Normalizacja do 3NF i świadoma denormalizacja

> Schemat złożony z tabel `klienci`, `zamowienia`, `pozycje` i `produkty`, na którym opierają się przykłady w całej tej książce, nie powstał z przypadku — jest wynikiem tego samego procesu, który ta strona przechodzi krok po kroku, zaczynając od jednej, płaskiej tabeli pełnej powtórzeń.

Skróty użyte na tej stronie (1)

Skrót	Co znaczy
SQL	Structured Query Language: język zapytań do relacyjnych baz danych

Po co to istnieje

> **Pojęcie: normalizacja.** Proces przekształcania struktury tabel tak, żeby każdy pojedynczy fakt był zapisany w bazie dokładnie w jednym miejscu, a nie powtarzany w wielu wierszach naraz. Punktem wyjścia jest zwykle jedna szeroka, płaska tabela, do której ktoś wrzucił wszystko naraz — dane zamówienia obok danych klienta obok listy zamówionych produktów. Taka tabela działa, dopóki nikt niczego w niej nie zmienia; problemy zaczynają się przy pierwszym `INSERT`, `UPDATE` czy `DELETE`, i to właśnie te problemy normalizacja eliminuje.

Zaczynamy od takiej właśnie płaskiej, niezormalizowanej tabeli:

```
 zamowienie_id | klient_nazwa    | klient_miasto | produkty                                  |
data       | status
---------------+-----------------+---------------+-------------------------------------------
+------------+-------------
 101           | Nowak i Wsp.    | Warszawa      | Monitor 24 cale x1, Kabel HDMI 2m x2      |
2024-01-15 | zrealizowane
 102           | Nowak i Wsp.    | Warszawa      | Klawiatura mechaniczna x1                 |
```

```
2024-02-03 | zrealizowane
 104            | Zielona Dolina  | Wrocław       | Biurko regulowane x1, Fotel biurowy x1   |
2024-03-10 | w realizacji
```

> **Pojęcie: anomalia wstawiania.** Sytuacja, w której nie da się zapisać pewnego faktu bez jednoczesnego zapisania innego, niezwiązanego z nim faktu, bo oba są zmieszane w jednej tabeli. W tabeli wyżej nie da się zapisać nowego klienta, który jeszcze nie złożył żadnego zamówienia — kolumny `klient_nazwa` i `klient_miasto` istnieją wyłącznie jako część wiersza zamówienia, więc bez zamówienia nie ma gdzie zapisać samego klienta.

> **Pojęcie: anomalia aktualizacji.** Sytuacja, w której zmiana jednego faktu wymaga znalezienia i poprawienia go we wszystkich miejscach, w których został powtórzony, z ryzykiem, że któreś z nich zostanie pominięte. Nazwa `Nowak i Wsp.` w tabeli wyżej powtarza się w dwóch wierszach — zmiana nazwy tego klienta na `Nowak i Wspólnicy Sp. z o.o.` wymaga zaktualizowania obu wierszy, i nic w strukturze tabeli nie gwarantuje, że ktoś nie zaktualizuje tylko jednego z nich, zostawiając bazę w stanie sprzecznym samą ze sobą.

> **Pojęcie: anomalia usuwania.** Sytuacja, w której usunięcie jednego faktu przypadkiem kasuje też inny fakt, logicznie od niego niezależny. Usunięcie jedynego zamówienia klienta `Zielona Dolina` (wiersz 104) kasuje jednocześnie jedyny zapis mówiący, że taki klient w ogóle istnieje i że mieszka we Wrocławiu — informacja o kliencie ginie razem z informacją o zamówieniu, mimo że to dwa niezależne fakty.

Jak to działa: 1NF, 2NF, 3NF krok po kroku

Rozwiń: jak to działa pod spodem

> **Pojęcie: pierwsza postać normalna (1NF).** Wymóg, żeby każda kolumna w każdym wierszu przechowywała dokładnie jedną, niepodzielną wartość — bez list, bez grup powtarzalnych upakowanych w jedno pole. Kolumna `produkty` w tabeli wyżej łamie ten wymóg: `Monitor 24 cale x1, Kabel HDMI 2m x2` to w rzeczywistości dwie różne pozycje sklejone w jeden tekst, którego baza nie potrafi ani przefiltrować po pojedynczym

produkcie, ani zsumować bez wcześniejszego rozbicia w aplikacji.

Doprowadzenie do 1NF rozbija tę listę na osobne wiersze, po jednym na pozycję zamówienia:

```
 zamowienie_id | klient_nazwa    | klient_miasto | produkt_nazwa           | ilosc |
data       | status
----------------+-----------------+---------------+-------------------------+-------
+------------+-------------
 101            | Nowak i Wsp.    | Warszawa      | Monitor 24 cale         |     1 |
2024-01-15 | zrealizowane
 101            | Nowak i Wsp.    | Warszawa      | Kabel HDMI 2m           |     2 |
2024-01-15 | zrealizowane
 102            | Nowak i Wsp.    | Warszawa      | Klawiatura mechaniczna  |     1 |
2024-02-03 | zrealizowane
 104            | Zielona Dolina  | Wrocław       | Biurko regulowane       |     1 |
2024-03-10 | w realizacji
 104            | Zielona Dolina  | Wrocław       | Fotel biurowy           |     1 |
2024-03-10 | w realizacji
```

Każda kolumna ma teraz jedną, niepodzielną wartość, ale powtórzenie danych klienta i zamówienia jest gorsze niż wcześniej — dane te powtarzają się teraz w każdym wierszu pozycji, nie tylko w każdym wierszu zamówienia.

Pojęcie: druga postać normalna (2NF). Wymóg, żeby każda kolumna niebędąca częścią klucza zależała od całego klucza głównego, a nie tylko od jego fragmentu — dotyczy tabel z kluczem złożonym z kilku kolumn. Kluczem naturalnym tabeli powyżej jest para `(zamowienie_id, produkt_nazwa)`, bo dopiero ta para jednoznacznie wskazuje jeden wiersz. Kolumny `klient_nazwa`, `klient_miasto`, `data` i `status` zależą wyłącznie od `zamowienie_id` — nie zmieniają się między różnymi produktami tego samego zamówienia — czyli zależą tylko od części klucza, nie od całego. To jest zależność częściowa, którą 2NF eliminuje.

Doprowadzenie do 2NF wydziela dane zależne tylko od `zamowienie_id` do osobnej tabeli, zostawiając w tabeli pozycji tylko to, co faktycznie zależy od pełnego klucza `(zamowienie_id, produkt_nazwa)`:

```
tabela zamowienia:
 zamowienie_id | klient_nazwa    | klient_miasto | data       | status
----------------+-----------------+---------------+------------+-------------
```

```
 101            | Nowak i Wsp.    | Warszawa      | 2024-01-15 | zrealizowane
 102            | Nowak i Wsp.    | Warszawa      | 2024-02-03 | zrealizowane
 104            | Zielona Dolina  | Wrocław       | 2024-03-10 | w realizacji

tabela pozycje:
 zamowienie_id | produkt_nazwa           | ilosc
---------------+-------------------------+-------
 101            | Monitor 24 cale         |     1
 101            | Kabel HDMI 2m           |     2
 102            | Klawiatura mechaniczna  |     1
 104            | Biurko regulowane       |     1
 104            | Fotel biurowy           |     1
```

Anomalia wstawiania nowego produktu do zamówienia bez wpływu na dane klienta jest już rozwiązana, ale `klient_nazwa` i `klient_miasto` wciąż powtarzają się w tabeli `zamowienia` za każdym razem, gdy ten sam klient złoży kolejne zamówienie — `Nowak i Wsp.` i `Warszawa` widnieją w dwóch wierszach, dokładnie ten sam problem co na początku, tylko przeniesiony do mniejszej tabeli.

> **Pojęcie: trzecia postać normalna (3NF).** Wymóg, żeby żadna kolumna niebędąca kluczem nie zależała od innej kolumny niebędącej kluczem — tylko bezpośrednio od klucza głównego. Zależność taka, przechodząca przez pośrednią kolumnę, nazywa się zależnością przechodnią. W tabeli `zamowienia` powyżej `klient_miasto` zależy od `klient_nazwa` (ten sam klient zawsze mieszka w tym samym mieście), a `klient_nazwa` zależy od `zamowienie_id` — czyli `klient_miasto` zależy od `zamowienie_id` tylko pośrednio, przez `klient_nazwa`. To jest właśnie zależność przechodnia, którą 3NF eliminuje.

Doprowadzenie do 3NF wydziela dane klienta do osobnej tabeli, zastępując powtórzoną nazwę i miasto pojedynczym odsyłaczem — kluczem obcym:

```
tabela klienci:
 id | nazwa           | miasto
----+-----------------+-----------
  1 | Nowak i Wsp.    | Warszawa
  3 | Zielona Dolina  | Wrocław

tabela zamowienia:
 id  | klient_id | data       | status
-----+-----------+------------+--------------
```

```
 101 |          1 | 2024-01-15 | zrealizowane
 102 |          1 | 2024-02-03 | zrealizowane
 104 |          3 | 2024-03-10 | w realizacji

tabela pozycje:
 zamowienie_id | produkt_id | ilosc
----------------+------------+-------
 101            |          1 |     1
 101            |          5 |     2
 102            |          2 |     1
 104            |          3 |     1
```

To jest dokładnie schemat `klienci` / `zamowienia` / `pozycje` / `produkty`, na którym opiera się reszta tej książki — anomalia aktualizacji znika, bo nazwa klienta istnieje w bazie dokładnie w jednym wierszu tabeli `klienci`, niezależnie od tego, ile ma zamówień; anomalia usuwania znika, bo usunięcie ostatniego zamówienia klienta nie rusza jego wiersza w `klienci`; anomalia wstawiania znika, bo nowego klienta da się zapisać, zanim złoży pierwsze zamówienie.

BCNF i wyższe postacie normalne

> **Pojęcie: postać normalna Boyce'a-Codda (BCNF).** Wersja 3NF domknięta ściślej — wymaga, żeby dla każdej zależności funkcyjnej w tabeli kolumna po lewej stronie zależności była kandydatem na klucz, nie tylko żeby kolumny niebędące kluczem zależały bezpośrednio od klucza. Różnica ujawnia się dopiero w tabelach z kilkoma nakładającymi się na siebie kandydującymi kluczami — rzadki przypadek w praktyce projektowej, ale teoretycznie 3NF dopuszcza wtedy anomalie, których BCNF już nie dopuszcza. Powyżej BCNF standard SQL i teoria baz danych wymieniają jeszcze czwartą i piątą postać normalną, zajmujące się zależnościami wielowartościowymi i zależnościami złączeniowymi — w codziennej pracy projektowej nad schematem transakcyjnym rzadko trzeba sięgać dalej niż 3NF, żeby uzyskać schemat wolny od praktycznych anomalii.

Denormalizacja

> **Pojęcie: denormalizacja.** Świadome cofnięcie części normalizacji — celowe wprowadzenie z powrotem redundancji, którą normalizacja usunęła, w zamian za szybszy odczyt kosztem trudniejszego utrzymania spójności. Denormalizacja nie jest błędem projektowym ani zapomnieniem o normalizacji — jest decyzją podjętą świadomie, po tym, jak w pełni znormalizowany schemat już istnieje, zwykle w odpowiedzi na konkretny, zmierzony problem z wydajnością odczytu.

Typowe, świadome przypadki denormalizacji obejmują kolumnę wyliczaną, przechowującą gotowy wynik obliczenia, żeby nie liczyć go przy każdym odczycie — na przykład `wartosc_brutto` obok `wartosc` w tabeli `zamowienia`, zamiast liczenia `wartosc * 1.23` za każdym razem. Obejmują też zduplikowaną nazwę klienta skopiowaną bezpośrednio do wiersza raportu, żeby raport nie musiał łączyć się z tabelą `klienci` za każdym uruchomieniem, jeśli ten raport jest odpytywany bardzo często i w pełni znormalizowane złączenie stało się wąskim gardłem. Obejmują wreszcie osobną tabelę podsumowań — na przykład dzienne albo miesięczne sumy zamówień per klient, przeliczone raz i przechowywane gotowe, zamiast liczenia agregatu z tabeli `zamowienia` przy każdym otwarciu raportu.

Cena denormalizacji jest zawsze ta sama: dane zduplikowane w dwóch miejscach muszą pozostać ze sobą zgodne, a baza sama z siebie tej zgodności nie pilnuje tak, jak pilnuje jej ograniczenie klucza obcego w schemacie znormalizowanym. Zmiana nazwy klienta w tabeli `klienci` nie propaguje się automatycznie do skopiowanej nazwy w tabeli raportowej — obowiązek utrzymania tej zgodności spada albo na logikę aplikacji, która musi pamiętać o aktualizacji obu miejsc, albo na wyzwalacz bazodanowy, który robi to automatycznie przy każdej zmianie źródłowego wiersza. Ten drugi wariant jest bezpieczniejszy, bo nie zależy od tego, czy programista pamięta o obu miejscach za każdym razem, ale dokłada własny koszt — wyzwalacz uruchamia się przy każdej zmianie w tabeli źródłowej, spowalniając ją dokładnie tak, jak każdy dodatkowy indeks spowalnia zapis, co opisuje strona Kiedy indeks szkodzi.

Częste błędy

Traktowanie denormalizacji jako domyślnego punktu wyjścia. Świadoma denormalizacja ma sens jako reakcja na zmierzony problem wydajności odczytu w schemacie, który wcześniej był w pełni znormalizowany — projektowanie od razu zdenormalizowanej struktury, bez znormalizowanego punktu odniesienia, zwykle odtwarza dokładnie te anomalie, które normalizacja miała eliminować, bez żadnej korzyści w zamian, bo problem wydajności jeszcze nie zaistniał.

Zapominanie o zależności przechodniej przy projektowaniu nowej tabeli. Dopisanie kolumny `klient_miasto` bezpośrednio do tabeli `zamowienia` zamiast trzymania jej w `klienci` i odwoływania się przez `klient_id` to dokładnie ten sam błąd, który krok 3NF wyżej naprawia — łatwo go popełnić ponownie przy dodawaniu nowej funkcjonalności do już istniejącego, znormalizowanego schematu.

Denormalizacja bez mechanizmu utrzymania spójności. Skopiowanie danych bez wyzwalacza ani procedury odświeżającej je regularnie prowadzi do rozjazdu, który ujawnia się dopiero po miesiącach, kiedy raport pokazuje starą nazwę klienta, a nikt nie pamięta już, że ta kolumna w ogóle jest kopią.

□□ Ćwiczenia

> Zadania sprawdzające.

Ćwiczenie 1 — rozpoznanie zależności przechodniej

Tabela `pracownicy_projekty` ma kolumny `pracownik_id`, `projekt_id`, `pracownik_dzial`, `dzial_kierownik`. Klucz to para `(pracownik_id, projekt_id)`. Czy ta tabela jest w 3NF? Jeśli nie, jaką zależność przechodnią trzeba usunąć?

Odpowiedź

Nie jest w 3NF. `pracownik_dzial` zależy tylko od `pracownik_id`, nie od pełnego klucza — to już naruszenie 2NF. Dodatkowo `dzial_kierownik` zależy od `pracownik_dzial`, a nie bezpośrednio od klucza — to zależność przechodnia przez `pracownik_dzial`. Rozwiązaniem jest wydzielenie osobnej tabeli `dzialy` z kolumnami `dzial_id` i `kierownik`, tabeli `pracownicy` z kolumną `dzial_id` jako kluczem obcym, i pozostawienie w `pracownicy_projekty` tylko `pracownik_id` i `projekt_id`.

Pytania kontrolne

1. Dlaczego anomalia usuwania w tabeli niezormalizowanej jest szczególnie niebezpieczna w porównaniu z anomalią wstawiania?

Odpowiedź

Anomalia wstawiania blokuje pewne operacje, ale nie niszczy danych — po prostu nie da się czegoś zapisać, dopóki struktura się nie zmieni. Anomalia usuwania kasuje dane bezpowrotnie: usunięcie jednego faktu (ostatniego zamówienia klienta) zabiera ze sobą inny, logicznie niezwiązany fakt (istnienie klienta), którego nikt nie chciał usuwać, a który po usunięciu trzeba

by odtwarzać z pamięci albo z kopii zapasowej.

2. Dlaczego w pełni znormalizowany schemat bywa wolniejszy do odczytu niż zdenormalizowany, mimo że eliminuje anomalie zapisu?

Odpowiedź

Normalizacja rozbija dane logicznie należące do jednego raportu na kilka osobnych tabel, więc odtworzenie pełnego obrazu — na przykład nazwy klienta obok jego zamówienia — wymaga złączenia tych tabel przy każdym odczycie, zamiast odczytania gotowej, wcześniej skleconej wartości z jednego wiersza. Ten koszt złączenia jest ceną płaconą za brak redundancji, i to właśnie ta cena bywa powodem świadomej denormalizacji, kiedy odczyt jest znacznie częstszy i bardziej krytyczny czasowo niż zapis.

Powiązane: Czym jest złączenie: iloczyn kartezjański i warunek, który go ogranicza · Kiedy indeks szkodzi · Model relacyjny: tabela, wiersz, kolumna, dziedzina · Relacje: 1:1, 1:N, N:M i tabela łącząca

Źródła: E.F. Codd, „A Relational Model of Data for Large Shared Data Banks” (1970) · dokumentacja PostgreSQL, rozdział „Data Definition” · Microsoft Docs, „Description of the database normalization basics”

Revision #46
Created 2026-09-24 22:42:48 UTC by Tomasz Konieczny
Updated 2026-10-26 14:11:50 UTC by Tomasz Konieczny