minimize per-element computation time to keep up with the incoming data.

2. Approximate Query Answering

- Since only limited memory is available, it is often impossible to produce exact answers.
- Instead, approximate answers are computed, which are usually sufficient for practical purposes.
- This is achieved using techniques like:
  - Sampling
  - Histograms
  - Wavelets
  - Sketches
- These techniques help summarize data into compact synopses that can provide nearaccurate results for many queries.

3. Sliding Windows

- Instead of considering the entire data history, queries are evaluated only on recent data within a fixed-size or time-based sliding window.
- For example, a query may consider only the last *10,000 transactions* or data from the *past week*.
- This method:
  - Emphasizes recent and relevant data.
  - Keeps memory usage bounded. ○ Allows queries like "average call duration in the last 10 calls" or "most frequent errors in the past hour."

4. Batch Processing, Sampling, and Synopses

- Another method for approximate answers is batch processing, where data is collected for a period, and queries are computed periodically rather than continuously.
- This sacrifices timeliness for accuracy and is effective when data streams are bursty.
- Sampling techniques evaluate queries on selected subsets of the data stream, skipping some data points to keep computation feasible.
- Synopses (or sketches) are compact data structures that summarize streams efficiently for approximate query answering.

5. Blocking Operators

- Certain query operators, known as blocking operators, cannot produce output until all input data has been processed (e.g., SORT, SUM, COUNT, AVG).
- Since data streams are unbounded, such operators would never finish processing.
- Therefore, traditional blocking operators must be avoided or replaced with incremental, non-blocking alternatives that can produce partial results continuously.

5. Explain the Content based Recommendation Systems and collaborative filtering-based recommendation system in detail.

1. Collaborative Filtering-Based Recommendation System

Collaborative filtering focuses on the relationship between users and items. It recommends items to a user based on the preferences of other users with similar tastes.

Key Concepts

- It relies on community data (ratings, reviews, preferences).
- The main assumption is:
    1. Users who agreed in the past will agree again in the future.
    2. Users with similar tastes will prefer similar items.
- Used widely in e-commerce platforms, movie, and music recommendation (e.g., Amazon, Netflix).

Types

1. User-Based Collaborative Filtering:
   Finds users similar to the active user and recommends items liked by them.
   - Example: "Users who liked the same movies as you also liked..."
2. Item-Based Collaborative Filtering:
   Finds items similar to those the user already liked and recommends them.
   - Example: "People who bought this book also bought..."

Nearest-Neighbor Technique

- Predicts rating of an unseen item for an active user by finding "neighbors" (users with similar preferences).
- Similarity between users is measured using Pearson Correlation Coefficient:

$$sim(a,b) = \frac{\sum_{p \in P} (r_{a,p} - \bar{r}_a)(r_{b,p} - \bar{r}_b)}{\sqrt{\sum_{p \in P} (r_{a,p} - \bar{r}_a)^2} \sqrt{\sum_{p \in P} (r_{b,p} - \bar{r}_b)^2}}$$

- Prediction function:

$$pred(a,p) = \bar{r}_a + \frac{\sum_{b \in N} sim(a,b)(r_{b,p} - \bar{r}_b)}{\sum_{b \in N} |sim(a,b)|}$$

Example

In a movie rating system, if users "Tom" and "Jack" have rated movies similarly to "Tim", the system predicts "Tim's" rating for a new movie based on their ratings.

Issues

1. Scalability – Large user bases increase computation.
2. Sparsity – Few users rate many items, leading to missing data.
3. Cold Start – Hard to recommend for new users or items with no ratings.

Methods to Improve Prediction

- Weight neighbors with higher variance or higher similarity.
- Set a similarity threshold to include only the most relevant peers.

- Use item-based collaborative filtering for scalability (used by Amazon).

2. Content-Based Recommendation System

A content-based system recommends items similar to what the user liked in the past, based on item features and user profile.

Key Concepts

- It uses attributes of items (content) such as genre, keywords, author, etc.
- No need for community data; recommendations depend solely on user history.
- Example: If a user watches many "science fiction" movies, recommend more movies with the same genre.

Steps in Content-Based Filtering a)

Item Profile Creation

Each item is represented by its important features.

For example, a movie can be represented by:

- Actors
- Director
- Year of release
- Genre

b) Feature Extraction

Features are extracted using Information Retrieval techniques.

Common measure: TF-IDF (Term Frequency–Inverse Document Frequency)

$$TF\text{-}IDF(i,j) = TF(i,j) \times IDF(i)$$

Where,

- $TF(i,j) = \frac{freq(i,j)}{max(freq(k,j))\ N}$
- $IDF(i) = \log(\frac{}{n(i)})$

This highlights important terms while reducing weight of common terms.

c) Similarity Computation

Similarity between items is measured using:

- Dice Coefficient

$$sim(b_1, b_2) = \frac{2\,|keywords(b_1) \cap keywords(b_2)|}{|keywords(b_1)| + |keywords(b_2)|}$$

- Cosine Similarity

$$a.b$$

$$sim^{(a,b)} = \frac{}{| a || b |}$$

d) User Profile Construction
User profile summarizes user preferences by analyzing features of items they liked earlier.

Advantages
- Personalized recommendations without needing data from other users.
- Useful when user feedback is limited to a single user's actions.

Limitations
- Cannot recommend new types of items (Cold Start Problem).
- Struggles with diversity (keeps recommending similar content).
- Needs detailed content description for every item.

6. **Define Social Networks and Social Network Mining.** a social network is a social structure made of nodes that are generally individuals or organizations.
It represents the relationships and flows between people, groups, organizations, computers, or other information-processing entities.The term *"Social Network"* was coined in 1954 by J. A. Barnes.

Characteristics of Social Networks 1.
Nodes and Entities:
Nodes typically represent people, but may also represent companies, documents, or computers.
2. Heterogeneous and Multi-relational:
A social network can be viewed as a heterogeneous and multi-relational dataset represented as a graph.
Nodes and edges may have attributes, and objects may have class labels.
3. Relationships Between Entities:
There is always at least one relationship between entities — for example, "Friends" in Facebook or "Connect" in LinkedIn.
4. Weighted or Graded Relationships:
Relationships can have degrees or strengths, not just binary (yes/no).
Example: Levels of endorsement such as *Novice*, *Intermediate*, or *Expert*.
5. Locality (Non-randomness):
Social networks exhibit locality, meaning nodes and edges tend to cluster into communities.
If person A is connected to both B and C, there's a higher probability that B and C are also connected.

2. Definition of Social Network Mining

Social Network Mining refers to the process of analyzing and extracting patterns from social network data to gain insights, make predictions, or support decision-making. Mining these structures for patterns yields knowledge that can be effectively used for predictions and decision-making.

Goals of Social Network Mining

- To discover patterns, relationships, and communities among network entities.
- To identify influential nodes (individuals or groups).
- To analyze behavior and information flow in networks.

Applications of Social Network Mining

1. Viral Marketing: Identifies influential users to optimize word-of-mouth advertising.
2. E-commerce: Groups customers with similar buying profiles to create personalized recommendation systems.
3. Network Analysis: Used in data aggregation, propagation modeling, user behavior analysis, recommender systems, and location-based analysis.
4. Business Intelligence: Helps in customer interaction, targeted marketing, and system development analysis.

Representation as a Graph

Social networks are naturally represented as graphs:

- Nodes (Vertices): represent entities such as users or organizations.
- Edges: represent relationships or interactions between entities.

Example:

On LinkedIn, a connection between two users represents an edge, and a node's out-degree may indicate its influence level.

7. **Which function is used to concatenate text values in R. Write a script to concatenate text and numerical values in R .?**
   Text 1: Ram has scored
   Text 2: 89
   Text 3: marks
   Text 4: in Mathematics

   Concatenating Text Values in R Function

   Used:

   In R, the function used to concatenate text values is paste() or paste0().

   - paste() joins text values together with a space (default separator).
   - paste0() joins text values together without any separator.

   R Script to Concatenate Text and Numerical Values

   # Define the text and numeric values text1 <- "Ram has scored" text2 <- 89 text3 <- "marks" text4 <-

"in Mathematics from book"

Concatenate all using paste() result <- paste(text1,
text2, text3, text4)

Display the result print(result)

Output:
"Ram has scored 89 marks in Mathematics from book"

Explanation:
- The paste() function automatically converts numeric values (like 89) to text while concatenating.
- The default separator between arguments is a single space (" "), so the final sentence is formatted correctly.

8. List and discuss various types of data structures in R.

In R programming, data structures are used to store, manage, and manipulate data efficiently. Each data structure has a specific purpose and can hold different types of data such as numeric, character, or logical values. The various types of data structures in R are:

1. Vectors
- A vector is the simplest and most common data structure in R.
- It contains elements of the same data type (numeric, character, or logical).
- Created using the c() function. Example:

```
num <- c(10, 20, 30)
char <- c("Apple", "Banana", "Mango")
```

2. Lists
- A list is a collection of elements that can be of different data types — including numbers, strings, vectors, or even other lists.
- Created using the list() function.

Example:

```
my_list <- list(name="Rutvi", age=21, marks=c(89, 92, 95))
```

3. Matrices
- A matrix is a two-dimensional data structure that contains elements of the same data type arranged in rows and columns.
- Created using the matrix() function. Example:

```
mat <- matrix(1:9, nrow=3, ncol=3)
```

4. Arrays
- An array is similar to a matrix but can have more than two dimensions.
- Created using the array() function. Example: arr <- array(1:8, dim=c(2,2,2))

5. Data Frames
- A data frame is a two-dimensional table-like structure.
- It can store data of different types (numeric, character, factor) in each column.
- Created using the data.frame() function.
- Most commonly used data structure in data analysis. Example: df <- data.frame(Name=c("Ram","Shyam","Mohan"), Age=c(20,21,19),

Marks=c(85,90,88))

6. Factors
- A factor is used to store categorical data (data with limited distinct values).
- It is useful for statistical modeling and data classification.
- Created using the factor() function.

Example:
gender <- factor(c("Male", "Female", "Female", "Male"))

9. Discuss the syntax of defining a function in R with example.

A function is a set of statements organized together to perform a specific task. R allows users to define their own functions in addition to the built-in ones. Syntax of a Function
function_name <- function(arguments) { statements return(object)
}
Explanation of Components:
- function_name – the name of the function created by the user.
- arguments – a comma-separated list of inputs or parameters.
- statements – body of the function containing computations or operations.
- return(object) – specifies the value returned by the function. If not specified, the last evaluated expression is returned automatically.

Example 1: Function to Add Two Numbers
add <- function(a, b) { result <- a + b
return(result)
}

Calling the function add(10,
20) Output:
[1] 30
Explanation:
- The function add() takes two arguments a and b.
- It computes their sum and returns the result using the return() statement.
- The function is called by its name with appropriate parameters.

Example 2: Function Without Arguments
hello <- function() { print("Welcome to
R programming")
}

hello() Output:
[1] "Welcome to R programming" Explanation:
- The function hello() does not require any input parameters.
- It simply prints a message when called.

Built-in vs User-defined Functions Built-in Functions:
R provides many pre-defined functions for common operations.
Example:
sum(c(1, 2, 3, 4, 5)) Output:
[1] 15
- sum() is a built-in function that calculates the sum of its arguments.
- Built-in functions are ready to use and do not need to be defined by the user.

User-defined Functions:
Users can create their own functions to perform specific tasks.
Example:
add <- function(a, b) {
result <- a + b return(result)
}

add(10, 20) Output:
[1] 30

10. Discuss any five data visualizations techniques in R.

Data visualization is the graphical representation of data to help understand patterns, trends, and insights. R provides extensive libraries and functions for creating different types of visualizations. Below are five common techniques:

1. Bar Chart
- Purpose: To represent categorical data with rectangular bars; the length of the bar indicates the value or frequency. • Function in R: barplot() • Example: # Sample data sales <- c(100, 150, 80, 200) products <- c("A", "B", "C", "D")

# Bar chart
barplot(sales, names.arg = products, col = "blue", main = "Product Sales", ylab = "Sales")
Explanation:
- sales contains numeric values.
- names.arg assigns names to the bars.
- col specifies the color.

2. Histogram
- Purpose: To show the frequency distribution of numerical data by dividing it into intervals (bins).
- Function in R: hist()
- Example: # Sample data ages <- c(22, 25, 30, 22, 27, 30, 28, 25, 24)

Histogram hist(ages, breaks = 5, col = "green", main = "Age Distribution", xlab = "Age") Explanation:
- breaks defines the number of bins.
- xlab and main add labels and title.

3. Pie Chart
- Purpose: To represent proportions of a whole as slices of a circle.
- Function in R: pie()
- Example: # Sample data market_share <- c(40, 30, 20, 10) companies <- c("Apple", "Samsung", "Xiaomi", "Others")

Pie chart
pie(market_share, labels = companies, col = rainbow(4), main = "Market Share of Companies") Explanation:
- Each slice represents a percentage of the total.
- rainbow(4) assigns different colors to slices.

4. Scatter Plot
- Purpose: To visualize the relationship between two numerical variables.
- Function in R: plot()
- Example: # Sample data height <- c(150, 160, 165, 170, 175) weight <- c(50, 55, 60, 65, 70)

Scatter plot
plot(height, weight, main = "Height vs Weight", xlab = "Height (cm)", ylab = "Weight (kg)", pch = 19, col = "red") Explanation:
- Each point represents a pair of values (height, weight).
- pch controls the shape of points; col controls color.

5. Box Plot
- Purpose: To show the distribution of data and identify outliers using quartiles.
- Function in R: boxplot() • Example: # Sample data scores <- c(55, 60, 65, 70, 80, 90, 95, 100, 75, 85)

Box plot

boxplot(scores, main = "Exam Score Distribution", ylab = "Scores", col = "orange")
Explanation:

- The box shows the interquartile range (IQR), median, and possible outliers.