DELETE vs TRUNCATE vs DROP

> Wszystkie trzy polecenia potocznie nazywa się „usunięciem", ale usuwają zupełnie różne rzeczy — `DELETE` wiersze, `TRUNCATE` całą zawartość tabeli naraz, `DROP` samą tabelę razem z jej definicją — a różnica ma konsekwencje dla wydajności, wycofywalności i uprawnień.

Skróty użyte na tej stronie (3)

Skrót	Co znaczy
DDL	Data Definition Language: polecenia SQL zmieniające strukturę (CREATE, ALTER, DROP)
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

OSTROŻNIE · NIEODWRACALNE

Usunięcie wybranych wierszy: DELETE z warunkiem

```
DELETE FROM zamowienia WHERE status = 'anulowane';
```

`DELETE` usuwa wiersze spełniające warunek `WHERE`, jeden po drugim, z pełnym logowaniem każdej usuniętej krotki do dziennika transakcyjnego — dokładnie tak, jak przy `UPDATE`, tylko że efektem jest zniknięcie wiersza, nie zmiana jego zawartości.

Przykład:

```
DELETE 2
```

Dwa zamówienia o statusie `anulowane` — 103 i 108 — znikają z tabeli `zamowienia`.

⚠ OSTROŻNIE · NIEODWRACALNE

Usunięcie wszystkich wierszy: DELETE bez warunku

```
DELETE FROM produkty_archiwalne;
```

`DELETE` bez `WHERE` usuwa każdy wiersz tabeli, ale wciąż robi to wiersz po wierszu, z pełnym logowaniem — to jest różnica względem `TRUNCATE`, opisana niżej, mimo że efekt końcowy na pierwszy rzut oka wygląda identycznie: pusta tabela.

Przykład:

```
DELETE 3
```

⚠ OSTROŻNIE · NIEODWRACALNE

Usuwanie partiami przy dużych tabelach

```
DELETE FROM logi_aplikacji WHERE data_zdarzenia < '2023-01-01' LIMIT 10000;
```

Jednorazowe usunięcie milionów wierszy jednym poleceniem `DELETE` bez warunku ograniczającego liczbę rozdyma dziennik transakcyjny do rozmiaru całej usuwanej partii naraz, bo silnik musi zapisać informację o każdym usuniętym wierszu, zanim zatwierdzi transakcję — a przy bardzo dużym dzienniku rośnie też czas potencjalnego `ROLLBACK` a i ryzyko zapełnienia miejsca na dysku przeznaczonego na logi. `LIMIT` w `DELETE` (odpowiednik w standardzie SQL to `FETCH FIRST ... ROWS ONLY`, w Oracle `ROWNUM`, w SQL Server `TOP`) ogranicza liczbę wierszy usuwanych jednym poleceniem, dzięki czemu każda partia zatwierdza się i zwalnia miejsce w dzienniku, zanim ruszy kolejna.

Przykład:

```
DELETE 10000
```

Polecenie powtarzamy w pętli, aż kolejne wywołanie zwróci `DELETE 0` — sygnał, że nie zostało już nic do usunięcia spełniającego warunek.

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

⚠ OSTROŻNIE · NIEODWRACALNE

Usunięcie całej zawartości tabeli: TRUNCATE

```
TRUNCATE TABLE zamowienia_robocze;
```

`TRUNCATE` nie usuwa wierszy pojedynczo — zwalnia strony danych zajmowane przez tabelę jako całość, bez sprawdzania i logowania każdego wiersza z osobna. Efekt jest ten sam co `DELETE` bez `WHERE` — pusta tabela — ale operacja jest rzędy wielkości szybsza na dużych tabelach, bo koszt nie zależy od liczby wierszy.

Przykład:

```
TRUNCATE TABLE
```

⚠ OSTROŻNIE · NIEODWRACALNE

TRUNCATE i klucze obce

```
TRUNCATE TABLE klienci;
```

```
ERROR:  cannot truncate a table referenced in a foreign key constraint
DETAIL:  Table "zamowienia" references "klienci".
HINT:  Truncate table "zamowienia" too, or use TRUNCATE ... CASCADE.
```

Tabela `klienci` jest stroną nadrzędną klucza obcego z tabeli `zamowienia` — `TRUNCATE` odmawia usunięcia jej zawartości, dopóki istnieją wiersze, które się do niej odwołują, bo w przeciwieństwie do `DELETE` nie sprawdza tego wiersz po wierszu, tylko z góry zakłada pustkę po obu stronach. `TRUNCATE ... CASCADE` w PostgreSQL usuwa też zawartość tabel zależnych, ale to operacja nieodwracalna bez transakcji obejmującej obie tabele naraz.

⚠ OSTROŻNIE · NIEODWRACALNE

Usunięcie tabeli razem z definicją: DROP TABLE

```
DROP TABLE produkty_archiwalne;
```

`DROP TABLE` usuwa nie tylko dane, ale całą definicję tabeli — kolumny, ograniczenia, indeksy związane wyłącznie z tą tabelą. Po wykonaniu tego polecenia tabela przestaje istnieć w katalogu systemowym, opisanym na stronie Katalog systemowy: skąd baza wie, co w sobie

ma, tak jakby nigdy nie została utworzona.

Przykład:

```
DROP TABLE
```

◫◫ OSTROŻNIE · NIEODWRACALNE

DROP TABLE i CASCADE

```
DROP TABLE klienci CASCADE;
```

```
NOTICE:  drop cascades to constraint fk_zamowienia_klient on table zamowienia
DROP TABLE
```

`CASCADE` przy `DROP TABLE` usuwa razem z tabelą również obiekty, które od niej zależą — tu klucz obcy w `zamowienia`, który odwoływał się do `klienci`. Sama tabela `zamowienia` zostaje, ale traci ograniczenie, które gwarantowało spójność jej kolumny `klient_id`. Bez `CASCADE` to samo polecenie kończy się błędem odmowy, dokładnie jak `TRUNCATE` bez `CASCADE` powyżej.

Trzy różne operacje pod jedną potoczną nazwą

Cecha	DELETE	TRUNCATE	DROP
Co usuwa	wybrane albo wszystkie wiersze	całą zawartość tabeli naraz	tabelę razem z definicją
Logowanie per wiersz	tak	nie	nie dotyczy (usuwa metadane)
Wycofywalne transakcją (ROLLBACK)	tak	w PostgreSQL i SQL Server tak, w MySQL z InnoDB tak, w Oracle nie w pełni bez specjalnych ustawień	tak w PostgreSQL i SQL Server, w Oracle DDL zatwierdza się niejawnie
Resetuje licznik autoinkrementacji	nie	domyślnie w wielu silnikach tak	dotyczy, licznik znika razem z tabelą
Odpala wyzwalacze wierszowe	tak	nie	nie dotyczy

Cecha	DELETE	TRUNCATE	DROP
Respektuje klucze obce	tak, wiersz po wierszu	wymaga braku zależności albo CASCADE	wymaga braku zależności albo CASCADE
Wymagane uprawnienie	DELETE na tabeli	zwykle DROP albo osobne uprawnienie TRUNCATE	DROP na tabeli

Uważaj

- Wiersz „wycofywalne transakcją" jest tu tabelą różnic, nie zdaniem — Oracle domyślnie traktuje `TRUNCATE` jako operację DDL, która zatwierdza bieżącą transakcję niejawnie przed wykonaniem i nie da się jej cofnąć `ROLLBACK`iem, w odróżnieniu od PostgreSQL, gdzie `TRUNCATE` w transakcji wycofuje się tak samo jak `DELETE`.
- Resetowanie licznika autoinkrementacji przy `TRUNCATE` też jest zależne od silnika i wersji — temat wraca w pełni na stronie Sekwencje, IDENTITY, AUTO_INCREMENT.

Po co to istnieje

Pojęcie: usunięcie danych a usunięcie struktury. Trzy polecenia z tej strony różnią się tym, na jakim poziomie działają: `DELETE` na poziomie pojedynczych wierszy (instancji relacji), `TRUNCATE` na poziomie całej zawartości tabeli naraz, ale wciąż zostawiając samą definicję tabeli nietkniętą, `DROP` na poziomie schematu relacji, czyli definicji zapisanej w `CREATE TABLE` — usuwając ją, usuwa niejako i wszystkie wiersze, które by ta definicja mogła kiedykolwiek zawierać. To rozróżnienie, schemat relacji kontra jej instancja, wraca ze strony Model relacyjny: tabela, wiersz, kolumna, dziedzina.

Jak to działa

`DELETE` musi sprawdzić warunek `WHERE` dla każdego wiersza z osobna i zapisać do dziennika transakcyjnego informację wystarczającą do odtworzenia usuniętego wiersza przy ewentualnym `ROLLBACK`u — stąd koszt rosnący liniowo z liczbą usuwanych wierszy i możliwość odpalenia wyzwalaczy `AFTER DELETE` czy `BEFORE DELETE` dla każdego z nich osobno. `TRUNCATE` pomija ten mechanizm: zamiast usuwać wiersze, zwalnia całe strony danych zajmowane przez tabelę, więc czas wykonania nie zależy od tego, czy tabela ma sto wierszy czy sto milionów.

Ta różnica tłumaczy też, dlaczego `TRUNCATE` nie odpala wyzwalaczy wierszowych — nie ma pojedynczych wierszy do przetworzenia, jest tylko decyzja „zwolnij te strony danych". Z tego samego powodu `TRUNCATE` w wielu silnikach resetuje licznik kolumny z automatycznie generowaną wartością do stanu początkowego, podczas gdy `DELETE` zostawia go tam, gdzie był — bo z punktu widzenia silnika `DELETE` nigdy nie zresetował niczego poza samymi danymi, a `TRUNCATE` traktuje tabelę tak, jakby dopiero co ją utworzono.

Częste błędy

TRUNCATE zablokowany przez klucz obcy bez CASCADE.

```
ERROR:  cannot truncate a table referenced in a foreign key constraint
DETAIL:  Table "zamowienia" references "klienci".
```

Patrz Fiszka wyżej — naprawa to `TRUNCATE ... CASCADE`, albo najpierw opróżnienie tabel zależnych, albo tymczasowe wyłączenie ograniczeń opisane na stronie Ograniczenia: NOT NULL, UNIQUE, CHECK, FOREIGN KEY, DEFAULT.

Oczekiwanie, że ROLLBACK cofnie TRUNCATE w Oracle.

```
BEGIN
    DELETE FROM zamowienia WHERE status = 'anulowane';
    -- za chwilę TRUNCATE w tej samej "transakcji" logicznej
END;
```

W Oracle `TRUNCATE` jest poleceniem DDL, które zatwierdza (`COMMIT`) każdą wcześniej otwartą transakcję, zanim samo się wykona, i nie da się go cofnąć `ROLLBACK`iem — w przeciwieństwie do PostgreSQL i SQL Server, gdzie `TRUNCATE` wewnątrz jawnej transakcji wycofuje się tak samo jak każde inne polecenie. Kod przenoszony między silnikami, który zakłada wycofywalność `TRUNCATE`, wymaga sprawdzenia tego zachowania osobno dla Oracle.

DROP TABLE bez CASCADE na tabeli z zależnościami.

```
ERROR:  cannot drop table klienci because other objects depend on it
DETAIL:  constraint fk_zamowienia_klient on table zamowienia depends on table klienci
HINT:  Use DROP ... CASCADE to drop the dependent objects too.
```

Silnik odmawia usunięcia tabeli, dopóki inne obiekty się do niej odwołują — dokładnie ta sama logika co przy `TRUNCATE`, tylko na poziomie metadanych zamiast danych.

DELETE bez WHERE na produkcyjnej tabeli o milionach wierszy.

Jednorazowy `DELETE FROM logi_aplikacji;` bez `WHERE` i bez podziału na partie potrafi zablokować tabelę na czas trwania całej operacji i rozdąć dziennik transakcyjny do rozmiaru całej tabeli naraz — na dużych tabelach właściwym narzędziem jest albo `TRUNCATE` (jeśli usuwamy rzeczywiście wszystko i nie potrzebujemy wycofywalności per wiersz), albo `DELETE` partiami z `LIMIT`, opisane w Fiszce.

□□ Ćwiczenia

> Zadania sprawdzające.

Ćwiczenie 1 — właściwe narzędzie do zadania

Mamy tabelę `sesje_tymczasowe` z milionem wierszy, którą chcemy całkowicie wyczyścić raz w tygodniu w oknie serwisowym, bez potrzeby cofania tej operacji. Które polecenie — `DELETE` bez `WHERE` czy `TRUNCATE` — jest tu właściwym wyborem i dlaczego?

Odpowiedź

`TRUNCATE` — efekt końcowy (pusta tabela) jest identyczny jak przy `DELETE` bez `WHERE`, ale operacja jest znacznie szybsza na dużej tabeli, bo nie loguje każdego wiersza z osobna, a skoro nie potrzebujemy możliwości cofnięcia pojedynczych wierszy ani odpalenia wyzwalaczy, nic nie tracimy, rezygnując z mechanizmu `DELETE`.

Pytania kontrolne

1. Dlaczego `TRUNCATE` jest szybszy od `DELETE FROM tabela;` bez warunku, mimo że oba dają tę samą pustą tabelę na końcu?

Odpowiedź

`DELETE` musi przetworzyć i zalogować każdy wiersz z osobna, żeby operacja była wycofywalna i żeby mogły odpalić się wyzwalacze wierszowe. `TRUNCATE` zwalnia całe strony danych zajmowane przez tabelę naraz, bez przetwarzania pojedynczych wierszy, więc jego koszt nie rośnie wraz z liczbą wierszy w tabeli.

2. Co różni `DROP TABLE` od `TRUNCATE TABLE` w sensie tego, co zostaje po wykonaniu polecenia?

Odpowiedź

Po `TRUNCATE` tabela nadal istnieje, pusta, z tą samą definicją kolumn, indeksów i ograniczeń co wcześniej — można od razu zacząć wstawiać do niej nowe wiersze. Po `DROP TABLE` sama definicja tabeli znika z katalogu systemowego; żeby czegokolwiek do niej wstawić, trzeba ją odtworzyć przez `CREATE TABLE` od nowa.

Powiązane: UPDATE: i dlaczego zawsze piszemy WHERE zanim napiszemy SET · Ograniczenia: NOT NULL, UNIQUE, CHECK, FOREIGN KEY, DEFAULT · Sekwencje, IDENTITY, AUTO_INCREMENT · BEGIN, COMMIT, ROLLBACK, SAVEPOINT

Źródła: dokumentacja PostgreSQL, rozdział „DELETE”, „TRUNCATE” i „DROP TABLE” · dokumentacja Microsoft, „TRUNCATE TABLE (Transact-SQL)” · dokumentacja Oracle, „TRUNCATE TABLE” · ISO/IEC 9075-2:2016 § 14.7, § 14.8, § 11.31

Revision #46
Created 2026-09-24 22:42:14 UTC by Tomasz Konieczny
Updated 2026-10-26 14:11:17 UTC by Tomasz Konieczny