UPDATE: i dlaczego zawsze piszemy WHERE zanim napiszemy SET

> `UPDATE` zmienia wartości w wierszach, które już istnieją, a `WHERE` decyduje, które wiersze zostaną dotknięte — ta strona pokazuje składnię, aktualizację na podstawie danych z innej tabeli i nawyk, który chroni przed nadpisaniem całej tabeli przez jeden zapomniany warunek.

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

ZMIENIA SYSTEM

Aktualizacja jednej kolumny w wierszach spełniających warunek

```
UPDATE zamowienia SET status = 'zrealizowane' WHERE id = 104;
```

`SET` wskazuje, którą kolumnę zmieniamy i na jaką wartość, `WHERE` wskazuje, które wiersze dostają tę zmianę. Bez `WHERE` `UPDATE` dotyka każdego wiersza tabeli — składniowo `WHERE` jest opcjonalne, co czyni jego brak jednym z najbardziej kosztownych literówek w SQL.

Przykład:

```
UPDATE 1
```

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 · **SQL** = Structured Query Language: język zapytań do relacyjnych baz danych

ZMIENIA SYSTEM

Aktualizacja wielu kolumn naraz

```
UPDATE zamowienia SET status = 'anulowane', wartosc = 0.00 WHERE id = 105;
```

Lista przypisań po `SET`, oddzielona przecinkami, zmienia dowolną liczbę kolumn w jednym poleceniu — silnik stosuje wszystkie zmiany do każdego pasującego wiersza jednocześnie, w ramach jednej operacji, nie kolejno jedna po drugiej.

Przykład:

```
UPDATE 1
```

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

Aktualizacja na podstawie bieżącej wartości kolumny

```
UPDATE produkty SET cena = cena * 1.05 WHERE kategoria = 'elektronika';
```

Prawa strona `SET` może odwoływać się do bieżącej wartości tej samej albo innej kolumny tego samego wiersza — silnik czyta starą wartość `cena`, liczy nowe wyrażenie i dopiero wtedy zapisuje wynik, więc `cena = cena * 1.05` podnosi każdą cenę o pięć procent względem jej aktualnego stanu, a nie względem jednej wspólnej wartości.

Przykład:

```
UPDATE 3
```

Trzy produkty z kategorii `elektronika` — `Monitor 24 cale`, `Kabel HDMI 2m` i `Klawiatura mechaniczna` — dostają nową cenę naraz, każdy policzoną od własnej starej wartości.

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

Aktualizacja z danymi z innej tabeli: FROM (PostgreSQL)

```
UPDATE zamowienia z
SET wartosc = p.suma
FROM (SELECT zamowienie_id, SUM(ilosc * cena) AS suma FROM pozycje GROUP BY
zamowienie_id) p
WHERE z.id = p.zamowienie_id;
```

Klauzula `FROM` w `UPDATE` dołącza dodatkową tabelę albo podzapytanie, z którego bierzemy nowe wartości — tu przeliczamy `wartosc` zamówienia na podstawie sumy jego pozycji, zamiast trzymać ją jako ręcznie utrzymywaną kopię. Warunek łączący obie tabele stoi w `WHERE`, dokładnie tak samo jak przy niejawnej, starej składni złączenia opisanej na stronie INNER JOIN — tylko że tu jedna ze stron jest aktualizowana, nie tylko odczytywana.

Przykład:

```
UPDATE 4
```

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

Aktualizacja z danymi z innej tabeli: JOIN (SQL Server / MySQL)

```
UPDATE z
SET z.wartosc = p.suma
FROM zamowienia z
JOIN (SELECT zamowienie_id, SUM(ilosc * cena) AS suma FROM pozycje GROUP BY
zamowienie_id) p
    ON z.id = p.zamowienie_id;
```

SQL Server zapisuje ten sam pomysł przez `UPDATE ... FROM ... JOIN` — nazwa tabeli po `UPDATE` jest aliasem, a właściwe `FROM` z `JOIN`em określa źródło nowych wartości. MySQL

używa uproszczonej wersji bez osobnego `FROM`:

```
UPDATE zamowienia z
JOIN (SELECT zamowienie_id, SUM(ilosc * cena) AS suma FROM pozycje GROUP BY
zamowienie_id) p
    ON z.id = p.zamowienie_id
SET z.wartosc = p.suma;
```

Przykład (SQL Server):

```
(4 rows affected)
```

□□ **SQL** = Structured Query Language: język zapytań do relacyjnych baz danych ·**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 ·**join** = połącz dwa pliki tekstowe po wspólnej kolumnie, jak złączenie tabel w bazie danych

□□ ZMIENIA SYSTEM

Aktualizacja z danymi z innej tabeli: MERGE albo podzapytanie skorelowane (Oracle)

```
UPDATE zamowienia z
SET wartosc = (
    SELECT SUM(ilosc * cena) FROM pozycje p WHERE p.zamowienie_id = z.id
)
WHERE EXISTS (SELECT 1 FROM pozycje p WHERE p.zamowienie_id = z.id);
```

Oracle w rdzeniu SQL nie ma klauzuli `FROM` w `UPDATE` ani składni `JOIN` bezpośrednio w nim — ten sam efekt osiąga się podzapytaniem skorelowanym w `SET`, powtórzonym w `WHERE EXISTS` , żeby nie nadpisywać `wartosc` na `NULL` dla zamówień bez żadnej pozycji. Alternatywą jest pełny `MERGE`, opisany osobno na stronie MERGE / UPSERT — jak to robi każdy silnik, który tę samą logikę zapisuje jednym poleceniem bez powtarzania warunku dwa razy.

Przykład:

```
4 rows updated.
```

□□ **SQL** = Structured Query Language: język zapytań do relacyjnych baz danych ·**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

TYLKO ODCZYT

Cztery składnie aktualizacji z drugiej tabeli

Silnik	Składnia
PostgreSQL	`UPDATE ... SET ... FROM ... WHERE`
SQL Server	`UPDATE ... SET ... FROM ... JOIN ... ON`
MySQL / MariaDB	`UPDATE ... JOIN ... ON ... SET ...`
Oracle	podzapytanie skorelowane w `SET` albo `MERGE`

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

TYLKO ODCZYT

Nawyk bezpieczeństwa: SELECT z tym samym WHERE przed UPDATE

```
SELECT id, status, wartosc FROM zamowienia WHERE status = 'nowe' AND wartosc = 0;
```

Zanim napiszemy `UPDATE`, uruchamiamy `SELECT` z dokładnie tym samym warunkiem `WHERE`, jaki planujemy w `UPDATE`, i patrzymy na liczbę zwróconych wierszy oraz na same dane. Dopiero gdy ten wynik zgadza się z oczekiwaniem, zamieniamy `SELECT id, status, wartosc` na `UPDATE zamowienia SET ...`, zostawiając `WHERE` bez zmian. To jest najtańsze zabezpieczenie przed przypadkową aktualizacją tysięcy wierszy zamiast jednego — warunek, który wyglądał poprawnie w głowie, czasem okazuje się dopasowywać znacznie więcej wierszy niż zakładaliśmy, i lepiej się o tym dowiedzieć z `SELECT`a niż z `UPDATE`a.

Przykład:

```
 id | status |   wartosc
----+--------+---------
(0 rows)
```

Zero wierszy — w naszych danych żadne zamówienie o statusie `nowe` nie ma jednocześnie kwoty zero, więc bezpiecznie wiemy, że odpowiadający `UPDATE` też nie zmieni żadnego wiersza, zanim go w ogóle uruchomimy.

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

Transakcja z ROLLBACK jako siatka bezpieczeństwa

```
BEGIN;
UPDATE zamowienia SET status = 'anulowane' WHERE data < '2024-02-01';
```

Otwieramy transakcję jawnie, wykonujemy `UPDATE` i przed `COMMIT`em sprawdzamy jeszcze raz stan tabeli w tej samej sesji — jeśli liczba zmienionych wierszy albo ich zawartość nie zgadza się z oczekiwaniem, `ROLLBACK` cofa całą zmianę, jakby nigdy się nie wydarzyła, opisane szerzej na stronie BEGIN, COMMIT, ROLLBACK, SAVEPOINT. Dopóki nie padnie `COMMIT`, zmiana jest widoczna tylko w bieżącej sesji.

Przykład:

```
UPDATE 2
```

```
ROLLBACK;
```

Przykład:

```
ROLLBACK
```

Tabela wraca do stanu sprzed `UPDATE`a — dwa zamówienia, które przed chwilą zmieniły status, mają go z powrotem takim, jaki był przed transakcją.

◫◫ **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

Po co to istnieje

Pojęcie: aktualizacja. Zmiana wartości jednej lub więcej kolumn w wierszach, które już istnieją w tabeli — w przeciwieństwie do `INSERT`, który tworzy nowe wiersze, `UPDATE` nigdy nie zmienia liczby wierszy w tabeli, tylko ich zawartość. `WHERE` w `UPDATE` działa dokładnie tak samo jak `WHERE` w `SELECT`, opisane na stronie WHERE: filtrowanie wierszy — to ten sam predykat sprawdzany dla każdego wiersza z osobna, z tą samą trójwartościową logiką i tymi samymi operatorami.

Ta identyczność `WHERE` w obu poleceniach jest właśnie podstawą nawyku bezpieczeństwa opisanego w Fiszce: skoro `SELECT ... WHERE warunek` i `UPDATE ... SET ... WHERE warunek` filtrują wiersze identycznie, sprawdzenie jednego pozwala przewidzieć drugie z pełną pewnością, zanim cokolwiek się zmieni.

Jak to działa

`UPDATE` z klauzulą odwołującą się do drugiej tabeli — `FROM`, `JOIN` albo podzapytanie skorelowane — logicznie wykonuje dwa kroki naraz: znajduje pasujące pary wierszy między tabelą aktualizowaną a tabelą źródłową, dokładnie jak zwykłe złączenie, a potem dla każdej takiej pary zapisuje nowe wartości do wiersza po stronie aktualizowanej. Jeśli warunek złączenia dopasowuje więcej niż jeden wiersz źródłowy do jednego wiersza docelowego, wynik zależy od silnika — część z nich zapisuje wartość z dowolnie wybranego dopasowania, bez gwarancji, które z nich wygra, co jest jednym z powodów, dla których warunek złączenia w `UPDATE` powinien być równie jednoznaczny jak klucz główny czy klucz unikalny.

Transakcja wokół `UPDATE`a nie zmienia tego, co polecenie robi logicznie — zmienia tylko to, kiedy zmiana staje się trwała i widoczna dla innych sesji. Dopóki trwa transakcja, `UPDATE` trzyma blokadę na zmienionych wierszach i inne sesje, które chciałyby je zmienić, czekają albo widzą starą wersję danych, zależnie od poziomu izolacji — temat rozwinięty w rozdziale Anomalie współbieżności: brudny odczyt, niepowtarzalny odczyt, fantom.

Częste błędy

Zapomniany WHERE, aktualizacja całej tabeli.

```
UPDATE zamowienia SET status = 'anulowane';
```

```
UPDATE 8
```

Osiem wierszy zamiast jednego — silnik nie ostrzega przed brakiem `WHERE`, bo składniowo jest to poprawne polecenie o dokładnie takiej treści, jaka została napisana. Jedyną obroną jest nawyk opisany w Fiszce: sprawdzenie `SELECT`em przed zamianą na `UPDATE`.

Aktualizacja typu niezgodnego z kolumną.

```
ERROR:  invalid input syntax for type numeric: "brak danych"
LINE 1: UPDATE zamowienia SET wartosc = 'brak danych' WHERE id = 1...
                                        ^
```

Wpisanie tekstu do kolumny liczbowej — częste przy budowaniu zapytania ze sklejanych fragmentów tekstu w aplikacji, gdzie brakująca wartość zamienia się w literał `'brak danych'` zamiast w `NULL`.

Aktualizacja przez JOIN dopasowująca więcej niż jeden wiersz źródłowy.

Gdy podzapytanie albo złączenie w `UPDATE ... FROM` zwraca więcej niż jeden wiersz dla tego samego wiersza docelowego, SQL Server i MySQL wykonają aktualizację z dowolnie wybranym dopasowaniem bez błędu, a PostgreSQL zrobi to samo — żaden z czterech silników nie sygnalizuje tej sytuacji automatycznie. Bezpiecznym nawykiem jest zagregowanie źródła (`GROUP BY`, jak w przykładzie z Fiszki) albo dodanie warunku, który gwarantuje dokładnie jedno dopasowanie na wiersz docelowy.

□□ Ćwiczenia

> Zadania sprawdzające.

Ćwiczenie 1 — SELECT przed UPDATE

Chcemy podnieść cenę wszystkich produktów z kategorii `meble` o dziesięć procent. Zanim napiszemy `UPDATE`, jakim zapytaniem `SELECT` sprawdzamy, które wiersze zostaną zmienione, i ile ich będzie w naszych danych z Fiszki tej strony i strony INNER JOIN?

Odpowiedź

`SELECT id, nazwa, cena FROM produkty WHERE kategoria = 'meble';` — w naszych danych to trzy produkty: `Biurko regulowane` (1100.00), `Fotel biurowy` (890.00) i `Lampka biurkowa` (90.00). Dopiero po zobaczeniu tych trzech wierszy zamieniamy `SELECT id, nazwa, cena` na `UPDATE produkty SET cena = cena * 1.10`, zostawiając `WHERE kategoria = 'meble'` bez zmian.

Pytania kontrolne

1. Dlaczego `UPDATE produkty SET cena = 100 WHERE kategoria = 'meble'` różni się od `UPDATE produkty SET cena = cena * 1.10 WHERE kategoria = 'meble'` w sposób istotny dla poprawności danych?

Odpowiedź

Pierwsze polecenie nadpisuje cenę każdego produktu z kategorii `meble` tą samą, stałą wartością, tracąc bezpowrotnie informację o tym, ile kosztował wcześniej każdy z nich osobno. Drugie liczy nową cenę względem starej wartości każdego wiersza z osobna, więc różnice cenowe między produktami zostają zachowane proporcjonalnie.

2. Co się stanie, jeśli transakcja z `UPDATE` zostanie przerwana awarią połączenia, zanim padnie `COMMIT` albo `ROLLBACK`?

Odpowiedź

Silnik traktuje przerwane połączenie jak niejawny `ROLLBACK` — transakcja, która nigdy nie doczekała się `COMMIT`u, nie zostaje zatwierdzona, więc zmiana wykonana przez `UPDATE` nie staje się trwała i inne sesje jej nie widzą. Dokładny mechanizm zależy od silnika, ale efekt końcowy — brak zmiany w danych — jest wspólny dla wszystkich czterech.

Powiązane: WHERE: filtrowanie wierszy · INSERT: jeden wiersz, wiele wierszy, INSERT ... SELECT · MERGE / UPSERT — jak to robi każdy silnik · BEGIN, COMMIT, ROLLBACK, SAVEPOINT

Źródła: dokumentacja PostgreSQL, rozdział „UPDATE” · dokumentacja Microsoft, „UPDATE (Transact-SQL)” · dokumentacja MySQL, „UPDATE Statement” · dokumentacja Oracle, „UPDATE” · ISO/IEC 9075-2:2016 § 14.14

Revision #47
Created 2026-09-24 22:42:12 UTC by Tomasz Konieczny
Updated 2026-10-26 14:11:15 UTC by Tomasz Konieczny