УДК 681.3.06

М.М. Колодницький, проф.
Житомирський інженерно-технологічний інститут

МОДЕЛЬ ДАНИХ – ЩО ВИБРАТИ: XML ЧИ РЕЛЯЦІЙНІ БАЗИ ДАНИХ?

Робота присвячена питанню порівняння двох технологій – Extensible Markup Language (XML) та реляційних баз даних (БД) – при розробці моделей даних програм з метою представлення даних при їх зберіганні, обробці та передачі між прикладними програмами. Після стислого викладення основних концепцій XML запропоновано метод представлення (моделювання) XML документа за допомогою набору таблиць БД. Викладений метод може бути корисним не тільки при розробці програм, що ґрунтуються на альтернативних до XML технологіях, але і для кращого розуміння спільних та відмінних рис цих двох технологій.

Вступ

При розробці сучасних програмних систем останнім часом помітилася тенденція, надаючи дань моді, використовувати мову розмітки (Extensible Markup Language, XML) [1–2] як базову технологію для представлення даних, в незалежності від області застосування прикладної програми. Така ситуація може бути пояснена успіхами застосування XML-технології в Інтернет застосуваннях взагалі та з метою міжплатформного обміну даних зокрема. Завдяки такому широкому застосуванню XML і з'явилися спроби поширення цієї технології на сфери, які напочатку не були призначені для цього, наприклад, звичайні desktop-застосування або на розробку інформаційних систем, де традиційно використовуються технології, основані на реляційних базах даних (БД) [3–5].

Таким чином, можна з впевненістю стверджувати, що на сьогодні існує певна конкуренція між цими двома технологіями – XML та БД [5–8]. Робота присвячена питанню порівняння цих технологій при розробці моделей даних програм з метою представлення даних при їх зберіганні, обробці та передачі між програмами. В роботі, після стислого викладення основних концепцій XML, запропоновано метод представлення (моделювання) XML-документа за допомогою набору таблиць БД. Викладений метод може бути корисним при розробці програм, побудованих на альтернативних до XML технологіях. Представлений в статті матеріал може бути також корисним для кращого розуміння спільних та відмінних рис цих двох технологій.

1. Основні концепції XML

Перед тим, як провести порівняння концепцій XML та БД, коротко нагадаємо основні ідеї, що лежать в основі XML [1, 2, 9]. Для цього розглянемо приклад [10]. Нехай потрібно представити архів поштових повідомлень. В XML його структура може бути задана за допомогою наступного набору правил:

```
<!ELEMENT Archive (Message)*>
<!ELEMENT Message (Head, Body)>
<!ELEMENT Head (To+, From, Subject?)>
<!ELEMENT Body (#PCDATA)>
<!ELEMENT To (Title, Content)*>
<!ELEMENT From (Title, Content)*>
<!ELEMENT Subject (Title, Content)*>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Content (#PCDATA)>
```

При цьому "елемент" **(element)** – задається "іменем елемента", наприклад, Archive або Message. "Зміст елемента" **(content)** може бути:

– або "порожнім" **(empty)**;

- або бути "значенням певного типу" **(value)** – звичайно довгий рядок тексту #PCDATA;
- або "бути списком вкладених елементів" **(embedded elements)**, тобто "елементів наступного рівня ієрархії".

Наприклад, зміст елемента Title є "рядок тексту"; зміст елемента Message є списком двох вкладених елементів Head та Body; зміст елемента Head є списком трьох вкладених елементів To, From та Subject.

"Зміст елемента" виділяється символами "дужки", тобто відкрита "(" та закрита ")" дужки. Якщо "зміст елемента" має більше, ніж один компонент, то для того, щоб вказати порядок, в якому ці компоненти з'являються, використовують "конектори" **(connectors)**. Існує лише два можливих типи "конекторів":

– "кома" **(comma)**, яка задає "послідовність" **(sequence)** компонентів; це означає логічну операцію "І" **(AND)**;

– "розділювач" **(vertical bar)**, який задає "вибір" **(alternation)**, що означає "один з двох (або декілька), але тільки не обидва (декілька) відразу" – логічна операція "Виключаюче АБО" (eXclusive OR – **XOR**).

В змісті елемента можна також вказати те, скільки разів той чи інший компонент може з'являтися (повторюватися). Для цього використовують "індикатори появ" **(occurrence indicator)**, які бувають:

– "**+**", що означає **1** або декілька появ;
– "*****", що означає **0** або декілька появ;
– "**?**", що означає **не більш ніж 1** (тобто **0 або 1**).

Компонентами "списку вкладених елементів" **(embedded elements)** можуть бути не тільки (так би мовити прості) елементи, але також "групи елементів" **(group)**. Наприклад, випадок:

```
<!ELEMENT Head (To+, From, Subject?)>
<!ELEMENT To (Title, Content)*>
<!ELEMENT From (Title, Content)*>
```

може бути переписаним з використанням "групи елементів" **(group)** наступним чином:

```
<!ELEMENT Head (((Title, Content)*)+, (Title, Content)*, Subject?)>
```

І навпаки.

Кожний елемент, в доповнення до свого змісту, може також мати певні "атрибути" **(attribute)**. Синтаксис їх опису трохи схожий на синтаксис опису елементів:

```
<!ATTLIST poem id ID #IMPLIED status (draft | revised | published) "draft" >
```

Перелік деяких (важливих) можливих типів значень атрибутів є таким:

· CDATA – будь-які допустимі символи, включаючи порожній символ та знаки пунктуації;
· NMTOKEN – тільки символи, які допускаються для позначення імен ідентифікаторів;
· ID – ідентифікатор, який є унікальним в даному контексті;
· IDREF – унікальний ідентифікатор-покажчик на деякий інший елемент;
· ENTITY – значення NMTOKEN, яке було вже продекларовано як макрос (див. нижче).

Кожне поле (кожний атрибут), крім свого типу, може також ще мати додаткову властивість:

· #REQUIRED – значення атрибуту повинно бути задано в обов'язковому порядку;
· #IMPLIED – значення атрибуту може бути не задано.

При описі елементів та їх атрибутів можуть також бути використані "макроси" (entity), які є звичайними макропідстановками ("нетерміналами" – говорячи термінами теорії компіляторів).

Отже, основними поняттями XML є:

– "елемент" (element);
– "зміст елемента" (content);
– "конектори" (connectors);
– "індикатор появлення" (occurrence indicator);
– "атрибути елемента" (attributes);
– "макропідстановки" (entity).

2. Представлення XML-документів за допомогою таблиць бази даних

Нагадаємо, що документи, які призначаються для проведення деякої розмітки над ними, тобто певного їх форматування з використанням XML техніки, є простими текстовими документами, тобто це просто текст, в якому потрібно виділити (розмітити) певну його структуру, тобто власне набір елементів та їх атрибутів. Ця структура звичайно описується в окремому файлі – Document Type Definition (DTD). Таким чином, DTD визначає *структуру даних* документа. Це означає, в свою чергу, що DTD є подібним до *структури* БД (тобто є як *порожня* БД), а "текст", який "відповідає"цьому "типу документа", відіграє роль "начинки бази даних", тобто значень, введених в цю наперед визначену структуру БД. (Останнім часом ця ідея отримала подальший розвиток і було запропоновано суттєве розширення потенційних можливостей DTD шляхом введення до розгляду нової концепції XML Schema – схема XML документа, – в якій більш ясно можна побачити зв'язок і аналогію між XML Schema (або DTD) та структурою бази даних.)

Нагадаємо також, що початкова мета введення XML техніки була направлена на забезпечення задачі обміну даними (документами) в мережі комп'ютерів, що мають різну платформу, тобто різну архітектуру hardware та різне програмне забезпечення. Одним з найбільш яскравих прикладів такої ситуації є Інтернет. Тобто головна мета XML була і є направлена на вирішення проблеми обміну інформацією (документами), а ніяк не на вирішення питання "найкращого" ("оптимального") *способу зберігання* даних. В той же час, саме таке призначення XML – "ефективний та універсальний" *засіб збереження* даних – і пропагандується рядом дослідників в різноманітних областях, які далекі від початкової ідеї розвитку XML – *розмітка* та передача *текстових* даних. Отже, і не дивно, що деякі із таких дослідників, після малоуспішних спроб пристособити XML до невласної для нього області застосування, відкривають нову для себе істину – для тих задач використання, наприклад, БД підходить краще, ніж XML, воно дає їм більше переваг.

Розглянемо далі, як XML-документи можуть бути представлені з використанням таблиць бази даних.

Отже, якщо початковий документ є "текстом", то для "тексту", без сумніву, XML є гарним засобом, оскільки текст – є "об'єкт" мови, а XML є також мовою, тобто це категорії однієї природи. В "тексті" розглядають його ієрархічну структуру – рис. 1, і ця структура чи не найкраще описується за допомогою XML. (Див. також вище приклад "Archive of Messages").

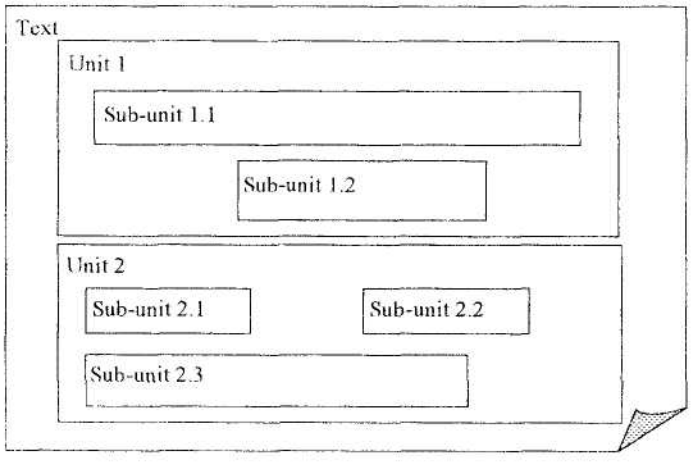


Рис. 1

В той же час, структура такого тексту може також бути представлена і в іншому вигляді: як (упорядкована) послідовність елементів, а точніше – як набір таких послідовностей – рис. 2. Для різних цілей текст може бути розбитий на набір різних блоків (units), різного типу і розміру. Таким чином, текст – це послідовність певних блоків (або певна послідовність блоків). Звернемо увагу тут на той факт, що послідовність тут означає "*упорядковану* множину блоків (units)". Відмітимо також, що ці блоки (units) є, по-суті, елементами XML-документа.

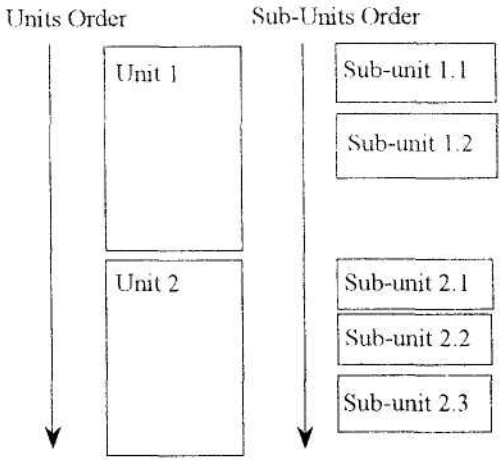


Рис. 2

Кожний блок (він же – елемент) має ряд атрибутів. Кількість цих атрибутів, як правило, залежить від конкретного XML-документа. Припустимо, що певний елемент заданого XML-документа має три типи атрибутів. Ці атрибути характеризують елемент, так би мовити, "одночасно", тобто для його характеристики потрібно вказати (перерахувати) значення всіх атрибутів одночасно (це – аналог логічної операції "І"). Значення кожного атрибуту може вибиратися з переліку можливих типів значень (див. попередній розділ).

Кожний "елемент" може бути представленим як таблиця БД. Таблиця ця має відносно просту структуру, яка, по-суті, визначається структурою "змісту елемента" (element content). Наприклад, елемент Message:

<!ELEMENT Message (Head, Body)>

має "зміст" (Head, Body). Отже, структура таблиці Message повинна мати два поля з посиланнями (foreign-keys) на таблиці Head, Body – рис. 3.

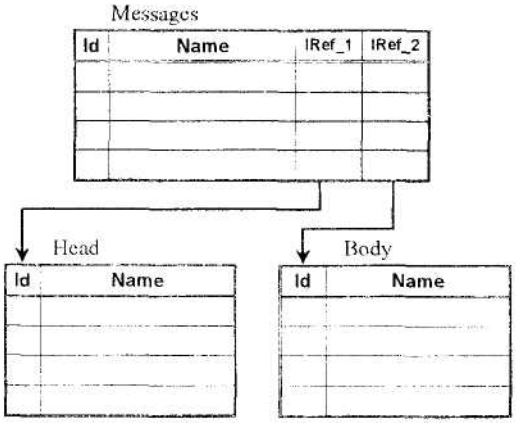


Рис. 3

Таким чином, "зміст елемента" (element content) визначає структуру відношень (*relationships*) таблиць БД.

Атрибути елемента є просто переліком додаткових полів таблиці БД. Якщо, наприклад, елемент Message має, крім свого змісту, ще й атрибути, наприклад, Type та Choice, то це означає, що таблиця Message, крім двох основних полів – посилань на таблиці Head, Body, повинна мати ще й ряд додаткових полів; будемо називати їх "секція *attribute*" – рис. 4. Типи цих полів, фактично, описані в секції *attribute* файла DTD.

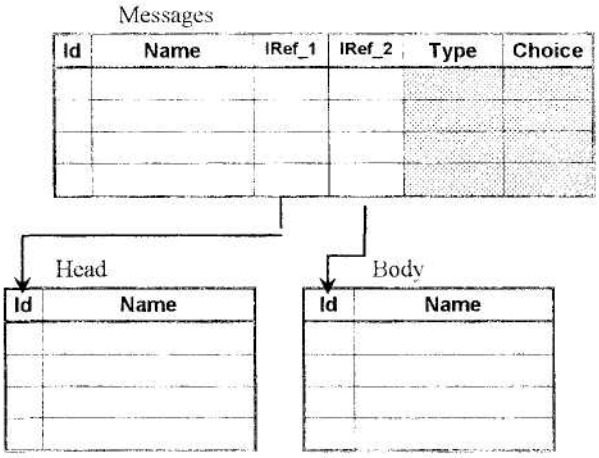


Рис. 4

Деякі з атрибутів можуть мати значення типу Choice або бути посиланнями на значення з якихось інших таблиць. У цьому випадку, по-суті, великої різниці між таким "складним атрибутом" та "елементом" немає – рис. 5.

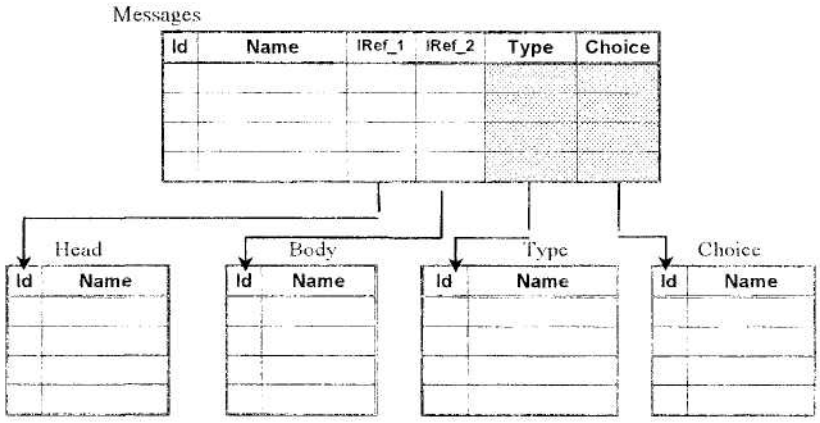


Рис. 5

Таким чином, модель даних для прикладу, що було розглянуто, може бути реалізована як за допомогою XML (назвемо його "варіант № 1"), так і з використанням реляційної бази даних ("варіант № 2").

Очевидно, що 1-й варіант має більші переваги при застосуваннях в таких, наприклад, задачах, як анотування лінгвістичних даних [3]. У цьому випадку блоками будуть фрагменти тексту, і різні блоки (різні фрагменти тексту) можуть мати різну довжину. Тоді зберігання таких блоків у реляційній базі даних може бути не ефективним.

Крім того, використання XML зручне, якщо в задачі приходиться працювати з різними типами документів, структура яких може суттєво варіюватися. У такому випадку, без сумніву, зручно мати такий гнучкий інструмент, як мова (XML) для опису таких розрізнених структур. "Увігнати" такі документи з суттєво різною структурою в жорсткі рамки наперед визначеної структури БД може бути досить складною задачею. Та це і не потрібно – краще використовувати XML.

Якщо ж структури різних типів документів є суттєво схожими (а в ідеалі – однаковими), то може бути зручним і ефективним використання БД для зберігання таких документів. Особливо ефективним буде така реалізація, якщо потрібно виконувати суттєву обробку змісту документа – виконувати певні запити (query) до БД, застосовуючи добре розроблену та таку, що себе зарекомендувала, технологію SQL для виконання запитів до БД і проведення різного роду екстракції інформації (information extraction).

Варіант № 2 є також більш придатним, коли довжина блоків не дуже сильно варіюється. Так, у випадку анотування розмовних (speech) або відіо-даних [3] файл, що анотується, є,

фактично, набором значень сегментів часу (time-stamps), тобто "блоками" (units) є сегменти часу, і вони ефективно можуть бути представленими (та збереженими) як таблиці БД.

Теоретично на основі опису DTD користувач може самостійно (безпосередньо) створити набір таблиць у середовищі певної СУБД та встановити необхідні відношення (*relationships*) між ними. Причому, користувач може проробити це у СУБД у "візуальному" режимі, що є аж ніяк не гіршим за створення такої ж самої структури документа за допомогою написання файлу DTD "вспіну". Було б непогано також, якби користувач міг би мати можливість автоматичного створення таблиці БД та встановлення їх relationships на основі DTD опису. Останнім часом для того, щоб полегшити процес написання DTD, було створено ряд спеціальних програм, таких, наприклад, як XML Spy, XML Notepad тощо, які дозволяють редагувати DTD та інші XML- файли у візуальному режимі, полегшуючи тим самим досить трудоємку і кропітку роботу написання DTD вручну.

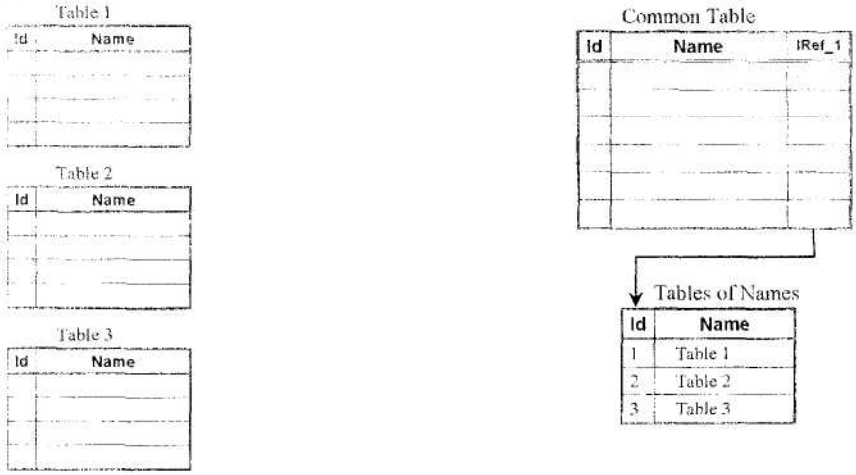
Як бачимо, DTD опис задає, серед іншого, опис *ієрархічної* структури документа. Так, в попередньому прикладі

<!ELEMENT Message (Head, Body)>

ми маємо дворівневу ієрархію – рис. 3.

Далі для випадку більш складної ієрархічної структури пропонується дещо спростити її представлення за допомогою таблиць БД, зробивши його єдиноманітним. А саме:

– таблиці одного рівня та однакової структури можна зберігати як одну таблицю з *додатковим* полем, в якому вказується номер "початкових" таблиць. По-суті, це має бути покажчик на значення в іншій таблиці – довіднику з переліком назв "початкових" таблиць – рис. 6:



a) *b)*

Рис. 6

– оскільки кожна таблиця може мати різну структуру, що визначається переліком додаткових атрибутів – рис. 7:

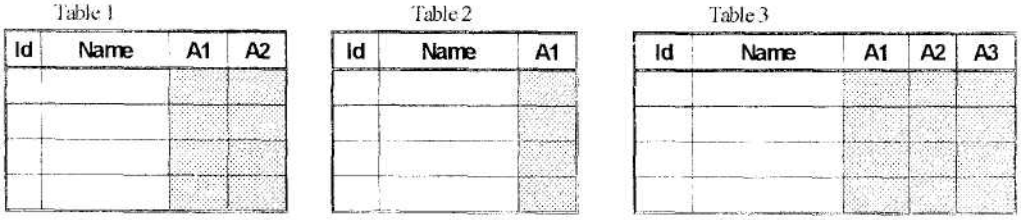


Рис. 7

то таку ситуацію також пропонується уніфікувати наступним чином:

– в подальшому всі таблиці одного рівня ієрархії повинні мати однакову структуру – рис. 8:

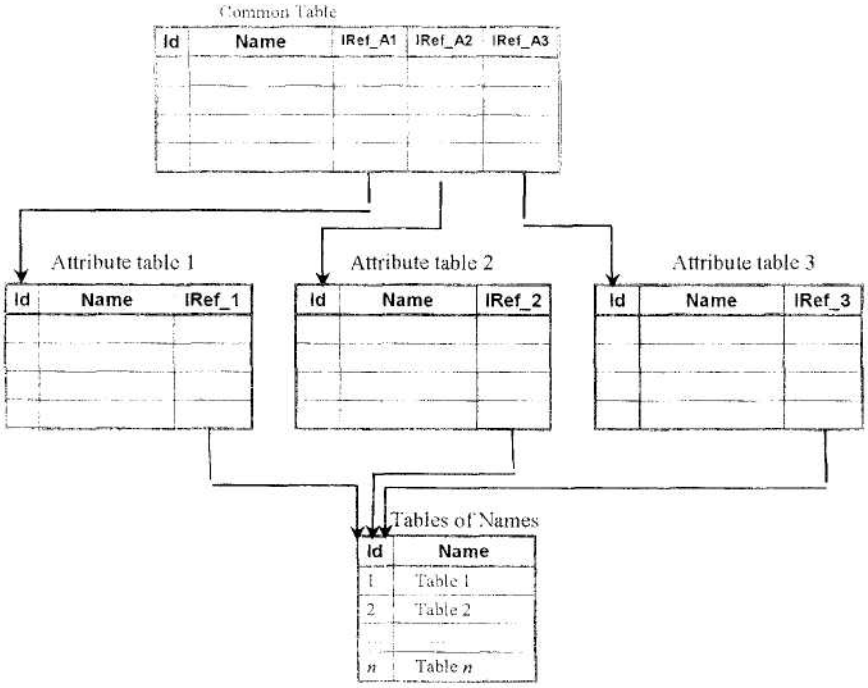


Рис. 8

- замість реальних значень атрибутів у цих таблицях будуть знаходитися покажчики на таблиці A_1, A_2, A_3, де зберігаються реальні значення атрибутів;
  (на перший погляд це може здатися надлишковим і непотрібним – це таки і є надлишковим, але така надлишковість забезпечує гнучкість та одноманітний підхід);
- реальні значення атрибутів зберігаються в окремій таблиці і є лише по одній таблиці для кожного типу даних-атрибутів: одна – для текстового типу (таблиця A_1), одна – для числового (таблиця A_2) та одна – для ще якого-небудь іншого, наприклад, data/time чи Yes/No типу (таблиця A_3);
- крім того, всі таблиці значень-атрибутів мають ще по одному додатковому полю – покажчик на таблицю Group/Cluster, за допомогою якого можна визначати групи значень атрибута (якщо це буде потрібно);
- якщо деяка таблиця Level_X не має значень атрибуту певного типу, то в цій таблиці просто не буде вказуватися покажчик на таблицю значень-атрибутів, тобто поле IRef_A*i* буде пустим;
- запропонована структура дозволяє також уникнути дублювання при зберіганні даних, якщо значення деякого атрибуту є однаковими (в такому випадку це значення буде зберігатися лише в одному екземплярі в таблиці значень-атрибутів, а в таблиці Level_X буде лише дублюватися відповідний покажчик).

Такий самий підхід дозволяє подібним чином вирішити проблему, коли різні таблиці Level_X мали атрибути, які були значеннями Choice з певного наперед визначеного набору ($val_1 \setminus/ val_2 \setminus/ \ldots \setminus/ val_n$). При звичайному підході всі такі набори Choice потрібно було реалізовувати як окремі Ref_Table. В той же час часто всі такі Ref_Table мають однакову структуру (лише поля Id та Name). Тому всі вони можуть бути об'єднані в одну таблицю – fig. 8, в якій в полі Id_Ref_Group буде вказано на те, до якої Ref_Table належить та чи інша група значень. Таблиця ж Ref_Group_Table буде містити назви таких груп. Тобто це – точно така ж сама ідея, що й та, яка була наведена вище для випадку атрибутів-значень.

Заключення

Таким чином, проведені дослідження підтверджують можливість представлення і реалізації ієрархічної моделі даних XML-документа за допомогою таблиць реляційної бази даних.

Вибір того чи іншого підходу в кінці кінців має бути визначеним на основі аналізу характеру задачі, що вирішується, а не суб'єктивними смаками розробника, його, так би мовити, позитивним чи негативним відношенням до тієї чи іншої технології.

Автор висловлює подяку проф. Niels Ole Bernsen, університету м. Одензе, Данія, за обговорення ряду питань, що розглянуті у статті. Дана робота частково виконувалася в рамках проекту NITE (Natural Interactivity Tools Engineering), EU/HLT – IST-2000-26095.

ЛІТЕРАТУРА:

1. http://www.w3.org/XML/
2. *Elliotte Rusty Harold.* XML™ Bible. – IDG Books Worldwide, Inc., 1999. – 1022 p.
3. *Carletta, J., Bernsen, N O., Cadüe N., Dybkjжr, L., Evert S., Heid, U., Isard A., Kolodnytsky M., Lauer C., Lezius W., Noldus L., Reithinger N, and Vugele A.* Software Specification and Work Plan. NITE Deliverable D1.1, 2001. – 37 p.
4. *Larry S. Hayashi, John Hatton.* Combining UML, XML and relational database technologies – the best of all worlds for robust linguistic databases // Proceedings of the IRCS Workshop on Linguistic Databases. – 11–13 December, 2001. University of Pennsylvania, Philadelphia, USA. – 115–124 pp.
5. *Villegas M., Bel N.* From DTD to relational dB. An automatic generation of a lexicographical station out off ISLE guidelines // Proceedings of the LREC 2002 Third International Conference On Language Resources And Evaluation. Las Palmas, Canary Islands – Spain, 29–31 May, 2002. – 694–700 pp.
6. *Tatarinov I., Viglas Stratis D., Beyer K., Shanmugasundaram J., Shekita E.* Zhang Storing and Querying Ordered XML Using a Relational Database System // Proceedings of the ACM SIGMOD Conference 2002: Madison, Wisconsin, USA. – 204–215 pp.
7. *Champion M.* Storing XML in Databases // eAI Journal, October, 2001. – 53–55 pp.
8. *Dayen I.* Storing XML in Relational Databases // http://www.xml.com/pub/a/2001/06/20/databases.html
9. http://www.tei-c.org/P4X/SG.html
10. *В. Вейтман.* Программирование для Web // Диалектика, 2000. – 368 с.

КОЛОДНИЦЬКИЙ Микола Михайлович – кандидат технічних наук, професор кафедри КІТ Житомирського інженерно-технологічного інституту.

Наукові інтереси:

- математичне моделювання систем;
- комп'ютерні інформаційні технології.

Подано 18.09.2002