Andrzej PTASZNIK[1]

PROJEKTOWANIE BAZ DANYCH Z WYKORZYSTANIEM NIERELACYJNYCH TYPÓW DANYCH

Streszczenie

Podejście do projektowania relacyjnych baz danych, przez wiele lat realizowane było zgodnie z metodyką ERD (ang. *Entity Relationship Diagrams*) lub ORM (ang. *Object-Role Modeling*). Wprowadzenie do standardu SQL typu XML, dodanie typów przestrzennych oraz hierarchicznych zmieniło uwarunkowania realizacji projektu bazy danych. W artykule opisane zostały możliwości wykorzystania nierelacyjnych typów danych przy projektowaniu baz danych z wykorzystaniem technologii MS SQL Server 2008 R2.

Abstract

Approach to designing relational databases, for many years was carried out in accordance with the ERD (*Entity Relationship Diagrams*) methodology, or ORM (*Object-Role Modeling*). Adding XML to the SQL standard as well as incorporating spatial and hierarchical types changed conditions of the database project realization. The article describes the possibilities of using non-relational data types when designing databases using Microsoft technology SQL Server 2008 R2.

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 opartych o XML możemy obserwować od ponad dziesięciu lat. W takiej sytuacji jest dość oczywiste, że język XML pojawił się w otoczeniu relacyjnych baz danych. Wprowadzenie w MS SQL Server 2008 typów przestrzennych oraz specjalnego typu danych *hierarchyid* jest kolejnym elementem, który należy rozważyć w trakcie projektowania baz danych. Dopełnieniem nowych możliwości i wyzwań, przed którymi stoją projektanci baz danych, jest możliwość (wprowadzona w MS SQL Server 2005) definiowania własnych obiektowych typów danych w środowisku .NET CLR. Klasyczne metodyki

[1] Mgr inż. Andrzej Ptasznik jest wykładowcą Warszawskiej Wyższej Szkoły Informatyki.

projektowania ERD i ORM nie uwzględniają wykorzystania nowych typów danych, ponieważ oparte są na modelu relacyjnym. Stosowanie nowych, nierelacyjnych typów danych zmienia znacząco możliwości projektowania baz danych, ale brak jest jednolitej metodyki, która odnosi się do tych zagadnień. Projektanci muszą bazować na doświadczeniu i intuicji nie mogąc swoich rozwiązań oprzeć o przyjętą metodykę. Korzyści, jakie można osiągnąć wykorzystując nierelacyjne typy danych, są znaczące i należy spodziewać się wielu prac, które będą podejmowały tematykę hybrydowego projektowania baz danych.

2 TYP DANYCH XML

Wraz z intensywnym rozwojem Internetu pojawił się, a właściwie nabrał większego znaczenia problem przekazywania danych pomiędzy różnymi systemami. Problem ten istniał od momentu 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 został sprowadzony 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 stro-

nom WWW. Prowadziło to do pogłębiania się chaosu w świecie technologii internetowych.

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.

Opisane wyżej uwarunkowania spowodowały potrzebę przetwarzania dokumentów XML w środowisku relacyjnych baz danych, co wymagało stworzenia możliwości realizowania trzech podstawowych operacji:

- składowanie dokumentów XML,
- przekształcanie danych relacyjnych do postaci XML,
- pobieranie danych z dokumentu XML.

W MS SQL Server 2000 wprowadzono pierwsze rozszerzenia umożliwiające korzystanie z dokumentów XML. Praktycznie umożliwiono przekształcanie wyników zapytań do postaci dokumentów XML wprowadzając do polecenia SELECT języka SQL klauzulę FOR XML. Do pobierania danych z dokumentu XML wprowadzono trzy funkcje:

- sp_xml_preparedocument,
- sp_xml_removedocument,
- OPENXML.

Cytowane funkcje umożliwiały pobieranie wyspecyfikowanych danych z dokumentu XML i zwracanie wyniku w postaci tabeli. Stosowanie ich było jednak kłopotliwe i dla dużych dokumentów XML bardzo niewydajne. Największym problemem, w MS SQL Server 2000, było składowanie XML w bazie danych. Praktycznie jedyną możliwością były kolumny typu *text*, czyli statyczne przechowywanie danych XML bez możliwości bezpośredniego odwoływania się do jego struktury. Pomimo problemów ze składowaniem, popularność przekazywania danych w postaci dokumentów XML powodowała konieczność radzenia sobie z opisanymi niedogodnościami i w wielu systemach korzystano z opisanych możliwości przechowywania i obróbki danych przechowywanych w postaci dokumentów XML.

Przełomem było wprowadzenie w MS SQL Server 2005 typu 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 (*typed XML*),
- nieskojarzone z XML Schema (*untyped XML*).

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

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 element w relacyjnych bazach danych (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 pobierane są 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 (*webservices*), które stają się w ostatnich latach coraz bardziej popularne.

Oprócz samej możliwości przechowywania dokumentów XML w kolumnie tabeli, typ danych XML oferuje jeszcze inne możliwości. Są one udostępniane jako metody, które można wywoływać na rzecz zawartości 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.

Język XQuery, jak i wspomniany już wcześniej XPath są rozwiązaniami stworzonymi i rozwijanymi przez organizacje W3C (ang. *World Wide Web Consortium*). 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 rozbudo-

wane wyrażenia FLWOR (nazwa to akronim pochodzący od 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 do postaci tabeli (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)** służy do sprawdzenia czy kolumna XML zawiera 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)** służy 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 – służące 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 danych.

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 SQL. Jest to bardzo istotne ze względu na łatwość zastosowania tych rozwiązań w praktyce.

3 TYP DANYCH HIERACHYID

Kolejnym, nierelacyjnym typem danych, wprowadzonych w MS SQL Server 2008, jest typ ***hierarchyid***. Został on stworzony z użyciem technologii .NET (wykorzystuje integrację systemu SQL Server ze środowiskiem uruchomieniowym CLR). Typ ten wspiera obsługę danych hierarchicznych, takich jak:

- struktury organizacyjne,
- system plików,
- hierarchia pracowników w organizacji,
- hierarchia stron WWW.

Projektowanie różnych hierarchii w modelu relacyjnym opierało się głównie na dodaniu do tabeli dodatkowego pola odwołującego się do jej klucza głównego. Struktura taka, zwana samozłączeniem, komplikowała proces pobierania danych, ponieważ przeszukiwanie hierarchii wymagało podejścia rekurencyjnego. Język SQL, przez długi czas, nie udostępniał możliwości rekurencyjnych zapytań, co wymuszało pisanie skomplikowanych procedur w innych językach (np. T-SQL) lub przenoszenie logiki przeszukiwania hierarchii na stronę klienta bazy danych. Trzeba dodać, że wprowadzone w MS SQL Server 2005 wyrażenia CTE (ang. *Common Table Expressions*) umożliwiają pisanie

rekurencyjnych zapytań w języku SQL. W kolumnie typu ***hierarchyid*** można przechowywać zapisaną w specjalnym formacie pozycję rekordu w drzewie hierarchii (wpisywaną jako tekst pozycję przechowywaną w postaci binarnej). Typ został wyposażony w zestaw metod, które umożliwiają sprawne poruszanie się po drzewie hierarchii:

- **GetAncestor(n)** – metoda zwraca przodka znajdującego się n poziomów wyżej w drzewie hierarchii od węzła, na rzecz którego została wywołana.
- **GetDescendant(wartość1, wartość2)** – metoda zwraca określonego przekazanymi wartościami potomka dla danego „rodzica".
- **GetLevel** – metoda zwraca poziom węzła w hierarchii (głębokość drzewa), na rzecz którego została wywołana.
- **GetRoot** – zwraca korzeń drzewa.
- **IsDescendantOf(wartość)** – metoda zwraca wartość 1, gdy przekazany węzeł jest potomkiem węzła, na rzecz którego metoda została wywołana.
- **Parse(wartość)** – metoda konwertuje przekazaną wartość kanoniczną węzła, na wartość binarną.
- **GetReparentedValue(wartość1, wartość2)** – metoda zwraca nową wartość węzła, na rzecz którego została wywołana, po zmianie gałęzi drzewa.
- **ToString** – metoda konwertuje binarną wartość węzła na wartość kanoniczną.

Wprowadzenie typu *hierarchyid* stwarza projektantom baz danych kolejną możliwość wyrażanie struktur, które w swej istocie nie są relacyjne.

4 TYPY PRZESTRZENNE

Wprowadzenie, w MS SQL Server 2008, przestrzennych typów danych otworzyło nowe możliwości w zakresie przechowywania danych o obiektach przestrzennych oraz ułatwiło budowę Systemów Informacji Geograficznej (GIS).

MS SQL Server 2008 dostarczył dwa typy danych przestrzennych – *geometry* i *geography*. Typ *geometry* służy do przechowywania i przetwarzania danych przestrzennych opartych na współrzędnych płaskich (planarnych), a typ *geography* służy do przechowywania i przetwarzania danych geograficznych opartych na współrzędnych geodezyjnych. Zarówno w *geometry*, jak i w *geography* opis położenia punktu możliwy jest tylko w dwóch wymiarach. SQL Server dostarcza nam również dwie dodatkowe wartości do opisu każdego punktu – Z i M. Dane te nie są wykorzystywane w żadnych obliczeniach, pozostają jedynie do dyspozycji użytkownika, np. do przechowywania informacji o wysokości położenia punktu nad poziomem morza.

Dane typów *geography* i *geometry* możemy przedstawiać w trzech formatach:

- WKT (ang. *Well-Known Text*),
- WKB (ang. *Well-Known Binary*),
- GML (ang. *Geography Markup Language*).

Formaty te zdefiniowane są w standardach dostarczanych przez Open Geospatial Consortium (OGC). SQL Server 2008 wspiera siedem typów obiektów służących do przedstawiania danych w przestrzeni. W tabeli 1 przedstawiono krótki opis obiektów reprezentowanych poprzez typy przestrzenne.

Tabela 1. Typy obiektów przestrzennych

Nazwa	Uwagi
Point	Reprezentuje punkt
LineString	Reprezentuje łamaną
Polygon	Reprezentuje wielokąt
MultiPoint	Reprezentuje zbiór punktów
MultiLineString	Reprezentuje zbiór łamanych
MultiPolygon	Reprezentuje zbiór wielokątów
GeometryCollection	Reprezentuje zbiór obiektów różnych typów

Typy przestrzenne są typami obiektowymi, dlatego oprócz mechanizmu składowania danych udostępniają szereg metod umożliwiających wykonywanie rożnych operacji na danych przestrzennych. Część metod to metody statyczne, służące głownie do tworzenia instancji obiektu przestrzennego. Instancję obiektu przestrzennego można utworzyć albo na podstawie standardu Well-Known Text albo Well-Known Binary. Przykładowe metody statyczne umożliwiające tworzenie instancji obiektu przestrzennego to:

- STGeomFromText – tworzy figurę na podstawie WKT,
- STPointFromText – tworzy punkt na podstawie WKT,
- STGeomFromWKB – tworzy figurę na podstawie WKB,
- STPointFromWKB – tworzy punkt na podstawie WKB.

Drugą grupę metod stanowią metody działające na rzecz instancji obiektu przestrzennego. Przykładowe metody instancyjne to:

- **STArea** – zwraca powierzchnię podanej figury,
- **AsGml** – zwraca wartość reprezentującą podaną figurę w postaci Geography Markup Language (GML),
- **STBuffer** – tworzy figurę będącą powłoką oddaloną od naszej figury o podaną odległość,
- **STCentroid** – zwraca punkt będący środkiem ciężkości podanej figury,

- **STConvexHull** – przekształca podaną figurę na możliwie najmniejszą figurę wypukłą,
- **STDifference** – zwraca figurę zawierającą punkty z jednego zbioru i niezawierającą punktów z innego zbioru,
- **STDistance** – zwraca najkrótszą odległość między figurami,
- **STLength** – zwraca długość linii bądź obwód figury.

Z punktu widzenia projektowania baz danych, typy przestrzenne nie wprowadzają istotnych zmian w podejściu do projektu a jedynie stwarzają pełne możliwości przechowywania i przetwarzania danych o obiektach przestrzennych oraz ułatwiają budowę aplikacji zobrazowujących różne aspekty danych zobrazowanych na mapach.

5 TYPY DANYCH DEFINIOWANE W ŚRODOWISKU CLR

Przełomową zmianą w technologii MS SQL Server była wprowadzona w MS SQL Server 2005 pełna integracja ze środowiskiem .Net CLR. Dzięki integracji z CLR możemy pisać własne procedury składowane, funkcje użytkownika oraz wyzwalacze wybierając do tego celu dowolny język .NET Framework. Stwarza to bardzo duże możliwości realizacji pełnej logiki biznesowej po stronie serwera. Te możliwości nie wpływają bezpośrednio na projektowanie baz danych a jedynie są potencjałem do wykorzystania na etapie jej implementacji.

Przełomową zmianą związaną z integracja z CLR jest możliwość definiowania typów danych SQL Server jako klas lub struktur w językach .Net. Takie typy danych mogą mieć bardzo złożoną strukturę przewidziana do przechowywania danych, a ich zachowanie jest zdeterminowane poprzez metody i właściwości klasy. Z powyższego opisu można łatwo wywnioskować, że wykorzystując w SQL Server obiektowe typy danych wykraczamy daleko poza relacyjne zasady i uwarunkowania. Typy definiowane w środowisku CLR są mocno zintegrowane z systemem typów i silnikiem SQL Server i mogą być wykorzystane w każdym kontekście, w jakim wbudowane typy są dozwolone. Projekt bazy danych, w którym wykorzystywane są typy obiektowe trudno nazwać projektem relacyjnym i z pewnością jest to zupełnie nowa jakość. Wykorzystanie typów CLR w środowisku SQL Server nie odbiega znacząco od klasycznych wbudowanych typów danych, ponieważ mogą być używane jako typ danych kolumny, typ zmiennej lub parametru procedury, a także można indeksować kolumny obiektowe i wykorzystywać je w procesie replikacji baz danych. W kontekście zapytań języka SQL, pobieraniu danych towarzyszyć może wywoływanie metod na rzecz odczytanej instancji. Tego typu działa-

nie jest dostępne dla wcześniej omówionych typów danych (XML, hierarchyid i typy przestrzenne).

Podczas definiowania typu CLR w środowisku .Net trzeba skonkretyzować niektóre deklaracje:

- wyspecyfikować SqlUserDefiniedTypeAttribute,
- określić atrybut Serializable,
- zaimplementować interfejs IsNullable w klasie lub strukturze, tworząc pustą metodę public static,
- zaimplementować metody konwersji public static Parse i ToString, które umożliwiają przekształcenie do ciągu znakowego reprezentującego obiekt i analizę jego składni.

Zasady definiowania typów CLR nie są zbytnio złożone a ich stosowanie otwiera nieograniczone możliwości podejścia do projektowanych baz danych.

6 WYKORZYSTANIE W PROJEKTOWANIU NIERELACYJNYCH TYPÓW DANYCH

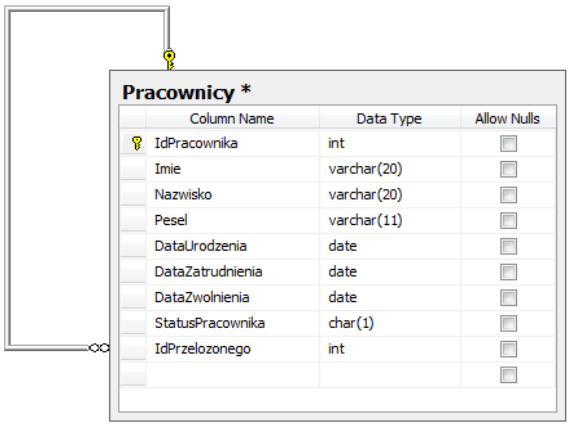
Omówione powyżej nowe typy danych stanowią pewien zbiór potencjalnych możliwości, które mogą być wykorzystane w procesie projektowania baz danych. W dalszych rozważaniach skoncentruję się na typach *XML* i ***hierarchyid***, ponieważ stosowanie tych typów danych pozwala zmieniać podejście do projektu w stosunku do rozwiązań realizowanych wcześniej.

W tabeli 2 przedstawione zostały kierunki zastosowań nierelacyjnych typów danych.

Tabela 2. Stosowanie nierelacyjnych typów danych

Typ danych	Baza danych	Aplikacja
Geometry, geography	Przechowywanie obiektów przestrzennych w powiązaniu z danymi relacyjnymi.	Wykorzystanie danych przestrzennych do zobrazowań na mapach.
Hierarchyid	Uproszczenie opisu rożnego typu hierarchii w schemacie bazy danych	Prezentacja hierarchii w postaci widoku drzewa.
XML	Przechowywanie złożonych struktur danych. Uproszczenie schematu bazy danych.	Uproszczenie komunikacji z bazą danych. Przekazywanie dużych struktur danych za pomocą jednego parametru.
Typy CLR	Wprowadzenie aspektów obiektowości do relacyjnej bazy danych. Bardzo duży potencjał rozmaitych zastosowań.	Uproszczenie modelu aplikacji poprzez korzystanie z klas, które maja bezpośrednie odwzorowanie w bazie danych

Aktualnie brak jest sprawdzonych metodyk i wiarygodnych zaleceń, kiedy i w jakim zakresie wykorzystywać typ danych XML i inne nierelacyjne typy danych w powiązaniu z relacyjną bazą danych. Projektanci, opierając się na własnych doświadczeniach, przecierają szlaki i ustalają standardy. Sprawa wygląda dość prosto w przypadku typu ***hierarchyid***, należy podjąć decyzję czy modelujemy dane hierarchiczne w sposób klasyczny z wykorzystaniem samozłączenia, czy wykorzystać nowy typ danych. W klasycznym – relacyjnym podejściu tabela opisująca pracowników i ich hierarchie podległości mogłaby wyglądać tak jak na rysunku 1.



Rys 1. Hierarchia pracowników jako samozłączenie

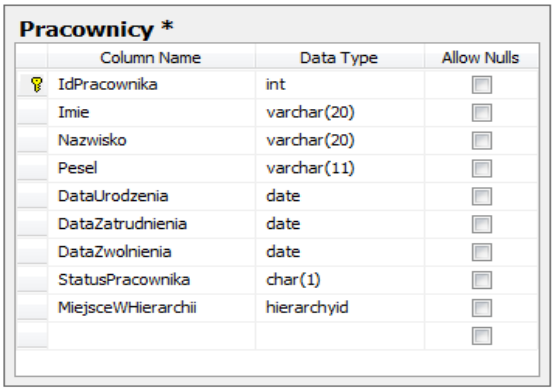
Przykładowe zapytanie, które ma pobrać nazwisko i imię pracowników, którzy są podwładnymi (na dowolnym poziomie hierarchii) pracownika o wartości Idpracownika= 2, mogłoby mieć następującą postać:

```
DECLARE @IdPracownika int=2;
WITH CTE AS
(
   SELECT Nazwisko,Imie,IdPracownika
   FROM Pracownicy
   WHERE IdPodwladnego=@IdPracownika
   UNION ALL
   SELECT Pracownicy.Nazwisko, Pracownicy.Imie, Pracownicy.IdPracownika
   FROM Pracownicy JOIN CTE
     ON Pracownicy.Idprzelozonego=CTE.IdPracownika
)
```

```
SELECT nazwisko, imie
FROM CTE;
```

Zapytanie wykorzystuje wyrażenie CTE do rekurencyjnego pobrania podwładnych dowolnego poziomu.

Tabela Pracownicy, w przypadku skorzystania z typu ***hierarchyid*** miałaby postać przedstawioną na rysunku 2.



Rys. 2. Tabela Pracownicy z kolumna typu ***hierarchyid***

Jak widać na rysunku 2, tabela nie wymaga samozłączenia i pozornie różnica w podejściu do projektu jest niewielka. Inaczej wygląda zapytanie, które ma pobrać nazwisko i imię pracowników, którzy są podwładnymi (na dowolnym poziomie hierarchii) pracownika o wartości Idpracownika=2:

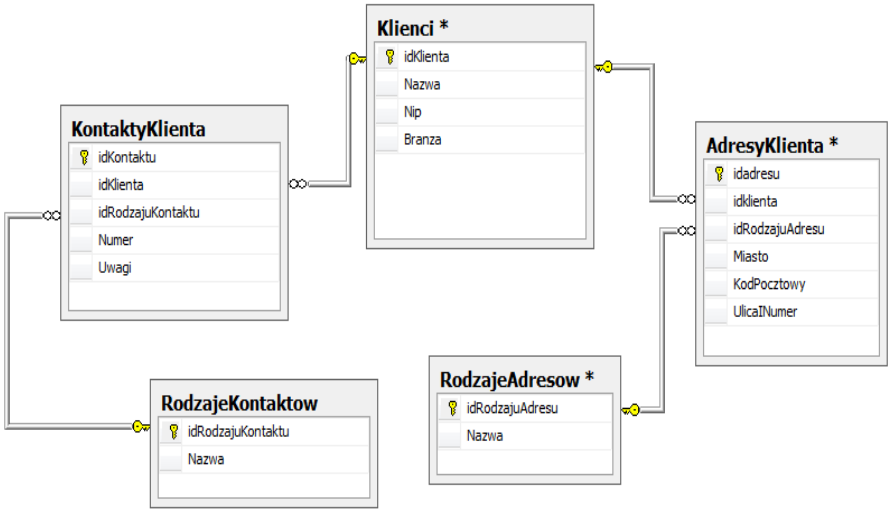
```
DECLARE @Pracownik hierarchyid=(SELECT MiejsceWHierarchii
                    FROM Pracownicy
                    WHERE IdPracownika=@IdPracownika);

SELECT Nazwisko,Imie,IdPracownika
FROM  Pracownicy
WHERE @Pracownik.IsDescendantOF(MiejsceWHierarchii)=1;
```

Myślę, że w praktyce znajdą się zwolennicy jednego i drugiego podejścia do projektowania struktur hierarchicznych w relacyjnych bazach danych.

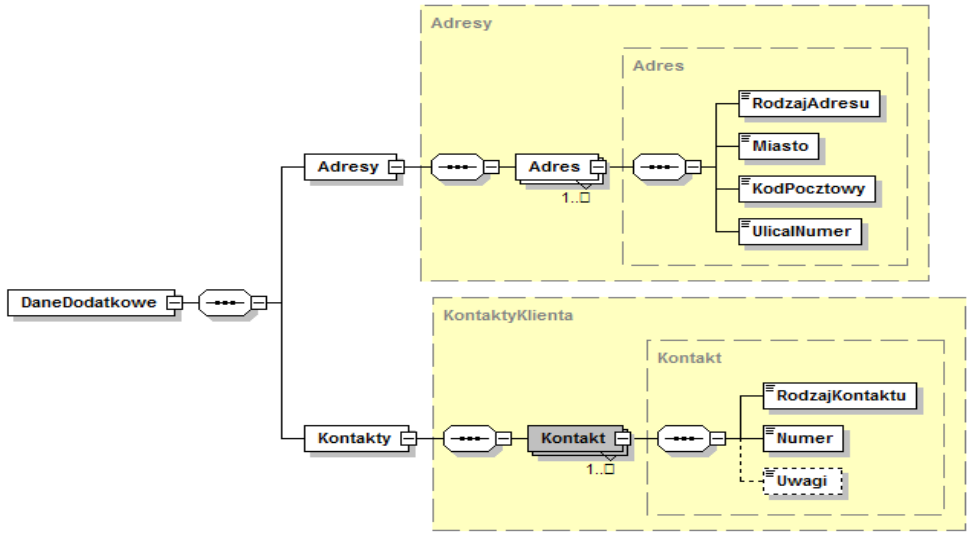
Największe zmiany, w podejściu do projektowania baz danych, wprowadził typ danych XML. Największą korzyścią jaką można osiągnąć, wykorzystując typ XML, jest

możliwość znacznego uproszczenia schematu bazy danych a tym samym uproszczenie działania aplikacji z niej korzystających. Na rysunku 3 przedstawiony został przykład bazy danych opisujący dane klientów. Chcąc uwzględnić mozliwość dowiązania do opisu klienta dowolnej ilości adresów róznego typu oraz dowolnej ilości kontaktów różnego typu, ten prosty schemat zawiera pięć tabel. W modelu relacyjnym trudno znaleźć lepsze rozwiązanie tego problemu. Warto w tym miejscu zauważyć, że adresy czy kontakty klienta są danymi, które najczęściej będą wykorzystywane w kontekście konkretnego klienta.



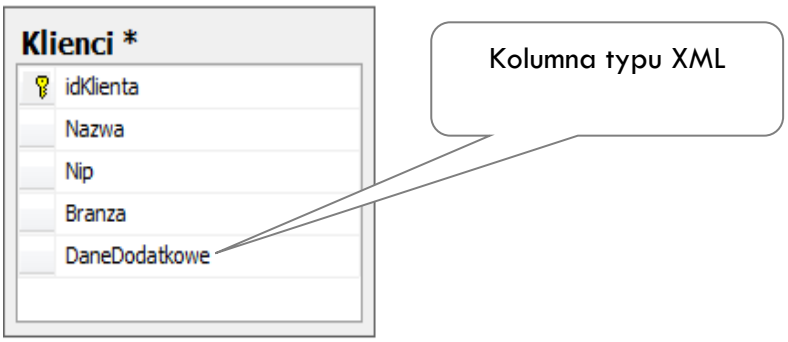
Rys. 3. Przykładowy schemat bazy danych

Mając do dyspozycji typ XML, można do tego problemu podejść inaczej. Na rysunku 3 przedstawiono schemat budowy dokumentu XML, który opisywałby adresy i kontakty klienta. Pokazany schemat opisuje budowę dokumentu i może zostać przekształcony do postaci dokumentu XSD(ang. *XML Schema Definition*), który po zdefiniowaniu go w bazie danych może służyć do walidacji dokumentu XML.



Rys 4. Schemat budowy przykładowego dokumentu XML

Można założyć, że niezbędne dane o adresach i kontaktach mogą być zapisane w odpowiednio sformatowanym dokumencie XML i przechowane w jednej kolumnie tabeli opisujacej klientów. Na rysunku 4 pokazano przykład tabeli opisujacej klientów z kolumną typu XML w której przechowywać będziemy dane o adresach i kontaktach.



Rys. 5. Tabela Klienci z kolumna typu XML

W efekcie końcowym zamiast pięciu tabel mamy jedna tabelę i zapewniamy przechowywanie i dostęp do takiej samej ilości danych jak w klasycznym relacyjnym ujęciu pokazanym na rysunku 2. Ten prosty przykład pokazuje pewien kierunek zmian w podejściu do projektowania baz danych. Bardzo często można osiagnąć bardziej spektakularne korzyści w uproszczeniu schematu bazy danych.

W jednym z zadań projektowych, realizowanych przez autora, należało zapewnić ewidencję wniosków o dofinansowanie w ramach pewnego funduszu pomocowego. Po zarejestrowaniu wniosku wszystkie szczegóły w nim opisane powinny być, w zależności od potrzeb, dostępne na poziomie aplikacji w procesie oceny wniosków. Zadanie wyglądało na dość złożone, ponieważ wniosek o dofinansowanie zawierał bardzo dużo danych różnego typu i chcąc zapewnić pobieranie dowolnego ich fragmentu należało zapewnić odpowiednie przechowywanie. Przeprowadzona analiza pokazała, że chcąc przygotować bazę danych w klasycznym relacyjnym ujęciu, należało utworzyć w bazie danych około 30 tabel do opisu różnych szczegółów wniosków o dofinansowanie. Obsługa programowa takiej bazy danych byłaby odpowiednio złożona i wykonanie wszystkich etapów bardzo pracochłonne.

W projekcie tym zdecydowano, że wszystkie dane zawarte we wniosku o dofinansowanie zostaną przechowane w dokumencie XML, co skutkowalo tym, że zamiast planowanych poczatkowo około 30 tabel powstała jedna. Na rysunku 5 pokazano fragment struktury dokumentu XML opisujacego dane zawarte we wniosku o dofinansowanie.

Rys. 6. Fragment dokumentu XML opisującego wniosek o dofinansowanie

Trudno, w ramach krótkiego artykułu, omówić wszystkie uwarunkowania omawianego projektu, ale ważne jest to, że przyjęte rozwiązanie doskonale sprawdziło się w praktyce. Omawiany system został wdrożony do eksploatacji w roku 2007 i w jej trakcie nie

wystąpiły problemy z tak przyjętym rozwiązaniem. Korzyści osiągnięte dzięki przyjęciu omawianego rozwiązania to:

- uproszczenie schematu bazy danych,
- zmniejszenie pracochłonności oprogramowania bazy danych (procedury składowane),
- zmniejszenie pracochłonności przy budowie interfejsu użytkownika.

Omawiane przykłady pokazują pewien kierunek wykorzystania typu danych XML w projektowaniu baz danych koncentrując się na korzyściach osiąganych z uproszczenia schematu.

Wprowadzenie do technonogii MS SQL Server nierelacyjnych typów danych stwarza dodatkowe możliwości i wyzwania dla projektantów baz danych. Problemem w tej sytuacji jest fakt, że nie istnieje przyjęta metodyka, która pozwalałaby jednoznacznie ocenić kiedy i w jakim zakresie typy nierelacyjne należy stosować. Dodatkowym problemem są sprawy wydajności i optymalizacji systemów baz danych. Brak badań, które dawałyby właściwe wskazówki odnośnie wpływu wykorzystania nierelacyjnych typów danych na wydajność bazy danych.