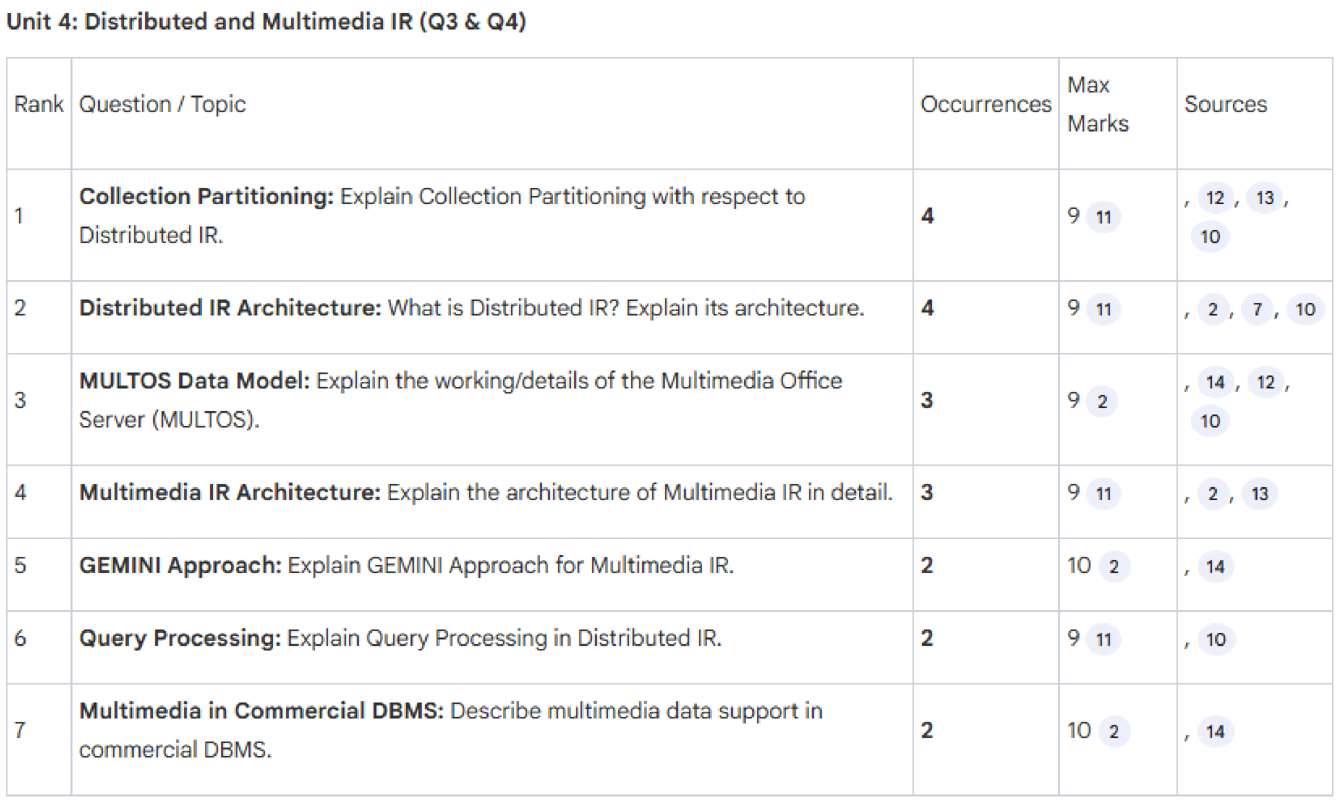
UNIT 4



Unit 4: Distributed and Multimedia IR (Q3 & Q4)

Rank	Question / Topic	Occurrences	Max Marks	Sources
1	**Collection Partitioning:** Explain Collection Partitioning with respect to Distributed IR.	**4**	9 11	, 12 , 13 , 10
2	**Distributed IR Architecture:** What is Distributed IR? Explain its architecture.	**4**	9 11	, 2 , 7 , 10
3	**MULTOS Data Model:** Explain the working/details of the Multimedia Office Server (MULTOS).	**3**	9 2	, 14 , 12 , 10
4	**Multimedia IR Architecture:** Explain the architecture of Multimedia IR in detail.	**3**	9 11	, 2 , 13
5	**GEMINI Approach:** Explain GEMINI Approach for Multimedia IR.	**2**	10 2	, 14
6	**Query Processing:** Explain Query Processing in Distributed IR.	**2**	9 11	, 10
7	**Multimedia in Commercial DBMS:** Describe multimedia data support in commercial DBMS.	**2**	10 2	, 14

q1) **What is distributed IR? Explain it with the help of Source Selection?**

Distributed Information Retrieval is a system architecture that allows a search engine to handle massive scales of data by spreading the storage and processing tasks across multiple computers (nodes) connected by a network.

MIMD Analogy: A distributed IR system is often viewed as a **MIMD (Multiple Instruction, Multiple Data)** parallel processor. Unlike tightly coupled parallel computers that share memory, distributed IR uses a **heterogeneous collection of processors** that communicate via relatively slow network protocols (like TCP/IP),.

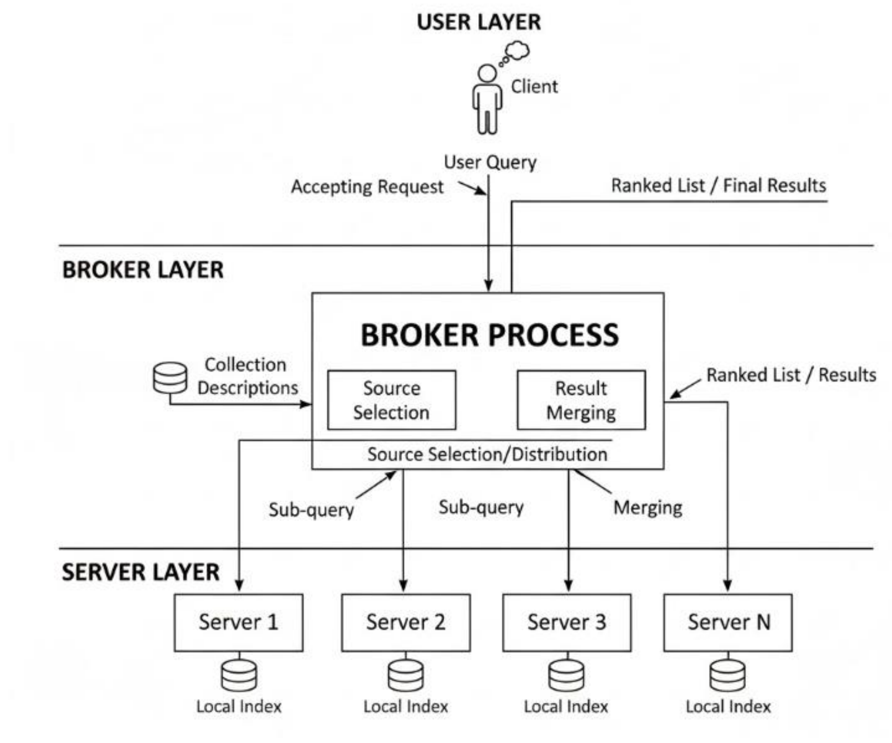
The Goal: To provide a single search interface to the user while the actual retrieval of documents is performed in parallel across many different servers, improving scalability and response time.

Architecture Component

To understand how it works, the system typically consists of two main types of processes,:

1. **Broker Process (Central Node):** This is the interface for the client. It is responsible for:
   a. Accepting the user's query.
   B. Deciding which servers to contact (**Source Selection**).
   C.Combining the returned results into a final list (**Result Merging**).
2. **Server Processes (Leaf Nodes):** These run on separate computers. Each server is responsible for:

a. maintaining an index for a specific portion of the document collection (**Indexing**).
b. Executing the search query on its local documents.



3 Main Algorithmic Challenges

1. **Collection Partitioning:** (How to distribute documents across servers?) - *You already have this*.

2. **Source Selection:** (Which servers to search?) - *You already have this*.

3. **Result Merging (Merging the Results):** (How to combine the lists?)

◦ *Explanation:* Since different servers return their own local rankings (e.g., Server A returns Top 10, Server B returns Top 10), the Broker must merge them into a single global ranked list.

◦ *Challenge:* A document ranked #1 on Server A might be less relevant than a document ranked #1 on Server B. The broker needs to normalize scores to merge them correctly

Explaining Distributed IR with Source Selection

Source Selection (

also known as Resource Selection) is the critical algorithmic decision making in Distributed IR that determines **which** of the many available document collections (servers) should be searched for a given query.

Why is Source Selection Necessary?

In a distributed system with hundreds or thousands of servers, broadcasting the user's query to **every single server** is inefficient. It causes:

- **High Network Traffic:** Sending queries to and receiving results from all servers clogs the network.
- **Wasted Resources:** Servers that do not contain relevant documents still have to process the query, wasting CPU cycles.

How Source Selection Works

To solve this, the Broker performs Source Selection to identify the subset of servers most likely to contain relevant documents. The process generally follows these steps:

1. **Collection Representation (The "Single Large Document" Approach):**

a. The Broker treats each distributed collection (server) as if it were a "single large document."

b. It creates a **Collection Vector** (or profile) for each server, which summarizes the vocabulary and term frequencies contained in that server's index.

2. **Ranking Collections:**

a. When a query arrives, the Broker compares the **Query Vector** against the **Collection Vectors** using similarity measures (like Cosine Similarity or probabilistic models).

b. This produces a **ranked list of collections**, ordered by how likely they are to contain relevant answers,.

3. **Selective Query Routing:**

a. The Broker selects only the top-ranked servers (e.g., the top 10% or top N collections).

b. The query is sent *only* to these selected servers, ignoring the rest.

Connection to Collection Partitioning

The effectiveness of Source Selection depends heavily on how the documents were distributed (Partitioned) in the first place,

- **Semantic Partitioning:** If documents are grouped by topic (e.g., Server A has "Sports," Server B has "Medicine"), Source Selection is very effective because the Broker can easily determine which server fits the query.
- **Random Partitioning:** If documents are distributed randomly for load balancing, Source Selection is difficult because relevant documents are scattered everywhere. In this case, the system often has to revert to broadcasting the query to all servers.

Q2) Explain Collection Partitioning with respect to Distributed IR.

In a Distributed IR system, the document collection is too large to be stored or indexed on a single machine. **Collection Partitioning** is the process of distributing the vast set of documents across multiple distinct processors or search servers (nodes) in the network,.

It answers the fundamental question: *"How do we decide which server stores which document?"*
There are two primary approaches to partitioning, each with distinct trade-offs between **efficiency** and **load balancing**.

1. Random Partitioning (Distributed Data Approach)

In this approach, documents are assigned to servers randomly or using a round-robin method without regard to their content.

How it works:

When new documents arrive, they are simply placed on the next available server or assigned a random ID that maps to a specific node.
Each server ends up with a statistically representative sample of the entire collection.

Implication for Search (Broadcasting):

Since relevant documents for a specific query (e.g., "cricket") are scattered randomly across *all* servers, the Broker cannot know which server has the answer.
Therefore, the Broker must **broadcast** the query to **every** server in the system.

Advantages:

Excellent Load Balancing: Since data is spread evenly, no single server becomes a "hotspot." The computational load and storage requirements are evenly distributed across the network

Disadvantages:

High Network Traffic: Sending every query to every server creates a bottleneck as the system scales. It is computationally expensive to search the entire network for every single request.

2. Semantic Partitioning (Source Selection Approach

In this approach, documents are grouped based on their content, topic, or source. This is often referred to as "Explicit Semantic Partitioning."

How it works:

Documents are clustered by subject area. For example, Server A stores "Sports," Server B stores "Medicine,"

and Server C stores "Politics".
This effectively creates specialized sub-collections.
Implication for Search (Selective Routing):
This partitioning enables **Source Selection**. When a user queries "World Cup," the Broker knows this relates to "Sports" and sends the query *only* to Server A.
Advantages:
Retrieval Efficiency: It dramatically reduces network traffic and server workload because only a few relevant servers process the query. It avoids the waste of searching irrelevant collections.
Disadvantages:
Load Imbalance: Popular topics can crash specific servers. If there is a major political election, Server C ("Politics") will be overwhelmed with traffic, while Server A ("Sports") sits idle. This defeats the parallel processing benefit of distributed systems.

3. Term Partitioning (Inverted Index Partitioning)

Although less common in distributed systems due to high communication overhead, the sources mention Term Partitioning as a theoretical alternative.
Concept: Instead of splitting *documents* (Document Partitioning), the system splits the *Inverted Index* itself.
Method: One server might hold the index for terms A-K, and another for L-Z.
Disadvantage: A single query typically contains multiple words (e.g., "Apple" and "Zoo"). To answer this, the Broker would need to contact multiple servers and transmit large lists of document IDs to perform the intersection (AND operation) across the network. This imposes heavy communication overhead.

Conclusion: Most modern Distributed IR systems struggle to find a balance. While **Semantic Partitioning** is smarter for retrieval, **Random Partitioning** is safer for system stability. Many systems use a hybrid or rely on **Source Selection** algorithms to try and cherry-pick the best collections even if they aren't perfectly semantically partitioned.

Q3) Explain in detail automatic feature extraction in Distributed IR

1. Introduction to Automatic Feature Extraction

In modern Information Retrieval, especially when dealing with distributed systems and multimedia data (images, audio, video), documents cannot be simply indexed by keywords. Instead, the system must automatically analyze the content to create a mathematical representation.

- **Definition:** Automatic feature extraction is the process of analyzing raw data (like an image or a time series) to generate a set of numerical values (a feature vector) that represents the object's essential characteristics.
- **The GEMINI Context:** This process is a fundamental step in the **GEMINI** (Generic Multimedia Object Indexing) approach. Feature extraction maps the data from a high-dimensional "original space" into a lower-dimensional "feature space" (f-dimensional point) to make indexing and searching efficient.

2. Types of Feature Extraction

The sources categorize feature extraction algorithms based on whether they use class labels:

1. **Supervised Feature Extraction:** Methods that utilize class labels of samples to guide the extraction process.
2. **Unsupervised Feature Extraction:** Methods that do not use class labels. These are the primary focus for general indexing tasks.

3. Automatic Feature Extraction Methods

The sources highlight two specific algorithms used for automatic feature extraction to handle high-dimensional data and reduce it to a manageable size: **Multidimensional Scaling (MDS)** and **FastMap**.

A. Multidimensional Scaling (MDS)

MDS is a nonlinear feature extraction technique designed to find a lower-dimensional representation of high-dimensional data.
Concept: *It attempts to arrange objects in a reduced space such that the distances between them in this new space correspond as closely as possible to the dissimilarities in the original space.*

Drawbacks: *The sources identify two major limitations of MDS for IR systems:*

- **Complexity:** It requires O(N^2) time, where N is the number of items. This makes it impractical for large distributed datasets.
- **Retrieval Issue:** It does not support "query-by-example" efficiently. If a user submits a new query object, MDS cannot easily map this new object into the existing k-dimensional space without re-calculating the entire map.

B. FastMap

FastMap is an algorithm designed to solve the limitations of MDS. It maps objects into points in a user-defined k-dimensional space while preserving dissimilarities.

- **Goals:**
  - It is a linear algorithm with **O(N)** complexity, making it much faster than MDS.
  - It supports fast indexing, allowing a new, arbitrary query object to be mapped into the k-d point with only O(k) distance calculations.
- **Algorithm Steps:**
  - **Pivot Selection:** The algorithm identifies two objects (pivot objects) that have the maximum distance between them among all documents.
  - **Projection:** It uses the **cosine law** to project all other objects onto the line formed by these two pivot objects.
  - **Recursion:** It calculates the distance function for the remaining dimensions and loops until the expected dimensional space is filled.

4. Example: Feature Extraction for Color Images

In distributed multimedia IR, automatic feature extraction is often applied to color images.

Color Histograms: A common feature is the color histogram, where the distribution of colors is analyzed.
The Cross-Talk Problem: A challenge in extracting color features is that standard histograms treat different colors (e.g., Orange and Red) as completely unrelated (orthogonal), even though they look similar to humans.
Solution (GEMINI Approach):

- The system calculates the **Average Color Vector** $\bar{x} = (R_{avg}, G_{avg}, B_{avg}) for an image.
- It extracts these numerical features to describe the image.
- To handle the similarity between colors (cross-talk), the system uses a weighted Euclidean distance or transforms the coordinate system so that the distance calculation accurately reflects human perception.

Q4) What is multimedia IR? Explain the architecture of multimedia IR in detail.

Multimedia Information Retrieval (MMIR) is a system associated with the storage, indexing, search, and delivery of multimedia data such as **images, videos, sounds, 3D graphics**, and their combinations.

- **Difference from Traditional IR:**
    - **Traditional IR:** Primarily deals with text (ASCII based). It typically uses keyword matching (exact match) and boolean logic.
    - **Multimedia IR:** Deals with **unstructured** and rich content. Searching for an image or a song cannot be done effectively using just text keywords. Instead, it relies on **Content-Based Retrieval**, where the system analyzes features like color, shape, texture, or audio frequency.
- **Similarity-Based Retrieval:** Unlike database queries (where a record either matches or doesn't), Multimedia IR is based on **similarity**. The system returns objects that are "closest" to the query (e.g., "Find images *similar* to this photo") using distance functions.

2. Architecture of Multimedia IR (Functional Components)

The architecture of a Multimedia IR system is designed to handle the complexity of extracting features from raw media files and comparing them. According to the sources, the architecture and retrieval process consist of the following key phases and components:

A. User Interface & Query Specification

The architecture begins with how the user interacts with the system. Unlike a simple text box, a Multimedia IR interface must support various query types:

1. **Attribute/Text-based Queries:** Using metadata like "Find songs by Artist X" or "Find videos from 2022."
2. **Content-based Predicates:** Queries based on the actual media content.
    a. *Example:* "Find all images containing a red car."
    b. *Query by Example (QBE):* The user provides an image and asks for similar images.
3. **Structural Predicates:** Queries based on the structure of the media.
    a. *Example:* "Find video clips where a scene changes to a specific frame".
4. **Fuzzy Predicates:** Handling imprecise queries like "Find *somewhat* red images".

B. Query Processing and Optimization

Once the query is received, the system must process it into a format the computer can understand.

- **Parsing:** The query is parsed and compiled into an internal form.
- **Feature Extraction (Query Side):** If the user submits an image as a query (QBE), the system must immediately extract its features (e.g., color histogram, texture vectors) to compare them against the database.
- **Optimization:** Since multimedia datasets are massive, the system optimizes the execution plan to reduce processing time.

C. Data Modeling and Storage (The Index)
This component handles how multimedia objects are stored and represented.

- **Feature Extraction (Indexing Side):** When new media is added to the database, the system automatically extracts "features" (numerical vectors representing color, shape, pitch, etc.).
- **Storage:** The raw media (BLOBs - Binary Large Objects) and their extracted feature vectors are stored. Commercial systems often use **Object-Relational DBMS** to extend standard relational tables to support these complex data types.
- **High-Dimensional Indexing:** To speed up searching, the system uses specialized indexing structures (like R-trees or GEMINI frameworks) to organize these high-dimensional feature vectors.

D. Searching / Matching Engine
This is the core computational engine.

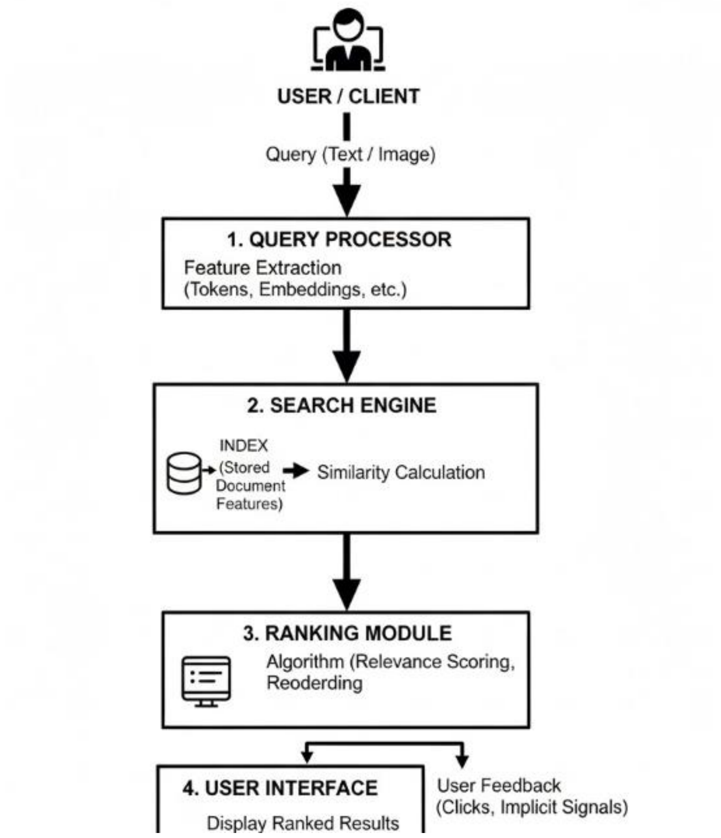
- **Similarity Comparison:** The engine compares the **Query Feature Vector** against the **Database Feature Vectors**.
- **Distance Function:** It calculates the mathematical distance (e.g., Euclidean distance) between the query and the stored objects. A smaller distance means higher similarity.
- **Ranking:** The system ranks the retrieved objects based on this similarity score (decreasing order of relevance).

E. Query Iteration (Relevance Feedback)
Multimedia retrieval is rarely perfect on the first try due to the "Semantic Gap" (the difference between low-level pixels and high-level human meaning).

- **Feedback Loop:** The system presents the initial results to the user.
- **Refinement:** The user marks items as "Relevant" or "Not Relevant." The system updates the query weights and re-runs the search to provide better results in the next iteration.

Summary of the Flow

1. **User** submits a query (text or image).
2. **Query Processor** extracts features from the input.
3. **Search Engine** calculates similarity between query features and stored **Index** features.
4. **Ranking Module** orders the results.
5. **Interface** displays results for **User Feedback**.



Q5) **What is Data Modeling in Multimedia IR? Explain it in detail.**

In the context of Multimedia Information Retrieval (MMIR), **Data Modeling** refers to the process of defining a structure or schema that allows users to specify the multimedia data to be stored, indexed, and retrieved by the system. Unlike traditional text-based IR, multimedia data is complex, unstructured, and high-dimensional.

1. Objectives of a Multimedia Data Model According to the sources, a robust Multimedia IR system must utilize a data model that is capable of:

• **Representation and Storage:** Efficiently storing multimedia objects (images, audio, video) in a way that ensures fast retrieval.

• **Handling Diversity:** Dealing with various media types and formats (e.g., .jpg, .mp3, .avi).

• **Semi-structured Data:** Handling data that doesn't fit neatly into rows and columns.

• **Feature Extraction:** Supporting the automatic extraction of features (like color histograms or texture vectors) from the raw media.

2. The Two Main Tasks of Data Modeling Addressing the data model in Multimedia IR involves two distinct tasks:

1. **User-Defined Model:** The system must provide a way for the user (or administrator) to specify the data to be stored. This implies integrated support for both the "conventional" data (metadata like Date, Author) and the "media" part (the actual video or image file).

2. **Internal Representation Model:** The system must provide a model for how it internally represents the multimedia data (e.g., converting an image into a multi-dimensional feature vector for similarity matching).

3. Integration of DBMS and IR Technologies Data modeling in MMIR is often viewed as a bridge between two technologies:

• **DBMS (Database Management Systems):** These excel at data modeling, concurrency control, and maintaining integrity for structured data.

• **IR (Information Retrieval):** These excel at similarity-based queries and handling unstructured content.

• **The Goal:** To combine the *data modeling capabilities* of DBMS with the *similarity-based query capabilities* of IR.

4. Problems and Challenges in Data Modeling The sources highlight several difficulties in modeling multimedia data:

• **Extending Relational Models:** The goal is to extend the standard Relational Model (tables) to represent complex data types (Abstract Data Types) while maintaining the simplicity of SQL.

• **Content Representation:** How is the "content" of an image represented? Is it described by a set of text attributes (metadata), or by extracted features (vectors)? The model must handle both.

• **Imprecision:** Feature extraction is not precise. A mathematical weight represents the uncertainty of a feature, which the data model must account for (unlike exact database values).

• **Standardization Issues:** While Object-Oriented DBMS (OODBMS) offer better storage techniques for objects, they lack the standardization of Relational DBMS. Therefore, most commercial systems rely on **Object-Relational** models.

---

q.6) Describe Multimedia Data Support in Commercial DBMS

To handle the requirements of multimedia data, commercial relational DBMS vendors have extended their systems to support **Object-Relational** features. This allows them to store large, unstructured binary data alongside traditional text and numbers.

1. Large Object (LOB) Support Most commercial DBMSs (like Oracle, Sybase, DB2) introduced specific data types to handle massive unstructured files:

• **BLOB (Binary Large Object):** Used to store binary data such as images, audio, and video clips directly in the database.

• **CLOB (Character Large Object):** Used to store large blocks of text (e.g., entire documents or XML files).

• **Oracle Specifics:**

◦ VARCHAR2: For variable-length character strings (limited size, typically 4000 bytes).

◦ LONG RAW: An older type for binary data.

◦ BFILE: Pointers to binary files stored *outside* the database (on the operating system's file system) but managed by the DB.

2. SQL3 / SQL:1999 Standard The SQL3 standard formalized the support for complex objects:

• **User Defined Types (UDTs):** Allows users to define their own complex structures (e.g., a Point type consisting of X and Y coordinates, or an Image type with attributes for height, width, and content).

• **User Defined Functions (UDFs):** Functions written to manipulate these new types (e.g., a function to Rotate(Image)).

• **Collection Types:** Support for sets, lists, and multisets within a database column.

3. Data Cartridges and Blades (Extensibility) To support specific media types efficiently, vendors introduced "plug-in" modules:

• **Oracle Data Cartridges:** These provide efficient and secure management of specific data types. For example, an **Image Cartridge** allows the database to understand image formats, perform conversions, and index image features (content-based retrieval).

• **Illustra (Informix) Data Blades:** These allow the modeling of complex data like **2D and 3D spatial objects**.

◦ They support spatial operators like INTERSECT, CONTAINS, and OVERLAPS.

◦ They implement specialized indexing structures like **R-trees** to represent and query spatial data efficiently (e.g., "Find all restaurants within 5km").

4. Sybase SQL Server Sybase supports IMAGE and TEXT data types to store respective media and provides a limited set of functions for searching and manipulating them.

Q7)

MULTOS (Multimedia Office Server) is a multimedia document server designed to provide advanced document retrieval capabilities. It was developed in the context of an **ESPRIT** project to serve the area of office systems.

- **Architecture:** It relies on a **Client/Server architecture**.
- **Purpose:** It allows for the storage, indexing, and retrieval of complex multimedia documents (containing text, images, and audio) based on their content and structure.

2. Types of Servers Supported

To manage different stages of a document's lifecycle efficiently, MULTOS supports three distinct types of document servers:

1. **Current Servers:** Used for active documents currently being worked on.
2. **Dynamic Servers:** Used for documents that change frequently.
3. **Archive Servers:** Used for the long-term storage of stable documents.

3. The MULTOS Data Model

The core of the MULTOS system is its data model, which allows it to understand the structure and content of office documents. The model supports three main classes of predicates or structures for describing a document:

A. Logical Structure

This defines the logical arrangement of the document's content.

- It organizes the document into components such as **titles, chapters, sections, and paragraphs**.
- This structure follows the **ODA (Open Document Architecture)** standard.

B. Layout Structure

This defines how the document is presented visually.

- It organizes the document into presentation elements such as **pages, frames, and blocks**.
- Like the logical structure, this is also defined according to ODA document representations.

C. Conceptual Structure (The Key Feature)

*This is the most critical aspect of MULTOS for Information Retrieval. It allows the definition of an **abstract level** of the document, describing its **semantic content**.*

- **Purpose:** It allows users to search for documents based on what they *are* (e.g., a "Business Letter" or an "Invoice") rather than just keywords.
- **Hierarchy:** Conceptual types are maintained in a **hierarchy of generalization**. Subtypes inherit from super-types.
- **Example - `Generic_Letter`:**
  - A document defined as a `Generic_Letter` might conceptually consist of components like **Sender**, **Receiver**, **Date**, and **Letter_Body**.
  - The `Letter_Body` might further break down into text and images.
  - Using this structure, a user can execute a precise query like: *"Find all letters where the Sender is 'Alice' and the Date is '1998'"*.

4. Spring Component Types

To handle the variability of multimedia documents (where one letter might look very different from another), MULTOS introduces the concept of **Spring Component Types** within the conceptual structure:

- **Strong Types:** These completely specify the structure of the instance. The document must strictly follow the defined model.
- **Weak Types (Spring Types):** These partially specify the structure. They allow for flexibility, where the exact internal structure (like the specific layout of a paragraph or image within a body) is left open or "unspecified" until the document is actually created.

Summary of Capabilities

The MULTOS system combines these structures to allow complex retrieval. It bridges the gap between **DBMS** (which handles structured data like "Date" and "Author") and **IR** (which handles unstructured content like the text body or images). Query processing in MULTOS involves transforming a query into executable plans that check both the attribute values (metadata) and the document structures.

----

q8) Explain GEMINI approach of Multimedia IR.

GEMINI is a framework designed to solve the performance issues associated with searching massive multimedia datasets. In multimedia IR, objects (like images or time series) are complex, high-dimensional, and require expensive distance computations (like Euclidean distance or cross-talk distance).

The core philosophy of GEMINI is to use a **"Quick-and-Dirty"** test to rapidly discard the vast majority of non-relevant items, leaving only a small set of candidates for precise comparison.

2. The Problem: Why is GEMINI needed?

Sequential scanning (checking every object one by one) is too slow for two reasons:

1. **Expensive Computation:** Calculating the distance/similarity between two complex objects (e.g., comparing two 256-color histograms) takes a lot of CPU time.
2. **Massive Database Size:** Checking millions of images sequentially is impractical.

3. The Solution: Three Key Steps of GEMINI

The GEMINI approach breaks the retrieval process down into specific steps to ensure efficiency without missing any relevant results.

Step 1: Define the Distance Function

First, the system must define a dissimilarity measure (Distance function) D(O1, O2) between two multimedia objects.

- If D(O1, O2) is small, the objects are similar.
- If a user asks for objects similar to Query Q within a tolerance ϵ, the system looks for all objects

  O where $D(O, Q) \leq \epsilon$ [1].

Step 2: Feature Extraction (Mapping to Lower Dimensions)

*To avoid comparing raw data, GEMINI maps the complex object into a **feature vector** (a point in a lower-dimensional space).*

- Let F be a feature extraction function.
- F(O) maps a high-dimensional object O into a point f in a k-dimensional space (where k is small).
- **Example:** Instead of comparing a whole time-series graph, we might extract just the average value or the first few coefficients of a Fourier Transform.

Step 3: The Lower Bounding Lemma (The Golden Rule)

*This is the most critical part of GEMINI. To ensure the "Quick-and-Dirty" test works, the distance in the feature space $D_{feature}$ must **never** be greater than the actual distance D_{actual} between the objects.*

The Condition:

$$D_{feature}(F(O_1), F(O_2)) \leq D_{actual}(O_1, O_2)$$

Why? This guarantees **No False Dismissals**.

- If the feature distance is large (larger than ϵ), the actual distance must also be large. Therefore, we can safely discard (prune) that object without checking the raw data.
- **False Alarms are allowed:** It is okay if the feature distance says "maybe relevant" but the actual object turns out to be irrelevant. These are filtered out in the final step.

4. The Retrieval Process using GEMINI

Based on the architecture described in the sources, the search process follows this flow:

1. **Map:** The query object Q is mapped to its feature vector F(Q).
2. **Search Feature Space:** The system uses a **Spatial Access Method (SAM)** (like an R-tree) to search the lower-dimensional feature space. It retrieves all points that are within distance ϵ of F(Q).
3. **Candidate List:** This search returns a set of candidate objects.
4. **Filter (Refinement):** For each candidate, the system retrieves the full, original object from the disk and calculates the **actual** distance $D_{actual}(Q, O)$.
5. **Final Result:** Objects that satisfy $D_{actual}(Q, O) <= \epsilon$ are returned to the user.

5. Examples of GEMINI in Action

A. 1D Time Series (e.g., Stock Prices)

- **Problem:** Comparing long sequences of stock prices.
- **Transformation:** Use **Discrete Fourier Transform (DFT)**.
- **Feature:** Keep only the first few coefficients (e.g., the first 2 or 3 frequencies) to represent the curve.
- **Lower Bounding: Parseval's Theorem** ensures that the Euclidean distance in the DFT feature space is preserved or lower-bounded, ensuring no false dismissals.

B. Color Images (Histograms)

- **Problem:** Comparing color histograms is expensive, especially with "Cross-talk" (where Orange and Red are considered similar).
- **Transformation:** Compute the **Average Color Vector** $(R_{avg}, G_{avg}, B_{avg})$ of the image.
- **Lower Bounding:** The distance between average colors is computationally cheap and mathematically lower-bounds the complex histogram distance.

Summary

The GEMINI approach allows Multimedia IR systems to index high-dimensional data efficiently. By mapping data to low-dimensional features and strictly adhering to the **Lower Bounding Lemma**, it achieves fast retrieval speeds while guaranteeing 100% recall (no missing results).

---

q9) Explain Query Processing in Distributed IR.

In a distributed IR system, query processing is the coordinated execution of a user's search request across multiple, physically separated server nodes. Unlike a centralized system where one machine does all the work, distributed processing involves a central **Broker** (or meta-searcher) that manages the flow of the query to various **Local Search Servers** and aggregates the results.

According to the sources, the query processing lifecycle consists of four distinct phases:

1. Collection Selection (Source Selection)

The first and most critical step is determining **where** to send the query.

- **The Problem:** In a massive system, broadcasting the query to **every** server is inefficient and causes high network traffic.
- **The Process:** The Broker analyzes the user's query and compares it against descriptions of the available collections (often using **Collection Vectors**).
- **Decision:** It selects a subset of servers that are most likely to contain relevant documents.
    - If documents are partitioned **semantically** (e.g., Sports vs. Medicine), the broker can select specific servers effectively.
    - If documents are partitioned **randomly**, the broker usually has to broadcast the query to all servers.

2. Query Distribution

Once the target servers are identified, the Broker distributes the query to them.

- **Mechanism:** The Broker acts as a client to the local servers, sending the query via network protocols (like TCP/IP).
- **Optimization:** This step may involve reformulating the query to match the specific query language or format required by the local servers if the system is heterogeneous (a meta-search engine scenario).

3. Local Query Execution

Each selected server processes the query independently, in parallel.
Local Indexing: The servers use their own local Inverted Indices to find matching documents.
Scoring: Each server calculates a **Relevance Score** (e.g., using TF-IDF or Cosine Similarity) for documents in its own collection.
Output: Each server produces a **local ranked list** of the top N relevant documents.

4. Result Merging (Data Fusion)

The final and most complex step is combining the partial results from different servers into a single, global ranked list to present to the user. The sources highlight two main approaches to this problem:

A. Round Robin Interleaving (Simple Approach)

Method: *The Broker takes the 1st document from Server A, then the 1st from Server B, then the 2nd from Server A, and so on.*
Drawback: *This is generally ineffective because a "relevant" document from a poor collection might be ranked higher than a "highly relevant" document from a good collection.*

B. Relevance Score Merging (Advanced Approach)

Method: *The Broker merges documents based on their actual similarity scores.*
The Challenge: *Scores from different servers may not be directly comparable. For example, a score of "0.8" on Server A might not mean the same as "0.8" on Server B because they calculate statistics (like Inverse Document Frequency - IDF) based only on their local data.*
Solution (Global Statistics): *To merge effectively, the Broker may need to distribute* ***Global Term Statistics*** *(global IDF) to the servers so they calculate standardized scores. Alternatively, the Broker can re-rank the results by applying weights based on the* ***quality of the collection*** *identified in Step 1.*

Summary of the Flow

1. **User** sends query to Broker.
2. **Broker** selects best Servers (**Source Selection**).
3. **Servers** execute query locally (**Local Processing**).
4. **Broker** gathers and combines lists (**Result Merging**).
5. **User** receives final result.
