

## IDŹ DO

PRZYKŁADOWY ROZDZIAŁ



SPIS TREŚCI

## KATALOG KSIĄŻEK

KATALOG ONLINE

ZAMÓW DRUKOWANY KATALOG

## TWÓJ KOSZYK

DODAJ DO KOSZYKA

## CENNIK I INFORMACJE

ZAMÓW INFORMACJE  
O NOWOŚCIACH

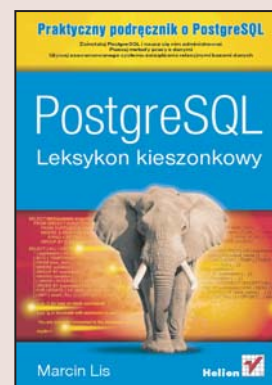
ZAMÓW CENNIK

## CZYTELNIA

FRAGMENTY KSIĄŻEK ONLINE

# PostgreSQL. Leksykon kieszonkowy

Autor: Marcin Lis  
ISBN: 83-246-0869-9  
Format: B6, stron: 160



### Rozpocznij pracę z bazami danych

PostgreSQL to jeden z najpopularniejszych systemów zarządzania relacyjnymi bazami danych (RDBMS) rozwijany na zasadzie wolnego oprogramowania. Zdaniem twórców jest to również najbardziej zaawansowany tego typu produkt na świecie. PostgreSQL umożliwia efektywne zarządzanie bazami danych w różnych systemach operacyjnych, w tym w licznych dystrybucjach Linuksa, systemach z rodziny Unix, Mac OS czy Windows. Jeśli chodzi o możliwości, wydajność i stabilność, PostgreSQL nie ustępuje komercyjnemu oprogramowaniu, a pod niektórymi względami nawet je przewyższa.

„PostgreSQL. Leksykon kieszonkowy” to zwięzły zbiór praktycznych informacji o jednym z najlepszych systemów RDBMS. Dzięki tej książce szybko zainstalujesz PostgreSQL oraz rozpoczniesz administrowanie tym systemem. Poznasz używane w nim typy danych, popularne instrukcje, funkcje i operatory. Nauczysz się obsługiwać tabele i tworzyć indeksy. Dowiesz się, jak stosować agregacje, złączenia i unie. Przeczytasz o technikach tworzenia widoków oraz używania transakcji, a także o złożonych instrukcjach PostgreSQL. Poznasz też metody obsługi znaków narodowych.

- Instalowanie PostgreSQL
- Administrowanie PostgreSQL
- Typy danych
- Instrukcje, funkcje i operatory
- Tworzenie i używanie indeksów
- Stosowanie złączeń, unii i widoków
- Agregowanie i grupowanie danych
- Stosowanie transakcji
- Korzystanie ze znaków narodowych

**Jeśli szukasz nowoczesnego i darmowego systemu RDBMS,  
PostgreSQL to doskonały wybór**

Wydawnictwo Helion  
ul. Kościuszki 1c  
44-100 Gliwice  
tel. 032 230 98 63  
e-mail: [helion@helion.pl](mailto:helion@helion.pl)



---

# Spis treści

<b>Wstęp .....</b>	<b>7</b>
<b>1. Instalacja .....</b>	<b>8</b>
W systemie Linux	8
W systemie Windows	9
<b>2. Administracja.....</b>	<b>11</b>
Uruchamianie i zatrzymywanie serwera	11
Obsługa kont użytkowników	12
Zakładanie i usuwanie baz danych	19
Uruchamianie klienta PostgreSQL	20
<b>3. Typy danych .....</b>	<b>22</b>
Typy znakowe	22
Typy numeryczne	23
Typy monetarne	26
Typy binarne	26
Typy daty i czasu	26
Typy logiczne	28
Typy bitowe	28
Typy specjalne	29

<b>4. Obsługa tabel .....</b>	<b>32</b>
Tworzenie tabel	32
Modyfikacja tabel	41
Usuwanie tabel	46
<b>5. Podstawowe instrukcje SQL .....</b>	<b>47</b>
Wstawianie danych	47
Pobieranie danych	49
Modyfikacja danych	54
Usuwanie danych	55
<b>6. Indeksy .....</b>	<b>57</b>
Typy indeksów	57
Tworzenie indeksów	58
Indeksy częściowe	59
Usuwanie indeksów	59
<b>7. Funkcje i operatory .....</b>	<b>60</b>
Funkcje	60
Operatory	85
<b>8. Agregacja i grupowanie danych .....</b>	<b>98</b>
Przykłady użycia funkcji agregujących	98
Grupowanie danych	100
Klauzula HAVING	102
<b>9. Złączenia, unie, widoki .....</b>	<b>103</b>
Łączenie wyników zapytań	103
Złączenia tabel	105
Widoki	111

<b>10. Złożone instrukcje SQL .....</b>	<b>114</b>
Podzapytania	114
Podzapytania w klauzuli FROM	115
Podzapytania w klauzuli WHERE	116
Podzapytania skorelowane	117
Złożona instrukcja INSERT	119
Złożona instrukcja DELETE	120
<b>11. Transakcje .....</b>	<b>121</b>
Ogólnie o transakcjach	121
Rozpoczynanie transakcji	122
Zatwierdzanie transakcji	122
Wycofywanie transakcji	123
Transakcje domyślne	123
Poziomy izolacji	123
<b>12. Obsługa znaków narodowych .....</b>	<b>126</b>
Standardy kodowania	126
Konwersje automatyczne	129
<b>13. Kody błędów .....</b>	<b>132</b>
<b>14. Słowa kluczowe .....</b>	<b>140</b>
<b>Skorowidz .....</b>	<b>153</b>

# Rozdział 5. Podstawowe instrukcje SQL

## Wstawianie danych

### Instrukcja INSERT

Do umieszczania danych w tabelach służy instrukcja `INSERT INTO`. Jej podstawowa forma ma ogólną postać:

```
INSERT [INTO] nazwa_tabeli [(kolumna1, kolumna2, ...,
kolumnaN)]
VALUES (wartość1, wartość2, ..., wartośćN)
```

Powoduje ona wprowadzenie do tabeli nowego wiersza, w którym w polu *kolumna1* została zapisana wartość *wartość1*, w polu *kolumna2* — wartość *wartość2* itd. Elementy instrukcji ujęte w nawias klamrowy są opcjonalne.

Przy założeniu, że w bazie została umieszczona tabela *osoby* o kolumnach:

- *id* — przechowuje identyfikator,
- *imie* — przechowuje imię,
- *nazwisko* — przechowuje nazwisko,

utworzona za pomocą instrukcji:

```
CREATE TABLE osoby
(
  id INTEGER PRIMARY KEY NOT NULL,
  imie VARCHAR(20),
  nazwisko VARCHAR(30),
)
```

wstawienie wiersza przechowującego dane Jana Kowalskiego, któremu został nadany identyfikator 1, zostanie wykonane przez instrukcję:

```
INSERT INTO osoby (id, imie, nazwisko) VALUES (1, 'Jan', 'Kowalski');
```

Taka instrukcja może być również rozbita na kilka wierszy, co w przypadku dużej ilości danych może zwiększyć jej czytelność, np.:

```
INSERT INTO
  osoby (id, imie, nazwisko)
VALUES
  (1, 'Jan', 'Kowalski');
```

W sytuacji gdy wstawiane są dane do wszystkich kolumn, a ich kolejność jest zgodna z kolejnością kolumn, nazwy kolumn można pominąć, np.:

```
INSERT INTO osoby VALUES (1, 'Jan', 'Kowalski');
```

Nie ma również konieczności wprowadzania danych zgodnie z kolejnością kolumn, kolejność ta może być dowolna, np.:

```
INSERT INTO osoby (nazwisko, id, imie) VALUES ('Kowalski', 1, 'Jan');
```

Możliwe jest także pominięcie niektórych kolumn, pod warunkiem że nie mają atrybutu NOT NULL, np.:

```
INSERT INTO osoby (id, nazwisko) VALUES (1, 'Kowalski');
```

Wartości kolumn o typie danych SERIAL mogą być generowane automatycznie. Zakładając, że pole id tabeli osoby jest tego typu, to aby wprowadzony wiersz miał automatycznie wygenerowaną wartość dla tego pola, w instrukcji wprowadzającej wiersz trzeba pominąć kolumnę id, np.:

```
INSERT INTO osoby (imie, nazwisko) VALUES ('Marceli', 'Przybysz');
```

lub też zastosować w niej wartość DEFAULT:

```
INSERT INTO osoby (id, imie, nazwisko) VALUES (DEFAULT, 'Marceli', 'Przybysz');
```

# Pobieranie danych

## Instrukcja SELECT

Dane zapisane w tabelach bazy danych można pobierać za pomocą instrukcji SELECT. Schemat jej podstawowej postaci wygląda następująco:

```
SELECT kolumna1, kolumna2, ..., kolumnaN  
FROM tabela  
[WHERE warunek]  
[ORDER BY kolumna1, kolumna2, ..., kolumnaN [ASC | DEC]]
```

Taka konstrukcja oznacza: pobierz wartości wymienionych kolumn z tabeli *tabela*, spełniających warunek *warunek*, a wyniki posortuj względem kolumn wymienionych w klauzuli ORDER BY, rosnąco (ASC) lub malejąco (DESC). Elementy ujęte w nawiasach kwadratowych są opcjonalne.

Przy założeniu, że w bazie istnieje tabela o nazwie *pracownicy*, o następujących kolumnach:

- *id* — typu INTEGER, będąca kluczem głównym i zawierająca identyfikator każdego wiersza,
- *imie* — typu VARCHAR(20), z atrybutem NOT NULL, zawierająca imię pracownika,
- *nazwisko* — typu VARCHAR(30), z atrybutem NOT NULL, zawierająca nazwisko pracownika,
- *płaca* — typu DECIMAL(7, 2), z atrybutem NOT NULL, zawierająca miesięczne wynagrodzenie pracownika,
- *stanowisko* — typu VARCHAR(20), z atrybutem NOT NULL, zawierająca stanowisko pracownika,
- *pesel* — typu CHAR(11), zawierająca PESEL pracownika,

do pobrania całej jej zawartości posłuży instrukcja:

```
SELECT * FROM pracownicy;
```

Symbol `*` oznacza tu, że w wyniku zapytania mają być uwzględnione wszystkie kolumny.

Jeżeli mają zostać wyświetlone tylko niektóre kolumny, nazwy tych kolumn należy umieścić za słowem `SELECT`, oddzielając poszczególne nazwy znakami przecinka, np.:

```
SELECT imie, nazwisko, stanowisko FROM pracownicy;
```

Istnieje również możliwość zmiany nazw kolumn w wynikach zapytania. Wystarczy, jeśli występujące w zapytaniu `SELECT` nazwy zostaną zastąpione sekwencjami o schematycznej postaci:

```
nazwa_kolumny AS alias
```

gdzie `nazwa_kolumny` to nazwa oryginalnej kolumny, a `alias` to nazwa, jaka ma się pojawić w wynikach zapytania, np.:

```
SELECT imie, nazwisko, placa AS wynagrodzenie FROM  
pracownicy;
```

## Sortowanie danych

Wyniki zapytania typu `SELECT` mogą być sortowane. Umożliwia to klauzula `ORDER BY`. Sortowanie może odbywać się w porządku rosnącym bądź malejącym, względem jednej bądź kilku kolumn. Porządkiem domyślnym jest porządek rosnący. Gdyby zatem istniała potrzeba wyświetlenia wszystkich wierszy tabeli `pracownicy` posortowanych względem nazwiska w porządku alfabetycznym rosnącym, należy zastosować konstrukcję:

```
SELECT * FROM pracownicy ORDER BY nazwisko ASC;
```

lub prościej:

```
SELECT * FROM pracownicy ORDER BY nazwisko;
```

Gdyby zaś sortowanie miało się odbywać w porządku malejącym — instrukcję:

```
SELECT * FROM pracownicy ORDER BY nazwisko DESC;
```

Sortowanie może się odbywać względem większej liczby kolumn. Przykładowo, tabela `pracownicy` może zostać posortowana najpierw względem nazwiska, a następnie względem płacy. Przy czym kierunek sortowania jest niezależny dla każdej kolumny, czyli można jednocześnie sortować względem nazwiska w porządku rosnącym i płacy w porządku malejącym, np.:

```
SELECT * FROM pracownicy ORDER BY nazwisko ASC, placa DESC;
```

## Kryteria pobierania danych

Otrzymanie określonego zestawu wierszy zapewnia klauzula `WHERE` instrukcji `SELECT`. Za klauzulą `WHERE` należy umieścić warunek, jaki muszą spełniać wiersze, aby znalazły się w wynikach zapytania. Warunek w klauzuli `WHERE` może zawierać różnego rodzaju operatory, które są przedstawione w rozdziale 7., „Funkcje i operatory”. Najczęściej stosuje się wszelkiego rodzaju operatory porównywania i logiczne. Poniżej zaprezentowanych zostało kilka przykładów zastosowania klauzuli `WHERE`, pobierających różne dane z tabeli `pracownicy` o strukturze takiej, jak przedstawiona w punkcie „Instrukcja `SELECT`”.

Jeśli pobrane mają zostać wiersze tabeli, które w kolumnie `nazwisko` zawierają wartość `Kowalski`, należy zastosować warunek `nazwisko='Kowalski'`, więc pełne zapytanie przyjmie postać:

```
SELECT * FROM pracownicy WHERE Nazwisko='Kowalski';
```

Jeśli pobrane mają zostać dane pracowników o płacy niższej niż 1600 zł, zapytanie będzie miało postać:

```
SELECT * FROM pracownicy WHERE placa < 1600;
```

Aby uzyskać w wyniku zapytania wiersze o identyfikatorach z przedziału 3 – 6, trzeba użyć dwóch warunków: `id >= 3` i `id <= 6` połączonych operatorem AND:

```
SELECT * FROM pracownicy WHERE id >= 3 AND id <= 6;
```

lub operatora BETWEEN:

```
SELECT * FROM pracownicy WHERE id BETWEEN 3 AND 6;
```

Jeśli konieczne jest uzyskanie wierszy o identyfikatorach ze zbioru 3, 5 i 7, można zastosować trzy instrukcje warunkowe: `id = 3`, `id = 5` i `id = 7` połączone za pomocą operatora OR (czyli sumy logicznej):

```
SELECT * FROM pracownicy WHERE id=3 OR id=5 OR id=7;
```

lub operator IN:

```
SELECT * FROM pracownicy WHERE id IN(3, 5, 7);
```

## Usuwanie powtórzonych wierszy

Instrukcja SELECT może być również uzupełniona klauzulą DISTINCT, która gwarantuje niepowtarzalność wierszy wynikowych, innymi słowy, eliminuje duplikaty z wyników zapytania. Klauzulę DISTINCT należy umieścić za słowem SELECT, ogólnie:

```
SELECT DISTINCT kolumna1, kolumna2, ..., kolumnaN  
FROM tabela  
[WHERE warunek]  
[ORDER BY kolumna1, kolumna2, ..., kolumnaN [ASC | DEC]]
```

np.:

```
SELECT DISTINCT nazwisko FROM pracownicy ORDER BY nazwisko;
```

## Etykiety kolumn

W wynikach zapytania można zmieniać nazwy kolumn przez nadanie im etykiet. Odbywa się to za pomocą słowa kluczowego AS. Ogólnie nazwę takiej kolumny należy zastąpić sekwencją:

```
nazwa_kolumny AS etykieta
```

np.:

```
SELECT id AS identyfikator, imie, nazwisko FROM pracownicy;
```

## Ograniczanie wyników zapytań

Do uzyskiwania ograniczonego podzbioru wierszy wynikowych zapytania służą klauzule `LIMIT` i `OFFSET`. Zapytanie je zawierające ma ogólną postać:

```
SELECT kolumna1, kolumna2, ..., kolumnaN  
FROM tabela  
[WHERE warunek]  
[ORDER BY kolumna1, kolumna2, ..., kolumnaN [ASC | DEC]]  
LIMIT [ile1 | ALL][OFFSET ile2]
```

gdzie *ile1* oznacza liczbę wierszy, które mają zostać uwzględnione w wynikach, natomiast *ile2* — liczbę wierszy, które mają być pominięte, zanim zaczną być prezentowane wyniki zapytania. Wystąpienie zamiast *ile1* słowa `ALL` będzie oznaczało, że mają być uwzględniane wszystkie wiersze (zatem zapytanie zachowa się tak, jakby klauzula `LIMIT` została pominięta), jeśli natomiast *ile2* będzie miało wartość 0, będzie to oznaczało, że żadne wiersze nie mają być pomijane (więc zapytanie zachowa się tak, jakby klauzula `OFFSET` została pominięta). Podczas stosowania klauzul `LIMIT` i `OFFSET` zwykle powinno się również stosować klauzulę sortującą `ORDER BY`.

Przykładowo, aby uzyskać 5 pierwszych wierszy z tabeli `pracownicy` sortowanej względem kolumny `id`, należy zastosować zapytanie:

```
SELECT * FROM pracownicy  
ORDER BY id  
LIMIT 5;
```

Natomiast w celu uzyskania 3 wierszy począwszy od 8. (czyli pomijając pierwsze 7) — instrukcję:

```
SELECT * FROM pracownicy
ORDER BY id
LIMIT 3 OFFSET 7;
```

## Modyfikacja danych

### Instrukcja UPDATE

Do modyfikacji danych zawartych w tabelach służy instrukcja UPDATE. Ma ona ogólną postać:

```
UPDATE nazwa_tabeli
SET kolumna1=wartość1, kolumna2=wartość2, ...,
kolumnaN=wartośćN
[WHERE warunek]
```

Oznacza ona: zmień w tabeli *nazwa\_tabeli*, w kolumnach spełniających warunek *warunek*, wartość kolumny *kolumna1* na *wartość1*, kolumny *kolumna2* na *wartość2* itd. Warunek występujący po klauzuli WHERE jest tutaj opcjonalny, a jego pominięcie oznacza, że zmiany będą dokonane we wszystkich wierszach.

Jeśli zatem w bazie istnieje tabela *osoby* zawierająca następujące kolumny:

- *id*,
- *imie*,
- *nazwisko*,
- *rok\_urodzenia*,

to zamianę wszystkich wartości w kolumnie *rok\_urodzenia* na wartość 1982 umożliwi instrukcja:

```
UPDATE osoby SET rok_urodzenia=1982;
```

Aby zmiana dotyczyła konkretnych wierszy, stosuje się warunek sekcji WHERE, np. zmianę roku urodzenia dla pracownika o identyfikatorze 8 można uzyskać za pomocą polecenia:

```
UPDATE osoby SET rok_urodzenia=1982 WHERE id=8;
```

Można oczywiście jednocześnie zmodyfikować kilka pól w danym wierszu, np.:

```
UPDATE osoby SET rok_urodzenia=1982, imie='Jan',  
nazwisko='Kowalski' WHERE id=1;
```

## Usuwanie danych

### Instrukcja DELETE

Do usuwania danych służy instrukcja DELETE o schematycznej postaci:

```
DELETE FROM tabela  
[WHERE warunek]
```

Oznacza ona: usuń z tabeli *tabela* wszystkie wiersze spełniające warunek *warunek*. Jeśli pominie się warunek, zostaną usunięte wszystkie dane (podobnie jak w przypadku instrukcji UPDATE, gdzie pominięcie warunku powodowało modyfikację wszystkich wierszy tabeli).

Aby usunąć wszystkie dane z tabeli *pracownicy*, należy wykonać instrukcję:

```
DELETE FROM pracownicy;
```

Po jej wykonaniu tabela *pracownicy* nie będzie zawierała żadnych danych. Taką konstrukcję trzeba zatem stosować z rozwagą, ponieważ serwer nie wygeneruje żadnego ostrzeżenia ani dodatkowego pytania.

Selektywne usuwanie danych jest możliwe dzięki użyciu klauzuli WHERE z odpowiednim wyrażeniem warunkowym, które konstruuje się na takich samych zasadach, jak w przypadku instrukcji SELECT czy UPDATE. Przykładowo, usunięcie z tabeli pracownicy wiersza, który w kolumnie id ma wartość 5, nastąpi po wykonaniu instrukcji:

```
DELETE FROM pracownicy WHERE id=5;
```