INNER JOIN

> INNER JOIN zwraca tylko te pary wierszy z dwóch tabel, które spełniają warunek złączenia — wszystko inne odpada, i ta strona pokazuje, jak to zapisać poprawnie, jak zapisywano to dawniej, i co się dzieje przy trzech tabelach naraz.

Skróty użyte na tej stronie (2)

Skrót	Co znaczy
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

TYLKO ODCZYT

Złączenie jawne: JOIN ... ON

```
SELECT k.id, k.nazwa, z.id AS zamowienie_id, z.data, z.status, z.wartosc
FROM klienci k
JOIN zamowienia z ON k.id = z.klient_id;
```

`JOIN` bez żadnego przedrostka to dokładnie `INNER JOIN` — słowo `INNER` jest opcjonalne i większość osób je pomija. Warunek złączenia stoi w osobnej klauzuli `ON`, oddzielony od reszty zapytania, co czyni go widocznym na pierwszy rzut oka.

Przykład:

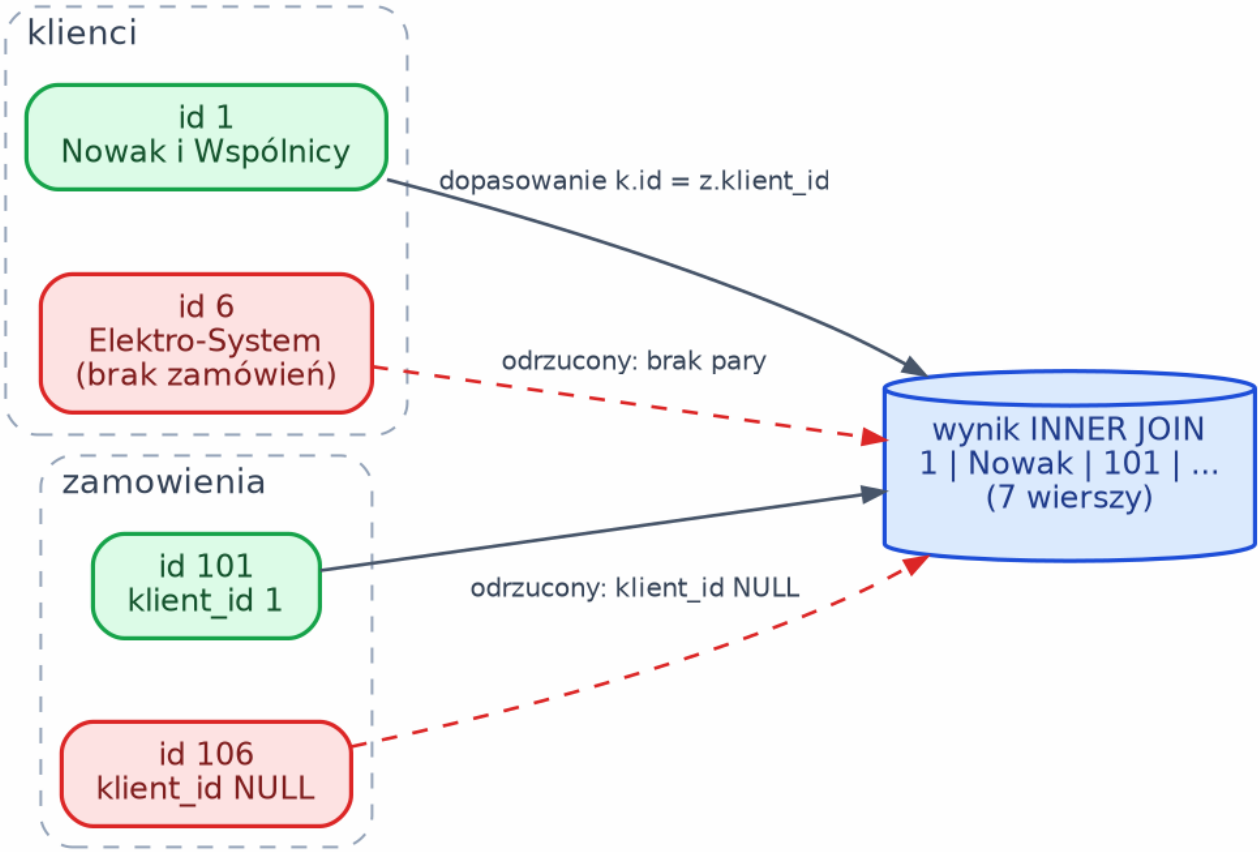
```
 id |           nazwa            | zamowienie_id |    data    |    status    | wartosc

----+----------------------------+---------------+------------+--------------
+---------
```

```
  1 | Nowak i Wspólnicy Sp. z o.o. |          101 | 2024-01-15 | zrealizowane | 1250.00
  1 | Nowak i Wspólnicy Sp. z o.o. |          102 | 2024-02-03 | zrealizowane |  430.50
  2 | Kowalski Handel              |          103 | 2024-01-20 | anulowane    |    0.00
  3 | Zielona Dolina S.A.          |          104 | 2024-03-10 | w realizacji | 2890.00
  2 | Kowalski Handel              |          105 | 2024-03-15 | nowe         |  150.00
  4 | Baltic Trade Sp. z o.o.      |          107 | 2024-03-25 | zrealizowane | 3200.00
  5 | Podlaski Handel              |          108 | 2024-03-28 | anulowane    |    0.00

(7 rows)
```

Tabela `klienci` ma sześć wierszy, `zamowienia` ma osiem, a wynik ma siedem — nie dwanaście czy czterdzieści osiem, jak przy iloczynie kartezjańskim ze strony Czym jest złączenie: iloczyn kartezjański i warunek, który go ogranicza. Dwa wiersze wypadły z dwóch różnych powodów: zamówienie 106 ma `klient_id` równe `NULL`, więc żaden klient go nie dopasuje, a klient o id 6 (Elektro-System Kowalczyk) nie ma żadnego zamówienia, więc nie znalazł pary po prawej stronie. INNER JOIN nie zna litości dla żadnej z tych sytuacji — po prostu odrzuca wiersz, który nie znalazł dopasowania.



Klient bez zamówień i zamówienie bez klienta odpadają z wyniku INNER JOIN (kliknięcie otwiera pełny rozmiar)

join = połącz dwa pliki tekstowe po wspólnej kolumnie, jak złączenie tabel w bazie danych

TYLKO ODCZYT

Stara składnia niejawna: przecinek i WHERE

```
SELECT k.id, k.nazwa, z.id AS zamowienie_id, z.data, z.status, z.wartosc
FROM klienci k, zamowienia z
WHERE k.id = z.klient_id;
```

Ten zapis daje dokładnie ten sam wynik co powyżej — silnik traktuje przecinek w `FROM` jako `CROSS JOIN`, a warunek w `WHERE` filtruje go tak samo, jak `ON` filtruje jawny `JOIN`. Różnica jest wyłącznie czytelnicza: warunek złączenia miesza się tu z resztą warunków biznesowych w `WHERE`, więc łatwo go przeoczyć przy czytaniu, a jeszcze łatwiej — pominąć przy pisaniu, co zamienia zapytanie w przypadkowy iloczyn kartezjański bez żadnego ostrzeżenia ze strony silnika. W tej książce zawsze piszemy jawne `JOIN ... ON`.

TYLKO ODCZYT

Złączenie po kolumnie o tej samej nazwie: USING

```
SELECT klient_id, k.nazwa, z.id AS zamowienie_id, z.data, z.status, z.wartosc
FROM klienci k
JOIN zamowienia z USING (klient_id);
```

`USING (klient_id)` działa tylko wtedy, gdy obie tabele mają kolumnę o dokładnie tej samej nazwie — tu akurat nie ma takiej kolumny wprost (`klienci.id` nazywa się inaczej niż `zamowienia.klient_id`), więc `USING` w tej postaci by nie zadziałało; przykład ma sens dopiero po nadaniu aliasu kolumnie albo przy tabelach, które faktycznie dzielą nazwę klucza. Tam, gdzie kolumny się nazywają tak samo po obu stronach, `USING` ma jedną przewagę nad `ON`: w wyniku pojawia się jedna kolumna `klient_id`, nie dwie identyczne kolumny `k.klient_id` i `z.klient_id` obok siebie.

Przykład:

```
 klient_id |            nazwa             | zamowienie_id |    data    |    status    |
wartosc
-----------+------------------------------+---------------+------------+--------------
+---------
         1 | Nowak i Wspólnicy Sp. z o.o. |           101 | 2024-01-15 | zrealizowane |
1250.00
         1 | Nowak i Wspólnicy Sp. z o.o. |           102 | 2024-02-03 | zrealizowane
|  430.50
         2 | Kowalski Handel              |           103 | 2024-01-20 | anulowane
|    0.00
         3 | Zielona Dolina S.A.          |           104 | 2024-03-10 | w realizacji |
2890.00
         2 | Kowalski Handel              |           105 | 2024-03-15 | nowe
|  150.00
         4 | Baltic Trade Sp. z o.o.      |           107 | 2024-03-25 | zrealizowane |
3200.00
         5 | Podlaski Handel              |           108 | 2024-03-28 | anulowane
|    0.00
(7 rows)
```

Sam wynik jest identyczny co przy `ON`, ale kolumna `klient_id` występuje w nim raz, nie dwa razy — przy `ON k.id = z.klient_id` obie kolumny wciąż istnieją osobno w wyniku i trzeba by jedną z nich pominąć ręcznie w liście `SELECT`, żeby uzyskać ten sam, czysty kształt.

join = połącz dwa pliki tekstowe po wspólnej kolumnie, jak złączenie tabel w bazie danych

TYLKO ODCZYT

NATURAL JOIN i dlaczego go unikamy

```
SELECT *
FROM klienci
NATURAL JOIN zamowienia;
```

`NATURAL JOIN` sam odgaduje warunek złączenia — bierze wszystkie kolumny o identycznych nazwach w obu tabelach i łączy po nich automatycznie, bez żadnego jawnego `ON` ani `USING`. Wygląda na wygodę, jest pułapką: jeśli ktoś doda do którejś tabeli nową kolumnę, która przypadkiem nazywa się tak samo jak kolumna w drugiej tabeli (`nazwa`, `opis`, `status`), `NATURAL JOIN` po cichu zacznie łączyć też po niej, zmieniając wynik zapytania bez żadnej zmiany w jego treści. Warunek złączenia przestaje być czymś, co widać w kodzie — staje się funkcją aktualnego kształtu schematu, którego autor zapytania nie kontroluje. W tej książce `NATURAL JOIN` nie pojawia się nigdzie indziej.

TYLKO ODCZYT

Złączenie trzech i więcej tabel

```
SELECT k.nazwa AS nazwa_klienta, z.id AS zamowienie_id, p.nazwa AS nazwa_produktu,
poz.ilosc, poz.cena
FROM klienci k
JOIN zamowienia z ON k.id = z.klient_id
JOIN pozycje poz ON poz.zamowienie_id = z.id
JOIN produkty p ON p.id = poz.produkt_id;
```

Każdy kolejny `JOIN` dokłada jedną tabelę i jeden własny warunek `ON` — silnik wykonuje je po kolei, zestawiając wynik poprzedniego złączenia z następną tabelą, jakby to było jedno, coraz szersze złączenie. Kolejność `JOIN`-ów w zapytaniu nie musi odzwierciedlać kolejności, w jakiej silnik je faktycznie wykona — o tym decyduje optymalizator, o ile logiczny wynik pozostaje taki sam.

Przykład:

```
        nazwa_klienta         | zamowienie_id |      nazwa_produktu      | ilosc |
cena
```

```
-----------------------------+--------------+------------------------+-------
+---------
 Nowak i Wspólnicy Sp. z o.o. |          101 | Monitor 24 cale        |     1 |  650.00

 Nowak i Wspólnicy Sp. z o.o. |          101 | Kabel HDMI 2m          |     2 |   25.00

 Nowak i Wspólnicy Sp. z o.o. |          102 | Klawiatura mechaniczna |     1 |  220.00

 Zielona Dolina S.A.          |          104 | Monitor 24 cale        |     2 |  650.00

 Baltic Trade Sp. z o.o.      |          107 | Biurko regulowane      |     1 | 1100.00

(5 rows)
```

Pięć wierszy — dokładnie tyle, ile ma tabela `pozycje` — bo to ona jest tu najwęższym ogniwem łańcucha: każda pozycja zamówienia wskazuje dokładnie jedno zamówienie i dokładnie jeden produkt, więc każda pozycja daje dokładnie jeden wiersz wyniku. Zamówienia bez żadnej pozycji (103, 105, 106, 108) nie pojawiają się wcale — INNER JOIN po drodze do `pozycje` je odrzucił, tak samo jak wcześniej odrzucał zamówienia bez klienta.

join = połącz dwa pliki tekstowe po wspólnej kolumnie, jak złączenie tabel w bazie danych

Po co to istnieje

> **Pojęcie: INNER JOIN.** Złączenie, które zwraca wyłącznie te pary wierszy z dwóch tabel, dla których warunek w `ON` dał prawdę — każdy wiersz bez dopasowania po drugiej stronie znika z wyniku całkowicie, bez śladu i bez NULL-i. To jest domyślny, najczęściej używany rodzaj złączenia i punkt odniesienia dla wszystkich pozostałych opisanych w tym rozdziale — LEFT JOIN i RIGHT JOIN różnią się od INNER JOIN wyłącznie tym, że nie odrzucają wierszy bez dopasowania z jednej wybranej strony, tylko wypełniają brakującą część NULL-ami.

Warunek w `ON` nie musi być równością — `JOIN zamowienia z ON z.wartosc > 0` jest składniowo tak samo poprawny jak `ON k.id = z.klient_id`, tyle że rzadko ma sens biznesowy. Złączenie po równości dwóch kluczy jest zdecydowanie najczęstszym przypadkiem, ale sama definicja INNER JOIN nie wymaga akurat tego — wymaga tylko warunku logicznego, który dla każdej pary da prawdę albo fałsz.

Jak to działa

Trzy sposoby zapisu warunku złączenia z tej strony — `ON`, `USING` i niejawny przecinek z `WHERE` — dają identyczny wynik logiczny, bo wszystkie sprowadzają się do tego samego filtra nałożonego na iloczyn kartezjański. Różnią się tylko czytelnością i tym, ile kolumn zostaje w wyniku. `ON` jest najbardziej jednoznaczny i działa dla dowolnego warunku, nie tylko równości nazw kolumn. `USING` jest krótszy, ale zawężony do sytuacji, w której obie strony faktycznie dzielą nazwę kolumny. Niejawny przecinek jest formalnie tym samym co `CROSS JOIN` plus filtr w `WHERE`, tyle że bez żadnego wizualnego rozgraniczenia między warunkiem złączenia a resztą logiki zapytania.

Przy trzech i więcej tabelach silnik nie robi nic jakościowo innego — po prostu stosuje tę samą regułę wielokrotnie, złączenie po złączeniu. To, w jakiej kolejności fizycznie połączy tabele, zależy od statystyk, indeksów i wybranego planu wykonania, nie od kolejności zapisania klauzul `JOIN` w kodzie źródłowym — o ile każda tabela ma jednoznaczny warunek `ON`, wynik końcowy jest ten sam niezależnie od kolejności wykonania.

Wsparcie dla USING i NATURAL JOIN w poszczególnych silnikach. Nie każdy z czterech silników z tej książki zna obie te konstrukcje:

Silnik	USING	NATURAL JOIN
PostgreSQL	tak	tak
MySQL / MariaDB	tak	tak
Oracle	tak	tak
SQL Server	brak	brak

SQL Server nie ma ani `USING`, ani `NATURAL JOIN` — każde złączenie tam zapisuje się w pełni jawnym `ON`, nawet gdy obie kolumny nazywają się identycznie. Kod przenoszony z PostgreSQL czy MySQL, który korzysta z `USING`, wymaga na SQL Server ręcznego rozpisania na `ON` z jawnymi aliasami po obu stronach.

Częste błędy

Niejednoznaczna kolumna bez aliasu tabeli. Skoro `klienci` i `zamowienia` po złączeniu obie mają kolumnę `id`, `SELECT id FROM klienci JOIN zamowienia ON ...` kończy się błędem, nie zgadywaniem, o którą kolumnę chodzi.

```
ERROR:  column reference "id" is ambiguous
LINE 1: SELECT id FROM klienci JOIN zamowienia ON klienci.id = zam...
               ^
```

Naprawa to alias tabeli i prefiks przy każdej niejednoznacznej kolumnie — `k.id` albo `z.id`, nigdy sama `id`, gdy więcej niż jedna tabela w złączeniu ma kolumnę o tej nazwie.

NATURAL JOIN po dodaniu kolumny do schematu. Zapytanie z `NATURAL JOIN`, które działało poprawnie przez rok, zaczyna zwracać inny wynik albo zero wierszy po tym, jak ktoś doda do jednej z tabel kolumnę `opis`, która przypadkiem nazywa się tak samo jak istniejąca kolumna w drugiej tabeli. Żaden komunikat błędu się przy tym nie pojawia — zapytanie po prostu zaczyna liczyć co innego. To jest właściwy powód, dla którego ta strona odradza `NATURAL JOIN` w ogóle, nie tylko styl.

Zapomniany `ON` przy jawnym JOIN. `FROM klienci k JOIN zamowienia z` bez żadnego `ON` ani `USING` zwraca błąd składni w większości silników — to jedna z niewielu zalet jawnej składni nad niejawną: brakujący warunek złączenia jest wykrywany od razu, zamiast po cichu zamienić zapytanie w iloczyn kartezjański, jak to się dzieje przy przecinku w `FROM` bez `WHERE`.

□□ Ćwiczenia

> Zadania sprawdzające.

Ćwiczenie 1 — dwa zamówienia bez pary

Bez uruchamiania zapytania przewidujemy, ile wierszy zwróci `SELECT z.id FROM zamowienia z JOIN klienci k ON k.id = z.klient_id;`, wiedząc, że tabela `zamowienia` ma osiem wierszy, jedno z nich (106) ma `klient_id IS NULL`, a pozostałych siedem wskazuje istniejących klientów.

Odpowiedź

Siedem. INNER JOIN odrzuca każdy wiersz bez dopasowania po drugiej stronie, a zamówienie 106 z `klient_id IS NULL` nigdy nie znajdzie dopasowania — `NULL = k.id` nie daje prawdy dla żadnego `k.id`, niezależnie od tego, jaką wartość `k.id` przyjmuje. To ta sama zasada trójwartościowej logiki, która działa w `WHERE`, opisana na stronie NULL: trzecia wartość logiczna i pułapki, które z niej wynikają.

Ćwiczenie 2 — dodanie trzeciej tabeli

Zapytanie łączy `klienci` z `zamowienia`. Chcemy dołożyć jeszcze `pozycje`, żeby zobaczyć pojedyncze pozycje zamówień, ale nie tracąc klientów, którzy mają zamówienia bez żadnej pozycji. Który `JOIN` na trzeciej tabeli jest tu właściwy?

Odpowiedź

`LEFT JOIN pozycje poz ON poz.zamowienie_id = z.id`, nie zwykły `JOIN`. Zwykły INNER JOIN na `pozycje` odrzuciłby każde zamówienie bez ani jednej pozycji, dokładnie tak jak INNER JOIN na `zamowienia` odrzuca klientów bez zamówień — mechanizm jest identyczny, tylko o jeden poziom głębiej w łańcuchu złączeń. Ten wzorzec — LEFT JOIN po to, żeby nie tracić wierszy „głównych" przy dołączaniu kolejnej, potencjalnie pustej tabeli — rozwija w pełni strona LEFT JOIN i RIGHT JOIN.

Pytania kontrolne

1. Czym różni się `JOIN ... ON` od `JOIN ... USING`, poza długością zapisu?

Odpowiedź

`USING` wymaga, żeby obie tabele miały kolumnę o dokładnie tej samej nazwie, i w zamian scala ją w jedną kolumnę w wyniku zamiast zostawiać dwie identyczne kolumny obok siebie. `ON` działa z dowolnym warunkiem logicznym, także takim, w którym kolumny po obu stronach nazywają się różnie, i zawsze zostawia obie kolumny osobno w wyniku.

2. Dlaczego wynik złączenia klienci+zamowienia+pozycje+produkty ma dokładnie tyle wierszy, ile ma tabela pozycje, a nie tabela zamowienia?

Odpowiedź

Bo `pozycje` jest tu tabelą po stronie „wielu" względem `zamowienia` — jedno zamówienie może mieć kilka pozycji, a każda pozycja wskazuje dokładnie jedno zamówienie i dokładnie jeden produkt. INNER JOIN mnoży wiersze w miejscu, gdzie jeden wiersz z jednej strony ma wiele dopasowań z drugiej, więc ostateczna liczba wierszy odpowiada najwęższemu, najbardziej szczegółowemu poziomowi w łańcuchu złączeń — tu akurat pozycjom, nie zamówieniom ani klientom.

Powiązane: Czym jest złączenie: iloczyn kartezjański i warunek, który go ogranicza · LEFT JOIN i RIGHT JOIN · Warunek w ON a warunek w WHERE: różnica, która zmienia wynik · Relacje: 1:1, 1:N, N:M i tabela łącząca

Źródła: ISO/IEC 9075-2:2016 § 7.7 · dokumentacja PostgreSQL, rozdział „Joins Between Tables" · dokumentacja MySQL, „JOIN Syntax"

Revision #46
Created 2026-09-24 22:41:56 UTC by Tomasz Konieczny
Updated 2026-10-26 14:10:59 UTC by Tomasz Konieczny