Rod Kent M. Ito BSIT 3B

Pre Activity Database

1. Explain what XML is and how it differs from HTML. Provide one practical use case where XML would be more suitable than HTML.

* XML is a markup language designed to store and transport data with custom, user-defined tags that must be properly closed, while HTML uses predefined tags specifically for displaying content in web browsers. An excellent practical use case for XML is application configuration files, where developers need to define custom hierarchical structures of settings that can be easily parsed and modified programmatically.

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?

* SQL Server allows XML as a data type because it provides flexibility for storing semi-structured or hierarchical data that doesn't fit neatly into traditional table structures. The benefits include maintaining complex data relationships within a single field, supporting standard XML querying via XQuery and XPath, and facilitating data exchange between systems; however, drawbacks involve increased storage requirements, potentially slower query performance, and higher complexity for query writing and maintenance.

1. 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>

* The provided XML structure has a root element <order> that contains two primary child elements: <customer>

which holds customer information including nested elements for name and email, and <items> which contains multiple <item> elements, each with its own nested product and quantity elements, thereby representing a hierarchical order structure with a clear one-to-many relationship between an order and its line items.

### **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>

SELECT OrderDetails.value('(/order/customer/email)[1]', 'nvarchar(100)') 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.

First, I would use the .nodes() method to shred the XML data into rows, with each row representing an individual product item from the orders. This method essentially creates a virtual table of nodes that match the XPath expression.

Next, I would apply the .value() method to extract specific data points from each node, such as product name, quantity, price, or any other relevant product attributes stored in the XML.

I would structure my SQL query by selecting from the base table that contains the XML column, then using CROSS APPLY with the .nodes() function to generate the rowset of products. Within the SELECT statement, I would use the .value() method to pull each specific piece of data needed for the report.

If additional contextual information is needed (like order date or customer information), I would include those fields from the parent table or join to other related tables as appropriate.

Finally, I would add any necessary grouping, filtering, or sorting to format the report according to business requirements - perhaps grouping by order ID, sorting by product name, or filtering for specific date ranges.

The resulting query would transform hierarchical XML data into a traditional tabular format suitable for reporting purposes, extracting all products across all orders while maintaining their relationships to the parent orders.

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.

* Storing data in XML format in a relational database is preferable when handling hierarchical, semi-structured data with variable attributes that doesn't fit neatly into traditional tables. A prime example is healthcare systems storing patient medical records, where each patient has highly variable data (diagnoses, treatments, medications) that changes unpredictably. This approach maintains the natural hierarchical relationships while allowing flexibility as requirements evolve, without requiring schema changes for new data elements. The system retains database benefits like transaction integrity and security, while accommodating data that would otherwise require numerous nullable columns or complex table relationships.