Experiment No. : 4

Title: Demonstrate the working of XML as Database

Batch:A1 **Roll No.: 16010422013** **Experiment No.:4**

Aim: Demonstrate the working of XML as Database

Resource needed: Notepad++, Web Browser

Theory:

XML can be used to store and manage data in a way resembling a database. There are two main approaches to this:

1. Storing XML in Relational Databases (XML-enabled databases):

- Structure: Imagine tables in a relational database, but each cell can hold an entire XML document instead of just simple text.
- Queries: Use extensions to SQL like XQuery or SQL/XML to navigate and extract data from the nested XML structures within the cells.

Example: A library database might store book information (title, author, etc.) in one table, with each book's chapters and paragraphs stored as XML in a separate column. XQuery could then be used to find chapters containing specific keywords.

2. Native XML Databases:

- Structure: Data is stored directly in its native XML format, not shoehorned into tables and rows.
- Storage: Optimized data structures handle the hierarchical nature of XML documents more efficiently than relational databases.
- Queries: Specialized languages like XQuery and XPath are used to navigate and extract data based on the XML structure itself.

Example: A scientific data archive might store complex research results as XML documents with equations, figures, and annotations. XQuery could then be used to retrieve all experiments involving a specific compound.

Data: A list of employees stored in XML format:

XML
```
<?xml version="1.0"?>
<employees>
  <employee id="1">
    <name>John Doe</name>
    <department>Marketing</department>
    <salary>50000</salary>
  </employee>
  <employee id="2">
    <name>Jane Smith</name>
    <department>IT</department>
    <salary>60000</salary>
  </employee>
</employees>
```

1. XML-enabled database:
Store the XML document in a single cell of a table named "employee_data".
Use XQuery to query the data, like finding all employees with salaries over 55000:

```
SQL
SELECT employee_data.extract(value(//employee/salary[text() > 55000]), '/employees')
FROM employee_data;
```

2. Native XML database:
Store the XML document directly in the database.
Use XQuery directly to navigate the structure and find employees from the IT department:

```
XML
for $emp in /employees/employee
where $emp/department = "IT"
return $emp
```

Benefits of using XML as a database:

- Flexibility: XML can handle diverse data structures and semi-structured data well.
- Integration: Easily exchange data with other systems that use XML.
- Self-describing: XML tags provide context and meaning to the data.

Activity:

1. **Create a small XML database**
2. **Query your XML database**
3. **Transform your XML data**

Results: (Program printout with output)

Xml plain Code –

```xml
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="Order.xsl"?>
<OrderCatalog>
    <Product>
        <Product_Name>Red Printed Shirt</Product_Name>
        <Size>M</Size>
        <Price>480</Price>
    </Product>
    <Product>
        <Product_Name>Black Track</Product_Name>
        <Size>L</Size>
        <Price>520</Price>
    </Product>
    <Product>
        <Product_Name>White Hoodie</Product_Name>
        <Size>S</Size>
        <Price>660</Price>
    </Product>
    <Product>
        <Product_Name>Blue Jacket</Product_Name>
        <Size>XL</Size>
        <Price>780</Price>
    </Product>
    <Product>
        <Product_Name>Blue Denim Jeans</Product_Name>
        <Size>L</Size>
        <Price>980</Price>
    </Product>
    <Product>
        <Product_Name>Black Crop Top</Product_Name>
        <Size>L</Size>
        <Price>580</Price>
    </Product>
    <Product>
        <Product_Name>Pink Kurti</Product_Name>
        <Size>XL</Size>
        <Price>680</Price>
    </Product>
    <Product>
        <Product_Name>Black Hoodie</Product_Name>
        <Size>M</Size>
        <Price>380</Price>
    </Product>
    <Product>
        <Product_Name>Black Jeggings</Product_Name>
        <Size>XL</Size>
        <Price>400</Price>
    </Product>
    <Product>
        <Product_Name>Black Skirt</Product_Name>
        <Size>L</Size>
        <Price>580</Price>
    </Product>
</OrderCatalog>
```

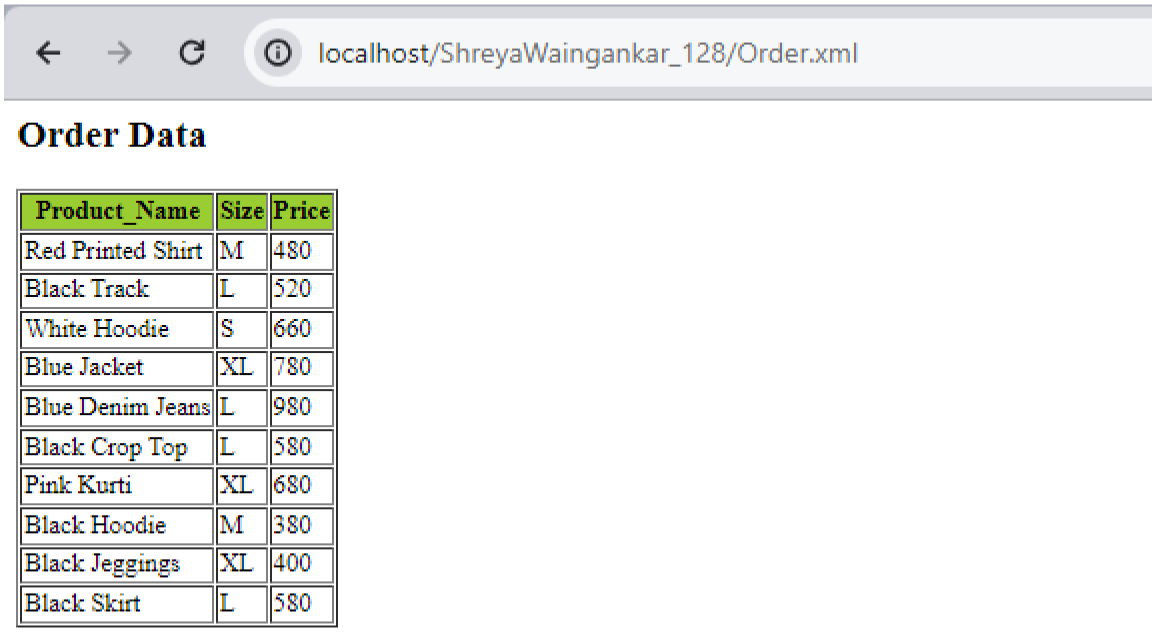
XSL Plain Code –

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
    <body>
      <h2>Order Data</h2>
      <table border="1">
        <tr bgcolor="#9acd32">
           <th>Product_Name</th>
           <th>Size</th>
           <th>Price</th>
        </tr>
        <xsl:for-each select="OrderCatalog/Product">
          <tr>
            <td><xsl:value-of select="Product_Name"/></td>
            <td><xsl:value-of select="Size"/></td>
            <td><xsl:value-of select="Price"/></td>
          </tr>
        </xsl:for-each>
      </table>
    </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

Result :



localhost/ShreyaWaingankar_128/Order.xml

Order Data

Product_Name	Size	Price
Red Printed Shirt	M	480
Black Track	L	520
White Hoodie	S	660
Blue Jacket	XL	780
Blue Denim Jeans	L	980
Black Crop Top	L	580
Pink Kurti	XL	680
Black Hoodie	M	380
Black Jeggings	XL	400
Black Skirt	L	580

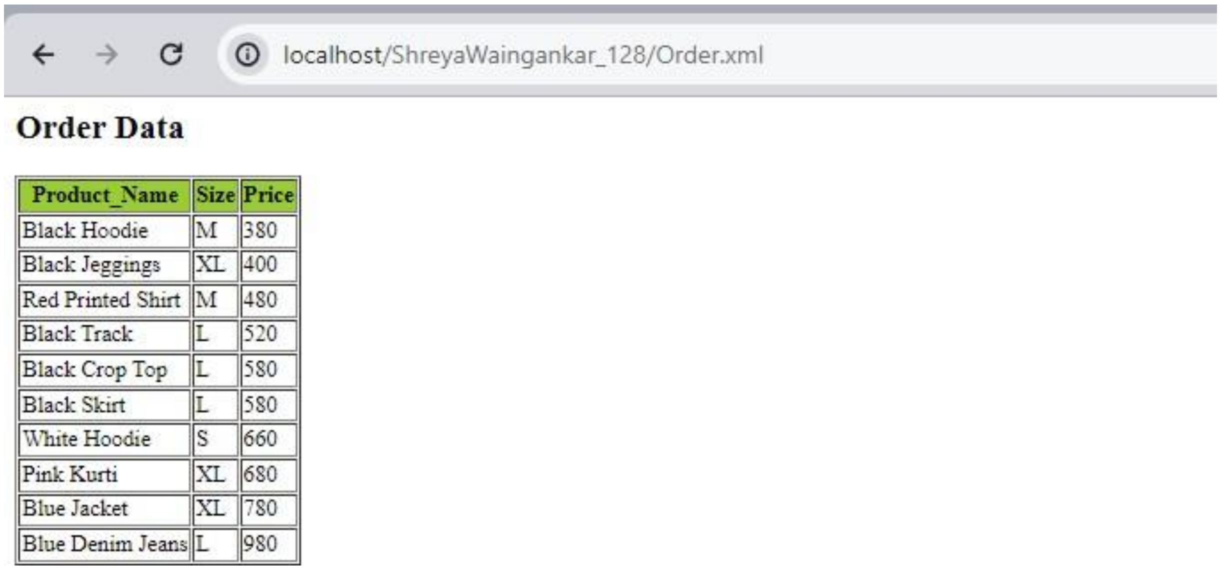
Query for Sorting by Price –
Code –

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
    <body>
      <h2>Order Data</h2>
      <table border="1">
        <tr bgcolor="#9acd32">
           <th>Product_Name</th>
           <th>Size</th>
           <th>Price</th>
        </tr>
        <xsl:for-each select="OrderCatalog/Product">
          <xsl:sort select="Price" data-type="number"/>
          <tr>
            <td><xsl:value-of select="Product_Name"/></td>
            <td><xsl:value-of select="Size"/></td>
            <td><xsl:value-of select="Price"/></td>
          </tr>
        </xsl:for-each>
      </table>
    </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

Result :


localhost/ShreyaWaingankar_128/Order.xml

Order Data

Product_Name	Size	Price
Black Hoodie	M	380
Black Jeggings	XL	400
Red Printed Shirt	M	480
Black Track	L	520
Black Crop Top	L	580
Black Skirt	L	580
White Hoodie	S	660
Pink Kurti	XL	680
Blue Jacket	XL	780
Blue Denim Jeans	L	980

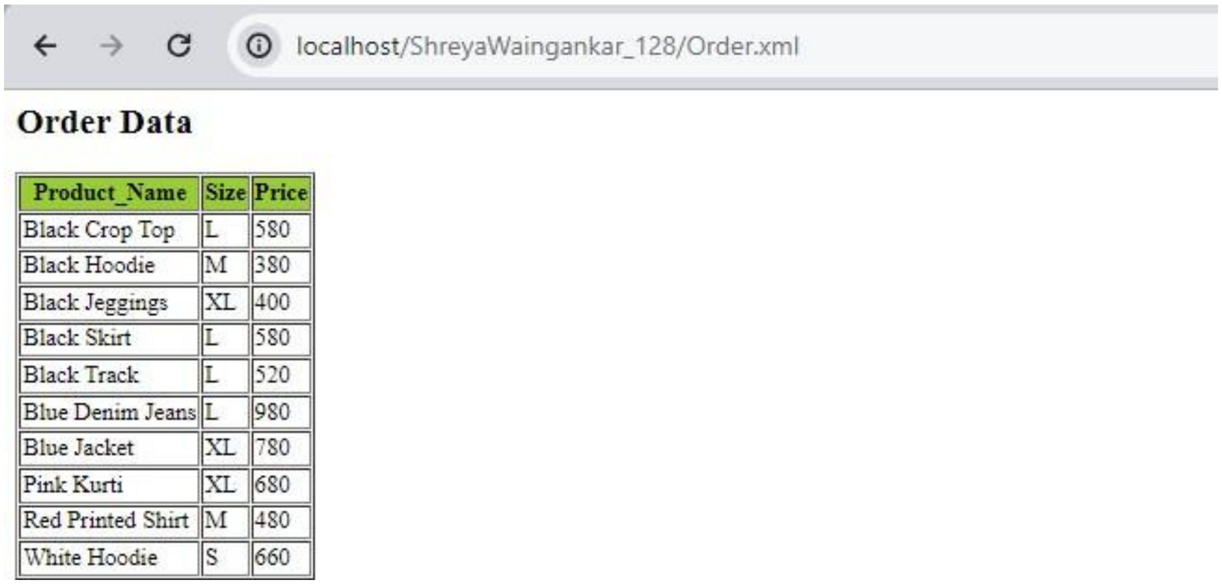
Query for sorting Alphabetically Product Name:
Code-

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
    <body>
      <h2>Order Data</h2>
      <table border="1">
        <tr bgcolor="#9acd32">
           <th>Product_Name</th>
           <th>Size</th>
           <th>Price</th>
        </tr>
        <xsl:for-each select="OrderCatalog/Product">
          <xsl:sort select="Product_Name"/>
          <tr>
            <td><xsl:value-of select="Product_Name"/></td>
            <td><xsl:value-of select="Size"/></td>
            <td><xsl:value-of select="Price"/></td>
          </tr>
        </xsl:for-each>
      </table>
    </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

Result –



localhost/ShreyaWaingankar_128/Order.xml

Order Data

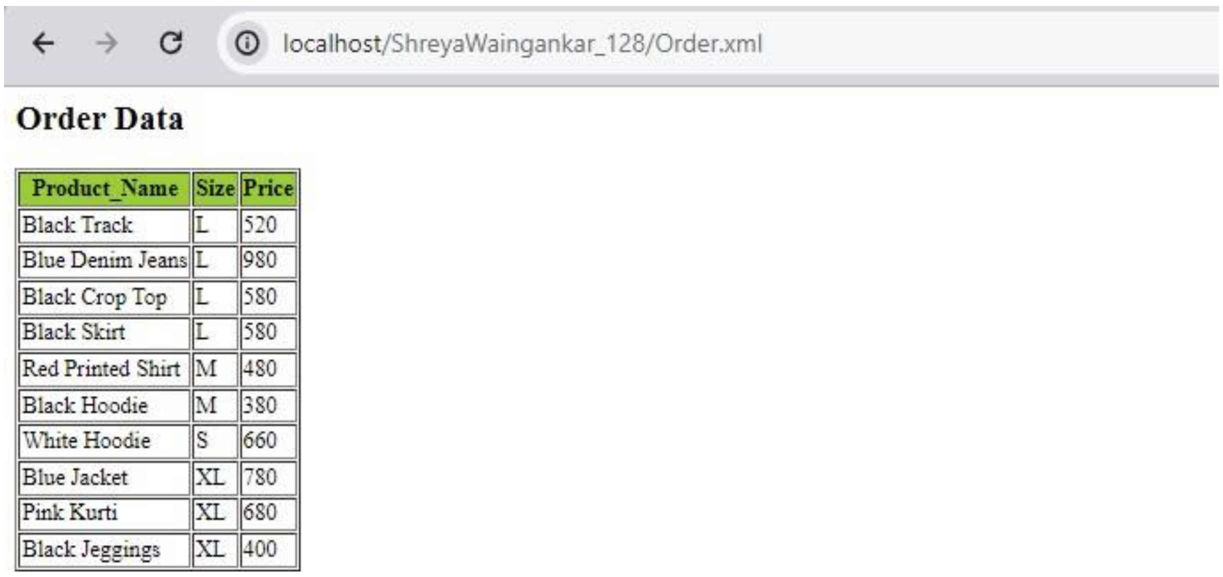
Product_Name	Size	Price
Black Crop Top	L	580
Black Hoodie	M	380
Black Jeggings	XL	400
Black Skirt	L	580
Black Track	L	520
Blue Denim Jeans	L	980
Blue Jacket	XL	780
Pink Kurti	XL	680
Red Printed Shirt	M	480
White Hoodie	S	660

Query for Size Wise Sort :
Code –

```
<xsl:template match="/">
  <html>
    <body>
      <h2>Order Data</h2>
      <table border="1">
        <tr bgcolor="#9acd32">
           <th>Product_Name</th>
           <th>Size</th>
           <th>Price</th>
        </tr>
        <xsl:for-each select="OrderCatalog/Product">
          <xsl:sort select="Size"/>
          <tr>
            <td><xsl:value-of select="Product_Name"/></td>
            <td><xsl:value-of select="Size"/></td>
            <td><xsl:value-of select="Price"/></td>
          </tr>
        </xsl:for-each>
      </table>
    </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

Result –



localhost/ShreyaWaingankar_128/Order.xml

Order Data

Product_Name	Size	Price
Black Track	L	520
Blue Denim Jeans	L	980
Black Crop Top	L	580
Black Skirt	L	580
Red Printed Shirt	M	480
Black Hoodie	M	380
White Hoodie	S	660
Blue Jacket	XL	780
Pink Kurti	XL	680
Black Jeggings	XL	400

Questions:

1. What is the difference between an element and an attribute in XML?
2. What is the role of a DTD in XML?
3. How can you comment on a section of code within an XML document?

Ans 1)
In XML, both elements and attributes are used to represent data, but they serve different purposes and have distinct characteristics:

1. **Elements**:
   - Elements are the building blocks of XML documents and represent the structure and content of the data.
   - They are enclosed within angle brackets < > and consist of a start tag, content, and an end tag. For example: **<book>Title of the book</book>**.
   - Elements can have child elements, allowing for hierarchical organization of data.
   - Elements can contain text, other elements, or a combination of both.
2. **Attributes**:
   - Attributes provide additional information about elements and are used to describe or qualify the element they belong to.
   - Attributes appear within the start tag of an element and consist of a name-value pair. For example: **<book category="fiction">Title of the book</book>**.
   - Attributes provide metadata or properties associated with the element, such as identifiers, formatting instructions, or other descriptive information.
   - Unlike elements, attributes cannot have child elements or contain complex data structures. They are typically used for simple, single values.

In summary, while elements define the structure and content of XML documents and can contain other elements, attributes provide additional information about elements and are used for metadata or descriptive purposes.

Ans 2)

A Document Type Definition (DTD) in XML serves several important roles:

1. **Defines Document Structure**:
    - A DTD specifies the structure and rules for the elements and attributes that can appear in an XML document.
    - It defines the allowable elements, their arrangement (hierarchy), attributes, and the relationships between them.
2. **Enforces Document Validity**:
    - The DTD acts as a schema or blueprint for XML documents, ensuring that they adhere to a predefined structure.
    - It validates XML documents against the rules specified in the DTD, checking whether the document conforms to the expected structure and content.
3. **Promotes Interoperability**:
    - By providing a standardized way to define document structure and content, DTDs facilitate interoperability between different systems and applications.
    - DTDs enable data exchange and integration between systems that agree on a common document structure defined by the DTD.
4. **Facilitates Document Authoring and Parsing**:
    - DTDs aid document authors by providing clear guidelines on the allowed elements and attributes, helping ensure consistency and correctness in document creation.
    - XML parsers use the DTD to validate XML documents during parsing, identifying and reporting errors or inconsistencies in the document structure.
5. **Supports Document Versioning**:
    - DTDs can evolve over time to accommodate changes in document structure or requirements.
    - Document versioning can be managed through version-specific DTDs, allowing for backward compatibility while introducing new elements or rules in updated versions of XML documents.

Overall, DTDs play a crucial role in defining, validating, and ensuring the consistency and interoperability of XML documents across different systems and applications.

Ans 3)

In XML, you can add comments using the **<!--** and **-->** delimiters. Anything between these delimiters is considered a comment and is ignored by XML parsers during document processing. Comments can span multiple lines and can be placed anywhere within the XML document where text content is allowed.

Here's an example of how to add comments in an XML document:

```
<root>
  <!-- This is a comment -->
  <element>Some content</element>
  <!--
    This is a multi-line comment.
    It can span across multiple lines.
  -->
  <another_element>More content</another_element>
</root>
```

In the example above:

- **<!-- This is a comment -->** is a single-line comment.
- **<!-- ... -->** is a multi-line comment spanning several lines.

Comments in XML are useful for documenting the purpose of elements or attributes, providing context for future modifications, or temporarily excluding parts of the document from processing without removing them entirely.

Outcomes:
CO-2: Create Web pages using HTML 5 and CSS.

Conclusion: (Conclusion to be based on the outcomes achieved)
Through this experiment,we learnt how create an XML document based Our mini project which is travel. I created a database wherein we have customer name, destination, number-of-days, arrival-date, destination-date. And we applied 3 queries to sort the data. We also learnt about the difference between an element and an attribute in XML, the role of DTD in XML and how to comment on a section.

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of faculty in-charge with date

References:

Books/ Journals/ Websites:

- "Web technologies: Black Book", Dreamtech Publications
- http://www.w3schools.com