Klucze: główny, obcy, unikalny, naturalny, sztuczny

Poziom: podstawy **Dotyczy: wszystkie silniki** **Czas: ~15 min**

Klucz główny, klucz obcy i klucz unikalny brzmią jak trzy warianty tej samej rzeczy, a rozwiązują trzy różne problemy.

Skróty użyte na tej stronie (4)

Skrót	Co znaczy
MSSQL	Microsoft SQL Server: silnik bazy danych Microsoftu, na nim w Zielonej stoi księgowość Optima
PESEL	Powszechny Elektroniczny System Ewidencji Ludności: jedenastocyfrowy numer identyfikacyjny osoby w Polsce, przykład danych, które filtr danych osobowych powinien wykrywać
SQL	Structured Query Language: język zapytań do relacyjnych baz danych
UUID	Universally Unique Identifier: długi, praktycznie niepowtarzalny identyfikator

Fiszka

ZMIENIA SYSTEM

Utworzenie tabeli z kluczem głównym

```
CREATE TABLE klienci (
    id SERIAL PRIMARY KEY,
    nazwa TEXT NOT NULL,
    nip TEXT UNIQUE
);
```

Kolumna `id` staje się kluczem głównym tabeli, a kolumna `nip` dostaje osobne ograniczenie unikalności.

Przykład:

```
CREATE TABLE
```

□□ **id** = pokaż użytkownika, jego numer UID oraz grupy, do których należy

□□ **ZMIENIA SYSTEM**

Utworzenie tabeli z kluczem obcym

```
CREATE TABLE zamowienia (
    id SERIAL PRIMARY KEY,
    klient_id INTEGER NOT NULL,
    data_zlozenia DATE NOT NULL DEFAULT CURRENT_DATE,
    FOREIGN KEY (klient_id) REFERENCES klienci(id)
);
```

Kolumna `klient_id` może przyjmować wyłącznie wartości, które istnieją w kolumnie `id` tabeli `klienci` — silnik odrzuci wstawienie zamówienia z nieistniejącym klientem.

Przykład:

```
CREATE TABLE
```

□□ **id** = pokaż użytkownika, jego numer UID oraz grupy, do których należy

□□ **ZMIENIA SYSTEM**

Dodanie klucza obcego do istniejącej tabeli

```
ALTER TABLE zamowienia
    ADD CONSTRAINT fk_zamowienia_klient
    FOREIGN KEY (klient_id) REFERENCES klienci(id);
```

Ten sam efekt na tabeli, która już istnieje i już ma dane — silnik przy wykonaniu tej komendy sprawdza wszystkie istniejące wiersze pod kątem zgodności, zanim ograniczenie zacznie obowiązywać.

Przykład:

```
ALTER TABLE
```

◫◫ TYLKO ODCZYT

Sprawdzenie kluczy tabeli przez information_schema

```
SELECT constraint_name, constraint_type
FROM information_schema.table_constraints
WHERE table_name = 'zamowienia';
```

Przykład:

```
      constraint_name       | constraint_type
----------------------------+-----------------
 zamowienia_pkey             | PRIMARY KEY
 fk_zamowienia_klient        | FOREIGN KEY
(2 rows)
```

Po co to istnieje

Tabela relacyjna, sama z siebie, nie ma pojęcia „ten konkretny wiersz". Ma zbiór kolumn i zbiór wierszy — żeby wskazać jeden, dokładnie jeden wiersz, potrzebny jest sposób na jego jednoznaczne odróżnienie od wszystkich pozostałych. Do tego służy klucz.

> **Pojęcie: klucz.** Minimalny zbiór kolumn, którego wartości jednoznacznie identyfikują wiersz w tabeli — żadne dwa wiersze nie mogą mieć w tych kolumnach tej samej kombinacji wartości. Słowo „minimalny" jest tu istotne: jeśli usunięcie którejkolwiek kolumny ze zbioru sprawia, że jednoznaczność znika, zbiór jest kluczem; jeśli można coś odjąć i jednoznaczność zostaje, to nie jest jeszcze klucz w ścisłym sensie, tylko coś większego.

Rodzaje kluczy

Zanim padnie wyliczenie, warto ustalić, że wszystkie poniższe pojęcia opisują tę samą podstawową ideę — jednoznaczną identyfikację wiersza — różniącą się tym, jak bardzo są minimalne i jaką rolę pełnią w tabeli.

Pojęcie: superklucz. Dowolny zbiór kolumn, którego wartości jednoznacznie identyfikują wiersz — bez wymogu minimalności. Cała tabela (wszystkie jej kolumny razem) zawsze jest superkluczem, bo dwa identyczne wiersze pod każdym względem są w praktyce nieodróżnialne i tak samo traktowane.

Pojęcie: klucz kandydujący. Superklucz, z którego nie da się odjąć żadnej kolumny bez utraty jednoznaczności — czyli superklucz minimalny. Tabela może mieć kilku kandydatów naraz: `klienci` z kolumnami `id`, `nip` i `email` (każda z osobna unikalna) ma trzy klucze kandydujące, jeśli każda kolumna sama w sobie jednoznacznie identyfikuje klienta.

Pojęcie: klucz główny (primary key). Ten jeden klucz kandydujący, który wybieramy jako oficjalny identyfikator wiersza. Silnik dodatkowo wymusza na nim `NOT NULL` — klucz główny nigdy nie może być pusty, bo pusty identyfikator nie identyfikuje niczego.

Pojęcie: klucz obcy (foreign key). Kolumna albo zestaw kolumn w jednej tabeli, którego wartości muszą odpowiadać wartościom klucza (zwykle głównego) w innej tabeli — albo być puste, jeśli kolumna dopuszcza `NULL`. To mechanizm, który fizycznie realizuje relację między tabelami, opisaną na stronie Relacje: 1:1, 1:N, N:M i tabela łącząca.

Pojęcie: klucz unikalny (unique key). Ograniczenie mówiące, że wartości w danej kolumnie (albo zestawie kolumn) się nie powtarzają — bez pretensji do bycia głównym identyfikatorem wiersza. Różni się od klucza głównego dwiema rzeczami: tabela może mieć wiele ograniczeń `UNIQUE` naraz, oraz — w większości silników — kolumna `UNIQUE` dopuszcza wielokrotne wystąpienie `NULL`, bo `NULL` w logice SQL nie jest równy samemu sobie, więc dwie wartości `NULL` nie są traktowane jako powtórzenie.

Pojęcie: klucz złożony (composite key). Klucz zbudowany z więcej niż jednej kolumny — żadna z kolumn osobno nie jest unikalna, ale ich para (albo trójka) już tak. Typowy przykład to tabela łącząca `uzytkownicy_role(uzytkownik_id, rola_id)`, gdzie kluczem głównym jest para tych dwóch kolumn naraz.

Zestawienie wszystkich pięciu obok siebie porządkuje, który termin odpowiada na jakie pytanie:

Rodzaj klucza	Pytanie, na które odpowiada	Ile może ich być w tabeli
Superklucz	jaki zbiór kolumn na pewno identyfikuje wiersz	dowolnie wiele
Klucz kandydujący	jaki minimalny zbiór kolumn identyfikuje wiersz	zero, jeden albo wiele
Klucz główny	który klucz kandydujący wybraliśmy jako oficjalny	dokładnie jeden
Klucz unikalny	jakie inne kolumny też się nie powtarzają	wiele, niezależnie od klucza głównego
Klucz obcy	skąd ta kolumna bierze dopuszczalne wartości	wiele, po jednym na każdą relację z inną tabelą

Naturalny czy sztuczny

Pojęcie: klucz naturalny. Identyfikator zaczerpnięty z danych, które i tak istnieją w świecie rzeczywistym opisywanym przez tabelę — NIP firmy, numer PESEL osoby, adres e-mail konta. Zaletą jest to, że nie wymaga dodatkowej kolumny wyłącznie na potrzeby bazy, ma sens dla człowieka czytającego dane wprost, i naturalnie zapobiega powstaniu dwóch wierszy dla tego samego bytu rzeczywistego, bo sam byt narzuca jedną wartość klucza. Wadą jest to, że dane ze świata rzeczywistego bywają zmienne — adres e-mail się zmienia, numer telefonu bywa przydzielany ponownie po latach, a to, co wyglądało na unikalne (PESEL), potrafi mieć wyjątki (obcokrajowcy bez nadanego numeru, błędy historyczne w danych migrowanych ze starszych systemów). Zmiana wartości klucza naturalnego jest kosztowna, bo trzeba zaktualizować każdy wiersz, który się do niej odwołuje przez klucz obcy.

Pojęcie: klucz sztuczny (surogatowy, surrogate key). Identyfikator wygenerowany przez sam system, bez żadnego znaczenia biznesowego — liczba z sekwencji, `SERIAL`, `IDENTITY`, `AUTO_INCREMENT` albo UUID. Zaletą jest stabilność: wartość nigdy się nie zmienia, bo nic w świecie rzeczywistym jej nie determinuje, złączenia między tabelami są tanie (porównanie liczb całkowitych), a dodanie nowego wiersza nigdy nie koliduje z istniejącym z powodu zmiany danych źródłowych. Wadą jest to, że sam w sobie nic nie mówi o wierszu — dwa wiersze o `id = 41` i `id = 42` mogą przypadkiem opisywać tego samego klienta wpisanego dwa razy, bo klucz sztuczny nie chroni przed duplikatem biznesowym; tę ochronę trzeba dołożyć osobno, zwykle właśnie przez ograniczenie `UNIQUE` na kolumnie z kluczem naturalnym, obok klucza sztucznego jako głównego.

W praktyce większość schematów na tej półce łączy oba podejścia: klucz sztuczny jako klucz główny, bo tani w złączeniach i stabilny, oraz ograniczenie `UNIQUE` na kluczu naturalnym tam, gdzie

taki istnieje, żeby nie stracić ochrony przed duplikatem.

Klucz obcy może wskazywać też klucz unikalny, nie tylko główny

Klucz obcy nie musi odwoływać się wyłącznie do klucza głównego tabeli nadrzędnej — wystarczy, że wskazuje kolumnę objętą jakimkolwiek ograniczeniem gwarantującym jednoznaczność, czyli kluczem głównym albo kluczem unikalnym. Jeśli tabela `klienci` ma klucz główny `id`, ale też unikalny `nip`, to klucz obcy w innej tabeli może równie dobrze odwoływać się do `nip`, o ile intencją jest powiązanie po numerze identyfikacji podatkowej, a nie po sztucznym identyfikatorze. Silnik i tak wymaga, żeby kolumna, do której odwołuje się klucz obcy, była objęta ograniczeniem unikalności — bez tego nie dałoby się jednoznacznie rozstrzygnąć, do którego wiersza tabeli nadrzędnej odwołanie się odnosi.

Jak to działa

Większość silników zakłada indeks pod kluczem głównym automatycznie — w PostgreSQL, MSSQL i MySQL `PRIMARY KEY` tworzy unikalny indeks B-drzewa bez dodatkowej komendy. Klucz obcy takiego automatycznego indeksu w PostgreSQL i MSSQL nie dostaje — kolumna `klient_id` w tabeli `zamowienia` jest sprawdzana pod kątem istnienia wartości w tabeli `klienci`, ale sama w sobie nie jest indeksowana, dopóki nie założymy indeksu ręcznie. MySQL z silnikiem InnoDB robi to inaczej i zakłada indeks pod kolumną klucza obcego automatycznie.

Sprawdzenie klucza obcego wykonuje się przy każdym `INSERT` i `UPDATE` na tabeli, która go posiada, oraz przy każdym `DELETE` na tabeli, do której się odwołuje — silnik musi zweryfikować, że usunięcie wiersza nadrzędnego nie zostawi osieroconych odwołań. Domyślne zachowanie w takiej sytuacji to zablokowanie usunięcia (`ON DELETE RESTRICT` albo `NO ACTION`, zależnie od silnika); można je zmienić na kaskadowe usunięcie zależnych wierszy (`ON DELETE CASCADE`) albo ustawienie klucza obcego na `NULL` (`ON DELETE SET NULL`), jeśli kolumna na to pozwala.

Częste błędy

Brak indeksu na kolumnie klucza obcego w PostgreSQL. Wielu programistów zakłada, że skoro `FOREIGN KEY` istnieje, kolumna jest zindeksowana. W PostgreSQL i MSSQL nie jest — każde usunięcie wiersza z tabeli nadrzędnej wymusza pełny skan tabeli podrzędnej w poszukiwaniu wierszy zależnych, co przy dużej tabeli oznacza zauważalne spowolnienie i długo trzymaną blokadę.

Traktowanie `UNIQUE` jak lżejszej wersji `PRIMARY KEY`. Różnica nie jest tylko kosmetyczna. Kolumna z ograniczeniem `UNIQUE` pozwala na wielokrotne `NULL`, więc próba użycia jej jako jedynego zabezpieczenia przed duplikatami bez dodatkowego `NOT NULL` nie zadziała tak, jak ktoś by oczekiwał — dwa wiersze z `NULL` w tej kolumnie przejdą bez błędu.

Traktowanie klucza naturalnego jako niezmiennego, bo tak wygląda na pierwszy rzut oka. Numer PESEL jest jedyny w swoim rodzaju dla obywatela, ale system, który migrował dane historyczne, ma w bazie klientów zagranicznych bez nadanego numeru albo z numerem tymczasowym, który później się zmienił. Jeśli PESEL jest kluczem głównym, a nie tylko atrybutem z ograniczeniem `UNIQUE`, każda taka zmiana wymusza aktualizację wszystkich wierszy, które się do niego odwołują przez klucz obcy — dokładnie tego problemu unika klucz sztuczny.

Dodanie klucza obcego do tabeli, która referuje jeszcze nieistniejącą tabelę. Komenda `CREATE TABLE zamowienia (... FOREIGN KEY (klient_id) REFERENCES klienci(id))` zwraca błąd, jeśli `klienci` jeszcze nie istnieje — silnik nie potrafi sprawdzić klucza obcego wobec czegoś, czego nie ma. Kolejność tworzenia tabel musi iść od tabeli nadrzędnej do podrzędnej, albo klucz obcy trzeba dodać później przez `ALTER TABLE`.

□□ Ćwiczenia

> Poniższe zadania wykonujemy na dowolnym silniku SQL z uprawnieniami do tworzenia tabel we własnej, testowej bazie.

Ćwiczenie 1 — złamanie klucza obcego

```
CREATE TABLE klienci (id SERIAL PRIMARY KEY, nazwa TEXT);
CREATE TABLE zamowienia (
    id SERIAL PRIMARY KEY,
    klient_id INTEGER REFERENCES klienci(id)
);
INSERT INTO zamowienia (klient_id) VALUES (999);
```

Co zwróci silnik i dlaczego?

Odpowiedź

Błąd naruszenia klucza obcego, bo w tabeli `klienci` nie ma jeszcze wiersza z `id = 999`:

```
ERROR:  insert or update on table "zamowienia" violates foreign key constraint
DETAIL:  Key (klient_id)=(999) is not present in table "klienci".
```

Ćwiczenie 2 — klucz naturalny kontra sztuczny

Projektujemy tabelę `pracownicy` w firmie, która zatrudnia zarówno obywateli z nadanym numerem PESEL, jak i pracowników zagranicznych bez niego. Który klucz wybieramy jako główny i co robimy z PESEL-em?

Odpowiedź

Kluczem głównym zostaje kolumna sztuczna, np. `id SERIAL PRIMARY KEY` — działa identycznie dla każdego pracownika, niezależnie od tego, czy ma PESEL. Kolumna `pesel` zostaje jako zwykły atrybut z ograniczeniem `UNIQUE`, które dopuszcza `NULL` dla pracowników bez numeru (w większości silników wiele wartości `NULL` w kolumnie `UNIQUE` nie koliduje ze sobą). Dzięki temu baza wciąż pilnuje, żeby dwóch różnych pracowników nie miało tego samego PESEL-u, nie wymuszając go tam, gdzie go po prostu nie ma.

Pytania kontrolne

1. Tabela `produkty` ma kolumny `id` (sztuczny klucz główny) i `kod_kreskowy` (unikalny w świecie rzeczywistym, ale bez ograniczenia `UNIQUE` w bazie). Co może pójść źle?

Odpowiedź

Nic w bazie nie broni wstawienia dwóch produktów z tym samym kodem kreskowym — każdy dostanie inny `id`, więc klucz główny się nie poskarży, a duplikat biznesowy powstanie po cichu. Naprawa to dodanie `ALTER TABLE produkty ADD CONSTRAINT uq_produkty_kod UNIQUE (kod_kreskowy);`.

2. Dlaczego klucz złożony `(uzytkownik_id, rola_id)` jako klucz główny tabeli łączącej jest lepszym rozwiązaniem niż dodanie do niej osobnej, sztucznej kolumny `id`?

Odpowiedź

Bo sens tabeli łączącej to właśnie ta para — jeden użytkownik z jedną rolą występuje najwyżej raz. Klucz złożony wymusza tę regułę za darmo, przy okazji bycia kluczem głównym. Dodatkowa kolumna `id` niczego by nie dodała poza koniecznością osobnego ograniczenia `UNIQUE (uzytkownik_id, rola_id)`, żeby uzyskać tę samą gwarancję, którą klucz złożony daje wprost.

Powiązane: Model relacyjny: tabela, wiersz, kolumna, dziedzina · Relacje: 1:1, 1:N, N:M i tabela łącząca · Ograniczenia: NOT NULL, UNIQUE, CHECK, FOREIGN KEY, DEFAULT · Sekwencje, IDENTITY, AUTO_INCREMENT

Źródła: dokumentacja PostgreSQL, rozdz. „Constraints" · C. J. Date, „An Introduction to Database Systems", rozdz. o kluczach kandydujących

---

Revision #47
Created 2026-09-24 22:41:36 UTC by Tomasz Konieczny
Updated 2026-10-26 14:10:34 UTC by Tomasz Konieczny