CREATE TABLE: typy danych i ich odpowiedniki między silnikami

> Każda kolumna dostaje typ danych, który silnik pilnuje przy każdym zapisie, a te same rodziny typów — liczby całkowite, tekst, data, wartość logiczna — noszą w czterech silnikach z tej książki cztery różne nazwy, czasem z pułapkami, które ujawniają się dopiero przy dużych wolumenach danych albo przy migracji między silnikami.

□□ Skróty użyte na tej stronie (6)

Skrót	Co znaczy
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
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
UTC	Coordinated Universal Time: uniwersalny czas bez strefy, punkt odniesienia dla wszystkich stref czasowych na świecie
UUID	Universally Unique Identifier: długi, praktycznie niepowtarzalny identyfikator

Fiszka

□□ ZMIENIA SYSTEM

Utworzenie tabeli z podstawowymi typami

```
CREATE TABLE produkty (
    id INTEGER PRIMARY KEY,
    nazwa VARCHAR(200) NOT NULL,
    kategoria VARCHAR(50) NOT NULL,
    cena NUMERIC(10, 2) NOT NULL,
    dostepny BOOLEAN NOT NULL DEFAULT true,
    utworzono TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);
```

Każda kolumna dostaje dokładnie jeden typ danych — silnik odrzuci próbę wstawienia wartości, która się w niego nie mieści, zamiast po cichu ją zaokrąglić czy obciąć. `NUMERIC(10, 2)` oznacza liczbę dziesiętną z maksymalnie dziesięcioma cyframi znaczącymi, z czego dwie po przecinku.

Przykład:

```
CREATE TABLE
```

id = pokaż użytkownika, jego numer UID oraz grupy, do których należy

Rodziny typów danych

> **Pojęcie: typ danych kolumny.** Deklaracja z góry narzucająca, jakiego rodzaju wartości wolno przechowywać w danej kolumnie i ile miejsca silnik ma na nie zarezerwować — to konkretna, techniczna realizacja pojęcia dziedziny opisanego na stronie Model relacyjny: tabela, wiersz, kolumna, dziedzina. Silnik wymusza typ przy każdym zapisie, bo bez tego wymuszenia porównania, sortowanie i operacje arytmetyczne na kolumnie nie miałyby jednoznacznego znaczenia — `'10' + '9'` jako tekst i jako liczba dają zupełnie inny wynik, a bez zadeklarowanego typu silnik nie wiedziałby, którego znaczenia użyć.

Typy danych dzielą się na kilka rodzin, z których każda odpowiada innemu rodzajowi informacji: liczby całkowite, liczby z ułamkiem (dalej rozbite na przybliżone i dokładne), tekst o długości stałej albo zmiennej, data i czas — z uwzględnieniem strefy czasowej albo bez niej — wartość logiczna, dane binarne, dokument JSON i identyfikator UUID. Każda rodzina ma swój własny zestaw nazw w czterech silnikach z tej książki, zestawiony w tabeli niżej.

Liczby całkowite

```
wiek SMALLINT,
liczba_sztuk INTEGER,
suma_transakcji BIGINT
```

Trzy rozmiary liczb całkowitych różnią się zakresem wartości i miejscem zajmowanym na dysku — wybór między nimi to kompromis między oszczędnością miejsca a zapasem na przyszły wzrost wartości, na przykład identyfikatora, który po latach działania systemu może przekroczyć zakres `INTEGER`.

Liczby zmiennoprzecinkowe kontra dokładne dziesiętne

```
cena NUMERIC(10, 2),
wspolczynnik REAL
```

> **Pojęcie: liczba zmiennoprzecinkowa.** Typ przechowujący liczby z ułamkiem w postaci przybliżonej, w formacie zbliżonym do notacji naukowej z ograniczoną precyzją binarną — szybki w obliczeniach, ale niezdolny do dokładnego zapisania wielu wartości dziesiętnych, które w systemie dziesiętnym wyglądają na proste, na przykład `0.1`. Typ dokładny dziesiętny (`NUMERIC` / `DECIMAL`) przechowuje cyfry wprost, bez przybliżenia, kosztem nieco wolniejszych obliczeń.

Kwoty pieniężne trzymamy zawsze w `NUMERIC` / `DECIMAL`, nigdy w `FLOAT` / `REAL` / `DOUBLE` — błąd przybliżenia liczby zmiennoprzecinkowej, rzędu jednej dziesięciotysięcznej part, po zsumowaniu tysięcy transakcji potrafi urosnąć do groszowej, a przy dużych wolumenach do znacznie większej rozbieżności w bilansie, której nie da się później jednoznacznie wytłumaczyć ani naprawić bez przeliczenia wszystkiego od nowa.

Przykład:

```
SELECT 0.1::real + 0.2::real = 0.3::real AS rownosc_float;
```

```
 rownosc_float
---------------
 f
(1 row)
```

Ten sam test na `NUMERIC` daje `true` — dokładny typ dziesiętny nie ma tego problemu, bo nie przybliża wartości binarnie.

Tekst o stałej i zmiennej długości

```
kod_kraju CHAR(2),
nazwa VARCHAR(200),
opis TEXT
```

`CHAR(n)` rezerwuje zawsze dokładnie `n` znaków i dopełnia krótszy tekst spacjami z prawej strony przy zapisie — `CHAR(2)` z wartością `'PL'` zajmuje tyle samo miejsca co z wartością `'P'`, uzupełnioną spacją. `VARCHAR(n)` przechowuje tylko tyle znaków, ile faktycznie wpisano, do limitu `n`, bez dopełniania. `TEXT` (w Oracle i starszym MSSQL zastępowany przez `VARCHAR(MAX)` / `CLOB`) nie ma górnego limitu długości zadeklarowanego z góry.

Uważaj: porównanie `CHAR(10)` z wartością `'Kowalski'` i `VARCHAR(10)` z tą samą wartością daje w większości silników ten sam wynik logiczny, bo silnik ucina końcowe spacje przy porównaniu tekstów `CHAR` — ale długość zwrócona przez funkcję długości tekstu (`LENGTH`, `LEN`, `CHAR_LENGTH`) różni się: `CHAR(10)` zwróci `10`, `VARCHAR(10)` zwróci `8`.

Data i czas ze strefą i bez

```
data_urodzenia DATE,
czas_zdarzenia TIMESTAMP,
czas_zdarzenia_tz TIMESTAMPTZ
```

`DATE` przechowuje samą datę bez godziny. `TIMESTAMP` przechowuje datę z godziną, ale bez informacji o strefie czasowej — dwie sesje w różnych strefach zapiszą tę samą wartość liczbową bez żadnego przeliczenia, co jest źródłem błędów przy systemach obsługujących klientów w wielu strefach naraz. `TIMESTAMPTZ` (w PostgreSQL) dodatkowo normalizuje wartość do UTC przy zapisie i przelicza z powrotem do strefy klienta przy odczycie.

Wartość logiczna

```
aktywny BOOLEAN
```

Typ o dokładnie dwóch wartościach właściwych, `true` i `false`, plus — jak każdy inny typ w SQL — możliwym `NULL` oznaczającym brak informacji, opisanym w pełni na stronie NULL: trzecia wartość logiczna i pułapki, które z niej wynikają.

Dane binarne

```
zawartosc_pliku BYTEA
```

Ciąg bajtów bez interpretacji tekstowej — obrazy, pliki, sumy kontrolne. Nie mylić z tekstem zapisanym w konkretnym kodowaniu; dane binarne nie mają kodowania znaków, bo nie są znakami.

JSON

```
atrybuty JSONB
```

Dokument w formacie JSON przechowywany natywnie przez silnik, z możliwością indeksowania i odpytywania poszczególnych pól bez konieczności parsowania całego dokumentu po stronie aplikacji. `JSONB` w PostgreSQL przechowuje dokument w formie już sparsowanej binarnie, szybszej do odpytywania kosztem nieco wolniejszego zapisu względem zwykłego `JSON`, który trzyma oryginalny tekst.

UUID

```
id UUID DEFAULT gen_random_uuid()
```

> **Pojęcie: UUID.** Uniwersalny, 128-bitowy identyfikator, praktycznie niepowtarzalny bez potrzeby centralnego licznika — przydatny, gdy identyfikatory muszą być generowane niezależnie w wielu miejscach naraz (różne serwery, offline w aplikacji mobilnej) bez ryzyka kolizji, kosztem większego rozmiaru niż zwykły `INTEGER`/`BIGINT` i gorszej lokalności fizycznej w indeksie B-drzewa, opisanej na stronie B-drzewo od środka.

Odpowiedniki typów między silnikami

Rodzina	PostgreSQL	Oracle	SQL Server	MySQL
Liczba całkowita mała	`SMALLINT`	`NUMBER(5)`	`SMALLINT`	`SMALLINT`
Liczba całkowita standardowa	`INTEGER`	`NUMBER(10)`	`INT`	`INT`
Liczba całkowita duża	`BIGINT`	`NUMBER(19)`	`BIGINT`	`BIGINT`

Rodzina	PostgreSQL	Oracle	SQL Server	MySQL
Liczba dziesiętna dokładna	`NUMERIC(p, s)`	`NUMBER(p, s)`	`DECIMAL(p, s)`	`DECIMAL(p, s)`
Liczba zmiennoprzecinkowa	`REAL` / `DOUBLE PRECISION`	`BINARY_FLOAT` / `BINARY_DOUBLE`	`REAL` / `FLOAT`	`FLOAT` / `DOUBLE`
Tekst stałej długości	`CHAR(n)`	`CHAR(n)`	`CHAR(n)`	`CHAR(n)`
Tekst zmiennej długości	`VARCHAR(n)`	`VARCHAR2(n)`	`VARCHAR(n)` / `NVARCHAR(n)`	`VARCHAR(n)`
Tekst bez limitu	`TEXT`	`CLOB`	`VARCHAR(MAX)`	`TEXT`
Data bez godziny	`DATE`	`DATE` (zawiera godzinę)	`DATE`	`DATE`
Data i godzina bez strefy	`TIMESTAMP`	`TIMESTAMP`	`DATETIME2`	`DATETIME`
Data i godzina ze strefą	`TIMESTAMPTZ`	`TIMESTAMP WITH TIME ZONE`	`DATETIMEOFFSET`	brak natywnego — przelicza się do UTC ręcznie
Wartość logiczna	`BOOLEAN`	brak natywnego — `NUMBER(1)` albo `CHAR(1)`	`BIT`	`BOOLEAN` (alias `TINYINT(1)`)
Dane binarne	`BYTEA`	`BLOB`	`VARBINARY(MAX)`	`BLOB`
JSON	`JSON` / `JSONB`	`JSON` (od 21c natywnie)	`NVARCHAR(MAX)` + walidacja `ISJSON`	`JSON`
UUID	`UUID`	`RAW(16)`	`UNIQUEIDENTIFIER`	`CHAR(36)` / `BINARY(16)`

Uważaj

- Oracle `DATE` w praktyce zawsze zawiera składnik godziny co do sekundy — to jedna z częstszych niespodzianek przy migracji z PostgreSQL, gdzie `DATE` jest czystą datą.
- Oracle nie ma natywnego typu logicznego w kolumnach tabel przed wersją 23c — konwencja `NUMBER(1)` z wartościami `0`/`1` albo `CHAR(1)` z `'T'`/`'N'` jest rozwiązaniem zastępczym, nie standardem.

Kodowanie i collation

> **Pojęcie: collation.** Reguła określająca, jak silnik porównuje i sortuje wartości tekstowe — czy rozróżnia wielkość liter, czy traktuje `ą` jako wariant `a` przy sortowaniu alfabetycznym, w jakiej kolejności stoją znaki spoza alfabetu łacińskiego. Kodowanie znaków (najczęściej `UTF8` we wszystkich czterech silnikach) decyduje o tym, ile bajtów zajmuje każdy znak i jakie znaki w ogóle da się zapisać; collation decyduje, jak te już zapisane znaki się ze sobą porównuje i sortuje. Dwie bazy z tym samym kodowaniem `UTF8`, ale różnym collation, mogą zwrócić różną kolejność tych samych polskich nazw w `ORDER BY nazwa` — jedna posortuje `Żelazny` przed `Zawadzki`, traktując `Ż` jak osobną literę alfabetu, druga zignoruje diakrytyki i posortuje odwrotnie.

Częste błędy

Kwota w FLOAT zamiast NUMERIC.

```
CREATE TABLE zamowienia_bledne (id INTEGER, wartosc FLOAT);
INSERT INTO zamowienia_bledne VALUES (1, 19.99), (2, 9.01);
SELECT SUM(wartosc) FROM zamowienia_bledne;
```

```
       sum
------------------
 29.000000000000004
```

Suma dwóch prostych kwot dziesiętnych nie wychodzi równa `29.00` przez błąd przybliżenia binarnego opisany wyżej — na `NUMERIC(10,2)` ten sam rachunek dałby dokładnie `29.00`.

CHAR zamiast VARCHAR na kolumnie o zmiennej, nieprzewidywalnej długości.

`CHAR(200)` na kolumnie z nazwami klientów, gdzie większość nazw ma kilkanaście znaków, marnuje miejsce na dopełnienie spacjami przy każdym krótszym wierszu i komplikuje porównania w aplikacjach, które nie oczekują końcowych spacji. `VARCHAR` jest właściwym wyborem wszędzie tam, gdzie długość tekstu naprawdę się różni między wierszami.

Przechowywanie czasu bez strefy w systemie obsługującym wiele stref.

Kolumna `TIMESTAMP` (bez strefy) zapisana z serwera działającego w strefie `Europe/Warsaw` i odczytana przez aplikację skonfigurowaną na `UTC` da błędną różnicę godzin bez żadnego komunikatu błędu — silnik nie ma jak wykryć tej niespójności, bo `TIMESTAMP` z definicji nie przechowuje informacji o strefie.

□□ Ćwiczenia

> Zadania sprawdzające.

Ćwiczenie 1 — wybór typu dla kwoty

Projektujemy tabelę `platnosci` z kolumną `kwota`, która ma przechowywać kwoty transakcji w złotówkach z groszami. Który typ wybieramy spośród `INTEGER`, `FLOAT` i `NUMERIC(12, 2)`, i dlaczego dwa pozostałe odpadają?

Odpowiedź

`NUMERIC(12, 2)` — `INTEGER` nie ma miejsca na grosze bez sztucznego mnożenia kwoty przez 100 i pamiętania o tym mnożeniu w każdym miejscu, gdzie kwota jest odczytywana, co jest źródłem błędów. `FLOAT` wprowadza błąd przybliżenia binarnego, który po zsumowaniu wielu transakcji przestaje się zgadzać z sumą kontrolną liczoną ręcznie — niedopuszczalne w danych finansowych.

Pytania kontrolne

1. Dlaczego `CHAR(10)` i `VARCHAR(10)` z tą samą wartością `'Kowalski'` mogą dać różny wynik funkcji liczącej długość tekstu?

Odpowiedź

`CHAR(n)` dopełnia wartość spacjami z prawej strony do pełnej zadeklarowanej długości przy zapisie, więc `LENGTH`/`LEN` zwróci `10`. `VARCHAR(n)` przechowuje tylko faktycznie wpisane znaki, więc ta sama wartość da długość `8`.

2. Czym różni się kodowanie znaków od collation?

Odpowiedź

Kodowanie znaków decyduje, jak tekst jest zapisany w bajtach i jakie znaki w ogóle da się w tym zapisie zmieścić. Collation decyduje, jak silnik porównuje i sortuje wartości tekstowe zapisane w tym kodowaniu — dwie bazy z identycznym kodowaniem mogą sortować te same

dane w różnej kolejności, jeśli mają różne collation.

Powiązane: Model relacyjny: tabela, wiersz, kolumna, dziedzina · Ograniczenia: NOT NULL, UNIQUE, CHECK, FOREIGN KEY, DEFAULT · NULL: trzecia wartość logiczna i pułapki, które z niej wynikają · ALTER TABLE: co blokuje tabelę, a co nie

Źródła: dokumentacja PostgreSQL, rozdział „Data Types” · dokumentacja Oracle, „Data Types” · dokumentacja Microsoft, „Data types (Transact-SQL)” · dokumentacja MySQL, „Data Types” · ISO/IEC 9075-2:2016 § 4.4-4.6

Revision #47
Created 2026-09-24 22:42:16 UTC by Tomasz Konieczny
Updated 2026-10-26 14:11:20 UTC by Tomasz Konieczny