Skąd się wziął SQL i dlaczego każdy silnik mówi nim inaczej

> SQL ma jeden standard i osiem dialektów, a ta strona tłumaczy, skąd się wzięła ta rozbieżność i jak rozpoznać, które polecenie należy do której z czterech grup języka.

Skróty użyte na tej stronie (10)

Skrót	Co znaczy
ACM	AWS Certificate Manager: usługa wydająca i odnawiająca certyfikaty TLS dla równoważników, CloudFront i API Gateway
ANSI	American National Standards Institute: tu chodzi o kody sterujące kolorem i pozycją kursora w terminalu
DDL	Data Definition Language: polecenia SQL zmieniające strukturę (CREATE, ALTER, DROP)
DML	Data Manipulation Language: polecenia SQL zmieniające dane (INSERT, UPDATE, DELETE)
IBM	International Business Machines: firma, w której laboratorium powstał model relacyjny i pierwszy działający silnik SQL
ISO	International Organization for Standardization: organizacja, która wydaje normy techniczne używane na całym świecie
JSON	JavaScript Object Notation: tekstowy format danych z klamrami i cudzysłowami
SQL	Structured Query Language: język zapytań do relacyjnych baz danych
T-SQL	Transact-SQL: dialekt SQL używany w Microsoft SQL Server, wywodzący się z języka zapytań oryginalnego silnika Sybase
XML	eXtensible Markup Language: format tekstowy do zapisu ustrukturyzowanych danych w znacznikach, tutaj używany przez PowerShell do zapisu zrzutu obiektów poleceniem Export-Clixml

Fiszka

□□ ZMIENIA SYSTEM

Rozpoznanie polecenia definiującego strukturę: DDL

```
CREATE TABLE test_ddl (id integer);
```

Polecenie tworzy, zmienia albo usuwa obiekt w bazie — tabelę, indeks, widok, schemat. Silnik odpowiada komunikatem o wykonanej operacji, nie zestawem wierszy.

Przykład:

```
CREATE TABLE
```

□□ **DDL** = Data Definition Language: polecenia SQL zmieniające strukturę (CREATE, ALTER, DROP)

□□ TYLKO ODCZYT

Rozpoznanie polecenia operującego na danych: DML

```
SELECT nazwa, miasto FROM klienci WHERE miasto = 'Warszawa';
```

Polecenie czyta albo zmienia zawartość tabel, nie ich strukturę. `SELECT`, `INSERT`, `UPDATE` i `DELETE` należą do tej samej grupy, mimo że tylko jedno z nich zwraca wiersze.

Przykład:

```
   nazwa                | miasto
------------------------+-----------
 Nowak i Wspólnicy Sp. z o.o. | Warszawa
(1 row)
```

□□ **DML** = Data Manipulation Language: polecenia SQL zmieniające dane (INSERT, UPDATE, DELETE)

□□ ZMIENIA SYSTEM

Rozpoznanie polecenia kontroli dostępu: DCL

```
GRANT SELECT ON zamowienia TO nazwa_usera;
```

Polecenie nadaje albo odbiera uprawnienia do obiektów bazy. Nie rusza ani struktury, ani danych — rusza wyłącznie to, kto ma do nich dostęp.

Przykład:

```
GRANT
```

□□ TYLKO ODCZYT

Rozpoznanie polecenia kontroli transakcji: TCL

```
COMMIT;
```

Polecenie zamyka albo cofa grupę zmian zebranych w jednej transakcji. `COMMIT`, `ROLLBACK` i `SAVEPOINT` do tej strony wracają jeszcze w rozdziale o transakcjach — tu liczy się tylko to, że stoją osobno od DML.

Przykład:

```
COMMIT
```

□□ **DML** = Data Manipulation Language: polecenia SQL zmieniające dane (INSERT, UPDATE, DELETE)

□□ OSTROŻNIE · NIEODWRACALNE

Rozpoznanie polecenia na granicy dwóch grup: TRUNCATE

```
TRUNCATE TABLE test_ddl;
```

Polecenie wygląda jak operacja na danych — usuwa wszystkie wiersze z tabeli, tak jak `DELETE` bez `WHERE`. Formalnie jednak większość silników traktuje je jako DDL: nie loguje usunięcia wiersz po wierszu, tylko zwalnia zajmowane przez tabelę strony danych naraz, przez co działa dużo szybciej niż `DELETE`, ale w niektórych silnikach nie da się go cofnąć poleceniem `ROLLBACK` wewnątrz otwartej transakcji.

Przykład:

```
TRUNCATE TABLE
```

□□ **DDL** = Data Definition Language: polecenia SQL zmieniające strukturę (CREATE, ALTER, DROP)

□□ TYLKO ODCZYT

Wszystkie grupy poleceń

Grupa	Rozwinięcie	Przykładowe polecenia
DDL	Data Definition Language	`CREATE`, `ALTER`, `DROP`, `TRUNCATE`
DML	Data Manipulation Language	`SELECT`, `INSERT`, `UPDATE`, `DELETE`, `MERGE`
DCL	Data Control Language	`GRANT`, `REVOKE`
TCL	Transaction Control Language	`COMMIT`, `ROLLBACK`, `SAVEPOINT`

> **Uważaj**
>
> - `TRUNCATE` wygląda jak operacja na danych, ale formalnie jest DDL — część silników nie da się jej cofnąć poleceniem `ROLLBACK`.
> - Różne książki zaliczają `MERGE` raz do DML, raz do osobnej kategorii — w praktyce liczy się to, że modyfikuje dane, nie strukturę.

DDL = Data Definition Language: polecenia SQL zmieniające strukturę (CREATE, ALTER, DROP) · **DML** = Data Manipulation Language: polecenia SQL zmieniające dane (INSERT, UPDATE, DELETE)

Po co to istnieje

> **Pojęcie: SQL.** Structured Query Language — deklaratywny język do opisywania, jakie dane mają się znaleźć w wyniku, oraz do definiowania i kontrolowania struktury, w której te dane leżą. Deklaratywny znaczy tyle, że zapytanie opisuje pożądany wynik, a nie kroki prowadzące do jego policzenia — do tego wracamy w SELECT: kolejność zapisu a kolejność wykonania.

Historia zaczyna się w 1970 roku, kiedy Edgar F. Codd, pracownik IBM, opublikował pracę opisującą model relacyjny — dane jako zbiory wierszy w tabelach powiązane przez wartości, a nie przez wskaźniki i fizyczne ścieżki dostępu, jak w ówczesnych bazach hierarchicznych i sieciowych. Ówczesny standard przemysłowy, opracowany przez organizację CODASYL, każdy sposób dotarcia do rekordu zapisywał wprost w strukturze bazy — program pytający o dane musiał znać fizyczną drogę do nich, wskaźnik po wskaźniku, rekord po rekordzie. Zmiana tej struktury oznaczała przepisanie każdego programu, który z niej korzystał. Model Codda odrywał sposób pytania o dane od sposobu ich fizycznego przechowywania — stąd zresztą wynika deklaratywność SQL, wyjaśniona niżej.

IBM uruchomił projekt badawczy System R, żeby sprawdzić, czy taki model da się w ogóle sensownie zaimplementować. Do zapytań do tego systemu Donald Chamberlin i Raymond Boyce zaprojektowali w 1974 roku język SEQUEL — Structured English Query Language, opisany w publikacji „SEQUEL: A Structured English Query Language" na konferencji ACM SIGMOD.

Nazwa SEQUEL nie przetrwała z powodów, które z bazami danych nie mają nic wspólnego: skrót był już zastrzeżony jako znak towarowy przez brytyjską firmę lotniczą Hawker Siddeley. IBM zmienił nazwę na SQL, zachowując wymowę zbliżoną do oryginału — stąd w wielu środowiskach nadal słyszy się „sequel" zamiast litera po literze.

System R był projektem badawczym i nie trafił do sprzedaży. Pierwszą komercyjną bazą danych z językiem SQL nie był produkt IBM, tylko Oracle w wersji 2, wydany w 1979 roku przez firmę Relational Software Inc. — cztery lata przed tym, jak sam IBM wypuścił DB2. Firma, która wymyśliła model relacyjny i pierwszy dialekt SQL, przegrała wyścig na rynek z konkurentem czytającym te same publikacje badawcze.

Równolegle z System R powstawał na Uniwersytecie Kalifornijskim w Berkeley konkurencyjny projekt badawczy, Ingres, z własnym językiem zapytań QUEL. Część badaczy tamtych lat uważała QUEL za czystszy składniowo niż SQL. Rynek zdecydował inaczej — to wokół SQL zbudowały swoje produkty Oracle i sam IBM, więc to SQL stał się w latach osiemdziesiątych faktycznym standardem branżowym, zanim jeszcze doczekał się standardu formalnego. Zwycięstwo SQL nad QUEL bywa przywoływane jako wczesny przykład tego, że o dominującym standardzie decyduje siła rynkowa produktów, które go używają, nie techniczna elegancja samego języka.

> **Pojęcie: standaryzacja.** Proces, w którym niezależna organizacja spisuje wspólną definicję języka, żeby polecenia napisane pod jeden silnik dało się — przynajmniej w teorii — uruchomić na innym. American National Standards Institute opublikował pierwszy standard SQL w 1986 roku, znany jako SQL-86 albo SQL-1. International Organization for Standardization przejęła standard rok później. Od tamtej pory standard był rewidowany regularnie: SQL-89 (drobne poprawki), SQL-92 (duże rozszerzenie, do dziś punkt odniesienia dla „podstawowej zgodności"), SQL:1999 (zapytania rekurencyjne, wyzwalacze, typy obiektowe), SQL:2003 (funkcje okna, kolumny generowane, XML), SQL:2006, SQL:2008, SQL:2011 (tabele czasowe), SQL:2016 (JSON, dopasowywanie wzorców w wierszach), SQL:2019 (dane wielowymiarowe) i SQL:2023 (typy JSON natywne, `GREATEST` / `LEAST` w standardzie).

Kolejne wydania standardu w skrócie:

Rok	Nazwa	Kluczowe dodatki
1986	SQL-86 / SQL-1	pierwszy standard, przyjęty przez ANSI
1989	SQL-89	integralność referencyjna
1992	SQL-92	duże rozszerzenie, do dziś punkt odniesienia zgodności

Rok	Nazwa	Kluczowe dodatki
1999	SQL:1999	zapytania rekurencyjne (`WITH RECURSIVE`), wyzwalacze, typy obiektowe
2003	SQL:2003	funkcje okna, kolumny generowane, integracja z XML
2006	SQL:2006	dalsza integracja z XML
2008	SQL:2008	`TRUNCATE` w standardzie, `INSTEAD OF` triggers
2011	SQL:2011	tabele czasowe (temporal)
2016	SQL:2016	JSON, dopasowywanie wzorców w wierszach
2019	SQL:2019	dane wielowymiarowe (MDA)
2023	SQL:2023	natywny typ JSON, `GREATEST`/`LEAST`

Standard nie jest jednym monolitem, który silnik albo wspiera, albo nie. Dzieli się na Core SQL — minimalny zestaw funkcji, bez którego producent nie ma prawa nazwać swojego produktu zgodnym z danym wydaniem standardu — oraz dziesiątki opcjonalnych pakietów funkcji, z których każdy silnik implementuje inny podzbiór. Dlatego pytanie „czy silnik jest zgodny z SQL:2016” nie ma sensownej odpowiedzi tak/nie — sensowne jest tylko pytanie o zgodność z konkretną, nazwaną funkcją standardu, na przykład z oknami analitycznymi z SQL:2003.

Jak to działa

□□ Rozwiń: jak to działa pod spodem

Standard opisuje rdzeń języka — składnię, którą każdy silnik zobowiązuje się rozumieć podobnie. Żaden producent nie implementuje standardu w całości i żaden nie poprzestaje na standardzie. Powód jest ekonomiczny: silnik, który różni się tylko szybkością od konkurencji, trudno sprzedać, więc każdy producent dokłada własne funkcje, własne typy danych i własną składnię do rzeczy, których standard nie przewidział w momencie powstawania silnika — bo standard bywa młodszy od funkcji, którą opisuje, albo starszy o dekadę od potrzeby, która się dopiero pojawiła.

> **Pojęcie: dialekt.** Wariant SQL właściwy dla konkretnego silnika — nadzbiór rdzenia standardu rozszerzony o funkcje, składnię i zachowania, których nie ma nigdzie indziej. T-SQL w SQL Serverze, PL/pgSQL w PostgreSQL i PL/SQL w Oracle to trzy różne dialekty,

> a każdy z nich zawiera więcej niż sam standard SQL.

Rozbieżność widać najwyraźniej tam, gdzie standard w ogóle nie narzucił jednego rozwiązania albo narzucił je już po tym, jak silniki zdążyły wypracować własne.

Operacja	PostgreSQL	MySQL / MariaDB	SQL Server (T-SQL)	Oracle
Ograniczenie liczby wierszy	`LIMIT 5`	`LIMIT 5`	`SELECT TOP 5 ...`	`FETCH FIRST 5 ROWS ONLY` (12c+)
Konkatenacja tekstu	`||`	`CONCAT(a, b)`	`+`	`||`
Bieżąca data i czas	`now()` / `CURRENT_TIMESTAMP`	`NOW()`	`GETDATE()`	`SYSDATE`
Rzutowanie typu	`x::integer` / `CAST(x AS integer)`	`CAST(x AS SIGNED)`	`CAST(x AS INT)`	`CAST(x AS NUMBER)`
Sprawdzenie wersji silnika	`SELECT version();`	`SELECT VERSION();`	`SELECT @@VERSION;`	`SELECT * FROM v$version;`

Nawet coś tak podstawowego jak sprawdzenie, na jakiej wersji silnika w ogóle pracujemy, nie ma wspólnej składni — w T-SQL to zmienna systemowa poprzedzona podwójnym `@@`, w Oracle to zapytanie do widoku systemowego zwracającego kilka wierszy zamiast jednego tekstu.

Rozdział ORDER BY, LIMIT, TOP, ROWNUM, FETCH: porcjowanie wyniku wraca do pierwszego wiersza tej tabeli ze szczegółami i przypadkami brzegowymi.

Ta sama rozbieżność dotyczy sposobu cytowania identyfikatorów — nazw tabel i kolumn, które kolidują ze słowem kluczowym albo zawierają wielkie litery czy spacje.

Silnik	Cytowanie identyfikatora
PostgreSQL	`"NazwaKolumny"`
MySQL / MariaDB	`` `NazwaKolumny` ``
SQL Server	`[NazwaKolumny]`
Oracle	`"NazwaKolumny"`

PostgreSQL i Oracle zgadzają się co do znaku cudzysłowu, bo obaj trzymają się tu litery standardu — SQL Server i MySQL poszły własną drogą, każdy w inną stronę.

Częste błędy

Traktowanie ANSI SQL jak języka w pełni przenośnego. Zapytanie napisane i przetestowane na jednym silniku po przeniesieniu na drugi potrafi nie zadziałać wcale, zadziałać z innym wynikiem albo zadziałać wolniej — konkatenacja przez `+` w T-SQL po przeniesieniu do PostgreSQL nie jest błędem składniowym, tylko operatorem dodawania, który dla dwóch tekstów zgłasza błąd typu.

Zakładanie, że NULL zachowuje się wszędzie tak samo. Oracle traktuje pusty ciąg znaków w kolumnie `VARCHAR2` jako `NULL`, podczas gdy PostgreSQL i MySQL rozróżniają pusty ciąg od braku wartości. Strona NULL: trzecia wartość logiczna i pułapki, które z niej wynikają opisuje tę pułapkę osobno.

Kopiowanie cudzysłowu identyfikatorów bez zmiany. PostgreSQL cytuje identyfikatory podwójnym cudzysłowem, MySQL domyślnie akceptuje odwrócony apostrof, a SQL Server nawiasy kwadratowe — zapytanie z `[NazwaTabeli]` przeniesione żywcem do PostgreSQL kończy się błędem składni, nie ostrzeżeniem.

Zakładanie identycznych nazw funkcji wbudowanych. Funkcja robiąca dokładnie to samo bywa dostępna pod inną nazwą w każdym silniku: wycięcie fragmentu tekstu to `SUBSTR` w Oracle i MySQL, `SUBSTRING` w SQL Server, a PostgreSQL rozumie akurat obie formy. Kod migrowany między silnikami trzeba przejrzeć pod kątem takich drobnych rozbieżności nazewniczych osobno od różnic w samej składni zapytania.

Traktowanie TRUNCATE jak bezpiecznego odpowiednika DELETE. `TRUNCATE` bywa nieodwracalny w ramach otwartej transakcji tam, gdzie `DELETE` dałoby się cofnąć — w MySQL z silnikiem InnoDB `TRUNCATE` wykonuje niejawny commit przed i po sobie, więc `ROLLBACK` po nim nie przywróci danych, nawet jeśli reszta transakcji się nie powiodła.

□□ Ćwiczenia

> Zadania sprawdzające.

Ćwiczenie 1 — rozpoznanie grupy

Dla każdego polecenia wskazujemy, czy jest to DDL, DML, DCL czy TCL: `ALTER TABLE zamowienia ADD COLUMN uwagi text;`, `REVOKE INSERT ON produkty FROM nazwa_usera;`, `DELETE FROM pozycje WHERE ilosc = 0;`, `SAVEPOINT przed_rabatem;`.

Odpowiedź

`ALTER TABLE` — DDL, zmienia strukturę tabeli. `REVOKE` — DCL, odbiera uprawnienie. `DELETE` — DML, usuwa dane, nie strukturę. `SAVEPOINT` — TCL, znacznik wewnątrz transakcji.

Ćwiczenie 2 — ta sama operacja, cztery silniki

Zapisujemy zapytanie sprawdzające bieżącą datę i pobierające pięć pierwszych wierszy z tabeli `klienci`, osobno w składni PostgreSQL i osobno w składni SQL Server.

Odpowiedź

PostgreSQL:

```
SELECT CURRENT_DATE, * FROM klienci LIMIT 5;
```

SQL Server:

```
SELECT TOP (5) GETDATE() AS CURRENT_DATE, * FROM klienci;
```

Obie wersje realizują to samo zadanie, ale różnią się miejscem, w którym stoi ograniczenie liczby wierszy, i nazwą funkcji zwracającej bieżącą datę — dokładnie tak, jak pokazuje tabela różnic w sekcji „Jak to działa”.

Pytania kontrolne

1. Dlaczego pierwszą komercyjną bazą z SQL nie był produkt IBM, mimo że to IBM sfinansował badania nad modelem relacyjnym i językiem SEQUEL?

Odpowiedź

Projekt System R był wewnętrznym projektem badawczym IBM, nie produktem. Firma Relational Software Inc. (późniejszy Oracle) przeczytała te same publikacje i wypuściła własną komercyjną implementację w 1979 roku, cztery lata przed komercyjnym DB2 od IBM.

2. Skąd wzięła się nazwa SQL, skoro pierwotny język nazywał się SEQUEL?

Odpowiedź

Nazwa SEQUEL kolidowała ze znakiem towarowym zastrzeżonym przez brytyjską firmę lotniczą Hawker Siddeley, więc IBM zmienił nazwę na SQL, zachowując zbliżoną wymowę.

3. Dlaczego zapytanie zgodne ze standardem SQL:2016 nie musi zadziałać identycznie na dwóch różnych silnikach?

Odpowiedź

Standard opisuje rdzeń języka, ale żaden silnik nie implementuje go w całości ani nie poprzestaje wyłącznie na nim — każdy dokłada własny dialekt z rozszerzeniami, których nie ma standard ani konkurencja.

4. Dlaczego TRUNCATE zalicza się do DDL, mimo że z punktu widzenia efektu — puste wiersze w tabeli — wygląda jak DELETE?

Odpowiedź

Bo mechanizm, jakim TRUNCATE osiąga ten efekt, jest inny niż w DELETE — nie usuwa wierszy pojedynczo z logowaniem każdej operacji, tylko zwalnia całe strony danych naraz, podobnie jak DROP zwalnia całą tabelę. To upodabnia go do operacji na strukturze, nie na pojedynczych rekordach, i tłumaczy, dlaczego w wielu silnikach nie da się go cofnąć tak samo łatwo jak DELETE.

5. Projekt Ingres z Berkeley i jego język QUEL przegrały z SQL na rynku, mimo że część badaczy uważała QUEL za lepiej zaprojektowany. Co z tego wynika dla oceny, dlaczego akurat SQL, a nie inny język, stał się standardem branżowym?

Odpowiedź

Wynika z tego, że o dominującym standardzie zdecydowała siła rynkowa produktów zbudowanych wokół danego języka — komercyjny Oracle i komercyjny DB2 od IBM — a nie techniczna elegancja samego języka zapytań. Standard formalny (ANSI, potem ISO) przyszedł już po tym, jak SQL faktycznie zdominował rynek, nie odwrotnie.

Powiązane: SELECT: kolejność zapisu a kolejność wykonania · NULL: trzecia wartość logiczna i pułapki, które z niej wynikają · ORDER BY, LIMIT, TOP, ROWNUM, FETCH: porcjowanie wyniku

Źródła: ISO/IEC 9075:2016 · dokumentacja PostgreSQL, MySQL, SQL Server, Oracle Database

Revision #46
Created 2026-09-24 22:41:43 UTC by Tomasz Konieczny
Updated 2026-10-26 14:10:45 UTC by Tomasz Konieczny