**NAME: JOHN LOUIE N. PURISIMA**

**BSIT—3B DATABASE**

Pre Activity Database

Explain what XML is and how it differs from HTML. Provide one practical use case where XML would be more suitable than HTML.  
**Definition:**  
-XML (eXtensible Markup Language) is a markup language designed to store and transport data. Unlike HTML, which focuses on how data is displayed (presentation), XML focuses on the structure and meaning of the data (content). **Key Differences:**

•XML is data-centric, while HTML is presentation-centric.

•XML is **customizable** (you define your own tags), HTML has a **fixed set of tags**.

•XML is used for **data exchange,** HTML is used for **web page rendering**.  
**Use Case:**   
-A **banking system** that sends transaction data between applications would benefit from using XML because the data structure can be easily interpreted by different systems regardless of platform or programming language.

1. In your own words, why might a relational database like SQL Server allow the use of XML as a data type? What are the potential benefits and drawbacks?

-Relational databases like SQL Server allow the use of XML as a data type to handle semi-structured data more effectively. While relational databases traditionally excel with structured tabular data, the inclusion of XML extends their flexibility, enabling them to manage hierarchical or tree-like data that doesn’t fit neatly into rows and columns.

**Potential Benefits:**

1. **Data Versatility**: XML accommodates nested, hierarchical data structures, making it ideal for scenarios where data relationships are complex or require flexibility.
2. **Interoperability**: XML is widely used in web services and data exchange. Having XML data natively stored in the database simplifies integration with external systems.
3. **Rich Querying**: SQL Server allows querying XML data using XQuery, enabling sophisticated operations on XML content without needing to transform it.
4. **Simplified Data Management**: XML reduces the need to maintain separate mechanisms for semi-structured data, streamlining application architecture.

### Potential Drawbacks:

1. **Performance Overhead**: XML data requires more storage and processing power compared to traditional data types, potentially affecting database performance.
2. **Complexity**: Using XML introduces additional complexity to database design and maintenance, especially for developers unfamiliar with working in hybrid relational-XML environments.
3. **Inefficiency in Purely Relational Scenarios**: For data that could be normalized, XML might be overkill, leading to inefficiencies in retrieval or updates.
4. Given the XML below, how would you describe its structure? Identify the root element, child elements, and any nested relationships.

<order>

<customer>

<name>John Doe</name>

<email>john@example.com</email>

</customer>

<items>

<item>

<product>Keyboard</product>

<quantity>1</quantity>

</item>

<item>

<product>Mouse</product>

<quantity>2</quantity>

</item>

</items>

</order>

**Answer:**

•**Root Element**: <order> The entire XML document is enclosed within this root element, which represents a single order.

•**Child Elements** of <order>:

* <customer>: Represents details about the customer who placed the order.
* <items>: Contains a collection of items included in the order.

• **Nested Relationships**:

* Within <customer>, there are two child elements:
  + <name>: Holds the customer's name, "John Doe."
  + <email>: Holds the customer's email address, "john@example.com."
* Within <items>, there are multiple <item> elements, each representing a product in the order.
  + Each <item> has:
    - <product>: Specifies the product name (e.g., "Keyboard" or "Mouse").
    - <quantity>: Indicates the number of units ordered (e.g., "1" or "2").

### **The following XML is stored in a column named** OrderDetails **in a table called** OrdersTable**. Write an SQL query using the** .value() **method to retrieve the customer’s email.**

<order>

<customer>

<name>John Doe</name>

<email>john@example.com</email>

</customer>

<items>

<item>

<product>Keyboard</product>

<quantity>1</quantity>

</item>

<item>

<product>Mouse</product>

<quantity>2</quantity>

</item>

</items>

</order>

**Answer:**

SELECT

OrderDetails.value('(/order/customer/email)[1]', 'NVARCHAR(MAX)') AS CustomerEmail

FROM

OrdersTable;

1. If you were asked to generate a report showing all products in all orders from an XML column, how would you approach the task in SQL Server using. nodes () and .value () methods? Describe your approach step by step.

**Step-by-Step**

* + - 1. **Examine the XML Structure**: The XML contains <order> as the root element, with <items> as a child element that holds multiple <item> nodes. Each <item> includes <product> and <quantity>.
      2. **Use the** .nodes() **Method to Extract Each** <item>: The .nodes() method is used to break the XML into rows, so each <item> node becomes a separate row. This ensures every product is treated individually.
      3. **Retrieve Values with** .value(): The .value() method is applied to extract specific elements from each <item> node, such as <product> for the product name and <quantity> for the quantity.
      4. **Write the SQL Query**: Combine .nodes() and .value() in a query to process the XML column and generate the desired output:

SELECT

OD.value('(product)[1]', 'NVARCHAR(MAX)') AS Product,

OD.value('(quantity)[1]', 'INT') AS Quantity

FROM

OrdersTable

CROSS APPLY

OrderDetails.nodes('/order/items/item') AS Items(OD);

* + - 1. **Run and Review the Output**: The query will generate a table with columns for product names and quantities:

Product | Quantity

------------------------

Keyboard | 1

Mouse | 2

1. Critical Thinking: In what scenarios would you prefer to store data in XML format in a relational database rather than using traditional table structures? Justify your answer with at least one example.

**Answer:**

- Storing data in XML is preferable when the information has a flexible or hierarchical structure. XML allows you to group related data together in a single container, which is especially useful if the details vary between records or have nested relationships. For example, an order with multiple items, each having unique attributes, can easily be stored in XML without needing extra tables or complex joins. This makes XML ideal for semi-structured data that doesn’t fit neatly into rows and columns, offering flexibility and simplicity for dynamic scenarios.

**Example:**

**<order>**

**<items>**

**<item>**

**<product>Notebook</product>**

**<quantity>3</quantity>**

**</item>**

**<item>**

**<product>Pen</product>**

**<quantity>5</quantity>**

**</item>**

**</items>**

**</order>**

The XML layout you provided is like a container that holds information about an order. The top level, <order>, represents the order itself. Inside it, there’s an <items> section that lists all the products included in the order. Each product is wrapped in an <item> tag, which contains two pieces of information:

* <product> tells you the name of the product (e.g., "Notebook" or "Pen").
* <quantity> shows how many of that product were ordered (e.g., "3" for Notebook and "5" for Pen).

So in this example, the order includes three Notebooks and five Pens. By using XML, all this information is kept neatly together in a single structure, which makes it easy to store and manage in a database. It’s a simple way to handle data that has multiple parts connected to one order, like items in a shopping cart.