Ograniczenia: NOT NULL, UNIQUE, CHECK, FOREIGN KEY, DEFAULT

> Ograniczenie to reguła zapisana raz przy tabeli, której silnik pilnuje przy każdym zapisie, zamiast liczyć na to, że każda aplikacja i każdy skrypt importujący dane będzie ją respektować samodzielnie.

Skróty użyte na tej stronie (3)

Skrót	Co znaczy
CSV	w Windowsie: Cluster Shared Volume, dysk widoczny naraz dla wszystkich węzłów klastra; w danych: Comma-Separated Values, plik z wartościami oddzielonymi przecinkami
ISO	International Organization for Standardization: organizacja, która wydaje normy techniczne używane na całym świecie
SQL	Structured Query Language: język zapytań do relacyjnych baz danych

Fiszka

ZMIENIA SYSTEM

Zakaz braku wartości: NOT NULL

```
CREATE TABLE klienci (
    id INTEGER PRIMARY KEY,
    nazwa VARCHAR(200) NOT NULL,
    miasto VARCHAR(100),
    nip VARCHAR(10)
);
```

`NOT NULL` przy kolumnie `nazwa` każe silnikowi odrzucić każdy `INSERT` i `UPDATE`, który zostawiłby tę kolumnę bez wartości — `miasto` i `nip` mogą być `NULL`, `nazwa` nigdy.

Przykład:

```
INSERT INTO klienci (id, miasto, nip) VALUES (10, 'Radom', '7961234567');
```

```
ERROR:  null value in column "nazwa" of relation "klienci" violates not-null constraint
DETAIL:  Failing row contains (10, null, Radom, 7961234567).
```

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

□□ **ZMIENIA SYSTEM**

Dodanie NOT NULL do istniejącej kolumny: ALTER TABLE

```
ALTER TABLE klienci ALTER COLUMN miasto SET NOT NULL;
```

Ten sam efekt na kolumnie, która już istnieje i już ma dane — silnik najpierw sprawdza, czy którykolwiek istniejący wiersz ma `NULL` w tej kolumnie, i odrzuca całą operację, jeśli choć jeden taki się znajdzie.

Przykład:

```
ERROR:  column "miasto" of relation "klienci" contains null values
```

□□ **SET** = Switch Embedded Teaming: zespół kart sieciowych zbudowany w przełączniku wirtualnym Hyper-V; łączy dwie karty w jedno łącze bez ustawień po stronie przełącznika fizycznego, więc bez LACP

□□ **ZMIENIA SYSTEM**

Zakaz duplikatów: UNIQUE

```
ALTER TABLE klienci ADD CONSTRAINT klienci_nip_key UNIQUE (nip);
```

Nazywamy ograniczenie sami — `klienci_nip_key` zamiast pozwolić silnikowi nadać nazwę automatyczną, bo nazwa nadana automatycznie (typu `klienci_nip_key1` po kolejnej migracji albo losowy ciąg znaków w niektórych silnikach) nie mówi nic czytelnego w komunikacie błędu ani w wynikach zapytań do katalogu systemowego, gdy trzeba tę regułę odnaleźć miesiące później.

Przykład:

```
ALTER TABLE
```

```
INSERT INTO klienci (id, nazwa, nip) VALUES (11, 'Duplikat Sp. z o.o.', '5261234567');
```

```
ERROR:  duplicate key value violates unique constraint "klienci_nip_key"
DETAIL:  Key (nip)=(5261234567) already exists.
```

◫◫ ZMIENIA SYSTEM

Warunek na wartości: CHECK

```
ALTER TABLE zamowienia ADD CONSTRAINT zamowienia_wartosc_check CHECK (wartosc >= 0);
```

`CHECK` sprawdza dowolne wyrażenie logiczne dla każdego wiersza z osobna — tu odrzuca ujemne kwoty zamówienia, niezależnie od tego, przez które polecenie (`INSERT` czy `UPDATE`) próbowano je wprowadzić.

Przykład:

```
UPDATE zamowienia SET wartosc = -100 WHERE id = 101;
```

```
ERROR:  new row for relation "zamowienia" violates check constraint
"zamowienia_wartosc_check"
DETAIL:  Failing row contains (101, 1, 2024-01-15, zrealizowane, -100.00).
```

◫◫ ZMIENIA SYSTEM

Odwołanie do innej tabeli: FOREIGN KEY

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

Klucz obcy gwarantuje, że wartość w `klient_id` odpowiada istniejącemu wierszowi w `klienci.id`, albo jest `NULL`, jeśli kolumna na to pozwala — mechanizm opisany od podstaw na stronie Klucze: główny, obcy, unikalny, naturalny, sztuczny.

Przykład:

```
INSERT INTO zamowienia (klient_id, data, status, wartosc) VALUES (999, CURRENT_DATE,
'nowe', 500.00);
```

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

ZMIENIA SYSTEM

Klucz obcy z regułą kasowania: ON DELETE CASCADE / SET NULL / RESTRICT / NO ACTION

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

`ON DELETE CASCADE` każe silnikowi usunąć razem z klientem wszystkie jego zamówienia — usunięcie klienta `Podlaski Handel` usunie automatycznie zamówienie 108, bez osobnego polecenia `DELETE FROM zamowienia`. Alternatywy dają inne konsekwencje: `SET NULL` zeruje `klient_id` w zamówieniach zamiast je kasować, `RESTRICT` i `NO ACTION` odmawiają usunięcia klienta, dopóki istnieje choć jedno jego zamówienie — różnica między `RESTRICT` a `NO ACTION` sprowadza się w większości silników do momentu sprawdzenia warunku (natychmiast kontra na końcu instrukcji), nie do efektu końcowego.

Przykład:

```
DELETE FROM klienci WHERE nazwa = 'Podlaski Handel';
```

```
DELETE 1
```

Bez `CASCADE` to samo polecenie kończy się błędem identycznym jak przy próbie wstawienia zamówienia z nieistniejącym klientem, tylko w drugą stronę:

```
ERROR:  update or delete on table "klienci" violates foreign key constraint
"fk_zamowienia_klient" on table "zamowienia"
DETAIL:  Key (id)=(5) is still referenced from table "zamowienia".
```

□□ **SET** = Switch Embedded Teaming: zespół kart sieciowych zbudowany w przełączniku wirtualnym Hyper-V; łączy dwie karty w jedno łącze bez ustawień po stronie przełącznika fizycznego, więc bez LACP

□□ ZMIENIA SYSTEM

Wartość domyślna: DEFAULT

```
ALTER TABLE zamowienia ALTER COLUMN status SET DEFAULT 'nowe';
```

`DEFAULT` podstawia wartość, gdy `INSERT` nie wymienia danej kolumny w ogóle — nie działa, gdy `INSERT` jawnie wstawia `NULL`, bo `NULL` też jest wartością, tylko oznaczającą brak informacji. Nowe zamówienie bez podanego statusu dostanie `'nowe'` automatycznie.

Przykład:

```
INSERT INTO zamowienia (klient_id, data, wartosc) VALUES (1, CURRENT_DATE, 250.00);
SELECT status FROM zamowienia ORDER BY id DESC LIMIT 1;
```

```
 status
--------
 nowe
(1 row)
```

□□ **SET** = Switch Embedded Teaming: zespół kart sieciowych zbudowany w przełączniku wirtualnym Hyper-V; łączy dwie karty w jedno łącze bez ustawień po stronie przełącznika fizycznego, więc bez LACP

Konsekwencje reguł ON DELETE

Reguła	Co się dzieje z zamówieniami usuwanego klienta
`CASCADE`	usuwane razem z klientem
`SET NULL`	`klient_id` zerowane na `NULL`, zamówienie zostaje
`RESTRICT`	usunięcie klienta odrzucone natychmiast, jeśli ma zamówienia
`NO ACTION`	usunięcie klienta odrzucone na końcu instrukcji, jeśli ma zamówienia

Uważaj: `SET NULL` wymaga, żeby kolumna klucza obcego dopuszczała `NULL` — próba dodania `ON DELETE SET NULL` na kolumnie z `NOT NULL` kończy się błędem już przy tworzeniu ograniczenia, nie dopiero przy pierwszym usunięciu.

Po co to istnieje

> **Pojęcie: ograniczenie.** Reguła zapisana raz w definicji tabeli, której silnik pilnuje przy każdej próbie zapisu — zamiast polegać na tym, że każda aplikacja, każdy skrypt migracyjny i każdy analityk piszący ręczny `INSERT` będzie samodzielnie pamiętał o poprawności danych. Ograniczenie działa niezależnie od tego, przez jaką drogę dane trafiają do tabeli — przez aplikację webową, import CSV czy konsolę administratora — bo silnik sprawdza je na poziomie samego zapisu, nie na poziomie konkretnego narzędzia, które ten zapis wykonuje.

Nazywanie ograniczeń samodzielnie, zamiast pozwalać silnikowi nadać nazwę automatyczną, ma znaczenie praktyczne widoczne w każdym przykładzie na tej stronie — komunikat błędu `violates foreign key constraint "fk_zamowienia_klient"` mówi wprost, o którą regułę biznesową chodzi, podczas gdy nazwa automatyczna w niektórych silnikach (na przykład `SYS_C0012345` w Oracle) niczego nie mówi bez dodatkowego zapytania do katalogu systemowego.

Jak to działa

Rozwiń: jak to działa pod spodem

Każde ograniczenie ma swoją składnię inline, zapisaną wprost w `CREATE TABLE` przy definicji kolumny, i swoją składnię przez `ALTER TABLE`, dodawaną do tabeli, która już istnieje:

```
CREATE TABLE produkty (
    id INTEGER PRIMARY KEY,
    cena NUMERIC(10, 2) NOT NULL CHECK (cena >= 0),
    kategoria VARCHAR(50) NOT NULL DEFAULT 'inne'
);
```

Ograniczenie dodane przez `ALTER TABLE` na tabeli z istniejącymi danymi wymaga, żeby silnik sprawdził każdy istniejący wiersz pod kątem nowej reguły, zanim ograniczenie zacznie obowiązywać — dokładnie tak jak w przykładzie `NOT NULL` w Fiszce, gdzie operacja odpada, jeśli choć jeden wiersz już narusza regułę.

Ograniczenia odroczone: DEFERRABLE (PostgreSQL, Oracle).

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

> **Pojęcie: ograniczenie odroczone.** Ograniczenie sprawdzane nie od razu po każdym poleceniu, tylko dopiero przy `COMMIT` całej transakcji — przydatne, gdy dwie tabele odwołują się do siebie nawzajem albo gdy w ramach jednej transakcji trzeba chwilowo przestawić kolejność wierszy tak, że pośredni stan narusza regułę, ale stan końcowy już nie. SQL Server i MySQL nie mają odpowiednika `DEFERRABLE` w standardowej postaci — ograniczenia w nich sprawdzają się zawsze natychmiast, po każdym poleceniu z osobna.

Włączanie i wyłączanie ograniczeń przy ładowaniu masowym.

```
ALTER TABLE zamowienia DISABLE TRIGGER ALL;
-- import danych z pominięciem części sprawdzeń
ALTER TABLE zamowienia ENABLE TRIGGER ALL;
```

Przy imporcie dużych wolumenów danych, o których wiadomo z góry, że są spójne (na przykład eksport z innego systemu, już zweryfikowany), tymczasowe wyłączenie sprawdzania ograniczeń przyspiesza ładowanie, bo silnik nie sprawdza każdego wiersza z osobna w locie. Oracle robi to jawnie przez `ALTER TABLE ... DISABLE CONSTRAINT nazwa`, MySQL przez `SET FOREIGN_KEY_CHECKS = 0` na czas sesji importu. Włączenie ograniczenia z powrotem po takim imporcie w większości silników wymaga jawnej weryfikacji istniejących danych (`ALTER TABLE ... ENABLE CONSTRAINT ... VALIDATE`), żeby nie zostawić w tabeli danych, które go naruszają.

Częste błędy

CHECK z warunkiem odwołującym się do innej tabeli.

```
ERROR:  cannot use subquery in check constraint
```

`CHECK` ocenia wyrażenie tylko w obrębie jednego wiersza tej samej tabeli — reguła w stylu „suma pozycji zamówienia nie może przekroczyć limitu klienta" wymaga wyzwalacza, nie `CHECK`a, bo dotyczy danych z więcej niż jednej tabeli naraz.

Dodanie NOT NULL na kolumnie z istniejącymi NULL-ami.

```
ERROR:  column "miasto" of relation "klienci" contains null values
```

Patrz Fiszka — naprawa to najpierw uzupełnienie brakujących wartości (`UPDATE klienci SET miasto = 'nieznane' WHERE miasto IS NULL`), dopiero potem dodanie ograniczenia.

Usunięcie klienta z zamówieniami bez CASCADE ani SET NULL.

```
ERROR:  update or delete on table "klienci" violates foreign key constraint
"fk_zamowienia_klient" on table "zamowienia"
DETAIL:  Key (id)=(2) is still referenced from table "zamowienia".
```

Reguła domyślna klucza obcego, gdy nie podano jawnie `ON DELETE`, odpowiada `NO ACTION` — silnik odrzuca usunięcie strony nadrzędnej, dopóki istnieją odwołujące się do niej wiersze podrzędne.

□□ Ćwiczenia

> Zadania sprawdzające.

Ćwiczenie 1 — wybór reguły ON DELETE

Projektujemy klucz obcy z tabeli `pozycje` do tabeli `zamowienia` (`pozycje.zamowienie_id → zamowienia.id`). Czy usunięcie zamówienia powinno kasować jego pozycje (`CASCADE`), czy blokować usunięcie, dopóki pozycje istnieją (`RESTRICT`)? A klucz obcy z `zamowienia` do `klienci` (`zamowienia.klient_id → klienci.id`) — ta sama reguła czy inna?

Odpowiedź

Dla `pozycje → zamowienia` sensowny jest `CASCADE` — pozycja zamówienia nie ma żadnego znaczenia bez samego zamówienia, więc usunięcie zamówienia powinno pociągnąć za sobą jego pozycje automatycznie. Dla `zamowienia → klienci` bezpieczniejszy jest `RESTRICT` (albo `NO ACTION`, efekt jest ten sam) — usunięcie klienta, który ma historię zamówień, prawdopodobnie powinno być świadomą decyzją, a nie efektem ubocznym, który po cichu kasuje historię transakcji.

Pytania kontrolne

1. Dlaczego nazywamy ograniczenia sami, zamiast pozwolić silnikowi nadać nazwę automatyczną?

Odpowiedź

Nazwa nadana przez nas trafia wprost do komunikatu błędu przy naruszeniu ograniczenia, więc od razu wiadomo, o którą regułę biznesową chodzi. Nazwa automatyczna bywa nieczytelna (losowy ciąg znaków albo numer w niektórych silnikach) i wymaga dodatkowego zapytania do katalogu systemowego, żeby ustalić, co właściwie zostało naruszone.

2. Czym różni się ograniczenie odroczone (DEFERRABLE) od zwykłego?

Odpowiedź

Zwykłe ograniczenie silnik sprawdza natychmiast po każdym poleceniu, które mogło je naruszyć. Ograniczenie odroczone sprawdza się dopiero przy `COMMIT` całej transakcji, co pozwala na chwilowe naruszenie reguły w trakcie wielu poleceń w ramach jednej transakcji, o ile stan końcowy przy zatwierdzeniu jest już poprawny.

Powiązane: Klucze: główny, obcy, unikalny, naturalny, sztuczny · CREATE TABLE: typy danych i ich odpowiedniki między silnikami · ALTER TABLE: co blokuje tabelę, a co nie · BEGIN, COMMIT, ROLLBACK, SAVEPOINT

Źródła: dokumentacja PostgreSQL, rozdział „Constraints” · dokumentacja Oracle, „Overview of Integrity Constraints” · dokumentacja Microsoft, „Unique constraints and check constraints” · dokumentacja MySQL, „Using FOREIGN KEY Constraints” · ISO/IEC 9075-2:2016 § 4.17

Revision #47
Created 2026-09-24 22:42:18 UTC by Tomasz Konieczny
Updated 2026-10-26 14:11:21 UTC by Tomasz Konieczny