Marek Szepski

Modelowanie zarządzania danymi w bazach danych

1. Problem modelowania zarządzania danymi

Modelowanie zarządzania danymi rzadko pojawia się literaturze poświęconej bazom danych. Jest jednak oczywiste, że dane przechowywane w BD są zarządzane i dzieje się to zgodnie z jakimś modelem. W relacyjnym systemie zarządzania bazą danych (RSZBD) model wynika z reguł sformułowanych przez Codda. Przyjmowany jest on wtedy jako „naturalny, wbudowany w bazę danych" i nie zawsze uświadamiamy sobie jak sztywny jest to model. Niewątpliwe zalety RSZBD okupione są niestety wieloma ograniczeniami.

Dominujące obecnie modelowanie obiektowe koncentruje się na analizie dziedziny problemu. Analiza diagramów klas typowych przykładów prezentowanych w literaturze, pokazuje, że zarządzanie danymi jest pominięte. Autorzy koncentrują się na modelowanym problemie, abstrahując od realizacji systemu. Można przypuszczać, że w wielu przypadkach system taki byłby realizowany jako baza hybrydowa obiektowo-relacyjna i zarządzanie danymi odbywałoby się według schematu relacyjnego. Modelowanie obiektowe daje jednak większe możliwości, a nie tylko proste sprowadzenie do znanych schematów RSZBD.

Dla baz danych budowanych w oparciu o model obiektowy nie istnieją sztywne standardy podobne do reguł Codda. Daje to twórcom systemu możliwości dowolnego modelowania rozwiązywanego problemu, ale zobowiązuje także do określenia modelu zarządzania danymi.

2. Modelowanie zarządzania danymi w RSZBD

W przypadku budowania bazy danych w oparciu o RSZBD problem modelowania zarządzania danymi został rozwiązany „odgórnie". Reguły sformułowane przez Egdara Codda precyzyjnie określają wszystkie aspekty RSZBD, a niektóre szczegółowo odnoszą się do zarządzania danymi [1], [6]. Przyjęto, że mianem RSZBD określa się jedynie te systemy, które są oparte na regułach Codda. Przeanalizujmy konsekwencje tych wybranych reguł.

> Reguła 1: Wszystkie informacje, znajdujące się w relacyjnej bazie danych, są reprezentowane wyłącznie na poziomie logicznym i w jednolity sposób – jako wartości w tabelach.

Ta prosta reguła ma daleko idące konsekwencje. Po pierwsze: nie mamy bezpośredniego dostępu do danych zapisanych na dysku. Decyzje o zapisie danych, sposobie zapisu czy momencie modyfikacji są przekazane systemowi zarządzania bazą danych i są realizowane automatycznie. Takie podejście zapewnia bezpieczeństwo danych, zdejmuje z projektantów i użytkowników konieczność pamiętania o utrwaleniu danych oraz uniemożliwia ich zmienienie z pominięciem systemu zarządzania. Inaczej mówiąc mamy dwie warstwy: warstwę tabel (struktur logicznych) dostępną dla użytkownika i warstwę danych (fizyczną, trwale zapisaną). Komunikację miedzy tymi warstwami zapewnia system zarządzania, na który

użytkownik nie ma wpływu. Projektując bazę danych nie musimy, a nawet więcej, nie możemy modelować zarządzania danymi, gdyż robi to za nas RSZBD.

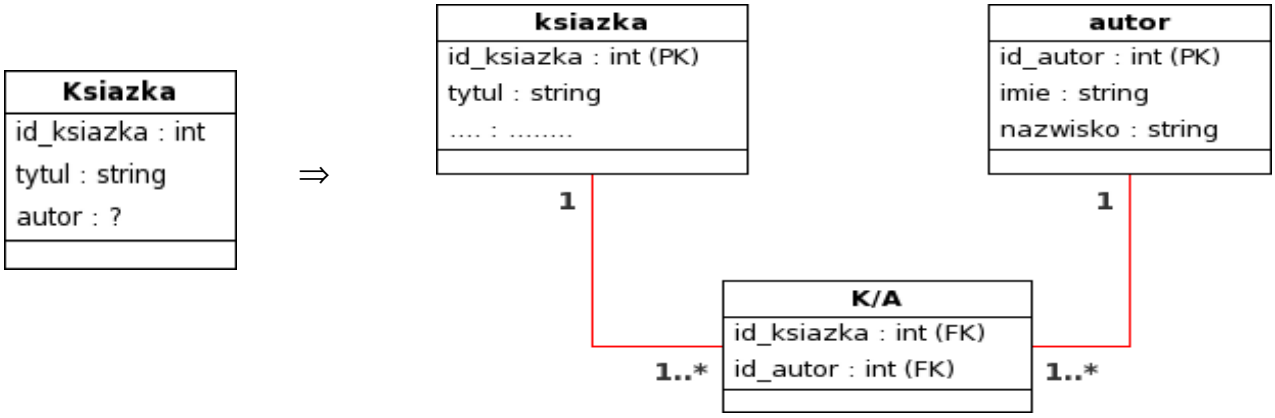
Po drugie: reguła ta ogranicza struktury, reprezentujące dane wyłącznie do tabel. Z punktu widzenia struktur danych możemy patrzeć na tabelę jako na wektor rekordów. W połączeniu z regułą nr 2 daje to duże ograniczenie.

> Reguła 2: Każda jednostka informacji (wartość atomowa, niepodzielna) w RSZBD musi być dostępna w sposób logiczny, przez odwołanie realizowane za pomocą kombinacji: nazwy tablicy, wartości klucza głównego i nazwy kolumny.

Ta reguła dotyczy przede wszystkim modelowania dziedziny problemu, ale z punktu zarządzania danymi, ze względu na wymóg niepodzielności danych, ogranicza te dane praktycznie do typów prostych. Oczywiście dostawcy RSZBD wprowadzają wiele różnych typów zmiennych np.: wiele typów liczbowych, różnorodne łańcuchy, różne postacie dat (pamiętajmy, że data jest liczbą, a typ oznacza jedynie sposób wyświetlania) czy nawet tablice, adresy URL, pliki itd. Pozostaje jednak istota niepodzielności, która oznacza, że dostęp do części składowych informacji wymaga napisania dodatkowych funkcji.

Konsekwencją tych, a także nie przywołanych tu reguł jest wymaganie określone w regule nr 5, którą można by streścić – istnieje język taki jak SQL i jest on wymaganą częścią RSZBD. W systemie mamy więc narzędzia służące do definiowania i przetwarzania danych, kontrolujących dostęp użytkowników, przebieg transakcji i integralność danych.

Integralność danych jest kluczowa dla użytkowników bazy danych. Reguły Codda 10 i 12 wprost mówią, że więzy integralności są częścią RSZBD (a nie aplikacji użytkownika) i są definiowane wraz z definiowaniem danych, oraz że nie jest dopuszczalne takie przetwarzanie danych, które mogłoby usunąć lub obejść te więzy. Konsekwencją chęci zachowania integralności danych jest proces normalizacji tabel, który wymaga podziału tabel na nowe tabele i prowadzi do zwiększenia ich liczby oraz zmniejszenia czytelności modelu. Efektem normalizacji jest powstanie tabel, które nie mają odpowiedników w istniejących obiektach. Rysunek 1 pokazuje typowe efekty poradzenia sobie ze związkiem wiele do wielu, czyli złożonymi wartościami jednego atrybutu. Ponieważ książka może mieć wielu autorów, to musimy stworzyć dodatkową tabele autorów i tabelę łączącą. Mamy więc książki (czy to jest jeszcze książka?) pozbawione autorów, autorów, o których nie wiadomo czy coś napisali, a rzeczywista książka ukrywa się pod kilkoma (!) rekordami w kompletnie nieczytelnej tabeli łączącej.



Rysunek.1 Normalizacja tabeli: Ksiazka

Integralność jest budowana na poziomie logicznym czyli na poziomie modelowania dziedziny problemu a nie na poziomie zarządzania danymi. Reguły opisujące wymagani integralności dotyczą w konsekwencji takich zagadnień jak wybór kluczy głównych i obcych czy przetwarzania danych o wartościach nieokreślonych (NULL).

Jak widać wymagania narzucone na RSZBD są bardzo mocne. Oparcie RSZBD na gruncie formalnej teorii, zdefiniowanie operacji na danych jako działań algebry relacyjnej i warunków, które muszą być spełnione aby otrzymywane wyniki tych działań były poprawne dało bardzo efektywne narzędzie baz danych.

3. Różnice w modelowaniu obiektowym i relacyjnym

Modelowanie obiektowe stało się obecnie standardowym podejściem do modelowania systemów. Nie narzuca ono ograniczeń związanych z modelem relacyjnym, jest bardziej elastyczne i pozwala budować modele powiązane w sposób naturalny z rzeczywistością.

Podstawowe dwa pojęcia modelowania, encja w modelu relacyjnym i obiekt są podobne. Encja jest definiowana np. następująco [1]:

> Encja jest osobą, miejscem, rzeczą lub pojęciem, które posiada cechy interesujące z punktu widzenia organizacji i o której chce się przechowywać informacje.

Obiekt może być definiowany następująco [7]:

> Obiekt to każdy byt – pojęcie lub rzecz – mający znaczenie w kontekście rozwiązywania problemy w danej dziedzinie przedmiotowej.

Mimo dużego podobieństwa encji i obiektu widać istotna różnicę. W przypadku obiektu nie mówi się przechowywaniu informacji, które to zadanie jest tylko jednym spośród możliwych jakie stawia się obiektom.

Uogólnieniem pojęcia obiektu jest klasa, a uogólnieniem encji są tabele. Niestety konieczna jest liczba mnoga, co wyraźnie widać z rysunku 1, gdzie encja: książka, została zamodelowana jako 3 tabele, choć w innych przypadkach tabela może być zbiorem encji. Porównanie pojęć tabela i klasa obrazuje różnice między tymi podejściami do modelowania.

Tabela	Klasa
Jest wzorcem oraz zbiorem obiektów	Jest tylko wzorcem obiektów
Zawiera trwałe rekordy	Tworzy ulotne lub trwałe obiekty
Musi zawierać atrybuty	Są klasy nie posiadające atrybutów
Atrybuty elementarne	Atrybuty elementarne lub złożone
Rekord musi posiadać unikalny identyfikator- klucz	Obiekty danej klasy są zawsze rozróżnialne niezależnie od wartości atrybutów
Pasywna	Aktywna

Tabela 1. Porównanie tabela – klasa

Jak widać różnice między tabelą i klasą są ogromne.

Definiując tabelę określamy strukturę rekordów, a równocześnie powstaje opakowanie zbiorcze rekordów, które pozwala odszukać utworzone rekordy. Niestety, obiekty utworzone przez klasy są jak wolne elektrony w przestrzeni i wymagają stworzenia innych struktur, w których będą przechowywane.

Inaczej też zachowują się rekordy i obiekty w chwili zakończenia działania aplikacji. Niestety obiekty istnieją jedynie w pamięci komputera i koniec pracy programu jest jednocześnie końcem ich istnienia. Koncepcja obiektowych baz danych oparta jest na trwałych obiektach tworzonych przez klasy określone stereotypem: <<persistence>>. Narzędzia CASE (np. Visual Paradigme) potrafią klasy <<persistence>> diagramu klas przekształcić automatycznie w diagram ERD. W modelu relacyjnym zadaniem encji jest przechowywanie danych. Zadania powierzane obiektom mogą być bardziej skomplikowane. Istnieją obiekty sterujące, kontrolujące przebieg programu, obiekty graniczne, odpowiedzialne za przekazywanie informacji i oczywiście obiekty przechowujące. Trwałość dotyczy jedynie części modelu, a dokładniej klas przechowujących.

W modelu obiektowym istnieją klasy abstrakcyjne tzn. takie, które nie tworzą obiektów. Dotyczy to klas nie mających atrybutów (np. interfejs) lub stanowią tylko wzorzec określający atrybuty lub operacje, wykorzystywany przez klasy dziedziczące. Takie klasy są ważne dla pełnego opisu systemu, nie posiadają jednak danych wymagających przechowywania.

Istotne różnice istnieją wśród dopuszczalnych typów atrybutów. Atrybuty obiektów mogą być strukturami złożonymi, a dokładniej mogą być typu określonej klasy, zarówno zdefiniowanej przez programistę jak i klasy standardowej. Mogą to być w szczególności:

- Zbiór, który zawiera nieuporządkowaną grupę elementów tego samego typu,
- Worek (ang. bag), który od zbioru odróżnia to, że może zawierać powtarzające się (duplikaty) elementy,
- Lista, uporządkowana grupa elementów tego samego typu,
- Tablica, analogiczna jak w językach programowania, do elementów której dostęp jest poprzez ich pozycję, ale o dynamicznym rozmiarze,
- Słownik, składający się z par: uporządkowany klucz i skojarzonej z nim wartości.

Złożone typy danych są potrzebne między innymi do zapewnienia nawigacji między obiektami. W przykładowej klasie: Ksiazka, atrybut: autor mógłby być np. listą i informacje o autorze mogłyby być dostępne po odwołaniu się do poszczególnych elementów tej listy.

Innym powodem stosowania złożonych atrybutów jest stworzenie dla obiektów utworzonych przez daną klasę, nowej klasy tzw. klasy kolekcji, przechowującej informacje o wszystkich tych obiektach. Klasa kolekcji pozwala uzyskać dostęp do wszystkich obiektów kolekcji.

Odwołania między obiektami są nawigacyjne. Oznacza to, że przetwarzanie informacji musi być wcześniej zaplanowane i zakodowane w programie. Tworzenie nowych zapytań do bazy w trakcie jej działania jest bardzo utrudnione. Jeśli chcemy wykorzystać mechanizmy relacyjne (np. zapytania SELECT z SQL) w bazie danych, wymaga to określenia kluczy głównych (patrz: reguła Codda nr 2) i przechowywania danych w tabelach. Otrzymujemy wtedy efektywny mechanizm wyszukiwania informacji w bazie (relacyjnej lub hybrydowej)

Modele zarządzania danymi w modelowaniu obiektowym

Problem modelowania zarządzania danymi bierze się z najczęściej obecnie stosowanego podejścia do modelowania systemów, opartego na języku UML (Unified Modeling Language). UML dostarcza bardzo szerokiego zestawu narzędzi do obrazowania i specyfikowania systemów, obejmującego zarówno dynamikę systemu jak i jego strukturę statyczną, oferuje obecnie 13 standardowych diagramów, które modelują różnorodne aspekty analizowanego systemu. Jednocześnie UML jest elastyczny i rozszerzalny, co pozwala dostosować go do specyficznych potrzeb konkretnego problemu. UML powstał z potrzeby prezentowania rozbudowanych systemów w sposób zrozumiały dla szerokiego grona osób współpracujących.

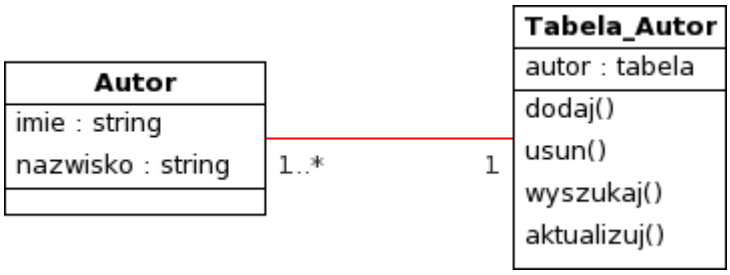
UML jest pomyślany w taki sposób, że koncentruje się na modelowanym systemie i jego złożonych aspektach, nie uwzględniając ograniczeń wprowadzanych przez narzędzia tworzenia aplikacji. Umożliwia to podjęcie decyzji o wyborze narzędzi programowania po zbudowaniu modelu i daje szansę wyboru narzędzi najlepszych. Analiza diagramu klas uwzględniająca np. dziedziczenie i inne struktury obiektowe, pozwala podjąć decyzję czy budujemy bazę w oparciu o narzędzia relacyjne, obiektowe, czy bazę hybrydową łączącą oba modle. Nawet jeśli z góry zakładamy, że będziemy tworzyć BD w oparciu o któryś RSZBD, to i tak mamy najpierw diagram klas, na podstawie którego budujemy diagram związków encji (ERD). Mamy wtedy obraz ograniczeń wprowadzonych przez model relacyjny.

Przekształcenie diagramu klas w diagram związków encji wymaga dokonania kilku operacji. Podobieństwa między tymi diagramami mogą sugerować, że jest to łatwe, ale tak być nie musi. Pierwszym krokiem powinno być zastąpienie złożonych atrybutów dodatkowymi tabelami. Elementem różniącym klasy od tabel są operacje. Pokusa prostego pominięcia operacji prowadzi do nieuzasadnionego uproszczenia modelu. Część operacji dotycząca zapisywania lub modyfikowania danych może być zamieniona w triggery – procedury zapisane jako element RSZBD i wyzwalane automatycznie, pozostałe staną się elementem warstwy reguł biznesowych.

Dalsze postępowanie jest już standardem przy modelowaniu relacyjnym i obejmuje: określenie kluczy głównych, dodanie tabel łączących, które zlikwidują związki wiele do wielu, określenie kluczy obcych i związków, określenie ograniczeń oraz przeprowadzenie normalizacji tabel.

Jeżeli chcemy zachować model obiektowy, a RSZBD użyć jedynie do przechowywania i wyszukiwania danych to prostym rozwiązaniem jest uzupełnienie modelu o klasy realizujące połączenie z RSZBD. Na przykład (rysunek 2) klasa: Autor jest powiązana z klasą: Tabela_Aktor. Operacje tej nowej klasy korzystając z bibliotek

odwołujących się do języka SQL, który jest wbudowany w RSZBD i realizują funkcje zarządzania danymi.



Rysunek 2. Klasa realizująca łączność z relacyjną bazą danych

Inny, bardziej rozbudowany model został zaproponowany w książce Argili i Yourdona [2]. Model wykorzystuje elementy modelowania obiektowego: dziedziczenie i agregację. Model systemu zawiera klasy opisujące charakterystykę tabel i kolumn. Klasy przechowujące dane dziedziczą po klasie opisującej obiekty jak i po abstrakcyjnej klasie charakteryzującej kolumny, a następnie zagregowanej w klasę opisującą tabele, która dodatkowo dziedziczy po klasie opisującej tabele. Celem takiego podejścia jest możliwość wielokrotnego użycia i łatwość modyfikacji, choć odbywa się to kosztem czytelności modelu.

Wybór modelu zarządzania danymi wymaga rozważenia najprostszego, czysto obiektowego rozwiązania. Złożone struktury atrybutów mogą być wprost zapisane w sposób trwały. Przy dużej liczbie danych określenie: wprost jest nieprecyzyjne, ale zapewne skorzystamy wtedy z któregoś z gotowych systemów obiektowych baz danych, który rozwiąże za nas powstające problemy.

Literatura:

1. Allen S.: Modelowanie danych; Helion 2006
2. Argila C., Yourdon E.: Analiza obiektowa i projektowanie; WNT 1999
3. Booch G., Rumbaugh J., Jacobson I.: UML przewodnik użytkownika; WNT 2002
4. Graham I.: Metody obiektowe w teorii i w praktyce; WNT 2004
5. Harrington J.L.: Obiektowe bazy danych dla każdego; Mikom2000
6. Whitehorn M., Marklyn B.: Relacyjne bazy danych; Helion
7. Wrycza S., Marcinkowski B., Wyrzykowski K.: Język UML 2.0 w modelowaniu systemów informatycznych; Helion 2005