Object Oriented DBMS

An object database is managed by an object-oriented database management system (OODBMS). The database combines object-oriented programming concepts with relational database principles.

> ***An Object-Oriented Database (OODB) is a database management system that supports the creation and modelling of data as objects. This approach is inspired by object-oriented programming (OOP) concepts, integrating database capabilities with object-oriented principles.***

- Objects are the basic building block and an instance of a class, where the type is either built-in or user-defined.
- Classes provide a schema or blueprint for objects, defining the behaviour.
- Methods determine the behaviour of a class.
- Pointers help access elements of an object database and establish relations between objects.

Key Features:

- **Objects**: Data is stored as objects, similar to objects used in OOP. Each object contains data in the form of fields (attributes) and methods (operations).
- **Classes and Inheritance**: Objects are instances of classes. Classes can inherit properties and methods from other classes, promoting reusability and modularity.
- **Encapsulation**: Objects encapsulate both data and behavior, ensuring that internal details are hidden from other objects and accessed only through defined interfaces.
- **Polymorphism**: Objects can be accessed and manipulated through interfaces that are defined at a higher level in the class hierarchy, allowing for flexible and dynamic behavior.
- **Abstraction**: Abstraction is the procedure of representing only the essential data features for the needed functionality. The process selects vital information while unnecessary information stays hidden. Abstraction helps reduce the complexity of modelled data and allows reusability.
- **Function Overloading**: Function overloading allows multiple functions to have the same name as long as their parameters (type, number, or both) are different.

Characteristics of ORDBMS:

Object-Relational Database Management System (ORDBMS) combines the features of both relational databases (RDBMS) and object-oriented databases (OODBMS). This hybrid approach offers several key characteristics:

1. **Complex Data Types**: ORDBMS supports complex data types such as multimedia, spatial, and user-defined types, extending the traditional scalar data types like integers and strings.
2. **Inheritance**: Classes and objects in an ORDBMS can use inheritance, allowing new classes to be derived from existing ones, promoting reusability and flexibility.

3. **Encapsulation**: ORDBMS allows encapsulation of methods within objects, meaning that data can be manipulated only through predefined interfaces, ensuring data integrity and consistency.
4. **Polymorphism**: This feature allows entities to be processed in different ways depending on their data type or class, enhancing flexibility in handling various data operations.
5. **Support for SQL**: ORDBMS integrates object-oriented features into the SQL language, providing extended SQL standards to handle complex data types and objects.
6. **Relationships and Constraints**: Like RDBMS, ORDBMS supports primary keys, foreign keys, and other relational constraints to maintain data integrity and relationships between tables.
7. **Enhanced Query Capabilities**: ORDBMS provides advanced querying capabilities, including the ability to query complex data types and objects using object-oriented extensions to SQL.
8. **Extensibility**: Users can define custom data types, functions, and operators, allowing the database to be tailored to specific application requirements.

Feature	Description
Query Language	Language to find objects and retrieve data from the database.
Transparent Persistence	Ability to use an object-oriented programming language for data manipulation.
ACID Transactions	ACID transactions guarantee all transactions are complete without conflicting changes.
Database Caching	Creates a partial replica of the database. Allows access to a database from program memory instead of a disk.
Recovery	Disaster recovery in case of application or system failure.

Advantages:

- **Versatility**: Can handle both traditional relational data and complex object-oriented data.
- **Reusability**: Inheritance and polymorphism promote code and data reuse.
- **Data Integrity**: Encapsulation and constraints ensure consistent and accurate data.

Disadvantages:

- **Complexity**: More complex to design, implement, and manage than traditional RDBMS.
- **Performance**: May have performance overhead due to the additional complexity and features.

Use Cases:

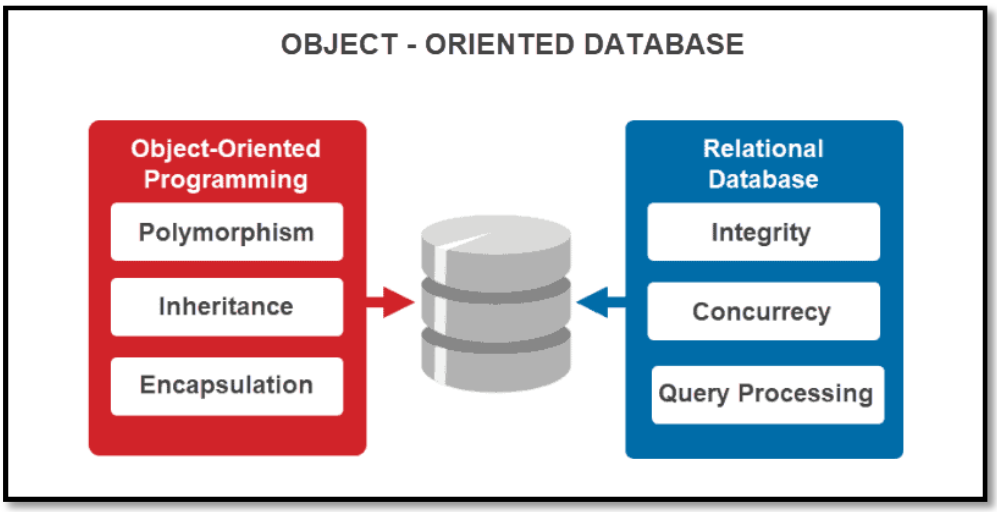
- **Engineering Applications:** CAD/CAM, telecommunications
- **Multimedia:** Image, video, and audio storage and retrieval
- **Scientific Data:** Bioinformatics, geospatial databases

ORDBMS provides a powerful and flexible solution for managing both relational and complex data types, bridging the gap between traditional relational databases and modern object-oriented applications.

NOTE: ***The main characteristic of objects in OODBMS is the possibility of user-constructed types. An object created in a project or application saves into a database as is.***

NOTE: ***Object-oriented databases directly deal with data as complete objects. All the information comes in one instantly available object package instead of multiple tables.***

NOTE: ***In contrast, the basic building blocks of relational databases, such as PostgreSQL or MySQL, are tables with actions based on logical connections between the table data.***



Example –

PostgreSQL -

PostgreSQL is one of the most popular and widely used ORDBMS. It supports advanced data types, custom functions, and object-oriented features such as inheritance and polymorphism.

Key Features:

- Support for complex data types and custom types
- Extensive support for SQL and advanced querying capabilities
- Inheritance and polymorphism
- JSON and XML data handling
- Strong ACID compliance and transactional integrity

Oracle Database -

Oracle Database offers extensive object-relational features and is widely used in enterprise environments. It provides robust support for complex data types, object types, and user-defined types.

Key Features:

- Support for user-defined types and object types
- Advanced data management capabilities
- Strong security features and performance optimization
- Comprehensive SQL support with object-oriented extensions

IBM Db2 -

IBM Db2 is another powerful ORDBMS that provides robust support for object-relational features, including structured data types, user-defined types, and object-oriented programming features.

Key Features:

- Support for complex data types and structured types
- Advanced analytics and data management tools
- High availability and scalability
- Comprehensive SQL support with object-oriented enhancements

Informix -

IBM Informix (previously developed by Informix Corporation) is known for its ability to handle complex data types and support object-oriented features, making it suitable for applications requiring sophisticated data modelling.

Key Features:

- Support for user-defined types and complex data types
- Advanced data replication and high availability features
- Robust performance and scalability
- Integration with SQL and NoSQL data models

ORDBMS	Key Features
PostgreSQL	Advanced data types, inheritance, polymorphism, strong ACID compliance
Oracle Database	User-defined types, object types, advanced data management, strong security
IBM Db2	Complex data types, advanced analytics, high availability, SQL with object-oriented enhancements
Informix	User-defined types, data replication, performance, SQL and NoSQL integration

Differences between RDBMS & ORDBMS

RDBMS (Relational Database Management System) -

1. **Data Model**:
   - Uses a tabular structure where data is stored in rows and columns.
   - Relationships are defined using primary and foreign keys.
2. **Complex Data Types**:
   - Supports simple data types like integers, strings, and dates.
   - Lacks support for complex data types and user-defined types.
3. **Inheritance and Encapsulation**:
   - Does not support object-oriented features such as inheritance and encapsulation.
4. **Query Language**:
   - Standard SQL (Structured Query Language) is used for querying and managing the database.
5. **Performance**:
   - Generally optimized for transaction processing and query performance.
   - Efficient for handling large volumes of simple data.
6. **Examples**:
   - MySQL, Microsoft SQL Server, SQLite

ORDBMS (Object-Relational Database Management System) -

1. **Data Model**:
   - Combines the tabular structure of RDBMS with object-oriented features.
   - Supports complex data types and object-oriented concepts.
2. **Complex Data Types**:
   - Supports complex data types, including multimedia, spatial, and user-defined types.
   - Can handle structured objects and methods.

3. **Inheritance and Encapsulation**:
    - Supports object-oriented features like inheritance, polymorphism, and encapsulation.
    - Allows for the creation of classes and objects within the database.
4. **Query Language**:
    - Extends SQL to support object-oriented features, providing enhanced querying capabilities.
5. **Performance**:
    - May have additional performance overhead due to the complexity of managing objects and relationships.
    - Suitable for applications requiring complex data representation and manipulation.
6. **Examples**:
    - PostgreSQL, Oracle Database, IBM Db2

Feature	RDBMS	ORDBMS
Data Model	Tabular structure	Combines tabular and object-oriented models
Complex Data Types	Simple data types	Complex and user-defined data types
Inheritance and Encapsulation	Not supported	Supported
Query Language	Standard SQL	Extended SQL with object-oriented features
Performance	Optimized for simple data	May have overhead due to complexity
Examples	MySQL, Microsoft SQL Server, SQLite	PostgreSQL, Oracle Database, IBM Db2

■ What do you mean by Complex Object in ORDBMS?

In an Object-Relational Database Management System (ORDBMS), a **complex object** is an object that can encapsulate other objects and data types, providing a richer and more flexible way to represent and manage data.

Characteristics of Complex Objects in ORDBMS:

1. **Composite Structure**:
    - A complex object can contain other objects, arrays, or collections, forming a hierarchical or nested structure.
    - For example, an Employee object might contain other objects like Address and ContactInfo.
2. **User-Defined Types (UDTs)**:
    - Users can define their own data types that encapsulate multiple attributes and methods, similar to classes in object-oriented programming.
    - These types can be used to model real-world entities more accurately.
3. **Encapsulation**:
    - Complex objects encapsulate both data (attributes) and behavior (methods).
    - This ensures that data is accessed and manipulated through well-defined interfaces.
4. **Inheritance**:

- Complex objects can inherit properties and behaviors from other objects, promoting reusability and modularity.
- For instance, a Manager object can inherit from an Employee object and add additional attributes or methods.

5. **Relationships**:
   - Complex objects can establish relationships with other objects, allowing for the representation of associations, aggregations, and compositions.
   - This helps in modeling complex real-world scenarios where entities are interconnected.

Example of a Complex Object:

Consider a database for a company where each Employee has an Address and a list of Projects.

```
CREATE TYPE Address AS OBJECT (
  street VARCHAR2(50),
  city VARCHAR2(50),
  zipcode NUMBER(6)
);

CREATE TYPE Project AS OBJECT (
  project_id NUMBER,
  project_name VARCHAR2(50)
);

CREATE TYPE ProjectList AS TABLE OF Project;

CREATE TYPE Employee AS OBJECT (
  emp_id NUMBER,
  name VARCHAR2(50),
  address Address,
  projects ProjectList
);
```

In this example:

- The Address type is a complex object that includes multiple attributes.
- The Project type is another complex object that represents a project.
- The Employee type encapsulates the Address object and a list of Project objects, making it a complex object.

Advantages of Using Complex Objects:

1. **Real-World Modeling**: Allows for a more accurate representation of real-world entities and their relationships.
2. **Data Integrity**: Encapsulation ensures that data is manipulated through defined methods, maintaining integrity.
3. **Reusability**: Inheritance and object composition promote code and data reuse.

NOTE: Complex objects in ORDBMS provide a powerful way to model and manage sophisticated data structures, making them ideal for applications that require detailed and interconnected data representations.

Specific Case Studies -

GemStone/S - GemStone/S is an object database system based on Smalltalk – an object-oriented programming language influenced by Java. Developers who write applications in Smalltalk adapt easily to this database. GemStone/S integrates seamlessly with existing Smalltalk applications, improving speed and productivity. Gemstone/S is best for high-availability projects. There are

multiple options for licensing depending on the project size. The database server is available for various platforms, including Linux, Windows, macOS, Solaris, AIX, as well as Raspberry Pi.

ObjectDB - ObjectDB is a NoSQL object database for the Java programming language. Compared to other NoSQL databases, ObjectDB is ACID compliant. ObjectDB does not provide an API and requires using one of the two built-in Java database APIs:

- JPA with JPA Query Language (JPQL) based on Java syntax.
- JDO with JDO Query Language (JDQL) based on SQL syntax.

ObjectDB includes all basic data types in Java, user-defined classes, and standard Java collections. Every object has a unique ID. The number of elements is limited only by the maximum database size (128 TB). ObjectDB is available cross-platform and the benchmark performance is exceptional.

ObjectDatabase++ - ObjectDatabase++ is a real-time embeddable object database designed for server-side applications. The required external maintenance is minimal.

ObjectDatabase++ supports:

- Multi-process with multi-threaded server applications.
- Full transaction control.
- Real-time recovery.
- C++ related languages, VB.NET as well as C#.

The object database is C++ based. One of the main features is advanced auto-recovery from system crashes without compromising the database integrity.

Objectivity/DB - Objectivity/DB utilizes the power of objects and satisfies the complex requirements within Big Data. The object database is flexible by supporting multiple languages:

C++/C#/Python/Java

The schema changes happen dynamically without the need for downtime, allowing real-time queries against any data type. Objectivity/DB is available for multiple platforms, including macOS, Linux, Windows, or Unix.

ObjectStore - ObjectStore integrates with C++ or Java and provides memory persistency to improve the performance of application logic. The object database is ACID-compliant. The responsiveness allows developers to build distributed applications cross-platform, whether on-premises or in the cloud. The main feature is cloud scalability, which allows database access from anywhere. ObjectStore simplifies the data creation and exchange process seamlessly.

Versant - Versant provides primary transparent object persistence from C++, Java, and .NET. However, there is also support for Smalltalk and Python. Versant supports different APIs depending on the language used. Standard SQL queries are also available, making Versant a NoSQL database. The object database is a multi-user client-server database. Versant performs best when used for online transaction systems with large amounts of data and concurrent users.

==========