Types of Database Management Systems

Last Updated : 28 Jun, 2024

A Database Management System (DBMS) is a software system that is designed to manage and organize data in a structured manner. It allows users to create, modify, and query a database, as well as manage the security and access controls for that database.

What is DBMS?

A DBMS (Database Management System) is a software or technology used to manage data from a database. It allows users to store, modify and retrieve data in an organized way. It also provides security to the database.

Types of Database Management Systems

1. Hierarchical DBMS

- Hierarchical DBMS organizes data in a tree-like structure with parent-child relationships.
- Each parent can have multiple children, but a child can have only one parent.
- Data navigation is done through paths.

Syntax:

```
DBD NAME(database_name)
SEGM NAME(segment_name)
FIELD NAME(field_name), BYTES(bytes), START(start_byte)
```

Example:

```
GET /employees/123/projects/456
```

This retrieves project 456 for employee 123.

2. Network DBMS

- Network DBMS allows more flexible relationships, where a child can have multiple parents.
- Uses a network model with sets and records.
- Data navigation is done through sets.

Syntax: (Define Record Type)

```
RECORD NAME IS record_name
CONTAINS field_specifications
```

Syntax: (Define Set)

```
SET NAME IS set_name
OWNER IS owner_record
MEMBER IS member_record
```

Example

```
FIND owner OF project 456
```

This finds the owner(s) of project 456.

3. Relational DBMS (RDBMS)

- Relational DBMS stores data in tables with rows and columns.
- Relationships are established using keys.
- Data access is done using SQL (Structured Query Language).

Syntax: (Create Table)

```
CREATE TABLE table_name (
    column1 datatype PRIMARY KEY,
    column2 datatype,
    ...
);
```

Syntax: (Insert Data)

```
INSERT INTO table_name (column1, column2, ...)
VALUES (value1, value2, ...);
```

Syntax: (Select Data)

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Example:

```
SELECT * FROM employees WHERE salary > 50000
```

This retrieves all employees with a salary greater than 50000.

4. NoSQL DBMS

- NoSQL DBMS is designed for handling large volumes of unstructured or semi-structured data.
- Includes document databases, key-value stores, column-family stores, and graph databases.
- Data access varies depending on the specific NoSQL type.

Syntax: (Insert Document)

```
db.collection.insertOne(
    { key1: "value1", key2: "value2", ... }
);
```

Syntax: (Find Document)

```
db.collection.find(
    { key: "value" }
);
```

Example:

```
db.inventory.find( { status: "A" } )
```

This retrieves inventory items with status "A" in MongoDB.

5. Object Oriented DBMS (ODBMS)

- Object Oriented DBMS stores data as objects, which can have attributes and methods.
- Supports inheritance, encapsulation, and polymorphism.
- Data access is done using object query language (OQL).

Syntax: (Store Object)

```
ObjectContainer db = Db4o.openFile("database.db4o");
db.store(new Person("John", 30));
db.close();
```

Syntax: (Retrieve Object)

```
ObjectContainer db = Db4o.openFile("database.db4o");
ObjectSet result = db.queryByExample(new Person(null, 0));
while(result.hasNext()) {
    Person p = (Person)result.next();
    System.out.println(p);
}
db.close();
```

Example:

```
SELECT e.name FROM employees e WHERE e.age > 30
```

This retrieves the names of employees older than 30.

6. Graph-based DBMS

- Graph-based DBMS stores data in graph structures with nodes (entities) and edges (relationships).
- Uses graph query languages like Gremlin or SPARQL.

Syntax: (Create Node)

```
CREATE (n:Person {name: 'Alice', age: 30});
```

Syntax: (Create Relationship)

```
MATCH (a:Person {name: 'Alice'}), (b:Person {name: 'Bob'})
CREATE (a)-[:FRIEND]->(b);
```

Syntax: (Query Nodes and Relationships)

```
MATCH (a:Person)-[:FRIEND]->(b:Person)
RETURN a, b;
```

Example:

```
g.V().has('name', 'Alice').out('knows').values('name')
```

This retrieves the names of people Alice knows.

7. Document Database

- Document Database stores data in flexible, semi-structured documents (e.g., JSON, XML, BSON).
- No predefined schema, allowing for dynamic data structures.
- Data access is done using query languages specific to the document database (e.g., MongoDB Query Language).

Syntax: (Insert Document)

```
db.collection.insertOne(
    { name: "John", age: 30, department: "HR" }
);
```

Syntax: (Query Document)

```
db.collection.find(
    { name: "John" }
);
```

Example:

```
db.collection.find({field: "value"})
```

This retrieves documents from a collection where a specific field matches the given value.

8. Centralized Database

- Centralized Database is used to stored data in a single, central location.
- Easier to manage and maintain data consistency.
- Can become a single point of failure if the central system fails.

Distributed Database

- Distribute Database is used to store data which is distributed across multiple physical locations.
- Improved performance, scalability, and fault tolerance.

Examples with Outputs

Example 1: Relational DBMS (MySQL)

Create Table and Insert Data:

```
CREATE TABLE Employees (
    ID INT PRIMARY KEY,
    Name VARCHAR(50),
    Department VARCHAR(50)
);

INSERT INTO Employees (ID, Name, Department)
VALUES (1, 'John Doe', 'HR'), (2, 'Jane Smith', 'IT');
```

Select Data:

```
SELECT * FROM Employees;
```

Output:

ID	Name	Department
1	John Doe	HR
2	Jane Smith	IT

Relational DBMS (MySQL)

Example 2: NoSQL DBMS (MongoDB)

Insert Document and Find Document:

```
db.users.insertOne(
    { _id: 1, name: "Alice", age: 30 }
);

db.users.find(
    { name: "Alice" }
);
```

Output:

```
{ "_id": 1, "name": "Alice", "age": 30 }
```

Example 3: Hierarchical DBMS (IBM IMS)

Define Segment and Field

```
DBD NAME(EmployeeDB)
SEGM NAME(Employee)
FIELD NAME(EmpID), BYTES(4), START(1)
FIELD NAME(EmpName), BYTES(30), START(5)
FIELD NAME(Dept), BYTES(20), START(35)
```

Output: This defines a database called *EmployeeDB* with a segment *Employee* containing three fields: *EmpID*, *EmpName*, and *Dept*.

Conclusion

In conclusion, different types of DBMSs serve different purposes and are suited to various types of applications. It acts as an interface between the database and its users or application programs, provides many operations e.g. creating a database, Storing in the database, updating an existing database, delete from the database. Understanding the