Web Technology Lab Works (Part-4)

33. Validating XML documents against internal and external DTDs

a) Consider an Employee Management System whose structure may look like this:

```
Employees
     ├── Employee (id)
    │     ├── Name
    │     ├── Position
    │     ├── Department
    │     ├── Salary (currency)
    │     ├── ContactDetails
    │          ├── Email
    │          ├── Phone
```

Now, create an **internal DTD** with following rules, create XML document reflecting above structure **validate the document against the DTD**. [Use any XML validator and show the output]

DTD Rules:

- <Employees> (root) must contain one or more <Employee> elements.
- Each <Employee> has:
  - – A required id attribute (<!ATTLIST Employee id CDATA #REQUIRED>).
  - – The child elements <Name>, <Position>, <Department>, <Salary>, and <ContactDetails>.
  - – <Salary> has an optional currency attribute (<!ATTLIST Salary currency CDATA #IMPLIED>).
  - – <ContactDetails> is a nested structure containing <Email> and <Phone>.

b) Consider a Library Management System whose structure may look like below:

```
Library
├── Book (isbn)
│     ├── Title
│     ├── Author
│     ├── Genre
│     ├── PublishedYear
│     ├── Price (currency)
│     ├── Copies
│          ├── Copy (copyID)
│               ├── Availability (status)
```

Now, create an **external DTD** with following rules, create XML document reflecting above structure **validate the document against the DTD**. [Use any XML validator and show the output]

DTD Rules:

- <Library> (root) must contain one or more <Book> elements.
- <Book> must have:
  – A required isbn attribute (<!ATTLIST Book isbn CDATA #REQUIRED>).
  – Elements <Title>, <Author>, <Genre>, <PublishedYear>, <Price>, and <Availability>.
  – <Author> can appear one or more times
  – <Price> has an optional currency attribute (<!ATTLIST Price currency CDATA #IMPLIED>).
  – <Copies> contains one or more <Copy> elements. <Copy> must have a unique copyID attribute and contain an <Availability> element.
  – <Availability> is an empty element with a required status attribute, which can only be "Available" or "Checked Out".

34. Validating XML documents against XML Schemas

a) Create an XML file student.xml containing student details such as name, rollNumber, age, and email. Define an XSD file student.xsd to validate the XML file. The XML file should have at least 3 student records and XSD schema should enforce the following rules:

– rollNumber should be an integer.
– age should be between 18 and 30.
– email should follow a valid email format.

Now, **validate the XML file using the XSD**. [Use any XML validator and show the output]

b) Create an XML file **library.xml** representing a library system containing a list of books. Each book should have title, author, ISBN, price, and publisher information. The XML file should have at least 5 books and XSD file, **library.xsd** should define following validation rules:

– A complex type for book, ensuring ISBN is exactly 13 digits long.
– A restriction for price to be a positive decimal number.
– A mandatory publisher element with name and yearEstablished (a year between 1900 and the current year).

Now, **validate library.xml against library.xsd**. [Use any XML validator and show the output]

c) Create an XML file employees.xml containing details about employees in a company. Each employee should have attributes for id and department, along with elements for name, salary, and joiningDate. Define an XSD file employees.xsd with following rules:

– id as a required attribute, restricted to numeric values.
– department as an attribute restricted to values: HR, IT, Finance, Sales.
– salary should be a positive number.
– joiningDate should follow the YYYY-MM-DD format.

Now Create an XML file with at least 3 employee records and **validate** it.

[Use any XML validator and show the output]

d) Create an XML file orders.xml to store customer orders. Each order should have customer details and a list of purchased items. Define an XML structure where:

– A customer has name, phone, and email.

– An order contains multiple items, each having itemName, quantity, and price.

Write an XSD file orders.xsd with:

– Reusable complex types for customer and item.

– Constraints ensuring quantity is a positive integer. –
price should be a positive decimal number.

Now, **validate the XML file against the schema.** [Use any XML validator and show the output]

35. Understanding XSL and XSLT Transformation

a) Create an XML file named students.xml that stores details of at least 5 students (name, roll number, grade, and address). Now write an XSLT stylesheet that transforms this XML into an HTML table displaying the student information.

b) Enhance the XSLT from (a) to highlight students with a grade 'A' using a different background color in the HTML table row.

36. Using XPath Expressions

Considering the students.xml file created above, Write XPath expressions to retrieve:

– All student names.
– Students who have a grade 'A'.
– The address of the student whose roll number is '103'.

Now, Validate and test your XPath expressions using an online XPath tester or XML tool.

37. Using XQuery

a) Create an XML file books.xml with a collection of at least 6 books. Each book should include title, author, price, publisher, and year. Now Write XQuery expressions to:

– Display all book titles.

– Find books published after the year 2015. – List
books by a specific publisher (e.g., "O'Reilly")

b) Considering the books.xml file created in (a), write an XQuery using the FLOWR expression to:

i. Find and return books with a price less than 500.
ii. Sort books by year in descending order

NOTE: For (i), display the result in following XML format

```
<affordableBooks>
  <book>
    <title>Book Title</title>
    <price>450</price>
  </book>
</affordableBooks>
```

and for (ii) display with all nested tag of <book>

c) Using the books.xml file you created earlier, write an XQuery expression that transforms the XML data into a formatted HTML document. Your goal is to create a webpage that displays the list of books in a clean and readable format. The HTML output should begin with a <html> tag, and include a <head> section containing a <title> tag with the text “Book List”. The <body> section should start with an <h1> heading labeled “Books”. Below the heading, include an unordered list (<ul>) where each book is displayed as a list item (<li>). Each list item should show the book’s title, followed by the word “by”, then the author's name, and finally the year of publication in parentheses. For example: “Homo Deus by Yuval Noah Harari (2015)”. This task demonstrates how XQuery can be used to dynamically generate HTML content from structured XML data.

[use necessary tools XQuery Testing Tools for above tasks]

38. Server-Side Scripting using PHP

a) Write a program in PHP having one indexed array and another associative array. Now use foreach loop to display all the elements of both arrays in unordered list in webpage.

b) Write a program in PHP to read the full fname and age values from user and display “welcome <full_name>” [replace <full_name> with exact name entered through form’]. If the age value entered is even display the multiplication table of that value and if age is odd display the number of vowel and consonant characters in ‘full_name’ value.

c) Create a simple registration form and apply server-side validation using PHP. (You must apply some validation rules and at least one of them should be validated using regular expression in PHP)

d) Write a PHP program to demonstrate login and logout features using session. Also include "remember me" feature using cookies in PHP. [use dummy username and password for demonstration]

e) Write a PHP program to demonstrate how Class and Object are crated in PHP.

f) Create a CRUD application using PHP and MySQL.

39. Working with AJAX

a. WAP using AJAX to read the contents of a text file and display on a webpage
b. WAP using AJAX with PHP to work with GET request.
c. WAP using AJAX with PHP to work with POST request.
d. WAP to demonstrate AJAX with jQuery using ajax() method.
e. WAP using AJAX, PHP and MySQL to show the records of item selected from dropdown

a. Create a web application to demonstrate "Ajax Live Database Search" using PHP, AJAX, jQuery and MySQL. [*Report can be printed for this task]*

40. Creating PHP based web application using Navbar and Event handling

Create a basic horizontal navigation bar with a dark background color and change the background color of the links when the user moves the mouse over them also add a dropdown menu that allows the user to choose an option from a list. One of the menus should be linked with a web page that simulates a product ordering system. The system should use different HTML event attributes (onclick, onchange, onblur) and handle the input data using PHP on form submission.

Your web page should include the following elements and behavior:

- Product Selection (onchange Event)

  A dropdown (<select>) to choose a product (e.g., Laptop, Mobile, Tablet).

  When a product is selected, display its unit price next to the dropdown using JavaScript (onchange event).

- Quantity Input (onblur Event)

  A number input for entering the quantity.

  When the user moves out of the quantity field (onblur), validate that the value is a positive number using JavaScript. Show an alert if it's invalid.

- Customer Name (onblur Event)

  A text field to enter the customer's name.

  Use onblur to check that the field is not left empty. If it is, display an alert message.

- Calculate Price Button (onclick Event)
  A "Calculate Total" button that uses the onclick event to submit the form.
- After submission, PHP should process the selected product and quantity, and display:

    The customer's name

    The selected product

    Quantity

    Unit price

    Total Price

  [*Report can be printed for this task]*

41. Being familiar with PHP frameworks and CMS

[Report can be printed for these tasks]

a) Install latest version of CodeIgniter and use it to create a web application using at least one CRUD operation

b) Install latest version of Laravel and create a MVC based web application using Eloquent ORM to perform CRUD operation.

c) Install latest version of WordPress and create a blogging website.
[list the steps followed and screenshots of output except installation parts]