Dokumenty XML w relacyjnych bazach danych – czyli wojna światów

Andrzej Ptasznik
Warszawska Wyższa Szkoła Informatyki
aptaszni@wwsi.edu.pl

Streszczenie

Przedmiotem wykładu jest wykorzystanie dokumentów XML w relacyjnych bazach danych. W pierwszej części wykładu opowiedziana zostanie krótka historia standardu XML i podstawowe zasady tworzenia dokumentów XML. Następnie omówione zostaną sposoby przekształcania danych relacyjnych do postaci XML oraz zasady zapytań pobierających dane z dokumentu XML. Prezentowane będą też przykłady wykorzystania typu danych XML na etapie projektowania baz danych, a także przykłady zastosowania XML w rozwiązywaniu konkretnych problemów. Przedstawione zostaną również elementy schematów XSD i ich znaczenie w procesie zapewnienia poprawności danych zapisanych w dokumentach XML.

Spis treści

1 WPROWADZENIE

Relacyjne bazy danych towarzyszą twórcom aplikacji już od wielu lat i ciężko znaleźć projektanta lub programistę, który nie zetknął się z tą problematyką. Podobna sytuacja występuje w obszarze związanym z językiem XML. Jego intensywny rozwój, a właściwie dynamiczne rozszerzanie zastosowań oraz rozbudowywanie technologii „z otoczki" XML możemy obserwować od ponad dziesięciu lat. W takiej sytuacji jest dość oczywiste, że język ten pojawił się w otoczeniu relacyjnych baz danych i rozpoczęła się ich „współpraca". Przejawami tej współpracy i wzajemnego przenikania się obu technologii zajmiemy się na niniejszym wykładzie.

W ramach wykładu zostanie zaprezentowana geneza oraz podstawy języka XML, co ułatwi lepsze poruszanie się po omawianej problematyce. Zasadniczą częścią wykładu będzie jednak związek języka XML z systemem SQL Server 2008 i wsparcie oferowane przez to narzędzie w zakresie obsługi danych w formacie XML. Zaprezentowane zostaną polecenia służące do zwracania rezultatów zapytań w postaci dokumentów XML, przedstawione będą możliwości typu danych XML, który służy do przechowywania dokumentów XML w bazie w postaci natywnej, a także zostaną opisane przykłady stosowania XML Schema do definiowania dopuszczalnej struktury dokumentów XML.

2 JĘZYK XML

Wraz z intensywnym rozwojem Internetu pojawił się (a właściwie nabrał większego znaczenia) problem przekazywania danych pomiędzy różnymi systemami. Problem ten wystąpił w momencie pierwszych prób łączenia komputerów w sieć. Od samego początku istniały dwa podejścia: korzystanie z binarnego zapisu danych oraz z postaci tekstowej. W okresie, gdy komputery miały bardzo ograniczone zasoby (pamięć operacyjną, pojemność dysków, transfer w sieci) popularniejsze były rozwiązania korzystające z postaci binarnej. Istniejące protokoły binarne powodowały jednak konieczność „tłumaczenia" danych pomiędzy heterogenicznymi systemami (np. w jednym systemie liczby całkowite są cztero-, a w drugim – dwubajtowe), co powodowało wzrost poziomu komplikacji systemów rozproszonych. Równocześnie obserwowano burzliwy rozwój usługi WWW, dla której pokonanie bariery pomiędzy różnymi architekturami nie stanowiło problemu, gdyż stosowanym rozwiązaniem było przekazywanie danych w postaci tekstowej, czego przejawem stał się język HTML. Jego sukces i rosnąca popularność ujawniły jednak wiele niedoskonałości. Wystarczy wspomnieć o tzw. wojnie przeglądarek, polegającej z grubsza na różnym sposobie interpretowania (czyli renderowania) dokumentu HTML przez różne przeglądarki oraz na dodawaniu przez producentów przeglądarek nowych elementów (spoza specyfikacji HTML), które były interpretowane przez ich produkt. W efekcie idea HTML – jako języka służącego do przekazywania treści wraz z informacjami nadającymi znaczenie poszczególnym fragmentom tekstu – została wypaczona, a sam język sprowadzono do poziomu języka opisu układu (ang. *layout*) dokumentu.

Przez kilka lat taka sytuacja była obowiązującym standardem, a twórcy stron WWW używali w dokumentach HTML wielu karkołomnych sztuczek dla osiągania konkretnych efektów wizualnych. Koniecznością było w takiej sytuacji tworzenie kilku wersji strony WWW, które działały poprawnie w konkretnych przeglądarkach. Jednocześnie rosły możliwości komputerów (graficzne, obliczeniowe) oraz wymagania stawiane stronom WWW. Prowadziło to do pogłębiania się chaosu w świecie technologii internetowych.

I tak w 1996 roku powstał pomysł stworzenia nowego języka dla potrzeb Internetu. Miał to być prosty język, z prostymi regułami dotyczącymi jego składni, czytelny zarówno dla człowieka, jak i dla maszyn, oraz możliwie uniwersalny jako nośnik danych i informacji o ich strukturze. W efekcie tych prac powstał **język XML** (ang. *Extensible Markup Language*). Wizualnie zbliżony do HTML (stosowanie znaczników i elementów atrybutów), choć od samego początku rygorystycznie podchodzący do poprawnego formułowania dokumentu. Mimo że XML jest tylko podzbiorem istniejącego od dawna **języka SGML** (ang. *Standarized General Markup Language*) o potężnych możliwościach, szybko okazało się, że z racji swojej prostoty, XML wyparł SGML z większości zastosowań.

Wraz ze wzrostem popularności języka XML pojawiły się narzędzia służące do jego „obróbki". Dziś trudno znaleźć jakikolwiek język programowania, w którym nie byłoby bibliotek obsługujących XML (w zakresie odczytu, tworzenia i weryfikacji poprawności). Właśnie łatwość sprawdzenia poprawności syntaktycznej i semantycznej dokumentów XML stanowi jeden z głównych argumentów przemawiających za jego stosowaniem. Po co tworzyć dla każdego systemu osobne formaty plików z danymi oraz mechanizmy sprawdzające ich poprawność, skoro można użyć w każdym przypadku jednego i tego samego narzędzia?

Reguły określone dla struktury dokumentu XML są proste i przejrzyste, i nie zawierają wyjątków. Aby dokument XML mógł być uznany za **poprawnie sformułowany** (tzn. zgodny z wszystkimi wymaganiami co do syntaktyki) musi spełniać następujące reguły:

- składać się z elementów, przy czym jeden z elementów jest elementem głównym i zawiera w sobie pozostałe;
- element składa się (podobnie jak w HTML) ze znaczników: otwierającego i zamykającego. Język HTML dopuszczał brak znacznika zamykającego w miejscach, w których można się było domyślić, że element powinien być zamknięty (np. przy definiowaniu komórek w tabeli znacznik ‹td› nie musiał być zamykany, bo wiadomo, że gdy po nim pojawiał się kolejny ‹td› to oznaczał początek kolejnej komórki tabeli). W dokumencie XML każdy element musi być zamknięty. Jeżeli element nie ma zawartości to można zastosować formę skróconą (zamiast znacznika zamykającego umieszcza się po nazwie znacznika otwierającego znak /. Na przykład: `<br />`);
- elementy muszą być poprawnie zagnieżdżone – zakres objęty elementem musi być jednoznacznie określony. Innymi słowy: jeden element może zawierać w sobie inny, ale tylko w całości (wraz ze znacznikiem zamykającym);
- każdy element może zawierać atrybuty. Są one definiowane wewnątrz znacznika otwierającego po nazwie elementu. Każdy atrybut musi mieć wartość – w przeciwieństwie do HTML, gdzie takiej konieczności nie było. Wartość atrybutu musi być ujęta w apostrofy lub cudzysłów;
- w dokumentach XML jest rozróżniana wielkość liter i jest ona istotna. Elementy `<dane />`, `<Dane />` i `<DANE />` to trzy różne elementy;
- podobnie jak w dokumencie HTML, jeżeli w treści dokumentu chcemy umieścić znak, który mógłby powodować niejasności w interpretacji dokumentu – zastępujemy go odpowiednią **encją**. Na przykład, w przypadku:

```
<wyrazenie>Ala<Ela</wyrazenie>
```

Znak ‹ pomiędzy słowami Ela i Ala mógłby zostać błędnie zinterpretowany jako początek elementu o nazwie Ala. Po zastąpieniu go encją otrzymujemy:

```
<wyrazenie>Ala&lt;Ela</wyrazenie>
```

Co jest już poprawnie interpretowane.

Poprawność dokumentów XML można rozpatrywać na dwóch poziomach: syntaktycznym i semantycznym. Pierwszy z nich jest określony przez opisane wyżej reguły dotyczące tworzenia dokumentów XML. Jeżeli dokument pod względem składniowym spełnia te reguły, to jest **dokumentem poprawnie sformułowanym** (ang. *well formed*).

Jeżeli ten poziom weryfikacji nie wystarcza, można sięgnąć po dodatkowe narzędzia, aby nałożyć ograniczenia semantyczne (np. w każdym elemencie „osoba" musi wystąpić co najmniej jeden element „imie", wartością atrybutu „pesel" musi być ciąg 11 cyfr). Najczęściej stosowane są rozwiązania: **DTD** (ang. *Document Type Definition*) i XML Schema.

DTD jest rozwiązaniem starszym i o ograniczonych możliwościach. Obecnie znacznie częściej stosuje się **XML Schema**, co nie znaczy że zapewnia ono możliwość zdefiniowania dowolnych reguł dotyczących dopuszczalnej zawartości dokumentu XML. Jako alternatywa dla XML Schema wymieniane jest też Relax NG. Jest to rozwiązanie porównywalne z XML Schema, o nieco innej specyfice i możliwościach. Zależnie od reguł, które chcemy wymusić, w dokumencie można stosować konkretne rozwiązania. Najistotniejsze jest jednak to, że niezależnie od platformy sprzętowej i programowej, raz określone reguły mogą być weryfikowane w taki sam sposób za pomocą narzędzi dostępnych dla konkretnej platformy.

Jeżeli dokument jest poprawnie sformułowany oraz spełnia reguły opisane w DTD lub XML Schema, to jest **dokumentem poprawnym** (ang. *valid*). Istnieje wiele narzędzi służących do sprawdzania poprawności dokumentów XML, są to tzw. **parsery walidujące**. Połączenie XML z XML Schema, przy wsparciu ze strony narzędzi, umożliwia bardzo łatwe rozwiązywanie problemów z projektowaniem formatów przekazywania czy przechowywania danych. Wystarczy opracować dla konkretnego problemu dokument XML Schema i przekazać zainteresowanym stronom. Twórcy dokumentów XML będą mieli wszystkie informacje, niezbędne do utworzenia dokumentów w postaci zgodnej z założeniami. Z kolei, twórcy oprogramowania, które ma takie dane interpretować, będą wiedzieli czego mogą spodziewać się w dokumencie oraz będą mieli możliwość łatwego sprawdzenia poprawności dokumentu przed rozpoczęciem jego przetwarzania. Taki schemat sprzyja rozkwitowi technologii związanych ze stosowaniem XML. Można tu wymienić dla przykładu **język SVG** (ang. *Scalable Vector Graphics)* czy **MathML**. Oba definiują postać dokumentu XML, który następnie jest przetwarzany na grafikę wektorową lub równanie matematyczne.

Podsumowując kwestię weryfikacji dokumentów XML: dla dowolnego problemu, dla którego specyfikuje się format dokumentu XML, można za pomocą standardowych narzędzi zdefiniować dodatkowe reguły semantyczne i weryfikować automatycznie ich spełnienie. Dzięki temu można ujednolicić wstęp do procesu przetwarzania danych z dokumentu XML do postaci sprawdzenia poprawności sformułowania i poprawności dokumentu. Dalsze przetwarzanie odbywa się już przy założeniu poprawności dokumentu, co upraszcza ten proces ze względu na gwarancję co to struktury i zawartości przetwarzanego dokumentu.

3 JĘZYK XML ALTERNATYWĄ DLA RELACYJNYCH BAZ DANYCH

Skoro język XML umożliwia łatwe budowanie dokumentów o hierarchicznej strukturze, to czy może stanowić alternatywę dla relacyjnych baz danych? Można przecież zastąpić relacje odpowiednim zagnieżdżaniem elementów. Teoretycznie jest to jak najbardziej możliwe, ale praktycznie raczej nie.

W dokumencie XML, co prawda, można zastąpić relacje odpowiednim zagnieżdżaniem elementów oraz stosowaniem atrybutów, ale to podejście sprawdza się wyłącznie przy małej liczbie danych. Przy niewielkim rozmiarze dokumentu może on śmiało rywalizować z bazą danych, lecz wraz ze wzrostem rozmiaru dokumentu, XML szybko zostaje w tyle i osiąga „masę krytyczną", co powoduje ogromny spadek wydajności przy jego przetwarzaniu. Podobnie wygląda definiowanie ograniczeń stosowanych do zawartości i struktury dokumentu XML. Narzędzie XML Schema, mimo rozbudowanych możliwości, okazuje się jednak niewystarczające i powstaje konieczność dobudowywania własnych mechanizmów służących zapewnieniu spójności danych. To wszystko powoduje, że stosowanie języka XML w charakterze bazy danych staje się coraz bardziej złożone, co szybko przerasta stopień złożoności prawdziwej bazy relacyjnej zastosowanej do rozwiązania tego samego problemu, a co za tym idzie stosowanie XML w tym przypadku staje się ekonomicznie nieuzasadnione.

Obszarem, w którym XML sprawdza się jednak bardzo dobrze w roli bazy danych jest pełnienie funkcji magazynu danych *off-line*. Polega to na tworzeniu aplikacji, które łącząc się z bazą danych, pobierają dane niezbędne do pracy aplikacji i przechowują je na komputerze użytkownika w postaci XML. Dalsza praca odbywa się na danych z XML już bez połączenia z bazą danych i dopiero w momencie synchronizacji dane są uaktualniane w bazie. Przykładem tego typu rozwiązania jest klasa DataSet z .NET Framework. Ma ona bardzo rozbudowane możliwości przechowywania danych i śledzenia ich modyfikacji.

Zasadniczo jednak język XML należy raczej rozpatrywać jako pewnego rodzaju uzupełnienie funkcjonalności baz danych, które może być stosowane w przypadkach wymagających tworzenia rozbudowanej struktury tabel, aby zamodelować zbiór cech informacyjnych o zróżnicowanej strukturze. Tu XML sprawdza się znakomicie, co potwierdza coraz większe wsparcie dla korzystania z XML w relacyjnych bazach danych oferowane przez poszczególnych producentów. W ramach niniejszego wykładu prezentowane będą takie właśnie mechanizmy w odniesieniu do systemu SQL Server 2008.

4 JĘZYK XML W MICROSOFT SQL SERVER 2008

Klauzula FOR XML polecenia SELECT

Pierwszym zagadnieniem dotyczącym zastosowania języka XML w bazach danych było zwracanie wyników zapytań w formie dokumentów XML. Jest to wygodny mechanizm, szczególnie gdy aplikacja korzysta z danych o bardziej złożonej strukturze. Zbudowanie dokumentu XML umożliwia stworzenie struktury adekwatnej do wymagań i pozwala uniknąć np. wielokrotnego komunikowania się z bazą danych w celu pobierania różnych fragmentów potrzebnych danych. Baza w odpowiedzi na jedno zapytanie zwraca odpowiedni dokument XML, a aplikacja zajmuje się jego przetworzeniem, co zwykle odbywa się z wykorzystaniem gotowych narzędzi i rozwiązań (np. jako transformacja dokumentu XML do postaci strony HTML lub do dokumentu PDF czy XLS).

SQL Server począwszy od wersji 2000 dopuszcza zwracanie rezultatu wykonania polecenia SELECT w postaci fragmentu lub kompletnego dokumentu XML. Żeby skorzystać z tej możliwości należy dobrać odpowiedni wariant klauzuli FOR XML. Występuje ona w czterech wariantach:

- RAW
- AUTO
- EXPLICIT
- PATH

W prostszych przypadkach mogą to być tryby RAW albo AUTO. Służą one do tworzenia prostej reprezentacji zbioru wynikowego jako fragmentu dokumentu XML z opcją ujęcia go w zdefiniowany element główny (ROOT). W przypadku istnienia wymagań generowania bardziej złożonej struktury dokumentu XML, mamy do dyspozycji klauzulę FOR XML w wariantach EXPLICIT oraz PATH. Wariant PATH, podobnie jak RAW czy AUTO, można zastosować do większości zapytań, gdyż bardzo łatwo dostosować zapytanie do postaci wymaganej przez ten wariant klauzuli FOR XML. Inaczej wygląda sytuacja w przypadku wariantu EXPLICIT, który ma ściśle określone wymagania co do postaci zbioru wynikowego, który ma być przekształcony na postać XML, tzw. **tablica uniwersalna**. Taka tablica jest zwykle generowana za pomocą wielu zapytań łączonych klauzulą UNION. Poszczególne zapytania zwracają wartości różnych kolumn zbiorczego wyniku zapytania.

Najprostszym do zastosowania jest tryb RAW. Jego działanie sprowadza się do wygenerowania dla każdego wiersza ze zbioru wynikowego jednego elementu XML o domyślnej nazwie ROW. Wartości poszczególnych kolumn dla wiersza stają się wartościami atrybutów elementu ROW. Można także określić własną nazwę dla elementu row oraz zamiast atrybutów wybrać generowanie wartości z kolumn jako elementów XML o takiej nazwie jak nazwa kolumny. W ramach trybu RAW można stosować dodatkowe opcje:

- Opcja ROOT powoduje dodanie do rezultatu zapytania elementu głównego o domyślnej nazwie ROOT bądź dowolnej innej określonej w ramach opcji.
- Korzystanie z opcji ROOT jest powszechną praktyką, gdyż tylko wtedy zwrócony dokument XML spełnia reguły składni i jest poprawnie sformułowany, co umożliwia jego dalsze przetwarzanie.
- Opcja ELEMENTS powoduje, że zamiast atrybutów, do umieszczenia zawartości kolumn są wykorzystane elementy o nazwach takich, jak poszczególne kolumny.

W tym momencie nasuwa się pytanie: Czy stosować elementy czy atrybuty? Jest ono jednym z częściej zadawanych pytań dotyczących planowania struktury dokumentów. Niestety nie ma na nie jednoznacznej odpowiedzi, warto natomiast zdawać sobie sprawę, że korzystanie z atrybutów:

- zmniejsza rozmiar wynikowego dokumentu;
- uniemożliwia nadawanie struktury zawartości atrybutu;
- ogranicza liczebność wystąpień atrybutów o tej samej nazwie w ramach elementu do jednego.

Korzystanie z elementów natomiast:

- zwiększa rozmiar wynikowego dokumentu;
- umożliwia w razie potrzeby nadawanie struktury wartościom elementu.

Dla przykładu weźmy dwa elementy:

```
<dane osoba='Jan Nowak'/>
```

i

```
<dane><osoba>Jan Nowak</osoba></dane>
```

Widać, że pierwszy wariant jest bardziej zwięzły. Nie można natomiast wykonać w nim dodania drugiej „osoby" do elementu `dane`, chyba że w karkołomny sposób:

```
<dane osoba='Jan Nowak'  osoba2='Tomasz Kowalski' />
```

W drugim wariancie nie ma takiego problemu:

```
<dane><osoba>Jan Nowak</osoba><osoba>Tomasz Kowalski
</osoba></</dane>
```

Podobnie, gdy chcemy nadać zawartości atrybutu czy elementu `osoba` jakąś strukturę, to pierwszy wariant skutecznie to uniemożliwia.

Drugi wariant umożliwia natomiast swobodne wykonanie:

```
<dane>
   <osoba>
     <imie>Jan</imie>
     <nazwisko>Nowak</nazwisko>
  </osoba>
</dane>
```

Kolejnym trybem dostępnym w ramach klauzuli FOR XML jest tryb AUTO. Ma on możliwości zbliżone do RAW z tą różnicą, że potrafi budować proste hierarchie w dokumencie XML. Proces budowania hierarchii jest oparty na **heurystykach**. Analizowane są kolejne wiersze i wartości kolumn. Umożliwia to przy odpowiednim skonstruowaniu zapytania (sortowanie, kolejność złączeń) na sterowanie postacią wyjściowej hierarchii dokumentu XML. Umożliwia także (podobnie jak RAW) osadzenie w wyjściowym dokumencie XML wygenerowanego dla niego dokumentu XML Schema, opisującego postać wyjściowego dokumentu XML.

W przypadku pojawienia się w wyniku zapytania pól binarnych, są one domyślnie kodowane metodą URL Encode. Jeżeli nie jest to odpowiednie rozwiązanie, można skorzystać z opcji BINARY BASE64.

Klauzula FOR XML w wariancie EXPLICIT ma największe możliwości, ale jest też najbardziej złożona i trudna w stosowaniu. Nie nadaje się w przeciwieństwie do RAW i AUTO do zastosowania w dowolnym zapytaniu. Żeby skorzystać z trybu EXPLICIT, tabela wejściowa musi być tzw. **tabelą uniwersalną**. Składa się ona z kolumn o ustalonej kolejności, nazwach i znaczeniu:

Tabela 1.
Przykład tabeli uniwersalnej

Tag	Parent	Customer!1!cid	Customer!1!name	Order!2!id	Order!2!date	OrderDetail!3!id!id	OrderDetail!3!pid!idref
1	NULL	C1	"Janine"	NULL	NULL	NULL	NULL
2	1	C1	NULL	01	1/20/1996	NULL	NULL
3	2	C1	NULL	01	NULL	OD1	P1
3	2	C1	NULL	01	NULL	OD2	P2
2	1	C1	NULL	02	3/29/1997	NULL	NULL

Przykładowo (patrz tab. 1) pierwszą kolumną musi być kolumna `Tag`, która zawiera unikatowy numer dla każdego elementu, który będzie generowany. Druga kolumna ma nazwę `Parent` i określa wartość Tag dla rodzica elementu. Kolejne kolumny definiują składowe elementów XML. Nazwy tych kolumn są tworzone w opar-

ciu o schemat *nazwaElementu!Tag!NazwaAtrybutu!Dyrektywa*. Umożliwia to dokładne sterowanie procesem definiowania struktury elementów i atrybutami. Dokładne omówienie mechanizmu działania trybu EXPLICIT wykracza poza zakres niniejszego opracowania. Klauzula FOR XML EXPLICIT pomimo dużych możliwości definiowania struktury wynikowego dokumentu XML jest obarczona wadami związanymi z trudnościami w przygotowaniu odpowiedniego zapytania wejściowego. Szczególnie nieprzyjemne jest modyfikowanie już istniejącej struktury. Jest to proces żmudny i podatny na błędy, często skutkujący pisaniem zapytania od nowa.

Wraz z SQL Server 2005 pojawił się tryb PATH. Jest on rozsądnym kompromisem pomiędzy możliwościami w zakresie definiowania struktury XML, a łatwością zapisu tych reguł. W uproszczeniu, tryb PATH bazuje na nazwach nadawanych kolumnom zapytania. Mają one postać zbliżoną do wyrażeń języka XPath (znanego każdemu, kto uważa się za eksperta od XML) wskazujących elementy bądź atrybuty. Na ich podstawie budowana jest wyjściowa struktura XML. Można ją dodatkowo komplikować poprzez zagnieżdżanie zapytań. Pamiętać należy jedynie o odpowiedniej kolejności występowania kolumn w zapytaniu – te definiujące atrybuty elementu muszą wystąpić przed definiującymi kolejne lub zagnieżdżone elementy.

Jeżeli kolumna w wyniku zapytania ma być elementem, to domyślną jego nazwą będzie nazwa kolumny. Jeżeli ma być atrybutem – należy nadać kolumnie alias zaczynający się od znaku @ (małpki). Alias może zawierać także znaki ukośnika, które określają kolejne poziomy zagnieżdżenia. Z kolei znak * (gwiazdka) powoduje, że w przypadku kolumny typu XML, jej zawartość będzie osadzona w wyniku zapytania wprost. Dla kolumn innych typów wstawiony będzie węzeł tekstowy z zawartością.

Wszystkie opisane warianty klauzuli FOR XML mają jeszcze kilka opcji, z którymi warto się zapoznać. Jako przykład można podać opcję TYPE powodującą, że zwrócona wartość jest traktowana jak zmienna typu XML, co umożliwia wygodne dalsze jej przetwarzanie. Równie ciekawa jest opcja XSNIL, która umożliwia umieszczenie w wynikowym dokumencie XML elementów, które mają w wejściowym zapytaniu wartość null i domyślnie nie byłyby umieszczone w wyniku. Jest to szczególnie istotne przy przetwarzaniu zwróconych danych w aplikacji, która niekoniecznie ma skąd wziąć pełną listę dopuszczalnych elementów, które można w ramach przetwarzania uzupełnić. Oczywiście można próbować ten problem rozwiązać za pomocą osadzania XML Schema, ale jest to bardziej złożone i pracochłonne.

Opisane pokrótce możliwości SQL Server 2008 w zakresie zwracania wyników zapytań w postaci XML nie wyczerpuje wszystkich możliwości istniejących w tym narzędziu, sygnalizują jedynie podstawowe mechanizmy i ich zastosowanie.

Typ danych XML

Kolejnym obszarem zastosowania XML w SQL Server 2008 jest typ danych XML. Służy on do przechowywania dokumentów lub fragmentów dokumentów XML bezpośrednio w bazie danych oraz do wygodnego manipulowania nimi i walidowania z zastosowaniem XML Schema. Dane XML w bazie mogą występować w dwóch wariantach:

- skojarzone z kolekcją dokumentów XML Schema (ang. *typed XML*),
- nieskojarzone z XML Schema (ang. *untyped XML*).

Skojarzenie kolumny typu XML ze schemą powoduje nadanie ograniczeń strukturze dokumentów XML, które mogą być umieszczone w tej kolumnie. Ograniczenia te są weryfikowane automatycznie przy każdej operacji dodania czy modyfikacji zawartości kolumny XML.

Zanim skorzysta się z typu danych XML warto zapoznać się z jego dokumentacją. Szczególnie chodzi tu o sposób przechowywania dokumentu oraz o ograniczenia związane z samym typem danych. Warto również zastanowić się nad korzystaniem z kolekcji XML Schema oraz indeksów XML. Istotną informacją jest to, że dokument nie jest zapisywany w bazie wprost, tylko przechodzi proces normalizacji (modyfikacja dokumentu, kodowanie w Unicode, eliminowanie niepotrzebnych ciągów i znaków itp.). Eliminuje to możliwość korzystania z tego typu danych wszędzie tam, gdzie ważna jest oryginalna postać dokumentu (np. kwestie podpisu elektronicznego).

Deklarowanie kolumn typu XML nie odbiega od deklarowania kolumn każdego innego typu. Jedyną specyficzną rzeczą jest – w przypadku kolumny ze skojarzoną kolekcją dokumentów XML Schema – umieszczenie w nawiasie w deklaracji typu nazwy tej kolekcji.

Sama kolekcja dokumentów XML Schema zawierać może jedną bądź wiele pojedynczych schem, które zapisuje się jedna pod drugą. Po skojarzeniu kolekcji z kolumną typu XML, każda wartość wpisywana do tej kolumny będzie walidowana pod kątem zgodności z którąś ze schem z kolekcji. W przypadku braku zgodności – operacja zapisu zostanie anulowana.

Tworzenie dokumentów XML Schema (zwane też modelowaniem dopuszczalnej struktury dokumentów XML) jest zagadnieniem bardzo rozbudowanym i wykracza poza ramy niniejszego wykładu. Przy założeniu, że mamy już określoną postać dokumentu XML Schema, stworzenie kolekcji schem jest proste i ogranicza się do wykonania jednego polecenia. Od tego momentu można używać zdefiniowanej w ten sposób kolekcji przy deklaracjach kolumn typu XML.

Kiedy w praktyce należy stosować typ danych XML? W tej kwestii zdania są podzielone. Jedni z definicji odrzucają XML traktując go jako niepotrzebny, a wręcz szkodliwy wodotrysk (stawiając m.in. zarzuty co do kiepskiej wydajności), drudzy używają go, gdzie tylko się da, zastępując jedną kolumną XML strukturę kilku tabel lub tworząc procedury składowane, którym przekazuje się tylko jeden parametr typu XML, z którego są potem „wydłubywane” konkretne wartości. W skrajnych przypadkach cała komunikacja z bazą danych sprowadza się do wymiany dokumentów XML. Z aplikacji przychodzi żądanie z parametrami w postaci XML, na które baza odpowiada zwracając dokument XML z odpowiednią strukturą danych. Sprowadza to komunikację z bazą danych do postaci zbliżonej do korzystania z usług sieciowych (ang. *webservices*), które stają się w ostatnich latach coraz bardziej popularne.

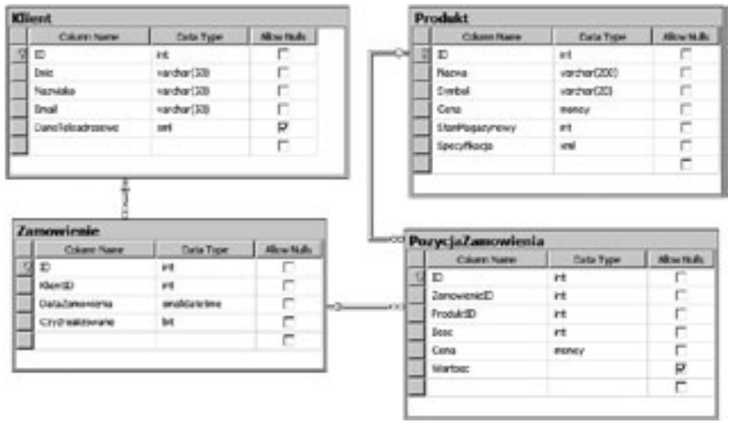
W przypadku niniejszego wykładu, z typu danych XML skorzystaliśmy w dwóch tabelach w przykładowej bazie danych (patrz rysunek 1).

W pierwszej tabeli (Klient) kolumna XML służy do przechowywania danych teleadresowych klienta (różne rodzaje adresów, identyfikatorów z komunikatorów itp.). W drugiej tabeli (Produkt), XML zostało zastosowane do przechowywania specyfikacji produktu. Ze względu na to, że produkty różnych kategorii mogą być opisywane zupełnie innym zbiorem cech – zastosowanie XML w połączeniu z kolekcją schem zapewni wygodny mechanizm przechowywania specyfikacji produktów. Podobnie wygląda sytuacja w przypadku pierwszej kolumny – tu kolekcja schem ma zapewnić opisanie i wymuszenie odpowiedniej postaci listy adresów/kontaktów klienta. W skład takiej listy mogą wchodzić adresy pocztowe, email czy identyfikatory różnych komunikatorów internetowych.

Skorzystanie z typu danych XML przyczyniło się do znacznego uproszczenia struktury bazy danych. W przypadku obu tabel przechowywane dane byłyby w postaci zmiennego zbioru cech i ich wartości przypisanych do konkretnego klienta czy produktu. Przy takim rozumieniu problemu jedna kolumna XML eliminuje ze struktury bazy co najmniej dwie tabele (słownik cech oraz przypisanie cechy i jej wartości do klienta/produktu). Dodatkowo, ze względu na niewielki rozmiar danych XML i sposób korzystania z nich, nie ma co się zbytnio przejmować spadkiem wydajności. W innych przypadkach jest to jednak bardzo istotne kryterium, które często staje się jedną z głównych przyczyn zarzucenia stosowania typu danych XML w konkretnym projekcie.

Manipulowanie danymi typu XML

Oprócz samej możliwości przechowywania dokumentów XML w kolumnie w bazie danych, typ danych XML oferuje jeszcze inne możliwości. Są one udostępniane jako metody, które można wywoływać na rzecz kolumny. Umożliwiają zaawansowane odpytywanie dokumentu XML oraz manipulowanie jego strukturą. Mechanizm ten działa w oparciu o język XQuery oraz wyrażenia XPath. Warto zaznaczyć, że implementacja XQuery i XPath w SQL Server 2008 nie zawiera wszystkich możliwości wynikających z ich specyfikacji.



Rysunek 1.
Tabele przykładowej bazy danych

Język XQuery, jak i wspomniany już wcześniej XPath, są rozwiązaniami stworzonymi i rozwijanymi przez organizacje W3C (ang. *World Wide Web Consortium*). Zastosowanie ich w miejsce własnych (Microsoftu) rozwiązań jest przejawem przykładania coraz większej wagi do otwartych technologii, co jest cechą charakterystyczną całego świata XML. Język XPath jest zaprojektowany do wskazywania (wybierania) odpowiednich węzłów dokumentu XML. Wyrażenia XPath z reguły nie funkcjonują samodzielnie, są natomiast szeroko stosowane w innych rozwiązaniach (np.: XSLT, XQuery).

Rola XQuery, jak sama nazwa wskazuje, polega na odpytywaniu dokumentu XML. Oprócz prostego wariantu bazującego na wyrażeniach XPath, można stosować rozbudowane wyrażenia FLWOR (nazwa pochodzi od angielskich słów: *For*, *Let*, *Where*, *Order by*, *Return*). XQuery bywa często porównywany do polecenia SELECT znanego z języka SQL, i mówi się, że XQuery jest dla XML tym, czym SELECT dla SQL.

Wróćmy jednak do możliwości typu danych XML. Otóż zawiera on pięć metod, które mają za zadanie ułatwienie korzystania z danych zawartych wewnątrz dokumentu XML oraz modyfikowanie samego dokumentu. Dodatkowo dają one możliwość wykonania operacji odwrotnej do działania klauzuli FOR XML – czyli do przetransformowania danych z XML w wiersz zbioru wyników (ROWSET).

Aby korzystać z metod typu XML, należy opanować dodatkowo podstawy wspomnianego języka XQuery oraz XPath, gdyż wyrażenia zbudowane w oparciu o nie są stosowane w parametrach wywołania metod typu XML.

Krótki opis metod typu XML zawiera poniższe zestawienie:

- Metoda `value(xquery,typ)` służy do wskazania poprzez wyrażenie XPath elementu lub atrybutu, którego wartość będzie pobrana z dokumentu XML, a następnie skonwertowana do typu wskazanego w drugim parametrze wywołania metody `value()`.
- Metoda `exist(xquery)` stosowana jest do sprawdzenia, czy kolumna XML zawiera w swojej wartości element lub atrybut wskazany przez wyrażenie XQuery. Podobny efekt da się osiągnąć za pomocą metody `value()`. Jeżeli jednak nie ma konieczności pobierania wartości z XML, a tylko sprawdzenia jej istnienia, to metoda `exist()` jest zalecana ze względu na szybsze działanie.
- Metoda `query(xquery)` służy do wskazania przez wyrażenie `xquery` zbioru węzłów z dokumentu XML, które są następnie zwracane także jako zmienna typu XML.
- Metoda `nodes(xquery)` jest używana do przetworzenia danych zawartych w dokumencie XML na postać relacyjną. Z jej pomocą (oraz z wykorzystaniem operatora CROSS APPLY) można wybrać węzły dokumentu, które będą tworzyły kolumny wiersza danych w zbiorze wynikowym zapytania SELECT. Rezultatem działania metody `nodes()` jest zbiór wierszy zawierający logiczne kopie węzłów dokumentu XML wybranych przez wyrażenie `xquery`. Funkcja ta jest szczególnie użyteczna i wygodna, gdy tworzymy zapytanie, które ma zawierać w kolumnach poszczególne informacje zaszyte w strukturze elementów i atrybutów dokumentu XML.
- Metoda `modify(xmldml)` służy do modyfikowania zawartości dokumentu XML. Modyfikacje te są realizowanie za pomocą poleceń języka XML DML (ang. *XML Data Manipulation Language*). W skład XML DML wchodzą trzy polecenia:
  - `insert` – służące do dodawania nowych węzłów (elementów, atrybutów, węzłów tekstowych itp.) do dokumentu XML
  - `delete` – stosowane do usuwania węzłów z dokumentu
  - `replace value of` – służące do zastępowania zawartości węzła dokumentu inną zawartością

    Możliwości tych trzech poleceń są dość ograniczone i łatwe do zastosowania wyłącznie w przypadku prostych modyfikacji operujących z reguły na wartościach podawanych w postaci stałych łańcuchów znaków. Gdy potrzebne są możliwości dynamicznego budowania wartości, która ma być wstawiona do dokumentu, to szybko okazuje się, że jest to trudne bądź wręcz niemożliwe. Dlatego, gdy wymagane są bardziej złożone operacje na dokumencie XML, to są realizowane one po stronie aplikacji, a ich gotowy wynik jest przekazywany do bazy.

Możliwości manipulowania danymi zapisanymi w kolumnach lub zmiennych typu XML są istotnym elementem składowym funkcjonalności obsługi XML w bazie danych. Umożliwiają realizowanie typowych operacji na danych XML w sposób zbliżony do znanego ze świata XML. Jest to bardzo istotne ze względu na łatwość zastosowania tych rozwiązań w praktyce.

PODSUMOWANIE

W ramach niniejszego wykładu zaprezentowany został pokrótce język XML oraz możliwości korzystania z niego w relacyjnych bazach danych. Z racji rosnącej popularności tego języka oraz coraz bogatszego wsparcia ze strony producentów serwerów baz danych, warto się nim zainteresować, aby móc rozważać go jako jedną z opcji przy projektowaniu baz danych. Jako przykład serwera baz danych, oferującego wsparcie dla XML, wykorzystany został SQL Server 2008. W ramach omawiania jego możliwości w kontekście XML, zasygnalizowane zostały główne obszary, w których serwer oferuje narzędzia służące do obsługi dokumentów XML. Przyjrzeliśmy się klauzuli FOR XML stosowanej do zwracania wyników zapytania w postaci dokumentów lub fragmentów dokumentów XML. Zapoznaliśmy się także z typem danych XML, który służy nie tylko do przechowywania danych w tym formacie, ale także do zaawansowanego odpytywania dokumentów XML oraz manipulowania ich zawartością. Dodatkowo, wspomnieliśmy o wykorzystywanych w ramach SQL Server 2008 innych technologiach i narzędziach wspierających XML. Warto tu wymienić choćby XQuery, XPath oraz XML Schema. Wszystkie te rozwiązania są otwarte, nie stanowią własności żadnej firmy, są rozwijane przez konsorcjum W3C i szeroko adoptowane w świecie IT. Umożliwia to ekspertom od XML na łatwe wykorzystanie nabytej wiedzy również przy używaniu z serwerów baz danych.

Jest jednak także druga strona medalu. Jeśli zachłyśniemy się wsparciem dla XML w SQL Server 2008, to bardzo szybko może nas czekać rozczarowanie. Otóż mechanizmy wspierające XML na pierwszy rzut oka wyglądają na potężne i wygodne. Faktycznie jest tak, ale tylko w zakresie przewidzianym przez ich twórców. Okazuje się, że nie zaimplementowali oni w pełni standardów XQuery i XPath (można szybko natrafić na wiele nieobsługiwanych funkcji bądź ograniczeń). Podobnie jest w przypadku XML Schema. Nie wszystkie możliwości tej technologii są dopuszczalne w ramach SQL Server 2008. W zakresie manipulowania strukturą dokumentów XML również okazuje się, że metoda `modify()` z poleceniami XML DML ma bardzo dużo ograniczeń, szczególnie gdy chce się polecenia tworzyć bardziej dynamicznie. To wszystko nie umniejsza jednak faktu, że wsparcie XML w relacyjnych bazach danych jest ciekawą możliwością, z której warto korzystać, po uprzednim gruntownym zapoznaniu się z dokumentacją, aby uniknąć przykrych niespodzianek w trakcie realizacji projektu.