CROSS JOIN i SELF JOIN

> CROSS JOIN to jawnie zamówiony iloczyn kartezjański, przydatny do generowania siatek danych — SELF JOIN to złączenie tabeli z samą sobą, przydatne wszędzie tam, gdzie jeden wiersz trzeba porównać z innym wierszem tej samej tabeli.

Skróty użyte na tej stronie (1)

Skrót	Co znaczy
ISO	International Organization for Standardization: organizacja, która wydaje normy techniczne używane na całym świecie

Fiszka

TYLKO ODCZYT

Generowanie siatki: CROSS JOIN

```
SELECT dz.dzial, dn.dzien
FROM (SELECT DISTINCT dzial FROM pracownicy) dz
CROSS JOIN (VALUES ('poniedziałek'), ('środa'), ('piątek')) AS dn(dzien)
ORDER BY dz.dzial, dn.dzien;
```

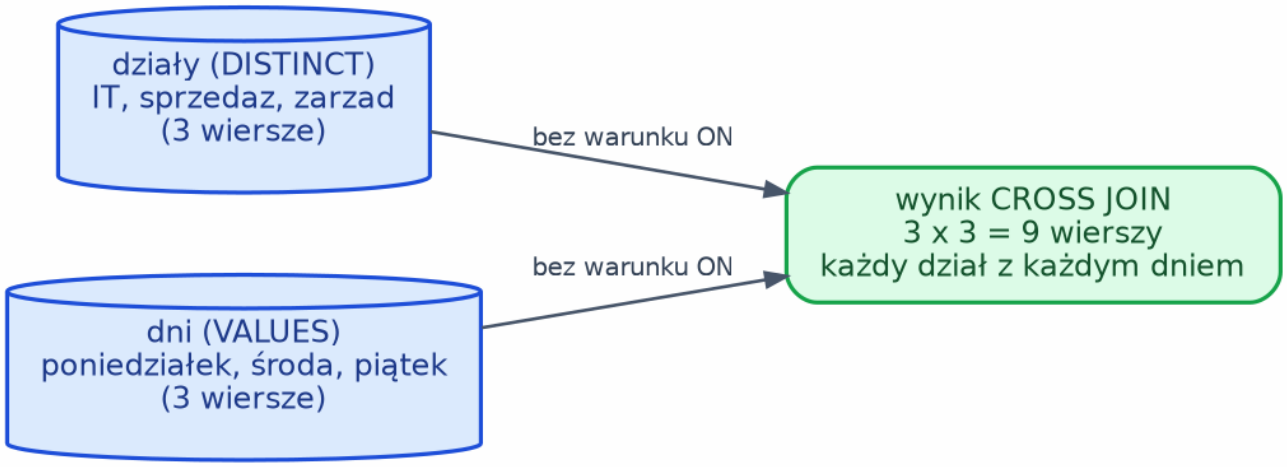
Zestawiamy każdy dział z każdym dniem tygodnia z osobna wybranej listy — to jest dokładnie ten sam iloczyn kartezjański co na stronie Czym jest złączenie: iloczyn kartezjański i warunek, który go ogranicza, tylko tym razem świadomie chciany, nie efekt uboczny zapomnianego warunku.

Przykład:

```
  dzial   |    dzien
----------+--------------
 IT       | piątek
```

```
 IT       | poniedziałek
 IT       | środa
 sprzedaz | piątek
 sprzedaz | poniedziałek
 sprzedaz | środa
 zarzad   | piątek
 zarzad   | poniedziałek
 zarzad   | środa
(9 rows)
```

Trzy działy razy trzy dni dają dziewięć wierszy — komplet par, gotowy na przykład jako szkielet grafiku dyżurów, do którego później dopisujemy, kto konkretnie ma dyżur danego dnia w danym dziale.



CROSS JOIN zestawia każdy dział z każdym dniem, dając iloczyn trzy razy trzy (kliknięcie otwiera pełny rozmiar)

TYLKO ODCZYT

Złączenie tabeli z samą sobą: SELF JOIN

```
SELECT p.imie || ' ' || p.nazwisko AS pracownik,
       prz.imie || ' ' || prz.nazwisko AS przelozony
FROM pracownicy p
JOIN pracownicy prz ON p.przelozony_id = prz.id
```

```
ORDER BY prz.id;
```

Tabela `pracownicy` występuje w tym zapytaniu dwa razy pod dwoma różnymi aliasami — `p` dla pracownika, `prz` dla jego przełożonego — mimo że fizycznie to jedna i ta sama tabela. Bez dwóch różnych aliasów silnik nie potrafiłby rozróżnić, o które wystąpienie kolumny `imie` chodzi w danym miejscu zapytania.

Przykład:

```
    pracownik    |   przelozony
-----------------+----------------
 Marek Zawadzki  | Ewa Kowalczyk
 Tomasz Wróbel   | Ewa Kowalczyk
 Anna Sikora     | Marek Zawadzki
 Paweł Górski    | Tomasz Wróbel
 Katarzyna Baran | Tomasz Wróbel
(5 rows)
```

Ewa Kowalczyk, jako najwyższa w hierarchii, ma `przelozony_id` równe `NULL` i dlatego nie pojawia się tu wcale — zwykły `JOIN` (INNER) wymaga dopasowania po obu stronach, a `NULL` nigdy go nie da. Gdybyśmy chcieli zobaczyć też Ewę, z pustą kolumną `przelozony`, potrzebny byłby `LEFT JOIN` zamiast `JOIN` — dokładnie ta sama zależność co na stronie LEFT JOIN i RIGHT JOIN, tylko że obie strony złączenia pochodzą teraz z tej samej tabeli.

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

ZMIENIA SYSTEM

Porównanie wiersza z poprzednim tego samego klienta

```
SELECT k.nazwa,
       z1.id AS zamowienie_biezace, z1.data AS data_biezaca,
       z2.id AS zamowienie_poprzednie, z2.data AS data_poprzednia, z2.wartosc AS
wartosc_poprzednia
FROM zamowienia z1
JOIN klienci k ON k.id = z1.klient_id
JOIN zamowienia z2 ON z2.klient_id = z1.klient_id AND z2.data < z1.data
WHERE NOT EXISTS (
    SELECT 1 FROM zamowienia z3
    WHERE z3.klient_id = z1.klient_id AND z3.data < z1.data AND z3.data > z2.data
```

```
);
```

`zamowienia` złącza się tu sama ze sobą dwukrotnie pod aliasami `z1` (zamówienie bieżące) i `z2` (kandydat na poprzednie), a podzapytanie `NOT EXISTS` odsiewa wszystkie kandydatury poza tą najbliższą w czasie — bez niego każde wcześniejsze zamówienie tego samego klienta pasowałoby jako „poprzednie" naraz.

Przykład:

```
            nazwa             | zamowienie_biezace | data_biezaca |
zamowienie_poprzednie | data_poprzednia | wartosc_poprzednia
------------------------------+--------------------+--------------
+----------------------+-----------------+--------------------
 Nowak i Wspólnicy Sp. z o.o. |                102 | 2024-02-03   |
101 | 2024-01-15      |            1250.00
 Kowalski Handel             |                105 | 2024-03-15   |
103 | 2024-01-20      |               0.00
(2 rows)
```

Tylko dwóch klientów ma więcej niż jedno zamówienie w naszych danych, więc tylko dla nich w ogóle istnieje jakiekolwiek „poprzednie zamówienie" — klienci z jednym albo zerem zamówień nie mają tu żadnej pary i nie pojawiają się wcale, bo to zwykły `JOIN`, nie `LEFT JOIN`. Ten wzorzec rozwiązuje problem, ale kosztem podzapytania `NOT EXISTS` w środku; strona Funkcje okna — OVER, PARTITION BY, ROW_NUMBER, RANK, LAG/LEAD pokazuje, jak tę samą potrzebę załatwia jedna funkcja `LAG()`, bez samozłączenia i bez podzapytania.

LAG = Link Aggregation Group: grupa łączy spięta w jedno logiczne; wariant MC-LAG rozciąga ją na dwa przełączniki · **join** = połącz dwa pliki tekstowe po wspólnej kolumnie, jak złączenie tabel w bazie danych

TYLKO ODCZYT

Wykrycie duplikatu przez SELF JOIN

Załóżmy, że import danych z innego systemu wprowadził do tabeli `klienci` dodatkowy, siódmy wiersz — tę samą firmę zapisaną drugi raz z tym samym NIP-em:

```
 id |            nazwa             |  miasto   |    nip
----+------------------------------+-----------+------------
  1 | Nowak i Wspólnicy Sp. z o.o. | Warszawa  | 5261234567
  7 | Nowak i Wspólnicy Sp. z o.o. | Warszawa  | 5261234567
```

```
SELECT a.id AS id_a, a.nazwa AS nazwa_a, b.id AS id_b, b.nazwa AS nazwa_b, a.nip
FROM klienci a
JOIN klienci b ON a.nip = b.nip AND a.id < b.id;
```

Warunek `a.id < b.id` jest tu równie ważny jak `a.nip = b.nip` — bez niego każdy wiersz sparowałby się sam ze sobą (bo `nip = nip` zawsze jest prawdą dla tego samego wiersza) i dodatkowo każda para duplikatów pojawiłaby się dwa razy, raz w każdą stronę.

Przykład:

```
 id_a |           nazwa_a            | id_b |           nazwa_b            |    nip
------+------------------------------+------+------------------------------+------------
    1 | Nowak i Wspólnicy Sp. z o.o. |    7 | Nowak i Wspólnicy Sp. z o.o. | 5261234567
(1 row)
```

Dokładnie jedna para — ten klasyczny wzorzec znajduje duplikaty niezależnie od tego, ile razy dana wartość się powtórzyła, bo `a.id < b.id` gwarantuje jedną, uporządkowaną parę na każde dwa identyczne wiersze.

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

Po co to istnieje

Pojęcie: CROSS JOIN. Złączenie bez żadnego warunku `ON`, zwracające pełny iloczyn kartezjański dwóch tabel — dokładnie to pojęcie, które strona Czym jest złączenie: iloczyn kartezjański i warunek, który go ogranicza wprowadziła jako fundament wszystkich pozostałych złączeń. Różnica jest tylko w intencji: tam iloczyn kartezjański był efektem ubocznym zapomnianego warunku, tutaj jest świadomie zamówionym narzędziem.

Legalne zastosowania `CROSS JOIN` sprowadzają się do generowania kompletu kombinacji, które nie wynikają z żadnej relacji między danymi, tylko mają powstać z osobna — siatka dat razy oddziały do raportu obecności, tabela kalendarza z każdym dniem roku, macierz rozmiarów razy kolorów w katalogu produktów. Gdy `CROSS JOIN` pojawia się gdzie indziej — w zapytaniu, które miało zwrócić rozsądną liczbę wierszy, a zwraca ich podejrzanie dużo — jest to niemal zawsze objaw zapomnianego warunku złączenia, opisany w Częstych błędach strony Czym jest złączenie, nie celowa decyzja.

> **Pojęcie: SELF JOIN.** Złączenie tabeli z samą sobą, wymagające nadania jej dwóch różnych aliasów, żeby silnik i autor zapytania mogli odróżnić, które wystąpienie tabeli ma się na myśli w danym miejscu. Sama tabela fizycznie istnieje raz — aliasy istnieją tylko na czas jednego zapytania. Klasyczne zastosowania to hierarchia pracownik-przełożony (jak w drugiej Fiszce), porównanie wiersza z sąsiednim wierszem tej samej tabeli po jakimś porządku (trzecia Fiszka) i wykrywanie duplikatów po kluczu naturalnym (czwarta Fiszka).

Jak to działa

`CROSS JOIN` i `SELF JOIN` nie są dwiema stronami tego samego mechanizmu — łączy je tylko to, że obie nazwy pojawiają się zwykle na tej samej stronie podręcznika, bo obie są mniej intuicyjne niż INNER czy LEFT JOIN. Mechanicznie `CROSS JOIN` to zwykłe złączenie bez warunku, liczone dokładnie tak jak każde inne, tylko że filtr `ON` po prostu nie istnieje. `SELF JOIN` mechanicznie nie różni się niczym od złączenia dwóch osobnych tabel — silnik nie wie ani nie musi wiedzieć, że `p` i `prz` w drugiej Fiszce wskazują na tę samą tabelę fizyczną; traktuje je jak dwa niezależne źródła wierszy, bo takie właśnie są z perspektywy planu wykonania, każde ze swoim aliasem.

Generowanie listy wartości bez tabeli źródłowej. Konstrukcja `VALUES (...), (...), (...)` użyta w pierwszej Fiszce jako źródło dni tygodnia nie działa identycznie we wszystkich silnikach:

Silnik	Lista wartości bez tabeli źródłowej
PostgreSQL	`VALUES ('a'), ('b')` wprost w `FROM`
SQL Server	`VALUES ('a'), ('b')` wprost w `FROM` (od SQL Server 2008)
MySQL / MariaDB	`VALUES ROW('a'), ROW('b')` jako tabela (od MySQL 8.0.19) albo `SELECT 'a' UNION ALL SELECT 'b'` wcześniej
Oracle	brak samodzielnego `VALUES` w `FROM` — zamiast tego `SELECT 'a' FROM dual UNION ALL SELECT 'b' FROM dual`

Oracle wymaga tu tabeli `DUAL` — jednowierszowej, jednokolumnowej tabeli systemowej istniejącej wyłącznie po to, żeby dało się napisać `SELECT` bez żadnej prawdziwej tabeli źródłowej. Zapytanie z pierwszej Fiszki przeniesione żywcem na Oracle nie zadziała bez tej zamiany.

Częste błędy

Zapomniany warunek na aliasach w SELF JOIN wykrywającym duplikaty. `JOIN klienci b ON a.nip = b.nip` bez `AND a.id < b.id` zwróci każdą parę dwa razy (raz jako `a`-`b`, raz jako `b`-`a`) i dodatkowo każdy wiersz sparowany sam ze sobą — bo `a.nip = b.nip` jest prawdą także wtedy, gdy `a` i `b` to ten sam fizyczny wiersz. Warunek nierówności na kluczu jest tu obowiązkowy, nie kosmetyczny.

Niejednoznaczne kolumny bez rozróżnienia aliasów. `SELECT imie FROM pracownicy p JOIN pracownicy prz ON ...` kończy się błędem niejednoznaczności identycznym jak przy dwóch różnych tabelach — silnik nie potraktuje `SELF JOIN` łagodniej tylko dlatego, że fizycznie jest to jedna tabela.

CROSS JOIN generujący więcej wierszy niż zamierzono. Siatka dat razy oddziały razy produkty, zamiast dwóch wymiarów, potrafi urosnąć do setek tysięcy wierszy, zanim ktoś zauważy, że trzeci wymiar wcale nie był potrzebny w tym raporcie. `CROSS JOIN` mnoży bez pytania — każdy dodatkowy wymiar mnoży, nie dodaje.

□□ Ćwiczenia

> Zadania sprawdzające.

Ćwiczenie 1 — hierarchia bez najwyższego szczebla

Zapytanie `SELECT p.imie, prz.imie AS szef FROM pracownicy p JOIN pracownicy prz ON p.przelozony_id = prz.id;` nie zwraca wiersza dla Ewy Kowalczyk. Dlaczego, i jak to poprawić, żeby Ewa też się pojawiła, z pustą kolumną `szef`?

Odpowiedź

Ewa ma `przelozony_id IS NULL`, a zwykły `JOIN` (INNER) wymaga dopasowania po obu stronach — `NULL` nigdy go nie da, więc wiersz Ewy odpada. Zamiana `JOIN` na `LEFT JOIN pracownicy prz ON p.przelozony_id = prz.id` zachowa Ewę w wyniku z `NULL`-em w kolumnie `szef`, dokładnie tak jak LEFT JOIN zachowuje klientów bez zamówień na stronie LEFT JOIN i RIGHT JOIN.

Ćwiczenie 2 — porównanie z kolegą z działu

Chcemy zestawić każdego pracownika z każdym innym pracownikiem z tego samego działu, bez porównywania pracownika z samym sobą i bez pokazywania tej samej pary dwa razy (raz jako A-B, raz jako B-A). Jakie dwa warunki musi mieć taki SELF JOIN?

Odpowiedź

```
SELECT a.imie AS pracownik_a, b.imie AS pracownik_b, a.dzial
FROM pracownicy a
JOIN pracownicy b ON a.dzial = b.dzial AND a.id < b.id;
```

Warunek `a.dzial = b.dzial` dobiera pary z tego samego działu, a `a.id < b.id` — dokładnie ten sam mechanizm co przy wykrywaniu duplikatów NIP w czwartej Fiszce — gwarantuje, że każda para pojawi się dokładnie raz, w jednym uporządkowanym kierunku, i że żaden pracownik nie sparuje się sam ze sobą.

Pytania kontrolne

1. Dlaczego CROSS JOIN nie potrzebuje klauzuli ON, skoro każde inne złączenie jej wymaga?

Odpowiedź

Bo `CROSS JOIN` z definicji nie filtruje niczego — zwraca cały iloczyn kartezjański, czyli dokładnie to, co pozostałe złączenia dopiero filtrują swoim warunkiem `ON`. Klauzula `ON` byłaby tu sprzeczna z sensem operacji, więc składnia jej po prostu nie przewiduje.

2. Ile razy trzeba wymienić nazwę tabeli `pracownicy` we frazie FROM/JOIN, żeby porównać każdego pracownika zarówno z jego przełożonym, jak i z kolegami z tego samego działu, w jednym zapytaniu?

Odpowiedź

Trzy razy, z trzema różnymi aliasami — raz jako pracownik główny, raz jako jego przełożony (złączenie po `przelozony_id`), raz jako kolega z działu (złączenie po `dzial`, z warunkiem wykluczającym porównanie pracownika z samym sobą). Liczba potrzebnych aliasów SELF JOIN rośnie wprost z liczbą niezależnych porównań, jakie zapytanie ma wykonać naraz.

Powiązane: Czym jest złączenie: iloczyn kartezjański i warunek, który go ogranicza · LEFT JOIN i RIGHT JOIN · Funkcje okna — OVER, PARTITION BY, ROW_NUMBER, RANK, LAG/LEAD · Podzapytania: skalarne, wierszowe, skorelowane, EXISTS

Źródła: ISO/IEC 9075-2:2016 § 7.7 · dokumentacja PostgreSQL, rozdział „Joins Between Tables" i „VALUES Lists"

Revision #46
Created 2026-09-24 22:42:00 UTC by Tomasz Konieczny
Updated 2026-10-26 14:11:03 UTC by Tomasz Konieczny