МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

ГНАТОВСЬКА Г.А.

КОНСПЕКТ ЛЕКЦІЙ

з дисципліни

«XML-ТЕХНОЛОГІЇ»

для студентів 4 курсу денної форми навчання

спеціальності 122 – «Комп'ютерні науки»

Дозволено до використання у навчальному процесі в електронному вигляді за рішенням методичної ради Одеського державного екологічного університету (протокол № ___ від _____)"

Одеса, 2019

УДК519.81(075.8)

Конспект лекцій з дисципліни «XML-ТЕХНОЛОГІЇ» для студентів 4 курсу денної форми навчання. Спеціальності 122 – “Комп’ютерні науки”

Укладачі:
ГНАТОВСЬКА Г.А., к. т. н., доцент кафедри інформатики

Розглядає питання використання теоретичних та практичних відомостей про синтаксис, основні конструкції та застосування мови XML для подальшого застосування при обміні інформацією у розподілених інформаційних системах.

ЗМІСТ

Вступ 6
1 ОГЛЯД XML-ТЕХНОЛОГІЙ 7
1.1 Сучасний стан розвитку web-технологий 7
1.2 Призначення та застосування XML 10
1.3 Історія XML 12
1.4 XML – універсальний стандарт для документів 13
1.5 Платформа XML та ii стандарти 16
1.6 Можливості XML-платформи та застосування стандартів XML 19
1.7 Застосування XML-платформи в технологіях баз даних 22
2 ОСНОВИ МОВИ XML 25
2.1 Документ, як одиниця інформації XML 25
2.2 Структура XML-документу 27
2.3 Декларація XML-документа 29
2.4 Пролог XML-документа 30
2.5 Елементи XML-документа 31
2.6 Атрібути XML-документа 34
2.7 Коментарі XML-документа 35
2.8 Базові правила побудови XML документа 35
3 ТЕХНОЛОГІЇ ОБРОБКИ ДАНИХ В ФОРМАТІ XML 38
3.1 XML-схема 38
3.2 DTD-схема 39
3.3 Визначення XML schema 42
4 МОВА ВИЗНАЧЕННЯ СХЕМ XSD 44
4.1 Призначення XSD 44
4.2 Простори імен 45
4.3 Прості елементи 46
4.4 Прості типи даних в схемах XSD 48
4.5 Елементи складних типів 49
4.6 Обмеження входжень у схемах XSD 50
4.7 Опис атрибутів 52
4.8 Використання XSL-таблиць стилів 55
5 МОВА ЗАВДАННЯ ШЛЯХІВ XPath 68
5.1 Путі розташування в XPath 69
5.2 Прості і складові маршрути 70
5.3 Осі вибірки XPath 71
5.4 Контекст XPath-виразів 74
5.5 Базові вираження XPath 75
ЛІТЕРАТУРА 77

ВСТУП

Дисципліна «XML-технології» викладається у напрямку підготовки Спеціальності 122: «Комп'ютерні науки» входить до складу вибіркової частини навчального плану підготовки бакалаврів. Викладається відповідно до робочого навчального плану підготовки бакалаврів.

Основною метою даної дисципліни є вивчення принципів реалізації роботи з даними за допомогою технології XML в програмних додатках, а також розгляд технологій обробки, форматування і перетворення даних в форматі XML.

Завдання дисципліни: в результаті вивчення дисципліни студенти повинні знати:

- Розробляти XML-документи;
- Реалізовувати обробку даних в форматі XML, в тому числі при розробці програмних додатків в середовищі Microsoft .NET Framework;
- Створювати DTD і XML-схеми для реалізації можливості валідації XML-документів;
- Використовувати можливості XPath при роботі зі структурою XML-документів.

1 ОГЛЯД XML-ТЕХНОЛОГІЙ

1.1 Сучасний стан розвитку web-технологий

Створення Всесвітньої павутини – World Wide Web – WWW, або просто Web, стало одним з найбільших науково-технічних досягнень останнього десятиліття XX століття, основою цілого ряду нових інформаційних технологій, що мають досить відчутні соціально-економічні наслідки. Ідеї проекту, який виник в стінах CERN (Європейський центр ядерних досліджень, Женева) в кінці 80-х років, в короткі терміни реалізувалися в глобальну і нескінченно масштабуємо розподілену гіпермедійну систему з прозорим для користувача розподілом і неоднорідністю ресурсів. Кількість користувачів і обсяг представлених в ній інформаційних ресурсів продовжують надзвичайно швидко нарощуватися. При цьому можливий вільний доступ до більшості інформаційних ресурсів Web в будь-який момент часу. З іншого боку, за кілька років бурхливого розвитку потенціал подальшого вдосконалення технологій існуючої версії Web (званої далі Web-1) виявився в значній мірі вичерпаним. Цьому сприяли недоліки мови HTML (основного структуроутворюючого засобу гіпермедійних інформаційних ресурсів, представлених в Web), а також обмежені функціональні можливості середовища підтримки цієї мови в Web.

Ці недоліки і обмеження мови полягають в наступному.

По-перше. Для HTML-документів не підтримуються метадані, що описують структурні, семантичні та інші їх властивості. На момент розробки мови HTML потреби в цих засобах не було, так як вона була орієнтована не на структурну розмітку документів, а на опис формату та їх подання на екрані. Відсутність підтримки метаданих для HTML-документів не дозволяє здійснювати перевірку цілісності їх структури і змісту. З цієї ж причини не може буди здійснено ефективний пошук необхідної користувачеві інформації в величезному накопиченому обсязі інформаційних ресурсів Web, а також виконання інших операцій з обробки інформаційних ресурсів. В умовах відсутності підтримки метаданих для

виконання пошукових операцій відповідні сервіси Web реалізують тільки техніку контекстного пошуку. Пошук документів в Web з урахуванням властивостей їх структурних компонентів є неможливий. Все це призводить до досить високого рівня «шуму» в результатах пошуку. Нарешті, без підтримки метаданих в середовищі Web неможлива ефективна інтеграція інформаційних ресурсів, які підтримуються в цьому середовищі і в інших взаємодіючих з Web середовищах. Технічно засоби мови HTML дозволяють інтегрувати в середу Web ресурси баз даних, великі архіви текстових документів, різні мультимедійні ресурси. Але ці чужорідні для гіпертексту ресурси хоча і стають доступними користувачу, залишаються, з точки зору їх семантики, для середовища Web «чорним ящиком». Така інтеграція зводиться по суті лише до забезпечення подання або відображення «зовнішніх» ресурсів за допомогою Web.

По-друге. У специфікації не передбачені засоби розширення функціональності HTML-документів. Іншими словами, HTML є закритою мовою, що не дозволяє користувачу доповнювати мову новими тегами при необхідності. Наслідком закритості мови HTML є обмежені можливості структурування документів, адекватно до потреб користувачів і відображати хоча б найпростіші аспекти семантики, що містяться в даних. Закритий характер мови призводить також до необхідності періодичного перегляду версій стандарту HTML для розширення його функціональності шляхом додавання нових тегів атрибутів тегів.

Перераховані недоліки мови зажадали подальшому розвитку інформаційного середовища Web шляхом додавання в неї технологій, заснованих на сучасних методах управління даними, пройшли випробування часом в технологіях баз даних і текстових пошукових системах. Основу цих методів складають моделі даних, засновані на явному поданні та підтримки метаданих, що забезпечують використання техніки пошуку документів на основі їх змісту.

З метою вирішення цих проблем консорціум W3C веде активну діяльність, спрямовану на радикальний перегляд основ Web-технологій, які зачіпають всі три базові елементи первинного проекту WWW, на яких побудована діюча сьогодні реалізація Web. Цими базовими елементами, як

відомо, є: мова гіпертекстової розмітки HTML, універсальний локатор ресурсів URL, протокол передачі гіпертекстових ресурсів HTTP. В рамках виконання цієї роботи створено ядро і триває процес формування незалежного від області додатків комплексу засобів, що базується на мові розмітки XML. Цей комплекс, який отримав назву XML-платформи, служить для опису і обробки інформаційних ресурсів Web, покликаної стати основою нового покоління Web, званого далі Web-2.

Нові технології Web базуються на відкритому для розширення комплексі стандартів, що становить XML-платформу і визначає функціональні можливості для доступу до інформаційних ресурсів Web і їх уявленню. Важливе місце в XML-платформі займають стандарти представлення метаданих, що описують структурні і семантичні властивості XML-документів, що дозволяє в перспективі вести мову про «семантичний Web». Завдяки введенню метаданих та стандартизації засобів їх опису відкрилися можливості для більш ефективної інтеграції XML-документів, заснованих на використанні семантичної інформації. Одна з принципових установок даної діяльності полягає в неодмінному забезпеченні наступності нової платформи з Web-1, що дозволить зберегти можливість використання і надалі величезних інформаційних ресурсів, представлених засобами мови HTML.

Незважаючи на наведені недоліки, HTML стала гігантським стрибком в еволюції мов розмітки, оскільки саме вона стала основою для переходу до наступного етапу розвитку Web, заснованого на XML-платформі.

Розмітка (*markup*) – це інформація, що додається до документа для полегшення сприйняття його змісту, т. к. визначає його частини і їх співвідношення один з одним. Іншими словами, *розмітка* – це засіб структурування документа.

Наприклад, при читанні будь-якої друкованої продукції частини тексту розрізняються по проміжків між ними і їх положенню на сторінці, а також шляхом використання різних шрифтів для заголовків різних рівнів. Це і є розмітка, але ця розмітка орієнтована на сприйняття тексту людиною, в той час як мова HTML, спрямована на структурування

документа для машинної обробки замість пробілів використовує інші символи. Текст без проміжків між розділами і з єдиним стилем був би просто нагромадженням символів, і витяг з нього інформації зажадало б великої роботи. Машинна програма не змогла б зробити навіть цього, оскільки володіє лише елементарними можливостями пошуку за шаблоном. Розмітка важлива в електронних документах, тому що вони обробляються комп'ютерними програмами. Якщо в документі немає міток або кордонів, програма не знатиме, як відрізнити один фрагмент тексту від іншого. При відсутності розмітки програмі доведеться працювати з усім документом як з безструктурним, що різко обмежує можливість обробки його вмісту.

Мова розмітки (markup language) – це спосіб іменування та виділення частин документа за допомогою тегів, що розміщуються в тексті документа. У документах, закодованих із застосуванням мови розмітки, стало можливим здійснювати редагування, форматування і пошук за допомогою різних програм, оскільки його теги дозволяли отримувати зміст даних. Бернерс-Лі і Берглунд створили версію документа SGML для гіпертекстових документів, який був компактним і ефективним, і назвали її HTML. Ця мова розмітки зробила простим написання програмного забезпечення і ще простіше – кодування документів. HTML швидко вийшов за межі лабораторії і відправився завойовувати світ. Але в теоретичному плані HTML була кроком у зворотному напрямку. З метою досягнення простоти, яка зробила цю мову дійсно корисною на практиці, довелося пожертвувати деякими принципами узагальненого кодування. Основний недолік мови HTML – це використання тегів, переважно орієнтованих на уявлення, що ускладнює їх використання в додатках.

1.2 Призначення та застосування XML

XML (англ. EXtensible Markup Language – розширювана мова розмітки) – універсальна мова розмітки, призначена для опису структурованих даних, обміну інформацією між програмами і створення спеціалізованих мов розмітки (XML-словників).

Метою створення XML було забезпечення сумісності при передачі структурованих даних між різними системами обробки інформації, особливо при передачі таких даних через Інтернет.

XML означає Extensible Markup Language, з акцентом на markup (розмітка). Мова XML дозволяє створювати текст і розмічати його за допомогою обрамляють тегів, перетворюючи кожне слово, пропозицію або фрагмент в ідентифіковану інформацію, що може сортуватися.

Створювані файли, або екземпляри документа, складаються з елементів (тегів) і тексту, причому елементи допомагають правильно розуміти документ при читанні на папері, або навіть обробляти його в електронному вигляді. Чим більше описаних елементів, тим більше частин документа можна ідентифікувати. З перших днів існування розмітки одне з її переваг полягає в тому, що в разі втрати комп'ютерної системи, роздруковані дані все одно залишаються читабельними завдяки тегами.

Засобами мови XML можливо створювати свої власні елементи, що дозволяє точно представляти фрагменти даних. Документи можна не просто розділяти на абзаци і заголовки, але і виділяти будь-які фрагменти всередині документа. Щоб це було ефективно, потрібно визначити кінцевий перелік своїх елементів і дотримуватися його. Елементи можливо визначати в Описі типу документа (Document Type Definition – DTD) або в схемі.

Компанія Microsoft була однією з перших організацій, яка стала активно використовувати XML в якості однієї з провідних технологій для обміну даними. В рамках її діяльності була створена група технологій, що підтримують різні API для роботи з форматом XML в залежності від мови розробки.

XML – гнучка мова розмітки, яку створюють розробники самі. Тому насамперед варто визначити XML-теги, які описують дані, а не дотримуватися стандартного набору тегів, як це робиться в HTML. Гнучкість дозволяє фірмам і компаніям створювати свої стандартні теги для опису даних, характерні для їх сфери діяльності та обраній предметній області.

XML, безсумнівно, входить в обойму найбільш перспективних технологій WWW, чим пояснюється інтерес, який приділяється йому і корпораціями-розробниками, і широкою публікою. Перш ніж перейти до його опису, видається доречним обговорити причини його появи та подальшого бурхливого розвитку. Для цього поглянемо ті проблеми WWW, які повинні бути вирішені засобами нового покоління Веб-технологій.

1.3 Історія XML

Мови розмітки пройшли шлях від перших форм, які створювались компаніями і держустановами, до Стандартної мови узагальненої розмітки (Standard Generalized Markup Language – SGML), гіпертекстова мова розмітки (Hypertext Markup Language HTML) і в кінцевому підсумку до XML. SGML може здатися складною, а HTML (який, по суті, спочатку був просто набором елементів) виявився недостатньо потужним для ідентифікації інформації. XML розроблялась як проста в застосуванні і зручна для розширення мова розмітки.

Роком народження XML можна вважати 1996 рік, в кінці якого з'явився чорновий варіант специфікації мови, а у 1998 році ця специфікація була затверджена. А почалося все з появи в 1986 році мови SGML (англ. Standard Generalized Markup Language – стандартна узагальнена мова розмітки) – гнучка і всеосяжна мета-мова для створення мов розмітки. Ця мова настільки універсальна. Ціною універсальності є складність і, як наслідок, дорожнеча застосування мови в практичних розробках. У підсумку більшість можливостей SGML просто незатребувані. Однак SGML знайшла своє застосування в якості основи для створення інших мов розмітки, зокрема HTML – більш проста і легка в освоєнні мова. Але, у міру зростання кількості і зміни якості документів в глобальній мережі Інтернет, росли і вимоги до них, і простота HTML перетворилася в її головний недолік. Обмеженість кількості тегів і повну байдужість до структури документа спонукали розробників в особі консорціуму W3C до створення такої мови розмітки, яка була би не настільки складною, як

SGML, і не настільки примітивною, як HTML. В результаті на світ з'явилася мова XML, що поєднує в собі простоту HTML, логіку SGML і задовольняє вимогам Інтернет.

XML – це незалежний стандарт, який підтримує консорціум World Wide Web Consortium (W3C). W3C був створений Массачусетським технологічним інститутом (MIT) спільно з центром CERN в Женеві за підтримки Європейської комісії. W3C користується великим авторитетом в галузі IT. Компанія Software AG також є активним членом цього консорціуму. XML (eXtensible Markup Language) – це спрощений діалект мови SGML, призначений для опису ієрархічних структур даних в World Wide Web.

1.4 XML – універсальний стандарт для документів

Оскільки XML це мета-мова, то вона визначається на двох різних рівнях: сам стандарт XML, який підтримується консорціумом World Wide Web Consortium (W3C), які продовжують його подальшу розробку, та конкретні додатки XML, які створюються незалежними групами користувачів.

XML – це універсальна та незалежна від платформи мова розмітки, яку можна використовувати для подання ієрархічних даних і уніфікації інформації, що передається.

XML – це мета-мова, за допомогою якої можливо визначати інші мови. Дійсно, шляхом створення нових тегів і визначення нових структур за допомогою цих тегів розробники фактично створюють нові мови з їх власним синтаксисом і семантикою. Ця мова використовується як засіб для опису граматики інших мов і контролю за правильністю складання документів. Тобто сама по собі XML не містить ніяких тегів, призначених для розмітки, вона просто визначає порядок їх створення.

XML-документи можуть служити проміжним форматом для передачі інформації від одного додатка до іншого (наприклад, як результат запиту до бази даних), тому їх вміст іноді генерується і обробляється програмами автоматично.

Визначимо основні характеристики мови XML: пропонує метод структуризації файлу у вигляді текстового файлу; схожа на HTML та достатньо гнучка; утворює ціле сімейство технологій; вільна від ліцензійних відрахувань, платформово-незалежна, має широку підтримку.

XML пропонує метод структуризації файлу у вигляді текстового файлу. Дуже часто трапляються завдання, коли необхідно дані однієї програми перемістити в іншу, але формати даних у цих програмах не збігається, а отже, і дані перемістити не можливо. Мова XML забезпечує таку можливість, оскільки будь-яке її застосування може працювати текстовими документами, і будь-яка людина може прочитати і зрозуміти текст.

XML дозволяє зберігати в текстовому форматі структуровані дані. Отже, XML – це набір правил для створення текстових форматів, простих для обробки комп'ютерами різних типів. Отримані текстові файли структуровані таким чином, що вони: точно виражені; розширені; платформово-незалежні.

Для розроблення XML-файлів можливо використовувати будь-який текстовий редактор. XML-документи, як правило, мають розширення *.xml, але спеціалізовані діалекти, створені в рамках технологій XML, можуть мати розширення:

.xls – файли розширеної таблиці стилів (Extensible Stylesheet Language);

*.xsd – визначення розширеної схеми (Extensible Schema Definition);

*.xdr – скорочена схема даних XML (XML Data Reduced Schema);

*.mml –математична мова розмітки (MATHML Mathematical Markup Language);

*.cdf – формат визначення каналів (Channel Definition Format).

Головна властивість XML – його гнучкість. XML дозволяє змінювати визначення XML-документа, не перериваючи існуючі процеси обробки даних. Виходячи з цього, можливо здійснювати оновлення даних, не змінюючи сам додаток, який ці дані обробляє.

XML - це не просто текстовий формат для опису документів, але перш за все це механізм для опису структурованих і псевдоструктурованих

даних, який забезпечує доступ до багатого сімейства технологій обробки таких даних. Потужні абстракції, такі як інформаційна множина XML, відкривають двері до обробки нетекстових даних, таких як файлові системи, реєстр Windows®, реляційні бази даних і навіть об'єкти мов програмування, за допомогою XML-технологій. XML – це ще один крок, що забезпечує універсальний доступ до даних.

Оскільки XML-документ містить елементи, які описують самі себе, він зрозумілий людині на інтуїтивному рівні. Семантика даних забезпечує «інтелектуальність», яка подана в XML-елементах і значеннях атрибутів. Не дивлячись ні на що, XML – це програмний код, який читається і використовується обробниками XML.

XML утворює цілу сім'ю технологій, в яку входить ряд важливих технологій:

DTD	Визначення типу документа.
XDR	Формат XML Reduced (схема Microsoft).
XSD	Визначення схеми XML (схема консорціуму W3C).
Простори імен	Метод визначення імен елементів та атрибутів.
XPath	Мова шляхів XML.
XLink	Мова посилань XML.
XPointer	Мова покажчиків XML.
DOM	Програмний інтерфейс API для об'єктної моделі документів.
SAX	Простий програмний інтерфейс API для XML – Simple API for XML.
XSL	Розширена мова таблиць стилів.
XSL-FO	Об'єкти форматування XSL.
XSLT	Мова перетворень XSL.
X Include	Синтаксис XML Include.
XBase	Синтаксис XML Base URI.

Деякі з перелічених компонент до сього часу знаходяться в процесі розроблення, хоча використовуються досить широко, і можуть зазнавати

значних змін. Тому особливу увагу необхідно приділяти тому, як та чи інша технологія описана в W3C.

XML, заснований на Unicode, забезпечує доступ до величезної кількості технологій з маніпулювання, структурування, трансформування і запиту даних.

1.5 Платформа XML та її стандарти

На відміну від Web-1, де всі основні функції управління інформаційними ресурсами системи базуються на єдиній мові HTML, творці XML-платформи обрали інший шлях, суть якого полягає в наступному:

– в першу чергу виділяються «фундаментальні» стандарти, складові концептуальну і синтаксичну основу платформи;
– потім їх засобами визначається комплекс інших стандартів, кожен з яких виконує специфічні функції;
– у разі потреби цей комплекс відкритий для поповнення новими стандартами.

Описана «модульність» організації платформи забезпечує її відкритий характер, можливості введення нових стандартів, не зачіпаючи вже існуючі.

Функціональність цієї платформи визначається комплексом взаємопов'язаних стандартів, частина з яких вже прийнята W3C, інші знаходяться в стадії розробки. Таким чином, у міру появи необхідності в рамках даної платформи можна ввести будь-який стандарт, який розширює її функціональність. Звідси випливає висновок, що поряд зі створенням стандарту мови XML консорціум W3C, що формує технічну політику розвитку Web і розробляє стандартизовані специфікації для цього середовища, насправді одночасно формує нову відкриту для розширення функціональності технологічну платформу, головною ланкою якої є XML.

Про функціональні можливості XML-платформи можна судити за наведеним нижче перерахуванню основних її стандартів.

Основні стандарти XML-платформи:

– *фундаментальні*: Namespace, XML;

– *уявлення метаданих*: XML Schema, RDF, WSDL;

– *допоміжні*: XInclude, XPath;

– *форматування і трансформація XML-документів*: CSS, XSLT, XForm;

– *інтерфейс прикладного програмування*: DOM;

– *мова запитів XML-даних*: XQuery;

– *протоколи передачі даних*: HTTP, SOAP.

Розглянемо призначення перерахованих компонентів XML-платформи. Спочатку опишемо *роль мови XML*. У складі стандартів розглянутої платформи вона виконує дві важливі функції.

Перш за все, вона забезпечує змістовну (структурну) розмітку інформаційних ресурсів, які називають в розглянутому середовищі *XML-документами*, а також надає засоби (деякий під'язик XML) для опису загальної структури документів, що цікавить користувача типу. Такий опис називається ***XML Schema Definition (XSD)***.

По-друге. Але основне призначення мови XML – це служити одним з фундаментальних стандартів платформи XML. Інші стандарти платформи, що доповнюють її функції, пов'язані з управлінням даними Web, визначаються в термінах синтаксису XML. У зв'язку з цим їх називають іноді *додатками XML*.

Повертаючись до виконуваної мовою XML функції розмітки, слід ще раз підкреслити, що вона (на відміну від HTML) не є повнофункціональною мовою, яка повинна вирішувати всі завдання уявлення, підтримки і обробки інформаційних ресурсів Web. Якщо проводити аналогію з технологіями баз даних, то *XML можна кваліфікувати як мову визначення даних*.

Специфіка XML як мови визначення даних полягає в тому, що в неї поєднуються можливості:

– опису властивостей екземплярів елементів XML-документів, що становлять зміст даного конкретного документа;

– визначення властивостей типу XML-документів (XSD) в термінах типів елементів цих документів.

Перша група засобів (теги розмітки) використовується за принципом самоопису, визначаючи деякі властивості елементів конкретного документа за допомогою вбудованих в нього тегів розмітки.

Що стосується ***XSD***, то він описує типові властивості елементів документа і властивості типів документів в цілому.

XML Schema поряд з RDF займають важливе місце в складі платформи XML, позиціонуються як стандарти представлення метаданих. Основне призначення XML Schema полягає в описі синтаксичних властивостей XML-документів, а RDF – в поданні їх семантики.

Роль XSD аналогічна ролі схеми бази даних. При цьому XSD відчужується від описуваних документів і зберігається в каталогах Web. Конкретні XML-документи посилаються на це визначення, хоча вони можуть і включати його безпосередньо в явному вигляді.

Стандарт Namespaces in XML (Namespace) поряд з XML також використовується для визначення інших стандартів платформи. Стандарт Namespace визначає для заданого XML-документа або безлічі документів допустимі теги розмітки і їх атрибути, асоціюючи з ними за замовчуванням деяку семантику. Зарезервовані W3C простір імен використовуються в синтаксисі мови XML і інших стандартів платформи.

Засоби для форматної розмітки XML-документів визначають стандарти ***каскадних таблиць стилів CSS*** і розширюваної ***мови таблиць стилів XSL***.

Друга частина стандарту XSL, звана ***XSLT***, дозволяє описувати форматні перетворення (трансформації) XML-документів.

До числа стандартів форматування даних можна віднести також ***XForms*** вдосконалений і адаптований до середовища XML-аналог механізму форм в мові HTML, що забезпечує введення і передачу даних, наприклад запитів, від Web-клієнта до Web-серверу.

Стандарт ***DOM*** об'єктної моделі XML – і HTML-документів визначає функції інтерфейсу прикладного програмування для їх обробки.

До складу XML-платформи входить також ***стандарт мови запитів XQuery***, що активно розробляється з другої половини 2000 року і представляє собою декларативний засіб обробки XML-даних.

У складі XML-платформи розроблені документи, що описують *вимоги* до розробляється мови запитів, *моделі* даних, на яких вона базується, приклади, що ілюструють її функціональні можливості, а також специфікації синтаксису XQuery в XML.

Стандарт XPath визначає поняття фрагмента XML-документа, що використовується в мовах ***XSLT, XQuery*** і в розробці нової версії DOM.

1.6 Можливості XML-платформи та застосування стандартів XML

Принципово важливою властивістю мови XML, що забезпечує нові функціональні можливості середовища Web, є її розширюваність. Досягнення розширюваності мови XML обумовлено двома її особливостями.

Перш за все, XML – це мова метарівню, що входить в підмножину відомої мови SGML, а не конкретну мову, подібну HTML. Це дозволяє XML виконувати функції мови визначення даних. Використовуючи її синтаксис, можливо визначати різні типи елементів, екземпляри яких утворюють зміст конкретних XML-документів, і вводити тим самим адекватний потребам набір тегів розмітки документів.

Друга особливість мови XML пов'язана з використанням просторів імен – іменованих множин символів, використовуваних як імена типів елементів і атрибутів елементів XML-документів. Введення просторів імен в XML-документи дозволяє неявним чином асоціювати задану семантику з введеними іменами елементів документів, їх атрибутів і допустимими для них значеннями.

Використовуючи перераховані особливості мови XML, окремі користувачі або спільноти користувачів можуть створювати потрібні їм мови розмітки для різних категорій документів з необхідною семантикою тегів розмітки. При цьому в одному XML-документі допускається посилання на різні простори імен для комбінації в ньому різних породжених XML-мов.

Розглянуті принципи забезпечують також розширюваність функціональних можливостей всієї XML-платформи. Ця розширюваність забезпечується тим, що основа кожного доповнює XML-стандарт, що підтримує нові функціональні можливості, визначається засобами тої ж мови XML. При цьому вводиться простір імен із зарезервованим ім'ям, що включає імена нових типів елементів даного стандарту і їх атрибутів. Семантика елементів цих типів, їх атрибутів і значень, які вони можуть приймати, визначаються в специфікаціях стандарту.

Та обставина, що для створення стандартів платформи XML використовується один і той же засіб, призводить до їх синтаксичної однорідності. Це дозволяє розглядати *всі стандарти XML-платформи як додатки XML*. Дана обставина має істотне значення, оскільки інформаційні ресурси, підтримувані різними стандартами (наприклад, XML-, XSD-, XSLT-, RDF-файли), залишаються XML-документами і можуть оброблятися в XML-середовищі як «чисті» XML-документи.

Хоча мова XML і платформа стандартів W3C, що на неї базується створювалися як засіб подання інформаційних ресурсів Web, але вони вже знаходять найширше застосування практично у всіх областях інформаційних технологій.

Цьому сприяють, перш за все, розвинені можливості платформи для подання інформаційних ресурсів широкого спектра, а також її адаптованість (шляхом створення нового стандарту з необхідною функціональністю) до умов застосування. Певне значення має також можливість метаопису інформаційних ресурсів з потрібним ступенем формалізації і відкритий характер стандартів, що дозволяють інтегрувати інформаційні ресурси, представлені в XML-форматі, в будь-яку середу.

Нарешті, важливу роль відіграють можливості XML, що дозволяють використовувати її в якості засобу, що забезпечує обмін XML-повідомленнями через Web. Ця можливість підтримується в глобальному комунікаційному середовищі Web і дозволяє забезпечувати взаємодію систем різної природи.

Наведемо кілька конкретних напрямів використання XML-платформи. В даний час створено і продовжує створюватися велика

кількість версій мови XML для розмітки документів в різних предметних областях і створення XSD, які використовуються різними професійними співтовариствами. Відомі, зокрема, версії XSD для застосування в хімії, географії, астрономії, історії, бібліографії, видавничої справі та ін.

Дуже важливим напрямком застосування XML-платформи є технології баз даних, в яких XML використовується в якості мови визначення даних. Поєднання можливостей XML-платформи і технологій баз даних дозволяє створювати розподілені системи на основі існуючих інформаційних баз.

Як приклад використання XML-платформи в даному напрямку можна назвати плани застосування XML для кодування повідомлень, якими обмінюються клієнт і сервер в стандарті ISO / IEC RDA / SQL (Remote Database Access for SQL) віддаленого доступу до систем SQL баз даних.

Як приклад конкретного застосування XML-платформи в області обміну даними через Web можна розглядати стандарт потоків робіт, що розробляється консорціумом Workflow Management Coalition (WfMC), що дозволяє здійснювати обмін повідомленнями на мові XML між програмними засобами потоків робіт для підтримки їх інтероперабельності.

XML застосовується також в системах документообігу, аналогічних тим, які засновані на стандарті SGML і вже багато років використовуються на практиці. Використання мови XML в цій сфері дозволяє інтегрувати системи документообігу в середу Web.

Мова XML застосовується також у стандартах інших інформаційних технологій, де вона використовується як мова-посередник для обміну інформацією між різного роду системами за допомогою Web.

Як приклади можна перелічити наступні стандарти:

– XMI (XML Metadata Interchange) обмінного формату метаданих для CASE, створений консорціумом OMG;
– OIM (Open Information Model) , розроблений консорціумом Meta Data Coalition;

– CWMI (Common Warehouse Metadata Interchange) , який визначає формат подання метаданих та обміну метаданими для сховищ даних, створений OMG.

Ще один важливий напрямок застосування стандартів платформи XML, актуальний для створення корпоративних інформаційних систем – інтеграція неоднорідних інформаційних ресурсів.

1.7 Застосування XML-платформи в технологіях баз даних

Доцільність спільного застосування цих технологій обумовлена тим, що вони представляють альтернативні і взаємодоповнюючі технології представлення даних в різних областях: технології баз даних, що використовують бінарне представлення даних, застосовуються в сфері інформаційно-розрахункових завдань, а XML-технології – в комунікаційній сфері, так як текстове представлення даних є інваріантним щодо інформаційних технологій, які застосовуються у різних системах.

Тісний зв'язок між Web-технологіями і технологіями баз даних склалася ще на ранніх етапах розвитку Інтернет. Він зводився до забезпечення віддаленого доступу до систем баз даних через середовище Web. В даний час створено і функціонує величезна кількість додатків такого роду в самих різних областях діяльності. Але при цьому не йдеться про справжню інтеграції інформаційних ресурсів Web і баз даних. Доступ до даних здійснюється за допомогою так званих розширень Web-серверів, що створюються програмістами інформаційних систем, а Web-сервер просто упаковує ці дані в HTML-файли і передає по каналах зв'язку. Система бази даних при цьому не впроваджена в Web-середовище і виступає по відношенню до неї як «чорний ящик».

Прагнення до забезпечення в Web повноцінних можливостей управління даними, підтримуваними в цьому середовищі, призвело до необхідності використання підходів і принципів, аналогічних тим, які протягом десятиліть пройшли випробування часом в технологіях баз даних. Дійсно, в стандартах Web-2 можливо угледіти цілий ряд аналогій з ідеями, втіленими в технологіях баз даних. У лексиконі специфікацій

стандартів платформи XML з'явилися такі ключові терміни технологій баз даних, як *модель даних, схема, обмеження цілісності*. В частині забезпечення обмеження цілісності в специфікації XSD з'явилися конструкції для визначення елементів з унікальними значеннями, *первинних і вторинних ключів.* Як і в технологіях баз даних в XML, з розробкою специфікації XSD втілилася концепція багаторівневого подання даних, за допомогою якої підтримуються «логічне» і «фізичне» уявлення даних XML-документів.

Згодом взаємодія цих напрямків в інформаційних системах стала проявлятися і на рівні розвитку практичних технологій у вигляді розробки систем баз даних XML. Деякі компанії, випустили для цих цілей комерційні програмні продукти, а інші адаптували свої продукти для зберігання XML-даних. Такі системи варто було б називати *системами баз даних XML-документів*. Збережені в таких базах даних документи є незалежними один від одного, і ніяких зв'язків між ними не підтримується.

Як схеми бази даних при цьому використовується DTD документи, зберігаються типи і опис XML-документів засобами стандарту XML Schema (для пізніх версій).

Деякі програмісти бачать аналогію між концептуальною схемою баз даних і специфікацією RDF. Для доступу до XML-документу розробляються мови запитів, як і в системах баз даних. Одна з мов цього роду – XQL – використовується в продукті Tamino компанії Software AG. У наявних проектах таких мов інформаційні ресурси розглядаються як безлічі незалежних XML-документів.

Спільними зусиллями фахівців в області XML-технологій і технологій баз даних ведуться роботи зі створення стандарту мови запитів XQuery для XML-платформи. Цей проект має таке ж важливе значення, як і розробка мови SQL в технологіях баз даних. Її значення не тільки в забезпеченні рішення безпосередньо в створенні розвиненої мови запитів, але і матиме досить істотний ефект у вигляді засобу декларативної обробки XML-даних. Крім цього аналіз функціональних можливостей XQuery, а також опублікованої W3C версії специфікацій мови запитів, що створюється дозволяє виявити, що дана мова має функціональність, яка

виходить за вузькі рамки потреб роботи тільки з XML-ресурсами. Використання такої мови може забезпечити можливості інтеграції неоднорідних інформаційних ресурсів, таких як XML-документи, дані ієрархічної і реляційної структур.

Проблема інтеграції неоднорідних інформаційних ресурсів вирішується і в рамках технологій реляційних баз даних, де також робляться спроби створення стандартних засобів інтеграції SQL і XML-даних.

В рамках XML-платформи розвиваються також процедурні засоби обробки XML-даних. В якості такого засобу можна розглядати стандарт Document Object Model (DOM) об'єктної моделі для XML-документів на основі якого можна створювати інтерфейси прикладного програмування для систем баз даних XML.

У частині інтеграції баз даних з XML-даними розвивається ще один напрямок в технологіях баз даних. В рамках цього напряму виконано ряд досліджень, пов'язаних з відображенням XML-даних в реляційні, об'єктно-реляційній і об'єктній базі даних. Ці розробки мають практичну спрямованість, і їх результати цілком можуть бути затребувані практикою.

Таким чином, в перспективі можна очікувати, що під впливом подальшого розвитку XML-платформи буде усвідомлена необхідність в єдиній базовій повнофункціональної моделі даних платформи, основу якої і створюються в рамках проекту XQuery.

2 ОСНОВИ МОВИ XML

Мова XML була створена для зберігання, транспортування та обміну даними, з її допомогою можна реалізувати обмін даними між різними системами. XML є мовою розмітки, це ріднить її з мовою HTML, основна ж відмінність в функціональному призначенні цих мов:

XML – формат спроектований для передачі і зберігання даних.

HTML – створений для відображення даних.

XML – це *програмно і апаратно-незалежний інструмент* для перенесення інформації. XML – це універсальна мова розмітки, що не залежить від платформи, яку можливо використовувати для подання ієрархічних даних і уніфікації інформації, що передається. Крім того, XML – це текстовий формат, призначений для обміну інформацією між програмами, а також для створення на його основі більш спеціалізованих мов розмітки, іноді званих словниками.

2.1 Документ, як одиниця інформації XML

Розширювана мова розмітки XML, створена як мова розмітки характеризуется наступними особливостями. XML – це засіб зберігання і передачі даних, а також створення інших мов розмітки. Будучи мовою розмітки XML:

- з одного боку, призначений для зберігання і передачі даних і є, перш за все, транспортним засобом для передачі інформації будь-якого роду;
- з іншого боку, являє собою набір правил для створення інших мов розмітки.

Будучи набором правил для створення інших мов розмітки, XML дозволяє зберігати і впорядковувати інформацію майже будь-якого роду в форматі, пристосованому до потреб користувача.

У XML користувач винаходить свої власні теги. Розглянемо приклад XML-документа.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<Заметка>
<Кому> Анне </Кому>
<От> Татьяны </От>
<Заголовок> Напоминание </Заголовок>
<Сообщение> Не забудь позвонить мне завтра! </Сообщение>
</Заметка>
```

У цьому документі міститься:

- повідомлення із заголовком всередині тегів *<Заголовок> </Заголовок>* і текстом всередині тегів *<Повідомлення> </Повідомлення>;*
- ім'я відправника повідомлення (*Татьяна*) всередині тегів *<Від> </Від>*;
- ім'я адресата повідомлення (*Анна*) всередині тегів *<Кому> </Кому>*.

З цих даних можна припустити, що в цьому XML-документі міститься замітка від Тетяни доАнни. І все цілком зрозуміло. Іншими словами, XML-документ містить дані з самоописом.

Щоб закодувати інформацію описаним чином, теги в XML-документі (наприклад, <Кому> і <Від>) вводяться розробником документа, які при цьому не обмежуються ніякими стандартами . Ці теги були «винайдені» автором цього XML-документа.

Якщо в HTML-документі можна використовувати тільки ті теги, які визначаються в стандартах HTML *(<p>, * і т. д.), то XML дозволяє автору визначати свої теги і свою структуру документа, т. е. в мові XML немає зумовлених тегів.

Ще одна важлива особливість XML-документа полягає в тому, що цей документ нічого не робить – *пасивність XML-документа*. Він був створений для структурування, зберігання і передачі інформації. Це просто інформація, загорнута в теги. Тому потрібно написати додатки, які будуть відсилати, отримувати і відображати ці дані.

XML відокремлює дані від HTML. В даний час XML також важливий для мережі, як колись був важливий HTML для народження сучасного Інтернет. XML – це загальний інструмент передачі даних між

усіма видами додатків. Якщо вам в HTML-документі необхідно відображати динамічні дані, то вести цей документ буде занадто витратно, т. к. щоразу, коли ці дані змінювались, доводиться редагувати HTML-документ. У стандарті мови XML-дані можна зберігати в окремих файлах XML, а для відображення можна скористатися мовами HTML / CSS або HTML / XSL. При цьому нові дані, що надходять в кожен раз не зажадають яких-небудь змін в коді HTML-документа.

XML спрощує розподіл даних. У глобальній мережі комп'ютерні системи та бази даних використовують дані в несумісних бінарних форматах. Засобами XML дані створюються і зберігаються в простому текстовому форматі, що забезпечує програмну і апаратну незалежність і дозволяє легко створювати дані, які можуть використовуватися різними додатками. XML спрощує передачу даних. Однією з найбільш значущих проблем розробників була і залишається досі проблема обміну даними між несумісними гетерогенними системами. Передача даних у вигляді XML значно знижує складність цієї проблеми, так як дані в цьому форматі можуть бути прочитані в різних несумісних системах.

XML спрощує модифікацію платформи. Перехід на нові системи (апаратні або програмні платформи) завжди займає багато часу. Безліч даних необхідно конвертувати в нові формати. При цьому часто несумісні дані губляться. Зберігання даних в текстовому XML-форматі значно облегшує розширення або модернізацію систем, а також перехід на нові додатки або браузери без небезпеки втратити даних.

2.2 Структура XML-документу

XML-документ повинен мати строго певну структуру, що задається *правилами синтаксису мови*.

Узгодженим або коректним (well-formed) документом називають документ, який відповідає специфікації XML. Якщо документ не відповідає специфікації мови, то він не може вважатися XML-документом.

Крім узгодженості для ряду завдань від XML-документа потрібно також враховувати додаткові обмеження предметної області, які описуються за допомогою моделі документа.

Дійсним або валідним (valid) XML-документом називають узгоджений документ, який відповідає всім обмеженням, визначеним в моделі XML-документа.

XML-документ складається з частин, які називаються ***елементами***. Елементи складають основу XML-документів. Вони утворюють структури, які можна обробляти програмно або з допомогою таблиць стилів. Елементи розмічають іменовані розділи інформації. Елементи будуються за допомогою тегів розмітки, позначають ім'я, початок і кінець елемента. Елементи можуть бути вкладені один удругий, на верхньому рівні знаходиться елемент, званий ***елементом документа або кореневим елементом*** в якому містяться інші елементи.

Коректно сформований XML-документ може містити коментарі, інструкції по обробці, а також порожні рядки. У наведеному прикладі XML-документа, після основного елемента «Документ», містяться включення, які користувач може додавати в кожну з частин.

Документ XML може розташовуватися в одному або декількох файлах, причому деякі з них можуть знаходитися на різних машинах. У XML використовується спеціальна розмітка для інтеграції вмісту різних файлів в один об'єкт, який можна охарактеризувати як *логічну структуру*. Завдяки тому, що документ не обмежений одним файлом, XML дозволяє створювати документ з частин, які можуть розташовуватися де завгодно.

Документ XML зазвичай містить такі розділи:

- XML-декларація;
- Пролог;
- Елементи;
- Атрибути;
- Коментарі.

На рис.2.1 наведена приблизна структура XML-документа, де зазначені деякі розділи.

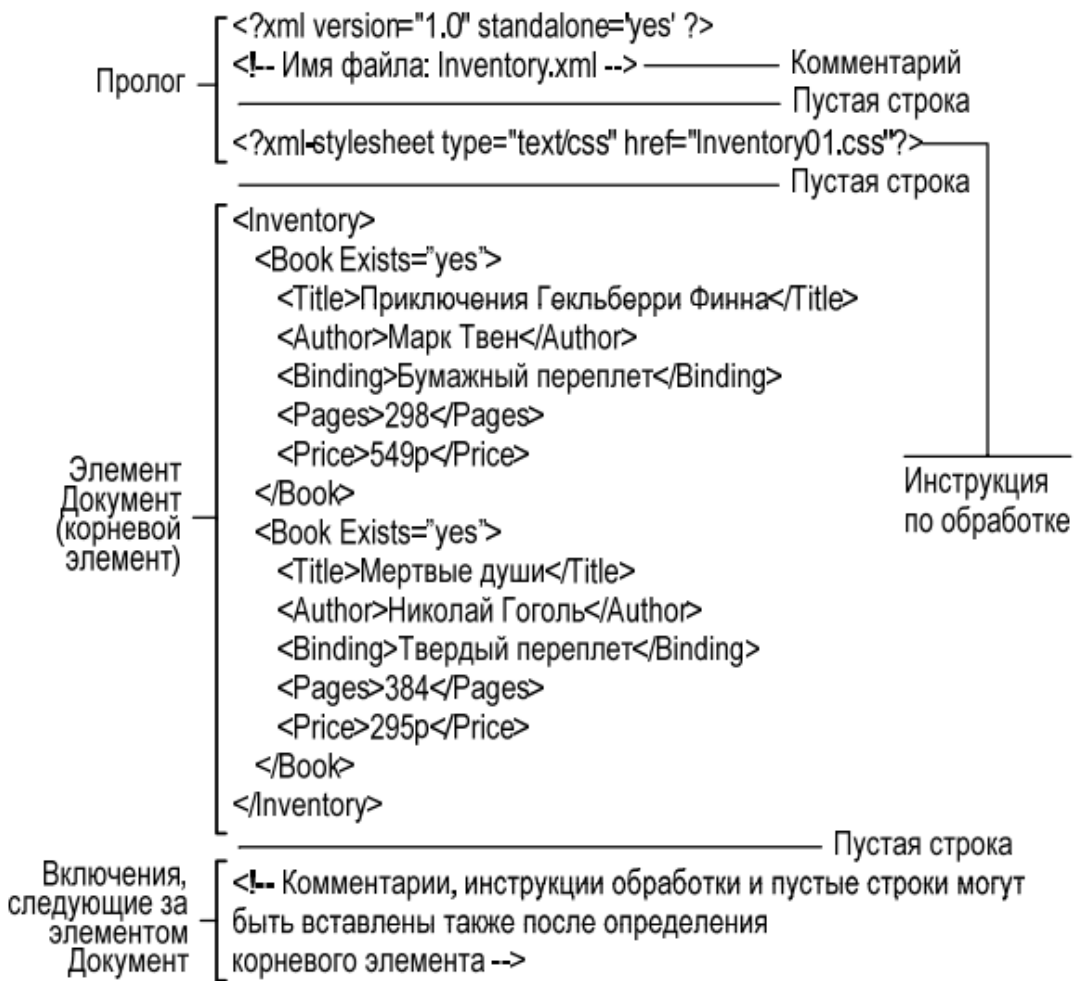

Рисунок 2.1 – Структура XML-файла Inventory.xml

2.3 Декларація XML-документа

XML-декларація зазвичай знаходиться в першому рядку XML-документа. XML-декларація не є обов'язковою. Однак, якщо вона існує, вона повинна розташовуватися в першому рядку документа, і до неї не повинно бути більше нічого, в тому числі прогалин. XML-декларація в схемі документа складається з наступних елементів:

1) Номер версії: `<? Xml version = "1.0"?>.`

Це обов'язковий аргумент. Поточна версія – 1.0. Номер версії може бути укладений як в одинарні, так і в подвійні лапки. Рядки в лапках в XML-розмітки є константи, включаються безпосередньо в текст документа, тобто літерали.

2) Декларація кодування:

```
<? Xml version ="1.0" encoding ="UTF-8"?>
```

Це необов'язковий параметр. Якщо він використовується, то декларація кодування повинна розташовуватися відразу після інформації про версії в XML-декларації. Декларація кодування повинна містити значення, що представляє собою існуючу систему кодування символів.

XML-документи можуть містити символи в різних міжнародних кодуваннях. Символи кодування визначає унікальний бінарний код для різних символів, які використовуються в документі. Юнікод – це промисловий стандарт для символьного кодіровки текстового документа і має два різновиди: UTF-8 і UTF-16: UTF-8 використовує один байт (8 біт) для подання загальноприйнятих символів і два (або три) байта для всіх інших символів. UTF-16 використовує два байта (16 біт) для більшості символів і три байта для всього іншого. Щоб не виникало помилок, необхідно вказувати, яка кодировка використовується в XML-документі, або зберігати файл в універсальній кодировкі UTF-8. UTF-8 є кодуванням за замовчуванням для XML-документів без інформації про кодування. Крім цього, більшість систем додатків XML працює з такими кодуваннями, як ISO-8859-1, Windows -1252 і ASCII.

3) Декларація автономності, наприклад:

```
<? Xml version = "1.0" encoding = "UTF-8" standalone = "yes"?>.
```

Декларація автономності (standalone = 'yes'), як і декларація кодування, необов'язкові. Якщо декларація автономності використовується, то вона повинна стояти на останньому місці в XML-декларації. На рис. 2.1 є дві інструкції по обробці в пролозі, а також інструкція по обробці в розділі. Декларація автономності використовуватися в деяких XML-документах з метою спростити обробку документа.

2.4 Пролог XML-документа

Прологом називаються дані, розташовані після відкриваючого тега документа або після кореневого елемента. Він включає відомості, що

відносяться до документа в цілому – кодування символів, структура документа, таблиці стилів.

```
<? Xml version = "1.0" encoding = "UTF-8"?>
<? Xml-stylesheet type = "text / xsl" href = "book.xsl"?>
<! DOCTYPE book SYSTEM "schema.dtd">
<! - Some comments ->
```

У мові XML є можливість включення в документ інструкцій, які несуть певну інформацію для додатків, які оброблятимуть той чи інший документ. Інструкції по обробці в XML створюються в такий спосіб.

```
<? Додаток Вміст?>
```

У першій частині процесінгової інструкції визначається додаток або система, яка призначена, друга частина цієї інструкції або її вміст. При цьому інструкції по обробці дійсні тільки для тих додатків, яким вони адресовані. Прикладом процессинговой інструкції може бути наступна інструкція:

```
<? Serv cache-document?> Винятком Подвійного тире -);
```

XML-процесор проігнорує його.

2.5 Елементи XML-документа

Основний зміст документ визначено в другій обов'язковій частини, що включає в себе елемент «Документ», або ***кореневий елемент***, який в свою чергу містить додаткові елементи. Елементи в XML документі відповідають за організацію інформації і є основними структурними одиницями мови XML.

Елементи оформлюються наступним чином:

```
<ElementName> Вміст елемента </ ElementName>
```

Теги встановлюють межі навколо вмісту елементу, якщо такі є.

У кожного елемента має бути ***ім'я***. Імена XML-елементів повинні підкорятися наступним ***правилам***:

- *назви можуть містити літери, цифри та інші символи;*
- *назви не можуть починатися з цифри або розділового знака;*

- *назви не можуть починатися з букв xml;*
- *у назві не повинно бути пробілів.*
- *не можна допускати прогалин у лапок (<);*
- *імена елементів є чутливі до регістру;*
- *всі елементи повинні мати тег, що закриває.*

У XML ***елементи*** можуть бути ***двох типів – порожні і непусті***.

Порожні елементи не містять в собі ніяких даних, таких як текст або інші конструкції, і можуть скорочено записуватися наступним чином:

```
<ElementName />
```

У XML документі *обов'язково повинен бути* присутнім ***єдиний кореневої елемент***, всі інші елементи є дочірніми по відношенню до єдиного кореневого елементу. При цьому повинен суворо дотримуватися *порядок вкладеності елементів*:

```
<Person>
<GivenName> Peter </ givenName>
<FamilyName> Kress </ familyName>
</ Person>
```

В даному випадку елемент `<person>` містить два інших елемента, `<GivenName>` і `<familyName>`. Елемент `<givenName>` містить текст Peter, а елемент `<familyName>` – текст `Kress.`

Елементи організовані в ієрархічну деревоподібну структуру, в якій одні елементи вкладені в інші. У XML-документі елементи визначають його логічну структуру і несуть в собі інформацію, що міститься в документі (в прикладі на рис. 2.1) це інформація про книги: назва, автор, ціна.

Типовий елемент складається з початкового тега, вмісту елемента і кінцевого тега. В прикладі (рис. 2.1) елемент «Документ» – Inventory, його початковий тег – `<Inventory>`, а кінцевий – `</Inventory>`, а вмістом елемента вважається текст, розташований між ними. Як вміст елемента можна використовувати такі типи повідомлень: вкладені елементи – на рис. 2.1 елемент Book вкладений в елемент Inventory, а елементи Book, в

свою чергу, мають у своєму вмісті вкладені елементи, як показано в наступному лістингу:

```
<Book>
<Title> Приключения Гекльберри Финна </Title>
<Author>Марк Твен</Author>
<Binding>Бумажный переплет</Binding>
<Pages>298</Pages>
<Price>355p</Price>
</Book>
```

Символьні дані – це текст, що виражає інформаційний зміст елемента, наприклад, назву певної книги в елементі `title`:

```
<Title> Пригоди Гекльберрі Фінна </ Title>;
```

Розділи CDATA – це текстовий блок, в якому можливо вільно розміщувати будь-які символи. Розділ CDATA використовується для включення в текст елемента символів, використовуваних для розмітки тексту, т. е. неприпустимих для використання в елементах всередині тегів. Приклад розділу CDATA всередині елемента, який використовується для включення в текст елемента таких символів, показаний в наступному прикладі:

```
<Text> Як ввести символи <! [CDATA [<,>, ', "і &]]> </ Text>
```

Можливо поміщати в документ порожній елемент, т. е. елемент, який не має вмісту. ***Порожній елемент*** створюється шляхом розміщення кінцевого тега відразу ж після початкового тега: `<Hr> </ hr>`, або можливо використовувати спеціальний тег порожнього елемента: ***<hr />.*** Обидві ці нотації є еквівалентними. Оскільки порожній елемент не має вмісту, то яке його призначення. Тут є два варіанти:

1) можливо використовувати порожній елемент, щоб вказати XML-додаткам виконати дію або відобразити об'єкт. Аналогом в HTML є порожній елемент *BR*, який є зазначенням браузеру вставити розрив рядка, а також порожній елемент *HR,* який вказує на вставку горизонтальної розділової лінії.

2) порожній елемент може нести інформацію за допомогою атрибутів. Аналогом в HTML є порожній елемент `IMG (зображення)`, що містить атрибути, які повідомляють процесору, де шукати графічний файл і як його відобразити.

2.6 Атрібути XML-документа

У XML елементи можуть містити атрибути з присвоєними їм значеннями, які поміщаються в одинарні або подвійні лапки. Атрибути дозволяють додавати відомості про елемент за допомогою пар «*Ім'я-значення»*. Атрибути часто використовуються для визначення тих властивостей елементів, які не зважають на вмістом елемента, хоча в деяких випадках (наприклад, HTML-елемент img) вміст елемента визначається значеннями атрибута. Атрибути можуть відображатися в відкривають або порожніх тегах, але не в тегах, що закриваються. Атрибут для елемента задається наступним чином:

```
<MyElement attribute = "value"> </ myElement>
```

Синтаксичні правила створення атрибута:

- *Декларуються в тегу, що відкривається;*
- *Кількість атрибутів не обмежена;*
- *Кілька атрибутів розділяються пробілами;*
- *Статус складається з імені та значення;*
- *Кожне ім'я має бути унікальним в рамках одного елемента;*
- *Не можна використовувати прогалини в іменах атрибутів;*
- *Значення атрибута повинно бути в лапках.*

Значення атрибутів можливо вставляти як в одинарні, так і в подвійні лапки. Можливо також використання одних лапок усередині інших, наприклад:

```
<MyElement attribute = "value"> </ myElement>
<MyElement attribute = 'value'> </ myElement>
<MyElement attribute = ' "value"'> </ myElement>
```

2.7 Коментарі XML-документа

Додавання коментарів в XML-документ не обов'язково, але дозволяє зробити його більш зрозумілим. коментар починається з символів **<! -** і закінчується символами **->.** Між цими двома групами символів ви можете помістити будь-який текст (за винятком подвійного тире); XML-процесор проігнорує його.

```
<! - Текст коментаря ->
```

Коментарі можуть перебувати в пролозі документа, в тому числі в визначенні типу документа (DTD), після документа і в текстовому вмісті. Коментарі не можуть перебувати всередині значень атрибутів. Вони також не можуть перебувати всередині тегів. Синтаксичний аналізатор вважає кінцем коментаря символ **>**; після цього символу він відновлює обробку XML-документа в звичайному режимі. Тому рядок **>** не може перебувати всередині коментаря. За винятком цього, в коментарях можуть використовуватися будь-які допустимі символи XML. Тому коментарі дуже корисні, якщо потрібно прибрати коментар з області обробки синтаксичного аналізатора, не видаляючи саме вміст з документа.

Для тимчасового видалення розмітки можна використовувати такі коментарі:

```
<! --- <test pattern = "SECAM" /> <test pattern = "NTSC" /> ->
<! - Текст коментаря ->
```

При створенні коментарів необхідно враховувати наступне: в тексті коментаря не може бути двох символів **«-»** поспіль; комментарій не може закінчуватися символом **«-».**

2.8 Базові правила побудови XML документа

Форматований XML-документ відповідає мінімальному набору правил, що забезпечує можливість обробки документа браузером або іншою програмою. Ці правила специфікації XML жорстко трактують все, що стосується структури. Вони прості і логічні, їх легко запам'ятати і легко

використовувати. Документ повинен мати таку розмітку, щоб не існувало двох способів інтерпретації імен, порядку та ієрархії елементів. Це значно зменшує число помилок і складність коду.

Програми не повинні ні про що «здогадуватися» або намагатися виправляти синтаксичні помилки, як це часто роблять браузери HTML, так як різні процесори XML повинні давати однакові результати.

При створенні XML-документа необхідно дотримуватися базових правил його побудови.

1) У XML документі допускається використання елементів з однаковим ім'ям незалежно від їх взаємного розташування. Смислове значення кожного з елементів визначається на етапі обробки документа.

2) Елементом верхнього рівня (кореневим елементом) може бути тільки один елемент. Нижче наведено некоректний документ, тому що містить два елемента верхнього рівня `<book>`.

Некоректний документ:

```
<? Xml version = "1.0"?>
<book> </book>
<book> </book>
```

3) Не допускається частковий перетин вмісту елементів. Приклад некоректного документу, в якому виконано частковий перетин типів елементів `book i magazine`**:**

```
<?xml version=" 1.0 "?>
<library>
<book>
<magazine>
</book>
</magazine>
</library>
```

4) На відміну від HTML в XML для кожного елемента обов'язковим є наявність початкового і кінцевого тегів. Винятком є скорочена версія порожнього тега.

5) Ім'я типу елемента в початковому тегу має в точності відповідати імені типу елемента в кінцевому тезі. Також слід враховувати, що імена тегів чутливі до регістру.

Наведено некоректний документ, в якому початковий тег `Book` і кінцевий тег `book` є різні імена (різні регістри).

```
<? Xml version = "1.0"?>
<Book> </book>
```

6) Імена атрибутів повинні бути унікальні в межах одного елемента. Не допускається використання в одному елементі декількох атрибутів з однаковими іменами.

7) Імена елементів, вміст елементів, імена атрибутів і вміст атрибутів повинні відповідати описаним вище вимогам.

Якщо XML-документ складений у відповідності з наведеними синтаксичними правилами, то кажуть, що це «синтаксично вірний »XML-документ.

3 ТЕХНОЛОГІЇ ОБРОБКИ ДАНИХ В ФОРМАТІ XML

3.1 XML-схема

Мова визначення XML-схем, або XSD (XML Schema Definition), є розширенням XML і на сьогоднішній день представляється основним інструментом опису структури XML-документів.

З використанням XML-схеми формалізується набір правил, з дотриманням яких складається XML-документ конкретного призначення. Базовий синтаксис XML визначає те, яким чином в тексті повинні виділятися елементи і їх атрибути, а також правила опису структурних відносин між елементами.

XML Schema виконує наступні функції:

- *Описує назви елементів і атрибутів (словник).*
- *Описує взаємозв'язок між елементами і атрибутами, а також їх структуру (модель змісту).*
- *Описує типи даних.*

XML-схема – це універсальний спосіб опису граматики даних, який може застосовуватися не тільки для верифікації XML-документів, але і опису баз даних, структур даних, використовуваних в мовах програмування і т. д. На сьогоднішній день область застосування XML Schema вельми обширна і з часом вона може стати основним стандартом моделювання даних, за допомогою якої можна описувати практично все.

Дані, які представлені в форматі XML, зазвичай проходять процедуру ***валідації,*** тобто перевірки граматики документа на відповідність певними схемами. В таких схемах знаходиться опис структур даних XML-документа. Необхідність перевірки граматики XML-документів полягає в наступному:

XML-документ може бути призначений для іншої системи;

XML-документ може містити некоректні дані;

XML-документ може містити помилки в структурі.

При обробці XML-даних валідація – це фундамент для подальших дій з XML-документом, інформація що до валідності XML-документа

може бути відправлена на подальшу обробку в цільовій модуль. Існують три основні різновиди схем: DTD, XDR і XML-схеми (XSD). На сьогоднішній день актуальними є більш сучасний підхід – XML-схеми (XSD). На початковому етапі розвитку XML-технологій для цілей моделювання документів використовувалася специфікація *ВизначеньТипу Документа (Document Type Definitions, DTD)*. Але в даний час дана специфікація практично повністю витіснена більш досконалою ***мовою опису XML Schema (XSD)***, тому наведемо лише короткий опис DTD.

3.2 DTD-схема

DTD (Document Type Definition, визначення типу документа) – це мова опису структури XML-документа, який використовується для перевірки граматики XML-документа і його відповідності визначеним стандартам. Це дозволяє парсеру на етапі обробки визначити, чи відповідає документ необхідним вимогам, тобто чи є документ дійсним.

DTD описує:

- *Які елементи можуть бути присутніми в документі;*
- *Входження елементів (повторення і т.п.);*
- *Можливі атрибути елементів;*
- *Обов'язкові / необов'язкові атрибути;*
- *PCDATA і CDATA;*
- *Застосовувані в документі суті.*

DTD головним чином складається з виразів мови розмітки, що включає елементи:

```
<! ELEMENT ...> і <! ATTRIBUTE ...>.
```

В якості тестового документа для опису за допомогою специфікації DTD виберемо XML-документ, розглянутий в прикладі, наведеному на рис.2.1. Кореневим елементом документа є елемент Inventory, який може містити необмежену кількість елементів `Book`, тому для:

- – вказівки включення помістіть Book в круглі дужки;

– визначення кількості вкладених елементів поставте знак «Нуль і більш» (*) після Book.

Якщо ж потрібна обов'язкова наявність елемента Book, потрібно поставити знак «один або більше» (+). Для визначення одного елемента нічого не потрібно ставити після елемента Book. Перераховані позначення завдання обмежень на кількість елементів представляють частину системи позначень в регулярних виразах. Отже, перший рядок DTD виглядає наступним чином:

```
<! ELEMENT Inventory (Book *)>
```

Кожна книга містить тільки по одному елементу `Title, Author, Binding, Pages і Price,` причому саме в цьому порядку (для цілей даного керівництва кілька авторів вводяться в один елемент `Author`). Отже, наступним виразом мови розмітки є:

```
<! ELEMENT Book (Title, Author, Binding, Pages, Price)>.
```

Решта елементів є вузлами-листям, що містять символьні дані. Як зазвичай використовуються круглі дужки для позначення включення. Необхідно оголосити тип символьних даних. Символьні рядки є аналізованими даними, вказуються літералом `#PCDATA`:

```
<! ELEMENT Title (#PCDATA)>
```

Можливо надати кожній книзі унікальний код ідентифікації через спеціальний тип атрибута `ID`. Крім того, можливо дозволити необов'язковий атрибут `image,` що містить `URL` зображення обкладинки книги. Мітка `ATTLIST` приймає в якості аргументу елемент, за яким йде послідовність для кожного атрибута, пов'язаного з цим елементом. Кожна послідовність складається з імені атрибута, його типу і індикатора того, є він обов'язковим чи ні.

Специфікація DTD дозволяє вказувати 10 типів атрибутів:

PCDATA – будь-яка символьний рядок, прийнятна в XML;

NMTOKEN – близький до XML-імені;

NMTOKENS – маркери NMTOKEN, розділені пробілами;

Enumeration – список тільки допустимих значень для атрибута;

ENTITY – пов'язує ім'я з підстановкою, подібно макросу;

ENTITIES – розділений пробілами список імен ENTITY;

ID – XML-ім'я, унікальне в усьому документі;

IDREF – посилання на атрибут ID всередині документа;

IDREFS – розділений пробілами список маркерів IDREF;

NOTATION – пов'язує ім'я з інформацією клієнта.

Необхідний атрибут вказується шляхом додавання `#REQUIRED` після типу. Можливо вказати необов'язковий аргумент шляхом додавання `#IMPLIED`.

DTD опис містить один атрибут кожного типу для елементу `Book`. Єдиний вираз `ATTLIST` виглядає так:

```
<! ATTLIST book id ID #REQUIRED image CDATA #IMPLIED>
```

Наведемо повністю описаний DTD розглянутого XML-документа:

```
<?xml version="1.0" encoding="UTF-8"?>
      <!ELEMENT Inventory (Book*)>
      <!ELEMENT Book (Title, Author, Binding, Pages, Price)>
      <!ELEMENT Title(#PCDATA)>
      <!ELEMENT Author(#PCDATA)>
      <!ELEMENT Binding(#PCDATA)>
      <!ELEMENT Pages(#PCDATA)>
      <!ELEMENT Price (#PCDATA)>
<!ATTLIST Book id ID #REQUIRED image CDATA #IMPLIED>
```

У розглянутій специфікації DTD відсутня підтримка типів даних XML-документа!

Типи даних є важливим засобом моделювання даних, так як вони накладають необхідні обмеження на структурні елементи описуваних даних. Відсутність підтримки типів даних в специфікації DTD істотно обмежувало використання XML-даних в мовах програмування і СУБД. Крім обмежень на діапазон можливих значень система типів визначає набір операцій над даними, якими розробники можуть оперувати в своїх програмах. Наведемо приклад. Припустимо, що ми маємо вираз «70 + 50 =?». Якщо прийняти, що тип наведених операндів – числовий, то

відповіддю є «120», але якщо їх тип – рядок, то результатом може бути «7050», якщо під оператором «+» мається на увазі операція «конкатенації».

До розробки специфікації XSD мова XML представляла приклад мови, в якої була відсутня система типів. В результаті інформація, що знаходиться в документі XML, могла інтерпретуватися тільки як текст. Але коли XML-дані почали використовуватися в мовах програмування і записуватися в базах даних, то для їх взаємно зворотного перетворення з'явилася необхідність знати про «дійсний тип» даних, щоб здійснювати необхідні приведення в коді програми або в базах даних.

Мова XML Schema (так часто називають специфікацію XSD) підтримує систему типів даних для інформаційних об'єктів, що моделюються за допомогою мови XML. Тому з появою в арсеналі засобів XML-платформи ***специфікації XSD*** мова XML стала таким ж засобом моделювання даних, як і мови програмування високого рівня і бази даних. Це дозволяє описувати типи даних XML-документа так само, як це робиться в мовах програмування і мовою SQL.

Це концептуально відрізняється від принципів роботи DTD, так як XML Schema пропонує потужні засоби для взаємно зворотного перетворення даних документа XML і змінних традиційних мов програмування або систем типів баз даних. Цей фактор перетворює XML в потужний засіб для обміну даними між гетерогенними системами, в яких дані, як правило, визначаються в різних форматах і системах кодування.

3.3 Визначення XML schema

XML-документ і XML-схема співвідносяться один до одного так само, як об'єкти і класи в об'єктно-орієнтованих мовах або записи в базах даних і схеми баз даних. Тому за допомогою мови XML Schema можна виконати перевірку достовірності примірника документа, яку здійснюють XML-процесори шляхом зіставлення імен елементів, оголошених в схемі, з іменами в даному екземплярі документа. У разі збігу імен для перевірки достовірності елемента використовується інформація про його тип і структуру, оголошена в схемі.

Схеми XDR визначають елементи, подані в екземплярі XML, а також будь-які атрибути, пов'язані з цими елементами, крім того в схемах XDR визначаються взаємозв'язки елементів XML-документа.

Дуже важливо усвідомити той факт, що XDR-схеми є екземплярами XML, тобто вони складаються з елементів і атрибутів XML, а отже, вони перевіряються на відповідність інші схеми. Тому, створюючи XML-документ згідно розробленій XDR-схемі, програміст має ще справу і з третім документом, що містить середовище для XDR-файлу.

Як працюють XDR-схеми. Перш за все, відзначимо, що XDR-схеми – зовнішні. Зв'язок XML-файлу з його XDR-схемою прописується як атрибут кореневого елемента. Наприклад,

```
<?xml version= ‘1.0’ encoding= ‘windows-1251’ ?>
<note xmlns=‘x-shemas:example_4.xdr’>
У вівторок о 14.00 відбудуться урочисті збори
співробітників </note>.
```

У другому рядку кореневий елемент `note` містить атрибут `xmlns`, який вказує, в якому файлі міститься відповідна XDR-схема.

4 МОВА ВИЗНАЧЕННЯ СХЕМ XSD

4.1 Призначення XSD

Мова XML Schema Definition Language, яку також називають XML Schema Language, багато в чому схожа на мову XDR, з якою ви познайомилися раніше. Схеми XSD здатні розв'язувати такі задачі:

- перелічення елементів у документі XML і перевірка наявності в документі тільки оголошених елементів;
- оголошення і визначення атрибутів, що модифікують елементи документа;
- визначення батьківсько-дочірніх стосунків між елементами;
- визначення станів і моделей вмісту для елементів і атрибутів;
- завдання типів даних;
- установка значень за замовчанням;
- можливість розширення;
- підтримка використання просторів імен.

Усі схеми, незалежно від використовуваних для їх створення словників і синтаксичних правил, призначені для завдання певних обмежень на документи XML, а отже, для забезпечення відповідності останніх певним правилам. Мови опису схем, які базуються на синтаксисі XML, володіють певними перевагами перед DTD, оскільки допускають розширення використовуваних дескрипторів розмітки, а також перевірку документів і схем за допомогою стандартного синтаксичного аналізатора XML. У схемах XSD дескриптори, що використовуються в документах XML, розділяються на дві категорії :

- складні типи. Елементи складних типів можуть містити інші елементи, а також володіють певними атрибутами, тобто мати змішаний вміст;
- прості типи.

Наприклад, до ***простого типу*** можна віднести дескриптор

```
<text> У вівторок о 14.00 відбудуться урочисті збори
співробітників </text>.
```

Фрагмент, наведений нижче, містить дескриптор note, який може вважатися за ***складний***.

```
<note time= ’14:00:03’
            date= ’17-05-2019’>
            <text>
      У вівторок о 14.00 відбудуться урочисті збори співробітників </text>
</note>.
```

Прості і складні типи елементів – це унікальні характеристики мови XSD.

4.2 Простори імен

Будь-яка схема повинна використовувати стандартний простір імен для всіх екземплярів XSD. Цей простір імен ідентифікує набір елементів і атрибутів, які утворюють словник XSD.

Кореневим елементом у схемі XML є елемент Schema, який містить решту всіх елементів у документі схеми.

У рамках кореневого елемента схеми XSD створюється простір імен, використовуючи атрибут xmlns. Наприклад, наведений нижче дескриптор належить угоді про використання префікса "xsd" для зв'язку елементів з іменованою колекцією:

```
<xsd:schema xmlns:xsd=’http://www.w3c.org/1999/XMLSchema’.
```

За загальною угодою префікс **xsd** використовується для зв'язку екземпляра XML-документа зі схемою. Проте і його використання має бути узгоджене з іменованою колекцією, тобто префікс має бути описаний шляхом використання атрибута xmlns

```
xmlns:xsi="http://www.w3.org/2000/10/
XMLSchema-instance".
```

Для зв'язку екземпляра документа XML із схемою найчастіше використовуються два атрибути:

```
xsi:schemaLocation
xsi:noNamespaceSchemaLocation.
```

Ці атрибути дозволяють пов'язувати документ із стандартом XML Schema консорціуму W3C. *Такий зв'язок* не обов'язковий, але дуже корисний, оскільки дозволяє синтаксичним аналізаторам швидше знаходити потрібну схему.

У випадку, якщо використовується локальний файл, використовують атрибут `xsi:noNamespaceSchemaLocation.`

Наприклад,

```
xsi:noNamespaceSchemaLocation=
'ім'я _файлу.xds'
```

З іншого боку, простір імен може бути оголошений разом з ім'ям файлу, тоді ідентифікатор URI для простору імен і ідентифікатор URI для схеми розділяються пропусками, утворюючи значення одного атрибута, як показано нижче:

```
xsi:schemaLocation=http://example.org/ns/
books/ ім'я_файлу.xsd.
```

4.3 Прості елементи

Концепція іменованих типів. На відміну від DTD і XDR схем існує концепція *іменованих типів.* Наприклад, при створенні визначень існує можливість привласнювати цьому визначенню ім'я, щоб повторно використовувати його в екземплярі XSD. Наприклад, розглянемо фрагмент, що містить два прості елементи

```
<date> Травень,14,2019 </date>
<note>Відвідати лекцію, дуже важливий матеріал </note>.
```

Як видно, типом вмісту обох елементів є рядки. Для повного визначення елементів згідно з XSD схемою необхідно вказати назву елемента і його тип:

```
<xsd: element name='date' type='xsd:string' />
<xsd: element name='note' type='xsd:string' />.
```

Як видно, обидва елементи містять опис type=’xsd:string’, що дублюється. Щоб уникнути такого, звертаються до концепції іменованих типів або іменованих обмежень. Іменовані ***обмеження*** задають так:

```
<xsd:simpeType name=’ім’я_обмеження’ base=‘xsd: тип_даних’>.
```

Наприклад, якщо в схему ввести іменоване обмеження

```
<xsd: simpeType name=’nt’ base=‘xsd: string’>.
```

Опис елементів виглядатиме так:

```
<xsd: element name= ‘date’ type=’nt’ />
<xsd: element name= ‘note’ type=’nt’ />.
```

Таким чином, у нашому прикладі, елемент типу `string` посилається на елемент типу `simpeType.`

Розглянемо приклад, в якому наведемо XML-документ, що містить прості елементи і XSD-схему, для XML-документа.

1) «Порожній» XML-документ

```
<?xml version=’1.0’>
<note>Відвідати лекцію, дуже важливий матеріал</note>
```

2) XSD-схема (example_11.xsd) для простого XML-документа

```
<?xml version=’1.0’>
<xsd:schema
xmlns:xsd=’http://www.w3.org/2000/10/
XMLSchema’>
<xsd:element name=’note’ type=’xsd:string’/>
</xsd:schema>
```

3) Підключаємо схему до XML-документа (example_11.xml)

```
<?xml version=’1.0’>
<note
xmlns:xsi=’http://www.w3.org/2000/10/XMLSchema-instance’
xsi:noNamespaceSchemaLocation=’example_11.xsd’>
Відвідати лекцію, дуже важливий матеріал
</note>.
```

4.4 Прості типи даних в схемах XSD

У таблиці наведені типи даних, які відносяться і до типів даних, і до атрибутів.

Простий тип	**Опис**
string	Буквено-цифровий рядок
normalizedString	Рядок без пропусків
byte	1,126
unsignedByte	0,126
Integer	-126789, -1, 0, 1, 126789
Positivelnteger	1, 126789
Negativefnteger	-1267879, -1
nonNegativelnteger	0, 1, 126789
nonPositivelnteger	-1267879, -1, 0
Int	-1, 126789675
Unsignedlnt	0, 1267896754
Long	-1, 12678967543233
unsignedLong	0, 12678967543233
float	
double	
Boolean	True, false, 1, 0
Time	13:20:00.000, 13:20:00.000-5:00
dateTime	1999-05-31T1313:20:00.000-5:00
date	1999-05-31
gMonth	--11--
gYear	2007
gYearMonth	2007-11
gDay	---14
gMonthDay	--11-14

4.5 Елементи складних типів

Вміст елементів складного типу може бути різним. Він може включати текст, інші елементи, атрибути і т.д. Наприклад.

1) Маємо XML-документ, який містить елемент складного типу:

```
<note>
        <text>
Завтра о 12.00 лекція з Web-дизайну
        </text>
</note>.
```

2) Розробимо XSD-схему обмежень для даного документа. Опис елемента складного типу, що має елемент, відбувається в строго обумовленому порядку. А саме:

```
<xsd:element name='ім'я_батьківського_елемента'>
<xsd: complexType>
<xsd:element name='ім'я_дочірнього_елемента'
type='простий_тип'>
</xsd:complexType>
</xsd:element>.
```

Згідно з вимогами XSD-схема документа (example_12.xsd) виглядатиме так:

```
<?xml version='1.0'>
<xsd:schema
xmlns:xsd='http://www.w3.org/2000/10/XMLSchema'>
<xsd:element name='text' type='xsd:string'/>
<xsd:element name='note'>
<xsd:complexType>
<xsd: sequence>
```

У даному випадку необов'язково указувати на послідовність дочірніх елементів:

```
<xsd: element ref='text'>
</xsd: sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>.
```

XML-документ з підключеною схемою XSD:

```
<?xml version='1.0'>
```

```
<note
xmlns:xsi=’http://www.w3c.org/2000/10/
XMLSchema’>
xsi:noNamespaceSchemaLocatin=’example_12.xsd’>
<text>
Відвідати лекцію, дуже важливий матеріал
</text>
</note>.
```

4.6 Обмеження входжень у схемах XSD

XSD дозволяє визначити кількість входжень елемента з певною точністю: задати мінімальну і максимальну кількість входжень елемента за допомогою атрибутів `minOccur` і `maxOccur` елементу `xsd:element` відповідно. Використання цих атрибутів може здатися схожим на їх використання в XDR-схемах, але це тільки на перший погляд.

У випадку мови XSD на можливі значення цих атрибутів накладаються певні обмеження. Це пов'язано з тим, що консорціум W3C визначив *бажаний діапазон входжень*. Це одна з найбільш відмінних характеристик XSD по відношенню до XDR, DTD і іншим мовам опису схем. Якщо ніщо інше не вказано, то значення за замовчанням атрибутів minOccur і maxOccur дорівнюють "1". Атрибут maxOccur також може набувати значення "unbounded", що аналогічно значенню "*" в мовах DTD і XDR.

```
1:   <?xml  version=’1.0’?>
2:   <note
3:     xmlns:xsi=’http://www.w3.org/2000/10/
XMLSchema-instance’
4:     xsi:noNamespaceSchemaLocation=
’example_13.xsd’>
5:   <notes>
6:           <number/>
7:     <text>Раптово випав сніг</text>
8:            <text>А завтра лекція з web-дизайну</text>
9: </notes>
10:<notes>
11:    <number/>
12:    <text>Забрати книгу</text>
13:    <text>Знайти літературу</text>
14:    <text> Підготуватися до семінару </text>
15:   </notes>
16:   </note>.
```

На документ передбачається накласти ряд обмежень:

- У документі допускається не більше двох елементів `notes`.
- Входження `елементів notes` необов'язкові.
- Елемент `number` повинен передувати елементу `text`.
- Повинен існувати як мінімум один елемент `text`.

Для створення схеми XSD, що відображає всі ці обмеження, необхідно вказати:

- кількість елементів;
- порядок їх слідування;
- обов'язкові і необов'язкові елементи.

Для того, щоб задати обмеження на кількість елементів, використовуйте атрибути minOccur і maxOccur відповідних елементів xsd:element. У XSD можна додати обмеження, що стосуються обов'язкової наявності елементів, використовуючи ті ж самі атрибути minOccur і maxOccur. Наприклад, якщо задати атрибут minOccur таким, що дорівнює 0 для певного елемента або взагалі виключити цей атрибут, щоб використовувати його значення за замовченням, цей елемент вважатиметься за обов'язковий. (Обов'язковим є як мінімум одне входження елемента.) Ви вже знаєте, що значення більше 1 для атрибута maxOccur позначається як "unbound". Розглянемо можливий варіант схеми XSD, що відображає вищенаведені обмеження

```
1:    <?xml   version="1.0"?>
2: <xsd:schema  xmlns:xsd="http://www.w3.org/2000/10/
XMLSchema">
3: <xsd:element name='text'   type='xsd:string'/>
4:      <xsd:element  name='number'/>
5:      <xsd:element name='note'>
6:                  <xsd:complexType>
7:                        <xsd:sequence>
8:     <xsd:element name='notes' minOccurs='0'
max0ccurs='2'/>
9:                        </xsd:sequence>
10:          </xsd:complexType>
11:   </xsd:element>
12:   <xsd:complexType name='notesType'>
<!-- СТВОРЮЄМО ІМЕНОВАНЕ ПРАВИЛО, ТИПУ notesType В
ДОКУМЕНТІ НЕ ІСНУЄ, ПРОТЕ ВСІ ЕЛЕМЕНТИ ПОВИННІ ЗАДОВОЛЬНЯТИ
ОБМЕЖЕННЯ, ЩО НАКЛАДАЮТЬСЯ notesType-->
13:   <xsd:sequence>
```

```
      14:          <xsd:element   ref="number"/>
      15:          <xsd:element   ref="message"
maxOccurs="unbounded"/>
      16:    </xsd:sequence>
      17: </xsd:complexType>
      18:    </xsd:schema>.
```

Складна послідовність називається notesType тільки для зручності; ви можете називати її, як тільки захочете.

4.7 Опис атрибутів

Оголошення атрибутів у схемах XSD дуже схоже на оголошення елементів, за винятком того, що атрибути оголошуються за допомогою оголошень **attributes**, а не element. Як вже наголошувалося раніше, оголошення атрибутів містяться у визначеннях складних типів у схемах.

Наприклад, example_14.xml

```
1:     <?xml   version='1.0'?>
2:     <note
3:  xmlns:xsi='http://www.w3.org/2000/lO/
XMLSchema-instance'
4:  xsi:noNamespaceSchemaLocation=
'example_14.xsd'>
5:  <text number='101'>
6:     Потрібно здати ДЗ. ТЕРМІНОВО!!!!
7:  </text>
8:     </note>.
```

Цьому документу в схемі XSD повинен відповідати складний тип. Вказівка типів даних у схемах XSD складається з декількох етапів. Ви починаєте обмежувати типи елементів однією досить загальною категорією, після чого звужуєте цю категорію до певних типів. Такий підхід може здатися надлишковим, проте він корисний при роботі з великими схемами, що містять велику кількість типів даних. Перший, найбільш загальний тип, вказується за допомогою елемента **xsd: restriction** з атрибутом **base**, що оголошує широку категорію. Зазвичай атрибут base='xsd:string'. Дочірні елементи елементів xsd:restriction містять докладні визначення типів даних. Все це показано нижче. Схема XSD з перевіркою атрибутів example_14.xsd

```
<?xml version="1.0"?>
<xsd:schema
xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
<xsd:complexType  name="textType">
<xsd:simpleContent>
//не містить атрибут name
<xsd:restriction base="xsd:string">
//оголошуємо широку категорію (базу для типів //даних)
<xsd:attribute  name="number"
type="xsd:integer"
use="required"/>
</xsd:restriction>
</xsd:simpleContent>
</xsd: complexType>
<xsd: element   name="note">
<xsd: complexType>
//не містить атрибут name
<xsd: sequence>
<xsd:element name="text"
type="textType"/>
//містить посилання на оголошення складного типу //з
оголошенням атрибута
</xsd: sequence>
</xsd: complexType>
</xsd: element>
</xsd:schema>.
```

Підтримка типів даних мовами XSD і XDR значно відрізняє їх від DTD, який такою підтримкою не володіє зовсім. Консорціум W3C доклав всі можливі зусилля, щоб реалізувати в XSD підтримку найбільш широкого набору типів даних.

XSD розділяє всі типи даних на дві групи:

- базові (primitive);
- похідні (derived).

Базові типи даних – це такі типи даних, які не виражаються за допомогою інших типів даних.

Похідні типи даних – це такі типи даних, які виражаються за допомогою інших базових типів даних.

Розглянемо, наприклад, такий **основний** тип даних, як **float**, а також *похідний* тип даних *integer*. Тип даних float – це чітко визначена математична концепція, яку не можна виразити в термінах інших типів даних, тоді як integer – це різновид більш загального типу даних decimal.

Таким чином, тип даних integer похідний від типу даних decimalРозглянемо ще один приклад:

```
1:     <?xml version="1.0"?>
2:     <note
3:     xmlns:xsi= "http://www.w3.org/2000/10/
XMLSchema-instance"
4:xsi:noNamespaceSchemaLocation=
"message06.xsd">
5:     <text number='101' date='2009-11-14'
from='Проценко О.Б.'>
6:     Коли я дочекаюся від Вас ОДЗ?!
7:     </text>
8:     </note>.
```

Опишемо всі атрибути:

```
<xsd:attribute name-"number" type="xsd:integer"
use="required"/>
      <xsd:attribute name="date" type="xsd:date"
use="required"/>
      <xsd:attribute name="from" type="xsd:string"
use="required"/>.
```

Кожен елемент attribute в цьому прикладі оголошується за допомогою атрибутів name (відповідає імені елемента в документі XML), type (оголошення типу даних відповідно до описаних раніше обмежень) і use (визначення того, обов'язковий чи ні атрибут). Якщо ви оголошуєте атрибути як обов'язкові, вам необхідно вказати атрибут use="required" для кожного елемента xsd:attribute.

```
1: <?xml   version="1.0"?>
2: <xsd:schema   xmlns:xsd="http://www.w3.org/2000/10/
XMLSchema">
3: <xsd:complexType   name="textType">
4: <xsd:simpleContent>
5:     <xsd:restriction   base="xsd:string">
6:         <xsd:attributename="number"
type="xsd:integer"
use="required"/>
7:          <xsd:attribute     name="date"
type="xsd:date"
use="required"/>
8:          <xsd:attribute     name="from"
type="xsd:string"
use="required"/>
9:    </xsd: restriction>
10: </xsd: simpleContent>
```

```
11: </xsd: complexType>
12: <xsd: element   name="note">
13:          <xsd: complexType>
14:                <xsd: sequence>
15:                      <xsd:element name="text"
type="textType"/>
16:                </xsd: sequence>
17:          </xsd: complexType>
18: </xsd: element>
19: </xsd: schema>.
```

Отже, було розглянуто три пов'язаних підходи, що відрізняються. Вони забезпечують перевірку документів на відповідність синтаксичним правилами і стандартам, які визначаються спеціальними обмеженнями. Схеми DTD використовуються найдовше і підтримуються великою кількістю варіацій застосувань. Якщо необхідні строгі вказівки обмежень, то DTD найменш гнучкий з трьох мов опису схем, що були розглянуті, – саме те, що необхідне. Оже, в схемах DTD не використовується синтаксис документів XML, що є важливою перевагою над іншими мовами. Хоча використання синтаксису XML в схемі необов’язково приводить до поліпшення процесу перевірки, воно приводить до додаткової гнучкості, а також дозволяє перетворення в інші мови розмітки, чим не може похвалитися DTD. Схеми, що базуються на словниках XML, можуть містити складнішу і виразнішу розмітку, а також складні обмеження змісту.

Можливо, найбільш важливим поліпшенням, пов'язаним із схемами на базі XML, є можливість перевірки типів даних. Мови XSD і XDR підтримують достатньо широкий діапазон типів даних. XSD підтримує навіть більше типів даних, ніж XDR.

4.8 Використання XSL-таблиць стилів

Існують два основні кроки для відображення XML-документа при використанні XSL-таблиць стилів.

1) Створення файлу XSL-таблиці стилів. XSL є додатком XML, тобто XSL-таблиця є коректно сформованим XML-документ, який відповідає правилам XSL. Подібно до будь-якого XML-документа, XSL-таблиця стилів

містить простий текст, і можливо створити її за допомогою будь-якого текстового редактора.

2) Пов'язування XSL-таблиці стилів з XML-документом. Можливо пов'язати XSL-таблицю стилів з XML-документом, включивши у документ інструкцію з обробки `xml-stylesheet`, яка має таку узагальнену форму запису:

```
<?xml-stylesheet type="text/xsl" href=XSLFilePath?>
```

Тут `XSLFilePath` є включений у лапки URL, який вказує місцезнаходження файлу таблиці стилів. Ви можете використовувати повний URL, наприклад:

```
<?xml-stylesheet type="text/xsl"
href="http:/www.my_domain.com/Inventory.xsl"?>
```

Частіше використовують неповний URL, який задає місцезнаходження XML-документа, що містить інструкцію з обробки `xml-stylesheet`, наприклад:

```
<?xml-stylesheet type="text/xsl" href="Inventory.xsl"?>
```

Відносний URL зустрічається частіше, оскільки зазвичай зберігає файл таблиці стилів у тій самій папці, де зберігається XML-документ, або в одній із вкладених в неї папок. Хоча можна зв'язати XSL-таблицю стилів з використанням повного URL, таблиця стилів при цьому повинна розміщуватися на тому ж домені, що і XML-документ, з яким він пов'язаний. Наприклад, якщо домен http://mspress.microsoft.com/ містить XML-документ, то і XSL-таблиця стилів повинна розміщуватися на тому ж домені.

Як правило, інструкція з обробки `xml-stylesheet` додається в пролог XML-документа за оголошенням XML. Якщо ви пов'язали XSL-таблицю стилів із XML-документом, можна відкрити цей документ безпосередньо в браузері, який відображатиме XML-документ з використанням інструкцій з перетворення стилів, що містяться в таблиці. На відміну від каскадних таблиць стилів, якщо ви пов'язуєте з XML-документом більше, ніж одну XSL-таблицю стилів, браузер використовує

першу таблицю і ігнорує всі останні. Якщо ви пов'яжете з XML-документом і CSS-таблицю, і XSL-таблицю стилів, браузер використає тільки XSL-таблицю стилів.

Використання одного шаблону XSL. На відміну від CSS, що містить правила, XSL-таблиця стилів включає один або декілька шаблонів, кожен з яких містить інформацію для відображення в певній гілці елементів в XML-документі. Існує можливість створити просту XSL-таблицю стилів, яка включає тільки один шаблон. Цей шаблон містить інформацію для відображення всього документа. Далі поданий перший приклад XSL-таблиці стилів example_15.xsl. Ця таблиця стилів пов'язана з XML-документом:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
   <xsl:template match="/">
    <H2>Book Description</H2>
    <SPAN STYLE="font-style:italic">Author: </SPAN>
    <xsl:value-of select="BOOK/AUTHOR"/><BR/>
    <SPAN STYLE="font-style:italic">Title: </SPAN>
    <xsl:value-of| select="BOOK/TITLE"/><BR/>
    <SPAN STYLE="font-style:italic">Price: </SPAN>
    <xsl:value-of select="BOOK/PRICE"/><BR/>
    <SPAN STYLE="font-style:italic">Binding type: </SPAN>
    <xsl:value-of select="BOOK/BINDING"/><BR/>
    <SPAN STYLE="font-style:italic">Number of pages: </SPAN>
    <xsl:value-of select="BOOK/PAGES"/>
 </xsl:template>
</xsl:stylesheet>.
```

Приклад відповідного example_15.xml файлу наведений нижче:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="example_15.xsl"?>
<BOOK>
   <TITLE>Moby-Dick</TITLE>
   <AUTHOR>
      <FIRSTNAME>Herman</FIRSTNAME>
      <LASTNAME>Melville</LASTNAME>
   </AUTHOR>
   <BINDING>hardcover</BINDING>
   <PAGES>724</PAGES>
   <PRICE>$9.95</PRICE>
</BOOK>.
```

На рис. 4.1 показано як браузер відображує XML-документ відповідно до інструкцій з таблиці стилів.

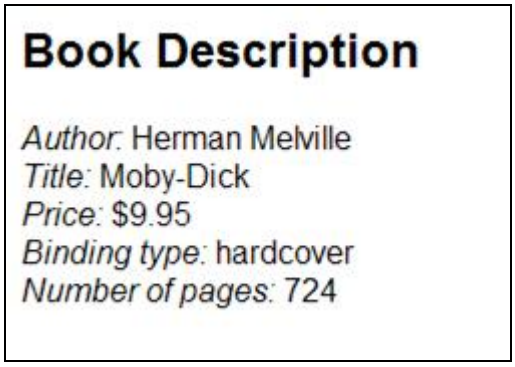

Рисунок 4.1– Результат відображення перетвореного xml-документа

Кожна XSL-таблиця стилів повинна мати елемент Документ, поданий нижче. Елемент Документ, відомий як кореневий елемент, є XML-елементом верхнього рівня, який містить всі інші елементи.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
   <!-- один або декілька елементів шаблону .-->
</xsl:stylesheet>
```

Елемент Документ xsl:stylesheet служить не лише сховищем для інших елементів, але також ідентифікує документ як XSL-таблицю стилів. Цей елемент є одним із XSL-елементів спеціального призначення, що використо-вуються в таблиці стилів. Всі XSL-елементи належать простору імен xsl – тобто ви передуєте імені кожного XSL-елемента префіксом xsl:, що позначає простір імен. Ви визначаєте цей простір імен в початковому тегу елемента xsl:stylesheet, наприклад, таким чином:

```
xmlns:xsl="http://www.w3.org/TR/WD-xsl"
```

Це визначення дозволяє вам використовувати простір імен всередині елементів таблиці стилів.

Елемент Документ `xsl:stylesheet` XSL-таблиці стилів повинен містити один або декілька шаблонів елементів, які скорочено називатимемо шаблонами. Даний Документ містить тільки один шаблон, який має таку форму:

```
<xsl:template match="/">
   <!-- дочірні елементи . -->
</xsl:template>.
```

Браузер використовує шаблон для відображення певної гілки елементів в ієрархії XML-документа, з яким ви пов'язуєте таблицю стилів. Атрибут match шаблону вказує на певну гілку. (Атрибут `match` аналогічний селектору в правилі CSS). Значення атрибута `match` носить назву зразка (pattern). Зразок в даному прикладі ("/") представляє кореневий елемент всього XML-документа. Цей шаблон, таким чином, містить інструкції для відображення всього XML-документа.

Кожна XSL-таблиця стилів повинна містити один і лише один шаблон з атрибутом match, який має значення "/". Також можливо включити один або декілька додаткових шаблонів з інструкціями для відображення певних підлеглих гілок в структурі XML-документа; кожна з них повинна мати зразок, що відповідає певній гілці.

Кореневий зразок ("/") не представляє елемент Документ (або кореневий елемент) XML-документа. Він представляє весь документ, для якого елемент Документ є дочірнім. Далі наведений повний опис шаблону з даної таблиці стилів:

```
<xsl:template match="/">
   <H2>Book Description</H2>
   <SPAN STYLE="font-style:italic">Author: </SPAN>
   <xsl:value-of select="BOOK/AUTHOR"/><BR/>
   <SPAN STYLE="font-style:italic">Title: </SPAN>
   <xsl:value-of select="BOOK/TITLE"/><BR/>
   <SPAN STYLE="font-style:italic">Price: </SPAN>
   <xsl:value-of select="BOOK/PRICE"/><BR/>
   <SPAN STYLE="font-style:italic">Binding type: </SPAN>
   <xsl:value-of select="BOOK/BINDING"/><BR/>
   <SPAN STYLE="font-style:italic">Number of pages: </SPAN>
   <xsl:value-of select="BOOK/PAGES"/>
</xsl:template>.
```

Шаблон містить два види XML-елементів.

XML-елементи, що представляють HTML-розмітку. Прикладами подібного виду XML-елемента з даної таблиці стилів є:

```
<H2>Book Description</H2>,який відображує заголовок другого рівня

<SPAN STYLE="font-style:italic">Author: </SPAN>,який
відображає  блок тексту, набраного курсивом (Author:), і
<BR/>, який створює пустий рядок.
```

Всі ці XML-елементи є коректно сформованими і представляють стандартні HTML-елементи. Браузер просто копіює кожен HTML-елемент безпосередньо на вихід HTML, який сприймає і відображує їх. Кожен з елементів, що представляють HTML-розмітку, має бути коректно сформованим XML-елементом, а також стандартним HTML-елементом. Отже, ви не можете використовувати HTML-конструкції, які не є коректно сформованим XML, такі, як елементи, що складаються тільки з початкового тегу. Наприклад, щоб задати елемент переведення рядка в HTML, ви не можете просто ввести
, як ви це робите для HTML-сторінки. Замість цього ви повинні використовувати коректно сформований тег порожнього XML-елемента
.

XSL-елементи. Приклади XSL-елементів з даної таблиці стилів є елементами `xsl:value-of`, наприклад:

```
<xsl:value-of select="BOOK/AUTHOR"/>.
```

Браузер відрізняє XML-елемент від елемента, який представляє HTML, оскільки перший має як префікс опис простору імен `xsl:`. XSL-елементи в шаблоні не копіюються на вихід HTML. Вони лише містять інструкції з вибору і модифікації даних XML, або використовуються для виконання інших завдань.

XSL-елемент `value-of` додає текстовий вміст певного XML-елемента, а також будь-яких його дочірніх елементів, які він має, – у вихідний модуль HTML, який сприймається і відображується браузером. Ви вказуєте певний XML-елемент, що задаєте зразком, який привласнюєте атрибуту `select` XSL-елемента `value-of`. У прикладі елемента, розглянутого вище, `value-of` атрибута `select` привласнений зразок "BOOK/AUTHOR", що приводить до виведення текстового вмісту елемента `AUTHOR` XML-документа. Текстовий вміст елемента `AUTHOR`

складається з символьних даних, що належать двом його дочірнім елементам, FIRSTNAME і LASTNAME.

Зверніть увагу, що XML-елемент у зразку задається за допомогою оператора шляху (в даному випадку BOOK/AUTHOR), який визначає місцезнаходження елемента в ієрархії XML-документа. (Оператор шляху схожий на шлях до файлу, який операційна система використовує для вказівки місцезнаходження файлу або папки).

Головний момент, на який потрібно звернути увагу, полягає в тому, що оператор шляху в значенні атрибута select відноситься до поточного елемента. Кожен контекст всередині XSL-таблиці стилів відноситься до поточного елемента. Оскільки даний приклад шаблону відноситься до кореневого елемента всього документа (за допомогою установки атрибута match="/"), поточним "елементом" для даного шаблону є кореневий елемент документа. (У даному випадку поточний елемент не володіє відповідним літералом, а є батьком елемента Документ). Таким чином, всередині цього шаблону оператор шляху BOOK/AUTHOR вказує на елемент AUTHOR, вкладений в елемент BOOK, вкладений у кореневий елемент документа. (Оператор шляху в значенні атрибута select аналогічний неповному шляху до файлу, що задає місцезнаходження файлу щодо поточної робочої папки).

Якщо ви опустите атрибут select для XSL-елемента value-of, елемент здійснюватиме виведення текстового вмісту плюс текстовий вміст всіх дочірніх елементів у поточний елемент. (У нашому прикладі, оскільки поточним є кореневий елемент, пропуск атрибута select приведе до виведення всіх символьних даних у XML-документ.)

Метою поданого в даному прикладі шаблону елементів є відображення тексту назви для кожного з дочірніх XML-елементів у документі (AUTHOR, TITLE, PRICE, BINDING і PAGES) плюс текстового вмісту кожного елемента. Зверніть увагу, що порядок елементів value-of у шаблоні визначає порядок, в якому браузер відображує ці елементи. Таким чином, навіть із цієї простої таблиці стилів ви можете зрозуміти, що XSL-таблиця стилів є набагато гнучкішою, ніж CSS, яка завжди відображує елементи в тому порядку, в якому вони знаходяться в

документі. Отже, XSL-таблиця стилів повідомляє браузеру, як відображувати XML-документ шляхом вибіркового перетворення XML-елементів у блок HTML-розмітки, який сприймається і відображується браузером аналогічно розмітці, що міститься на HTML-сторінці. Відмітимо, що вам не потрібно включати в XSL-шаблон елементи, що представляють елементи HTML або `BODY`, які є стандартними складовими частинами HTML-сторінки, оскільки браузер сам ефективно їх формує. На рис. 4.2 показано як браузер генерує першу частину блоку HTML-розмітки для документа і таблиці стилів.

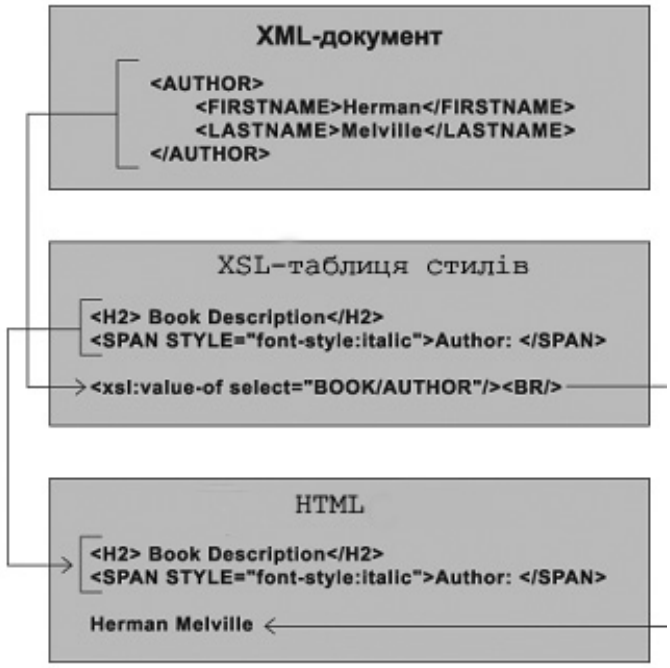

Рисунок 1– Генерація html-коду браузером

XML-документ

```
<AUTHOR>
  <FIRSTNAME>Herman</FIRSTNAME
  <LASTNAME>Melville</LASTNAME>
  </AUTHOR>.
```

XSL-таблиця стилів

```
<H2>Book Description</H2>
  <SPAN STYLE="font-style:italic">Author: </SPAN>
  <xsl:value-of select="BOOK/AUTHOR"/><BR/>.
```

HTML, що відображується

```
  <H2>Book Description</H2>
```

```
<SPAN STYLE="font-style:italic">Author: </SPAN>
Hermann Melville.
```

Відображення змінного числа елементів. У прикладі, що розглянуто вище, XML-документ містив тільки один елемент BOOK. У випадку, якщо документ містить декілька елементів BOOK, розглянута методика здатна відображувати тільки один з елементів. Візьмемо для прикладу ще один XML-документ:

```
<INVENTORY>
   <BOOK>
   <TITLE>The Adventures of Huckleberry Finn</TITLE>
   <AUTHOR>
      <FIRSTNAME>Mark</FIRSTNAME>
      <LASTNAME>Twain</LASTNAME>
   </AUTHOR>
   <BINDING>mass market paperback</BINDING>
   <PAGES>298</PAGES>
   <PRICE>$5.49</PRICE>
   </BOOK>
   <BOOK>
   <TITLE>The Adventures of Tom Sawyer</TITLE>
   <AUTHOR>
      <FIRSTNAME>Mark</FIRSTNAME>
      <LASTNAME>Twain</LASTNAME>
   </AUTHOR>
   <BINDING>mass market paperback</BINDING>
   <PAGES>205</PAGES>
   <PRICE>$4.75</PRICE>
   </BOOK>
   <BOOK>
   <TITLE>The Ambassadors</TITLE>
   <AUTHOR>
      <FIRSTNAME>Henry</FIRSTNAME>
      <LASTNAME>James</LASTNAME>
   </AUTHOR>
   <BINDING>mass market paperback</BINDING>
   <PAGES>305</PAGES>
   <PRICE>$5.95</PRICE>
   </BOOK> </INVENTORY>.
```

Припустимо, що таблиця стилів, яка використана для відображення цього документа, містить такий шаблон:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
   <xsl:template match="/">
      <H2>Book Description</H2>
      <SPAN STYLE="font-style:italic">Author: </SPAN>
      <xsl:value-of select="INVENTORY/BOOK/AUTHOR"/><BR/>
      <SPAN STYLE="font-style:italic">Title: </SPAN>
      <xsl:value-of select="INVENTORY/BOOK/TITLE"/><BR/>
      <SPAN STYLE="font-style:italic">Price: </SPAN>
      <xsl:value-of select="INVENTORY/BOOK/PRICE"/><BR/>
     <SPAN STYLE="font-style:italic">Binding type: </SPAN>
      <xsl:value-of select="INVENTORY/BOOK/BINDING"/><BR/>
  <SPAN STYLE="font-style:italic">Number of pages: </SPAN>
      <xsl:value-of select="INVENTORY/BOOK/PAGES"/>
   </xsl:template>
</xsl:stylesheet>.
```

Зразок привласнених кожному атрибуту значень select, починається з вказівки елемента Документ, у даному випадку `INVENTORY` (наприклад, `"INVENTORY/BOOK/AUTHOR"`). Проте кожен зразок відповідає трьом різним елементам. Наприклад, `"INVENTORY/BOOK/AUTHOR"` відповідає елементу `AUTHOR` для всіх трьох елементів `BOOK`. У подібній ситуації браузер використовує тільки перший із відповідних елементів. Таким чином, таблиця стилів відображуватиме вміст тільки першого елемента `BOOK`, як показано на рис.4.3.

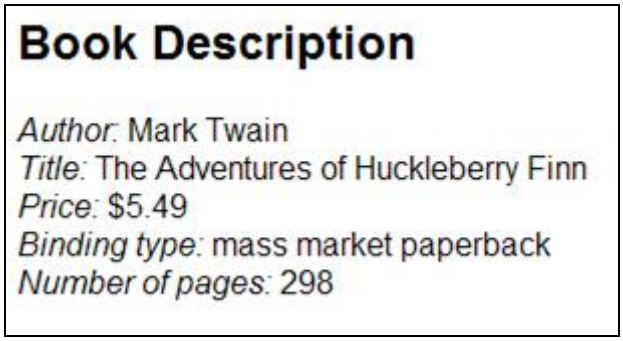

Рисунок 2 – Приклад відображення змінного числа елементів

Щоб відображувати всі елементи, що відповідають зразку, слід використовувати XSL-елемент for-each, який викликає повторне відображення для кожного з елементів, що містяться в XML-файлі. XSL-таблиця стилів, представлена в example_16.xsl, демонструє дану методику. Ця таблиця стилів пов'язана з XML-документом.

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
   <xsl:template match="/">
      <H2>Book Inventory</H2>
      <xsl:for-each select="INVENTORY/BOOK">
         <SPAN STYLE="font-style:italic">Title: </SPAN>
         <xsl:value-of select="TITLE"/><BR />
         <SPAN STYLE="font-style:italic">Author: </SPAN>
         <xsl:value-of select="AUTHOR"/><BR />
       <SPAN STYLE="font-style:italic">Binding type: </SPAN>
         <xsl:value-of select="BINDING"/><BR />
  <SPAN STYLE="font-style:italic">Number of pages: </SPAN>
         <xsl:value-of select="PAGES"/><BR />
         <SPAN STYLE="font-style:italic">Price: </SPAN>
         <xsl:value-of select="PRICE"/><P />
      </xsl:for-each>
   </xsl:template>
</xsl:stylesheet>.
```

Шаблон у таблиці стилів із даного прикладу містить такий елемент `for-each`:

```
<xsl:for-each select="INVENTORY/BOOK">
   <SPAN STYLE="font-style:italic">Title: </SPAN>
   <xsl:value-of select="TITLE"/><BR />
   <SPAN STYLE="font-style:italic">Author: </SPAN>
   <xsl:value-of select="AUTHOR"/><BR />
   <SPAN STYLE="font-style:italic">Binding type: </SPAN>
   <xsl:value-of select="BINDING"/><BR />
  <SPAN STYLE="font-style:italic">Number of pages: </SPAN>
   <xsl:value-of select="PAGES"/><BR />
   <SPAN STYLE="font-style:italic">Price: </SPAN>
   <xsl:value-of select="PRICE"/><P />
</xsl:for-each>.
```

Елемент for-each виконує два основні завдання:

– здійснює виведення блоку елементів, що містяться усередині елемента for-each, повторюючи його для кожного XML-елемента в документі, що відповідає зразку, привласненому атрибуту select елементу for-each. У даному прикладі цикл виконується один раз для кожного елемента BOOK, знайденого в елементі Документ з ім'ям INVENTORY. Зразок, що привласнюється атрибуту select, працює так само, як і зразок, що привласнюється атрибуту select елементу value-of;
– в елементі for-each задається поточний елемент, що встановлюється атрибутом select елемента for-each (/INVENTORY/BOOK у нашому прикладі вказує на елемент BOOK усередині елемента INVENTORY, що входить у кореневий елемент документа) таким чином:

```
<xsl:stylesheet xmlns:xsl=http://www.w3.org/TR/WD-xsl>
   <xsl:template match="/">
      <!-- Тут поточним є кореневий "елемент"
         документа, "/". -->
   <xsl:for-each select="INVENTORY/BOOK">
   <!--Тут поточним є елемент /INVENTORY/BOOK-->
   </xsl:for-each>
   </xsl:template>
</xsl:stylesheet>
```

Аналогічно в елементі for-each кожен дочірній елемент може бути вибраний шляхом завдання зразка, що містить тільки ім'я елемента, наприклад: <xsl:value-of select="TITLE"/>.

У результаті виводяться дані зі всіх елементів BOOK, знайдених у документі, незалежно від того, скільки цих елементів містить документ. На рисунку 4.4 показано як виглядає результат у браузері. (У вікні відображують тільки перші три елементи BOOK; щоб відображувати останні, необхідно здійснити прокрутку вниз).

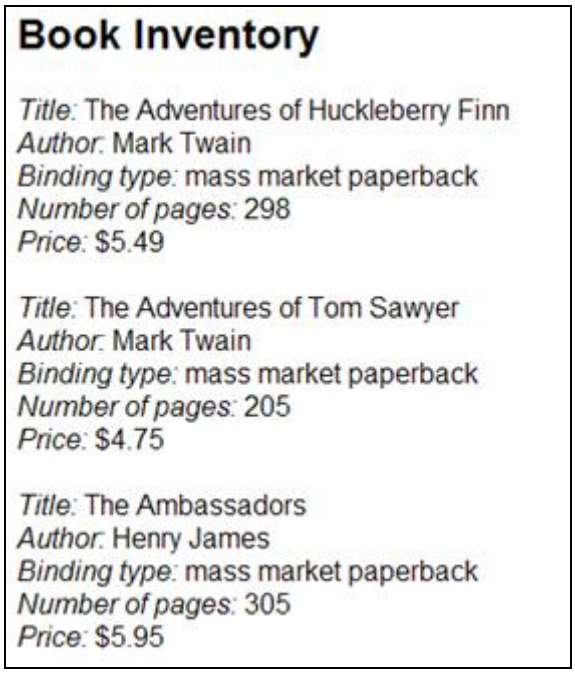

Рисунок 3 – Відображення елементів документа

5 МОВА ЗАВДАННЯ ШЛЯХІВ XPath

Згідно з рекомендаціями консорціуму W3C, з метою форматування і перетворення XML-документів використовується мова перетворень XSLT, яка є одним з напрямків XSL. XSL (eXtensible Stylesheet Language) і являє собою сімейство рекомендацій консорціуму W3C, яке описує мови перетворення і візуалізації XML-документів. Основні напрямки XSL:

- XSL Transformations (XSLT) – мова перетворень XML-документів.
- XPath – мова шляхів і виразів, що використовується в XSLT для доступу до окремих частин XML-документа.

XPath є мова завдання шляхів до елементів XML-документа. Розроблено для організації доступу до частин документа XML в файлах трансформації XSLT і є стандартом консорціуму W3C. Для адресації частин XML-документа у вираз мови XPath використовує позначення шляху, схоже на позначення в URL-адресу. Виконується оцінка вираження для того, щоб задати об'єкту один з типів: *тип набору вузлів, логічний, числовий або строковий тип.* Наприклад, вираз `book/author` повертає набір вузлів елементів <Author>, що містяться в елементах <book>, за умови, що такі елементи оголошені в початковому XML-документі. Крім того, вираз XPath може містити предикати (критерії фільтра) або виклики функцій. Наприклад, вираз `book [@ type = "Fiction"]` посилається на елементи <book>, для яких атрибут type приймає значення "Fiction"

Мова запитів до елементів XML документа XPath дозволяє здійснювати вибірку вершин або набору вершин з дерева XML-документа по заданих користувачем критеріями (положенню, відносного положення, типу, вмісту і ін.). Мова XPath розрізняє різні типи вершин, які також називаються вузлами: кореневий вузол (весь XML документ), елемент, текст, атрибут, коментар, інструкції обробки, простір імен. XPath виконує пошук за кінцевим деревом XML, тобто з урахуванням підставлених значень сутностей і значень за замовчуванням. Таким чином можна відрізнити конструкції, що генеруються від вмісту XML-документу.

Опис послідовності переходів між ду вузлами XML дерева називається ***маршрутом***. У XPath существуют *прості маршрути, складові*

маршрути і складні маршрути. Простий маршрут визначає шуканий вузол дерева XML без вказівки переходів. Складовий маршрут містить об'єднання декількох маршрутів. Результатом складеного маршруту буде об'єднаннями результатів вихідних маршрутів. Складний маршрут містить комбінацію простих маршрутів у вигляді послідовності переходів між вузлами XML дерева.

5.1 Путі розташування в XPath

Шлях розташування являє собою вираз XPath, який використовується для вибору набору вузлів, що відносяться до вузла контексту. Результатом оцінки вираження шляху розташування є набір вузлів, що містить вузли, визначені шляхом розташування. В шлях розташування можуть рекурсивно входити вирази, які використовуються для фільтрації наборів вузлів. Синтаксично шлях розташування складається з одного і більше кроків визначення розташування, відокремлених один від одного косою рисою (/):

```
locationstep / locationstep / locationstep
```

Кожен крок визначення розташування в порядку черги вибирає набір вузлів, що відносяться до вузла контексту, т. е. до вузла, заданої попереднім кроком визначення розташування. Шлях розташування, виражений подібним чином, є *відносним шляхом* розташування. *Абсолютний шлях розташування* бере початок від кореневого елемента:

```
/ Locationstep / locationstep / locationstep
```

Кроки визначення розташування в шляху розташування оцінюються зліва направо. Крайній лівий крок визначення розташування вибирає набір вузлів, що відносяться до вузла контекста. Ці вузли потім стають вузлами контексту для обробки наступного кроку визначення розташування. Обробка вузлів і зміна вузла контексту повторюється до тих пір, поки не будуть оброблені всі кроки визначення розташування. Шлях розташування може мати як повний, так і скорочений синтаксис. Шлях розташування з повним синтаксисом виглядає наступним чином:

```
ось :: вузол [предикат]
```

У цьому синтаксисі елемент «ось» визначає, як вузли, обрані кроком визначення розташування, розташовуються щодо вузла контексту. Елемент «вузол» визначає тип вузла і розгорнуте ім'я вузлів, обраних кроком визначення розташування. Елемент «предикат» є критерієм фільтра, використовуваним для подальшого уточнення вибору вузлів в кроці визначення розташування. Предикати є необов'язковими елементами. В скороченому синтаксисі шляху розташування покажчик осі (axis: :) виражається в кроці визначення розташування неявно, замість цього він може бути охарактеризована ярликів. В таблиці наводиться список деяких скорочень синтаксису шляхів:

Повний синтаксис	Скорочений синтаксис
child:: *	*
attribute:: *	@ *
/ Descendant-or-self:: node ()	//
self:: node ()	.
parent:: node ()	..

5.2 Прості і складові маршрути

Виділяють наступні прості маршрути:

- маршрут до кореневого вузла – позначається за допомогою знака "/" і вибирає все дерево XML;
- маршрут до елемента – позначається ім'ям елемента і вибирає всі елементи із зазначеним ім'ям щодо поточного контексту;
- маршрут до атрибута – позначається у вигляді комбінації символу "@" і назви ознаки і вибирає вміст атрибута щодо поточного контексту;
- маршрут до текстового вмісту – позначається у вигляді функції "text ()" і вибирає весь текстовий вміст елементів щодо поточного контексту;

– маршрут до коментарю – позначається у вигляді функції "Comment ()" і вибирає вміст XML коментарів щодо поточного контексту.

Для всіх маршрутів крім маршруту до кореневого вузла важливе значення поточного контексту, тобто розташування щодо якого здійснюється пошук в дереві XML. Вказівка поточного контексту виконується за допомогою символів "//", наприклад, "// @ person".

5.3 Осі вибірки XPath

Крок визначення розташування задає набір вузлів (node-set) по відношенню до вузла контексту. Крок доступу складається з трьох частин: необов'язковою осі, перевірки вузла і необов'язкового предиката.

Синтаксис кроку визначення розташування виглядає наступним чином: ім'я осі, подвійна двокрапка, елемент перевірки вузла, нуль або додаткові предикати, кожен з яких укладено в квадратні дужки. Базова форма синтаксису має наступний вигляд:

```
ось :: вузол [предикат]
```

Ось визначає деревоподібний зв'язок між вузлом контексту і вузлами, які вибираються кроком визначення розташування. Іншими словами, ось вказує загальний напрямок, в якому виконується крок визначення розташування по відношенню до вузла контексту. За крок визначення розташування вісь є необов'язковим елементом. Якщо не вказати ось, за замовчуванням їй присвоюється значення child ::. Крім того, деякі осі мають ярлики, наприклад символ @ є ярликом для осі атрибутів. Список осей представлений в наступній таблиці.

Осі	Опис
ancestor ::	Предки вузла контексту. До предків вузла контексту відносяться батьківський вузол вузла контексту, батько батька і т. д .; таким чином, вісь ancestor :: завжди включає в себе кореневий вузол, якщо тільки контекстний вузол сам не є кореневих вузлом.
ancestor-orself ::	Вузол контексту і його предки. Ось ancestor-or-self :: завжди включає кореневий вузол.

attribute ::	Атрибути контекстного вузла. Ось буде порожня, якщо вузол контекст не елемент.
child ::	Дочірні елементи вузла контексту. Дочірнім є будь-який вузол, розташований на дереві безпосередньо під вузлом контексту. Однак ні вузли атрибутів, ні вузли простору імен не розглядаються в якості дочок вузла контексту.
descendant ::	Нащадки контекстного вузла. Нащадком є дочірній об'єкт або дочірній об'єкт дочірнього об'єкта і т. д .; таким чином, вісь descendant :: ніколи не містить вузлів атрибутів або простору імен.
descendant-or-self ::	Вузол контексту і його нащадки.
following ::	Всі вузли, розташовані на дереві після вузла контексту, за винятком нащадків, вузлів атрибутів і вузлів простору імен.
Following-sibling ::	Всі наступні елементи вузла контексту з загальним батьком. На осі following-sibling :: вказуються лише ті дочірні об'єкти батьківського вузла, які відображаються на дереві після вузла контексту. На цій осі не вказуються всі інші дочірні об'єкти, розташовані перед вузлом контексту. Якщо вузол контексту є вузлом атрибута або вузлом простору імен, вісь followingsibling :: порожня.
namespace ::	Вузли простору імен вузла контексту. На кожний простір імен, розташований в області вузла контексту, доводиться по одному вузлу простору імен. Ось буде порожня, якщо вузол контекст не елемент.
parent ::	Батько вузла контексту, якщо такий є. Батьком є вузол, розташований на дереві безпосередньо над вузлом контексту.
preceding ::	Всі вузли, розташовані на дереві перед вузлом контексту, за винятком предків, вузлів атрибутів і вузлів простору імен. Щоб краще зрозуміти, що таке попередня ось, потрібно уявити собі всі вузли, вміст яких у всій своїй повноті розміщується до початку вузла контексту.

Preceding-sibling ::	Всі попередні елементи вузла контексту із загальним батьком. На осі preceding-sibling :: вказуються лише ті нащадки батьківського вузла, які відображаються на дереві до вузла контексту. На цієї осі не вказуються всі інші дочірні об'єкти, розташовані після вузла контексту. Якщо вузол контексту є вузлом атрибута або вузлом простору імен, вісь precedingsibling :: порожня.
self ::	Тільки сам контекстний вузол

Вузол визначає тип вузла або розгорнуте ім'я вузлів, які спочатку вибираються кроком визначення розташування. Перевірка вузлів є єдиною необхідною частиною кроку визначення розташування XPath. З урахуванням цієї обставини розуміння її є найважливішою умовою успішного застосування виразів XPath.

Існує три основних типи перевірки вузлів: перевірка імені, в якій використовується розгорнуте ім'я і зв'язок цього імені з заданою віссю для вказівки вузлів, які необхідно виділити; перевірка типу вузла, при якої вузли вибираються строго за типами вузлів; перевірка цільової інструкції по обробці, при якій виділяються тільки ті інструкції по обробці вузлів, які відповідають вказаним типу

Предикат використовує вираз XPath (умова, яка повинна бути виконано) для подальшого уточнення набору вузлів, обраного кроком визначення розташування. Предикат являє собою фільтр, який визначає критерій вибору для подальшого уточнення списку вузлів-кандидатів. Предікат є необов'язковим елементом. Якщо предикат не вказано, в кроці вкажіть місце, куди немає квадратних дужок ([]). Власне процес фільтрації включає в себе послідовне обчислення предиката для кожного вузла в наборі. Кожен раз, коли предикат обчислюється для вузла, відбувається наступне:

Контекстним вузлом є вузол, для якого предикат обчислюється в даний момент. Розмір контексту представляє собою число вузлів в наборі, для якого обчислюється предикат. Позиція контексту є положення контекстного вузла в наборі вузлів. Цей останній контекст, положення контекстного вузла в наборі вузлів, є відносним і залежить від напрямку, в

якому вісь, задана в кроці визначення розташування даних, обходить дерево документа. Зазвичай вісь обходить дерево в прямому або зворотному напрямку. Набір вузлів, обираний кроком визначення розташування, є результатом створення вихідного набору вузлів, заснованого на зв'язку між віссю і перевіркою вузла, і подальшої фільтрації вихідного набору по кожному включеному предикату. вихідний набір складається з вузлів, для яких виконуються два наступних умови.

Вузли мають зв'язок з вузлом контексту, заданим віссю. Вузли мають тип і розгорнуте ім'я, задане перевіркою вузла. Потім вираз XPath використовує перший предикат в кроці визначення розташування, виконуючи фільтрацію вихідного набору вузлів для створення нового набору вузлів. Далі вираз XPath використовує другий предикат для фільтрації набору вузлів, отриманого за допомогою першого предиката.

Процес фільтрації повторюється до тих пір, поки вираз XPath НЕ оцінить всі предикати. Набір вузлів, отриманий в результаті застосування всіх предикатів, є набором вузлів, обраним кроком визначення розташування.

5.4 Контекст XPath-виразів

Оцінка вираження XPath залежить від контексту, до якого звертається вираз. Контекст складається з вузла, за яким оцінюється вираз, а також з пов'язаної з ним середовища, що включає наступні компоненти:

- Положення вузла контексту в порядку документа щодо однорівневих елементів.
- Розмір контексту, т. е число однорівневих елементів вузла контексту плюс один.
- Прив'язки змінних, з якими вирішуються посилання на змінні.
- Бібліотека функцій.
- Оголошення просторів імен в області вираження.

Для того, щоб краще зрозуміти концепцію контексту, уявіть дерево, що містить вузли. Запит всіх вузлів X з кореня дерева поверне один набір результатів, в той час, як запит цих вузлів з гілки дерева поверне інший

набір результатів. Таким чином, результат вираження залежить від контексту, до якого воно звертається при виконанні.

Вирази XPath можуть зіставити спеціальні шаблони в одному конкретному контексті, потім повернути результати і виконати додаткові звернення до контексту повернутих вузлів. Це забезпечує виразами XPath виняткову гнучкість при пошуку по дереву документа.

5.5 Базові вираження XPath

Нижче наведено базові типи виразів XPath:

– Поточний контекст;
– Корінь документа;
– Кореневий елемент;
– Рекурсивний спуск;
– Конкретний елемент.

Поточний контекст – вираз з префіксом у вигляді точки і косою рисою явно використовує в якості контексту поточний контекст. Наприклад, такий вираз посилається на всі елементи <author> всередині поточного контексту: ./author Цей вислів еквівалентно наступному: Author.

Корінь документа – вираз з префіксом у вигляді косої риски (/) використовує в якості контексту корінь дерева документа. наприклад, такий вираз посилається на всі елементи <bookstore> в корені основних напрямів: / bookstore. ***Кореневий елемент*** – вираз, що використовує косу риску і зірочку (/ *), використовує в якості контексту кореневий елемент. Наприклад, такий вираз знаходить кореневий елемент документа: / *. ***Рекурсивний спуск*** – вираз, що використовує подвійну косу рису (//), вказує на пошук, який може включати нуль або більше рівнів ієрархії. Якщо цей оператор відображається на початку шаблону, то контекст є відносним по відношенню до кореня документа. Наприклад, такий вираз посилається на всі елементи <author> всередині в будь-якому місці всередині поточного документа: // author. Префікс .// вказує, що контекст починається на рівні ієрархії, зазначеному в поточному контексті.

Конкретні елементи – вираз, який починається з імені елемента, посилається на запит конкретного елемента, який починається від поточного вузла контексту. Наприклад, такий вираз посилається на елемент <background.jpg> всередині елемента <images> в поточного вузлі контексту: images / background.jpg

Наступне вираз посилається на колекцію елементів <book> всередині елементів <bookstore> в поточного вузлі контексту: bookstore / book. Представлене далі вираз посилається на всі елементи <first.name> всередині поточного вузла контексту: first.name.

ЛІТЕРАТУРА

1. Проценко О.Б. Web-програмування та web-дизайн. Технологія XML. Навчальний посібник. – Суми: Видавництво СумДУ, 2009. – 126 с.
2. В.П. Молчанов, О. К. Пандорін. Технології розробки WEB-ресурсів: навчальний посібник. – Харків: ХНЕУ ім. С. Кузнеця, 2019. – 130 с.
3. В.П. Молчанов. Основи проектування WEB-видань: навч. посіб. Харків: ХНЕУ ім. С. Кузнеця, 2017. – 150 с.
4. О.О. Мосіюк. WEB-технології. Частина 1. Житомир: Вид-во ЖДУ ім. Івана Франка, 2020. – 56 с.
5. Степаненко О.О. Програмування Інтернет-застосувань: конспект лекцій для студентів спеціальності «Інженерія програмного забезпечення» усіх форм навчання. Запоріжжя, 2016. – 66 с.
6. О.О.Шумейко. Конспект лекцій з дисципліни «Технології створення Web-застосувань» для здобувачів вищої за ОПП «Інженерія програмного забезпечення» зі спеціальності 121 – «Інженерія програмного забезпечення». – Кам’янське: ДДТУ, 2019– 124 с.