Artur GRAMACKI, Jarosław GRAMACKI
UNIWERSYTET ZIELONOGÓRSKI, INSTYTUT INFORMATYKI I ELEKTRONIKI

Modelowanie typu Entity-Attribute-Value w bazach danych

Dr inż. Artur GRAMACKI

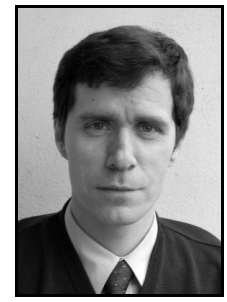
Pracuje w Instytucie Informatyki i Elektroniki Uniwersytetu Zielonogórskiego na stanowisku adiunkta. Interesuje się zagadnieniami związanymi z bazami danych, w szczególności firmy Oracle ale też rozwiązaniami Open Source. Oprócz prowadzenia zajęć dydaktycznych stara się wykorzystywać swoją wiedzę uczestnicząc w różnych projektach informatycznych z tego zakresu. Brał udział w czterech projektach, których celem było przygotowanie systemów wspomagających działalność UZ.

e-mail: a.gramacki@iie.uz.zgora.pl

Dr inż. Jarosław GRAMACKI

Pracuje w Instytucie Informatyki i Elektroniki Uniwersytetu Zielonogórskiego na stanowisku adiunkta. Jego zainteresowania koncentrują się wokół zagadnień związanych z bazami danych, hurtowniami oraz kopalniami danych. Pracuje głownie w środowisku bazy Oracle, jednak z dużą sympatią odnosi się do baz danych związanych z ruchem Open Source. Swoje doświadczenie stara się wykorzystywać zarówno w pracy dydaktycznej, jaki i aktywnie uczestnicząc w różnych projektach bazodanowych.

e-mail: j.gramacki@iie.uz.zgora.pl



Streszczenie

W artykule omówiono pewien specyficzny sposób reprezentacji danych, który w pewnych sytuacjach może być stosowany w miejsce klasycznego podejścia relacyjnego. Jest on stosunkowo mało znany i choć zakres jego stosowalności jest ograniczony a pewne niekorzystne właściwości znaczne, to warto go poznać, gdyż istnieją obszary, gdzie jego stosowanie może przynieść wymierne korzyści. Mowa tu o tzw. podejściu Entity-Attribute-Value, w skrócie EAV. Jest ono całkowicie różne od podejścia relacyjnego. W pracy pokazano główne założenia EAV wraz z przykładami oraz omówiono jego podstawowe zalety i wady. Krótko omówiono również pewne alternatywne rozwiązanie do EAV, które posiada niektóre cechy EAV, a jednocześnie nie burzy tak bardzo podejścia relacyjnego.

Słowa kluczowe: Entity-Attribute-Value, EAV, bazy relacyjne, generyczne struktury bazodanowe, projektowanie baz danych.

Entity-Attribute-Value approach in databases

Abstract

In the paper a specific kind of data representation is presented. In some situations it can be used in place of classical relational approach. This data representation, although not new, is not widely known, and in spite of some serious limitations, can be used in many specific database areas. We describe the Entity-Attribute-Value (EAV) approach. This approach is totally different to the relational one. In the paper we give an overview of the EAV as well as we show the main advantages and disadvantages. Also, we show an alternative approach to EAV which does not suffer the EAV limitations.

Keywords: Entity-Attribute-Value, EAV, relational databases, generic database structures, database design.

1. Wstęp

Stosunkowo często projektując model bazy danych (w większości przypadków relacyjny) nie jesteśmy w stanie przewidzieć dokładnej struktury przechowywanych w niej danych. Jako przykład można podać aplikację do obsługi sklepu internetowego z branży informatycznej, w którym każdy sprzedawany produkt opisywany jest specyficznym zestawem atrybutów. Ponieważ powstają coraz to nowe produkty, w momencie tworzenia struktury bazy danych nie jesteśmy w stanie przewidzieć, jakie dokładnie kategorie produktów będą w niej przechowywane i jakimi atrybutami będą one opisywane. Gdy struktura bazy danych utworzona jest według klasycznych wzorców relacyjnych, dodanie nowego rodzaju produktu zwykle wiąże się albo z dodaniem do schematu dodatkowej tabeli albo też z modyfikacją kolumn w istniejącej już tabeli. Zwykle w takich sytuacjach wymagane zmiany w działających na tym schemacie aplikacjach są spore i uciążliwe.

Inny przykład o podobnych „niedogodnościach" podany jest w pracy [1]. Tam problemem jest konieczność rejestrowania ogromnych ilości danych z czujników pomiarowych o bardzo dużej (trudnej do przewidzenia z góry) różnorodności.

W takich przypadkach warto rozważyć zastosowanie podanych w tytule pracy zbudowanych jako struktury konfigurowane tzw. metadanymi.

W literaturze nie jest on prawie wcale opisywany, autorzy natknęli się tylko na nieliczne źródła: [2, 3, 4]. Podejmowano również nieliczne próby jego bardziej formalnego opisu [5].

Model EAV jest to uniwersalna struktura, która zawiera oprócz danych merytorycznych (tak jak w klasycznej strukturze relacyjnej) dodatkowo również pewne dane opisujące strukturę samej bazy (czyli wspomniane metadane). Dzięki takiemu rozwiązaniu, gdy zajdzie konieczność wprowadzenia do bazy/aplikacji nowej kategorii obiektu, nie ma potrzeby modyfikowania aplikacji a jedynie wprowadzenie opisu nowego obiektu do metadanych. W ten sposób odpowiednio napisana aplikacja, bez konieczności wprowadzania do niej jakichkolwiek zmian, będzie w stanie automatycznie dostosowywać się do zmieniającej się struktury danych. Niestety model (struktura) EAV wprowadza pewne problemy, gdyż bardziej ogólny (czasami nazywany generycznym) model danych prowadzi do większej liczności wierszy w tabelach, bardziej złożonych zapytań SQL oraz trudności w zapewnieniu wymaganej integralności danych. Nie zawsze więc model EAV będzie możliwy do zastosowania. Zawsze jednak warto rozważyć potencjalne korzyści i straty i wybrać rozwiązanie lepsze.

2. Podstawowe założenia modelowania EAV

O klasycznej tabeli relacyjnej można powiedzieć, że przechowuje informacje w postaci zbioru rekordów, każdy o tej samej, ustalonej już na etapie projektowania, strukturze. Z kolei każdy rekord to zbiór pewnych atrybutów. Każdy atrybut reprezentowany jest przez dokładnie jedną kolumnę. Gdy z jakiegoś powodu zaistnieje konieczność zarejestrowania kolejnego atrybutu należy zmodyfikować tabelę dodając nową kolumnę.

W podejściu EAV mamy do czynienia z zupełnie innym założeniem. Dane przechowywane są również w tabelach (z reguły relacyjnych, bo one niepodzielnie królują we współczesnych systemach zarządzania bazami danych), jednak każdy wiersz przechowuje dokładnie jeden atrybut. Teoretycznie rzecz biorąc wszelkie dane mogą być przechowywane tylko w jednej tabeli zawierającej 3 kolumny: pierwsza kolumna zawiera identyfikator encji (ang. **E**ntity), druga nazwę atrybutu (ang. **A**ttribute), wreszcie trzecia zawiera wartość atrybutu (ang. **V**alue). Aby dodać do modelu nowy atrybut nie musimy modyfikować struktury tabeli tylko po prostu musimy wstawić do niej kolejny rekord. Na rysunkach 1 oraz 2 pokazano przykład konwencjonalnej tabeli relacyjnej oraz tabeli zaprojektowanej według koncepcji EAV.

PRAC_REL

prac_id	nazwisko	data_ur	zarobki	Stanowisko
1	Nowak	19-09-1975	2500	Kierownik
2	Kowalski	26-01-1967	1500	Magazynier

Rys. 1. Przykładowa konwencjonalna tabela relacyjna
Fig. 1. A sample of conventional relational table

PRAC_EAV

prac_id	atrybut	Wartość
1	Nazwisko	Nowak
1	Dat_ur	19-09-1975
1	Zarobki	2500
1	Stanowisko	Kierownik
2	Nazwisko	Kowalski
2	Dat_ur	26-01-1967
2	Zarobki	1500
2	Stanowisko	Magazynier

Rys. 2. Tabela w konwencji EAV
Fig. 2. EAV-type table

Już z tego prostego i najbardziej podstawowego przykładu można wyciągnąć pewne wnioski co do zalet i wad podejścia EAV:

Zalety:

- nie ma żadnych ograniczeń co do ilości rejestrowanych atrybutów. Każdy nowy atrybut to po prostu kolejny wiersz (wiersze) w tabeli. Dzięki temu nie ma potrzeby zmieniania aplikacji klienckich, gdy zmianie ulegnie model danych,
- w podejściu EAV w sposób naturalny likwidowane są problemy jakie mogą pojawić się przy pracy z tabelami rzadkimi (ang. *sparse*), czyli zawierającymi dużo wartości *NULL*,
- układ danych w tabeli *PRAC_EAV* jest niezwykle podobny do plików XML, co przy bardzo dużej popularności tego formatu może być bardzo interesujące. Obecnie większość współczesnych SZBD coraz mocniej wspiera ten format danych,
- z uwagi na bardziej jednorodną niż modele relacyjne strukturę, istnieje możliwość tworzenia narzędzi do automatycznego generowania aplikacji (formatki ekranowe i raporty).

Wady:

- dane przechowywane są w sposób trudniejszy do ogarnięcia przez człowieka. Brak kolumn-atrybutów utrudnia szybkie odczytanie np. danych o jednym wybranym pracowniku. Można jednak utworzyć odpowiedni widok (ang. *view*), który ukryje przed użytkownikiem tą niedogodność,
- typowe zapytania o pewien podzbiór przechowywanych danych (np. „wyświetl wszystkich pracowników zarabiających więcej niż 1000 zł i urodzonych po 1970 roku") są bardziej złożone niż ich relacyjne odpowiedniki, a tym samym zwiększa się czas ich wykonywania,
- tabela *PRAC_EAV* jest o wiele bardziej liczna niż jej relacyjny odpowiednik,
- integralność danych nie może być kontrolowana przez samą bazę danych, musi być niejako ręcznie implementowana przez użytkownika (np. z wykorzystaniem wyzwalaczy bazodanowych). Przykładowo nie da się zapewnić „systemowej" kontroli poprawności danych w polu *prac_id*. Podobnie nie da się na przykład utworzyć dla pola *nazwisko* odpowiednika ograniczenia *NOT NULL*. Nie ma zastosowania również pojęcie klucza głównego.

Model pokazany na rysunku 2 zwykle w praktyce będzie wymagał rozbudowy. Podstawową jego wadą jest to, że wartości wszystkich atrybutów są zapisywane z wykorzystaniem tego samego typu danych (zwykle musi to być jakiś typ łańcuchowy, np. *VARCHAR2* dla bazy Oracle). Nie mamy więc żadnej informacji o rzeczywistych typach danych przechowywanych w tabeli (a warto np. wiedzieć, że kolumna *zarobki* to typ numeryczny). Wobec tego naszą podstawową tabelę *PRAC_EAV* trzeba uzupełnić o przynajmniej jedną tabelą pomocniczą, pozwalającą zanotować typ danego atrybutu. Na rysunku 3 pokazano nowy model. Dodatkowa tabela przechowuje, jeżeli można tak powiedzieć, dane na temat danych, czyli tzw. metadane.

Bądźmy również realistami i przyjmijmy, że w rzeczywistości pewne atrybuty są oczywiste, znane na etapie projektowania oraz niezmienne w czasie całego „życia" naszej bazy. Nasz model przyjmie więc postać pokazaną na rysunku 4[1]. Tabele *PRAC_EAV* oraz *PRAC_METADATA* będą używane, gdy zajdzie potrzeba rozbudowania tabeli *PRACOWNICY* o pewne dodatkowe, nieznane na etapie projektowania, atrybuty. Mamy więc obecnie do czynienia z trzema rodzajami tabel (w kontekście spełnianej funkcji): klasyczną tabelę relacyjną, tabelę EAV oraz tabelę z metadanymi.

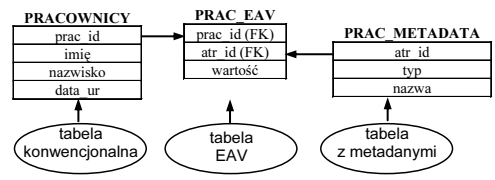
PRAC_EAV

prac_id	atr_id	wartość
1	1	Nowak
1	2	19-09-1975
1	3	2500
1	4	Kierownik
2	1	Kowalski
2	2	26-01-1967
2	3	1500
2	4	Magazynier

PRAC_METADATA

atr_id	typ	nazwa
1	varchar2	nazwisko
2	date	data_ur
3	numer	zarobki
4	integer	stanowisko

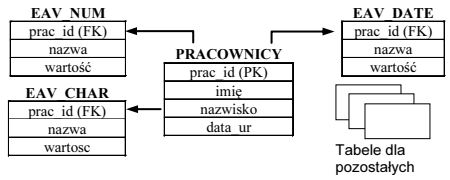
Rys. 3. Zmodyfikowany model EAV
Fig. 3. Modified EAV model



Rys. 4. Model hybrydowy: tabela relacyjna oraz EAV
Fig. 4. Hybrid model: relational and EAV tables

Zwróćmy również uwagę, że nadal kolumna *wartość* w tabeli *PRAC_EAV* musi być typu znakowego (np. *VARCHAR2*), aby mogła „pomieścić" wszelkie możliwe typy danych[2]. Kontrolę, czy wprowadzane dane są rzeczywiście zgodne z zapisanym w tabeli *PRAC_METADATA* typem musimy oczywiście wykonać ręcznie, np. odpowiednio oprogramowując wyzwalacze bazodanowe lub tworząc odpowiednie procedury walidacyjne na poziomie aplikacji użytkowej.

Można również postąpić inaczej i, kosztem kolejnej rozbudowy modelu, doprowadzić do sytuacji, że poszczególne rodzaje danych (np. liczbowe, znakowe, daty i czasu) przechowywane będą we właściwych typach kolumn. Dzięki temu można je na przykład dużo bardziej efektywnie zaindeksować. W nowym rozwiązaniu na przechowywanie danych każdego typu zarezerwowana jest oddzielna tabela. Pokazano to na rysunku 5.



Rys. 5. Modyfikacja modelu z rysunku 4
Fig. 5. Modification of the model from figure 4

3. Zapytania SQL na modelu EAV

Podejście EAV będzie nastręczać problemy w tworzeniu zapytań SQL. Język SQL bowiem był (i jest nadal) tworzony do kon-

[1] Aby ograniczyć ilość miejsca zajmowanego przez rysunek nie zamieszczamy już przykładowych danych z tabel a tylko ich strukturę

[2] Zakładamy, że nasza baza nie potrzebuje mieć możliwości przechowywać danych binarnych lub wielkich tekstowych (w bazie Oracle np. typy BLOB, CLOB, TEXT itp.). Gdyby była taka konieczność nasza struktura musiałaby ponownie zostać rozbudowana

struowania zapytań dla modelu relacyjnego. Wszelkie więc próby obejścia tego założenia muszą wiązać się z nieuniknionymi negatywnymi konsekwencjami. Rozważmy następujące polecenia SQL (pierwsze dla tabeli relacyjnej, drugi dla tabeli EAV):

```
SELECT prac_id FROM prac_rel WHERE zarobki > 1500;   --1a
SELECT prac_id FROM prac_eav                         --1b
WHERE atrybut LIKE 'zarobki' AND wartosc > 1500;
```

Inny przykład niech będzie taki jak poniżej:

```
SELECT prac_id, nazwisko FROM prac_rel               --2a
WHERE nazwisko LIKE 'N%';
SELECT prac_id, wartosc NAZWISKO FROM prac_eav       --2b
WHERE atrybut LIKE 'nazwisko' AND wartosc LIKE 'N%';
```

Nie wyciągajmy jednak zbyt pochopnie wniosku, że zapytania dla tabel relacyjnych oraz dla tabel EAV wyglądają podobnie. Tu mamy do czynienia ze szczególnym przypadkiem odwołania się tylko do kolumny „Entity" (*prac_id*) oraz tylko do jednej innej kolumny (*nazwisko*). Następny przykład pokazuje, że sprawy bardzo szybko się komplikują. Rozważmy mianowicie przypadek, gdy chcemy wyświetlić dwie kolumny. Zapytanie dla modelu relacyjnego będzie bardzo proste. Dla modelu EAV, aby otrzymać prawidłowy wynik trzeba tabelę bazową potraktować jako dwie „wirtualne" tabele. Pojawiają się więc złączenia typu *self-joins*:

```
SELECT prac_id, nazwisko, zarobki FROM prac_rel      --3a
WHERE nazwisko LIKE 'N%';
SELECT p1.prac_id, p1.wartosc, p2.wartosc            --3b
FROM prac_eav p1, prac_eav p2
WHERE p1.prac_id = p2.prac_id    AND
      p1.atrybut LIKE 'nazwisko' AND
      p2.atrybut LIKE 'zarobki'  AND
      p1.wartosc LIKE 'N%';
```

Gdy w zapytaniu podamy warunek oparty o kolejną, trzecią kolumnę musimy wprowadzić kolejną tabelę wirtualną:

```
SELECT prac_id, nazwisko, zarobki FROM prac_rel      --4a
WHERE stanowisko LIKE 'Kierownik';
SELECT p1.prac_id, p1.wartosc, p2.wartosc            --4b
FROM prac_eav p1, prac_eav p2, prac_eav p3
WHERE p1.prac_id = p2.prac_id      AND
      p2.prac_id = p3.prac_id      AND
      p1.atrybut LIKE 'nazwisko'   AND
      p2.atrybut LIKE 'zarobki'    AND
      p3.atrybut LIKE 'stanowisko' AND
      p3.wartosc LIKE 'Kierownik';
```

W przypadku, gdy korzystamy z bardziej realistycznego modelu pokazanego na rysunku 3, a jeszcze lepiej 4 lub 5, zapytania będą jeszcze bardziej złożone. Poniżej pokazano wyniki działania poszczególnych poleceń pod kątem ich efektywności.

Tab. 1. Statystyki badanych pleceń SQL
Tab. 1. Statistics for SQL statements

Nr	count	cpu	elapsed	disk	query	current	rows
1a	632	0.04	0.04	0	692	0	9435
1b	632	0.14	0.07	0	810	0	9435
2a	32	0.00	0.00	0	97	0	429
2b	32	0.02	0.01	0	213	0	429
3a	32	0.02	0.00	0	97	0	429
3b	32	0.05	0.03	0	397	0	429
4a	55	0.00	0.00	0	120	0	770
4b	55	0.06	0.06	0	604	0	770

Wykorzystano bazę Oracle, tabela *PRAC_REL* zawierała 10000 przykładowych rekordów (tabela EAV w tym przypadku zawiera oczywiście 40000 rekordów). Statystyki zebrano za pomocą narzędzia *SQL Trace* oraz programu *tkprof*. Z braku miejsca zamieszczono tylko sumaryczne wielkości (*total*) z pominięciem rozbicia na fazy *Parse*, *Execute* i *Fetach*. Na tabelach nie zakładano żadnych indeksów ani też w żaden inny sposób ich nie optymalizowano. Wyniku uzyskane dla zapytań EAV są tylko nieznacznie gorsze od analogicznych zapytań relacyjnych. Jednak nie należy tych wyników uogólniać. Być może bardziej złożone modele z większą ilość danych będą wymagały dostrojenia.

4. Pewne rozwiązanie alternatywne

Gdy uznamy, że podejście EAV wydaje się zbyt ryzykowne, a mimo wszystko trudno jest w sposób jednoznaczny podać jakie tabele będą musiały istnieć w docelowej bazie oraz jakie będą zawierały atrybuty można rozważyć inne rozwiązanie. Można mianowicie dodać do tabeli pewne nadmiarowe kolumny, które będą wykorzystywane „gdy zajdzie taka potrzeba". Nasza tabela może więc wyglądać tak, jak ją pokazano na rysunku 6.

PRACOWNICY

prac_id	imię	nazwisko	dat_ur	pole_1	pole_2	…	poleN

Rys. 6. Alternatywne podejścia do EAV
Fig. 6. Alternative approach to EAV

Dodatkowe kolumny najsensowniej jest utworzyć jako znakowe o jak największej wielkości (aby pomieściły wszystkie potencjalne typy danych). Aby poprawić czytelność można również utworzyć stosowny widok. Można również utworzyć tabelę podobną do EAV z rysunku 4, aby mieć dane potrzebne do kontrolowania, czy do pól *pole_xxx* wstawiane są poprawne (w kontekście typów danych) wartości. Definicja widoku może być przykładowo taka jak pokazano poniżej:

```
CREATE VIEW pracownicy_v AS SELECT
  prac_id, nazwisko, data_ur, zarobki, to_number(pole_1)
  pesel, to_date( pole_2, 'dd-mm-yyyy' ) data_zatr
FROM pracownicy;
```

5. Podsumowanie

W artykule zaprezentowano podstawowe założenia tzw. podejścia Entity-Attribute-Value w projektowaniu baz danych. Warto je stosować tylko wówczas, gdy wiemy z całą pewnością, że definicje tabel (też dodawanie nowych) będą bardzo często się zmieniały. Innymi słowy nie znamy dokładnie na etapie projektowania jakie będą w docelowej bazie tabele oraz jakie będą one posiadały atrybuty. Jak pokazano w artykule EAV ma sporo wad, jednak jest możliwe przynajmniej częściowe zniwelowanie ich niekorzystnego działania. Ponieważ współczesne bazy danych są projektowane pod kątem wspierania teorii relacyjnej, więc podejście EAV (które niejako ignoruje tą teorię) musi siłą rzeczy dawać gorsze wyniki i niemal zawsze będzie mniej efektywne niż odpowiedniki relacyjne. Każdorazowa decyzja o zastosowaniu EAV w konkretnym projekcie musi być poprzedzona dogłębną analizą i testami praktycznymi, które pozwolą odpowiedzieć na pytanie, czy EAV ma szansę poprawić finalny produkt, czy też będzie tylko niepotrzebnie go komplikował. Może się bowiem okazać, że obiektywnie gorsze parametry dla modelu EAV (np. dłuższe czasy wykonywania zapytań) są jednak w sumie akceptowalne przez użytkownika.

6. Literatura

[1] Traczyk T.: Rozproszona heterogeniczna baza danych wspierająca budowę wielkiego eksperymentu fizyki wysokich energii, Warszawa, Wydawnictwa Komunikacji i Łączności, 2005, ISBN: 83-206-1572-0, 127-134

[2] Eggebraaten T. J., Tenner J. W., Dubbels J. C.: A health-care data model based on the HL7 Reference Information Model, IBM Systems Journal 46(1), 2007, 5-18

[3] Anhøj J.: Generic Design of Web-Based Clinical Databases, Journal of Medical Internet Resources, 2003, 5(4) (www.jmir.org)

[4] Nadkarni P. M.: An Introduction to EAV systems, (ycmi.med.yale.edu)

[5] Geppert A., Dittrich K. R.: EAV: An Algebra for the Representation and Realization of Internal, Conceptual Objects in Configurable Database Management Systems, Scientific Literature Digital Library, (http://citeseer.ist.psu.edu/geppert93eav.html), 1993

Artykuł recenzowany