1. Write a short note on emerging databases : [9]

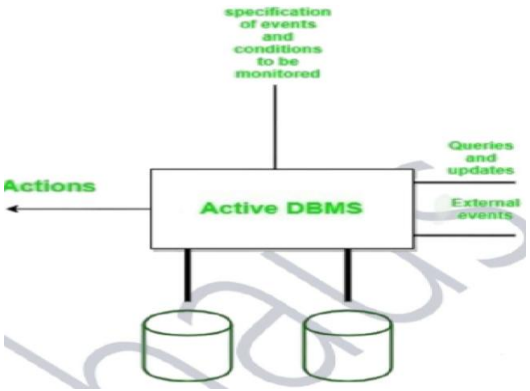
i) Active and Deductive Databases

ii) Main Memory Databases

1.1 Active Database

An **Active Database** is a database that can **automatically respond to certain events or conditions** without user involvement.

It uses **rules** that decide what action should occur when a specific event happens.
This makes the database **active** instead of passive.



Components of Active DBMS

- **Specification of events and conditions** → Defines what system should monitor.
- **Queries and updates** → Performed normally by users.
- **External events** → Inputs from outside systems.
- **Actions** → Triggered automatically by DBMS.
- **Active DBMS** → Monitors events and performs actions based on rules.

Example: Trigger in SQL

```
CREATE TRIGGER reorder_trigger

AFTER UPDATE ON products

FOR EACH ROW

WHEN (NEW.quantity < 10)

BEGIN

   INSERT INTO orders(product_id, order_date)
```

VALUES (NEW.product_id, SYSDATE);

END;

✓ Automatically generates an order when product quantity falls below 10.

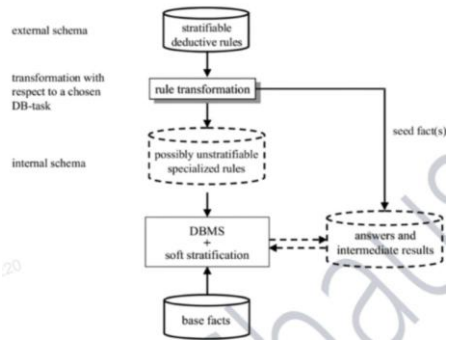
Applications

- Banking systems → fraud detection
- Inventory management → automatic reorder
- Security systems → unauthorized access alerts
- Workflow systems → automatic next-step execution

1.2 Deductive Database (DDB)

A **Deductive Database** integrates **traditional database features** with **logic-based reasoning**.

It can **derive new facts** from existing data using **rules and logical inference**, not just store data.
It can "**reason**" from known facts.



Key Components

1. Base Facts

Actual stored data (like relational tables).

2. Deductive Rules

Logical rules used to infer new facts.
Example:

grandparent(X, Y) :- parent(X, Z), parent(Z, Y).

3. Inference Engine

Applies rules to base facts to derive new information.

Example

Given:

parent(A, B)

parent(B, C)

Rule:

grandparent(X, Y) :- parent(X, Z), parent(Z, Y).

Deductive database deduces:

grandparent(A, C)

→ Even though it is **not explicitly stored**, it is **derived logically**.

Advantages of Deductive Databases

- Intelligent querying through logical inference.
- Supports complex relationships and recursive rules.
- Reduces data redundancy by deriving new facts instead of storing everything.
- Useful for expert systems, AI, and knowledge-based systems.

Disadvantages of Deductive Databases

- Complex query processing because of logical inference.
- Performance overhead for large datasets.
- Difficult debugging of recursive or dependent rules.
- Requires specialized query languages like *Datalog* (not SQL).
- Has limited support in traditional DBMS systems.

Applications of Deductive Databases

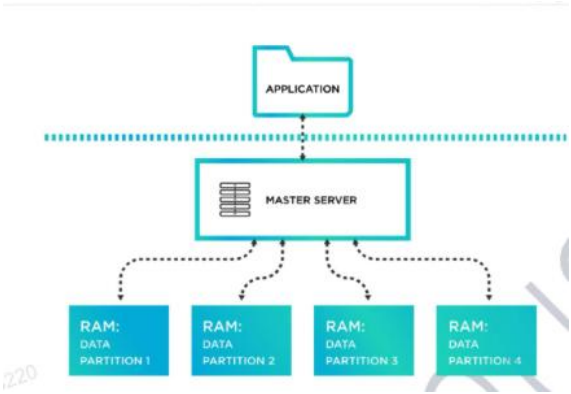
1. **Artificial Intelligence (AI) Systems**
2. **Expert Systems**
3. **Knowledge-Based Systems**
4. **Medical and Scientific Research Systems**
5. **Data Mining and Pattern Recognition**

Main Memory Databases

A Main Memory Database (MMDB), also known as an In-Memory Database, is a database system that stores all data in the main memory (RAM) instead of disk storage.

Because accessing RAM is much faster than accessing disks, MMDBs provide extremely high-speed data processing and real-time performance.

Working



- **Data Storage in RAM:**
  The entire database is divided into data partitions, and each partition is stored directly in the system's main memory (RAM) instead of hard drives
- **Master Server**: The master server manages these data partitions, coordinates transactions, and ensures that data remains consistent across memory segments.
- **Application Interaction:**
  The application interacts with the master server for queries, updates, and transactions. Since data is stored in memory, responses are much faster compared to disk-based systems.
- **Persistence & Backup:**
  Though data resides in RAM, the system periodically creates snapshots or logs on disk to ensure data recovery in case of power failure or crash.

Applications

- Real-time analytics systems (stock trading, IoT).
- Telecommunication systems (fast call processing).
- Gaming and caching servers.
- E-commerce sites (handling rapid transactions and user sessions).

2. Write a short note on complex data types: [9]

i) Semi-structured data

ii) Features of semi-structured data models

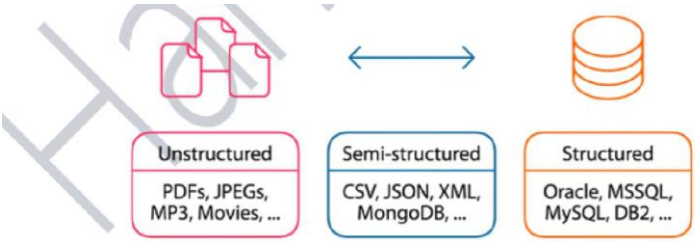
Semi-Structured Data

Semi-structured data is data that **does not follow a strict fixed schema** like relational databases, but still contains **tags, keys, or markers** to organize the data.

It lies **between structured and unstructured data**.

Examples:

- **XML, JSON, MongoDB documents**



Type	Examples	Description
Structured Data	Oracle, MySQL, DB2	Organized in fixed rows and columns (tables) with predefined schema.

Characteristics of Semi-Structured Data

- **Flexible schema:** Structure can vary from one record to another.
- **Self-describing:** Data contains metadata or tags to describe itself.
- **Hierarchical or nested representation:** Data is stored in tree-like form (e.g., XML, JSON).
- **Easily extendable:** New fields can be added without altering existing data.
- **Common in web and NoSQL databases.**

📌 **Applications**

- **Web data exchange** (JSON, XML APIs)
- **NoSQL databases** (MongoDB, Cassandra)
- **E-commerce and social media data**

- **Big Data analytics**
- **REST APIs and mobile apps**

---

2)Features of Semi-Structured Data Models

1. **Flexible Schema (Schema-less Nature)**
    - No fixed schema is required (unlike relational databases).
    - Each record can have a different structure.
    - *Example:* One student record may have an “email” field, another may not.
2. **Self-Describing Structure**
    - Data carries its own description using tags or key-value pairs.
    - *Example (XML):*
    - <student name="Tanish" cgpa="9.86" />
3. **Hierarchical / Graph Representation**
    - Data can be stored in tree-like or graph-like structures.
    - Supports nested relationships.
    - *Example (JSON):*
    - {
    - "name": "Tanish",
    - "subjects": ["DBMS", "CG", "TOC"]
    - }
4. **Support for Irregular & Heterogeneous Data**
    - Records may vary in the number or type of attributes.
    - Ideal for multimedia, documents, and web data where uniformity isn’t guaranteed.
5. **Extensibility**
    - New fields/attributes can be added without affecting existing records.
    - *Example:* Adding "email": "tanish@gmail.com" to a JSON record doesn’t affect other records.
6. **Data Exchange & Interoperability**
    - Commonly used for exchanging data between heterogeneous systems.

- XML and JSON are widely used in APIs and web services.

7. **Human-Readable and Machine-Readable**
    - Tags/keys make it easy for both humans and programs to understand the data.
8. **Support for Semi-Structured Query Languages**
    - Uses specialized query tools such as:
        - **XQuery / XPath** for XML
        - **JSONPath** or MongoDB Query Language for JSON

---

3)What is object relational database? What are its advantages and disadvantages?

4.1 Object–Relational Database System (ORDBMS)

Definition

An **Object–Relational Database System (ORDBMS)** is a database system that **combines features of both Relational Databases (RDBMS)** and **Object-Oriented Databases (OODBMS)**.

It enhances the traditional relational model by supporting **objects, classes, methods, inheritance**, and **user-defined types**, enabling storage of **complex and real-world data** such as multimedia, images, spatial data, etc.

Purpose

The main goal of ORDBMS is to **bridge the gap between object-oriented programming languages** (Java, C++, Python) **and relational databases**, allowing developers to work with complex data in a natural and efficient way.

Features of Object–Relational Database System

1. Support for Complex Data Types

Can store multimedia (images, audio), spatial data, and user-defined objects.

2. User-Defined Types (UDTs)

Allows creation of custom data types for special structures.

3. Inheritance

New types can be derived from existing types (similar to OOP).

4. Encapsulation

Data (attributes) and behavior (methods) can be stored together.

5. Polymorphism

Methods can behave differently depending on data type.

6. Extensible SQL (Object SQL)

Extends SQL to support objects and methods.

7. Backward Compatibility

Still supports relational tables → easy migration from traditional RDBMS.

Examples (SQL/Object SQL Syntax)

Step 1: Create an Object Type

```
CREATE TYPE Student AS OBJECT (
   rollno NUMBER,
   name VARCHAR2(30),
   course VARCHAR2(30)
);
```

Step 2: Create a Table Using the Object Type

```
CREATE TABLE Student_Table OF Student;
```

Step 3: Insert a Row (Object) into the Table

```
INSERT INTO Student_Table
VALUES (Student(101, 'Ayushi', 'Computer Science'));
```

Step 4: Query the Table

```
SELECT name, course FROM Student_Table;
```

Advantages of ORDBMS

- Supports **complex and multimedia data**.
- Backward compatible with traditional SQL + RDBMS.
- Bridges the gap between **object-oriented programming** and **relational storage**.
- Reusability through inheritance and user-defined types.
- Suitable for modern applications (CAD, GIS, multimedia, scientific databases).

Disadvantages

- **Complex implementation** compared to pure relational databases.
- **Performance overhead** due to object handling.
- **Increased storage requirements** for complex objects.
- **Limited standardization** across vendors (not all ORDBMS behave the same).

---

4) Describe the significance of JSON data type and object. Discuss with syntax all JSON data types with suitable example.

JSON – JavaScript Object Notation

Significance of JSON

1. **Lightweight & Fast** – Uses simple syntax, making data transmission quick.
2. **Language Independent** – Supported by almost all languages (Java, Python, C++, etc.).
3. **Human-readable & self-describing** – Uses key–value pairs.
4. **Hierarchical representation** – Supports nesting of objects and arrays.
5. **Ideal for APIs and NoSQL databases** – REST APIs and databases like MongoDB use JSON.

✅ **JSON Data Types**

JSON supports **six basic types**:

1. **Object** – Collection of key–value pairs
2. **Array** – Ordered list of values
3. **String** – Text data
4. **Number** – Integer or floating-point
5. **Boolean** – true/false
6. **Null** – Represents empty value

✅ **Example (All Data Types Combined)**

```
{
  "student_id": 101,
```

```
 "name": "Tanish",
 "cgpa": 9.86,
 "isActive": true,
 "subjects": ["DBMS", "CG", "TOC"],
 "address": { "city": "Pune", "pin": 411001 },
 "graduationDate": null
}
```

✅ **Advantages of JSON**

- Fast and efficient for data transfer.
- Widely used in APIs and web services.
- Human-readable and easy to debug.
- Easily convertible to objects in programming languages.
- Compact format for real-time communication.

✅ **Disadvantages of JSON**

- **Limited data types** – Cannot store functions or complex objects directly.
- **No schema enforcement** – May lead to inconsistent records.
- **Parsing errors** – Missing comma or bracket breaks the structure.
- **Less secure** if not validated properly.

✅ **Applications of JSON**

- **Web APIs** (client ↔ server communication)
- **NoSQL databases** like MongoDB, Couchbase
- **Configuration files** (package.json in Node.js)
- **Data serialization and data exchange**

---

5)Explain how encoding and decoding of JSON object is done in JAVA with example

1. JSON Object Encoding in Java

Encoding means converting **Java data → JSON format**.

We use the JSONObject class from **org.json.simple** to create and print data in JSON format.

Example:

```
import org.json.simple.JSONObject;

public class EncodeExample {
   public static void main(String[] args) {
      JSONObject obj = new JSONObject();
      obj.put("name", "Ayushi");
      obj.put("age", 20);
          System.out.println(obj);   // JSON output
           }
          }
```

Output

```
{"name":"Ayushi","age":20}
```

- JSONObject stores data in key–value pairs.
- .put() is used to insert data.
- Printing the object displays data in proper JSON format — this is encoding.

---

2. JSON Object Decoding in Java

Decoding means converting **JSON format → Java data types**.
We use the **JSONValue** class to parse the JSON string and extract values.

Example:

```
import org.json.simple.JSONObject;

import org.json.simple.JSONValue;

  public class DecodeExample {

  public static void main(String[] args) {

      String jsonData = "{\"name\":\"Ayushi\", \"age\":20}";
```

```
    JSONObject obj = (JSONObject) JSONValue.parse(jsonData);
     String name = (String) obj.get("name");
      long age = (Long) obj.get("age");
          System.out.println(name + " " + age);
  }
}
```

Output:

Ayushi 20

Explanation:

- JSONValue.parse() reads and converts the JSON string into a JSONObject.
- .get() retrieves each value by key.
- The data is then converted into Java types (String, Double, Long).

6)What is the significance of XML databases? Explain with example the use of XML databases.

XML Databases

An **XML Database** is a type of database designed to store, manage, and query data in **XML format**.
It can either store entire XML documents or data structured as XML elements and attributes.

XML databases are widely used when the data is **hierarchical, document-oriented, or semi-structured**.

Significance of XML Databases

XML databases are significant because they combine the **flexibility of XML** with the **efficiency of database management systems (DBMS)**.

---

1. Handles Semi-Structured and Hierarchical Data

- XML databases can store data that doesn't fit neatly into tables (like relational databases).
- Useful when data varies in structure (e.g., different records having different fields).

---

2. Platform and Language Independent

- XML is a **standardized format**, making XML databases ideal for data exchange between heterogeneous systems (e.g., Java ↔ .NET ↔ Python).

---

3. Data Sharing and Web Integration

- XML databases are often used in **web services (SOAP, REST)** and business applications where data is exchanged in XML format.

---

4. Self-Descriptive and Readable

- The data is stored along with its tags, making it **self-descriptive and easy to interpret**.

---

5. Integration with Existing Databases

- Relational databases can store XML data types (e.g., XMLType in Oracle, FOR XML in SQL Server).
- Allows structured and semi-structured data to coexist.

When to Use XML Database:

Suppose a university application stores student data, teacher data, and research reports —

- Each document has **different structures** (students, research, departments).
- The system must **exchange data** with external services in **XML format** (like educational portals).
  In this case, an XML database is ideal because:
    - It can handle **different structures** easily.
    - It allows **data exchange** between multiple systems.
    - Data can be **queried using XQuery or XPath** directly.

Example of Querying XML Data

Using XQuery:

```
for $s in /Students/Student

where $s/Course = "Computer Engineering"
```

return $s/Stu_Name

Output:

Tanish

---

7)Diffrence between relational databases and object relational databases with example

Feature	Relational Database (RDBMS)	Object–Relational Database (ORDBMS)
Data Representation	Data is stored in **tables** with rows and columns.	Data is stored in **tables + objects** (supports both).
Data Type Support	Supports only **primitive types** (INT, CHAR, FLOAT, etc.).	Supports **user-defined types (UDTs)**, arrays, multimedia, and complex data.
Relationship Handling	Relationships handled through **foreign keys and joins**.	Supports **inheritance** and object relationships directly.
Programming Integration	Does **not integrate** directly with object-oriented languages.	Integrates with **object-oriented programming concepts** (like Java or C++).
Methods (Functions)	Only supports **stored procedures and SQL functions**.	Supports **methods and functions inside object types** (encapsulation).
Schema Flexibility	Schema is **rigid and predefined**.	Schema is **extensible** — can add new data types and methods.
Query Language	Uses **standard SQL**.	Uses **Object SQL (SQL:1999 / SQL:2003)** with object extensions.
Inheritance	**Not supported.**	Supported — tables (or object types) can **inherit** from parent tables.
Use Case	Best for **traditional business applications** with structured data.	Best for **modern applications** handling complex, multimedia, or scientific data.
Examples	MySQL, SQL Server, DB2 (basic mode).	Oracle (Object extensions), PostgreSQL, Informix, IBM DB2 (object-relational).

1. RDBMS Example (Traditional SQL)

SQL Code (RDBMS):

```
CREATE TABLE Student (
   rollNo INT,
   name VARCHAR(50),
 course VARCHAR(50)
);
INSERT INTO Student VALUES (101, 'Tanish', 'DBMS');
SELECT * FROM Student;
```

---

2. ORDBMS Example (Object-Relational Database System)

Step 1 — Create Object Type

```
CREATE TYPE Student AS OBJECT (
   rollNo INT,
   name VARCHAR(50),
   course VARCHAR(50),
   MEMBER FUNCTION display RETURN VARCHAR
);
```

Step 2 — Create Table Using Object Type

```
CREATE TABLE Student_Table OF Student;
```

Step 3 — Insert Object in Table

```
INSERT INTO Student_Table
VALUES (Student(101, 'Tanish', 'DBMS'));
```

8)Explain table inheritance

4.2 Table Inheritance

Table Inheritance is a feature of **Object–Relational Database Systems (ORDBMS)** that allows a table (child) to inherit the structure and properties of another table (parent).

It is similar to **class inheritance in Object-Oriented Programming (OOP)**, where the child table can reuse the parent's attributes and also define its own additional attributes.

- **The child table** automatically contains all **columns (attributes)** of the parent table.

- The child can also **add extra columns** specific to its own purpose.
- **Queries on the parent table** can automatically include data from all its child tables (if desired).

Example

Parent Table:

```
CREATE TABLE Employee (
   emp_id      INT,
   name        VARCHAR(50),
   salary      FLOAT
);
```

Child Table (inherits Employee):

```
CREATE TABLE Manager (
   department VARCHAR(50)
) INHERITS (Employee);
```

Insert Data:

```
INSERT INTO Employee VALUES (101, 'Tanish', 45000);


INSERT INTO Manager VALUES (102, 'Riya', 60000, 'HR');
```

Query:

```
SELECT * FROM Employee;
```

Output:

emp_id name salary department

101 Tanish 45000 null

102 Riya 60000 HR

9)What are spatial data. Explain Geographic and Geometric data.

5. Spatial Data

Spatial Data refers to data that represents **objects or information about the physical location, shape, and relationship** of features in space (geographical or geometric). It describes **where things are** on Earth or in a specific coordinate system.

Explanation

In databases, **spatial data** is used to store and process **geographic and geometric information** such as maps, roads, buildings, rivers, and city locations.

Spatial data defines real-world objects in terms of their:

- **Position (location)** — where it is.
- **Shape (geometry)** — what it looks like.
- **Relationship** — how it connects to or interacts with other spatial objects.

These data are stored using **coordinates** (latitude, longitude, or x-y-z values) and represented through geometric models.

Examples

- **PostGIS** (extension of PostgreSQL)
- **Oracle Spatial**
- **MySQL Spatial Extensions**

.1 Geographic Data

Geographic Data (also called Geospatial Data) refers to data that describes the location and shape of natural or man-made features on Earth's surface, along with their spatial relationships.

It is used to represent real-world geographic objects such as cities, roads, rivers, buildings, and land areas — using latitude, longitude, and elevation coordinates.

Components of Geographic Data

1. **Spatial Component (Location):**
   Describes where an object is located using coordinates.
   Example: Pune → (18.5204° N, 73.8567° E)
2. **Attribute Component (Description):**
   Describes what the object is.
   Example: Pune → City, Population → 7 million

3. **Topological Component (Relationship):**
   Describes how different objects are related spatially.
   Example: Pune is connected to Mumbai by NH-60.

Representation in Databases

Modern databases (like PostGIS, Oracle Spatial, MySQL Spatial) support geographic data using special data types:

Type	Meaning	Example
ST_Point	A single location (x, y) or (lat, long)	(18.5204, 73.8567)
ST_LineString	A line connecting multiple points	Road or river path
ST_Polygon	A closed area	City boundary or region

Geographic Query Example:

SELECT name

FROM City

WHERE ST_Distance(location, ST_Point(18.5204,73.8567)) < 50000;

Finds all cities **within 50 km** of a given location.

Applications

• Mapping and navigation systems (Google Maps, GPS)
• Transportation and route optimization
• Agriculture and land use planning
• Disaster management and environmental monitoring
• Urban planning and infrastructure development

5.2 Geometric Data

Geometric Data refers to data that represents the shape, size, and position of objects in a geometric space (usually a 2D or 3D coordinate system).

It is used to model points, lines, curves, and polygons that define the structure and spatial relationships of objects — such as buildings, roads, and boundaries — in computer graphics, CAD, and spatial databases.

Geometric Data Types (in Databases)

Data Type	Description
ST_Point	Represents a single coordinate point.
ST_LineString	Represents a sequence of points forming a line.
ST_Polygon	Represents an enclosed area (like boundary or region).
ST_Circle, ST_MultiPolygon, etc.	Represent complex geometric shapes.

Example (in SQL)

```
CREATE TABLE Shapes (
  id INT,
  shape_name VARCHAR(30),
  shape_data GEOMETRY
);

INSERT INTO Shapes VALUES
(1, 'Triangle', ST_Polygon('((1, 5, 1, 3, 4, 1))')),
(2, 'Line', ST_LineString('((2, 3, 5, 7))')),
(3, 'Point', ST_Point('(4 6)'));
```

Here, geometric objects are stored directly in the database using the GEOMETRY data type.

Applications

- Computer-Aided Design (CAD)
- Spatial Databases
- Computer Graphics and Animation
- Simulation and Robotics
- Architecture and Construction Design

Difference between Geometric and Geographic Data:

Feature	Geometric Data	Geographic Data
Definition	Represents abstract shapes in a 2D/3D coordinate system.	Represents real-world objects on Earth.
Coordinates	Cartesian (x,y,z).	Latitude, Longitude, Elevation.
Use Case	CAD, computer graphics, design systems.	GIS, maps, navigation.
Example	A building floor plan (in meters).	A city map (in degrees).