Model relacyjny: tabela, wiersz, kolumna, dziedzina

Poziom: podstawy **Dotyczy: wszystkie silniki** **Czas: ~15 min**

Tabela, wiersz i kolumna wydają się oczywiste, dopóki nie trzeba wytłumaczyć, dlaczego zapytanie bez `ORDER BY` może za każdym razem zwrócić dane w innej kolejności.

Skróty użyte na tej stronie (3)

Skrót	Co znaczy
IBM	International Business Machines: firma, w której laboratorium powstał model relacyjny i pierwszy działający silnik SQL
MSSQL	Microsoft SQL Server: silnik bazy danych Microsoftu, na nim w Zielonej stoi księgowość Optima
SQL	Structured Query Language: język zapytań do relacyjnych baz danych

Fiszka

TYLKO ODCZYT

Zapytanie bez gwarancji kolejności

```
SELECT id, nazwa FROM produkty;
```

Silnik nie ma obowiązku zwrócić wierszy w kolejności, w jakiej je wstawiliśmy — zwraca je w kolejności, która akurat jest dla niego najwygodniejsza.

Przykład:

```
 id |    nazwa
----+------------
  3 | Klawiatura
```

```
  1 | Monitor
  2 | Mysz
(3 rows)
```

TYLKO ODCZYT

Zapytanie z jawną kolejnością

```
SELECT id, nazwa FROM produkty ORDER BY id;
```

Jedyny sposób, żeby kolejność wyniku była gwarantowana i powtarzalna.

Przykład:

```
 id |    nazwa
----+------------
  1 | Monitor
  2 | Mysz
  3 | Klawiatura
(3 rows)
```

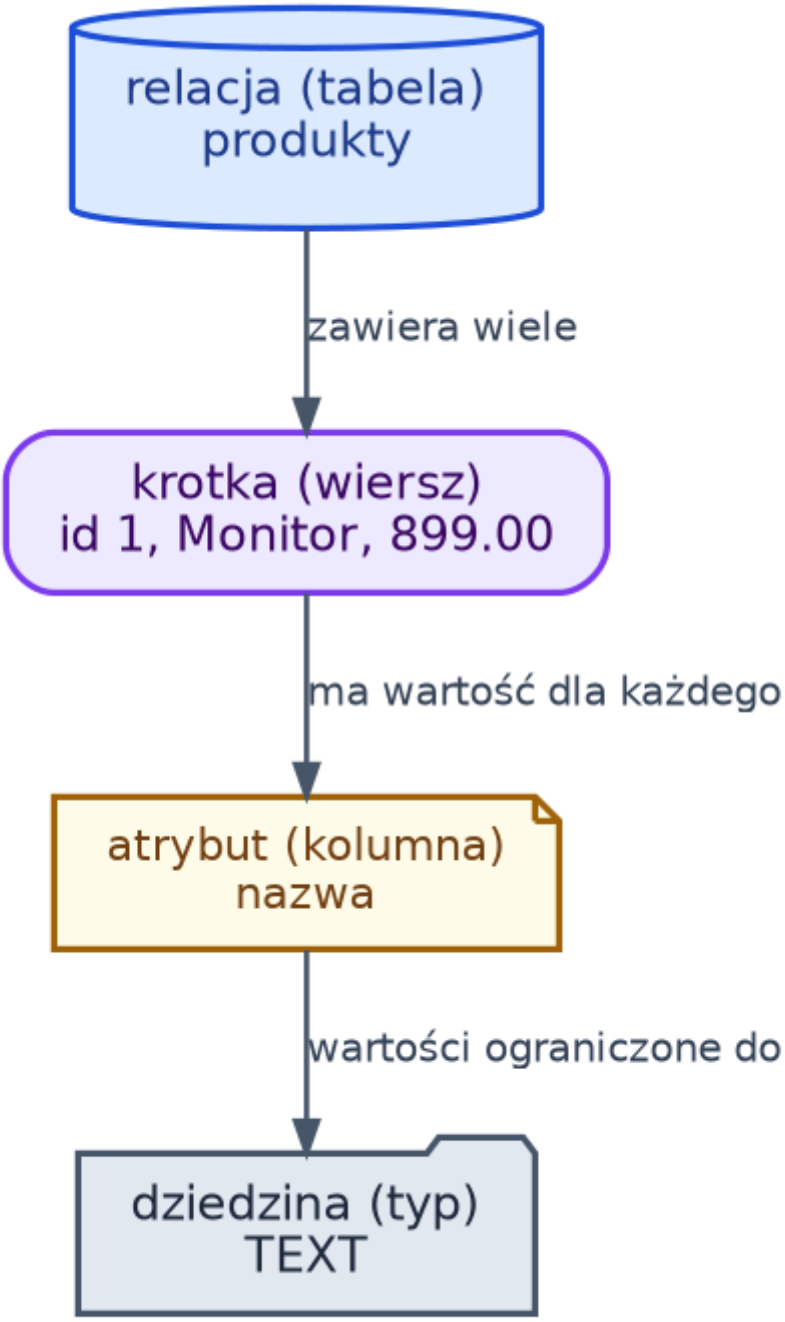
Po co to istnieje

Model relacyjny jest matematyczną podstawą, na której stoi każdy z silników SQL na tej półce — PostgreSQL, Oracle, MSSQL, MySQL, Sybase. Zanim zaczniemy pisać `SELECT` czy `JOIN`, warto wiedzieć, skąd biorą się słowa „tabela", „wiersz" i „kolumna" i dlaczego niektóre ich własności — jak brak gwarantowanej kolejności — nie są usterką konkretnego silnika, tylko wynikają wprost z definicji sprzed ponad pięćdziesięciu lat.

Model relacyjny opisał w 1970 roku Edgar F. Codd, matematyk pracujący w IBM, w artykule „A Relational Model of Data for Large Shared Data Banks". Ówczesne bazy danych (hierarchiczne i sieciowe) wymagały od programisty znajomości fizycznej struktury danych — żeby coś odczytać, trzeba było wiedzieć, jak dane są powiązane wskaźnikami na dysku. Codd zaproponował coś innego: dane opisane wyłącznie w kategoriach matematycznej teorii zbiorów i logiki predykatów, gdzie sposób zapytania o dane jest całkowicie oddzielony od sposobu ich fizycznego przechowywania. Pierwsze komercyjne wdrożenia tego pomysłu — w tym prototyp SQL — powstały w IBM i Oracle (wtedy jeszcze pod nazwą Relational Software) w drugiej połowie lat siedemdziesiątych.

Relacja, krotka, dziedzina — pojęcia matematyczne pod spodem

Pojęcie: relacja. W sensie matematycznym, w jakim używał tego słowa Codd, relacja to zbiór krotek o jednakowej strukturze — dokładnie to, co w praktyce nazywamy tabelą. To słowo wraca też w innym znaczeniu — jako „relacja między tabelami" (klucz obcy, powiązanie 1:N) — i to jest inne pojęcie, opisane osobno na stronie Relacje: 1:1, 1:N, N:M i tabela łącząca. Tu, mówiąc „relacja", mamy na myśli tabelę.



Relacja zawiera krotki, każda krotka ma wartość dla każdego atrybutu, a wartości atrybutu ogranicza jego dziedzina (kliknięcie otwiera pełny rozmiar)

> **Pojęcie: krotka (ang. tuple).** Pojedynczy element relacji — w praktyce jeden wiersz tabeli. Matematycznie krotka to uporządkowany zestaw wartości, po jednej z każdej dziedziny relacji.

Pojęcie: dziedzina (ang. domain). Zbiór dopuszczalnych wartości dla danej kolumny — w praktyce to, co nazywamy typem danych kolumny (`INTEGER`, `DATE`, `VARCHAR(50)`). Dziedzina ogranicza, co w ogóle może się znaleźć w danej pozycji krotki — kolumna zadeklarowana jako `DATE` nie przyjmie tekstu „poniedziałek".

Pojęcie: kardynalność relacji. Liczba krotek w relacji — czyli, w praktyce, liczba wierszy tabeli. Nie mylić z kardynalnością relacji między tabelami (1:1, 1:N, N:M) — to inne użycie tego samego słowa, opisane na stronie o relacjach.

Pojęcie: stopień relacji (ang. degree, arity). Liczba atrybutów (kolumn) relacji. Tabela o pięciu kolumnach ma stopień piąty, niezależnie od tego, ile ma wierszy.

Pojęcie: schemat relacji. Nazwa relacji razem z listą jej atrybutów i ich dziedzin — czyli, w praktyce, to, co zapisujemy w `CREATE TABLE`: definicja, jak tabela ma wyglądać, zanim jeszcze trafi do niej choć jeden wiersz. Odróżnia się od instancji relacji, czyli faktycznego zestawu krotek znajdujących się w tabeli w danym momencie. To rozróżnienie tłumaczy, dlaczego `ALTER TABLE` (zmiana schematu relacji) i `INSERT`/`UPDATE`/`DELETE` (zmiana instancji relacji) są operacjami tak różnej wagi — pierwsza zmienia definicję dla wszystkich przyszłych i istniejących wierszy naraz, druga dotyczy tylko konkretnych danych.

Pojęcie: atrybut. Nazwana kolumna relacji — w czystym modelu matematycznym atrybuty, podobnie jak krotki w relacji, nie mają ustalonej kolejności; odwołujemy się do nich po nazwie, nie po pozycji. SQL tej czystości nie zachowuje: kolumny tabeli mają zdefiniowaną kolejność zapisaną przy `CREATE TABLE`, `SELECT *` zwraca je zawsze w tej samej kolejności, a `INSERT` bez jawnej listy kolumn polega właśnie na tej kolejności.

Czysta relacja w sensie matematycznym ma dwie własności, których tabela SQL formalnie nie musi mieć. Po pierwsze, relacja jest zbiorem, więc nie zawiera duplikatów — dwie identyczne krotki to wciąż jeden element zbioru. Tabela SQL bez klucza głównego duplikaty jak najbardziej dopuszcza; `SELECT * FROM tabela` może zwrócić dwa identyczne wiersze. Po drugie, zbiór z definicji nie ma kolejności elementów — a to prowadzi wprost do sekcji niżej.

Analogia

Tabela relacyjna przypomina arkusz kalkulacyjny — wiersze, kolumny, komórka na przecięciu. To porównanie działa dobrze jako pierwsze wprowadzenie i tłumaczy, dlaczego w ogóle mówimy o „wierszach" i „kolumnach", a nie o czymś bardziej abstrakcyjnym.

Ale porównanie zaczyna zawodzić dość szybko. W arkuszu każda komórka może mieć inny format i inną zawartość niezależnie od reszty kolumny — jedna komórka liczbę, sąsiednia w tej samej kolumnie tekst albo formułę. W tabeli relacyjnej cała kolumna dzieli jedną dziedzinę: jeśli kolumna jest typu `INTEGER`, każda wartość w niej jest liczbą całkowitą, bez wyjątków. Arkusz nie ma też pojęcia klucza — nic nie stoi na przeszkodzie, żeby dwa wiersze były identyczne, i arkusz się tym nie przejmuje; tabela z kluczem głównym fizycznie nie pozwoli takiej sytuacji powstać.

Największa różnica dotyczy kolejności. W arkuszu kolejność wierszy jest ich integralną częścią — przesuwamy wiersz siódmy nad trzeci przeciągnięciem myszką, i to jest trwała zmiana zapisana w pliku. W tabeli relacyjnej kolejność wierszy nie jest w ogóle częścią danych. To jest najczęstsze zaskoczenie osób, które uczą się SQL po latach pracy w arkuszach.

Jak to działa

Relacja w sensie matematycznym jest zbiorem, a zbiór z definicji nie ma kolejności — {1, 2, 3} i {3, 1, 2} to ten sam zbiór. SQL dziedziczy tę własność: `SELECT` bez klauzuli `ORDER BY` nie daje żadnej gwarancji co do kolejności zwracanych wierszy, niezależnie od silnika.

W praktyce silnik musi fizycznie przechowywać wiersze w jakiejś kolejności na dysku, więc zapytanie bez `ORDER BY` zwykle zwraca dane w jakiejś powtarzalnej kolejności — najczęściej zbliżonej do kolejności zapisu na stronach danych. To złudzenie stabilności jest niebezpieczne, bo ta kolejność potrafi się zmienić bez żadnej zmiany w danych: po przebudowie indeksu, po `VACUUM FULL` w PostgreSQL, po tym jak optymalizator wybierze inny plan wykonania (skan indeksu zamiast skanu sekwencyjnego), albo po prostu po aktualizacji wersji silnika. Kod, który zakłada konkretną kolejność bez `ORDER BY`, jest kodem, który działa do pierwszej migracji albo pierwszego większego `UPDATE`.

To samo dotyczy stronicowania wyników. Klauzula `LIMIT 10 OFFSET 20` (albo jej odpowiednik w innym silniku) bez `ORDER BY` może zwrócić na stronie drugiej część wierszy, które już były na stronie pierwszej, oraz pominąć inne — bo silnik nie obiecał żadnej konkretnej kolejności, na której stronicowanie mogłoby się oprzeć.

Częste błędy

Poleganie na kolejności kolumn zamiast na ich nazwach. `INSERT INTO produkty VALUES (1, 'Monitor', 899.00)` działa tylko tak długo, jak nikt nie doda nowej kolumny ani nie zmieni kolejności istniejących. Dodanie kolumny `kategoria_id` w środku definicji tabeli przesuwa pozycje wszystkich kolumn po niej, a taki `INSERT` zacznie wstawiać cenę tam, gdzie wcześniej była nazwa. Bezpieczna forma zawsze wymienia kolumny wprost: `INSERT INTO produkty (id, nazwa, cena) VALUES (...)`.

Poleganie na kolejności wstawiania bez `ORDER BY`. Zapytanie testowane na małej tabeli deweloperskiej zwraca wiersze „w kolejności, w jakiej je dodaliśmy", więc programista zakłada, że

tak będzie zawsze. Na produkcji, po naroście danych i przebudowie indeksów, ta sama kwerenda zaczyna zwracać wiersze w innej kolejności — i raport, który miał pokazywać „ostatnie dziesięć zamówień", zaczyna pokazywać przypadkowe dziesięć.

Zakładanie, że tabela SQL nigdy nie ma duplikatów. Bez klucza głównego albo ograniczenia `UNIQUE` nic nie broni wstawienia dwóch identycznych wierszy — `INSERT` wykonany dwa razy z tymi samymi wartościami po prostu doda drugi, identyczny wiersz. To zachowanie zgodne z tym, jak działa tabela SQL, ale niezgodne z czystą definicją relacji jako zbioru, i bywa źródłem błędów w raportach, które liczą wiersze zamiast unikalnych wartości.

Porównywanie `NULL` operatorem `=`. `NULL` reprezentuje brak wartości w dziedzinie, a nie jedną z jej wartości, więc `kolumna = NULL` nie zwraca ani prawdy, ani fałszu — zwraca `NULL`, co w klauzuli `WHERE` liczy się jako „nie pokazuj tego wiersza". Właściwym operatorem jest `IS NULL`. Temat wraca osobno przy pełnym omówieniu trójwartościowej logiki SQL.

□□ Ćwiczenia

> Poniższe zadania wykonujemy na dowolnym silniku SQL, do którego mamy dostęp.

Ćwiczenie 1 — brak gwarancji kolejności

```
CREATE TABLE test_kolejnosci (id INTEGER, nazwa TEXT);
INSERT INTO test_kolejnosci VALUES (3, 'c'), (1, 'a'), (2, 'b');
SELECT * FROM test_kolejnosci;
```

Uruchamiamy to zapytanie kilka razy z rzędu, a potem po dodaniu i usunięciu kilkuset dodatkowych wierszy. Czy kolejność wyniku się zmienia?

Odpowiedź

Na małej, świeżo wstawionej tabeli kolejność zwykle pozostaje zbliżona do kolejności wstawiania, bo silnik czyta dane po prostu w kolejności zapisu na dysku. Po większych operacjach `DELETE`/`INSERT`, `VACUUM`, przebudowie indeksu albo po prostu wzroście liczby wierszy powyżej progu, przy którym optymalizator zaczyna wybierać skan indeksu zamiast sekwencyjnego, kolejność może się zmienić — bo nic w definicji SQL jej nie gwarantowało.

Ćwiczenie 2 — atrybuty a pozycja kolumn

```
CREATE TABLE produkty (id INTEGER, nazwa TEXT, cena NUMERIC);
INSERT INTO produkty VALUES (1, 'Monitor', 899.00);
ALTER TABLE produkty ADD COLUMN kategoria_id INTEGER;
INSERT INTO produkty VALUES (2, 'Mysz', 79.00);
```

Co się stanie z drugim `INSERT` i dlaczego to jest niebezpieczne w kodzie produkcyjnym?

Odpowiedź

`ALTER TABLE ... ADD COLUMN` w większości silników dodaje nową kolumnę na końcu listy, więc pozycje istniejących kolumn się nie zmieniają i ten konkretny `INSERT` zadziała poprawnie — ale tylko dlatego, że nowa kolumna wylądowała na końcu, a nie w środku. Gdyby ktoś wstawił kolumnę między `nazwa` i `cena` (co niektóre silniki pozwalają zrobić przez przebudowanie tabeli), każdy `INSERT` bez jawnej listy kolumn zacząłby wstawiać wartości w złych miejscach bez żadnego ostrzeżenia ze strony silnika.

Pytania kontrolne

1. Czym różni się dziedzina kolumny od typu danych kolumny w SQL?

Odpowiedź

W praktyce, na poziomie pojedynczej kolumny w konkretnym silniku, to niemal to samo — typ danych jest realizacją dziedziny. Różnica pojawia się, gdy dziedzina jest węższa niż typ: kolumna `wiek INTEGER CHECK (wiek >= 0)` ma typ `INTEGER`, ale jej rzeczywista dziedzina to nieujemne liczby całkowite — ograniczenie `CHECK` domyka dziedzinę ciaśniej niż sam typ danych.

2. Dlaczego dwa identyczne pod względem wartości wiersze w tabeli SQL nie łamią żadnej reguły silnika, mimo że łamią definicję relacji jako zbioru?

Odpowiedź

Bo SQL jako język nie wymusza czystości modelu matematycznego — tabela bez klucza głównego ani ograniczenia `UNIQUE` jest w SQL wielozbiorem (ang. multiset albo bag), nie zbiorem. Uzyskanie prawdziwie relacyjnej semantyki „brak duplikatów" wymaga świadomego dodania ograniczenia, które to wymusi.

Powiązane: Baza danych, silnik, instancja, schemat: cztery pojęcia, które się myli · Klucze: główny, obcy, unikalny, naturalny, sztuczny · ORDER BY, LIMIT/TOP/ROWNUM/FETCH — porcjowanie wyniku

Źródła: E. F. Codd, „A Relational Model of Data for Large Shared Data Banks" (1970) · C. J. Date, „An Introduction to Database Systems" · dokumentacja PostgreSQL, rozdz. „Query Language (SQL)"

Revision #47
Created 2026-09-24 22:41:34 UTC by Tomasz Konieczny
Updated 2026-10-26 14:10:33 UTC by Tomasz Konieczny