Katalog systemowy: skąd baza wie, co w sobie ma

Poziom: podstawy **Dotyczy: wszystkie silniki** **Czas: ~20 min**

Baza danych wie, jakie ma tabele i kolumny, bo ta informacja też jest zapisana w tabelach — tyle że w tabelach należących do samej bazy, nie do nas.

Skróty użyte na tej stronie (4)

Skrót	Co znaczy
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)
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

Wylistowanie tabel przez information_schema (standard ANSI)

```
SELECT table_schema, table_name
FROM information_schema.tables
WHERE table_type = 'BASE TABLE'
ORDER BY table_schema, table_name;
```

Ten sam zapis działa bez zmian w PostgreSQL, MySQL i MSSQL, bo `information_schema` jest częścią standardu SQL, nie wynalazkiem jednego silnika.

Przykład (PostgreSQL, psql):

```
 table_schema |    table_name
--------------+------------------
 public       | klienci
 public       | pozycje_zamowienia
 public       | zamowienia
(3 rows)
```

ANSI = American National Standards Institute: tu chodzi o kody sterujące kolorem i pozycją kursora w terminalu · **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

TYLKO ODCZYT

Wylistowanie kolumn tabeli przez information_schema

```
SELECT column_name, data_type, is_nullable
FROM information_schema.columns
WHERE table_name = 'zamowienia'
ORDER BY ordinal_position;
```

Przykład:

```
 column_name   |     data_type     | is_nullable
---------------+-------------------+-------------
 id            | integer           | NO
 klient_id     | integer           | NO
 data_zlozenia | date              | NO
(3 rows)
```

TYLKO ODCZYT

Wylistowanie tabel przez katalog natywny PostgreSQL: pg_class

```
SELECT relname
FROM pg_catalog.pg_class
WHERE relkind = 'r'
  AND relnamespace = 'public'::regnamespace
ORDER BY relname;
```

`pg_class` to jedna z centralnych tabel katalogu PostgreSQL — trzyma po jednym wierszu na każdy obiekt „podobny do tabeli": tabelę, widok, indeks, sekwencję. Kolumna `relkind` odróżnia je od siebie (`r` to zwykła tabela).

Przykład:

```
       relname
---------------------
 klienci
 pozycje_zamowienia
 zamowienia
(3 rows)
```

ZMIENIA SYSTEM

Wylistowanie kolumn tabeli przez katalog natywny PostgreSQL: pg_attribute

```
SELECT attname, atttypid::regtype
FROM pg_catalog.pg_attribute
WHERE attrelid = 'zamowienia'::regclass
  AND attnum > 0
  AND NOT attisdropped
ORDER BY attnum;
```

Warunek `attnum > 0` odsiewa kolumny systemowe (`ctid`, `xmin`), a `NOT attisdropped` odsiewa kolumny formalnie usunięte, ale wciąż fizycznie zajmujące miejsce w definicji wiersza.

Przykład:

```
    attname     | atttypid
----------------+-----------
 id             | integer
 klient_id      | integer
 data_zlozenia  | date
(3 rows)
```

TYLKO ODCZYT

Wylistowanie tabel w Oracle: ALL_TABLES

```
SELECT owner, table_name
FROM all_tables
WHERE owner = 'SKLEP'
ORDER BY table_name;
```

Przykład (sqlplus):

```
OWNER      TABLE_NAME
---------- ------------------------------
SKLEP      KLIENCI
SKLEP      POZYCJE_ZAMOWIENIA
SKLEP      ZAMOWIENIA

3 rows selected.
```

TYLKO ODCZYT

Wylistowanie kolumn tabeli w Oracle: USER_TAB_COLUMNS

```
SELECT column_name, data_type, nullable
FROM user_tab_columns
WHERE table_name = 'ZAMOWIENIA'
ORDER BY column_id;
```

`USER_TAB_COLUMNS` pokazuje wyłącznie obiekty należące do schematu, którym właśnie jesteśmy zalogowani — bez potrzeby filtrowania po właścicielu, w przeciwieństwie do `ALL_TAB_COLUMNS`.

Przykład:

```
COLUMN_NAME        DATA_TYPE       N
------------------ --------------- -
ID                  NUMBER          N
KLIENT_ID           NUMBER          N
DATA_ZLOZENIA       DATE            N

3 rows selected.
```

TYLKO ODCZYT

Wylistowanie tabel w MSSQL: sys.tables

```
SELECT SCHEMA_NAME(schema_id) AS schemat, name
FROM sys.tables
ORDER BY schemat, name;
```

Przykład (sqlcmd):

```
schemat      name
------------ --------------------
dbo          Klienci
dbo          PozycjeZamowienia
dbo          Zamowienia

(3 rows affected)
```

MSSQL = Microsoft SQL Server: silnik bazy danych Microsoftu, na nim w Zielonej stoi księgowość Optima

TYLKO ODCZYT

Wylistowanie kolumn tabeli w MSSQL: sys.columns

```
SELECT name, TYPE_NAME(system_type_id) AS typ, is_nullable
FROM sys.columns
WHERE object_id = OBJECT_ID('dbo.Zamowienia')
ORDER BY column_id;
```

Przykład:

```
name            typ        is_nullable
--------------- ---------- -----------
Id              int        0
KlientId        int        0
DataZlozenia    date       0

(3 rows affected)
```

MSSQL = Microsoft SQL Server: silnik bazy danych Microsoftu, na nim w Zielonej stoi księgowość Optima

Po co to istnieje

Zapytanie `SELECT * FROM zamowienia` zwraca dane z tabeli `zamowienia`, ale ktoś — albo coś — musiało wcześniej wiedzieć, że taka tabela w ogóle istnieje, ile ma kolumn i jakiego są typu, zanim silnik zdołał ten `SELECT` w ogóle sparsować. Ta wiedza nie jest wbudowana w kod silnika na sztywno — jest zapisana w bazie, obok naszych własnych danych, w postaci zwykłych tabel.

> **Pojęcie: katalog systemowy (słownik danych).** Zbiór tabel i widoków, które opisują strukturę samej bazy — jakie ma tabele, kolumny, indeksy, użytkowników, uprawnienia. To dane o danych: metadane. Katalog systemowy jest tworzony i aktualizowany automatycznie przy każdej komendzie `CREATE`, `ALTER` czy `DROP` — nie edytujemy go bezpośrednio, tylko przez zwykłe polecenia DDL, ale czytać go możemy zwykłym `SELECT`, dokładnie tak jak każdą inną tabelę.

Każdy z czterech głównych silników SQL na tej półce ma dwie równoległe drogi dostępu do tej samej informacji: standard ANSI (`information_schema`) i katalog natywny, specyficzny dla silnika.

> **Pojęcie: information_schema.** Zestaw widoków zdefiniowany w standardzie SQL, jednakowo nazwany i jednakowo ustrukturyzowany w każdym silniku, który go implementuje — PostgreSQL, MySQL i MSSQL. Zaletą jest przenośność: to samo zapytanie o listę tabel działa bez zmian na trzech różnych silnikach. Wadą jest to, że standard opisuje wspólny mianownik, więc szczegóły specyficzne dla danego silnika — fizyczny rozmiar tabeli na dysku, typ metody dostępu indeksu, statystyki optymalizatora — w `information_schema` po prostu nie istnieją.
>
> **Pojęcie: katalog natywny.** Własny, wewnętrzny zestaw tabel systemowych danego silnika, nieopisany żadnym zewnętrznym standardem — `pg_catalog` w PostgreSQL, `ALL_`/`USER_`/`DBA_` w Oracle, schemat `sys` w MSSQL. Daje dostęp do wszystkiego, co silnik o sobie wie, łącznie ze szczegółami niedostępnymi przez `information_schema`, kosztem przenośności — zapytanie do `pg_catalog` nie zadziała na Oracle.

Oracle nie implementuje `information_schema` w ogóle — jedyną drogą jest tam katalog natywny, z rozróżnieniem na trzy rodziny widoków: `USER_*` (obiekty własne bieżącego schematu), `ALL_*` (obiekty własne plus te, do których mamy nadane uprawnienia) i `DBA_*` (wszystkie obiekty w całej bazie, wymaga uprawnień administracyjnych).

Jak to działa

Katalog systemowy to nie magiczna warstwa poza bazą — to zwykłe tabele, żyjące w tej samej bazie co nasze dane, tyle że należące do silnika, a nie do nas. Kiedy wykonujemy `CREATE TABLE zamowienia (...)`, silnik robi dwie rzeczy: zakłada fizyczne miejsce na dane nowej tabeli oraz wstawia wiersze opisujące tę tabelę do własnych tabel katalogowych — w PostgreSQL nowy wiersz ląduje w `pg_class`, a po jednym wierszu na kolumnę w `pg_attribute`. `DROP TABLE` usuwa te wiersze z powrotem. To dlatego zapytanie do katalogu jest zwykłym `SELECT`, a nie osobną komendą — bo katalog jest tabelą jak każda inna, tyle że jej zawartość aktualizuje sam silnik przy okazji DDL.

Widoki `information_schema` w praktyce są zbudowane właśnie nad katalogiem natywnym — w PostgreSQL `information_schema.tables` jest widokiem, który pod spodem odpytuje `pg_class` i `pg_namespace`, tylko w formacie zgodnym ze standardem. To dlatego zapytanie do `information_schema` bywa odrobinę wolniejsze niż to samo pytanie zadane wprost do katalogu natywnego — przechodzi przez dodatkową warstwę widoku.

Częste błędy

Zapytanie do `information_schema.columns` bez filtra na schemat. W bazie, która ma wiele schematów (typowe w PostgreSQL i MSSQL), zapytanie `WHERE table_name = 'zamowienia'` bez dodatkowego `AND table_schema = 'public'` zwróci po jednym wierszu z każdego schematu, w którym istnieje tabela o tej nazwie — myląco sugerując więcej kolumn niż faktycznie ma nasza konkretna tabela.

Filtrowanie po nazwie z niewłaściwą wielkością liter. PostgreSQL składuje niecytowane nazwy identyfikatorów małymi literami — `CREATE TABLE Zamowienia` tworzy w rzeczywistości tabelę `zamowienia`, więc `WHERE table_name = 'Zamowienia'` w `information_schema.tables` nie zwróci nic. Oracle robi odwrotnie: niecytowane identyfikatory zapisuje wielkimi literami, więc tam trzeba filtrować `WHERE table_name = 'ZAMOWIENIA'`. Ten sam nawyk przeniesiony między silnikami gwarantuje pusty wynik.

Odpytywanie `ALL_TAB_COLUMNS` zamiast `USER_TAB_COLUMNS` na dużej, wielo-schematowej bazie Oracle. `ALL_TAB_COLUMNS` przegląda wszystkie obiekty, do których mamy jakiekolwiek uprawnienia w całej bazie — na środowisku z dziesiątkami schematów to zapytanie bywa zauważalnie wolniejsze niż to samo pytanie zawężone przez `USER_TAB_COLUMNS` do własnego schematu.

□□ Ćwiczenia

>

> Poniższe zadania wykonujemy na dowolnym silniku SQL, do którego mamy dostęp z prawem odczytu katalogu.

Ćwiczenie 1 — porównanie dwóch dróg do tej samej informacji

Na PostgreSQL wykonujemy oba zapytania o listę kolumn tabeli `zamowienia` — jedno przez `information_schema.columns`, drugie przez `pg_catalog.pg_attribute` — i porównujemy liczbę zwróconych wierszy.

Odpowiedź

Bez dodatkowych warunków `pg_attribute` może zwrócić więcej wierszy niż `information_schema.columns`, jeśli tabela ma kolumny formalnie usunięte przez `ALTER TABLE ... DROP COLUMN` — PostgreSQL fizycznie nie usuwa takiej kolumny od razu, tylko oznacza ją jako `attisdropped = true` i ukrywa z widoków standardowych. `information_schema.columns` nigdy takich kolumn nie pokazuje; `pg_attribute` pokaże je, jeśli nie dodamy warunku `NOT attisdropped`.

Pytania kontrolne

1. Dlaczego zapytanie do `information_schema.tables` napisane raz działa bez zmian na PostgreSQL, MySQL i MSSQL, ale nie na Oracle?

Odpowiedź

Bo `information_schema` jest częścią standardu SQL i te trzy silniki go implementują z tą samą strukturą widoków. Oracle nigdy nie zaimplementował `information_schema` — jego jedyną drogą dostępu do metadanych jest katalog natywny (`ALL_TABLES`, `USER_TABLES`, `DBA_TABLES`), więc zapytanie napisane pod standard trzeba tam przepisać.

2. Czym różni się `USER_TABLES` od `ALL_TABLES` w Oracle?

Odpowiedź

`USER_TABLES` pokazuje wyłącznie tabele należące do schematu, którym jesteśmy aktualnie zalogowani. `ALL_TABLES` pokazuje te same tabele plus wszystkie inne, do których nasz

użytkownik ma nadane jakiekolwiek uprawnienie (np. przez `GRANT SELECT`), niezależnie od tego, w czyim są schemacie.

Powiązane: Baza danych, silnik, instancja, schemat: cztery pojęcia, które się myli · Klucze: główny, obcy, unikalny, naturalny, sztuczny · Konta, role i uprawnienia: GRANT, REVOKE, zasada najmniejszych uprawnień

Źródła: dokumentacja PostgreSQL, rozdz. „The Information Schema" i „System Catalogs" · Oracle Database Reference, rozdz. „Static Data Dictionary Views" · Microsoft Docs, „Catalog Views (Transact-SQL)"

Revision #47
Created 2026-09-24 22:41:39 UTC by Tomasz Konieczny
Updated 2026-10-26 14:10:37 UTC by Tomasz Konieczny