Ayush Sagar
24MCA10066
DBMS Assignment: 4

Q1. Explain the key differences between an Object-Relational Database Management System (ORDBMS) and an Object-Oriented Database Management System (OODBMS). Provide at least two examples where each type would be most suitable for use in real-world applications.

ANSWER:

OODBMS	ORDBMS
It stands for Object Oriented Database Management System.	It stands for Object Relational Database Management System.
Object-oriented databases, like Object Oriented Programming, represent data in the form of objects and classes.	An object-relational database is one that is based on both the relational and object-oriented database models.
OODBMSs support ODL/OQL.	ORDBMS adds object-oriented functionalities to SQL.
Every object-oriented system has a different set of constraints that it can accommodate.	Keys, entity integrity, and referential integrity are constraints of an object-oriented database.
The efficiency of query processing is low.	Processing of queries is quite effective.

Examples of Suitable Real-World Applications:

ORDBMS (Object-Relational DBMS)

➢ Financial Systems:
o In financial applications, where traditional structured data (like transactions and account balances) are combined with more complex, user-defined types (like financial instruments or contracts). ORDBMS can handle the relational data while accommodating complex types when needed.

➢ Geographic Information Systems (GIS):
o GIS applications need to handle both spatial data (such as maps, regions, and boundaries) and tabular data. ORDBMS is ideal for these applications as it provides a mix of relational tables for structured data and extended support for spatial data types.

ODBMS (Object-Oriented DBMS)

➢ CAD/CAM Systems (Computer-Aided Design/Manufacturing):
o CAD systems handle highly complex objects, such as designs and components, which often involve intricate relationships between parts and assemblies. An ODBMS provides the best way to model, store, and manage these objects while retaining their behaviours and relationships.

➢ Telecommunications Systems:
o In real-time telecommunications systems, call records, event logs, and network objects are naturally modelled as objects. ODBMS allows efficient object storage and retrieval, with seamless handling of real-time event processing and complex relationships between different entities.

Q2. Describe how structured types and inheritance are implemented in SQL. Create an example where a structured type is used to define an object, and demonstrate how table inheritance can be utilized to extend this structured type.

ANSWER:
Structured types and inheritance in SQL are primarily used in Object-Relational Databases (ORD), such as PostgreSQL and Oracle, to model complex data and relationships between different entities.

1. Structured Types:
Structured types in SQL define a composite data type, similar to how objects in object-oriented programming encapsulate multiple attributes. These types can be used to store complex data like objects, where each attribute has its own type.

- Defining Structured Types:
Structured types allow you to group several related attributes into a single data type. These types can be reused in various tables, simplifying the schema and encouraging modular design.

Sql- Define a structured type for “Person"
```
CREATE TYPE Person AS (
   first_name VARCHAR(50),
   last_name VARCHAR(50),
   date_of_birth DATE
);
```

Here, we have created a structured type `Person` that contains attributes `first_name`, `last_name`, and `date_of_birth`. This type can now be used in other table definitions.

2. Inheritance in SQL:
Some SQL databases, particularly PostgreSQL, support table inheritance, allowing one table to inherit the structure and data from another table, much like class inheritance in object-oriented programming.

- Table Inheritance:

When you define a new table, you can specify that it inherits from an existing table. The child table automatically inherits the parent table's columns, and you can add additional columns or constraints specific to the child table.

Example: Combining Structured Types and Table Inheritance
Let's say we have a structured type `Person` and we want to create a table `Employee` that represents employees as a specific type of person. We'll use inheritance to extend the `Person` structure.

Sql- Create a table using the structured type Person

```
CREATE TABLE PersonTable (
   person_id SERIAL PRIMARY KEY,
   person_info Person
);
```

Insert into PersonTable

```
INSERT INTO PersonTable (person_info)
VALUES
   (ROW('John', 'Doe', '1990-01-01')),
   (ROW('Jane', 'Smith', '1985-07-12'));
```

Select from PersonTable

```
SELECT * FROM PersonTable;
```

Create a table Employee that inherits from PersonTable

```
CREATE TABLE EmployeeTable (
   employee_id SERIAL PRIMARY KEY,
   salary DECIMAL(10, 2)
) INHERITS (PersonTable);
```

Insert into EmployeeTable

```
INSERT INTO EmployeeTable (person_info, salary)
VALUES
   (ROW('Mike', 'Johnson', '1992-03-15'), 60000.00),
   (ROW('Sara', 'Williams', '1991-09-30'), 75000.00);
```

Select from EmployeeTable

```
SELECT * FROM EmployeeTable;
```

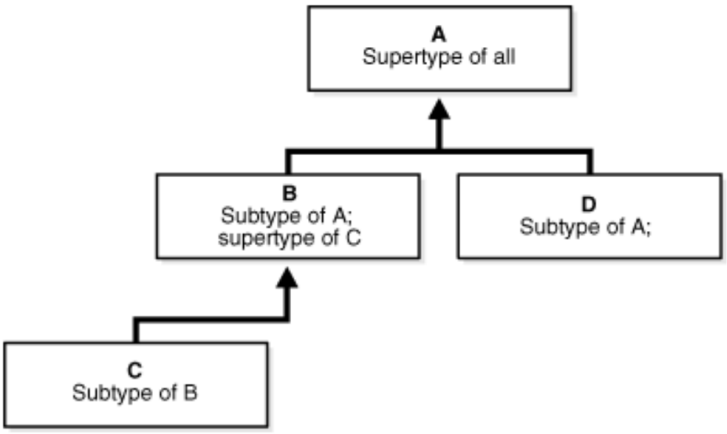
Explanation:
1. Person Structured Type:
   `Person` is a composite type encapsulating `first_name`, `last_name`, and `date_of_birth`.
   - We define a table `PersonTable` to hold `Person` objects in its `person_info` column.

2. Inheritance:
`EmployeeTable` inherits from `PersonTable`. It extends the parent table by adding an `employee_id` and a `salary` column.
When querying `EmployeeTable`, we can access both the `person_info` attributes from `PersonTable` and the new attributes (`employee_id`, `salary`) defined specifically for employees.

3. Inserting and Querying:

You can insert and query from both tables. `PersonTable` holds general person data, while `EmployeeTable` inherits this data and adds employee-specific information.

Figure 2-1 Supertypes and Subtypes in Type Hierarchy



Q3. What is object-identity in the context of SQL, and why is it important in Object-Relational Database Management Systems (ORDBMS)? Explain how reference types work in SQL with an example that demonstrates how object identity can be established between two related tables.

ANSWER:

Object-Identity refers to the unique identification of an object within a database, which is crucial for maintaining relationships and integrity among data entities. In the context of Object-Relational Database Management Systems (ORDBMS), object-identity allows objects to be recognized and referenced uniquely, even if their attributes change over time. This is particularly important in applications that model real-world entities and their relationships, such as in complex data management systems.

Importance of Object-Identity in ORDBMS

1. Uniqueness: Object-identity ensures that each object can be uniquely identified, which is essential for operations like updates and deletions.
2. Referential Integrity: It helps maintain referential integrity by allowing foreign keys to reference objects based on their identity rather than their attribute values, which may change.
3. Complex Data Types: ORDBMS supports complex data types and relationships, making object-identity crucial for managing these types effectively.
4. Encapsulation: It allows for encapsulation of data and behavior, where objects can contain both data attributes and methods that operate on that data.

Reference Types in SQL

In SQL, reference types allow for the creation of relationships between tables, typically through foreign keys. A reference type can point to an object (or row) in another table, establishing a connection between the two entities based on their unique identifiers.

Example Demonstrating Object-Identity

Let's consider a scenario where we have two related tables: Customers and Orders. Each Customer has a unique identity, and each Order is associated with a specific Customer.

```
CREATE TABLE Customers (
    CustomerID SERIAL PRIMARY KEY,
    Name VARCHAR(100),
    Email VARCHAR(100)
);
```

Here, CustomerID serves as the unique identifier for each customer, ensuring object-identity.

```
CREATE TABLE Orders (
    OrderID SERIAL PRIMARY KEY,
    OrderDate DATE,
    CustomerID INT,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
```

In the Orders table, CustomerID acts as a foreign key referencing the CustomerID in the Customers table. This establishes a relationship between orders and customers based on their unique identities.

Inserting Data into Customers

```
INSERT INTO Customers (Name, Email) VALUES ('Alice Smith', 'alice@example.com');
INSERT INTO Customers (Name, Email) VALUES ('Bob Johnson', 'bob@example.com');
```

Inserting Data into Orders:

```
INSERT INTO Orders (OrderDate, CustomerID) VALUES ('2023-10-01', 1); -- Order for Alice
INSERT INTO Orders (OrderDate, CustomerID) VALUES ('2023-10-02', 2); -- Order for Bob
```

Querying Data to Demonstrate Object-Identity

To retrieve orders along with customer details, you can perform a join:

```
SELECT o.OrderID, o.OrderDate, c.Name, c.Email
FROM Orders o
JOIN Customers c ON o.CustomerID = c.CustomerID;
```

Q4. Discuss the role of XML in managing semi-structured data within databases. Explain the difference between an XML Document Type Definition (DTD) and an XML Schema.
Provide an example of when each would be used in an application.

ANSWER:
XML (eXtensible Markup Language) is a versatile markup language that plays a crucial role in managing semi-structured data within databases. Unlike traditional structured data, which fits neatly into tables with predefined schemas, semi-structured data lacks a rigid structure but still contains tags or markers that provide context for the data. Here's how XML facilitates the management of such data:

1. Flexibility and Hierarchical Structure

- Schema-less Nature: XML allows for a flexible data model where elements can be added or modified without requiring changes to a fixed schema. This is particularly useful for applications where data formats evolve over time.
- Hierarchical Representation: XML supports a tree-like structure, enabling the representation of complex relationships and nested data. This is beneficial for modeling data that naturally fits into a hierarchy, such as organizational structures or product catalogs.

2. Self-Describing Data

- Metadata Inclusion: XML documents include tags that describe the data, making it self-describing. This means that the context and meaning of the data are embedded within the document itself, which enhances data interchange and understanding across different systems.
- Interoperability: The self-describing nature of XML facilitates data sharing between heterogeneous systems, as the structure and semantics of the data are clear and accessible.

3. Data Storage and Retrieval

- XML Databases: Specialized databases, known as XML databases, are designed to store and manage XML data efficiently. They provide features like indexing, querying, and validation specific to XML content.
- Querying with XQuery: XML data can be queried using languages like XQuery, which is designed to extract and manipulate XML data. This allows for complex queries that can navigate the hierarchical structure of XML documents.

4. Integration with Other Technologies

- Web Services: XML is widely used in web services (e.g., SOAP) for data exchange, enabling seamless integration between different applications and platforms.

- Data Serialization: XML serves as a format for serializing data structures, making it easier to transmit data over networks or store it in files.

5. Support for Standards and Protocols

- Widespread Adoption: XML is supported by many standards and protocols, such as XML Schema for validation and XSLT for transformation. This broad support enhances its usability in various applications, including databases.
- Data Exchange Standards: XML is often used in data exchange standards, such as HL7 in healthcare or XBRL in financial reporting, which require the representation of semi-structured data.

Difference Between DTD and XSD

DTD	XSD
DTD are the declarations that define a document type for SGML.	XSD describes the elements in a XML document.
It doesn't support namespace.	It supports namespace.
It is comparatively harder than XSD.	It is relatively more simpler than DTD.
It doesn't support datatypes.	It supports datatypes.
SGML syntax is used for DTD.	XML is used for writing XSD.
It is not extensible in nature.	It is extensible in nature.
It doesn't give us much control on structure of XML document.	It gives us more control on structure of XML document.
It specifies only the root element.	Any element which is made global can be done as root as markup validation.

DTD	XSD
It doesn’t have any restrictions on data used.	It specifies certain data restrictions.
It is not much easy to learn .	It is simple in learning.
File here is saved as .dtd	File in XSD is saved as .xsd file.
It is not a strongly typed mechanism.	It is a strongly typed mechanism.
It uses #PCDATA which is a string data type.	It uses fundamental and primitive data types.

An example where we define the structure of a bookstore catalog. Both DTD and XML Schema (XSD) will be used to describe the structure, but the two approaches will demonstrate their differences in terms of validation and syntax.

1. XML DTD Example

```
<!DOCTYPE bookstore SYSTEM "bookstore.dtd">

<bookstore>

  <book>

    <title>XML Basics</title>

    <author>John Doe</author>

    <price>19.99</price>

  </book>

  <book>

    <title>Advanced XML</title>

    <author>Jane Smith</author>

    <price>29.95</price>

  </book>

</bookstore>
```

DTD Definition (bookstore.dtd):

<!ELEMENT bookstore (book+)>

<!ELEMENT book (title, author, price)>

<!ELEMENT title (#PCDATA)>

<!ELEMENT author (#PCDATA)>

<!ELEMENT price (#PCDATA)>

Q5. Explain the primary reasons for using a NoSQL database over a traditional RDBMS. Describe the main types of NoSQL.

ANSWER:

NoSQL databases have gained popularity due to their ability to handle specific use cases and challenges that traditional relational database management systems (RDBMS) may struggle with. Here are the primary reasons for choosing NoSQL:

1. Scalability:
    - Horizontal Scaling: NoSQL databases are designed to scale out by adding more servers, making them better suited for handling large volumes of data and high traffic loads.
    - Elasticity: They can dynamically adjust to varying workloads, allowing for more flexible resource management.
2. Flexibility in Data Models:
    - Schema-less Design: NoSQL databases allow for unstructured or semi-structured data, enabling developers to store data without a predefined schema. This is particularly useful for applications that evolve rapidly.
    - Variety of Data Types: They can handle various data formats (e.g., JSON, XML, key-value pairs), accommodating diverse data needs.
3. Performance:
    - High Throughput: NoSQL databases can provide faster read and write operations, especially for large datasets, due to their optimized data structures and storage mechanisms.
    - Reduced Latency: They often offer lower latency for data access, which is critical for real-time applications.
4. Distributed Architecture:
    - Data Distribution: Many NoSQL databases are inherently designed to distribute data across multiple nodes, enhancing fault tolerance and availability.
    - Replication and Sharding: They support data replication and sharding, which helps in load balancing and ensures data redundancy.
5. Handling Big Data:

- Large Volume of Data: NoSQL databases are well-suited for big data applications that require the storage and processing of vast amounts of information from various sources.
- Real-time Analytics: They can support real-time analytics and processing, making them ideal for applications that require immediate insights.

6. Support for Complex Data Structures:
    - Nested Data: NoSQL databases can efficiently store complex data structures, such as arrays and objects, which can be cumbersome in traditional RDBMS.
    - Graph and Document Models: Some NoSQL databases, like graph databases and document stores, are optimized for specific data relationships and structures.
7. Cost-Effectiveness:
    - Open Source Options: Many NoSQL databases are open-source, reducing licensing costs compared to traditional RDBMS.
    - Commodity Hardware: They can run on inexpensive hardware, making it cost-effective to scale out.

NoSQL is a non-relational database that is used to store the data in the nontabular form. NoSQL stands for Not only SQL. The main types are documents, key-value, wide-column, and graphs.
Types of NoSQL Database:

- Document-based databases
- Key-value stores
- Column-oriented databases
- Graph-based databases

Document-Based Database:

The document-based database is a nonrelational database. Instead of storing the data in rows and columns (tables), it uses the documents to store the data in the database. A document database stores data in JSON, BSON, or XML documents.

Documents can be stored and retrieved in a form that is much closer to the data objects used in applications which means less translation is required to use these data in the applications. In the Document database, the particular elements can be accessed by using the index value that is assigned for faster querying.

Collections are the group of documents that store documents that have similar contents. Not all the documents are in any collection as they require a similar schema because document databases have a flexible schema.

Key features of documents database:

- Flexible schema: Documents in the database has a flexible schema. It means the documents in the database need not be the same schema.
- Faster creation and maintenance: the creation of documents is easy and minimal maintenance is required once we create the document.
- No foreign keys: There is no dynamic relationship between two documents so documents can be independent of one another. So, there is no requirement for a foreign key in a document database.
- Open formats: To build a document we use XML, JSON, and others.

Key-Value Stores:

A key-value store is a nonrelational database. The simplest form of a NoSQL database is a key-value store. Every data element in the database is stored in key-value pairs. The data can be retrieved by using a unique key allotted to each element in the database. The values can be simple data types like strings and numbers or complex objects.

A key-value store is like a relational database with only two columns which is the key and the value.

Key features of the key-value store:

- Simplicity.
- Scalability.
- Speed.

Column Oriented Databases:

A column-oriented database is a non-relational database that stores the data in columns instead of rows. That means when we want to run analytics on a small number of columns, you can read those columns directly without consuming memory with the unwanted data.

Columnar databases are designed to read data more efficiently and retrieve the data with greater speed. A columnar database is used to store a large amount of data. Key features of columnar oriented database:

- Scalability.
- Compression.
- Very responsive.

Graph-Based databases:

Graph-based databases focus on the relationship between the elements. It stores the data in the form of nodes in the database. The connections between the nodes are called links or relationships.

Key features of graph database:

- In a graph-based database, it is easy to identify the relationship between the data by using the links.
- The Query’s output is real-time results.
- The speed depends upon the number of relationships among the database elements.
- Updating data is also easy, as adding a new node or edge to a graph database is a straightforward task that does not require significant schema changes.