**ADBMS – MODULE 4**

**OBJECT-ORIENTED DATABASE OR OBJECT DATABASE:** It is a type of database in which the data is organized into objects. These objects can have relationships with other objects, similar to the way that objects in a class can have relationships with other objects in that class. This allows for more complex data structures and makes it possible to model relationships between objects in the database.

**COMPONENTS OF AN OBJECT-ORIENTED DATABASE:**

**Object**: This is the basic unit of data in an Object-oriented Database. An object has a unique identifier and stores its data in fields.

**Class:** A class is a template for creating objects. It defines the structure of an object and the methods that can be used to manipulate its data.

**Method:** A method is a function that is associated with a class. It can be used to manipulate the data in the fields of an object.

**Collection:** A collection is a container for objects. It allows you to store and manipulate a group of objects as a single unit.

**Design:** Object-oriented Database is a design pattern. It allows you to structure your data in a way that is consistent with the principles of object-oriented programming.

**NEED OF AN OBJECT-ORIENTED DATABASE:**

* Complex Data Structures:

Use of Objects: If the data in your application can be naturally modeled as objects with attributes and methods, an OODB can provide a more intuitive and efficient way to represent and manipulate that data.

* Object-Oriented Programming Integration:

Seamless Integration: If your application is developed using an object-oriented programming language, using an OODB can provide seamless integration between the programming language and the database, reducing the impedance mismatch between the application code and the database.

* Inheritance and Polymorphism:

Hierarchical Data Model: If your data has a hierarchical structure and involves concepts like inheritance and polymorphism, OODBs can represent these relationships more naturally than relational databases, making it easier to model and manage complex data structures.

* Data Modeling Flexibility:

Dynamic Schema: OODBs often support a dynamic schema, allowing for changes in the data model without significant impact on the application. This flexibility is particularly useful in scenarios where the data model is expected to evolve over time.

* Improved Performance:

Reduced Join Operations: In certain scenarios, OODBs can offer improved performance over relational databases, especially when dealing with complex, nested structures. Retrieving an entire object and its associated data can be more efficient than performing multiple joins.

* Encapsulation and Code Reusability:

Encapsulation: OODBs support encapsulation, allowing for better code organization and modularity. Data and the operations on that data are encapsulated within objects, promoting code reusability and maintainability.

* Application Development Speed:

Rapid Prototyping: For applications that require rapid prototyping and development, OODBs can be advantageous as they allow developers to work with a data model that closely mirrors the application's object-oriented design.

* Complex Relationships:

Graph Relationships: If your data has complex graph-like relationships (e.g., social networks, network topologies), OODBs or graph databases might be more suitable than traditional relational databases.

**IMPEDENCE MISMATCH (PROBLEM BETWEEN OO and RELATIONAL MODEL):**

* In the context of databases, impedance mismatch refers to the discrepancy between the object-oriented programming (OOP) model used in application code and the relational model used in database management systems (DBMS).
* While OOP models are designed to represent data as objects with properties and methods, relational models represent data as tables with columns and rows.
* This impedance mismatch can create challenges when it comes to mapping objects in code to tables in a database or vice versa.
* To address the impedance mismatch problem, developers often use object-relational mapping (ORM) tools or other middleware solutions that provide a bridge between the OOP model and the relational model.
* These tools can automate the mapping process, allowing developers to work with objects in code while transparently interacting with the underlying database
* For example, an object hierarchy in code may need to be flattened into a single table for storage in a database, or multiple related tables may need to be joined together to represent a single object in code. These conversions can be complex, and can lead to performance issues, data inconsistency, and increased development time and costs.
* Problems:
  + The first problem that may occur is that is data type mismatch means the programming language attribute data type may differ from the attribute data type in the data model. Hence it is quite necessary to have a binding for each host programming language that specifies for each attribute type the compatible programming language types.
  + The second problem that may occur is because the results of most queries are sets or multisets of tuples and each tuple is formed of a sequence of attribute values. In the program, it is necessary to access the individual data values within individual tuples for printing or processing. Hence there is a need for binding to map the query result data structure which is a table to an appropriate data structure in the programming language. A mechanism is needed to loop over the tuples in a query result in order to access a single tuple at a time and to extract individual values from the tuple. The extracted values are typically copied to appropriate program variables for further processing by the program. A cursor or iterator is a variable which is used for looping over the tuples in a query result. Individual values within each tuple are extracted into different or unique program variables of the appropriate datatype.
* Advantages:
  + Increased flexibility: Impedance mismatch allows for more flexibility in the way data is stored and manipulated. Applications can use their own data structures, which may be better suited to their specific needs than the database schema.
  + Better performance: When applications use their own data structures, it can be easier to optimize queries and improve performance. This is because the data structures can be tailored to the specific queries being performed.
  + Easier development: Developers can work with familiar data structures and APIs, which can make development easier and faster.
* Disadvantages:
  + Increased complexity: Impedance mismatch can add complexity to the development process, as developers must manage the mapping between the application data structures and the database schema.
  + Higher risk of errors: Mapping data between different structures can increase the risk of errors and data inconsistencies, particularly if the mapping is not well-designed or well-implemented.
  + Limited functionality: Impedance mismatch can limit the functionality of the DBMS, as applications may not be able to take advantage of all of the features provided by the DBMS. This can also lead to increased maintenance costs and slower development cycles.

**DB4O:**

* db4o (database for objects) is an open-source object-oriented database management system (OODBMS) that is designed to work seamlessly with Java and .NET (C#) programming languages.
* It provides a way to persistently store and retrieve objects directly, eliminating the need for an intermediary mapping layer between the object-oriented code and the database.
* **Object-Oriented Approach:** db4o follows an object-oriented paradigm, treating data as objects with attributes and behaviors. This aligns well with the way data is often represented in object-oriented programming languages.
* **Java and .NET Compatibility:** It is specifically designed to be used with Java and .NET environments, offering native support for these programming languages.
* **Embeddable Database:** db4o can be embedded within an application, allowing developers to use it without the need for a separate database server. This makes it suitable for applications with a need for simplicity and lightweight data storage.
* **Transparent Persistence:** Objects in db4o are made persistent transparently, meaning developers can work with objects in their code without needing to perform explicit conversion to and from a database-friendly format.
* **Dynamic Schema:** The database supports a dynamic schema, enabling developers to modify the structure of stored objects without requiring a predefined schema or complex migrations.
* **Query-by-Example (QBE):** db4o supports a Query-by-Example approach, allowing developers to express queries by providing an example object with fields set to the desired values.
* **Simplicity and Ease of Use:** db4o is known for its simplicity and ease of use, making it suitable for projects where a straightforward and uncomplicated database solution is desired.
* **Open Source and Community Support:** Being an open-source project, db4o allows developers to access the source code, contribute to its development, and benefit from community support.
* **Portability:** db4o databases are portable, meaning that the database file can be moved between different environments without facing compatibility issues.

**DOCUMENT-ORIENTED DATABASE:**

* A document database, also known as a document-oriented database, is a type of NoSQL database that stores data in the form of documents, rather than in tables with rows and columns like a traditional relational database.
* These documents can be in a variety of formats, such as JSON, BSON, or XML.
* They often include nested data structures, which make it easier to store and query complex data.
* Document databases are well suited for storing semi-structured data and are often used in web and mobile applications.
* An example of a document data model could be a collection of “users” where each document represents a single user and contains fields such as name, email, address, and preferences. An example of a document for a user could be:

|  |  |
| --- | --- |
| {  "\_id": "12345",  "name": "John Smith",  "email": "john.smith@example.com",  "address": {  "street": "123 Main St",  "city": "Anytown",  "state": "NY",  "zip": "12345"  }, | "preferences": {  "language": "en",  "notifications": true  },  "orders": [  { "order\_id": "54321", "total": 100 },  { "order\_id": "67890", "total": 50 }  ]  } |

NEED OF DOCUMENT\_ORIENTED DATABASE:

1. Flexible Schema: Documents in a document-oriented database can have different structures, allowing for flexibility in data representation.
2. Semi-Structured or Unstructured Data: These databases are suitable for data that doesn't fit neatly into traditional tabular structures, such as JSON or XML documents.
3. Complex Nested Structures: Document-oriented databases support hierarchical and nested structures, making them effective for representing relationships in data.
4. Fast and Scalable Reads: Retrieving specific documents or subsets is efficient, making them well-suited for read-heavy workloads.
5. Agile Development: These databases easily adapt to changes in data structures, aligning with iterative and agile development processes.
6. Horizontal Scalability: They can scale horizontally by distributing data across multiple servers, accommodating large volumes of data.
7. Developer Productivity: Aligning with developers' preferences, especially when working with languages like JavaScript or Python that use JSON-like structures.
8. Support for Indexing: Document-oriented databases often provide indexing mechanisms, allowing for fast and optimized queries on specific fields within documents.

* NOTE: Popular examples of document-oriented databases include **MongoDB**, **CouchDB**, and **Elasticsearch**.

**DIFFERENCE BETWEEN RELATIONAL AND DOCUMENT MODEL:**

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Relational Model** | **Document Model** |
| 1. | It is row-based. | It is document-based. |
| 2. | Not suitable for hierarchical data storage. | Generally used for hierarchical data storage. |
| 3. | It consists of a predefined schema. | It consists a dynamic schema. |
| 4. | ACID properties are followed by this model. (Atomicity, Consistency, Isolation, and Durability). | CAP theorem are followed by this model. (Consistency, Availability, and Partition tolerance). |
| 5. | It is slower . | It is faster than Relational Model. |
| 6. | Supports complex joins. | Does Not support for complex joins. |
| 7. | It is column-based. | It is field-based. |
| 8. | They are vertically scalable | They are horizontally scalable |
| 9. | Fast replication support is not provided. | They provide easy replication support |
| 10. | To add attributes in a relational model, database schema needs to be modified for including additional columns and their data types. | In a document-based database, you need to add additional key-value pairs in the document for representing new fields. |

**XML:** XML, or eXtensible Markup Language, is a text-based data format using tags to create hierarchies. It organizes information in a readable way, allowing easy data exchange between different systems. XML's versatility and human-friendly structure make it a widely used standard for representing and sharing structured data in various applications. XML, or eXtensible Markup Language, uses tags to structure data. For instance:

<person>

<name> John </name>

<age> 25 </age>

<address city="New York"> 123 Main Street </address>

</person>

Here, "person" is a tag, and it contains nested tags like "name" and "age" with associated data, and "address" with an attribute ("city").

**JSON:** JSON (JavaScript Object Notation) is a text-based, human-readable data interchange format used to exchange data between web clients and web servers. The format defines a set of structuring rules for the representation of structured data. Data in JSON is written in name and value pairs, similar to JavaScript object properties. JSON object literals are surrounded by curly braces {}. JSON object literals contains key/value pairs. Keys and values are separated by a colon. Keys must be strings, and values must be a valid JSON data type: string, number, object, array, boolean, null. Each key/value pair is separated by a comma.

For example, an array of employee names may look like this:

"employees":[

{ "firstname" : "John", "lastname" : "Doe" },

{ "firstname" : "Jane", "lastname" : "Doe" },

]

Each line is an object, and both lines together would be part of an array. Names in the name and value pairs include firstname and lastname, while the value pairs would be the actual appearing names, like John, Jane, and Doe.

**BSON:** BSON stands for Binary JSON developed by MongoDB. It is a binary file format that is used to store serialized JSON documents in a binary-encoded format. BSON can be parsed easily and very quickly because it supports type and length encoding, thanks to its binary structure. BSON objects are designed to be highly traversable and lightweight in nature, which makes it a better option for data transfer. It takes up less space and offers faster scan speed as compared to JSON objects. It offers a wide range of data types, like – date type, etc.), many of which are not supported by JSON.

Let’s take the following JSON document example.

{ "hello" : "world" }

When storing the JSON document, it will be converted to the following.

\x16\x00\x00\x00 // total document size

\x02 // 0x02 = type String

hello\x00 // field name

\x06\x00\x00\x00world\x00 // field value (size of value, value, null terminator

\x00 // 0x00 = type EOO ('end of object')

**MariaDB**: It is one of the most popular open-source relational database management system and could be used as a great replacement of MySQL. It is developed by original developers of MySQL. MariaDB can also stores data in various tables. It is also part of most cloud offerings and default in most Linux distributions. To establish the relationship between these tables, primary keys and foreign keys are used.

Features of MariaDB

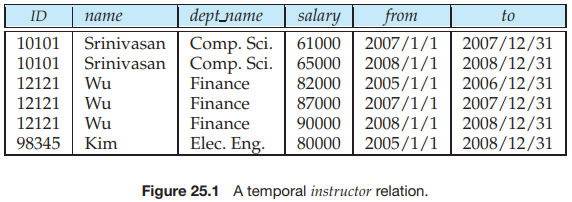
* It is fast, scalable, and robust.
* It has rich ecosystem of plugins, storage engines, etc which makes it very versatile for a wide variety of use cases.
* It is licensed under GPL, LGPL, or BSD.
* It uses a standard and popular querying language. It runs on different operating systems and supports variety of programming languages.
* It also offers support for PHP and offers Galera cluster technology.
* One of application is content management systems, also known as CMS. A CMS is a publication system that enables users to publish and manage large volumes of content on a website. CMS uses the MariaDB database without you needing to program it, work directly in the website source code, or write enquiries to the database resulting in a professional result for all your content publications.

|  |  |
| --- | --- |
| **JSON** | **XML** |
| JSON object has a type | XML data is typeless |
| JSON types: string, number, array, Boolean | All XML data should be string |
| Data is readily accessible as JSON objects | XML data needs to be parsed. |
| JSON is supported by most browsers. | Cross-browser XML parsing can be tricky |
| JSON has no display capabilities. | XML has the capability to display data because it is a markup language. |
| JSON supports only text and number data type. | XML support various data types such as number, text, images, charts, graphs, etc. It also provides options for transferring the structure or format of the data with actual data. |
| Retrieving value is easy | Retrieving value is difficult |
| Supported by many Ajax toolkit | Not fully supported by Ajax toolkit |
| A fully automated way of deserializing/serializing JavaScript. | Developers have to write JavaScript code to serialize/de-serialize from XML |
| Native support for object. | The object has to be express by conventions – mostly missed use of attributes and elements. |
| It supports only UTF-8 encoding. | It supports various encoding. |
| It doesn’t support comments. | It supports comments. |
| JSON files are easy to read as compared to XML. | XML documents are relatively more difficult to read and interpret. |
| It does not provide any support for namespaces. | It supports namespaces. |
| It is less secured. | It is more secure than JSON. |

| **JSON** | **BSON** |
| --- | --- |
| Standard file format Type. | Binary file format Type. |
| The language-free organization is utilized for offbeat server browser communication. | Binary JSON comprises a list of requested components containing a field title, sort, and esteem. Field title sorts are ordinarily a string. |
| Broadly JSON comprises of question and cluster where the question could be a collection of key-value sets and the cluster is a requested list of values. | The parallel encoding method comprises extra data such as lengths of strings and the protest subtypes. In addition, BinData and Date information sorts are the information sorts that are not upheld in JSON. |
| JSON stands for JavaScript Object Notation. | BSON stands for Binary JavaScript Object Notation. |
| JSON data contains its data basic in JSON format. | BSON gives extra datatypes over the JSON data. |
| Database like AnyDB, redis, etc stores information in JSON format. | MongoDB stores data in BSON format. |
| JSON uses less space in comparison to BSON. | BSON uses more space as compared to JSON. |
| It is slow as compared to BSON. | It is faster than JSON. |
| It is used for the transmission of data. | It is used for the storage of the data. |
| It has no encoding and decoding technique. | It has encoding and decoding technique. |

**ADBMS – MODULE 5**

**TEMPORAL DATABASES**: Temporal databases, in the broadest sense, encompass all database applications that require some aspect of time when organizing their information. Hence, they provide a good example to illustrate the need for developing a set of unifying concepts for application developers to use.

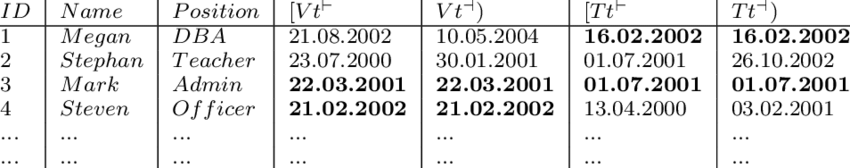
When considering the issue of time in database systems, we must distinguish between time as measured by the system and time as observed in the real world. The **valid time** for a fact is the set of time intervals during which the fact is true in the real world. The **transaction time** for a fact is the time interval during which the fact is current within the database system. This latter time is based on the transaction serialization order and is generated automatically by the system. **Note that valid-time intervals, being a real-world concept, cannot be generated automatically and must be provided to the system**. A **temporal relation** is one where each tuple has an associated time when it is true; the time may be either valid time or transaction time. Both valid time and transaction time can be stored, in which case the relation is said to be a **bitemporal relation**.

* **Valid Time:** It is a time in which the facts are true with respect to the real world.
* **Transaction Time:** It is the time at which the fact is currently present in the database.
* **Decision Time:** It is the time at which the decision is made about the fact.

**1. Uni-Temporal Relation:** The relation which is associated with valid, or transaction time is called Uni-Temporal relation. It is related to only one time.

**2. Bi-Temporal Relation:** The relation which is associated with both valid time and transaction time is called a Bi-Temporal relation. Valid time has two parts namely start time and end time, similar in the case of transaction time.

Eg: A table that records the employment history of individuals, including their positions and the times they held those positions. It uses two types of time: valid time (V) and transaction time (T). Valid time refers to the period during which a fact is true in the real world, and transaction time refers to the time period during which a fact is stored in the database.



**3. Tri-Temporal Relation:** The relation which is associated with three aspects of time namely Valid time, Transaction time, and Decision time called as Tri-Temporal relation.

**SPATIAL MODEL:** Spatial data include geographic data, such as maps and associated information, and computer-aided-design data.

Vector and raster are common data formats used to store geospatial data.

* Vectors are graphical representations of the real world. There are three main types of vector data: points, lines, and polygons. The points help create lines, and the connecting lines form enclosed areas or polygons. Vectors often represent the generalization of features or objects on the planet's surface. Vector data is usually stored in shapefiles, sometimes referred to as .shp files.
* Raster represents information presented in a pixel grid. Each pixel stored within a raster has value. This can be anything from a unit of measurement, color or information about a specific element. Typically, raster refers to imagery, but in spatial analysis it frequently refers to an orthoimage or the photos taken from aerial devices or satellites.

There is also something called an attribute. Whenever spatial data contains additional information or non-spatial data, it is called an attribute. Spatial data can have any number of attributes about a location. For example, this may be a map, photographs, historical information or anything else that may be deemed necessary.

* **Difference between MySQL and PostgreSQL:**

| **MySQL** | **PostgreSQL** |
| --- | --- |
| It is a **relational-based** DBMS. | It is an **object-based**relational DBMS |
| GUI tool provided is **MySQL Workbench** | **PgAdmin** is provided |
| Does not support partial, bitmap, & expression indexes. | It supports all of these |
| It doesn’t provide support for Materialised views and Table inheritance. | PostgreSQL provides both of them. |
| SQL only supports **Standard data types**. | It supports **Advanced data types** such as arrays, hstore, and user-defined types. |
| SQL provides **limited MVCC support**( in InnoDB) | **Full MVCC** support. |
| It is reliable, simple, and faster. | It is slower and more complex. |
| Troubleshooting MySQL is easy. | It is difficult to troubleshoot PostgreSQL. |
| It is best suitable for simple operations like write and reading. | It is commonly used for large and complex operations. |
| In MySQL, every connection created is an OS thread. | In PostgreSQL, every connection created is an OS process. |

* PostgreSQL is an advanced relational database system. PostrgeSQL supports both relational (SQL) and non-relational (JSON) queries. PostgreSQL is free and open-source. PostgreSQL supports various indexing techniques, including B-tree, hash, and GiST (Generalized Search Tree), which enhances query performance. Postgres includes features such as full-text search, support for spatial data, and extensive support for JSON and other semi-structured data. PostgreSQL allows users to define functions that are stored and executed on the server. PostgreSQL also supports function overloading. The PostgreSQL system architecture follows the process-per-transaction model. A running PostgreSQL site is managed by a central coordinating process, called the postmaster. The postmaster process is responsible for initializing and shutting down the server and also for handling connection requests from new clients

MOBILE DATABASES: A Mobile database is a database that can be connected to a mobile computing device over a mobile network (or wireless network). Here the client and the server have wireless connections. In today’s world, mobile computing is growing very rapidly, and it is huge potential in the field of the database. It will be applicable on different-different devices like android based mobile databases, iOS based mobile databases, etc. Common examples of databases are Couch base Lite, Object Box, etc.

Features of Mobile database:

* A cache is maintained to hold frequent and transactions so that they are not lost due to connection failure.
* As the use of laptops, mobile and PDAs is increasing to reside in the mobile system.
* Mobile databases are physically separate from the central database server.
* Mobile databases resided on mobile devices.
* Mobile databases are capable of communicating with a central database server or other mobile clients from remote sites.

Mobile Database typically involves three parties :

* **Fixed Hosts –** It performs the transactions & data management functions with the help of database servers.
* **Mobiles Units –**These are portable computers that move around a geographical region that includes the cellular network that these units use to communicate to base stations.
* **Base Stations –**  These are two-way radios installation in fixed locations, that pass communication with the mobile units to and from the fixed hosts.

**Limitations**

* It has Limited wireless bandwidth.
* In the mobile database, Wireless communication speed.
* It required Unlimited battery power to access.
* It is Less secured.
* It is Hard to make theft-proof.

**ADBMS – MODULE 6**

Database security is a broad area that addresses many issues, including the following:

■ Various legal and ethical issues regarding the right to access certain information—for example, some information may be deemed to be private and cannot be accessed legally by unauthorized organizations or persons. In the United States, there are numerous laws governing privacy of information.

■ Policy issues at the governmental, institutional, or corporate level as to what kinds of information should not be made publicly available—for example, credit ratings and personal medical records.

■ System-related issues such as the system levels at which various security functions should be enforced—for example, whether a security function should be handled at the physical hardware level, the operating system level, or the DBMS level.

■ The need in some organizations to identify multiple security levels and to categorize the data and users based on these classifications—for example, top secret, secret, confidential, and unclassified. The security policy of the organization with respect to permitting access to various classifications of data must be enforced.

Threats to Databases. Threats to databases can result in the loss or degradation of some or all of the following commonly accepted security goals: integrity, availability, and confidentiality.

■ Loss of integrity. Database integrity refers to the requirement that information be protected from improper modification. Modification of data includes creation, insertion, updating, changing the status of data, and deletion. Integrity is lost if unauthorized changes are made to the data by either intentional or accidental acts. If the loss of system or data integrity is not corrected, continued use of the contaminated system or corrupted data could result in inaccuracy, fraud, or erroneous decisions.

■ Loss of availability. Database availability refers to making objects available to a human user or a program to which they have a legitimate right.

■ Loss of confidentiality. Database confidentiality refers to the protection of data from unauthorized disclosure. The impact of unauthorized disclosure of confidential information can range from violation of the Data Privacy Act to the jeopardization of national security. Unauthorized, unanticipated, or unintentional disclosure could result in loss of public confidence, embarrassment, or legal action against the organization.