# Bazy danych

# 3. Normalizacja baz danych

P. F. Góra

http://th-www.if.uj.edu.pl/zfs/gora/

2011/12

## Pierwsza postać normalna

Tabela jest w pierwszej postaci normalnej (1PN), jeżeli

1. Tabela posiada klucz.
2. Wszystkie składowe krotek są atomowe.

Można powiedzieć, że pierwsza postać normalna jest warunkiem tego, żeby w ogóle można było mówić o sytemie *relacyjnym*. Warunek posiadania klucza jest równoważny temu, że *tabela jest zbiorem krotek*.

## Atomowość danych

Atomowość danych oznacza, że składowych krotek nie można podzielić. Warunek atomowości uniemożliwia to, żeby składowymi krotek były złożone struktury danych, takie jak tablice, listy itp.

W zasadzie wymóg atomowości nakazuje dzielić też atrybuty, które *można podzielić*, na części logicznie niepodzielne. Na przykład zamiast atrybutu "`Imię i Nazwisko`", powinniśmy mieć dwa atrybuty: "`Imię`", "`Nazwisko`". Można sobie jednak wyobrazić sytuacje, w których taki podział byłby niepotrzebny lub niewskazany*. O ile zatem pierwsza postać normalna z całą pewnością wyklucza złożone struktury danych, o tyle interpretacja pojęcia "można podzielić" może niekiedy zależeć od natury samych danych, które reprezentować ma konstruowana przez nas baza danych.

*Rozważmy na przykład nazwy osobowe z Chin czy Korei.

## Anomalie baz danych

- Redundancja — ta sama informacja jest niepotrzebnie przechowywana w kilku krotkach.
- Anomalia modyfikacji — informacja zostanie zmodyfikowana w pewnych krotkach, a w innych nie. Która informacja jest wówczas prawdziwa?
- Anomalia usuwania — usuwanie części informacji powoduje utratę innej informacji, której nie chcielibyśmy stracić.
- Anomalia dołączania — wprowadzenie pewnej informacji jest możliwe tylko wtedy, gdy jednocześnie wprowadzamy jakąś inną informację, która może być obecnie niedostępna.

Celem normalizacji baz danych jest unikanie powyższych anomalii.

## Bezstratne złączenie

Normalizację baz danych (powyżej 1PN) przeprowadza się dzieląc tabele "wertykalnie" na tabele potomne. Tabele te jednak **muszą** pozwalać na pełne odtworzenie wyjściowej informacji po dokonaniu złączenia. Niedopuszczalne jest także, aby złączenia kreowały informację fałszywą.

Dekompozycję tabeli $R$ na tabele $R_1$, $R_2$, ..., $R_n$ nazywamy dekompozycją bezstratnego złączenia (ze względu na pewien zbiór zależności funkcyjnych), jeśli naturalne złączenie $R_1$, $R_2$, ..., $R_n$ jest równe tabeli $R$.

Dekompozycja tabeli $R$ na dwie tabele $R_1$, $R_2$ jest dekompozycją bezstratnego złączenia, jeśli spełniony jest jeden z dwu warunków (symbol $\rightarrow$ oznacza zależność funkcyjną):

$$(R_1 \cap R_2) \rightarrow (R_1 \backslash R_2)$$

lub

$$(R_1 \cap R_2) \rightarrow (R_2 \backslash R_1)$$

Innymi słowy, wspólna część atrybutów $R_1$, $R_2$ musi zawierać klucz kandydujący $R_1$ lub $R_2$.

## Przykład negatywny

Przypuśćmy, że w pewnej tabeli przechowujemy informację na temat studentów zapisujących się na kurs baz danych. Przykładowa instancja tabeli mogłaby wyglądać tak:

$T_0$

| NrStudenta | NrKursu | DataZapisu | Sala | Prowadzący |
|---|---|---|---|---|
| 1010057 | K202 | 01.10.2011 | A1 | Góra |
| 1010132 | K203 | 01.10.2011 | A2 | Łachwa |
| 1015678 | K202 | 01.10.2011 | A2 | Łachwa |
| 1020159 | K204 | 04.10.2011 | A1 | Góra |
| 1026789 | K205 | 11.10.2011 | C2 | Oramus |

Rozbijamy tę tabele na dwie, dzieląc je “wzdłuż” kolumny `DataZapisu`.

$T_1$

| NrStudenta | NrKursu | DataZapisu |
|---|---|---|
| 1010057 | K202 | 01.10.201 |
| 1010132 | K203 | 01.10.2011 |
| 1015678 | K202 | 01.10.2011 |
| 1020159 | K204 | 04.10.2011 |
| 1026789 | K205 | 11.10.2011 |

$T_2$

| DataZapisu | Sala | Prowadzący |
|---|---|---|
| 01.10.2011 | A1 | Góra |
| 01.10.2011 | A2 | Łachwa |
| 01.10.2011 | A2 | Łachwa |
| 04.10.2011 | A1 | Góra |
| 11.10.2011 | C2 | Oramus |

Dokonujemy złączenia tabel $T_1$ i $T_2$. Okazuje się, że dzielenie pierwotnej tabeli "wzdłuż" kolumny `DataZapisu` to był kiepski pomysł. Otrzymujemy

$T_1 \bowtie T_2 \neq T_0$

| NrStudenta | NrKursu | DataZapisu | Sala | Prowadzący |
|---|---|---|---|---|
| 1010057 | K202 | 01.10.2011 | A1 | Góra |
| 1010132 | K203 | 01.10.2011 | A1 | Góra |
| 1015678 | K202 | 01.10.2011 | A1 | Góra |
| itd | | | | |

Zaznaczone krotki nie występują w pierwotnej tabeli. Na skutek podziału tabeli nie spełniającego warunku bezstratnego złączenia, utraciliśmy informację o tym, kto kogo uczy.

## Druga postać normalna

Tabela jest w drugiej postaci normalnej (2PN), jeżeli

1. Tabela jest 1PN.
2. Wszystkie atrybuty niekluczowe zależą funkcyjnie od pełnego klucza.

Atrybuty niekluczowe mają zależeć od *pełnego* klucza, a nie od jego podzbioru właściwego (który nie musi być kluczem!).

Wszystkie tabele 1PN, które mają klucze jednokolumnowe, są automatycznie 2PN.

## Przykład

Załóżmy, że zależności funkcyjne pomiędzy pewnymi atrybutami mają postać $A, B \to C$, $A \to D$. Poniższa tabela (podkreślenia oznaczają klucz)

| $\underline{A}$ | $\underline{B}$ | $C$ | $D$ |
|---|---|---|---|
| $a_1$ | $b_1$ | $c_1$ | $d_1$ |
| $a_1$ | $b_2$ | $c_2$ | $d_1$ |
| $a_1$ | $b_3$ | $c_3$ | $d_1$ |
| $a_2$ | $b_1$ | $c_4$ | $d_2$ |

**nie** jest 2PN, gdyż atrybut $D$ zależy *tylko* od atrybutu $A$, a więc od części klucza, nie od całego klucza. Zauważmy jednocześnie, że wartość $d_1$ przechowywana jest niepotrzebnie w trzech różnych krotkach.

Po rozbiciu powyższej tabeli na dwie tabele będące w 2PN, wartość $d_1$ przechowywana jest tylko w *jednej* krotce.

| $\underline{A}$ | $\underline{B}$ | $C$ |
|---|---|---|
| $a_1$ | $b_1$ | $c_1$ |
| $a_1$ | $b_2$ | $c_2$ |
| $a_1$ | $b_3$ | $c_3$ |
| $a_2$ | $b_1$ | $c_4$ |

| $\underline{A}$ | $D$ |
|---|---|
| $a_1$ | $d_1$ |
| $a_2$ | $d_2$ |

Proszę pomyśleć, że w początkowym przykładzie $d_1$ mogłoby występować nie w 3, ale w 300 lub w 3000 krotek.

## Trzecia postać normalna

Tabela jest w trzeciej postaci normalnej (3PN), jeżeli

1. Tabela jest 2PN.
2. Dla wszystkich atrybutów tabeli zachodzi:
   Jeżeli $A_1, A_2, \ldots, A_n \rightarrow A_m$, to albo $\{A_1, A_2, \ldots, A_n\}$ jest nadkluczem, albo $A_m$ jest elementem innego klucza.

Trzecia postać normalna jest postacią najczęściej występującą w zastosowaniach praktycznych. Druga część warunku definicyjnego ("albo $A_m$ jest elementem innego klucza") ma znaczenie tylko wówczas, gdy w tabeli występują zależności cykliczne (lub częściowe zależności cykliczne).

## Zależności przechodnie

Jeżeli w tabeli nie występują zależności cykliczne, powiada się, że 3PN zakazuje wsytępowania zależności przechodnich. Istotnie, przyjmijmy, że spełnione są zależności funkcyjne $A \to B$, $B \to C$, $A \to C$; ostatnia z tych zależności wynika z dwu pierwszych na zasadzie przechodniości. Następująca tabela

| $\underline{A}$ | $B$ | $C$ |
|---|---|---|
| | | |

jest 2PN, ale **nie** jest 3PN, gdyż zachodzi zależność $B \to C$, zaś atrybut $B$ nie jest nadkluczem. Sprowadzenie do 3PN oznacza rozbicie powyższej tabeli na dwie tabele:

| $\underline{A}$ | $B$ |
|---|---|
| | |

| $\underline{B}$ | $C$ |
|---|---|
| | |

## Przykład

Niech zależności funkcyjne będą takie, jak na poprzednim ekranie. Rozważmy następującą instancję pierwszej tabeli:

| $\underline{A}$ | $B$ | $C$ |
|---|---|---|
| $a_1$ | $b_1$ | $c_1$ |
| $a_2$ | $b_1$ | $c_1$ |
| $a_3$ | $b_1$ | $c_1$ |
| $a_4$ | $b_2$ | $c_2$ |

Wielkość $c_1$ przechowywana jest w trzech krotkach. Po sprowadzeniu do 3PN

| $\underline{A}$ | $B$ |
|---|---|
| $a_1$ | $b_1$ |
| $a_2$ | $b_1$ |
| $a_3$ | $b_1$ |
| $a_4$ | $b_2$ |

| $\underline{B}$ | $C$ |
|---|---|
| $b_1$ | $c_1$ |
| $b_2$ | $c_2$ |

wartość $c_1$ przechowywana jest tylko w jednej krotce.

## Cykliczne zależności funkcyjne

Przykładem cyklicznych zależności funkcyjnych jest $A \to B, B \to C, C \to A$. W takiej sytuacji atrybuty $A, B, C$ są sobie równoważne — określenie wartości jednego z nich, jednoznacznie ustala wartość dwu pozostałych. **Każda** z trzech tabel

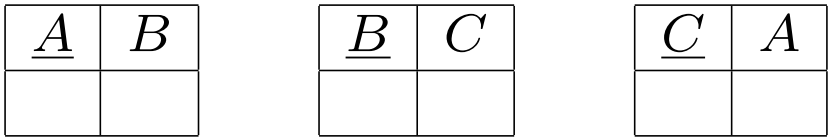
| $\underline{A}$ | $B$ | $C$ |
|---|---|---|
| | | |

| $A$ | $\underline{B}$ | $C$ |
|---|---|---|
| | | |

| $A$ | $B$ | $\underline{C}$ |
|---|---|---|
| | | |

jest 3PN, gdyż co prawda występują zależności przechodnie, ale atrybuty niekluczowe są elementami *innych kluczy* (de facto *są* innymi kluczami).

Uwaga! Przy cyklicznych zależnościach funkcyjnych jak poprzednio, tabele rozbite w taki sposób:




technicznie rzecz biorąc także są 3PN, a nie zawierają zależności przechodnich. Jednak taki projekt nie zapobiega redundancji, przeciwnie, wymusza ją, gdyż każda wartość każdego z atrybutów $A, B, C$ jest przechowywana dwa razy. Projekty z poprzedniej strony są z tego względu zdecydowanie lepsze.

## Procedura postępowania

1. Dany jest zbiór atrybutów, które chcemy reprezentować, i zbiór zależności funkcyjnych pomiędzy atrybutami.
2. Znajdujemy bazę minimalną zbioru zależności funkcyjnych.
3. Mając bazę minimalną, sumujemy zależności funkcyjne o takich samych lewych stronach i dla każdej wysumowanej zależności tworzymy tabelę z odpowiednią lewą stroną zależności jako kluczem.

Taki sposób postępowania prowadzi do projektu bazy, w której tabele są 3PN.

## Procedura szukania bazy minimalnej

1. Każdy atrybut musi występować z lewej lub z prawej strony jednej zależności funkcyjnej w zbiorze.
2. Jeśli jakaś zależność funkcyjna jest włączona do zbioru, nie wszystkie zależności funkcyjne potrzebne do jej wyprowadzenia mogą występować w tym zbiorze.
3. Jeśli jakaś zależność funkcyjna zostaje wyłączona ze zbioru, zależności funkcyjne potrzebne do jej wyprowadzenia muszą zostać doń dołączone.

Jeżeli w zbiorze zależności funkcyjnych występują (częściowe) cykle, może istnieć więcej niż jedna baza minimalna.

## Postać normalna Boyce'a-Codda

Tabela jest w postaci normalnej Boyce'a-Codda (BCNF, PNBC), jeżeli

1. Tabela jest 2PN.
2. Dla każdej zależności nietrywialnej, jeżeli $A_1, A_2, \ldots, A_n \to A_m$, to zbiór $\{A_1, A_2, \ldots, A_n\}$ jest nadkluczem.

PNBC jest "silniejszą" wersją 3PN. Każda tabela będąca PNBC jest także 3PN, ale wynikanie odwrotne nie musi zachodzić.

## Do czego dążymy?

Przeprowadzając normalizację bazy danych, staramy się **jednocześnie** spełnić trzy warunki:

1. Bezstratne złączenie.
2. Zachowanie wszystkich zależności funkcyjnych.
3. PNBC.

Czy zawsze jest to możliwe?

Rozważmy tabelę

| $A$ | $B$ | $C$ |
|---|---|---|
| | | |

z następującymi zależnościami funkcyjnymi:

$$C \to A$$
$$A, B \to C$$

(Na przykład: $A$ — nazwa banku, $B$ — nazwisko klienta uprawnionego do *personal banking*, $C$ — nazwisko bankiera. Pierwsza zależność mówi, że każdy bankier pracuje w określonym banku, druga, że każdego uprawnionego klienta w danym banku obsługuje określony bankier. Ale klient może mieć konta w więcej niż jednym banku. . . )

Powyższa tabela nie jest PNBC, gdyż $C$ nie może być nadkluczem. Jednak żadne rozbicie na dwie tabele dwuatrybutowe albo nie zachowuje kompletu zależności funkcyjnych, albo nie jest dekompozycją bezstratnego złączenia, albo jedno i drugie. W tej sytuacji zadowalamy się niższą postacią normalną. Zachowanie kompletu zależności funkcyjnych oraz bezstratność złączeń muszą mieć priorytet! Nie każdą tabelę daje się znormalizować do PNBC.

## Normalizacja a wydajność

Normalizacja baz danych dostarcza **mechanizmu** pozwalającego unikać anomalii. Ma to jednak swoją cenę: Dostęp do danych w bazie znormalizowanej może być wolniejszy, gdyż RDBMS musi wykonywać złączenia.

Dlatego w wielkich bazach danych zoptymalizowanych na odczyt (na przykład w hurtowniach danych) często rezygnuje się z wyższych postaci normalnych, przechowując dane w tabelach 1PN. *To* także ma swoją cenę: Wprowadzając dane do takich tabel lub modyfikując istniejące dane należy dołożyć szczególnej staranności, aby nie dopuścić do anomalii usuwania lub dołączania, a szczególnie do anomalii modyfikacji (redundancja jest w tego typu bazy niejako wbudowana).