# Replikacja logiczna

Poziom: podstawy | Dotyczy: PostgreSQL 10–17 | Czas: ~25 min

*Replikacja strumieniowa kopiuje cały klaster bajt po bajcie — replikacja logiczna pozwala wybrać pojedyncze tabele i przesłać je między różnymi wersjami głównymi silnika, kosztem tego, że sami musimy zadbać o strukturę po drugiej stronie.*

**Skróty użyte na tej stronie (4)**

| Skrót | Co znaczy |
|---|---|
| **DDL** | Data Definition Language: polecenia SQL zmieniające strukturę (CREATE, ALTER, DROP) |
| **LSN** | Log Sequence Number: rosnący numer pozycji w dzienniku transakcyjnym, po którym silnik rozpoznaje kolejność zmian |
| **SQL** | Structured Query Language: język zapytań do relacyjnych baz danych |
| **WAL** | Write-Ahead Log: dziennik PostgreSQL, zapis zmian przed zapisem danych |

# Fiszka

ZMIENIA SYSTEM

## Utworzenie publikacji wszystkich tabel: CREATE PUBLICATION FOR ALL TABLES

```
CREATE PUBLICATION NazwaPublikacji FOR ALL TABLES;
```

> **Pojęcie: publikacja.** Obiekt po stronie instancji źródłowej, definiujący zbiór tabel, których zmiany mają być udostępniane subskrybentom — odpowiednik listy „co wysyłamy", niezależny od tego, kto akurat subskrybuje.

Przykład:

```
CREATE PUBLICATION
```

◫◫ ZMIENIA SYSTEM

## Utworzenie publikacji wybranych tabel: CREATE PUBLICATION FOR TABLE

```
CREATE PUBLICATION NazwaPublikacji FOR TABLE NazwaSchematu.NazwaTabeli;
```

Ograniczamy publikację do jednej albo kilku tabel wymienionych po przecinku — typowy wybór, gdy replikujemy tylko fragment bazy, na przykład jedną tabelę do systemu raportowego.

Przykład:

```
CREATE PUBLICATION
```

◫◫ ZMIENIA SYSTEM

## Dodanie kolejnej tabeli do istniejącej publikacji: ALTER PUBLICATION ADD TABLE

```
ALTER PUBLICATION NazwaPublikacji ADD TABLE NazwaSchematu.NazwaTabeli;
```

Rozszerzamy zakres publikacji bez konieczności usuwania jej i tworzenia od nowa — subskrypcje korzystające z tej publikacji zaczną odbierać zmiany dodanej tabeli po najbliższym odświeżeniu.

Przykład:

```
ALTER PUBLICATION
```

◫◫ ZMIENIA SYSTEM

## Utworzenie subskrypcji po stronie docelowej: CREATE SUBSCRIPTION

```
CREATE SUBSCRIPTION NazwaSubskrypcji
    CONNECTION 'host=pg01 port=5432 dbname=produkcja user=replikator
password=HasloUzytkownika'
    PUBLICATION NazwaPublikacji;
```

> **Pojęcie: subskrypcja.** Obiekt po stronie instancji docelowej, wskazujący, do jakiej publikacji się podłącza i jakim połączeniem — od chwili utworzenia silnik sam wykonuje początkową synchronizację danych tabel objętych publikacją, a potem przechodzi w tryb ciągłego odbierania zmian.

Przykład:

```
NOTICE:  created replication slot "nazwasubskrypcji" on publisher
CREATE SUBSCRIPTION
```

Reguła w `pg_hba.conf` na źródle, dopuszczająca to połączenie, wygląda i działa dokładnie tak samo jak reguła dla replikacji fizycznej opisana na stronie Uwierzytelnianie i bezpieczeństwo połączeń — różni je jedynie fakt, że subskrypcja łączy się jako zwykłe połączenie do konkretnej bazy, nie do pseudobazy `replication`.

# Z życia wzięte

**Z ŻYCIA WZIĘTE**

Ola chce migrować tylko jedną tabelę `zamowienia` z produkcyjnej bazy na pg01 do nowego serwisu raportowego bez kopiowania całego klastra. Zakładamy publikację i subskrypcję, ale subskrypcja zatrzymuje się od razu błędem o brakującym `REPLICA IDENTITY`, bo tabela nie ma klucza głównego. Po ustawieniu `REPLICA IDENTITY FULL` replikacja rusza, choć każdy `UPDATE` przesyła teraz cały stary wiersz.

# Czym jest replikacja logiczna

> **Pojęcie: replikacja logiczna.** Mechanizm przesyłający między instancjami nie same strony danych, tylko rozkodowane z WAL zmiany na poziomie wierszy — `INSERT`, `UPDATE`, `DELETE` — odtwarzane po drugiej stronie jako zwykłe operacje SQL na już istniejącej tabeli. W przeciwieństwie do repliki fizycznej opisanej na stronie Replikacja strumieniowa, instancja docelowa jest w pełni samodzielną, zapisywalną bazą, a nie tylko odbiorcą w trybie odtwarzania.




*Publikacja na tabeli zamowienia wysyła zmiany wierszy do subskrypcji na serwerze raportowym, tabela bez klucza wymaga REPLICA IDENTITY FULL (kliknięcie otwiera pełny rozmiar)*

| Cecha | Replikacja fizyczna | Replikacja logiczna |
|---|---|---|
| Zakres | cały klaster, wszystkie bazy naraz | wybrane tabele w jednej bazie |
| Wersje silnika po obu stronach | muszą być tą samą wersją główną | mogą się różnić, co jest podstawą migracji między wersjami |
| Zmiany struktury (DDL) | replikowane automatycznie, bo są częścią WAL | nie replikowane automatycznie — wykonujemy je ręcznie po obu stronach |
| Wymóg klucza głównego | brak | wymagany `REPLICA IDENTITY`, zwykle klucz główny |
| Kierunek | jeden, instancja docelowa tylko do odczytu | możliwość wielu niezależnych publikacji i subskrypcji, także między więcej niż dwiema instancjami |

# Włączenie replikacji logicznej

```
wal_level = logical
```

Poziom `logical` to nadzbiór poziomu `replica` opisanego na stronie Logi WAL: włączenie i archiwizacja — dokłada do WAL dodatkowe informacje o strukturze wierszy, bez których dekodowanie zmian na poziom logiczny nie jest możliwe. Zmiana wymaga restartu instancji.

# REPLICA IDENTITY i tabele bez klucza głównego

> **Pojęcie: REPLICA IDENTITY.** Ustawienie tabeli decydujące, jakie kolumny silnik wysyła w ramach `UPDATE` i `DELETE`, żeby strona subskrybująca wiedziała, który dokładnie wiersz zmienić albo usunąć. Domyślna wartość, `DEFAULT`, korzysta z klucza głównego tabeli — jeśli klucz główny istnieje, nic więcej nie trzeba robić.

```
ALTER TABLE NazwaTabeli REPLICA IDENTITY FULL;
```

Tabela bez klucza głównego nie ma z czego skorzystać przy `DEFAULT` — `UPDATE` i `DELETE` na takiej tabeli kończą się błędem uniemożliwiającym replikację, dopóki nie ustawimy `FULL`, każącej wysyłać wszystkie kolumny starej wersji wiersza jako identyfikator. Jest to rozwiązanie działające, ale kosztowne: każdy `UPDATE` i `DELETE` przenosi przez sieć pełną starą wersję wiersza, nie tylko kilka bajtów klucza.

Przykład błędu na tabeli bez klucza głównego i bez ustawionego `REPLICA IDENTITY FULL`:

```
ERROR:  cannot update table "zdarzenia_bez_pk" because it does not have a replica identity and
publishes updates
HINT:  To enable updating the table, set REPLICA IDENTITY using ALTER TABLE.
```

# Monitorowanie

## Sprawdzenie stanu subskrypcji: pg_stat_subscription

```
SELECT subname, pid, received_lsn, latest_end_lsn, latest_end_time
FROM pg_stat_subscription;
```

Wykonane po stronie instancji docelowej — brakujący wiersz albo pusty `pid` oznacza, że proces subskrypcji nie działa, mimo że sama subskrypcja istnieje w katalogu.

Przykład:

```
       subname       |  pid  | received_lsn | latest_end_lsn | latest_end_time
----------------------+-------+--------------+----------------+------------------------
 nazwasubskrypcji      | 28104 | 0/6000210     | 0/6000210        | 2026-09-25 08:12:03+02
(1 row)
```

## Sprawdzenie slotów logicznych po stronie źródła: pg_replication_slots

```
SELECT slot_name, slot_type, active, confirmed_flush_lsn
FROM pg_replication_slots
WHERE slot_type = 'logical';
```

Każda subskrypcja rezerwuje po stronie źródła własny slot logiczny — działa on na tej samej zasadzie ochrony WAL, co slot repliki fizycznej opisany na stronie Replikacja strumieniowa: subskrypcja odłączona na dłużej, a nieusunięta, trzyma segmenty WAL tak samo, jak zrobiłaby to porzucona replika fizyczna.

Przykład:

```
     slot_name      | slot_type | active | confirmed_flush_lsn
----------------------+-----------+--------+----------------------
 nazwasubskrypcji     | logical   | t      | 0/6000210
(1 row)
```

# Sprawdzenie opóźnienia subskrypcji

```
SELECT pg_current_wal_lsn() - confirmed_flush_lsn AS bajtow_opoznienia
FROM pg_replication_slots
WHERE slot_name = 'nazwasubskrypcji';
```

Wykonane na źródle, ta sama logika co przy replikacji fizycznej — różnica w LSN między pozycją bieżącą a ostatnią potwierdzoną przez subskrybenta, przeliczona na bajty.

Przykład:

```
 bajtow_opoznienia
--------------------
              16384
(1 row)
```

# Typowe zastosowanie: migracja między wersjami głównymi bez długiego przestoju

Ponieważ replikacja logiczna nie wymaga tej samej wersji silnika po obu stronach, jest standardowym sposobem migracji na nowszą wersję główną bez okna serwisowego, jakiego wymaga `pg_upgrade` opisany na stronie Upgrade silnika PostgreSQL. Instancja docelowa, na nowej wersji, dostaje subskrypcję wskazującą na wciąż działającą, starą instancję produkcyjną — dogania dane w tle, podczas gdy aplikacja nadal pisze do starej bazy. Dopiero gdy subskrypcja pokazuje zerowe opóźnienie, przełączamy connection string aplikacji na nową instancję i usuwamy subskrypcję.

**Uważaj:** ta ścieżka migracji nie przenosi automatycznie sekwencji, ról ani obiektów DDL — strukturę tabel na instancji docelowej trzeba założyć wcześniej, ręcznie albo przez `pg_dump --schema-only`, opisany na stronie pg_dump, pg_dumpall, pg_restore, zanim subskrypcja zacznie synchronizować dane.

# Konflikty i jak je rozwiązujemy

> **Pojęcie: konflikt replikacji logicznej.** Sytuacja, w której operacja odebrana przez subskrybenta nie daje się zastosować na jego kopii danych — najczęściej naruszenie unikalnego klucza, bo wiersz o tym samym kluczu już istnieje po stronie docelowej z innego źródła. Proces subskrypcji zatrzymuje się na takiej operacji i nie idzie dalej, dopóki konflikt nie zostanie rozwiązany.

Przykład wpisu w logu subskrybenta:

```
ERROR:  duplicate key value violates unique constraint "klienci_pkey"
DETAIL:  Key (id)=(4021) already exists.
CONTEXT:  processing remote data for replication origin "pg_16398" during message type
"INSERT"
```

Rozwiązujemy to na dwa sposoby, zależnie od przyczyny: jeśli wiersz po stronie docelowej jest błędny albo zdublowany przypadkowo, usuwamy go ręcznie i pozwalamy subskrypcji wznowić się samodzielnie; jeśli konflikt wynika z tego, że ten sam wiersz rzeczywiście powinien istnieć po obu stronach z innego powodu, ręcznie pomijamy skonfliktowaną transakcję funkcją `pg_replication_origin_advance()`, wskazując LSN o jeden większy niż ten, na którym subskrypcja utknęła. Do czasu jednego z tych dwóch kroków subskrypcja pozostaje wstrzymana, a dane po stronie docelowej nie są aktualizowane w ogóle — nie tylko dla skonfliktowanej tabeli, ale dla całej subskrypcji.

# Częste błędy

**Założenie, że replikacja logiczna przeniesie też strukturę tabeli.** DDL nie jest częścią strumienia logicznego — nowa kolumna dodana na źródle bez odpowiadającego `ALTER TABLE` na subskrybencie zatrzymuje replikację błędem niedopasowania liczby kolumn.

**Brak REPLICA IDENTITY na tabeli bez klucza głównego, wykryty dopiero na produkcji.** Warto sprawdzić obecność klucza głównego na każdej tabeli objętej publikacją, zanim, a nie po tym, jak pierwszy `UPDATE` skończy się błędem.

**Powiązane:** Replikacja strumieniowa · Logi WAL: włączenie i archiwizacja · Upgrade silnika PostgreSQL · pg_dump, pg_dumpall, pg_restore · Uwierzytelnianie i bezpieczeństwo połączeń

**Źródła:** dokumentacja PostgreSQL, rozdział „Logical Replication"

---

Revision #40
Created 2026-09-25 00:33:27 UTC by Tomasz Konieczny
Updated 2026-10-26 14:12:57 UTC by Tomasz Konieczny