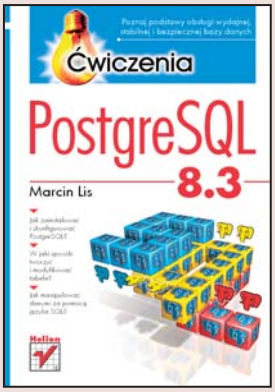
PostgreSQL 8.3. Ćwiczenia

Autor: Marcin Lis
ISBN: 978-83-246-0859-1
Format: B5, stron: 182



Poznaj podstawy obsługi wydajnej, stabilnej i bezpiecznej bazy danych

- Jak zainstalować i skonfigurować PostgreSQL?
- W jaki sposób tworzyć i modyfikować tabele?
- Jak manipulować danymi za pomocą języka SQL?

Oprogramowanie dostępne na licencji open source staje się coraz popularniejsze. Nikogo już chyba nie dziwi fakt, że nawet wśród największych i najbardziej rozbudowanych systemów znajdziemy produkty bezpłatne. Jednym z nich jest system zarządzania bazami danych PostgreSQL – powszechnie uważany za najbardziej uniwersalny i stabilny spośród baz danych rozprowadzanych na zasadach wolnego dostępu. PostgreSQL ma dwie wersje – dla systemów Windows i Linux – a jego możliwości wykorzystywane są zarówno przez twórców portali sieciowych, jak i potężnych systemów korporacyjnych przetwarzających ogromne ilości danych.

Książka „PostgreSQL 8.3. Ćwiczenia" to krótkie i skuteczne wprowadzenie w zasady używania tego systemu baz danych. Czytając ją i wykonując zawarte w niej ćwiczenia, dowiesz się, jak zainstalować PostgreSQL w Windows i Linuksie, oraz szybko poznasz podstawy administrowania serwerem bazy danych i kontami jej użytkowników. Nauczysz się tworzyć tabele, dobierać odpowiednie typy danych i budować indeksy. Opanujesz język SQL służący do manipulowania danymi w tabelach. Przeczytasz także o transakcjach i więzach integralności.

- Instalacja serwera PostgreSQL w systemach Windows i Linux
- Uruchamianie i zatrzymywanie serwera
- Obsługa kont użytkowników
- Zarządzanie bazami danych
- Podstawowe koncepcje relacyjnych baz danych
- Typy danych w PostgreSQL
- Tworzenie, modyfikowanie i usuwanie tabel
- Pobieranie danych i przetwarzanie wyników zapytania
- Złożone zapytania SQL
- Obsługa transakcji

Przekonaj się, dlaczego PostgreSQL zyskał tak ogromne uznanie

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

Spis treści

Podstawowe instrukcje SQL

Wprowadzanie danych

Instrukcja INSERT INTO

Tabele, których najróżniejsze sposoby tworzenia i modyfikacji poznaliśmy w rozdziale 4., trzeba w jakiś sposób wypełnić danymi. Służy do tego instrukcja `INSERT INTO`, którą poznamy na kolejnych stronach tego rozdziału. 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 kwadratowy są opcjonalne.

Przygotujmy więc przykładową tabelę, która posłuży do wykonywania kolejnych ćwiczeń. Niech będzie to tabela `osoby` utworzona za pomocą instrukcji:

```
CREATE TABLE osoby
(
  id INTEGER PRIMARY KEY NOT NULL,
  imie VARCHAR(20) NOT NULL,
  nazwisko VARCHAR(30) NOT NULL,
  pesel CHAR(11)
);
```

ĆWICZENIE

5.1 Wprowadzenie wiersza do tabeli

Wprowadź do tabeli `osoby` przykładowy wiersz danych.

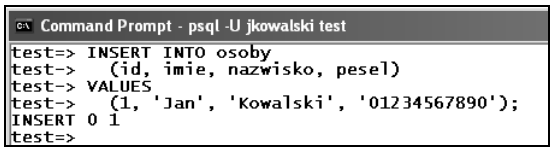
Zakładając, że nowy wiersz ma zawierać dane Jana Kowalskiego posiadającego PESEL 01234567890, któremu został nadany identyfikator `1`, należy użyć instrukcji `INSERT INTO` w postaci:

```
INSERT INTO osoby (id, imie, nazwisko, pesel) VALUES (1, 'Jan',
↪'Kowalski', '01234567890');
```

W celu zwiększenia czytelności można ją rozbić na kilka wierszy (rysunek 5.1), np.:

```
INSERT INTO osoby
  (id, imie, nazwisko, pesel)
VALUES
  (1, 'Jan', 'Kowalski', '01234567890');
```

Efekt wykonania zapytania jest widoczny na rysunku 5.1.



Rysunek 5.1. *Wykonanie zapytania wprowadzającego dane do tabeli osoby*

Zwróćmy przy tym uwagę, że wszystkie wprowadzone ciągi znaków zostały ujęte w apostrofy. Jest to niezbędne, aby zapytanie zostało wykonane prawidłowo. Nie ma natomiast potrzeby ujmowania w znaki apostrofu wartości liczbowych.

Jeśli jednak wprowadzamy wartości wszystkich pól, to nazwy kolumn w instrukcji `INSERT` są tak naprawdę opcjonalne i można je pominąć. Kolejność danych powinna być wtedy taka jak kolejność kolumn w definicji tabeli. Sprawdźmy to w praktyce.

ĆWICZENIE

5.2 Wprowadzenie danych z pominięciem nazw kolumn

Wprowadź do tabeli `osoby` przykładowy wiersz bez używania nazw kolumn.

Wykonanie ćwiczenia zapewni nam instrukcja:

```
INSERT INTO osoby VALUES (2, 'Adam', 'Nowak', '12345678901');
```

Nie ma jednak konieczności każdorazowego wprowadzania danych do wszystkich kolumn. Część z nich może zostać pominięta, pod warunkiem oczywiście że nie mają one przypisanego atrybutu `NOT NULL`.

ĆWICZENIE

5.3 Pominięcie danych dla wybranych kolumn

Wprowadź do tabeli `osoby` wiersz zawierający jedynie dane o identyfikatorze, imieniu oraz nazwisku.

Operację wprowadzenia do tabeli `osoby` wiersza bez danych dla kolumny `pesel` można przeprowadzić na dwa sposoby. Pierwszy z nich to instrukcja:

```
INSERT INTO osoby VALUES (3, 'Janusz', 'Nowak', NULL);
```

W takim przypadku w kolumnie `pesel` jawnie wstawiamy wartość pustą `NULL`. Drugą możliwość stanowi instrukcja:

```
INSERT INTO osoby (id, imie, nazwisko) VALUES (3, 'Janusz', 'Nowak');
```

W tym przypadku niezbędne jest użycie nazw kolumn.

Taka operacja jak w ćwiczeniu 5.3 nie byłaby jednak możliwa w przypadku kolumny `imie` bądź `nazwisko`, gdyż mają one atrybut `NOT NULL`, czyli nie mogą być puste. Sprawdźmy, czy tak jest faktycznie.

ĆWICZENIE

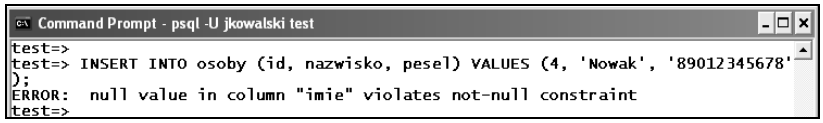
5.4 Pominięcie danych dla kolumny z atrybutem NOT NULL

Wykonaj próbę wprowadzenia do tabeli `osoby` wiersza niezawierającego danych o imieniu.

Pominięcie danych o imieniu zapewni przykładowa instrukcja w postaci:

```
INSERT INTO osoby (id, nazwisko, pesel)
VALUES (4, 'Nowak', '89012345678');
```

której wykonanie zakończy się zgłoszeniem komunikatu o błędzie widocznym na rysunku 5.2. Skoro bowiem kolumna `imie` ma nadany atrybut `NOT NULL` oraz nie została jej nadana wartość domyślna, to nie można wprowadzić do tabeli wiersza niezawierającego danych dla tej kolumny.



Rysunek 5.2. *Próba pominięcia wartości wymaganej ze względu na atrybut NOT NULL*

Warto również wiedzieć, że kolejność wprowadzania danych w instrukcji `INSERT` nie musi być taka sama jak struktura kolumn w tabeli.

ĆWICZENIE

5.5 Zmiana kolejności kolumn w zapytaniu wprowadzającym dane

Dodaj do tabeli `osoby` wiersz danych tak, aby kolejność kolumn była inna niż zdefiniowana w tabeli.

Wykonanie ćwiczenia zapewni nam instrukcja:

```
INSERT INTO osoby (pesel, id, nazwisko, imie)
VALUES ('23456789012', 5, 'Arkuszewski', 'Janusz');
```

Zobaczmy teraz, co się stanie, jeśli, np. przez przeoczenie, spróbujemy dwukrotnie wykonać instrukcję wprowadzającą dane z ćwiczenia 5.2. Czy taka operacja będzie możliwa?

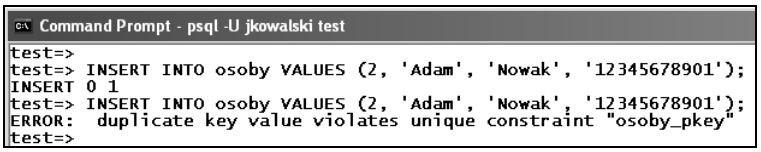
ĆWICZENIE

5.6 Próba wprowadzenia duplikatu danych

Wykonaj ponownie instrukcję z ćwiczenia 5.2. Zaobserwuj zachowanie serwera.

Po ponownym wykonaniu instrukcji `INSERT INTO` z ćwiczenia 5.2 serwer zgłosi nam błąd. Komunikat będzie miał postać widoczną na

rysunku 5.3. Przyczyna wydaje się jasna. Otóż wartości dla kolumn `imie`, `nazwisko` oraz `pesel` mogą się powtarzać, ale wartość dla kolumny `id` — już nie, jest ona bowiem kluczem podstawowym.



Rysunek 5.3. *Próba wprowadzenia duplikatu klucza podstawowego*

Jak jednak uniknąć przypadkowych duplikatów klucza głównego? Najlepiej pozostawić jego wygenerowanie serwerowi baz danych. Usuńmy więc tabelę `osoby`, wykonując instrukcję:

```
DROP TABLE osoby;
```

i utwórzmy ponownie, tym razem w taki sposób, aby kolumna `id` umożliwiała automatyczne generowanie kolejnego identyfikatora. Z rozdziału 4. wiemy, że definicja takiej kolumny będzie miała postać (należy ją wstawić do przedstawionej na początku tego rozdziału instrukcji `CREATE TABLE`):

```
id SERIAL PRIMARY KEY NOT NULL
```

ĆWICZENIE

5.7 Automatyczne generowanie wartości dla kolumny

Wprowadź do tabeli `osoby` wiersz danych w taki sposób, aby wartość dla kolumny `id` została wygenerowana automatycznie.

Wartość kolumny `id` może zostać wygenerowana automatycznie, gdyż w trakcie tworzenia tabeli została ona odpowiednio zadeklarowana. Jeśli chcemy skorzystać z takiego udogodnienia, w instrukcji wprowadzającej wiersz należy pominąć kolumnę `id`, np.:

```
INSERT INTO osoby (imie, nazwisko, pesel) VALUES ('Marceli', 'Przybysz',
↪'56789012345');
```

lub też zastosować w niej wartość `DEFAULT`:

```
INSERT INTO osoby (id, imie, nazwisko, pesel) VALUES (DEFAULT,
↪'Marceli', 'Przybysz', '56789012345');
```

Wprowadzony w ten sposób wiersz będzie miał identyfikator o jeden większy niż maksymalna wartość zapisana w kolumnie `id`. Jeśli zatem przed wykonaniem jednej z wymienionych instrukcji największą wartością[1] w kolumnie `id` było np. 14, to wprowadzony wiersz będzie miał w tej kolumnie wartość 15.

Wprowadzanie wielu wierszy

Omówiona w poprzednim punkcie instrukcji `INSERT INTO` występuje również w wersji pozwalającej na jednoczesne wstawienie do tabeli wielu wierszy. Dane każdego z nich muszą być jednak wtedy ujęte w nawias okrągły i oddzielone od siebie znakami przecinka. Taka konstrukcja będzie miała ogólną postać:

```
INSERT [INTO] tabela [(kolumna1, kolumna2, ..., kolumnaN)]
VALUES
    (wartość1A, wartość2A, ..., wartośćNA),
    (wartość1B, wartość2B, ..., wartośćNB),
    ...
    (wartość1Z, wartość2Z, ..., wartośćNZ)
```

Zaprezentowana instrukcja została rozbita na kilka wierszy w celu zwiększenia czytelności danych, choć można ją również zapisać w całości tylko w jednym wierszu (nie byłoby to jednak zbyt wygodne).

Wykonajmy trzy ćwiczenia wykorzystujące tę wersję.

ĆWICZENIE

5.8 Umieszczenie w tabeli wielu wierszy

Za pomocą pojedynczej instrukcji `INSERT INTO` wprowadź do tabeli `osoby` dane trzech osób.

Aby wykonać zadanie, możemy zastosować instrukcję SQL o postaci:

```
INSERT INTO osoby (id, imie, nazwisko, pesel)
VALUES
  (15, 'Anna', 'Kowalska', '07902657182'),
  (16, 'Janina', 'Nowak', '17912652182'),
  (17, 'Dariusz', 'Malinowski', '02699657182');
```

[1] Chodzi o największą kiedykolwiek wprowadzoną do kolumny wartość, a nie o aktualnie znajdującą się w kolumnie największą wartość.

ĆWICZENIE

5.9 Wprowadzenie wielu wierszy z pominięciem wybranych danych

Za pomocą pojedynczej instrukcji `INSERT` wprowadź do tabeli `osoby` trzy wiersze tak, aby dane z kolumny `pesel` zostały pominięte.

Również w przypadku umieszczania w tabeli wielu wierszy część kolumn może być pomijana. Możemy np. dodać dane trzech użytkowników, rezygnując z wprowadzania ich PESEL-i:

```
INSERT INTO osoby (id, imie, nazwisko)
VALUES
  (15, 'Anna', 'Kowalska'),
  (16, 'Janina', 'Nowak'),
  (17, 'Dariusz', 'Malinowski')
```

ĆWICZENIE

5.10 Wprowadzenie wielu wierszy z automatycznym generowaniem identyfikatorów

Za pomocą pojedynczej instrukcji `INSERT` wprowadź do tabeli `osoby` trzy wiersze tak, aby wartości kolumny `id` zostały wygenerowane automatycznie.

Jeśli identyfikatory wprowadzanych osób mają być generowane automatycznie, zastosujemy instrukcję (pod warunkiem oczywiście że kolumna `id` została odpowiednio zdefiniowana):

```
INSERT INTO osoby (id, imie, nazwisko, pesel)
VALUES
  (DEFAULT, 'Anna', 'Kowalska', '07902657182'),
  (DEFAULT, 'Janina', 'Nowak', '17912652182'),
  (DEFAULT, 'Dariusz', 'Malinowski', '02699657182');
```

lub:

```
INSERT INTO osoby (imie, nazwisko, pesel)
VALUES
  ('Anna', 'Kowalska', '07902657182'),
  ('Janina', 'Nowak', '17912652182'),
  ('Dariusz', 'Malinowski', '02699657182');
```

Pobieranie danych

Instrukcja SELECT

Dane zapisane w tabelach bazy można pobierać za pomocą instrukcji SELECT. Instrukcja ta posiada wiele opcji i klauzul dodatkowych, na początku poznajmy jednak jej podstawową postać, której schematyczna struktura wygląda następująco:

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

Oznacza ona: pobierz wartości wymienionych kolumn z tabeli *tabela*, które spełniają warunek *warunek*, a wyniki posortuj względem kolumn wymienionych w klauzuli ORDER BY rosnąco (ASC) lub malejąco (DESC). Aby zobaczyć, jak w praktyce działają proste zapytania typu SELECT, utworzymy tabelę przechowującą dane o osobach: imię, nazwisko oraz rok i miejsce urodzenia. Wykorzystamy do tego celu instrukcję CREATE TABLE w postaci:

```
CREATE TABLE osoba
(
  id INTEGER PRIMARY KEY,
  imie VARCHAR(20),
  nazwisko VARCHAR (35),
  rok_urodzenia INT2,
  miejsce_urodzenia VARCHAR(35)
);
```

Ponieważ użyteczny będzie jedynie rok urodzenia, a nie pełna data, dla kolumny rok_urodzenia został zastosowany typ INT2 zamiast DATE. Uprości to wykonywane operacje.

Za pomocą poznanej na wcześniejszych stronach instrukcji INSERT wprowadzimy teraz do tak utworzonej tabeli przykładowe dane, w sumie 10 wierszy:

```
INSERT INTO osoba VALUES
  (1, 'Adam', 'Kowalski', 1964, 'Bydgoszcz'),
  (2, 'Adam', 'Nowak', 1972, 'Szczecin'),
  (3, 'Andrzej', 'Kowalski', 1986, 'Nidzica'),
  (4, 'Arkadiusz', 'Malinowski', 1986, 'Kielce'),
```

```
(5, 'Andrzej', 'Malinowski', 1989, 'Kielce'),
(6, 'Krzysztof', 'Nowicki', 1986, 'Bydgoszcz'),
(7, 'Kacper', 'Adamczyk', 1971, 'Kielce'),
(8, 'Kamil', 'Andrzejczak', 1971, 'Radom'),
(9, 'Krzysztof', 'Arkuszewski', 1989, 'Szczecin'),
(10, 'Kamil', 'Borowski', 1976, 'Skierniewice');
```

Przygotowana w ten sposób tabela posłuży do wykonywania dalszych ćwiczeń.

ĆWICZENIE

5.11 Wyświetlenie całej zawartości tabeli

Wyświetl wszystkie dane z tabeli `osoba`.

Instrukcja `SELECT`, która pozwoli nam na pobranie wszystkich wierszy zawartych w tabeli osoba, ma postać:

```
SELECT * FROM osoba;
```

Symbol * oznacza tu, że interesują nas wszystkie kolumny. Efekt działania tego polecenia jest widoczny na rysunku 5.4. Widzimy, że faktycznie wyświetlone zostały wszystkie dane wprowadzone uprzednio do tabeli `osoba`. Widzimy również, że kolejność wierszy jest taka, w jakiej zostały one wprowadzone do bazy.

```
test=> SELECT * FROM osoba;
 id |   imie    |  nazwisko   | rok_urodzenia | miejsce_urodzenia
----+-----------+-------------+---------------+-------------------
  1 | Adam      | Kowalski    |          1964 | Bydgoszcz
  2 | Adam      | Nowak       |          1972 | Szczecin
  3 | Andrzej   | Kowalski    |          1986 | Nidzica
  4 | Arkadiusz | Malinowski  |          1986 | Kielce
  5 | Andrzej   | Malinowski  |          1989 | Kielce
  6 | Krzysztof | Nowicki     |          1986 | Bydgoszcz
  7 | Kacper    | Adamczyk    |          1971 | Kielce
  8 | Kamil     | Andrzejczak |          1971 | Radom
  9 | Krzysztof | Arkuszewski |          1989 | Szczecin
 10 | Kamil     | Borowski    |          1976 | Skierniewice
(10 rows)
```

Rysunek 5.4. *Efekt działania instrukcji wyświetlającej wszystkie wiersze tabeli osoba*

Sortowanie wyników

Gdybyśmy chcieli, aby wyniki zostały posortowane, musielibyśmy użyć dodatkowej klauzuli `ORDER BY`. W najprostszym przypadku sortowanie może się odbywać względem jednej kolumny. Domyślnie jest

to sortowanie w porządku rosnącym (czyli domyślnie stosowana jest opcja `ASC`). Porządek sortowania można zmienić na malejący, stosując opcję `DESC`.

ĆWICZENIE

5.12 Sortowanie w porządku rosnącym

Wyświetl wszystkie dane z tabeli `osoba` posortowane według kolumny `nazwisko` w porządku rosnącym.

Jeśli chcemy wyświetlić wszystkie wiersze tabeli posortowane względem nazwiska w porządku alfabetycznym rosnącym, powinniśmy zastosować konstrukcję:

```
SELECT * FROM osoba ORDER BY Nazwisko;
```

lub, co ma analogiczne znaczenie:

```
SELECT * FROM osoba ORDER BY Nazwisko ASC;
```

Wynik działania takiego zapytania został przedstawiony na rysunku 5.5.

```
test=> SELECT * FROM osoba ORDER BY Nazwisko;
 id |   imie    |  nazwisko   | rok_urodzenia | miejsce_urodzenia
----+-----------+-------------+---------------+-------------------
  7 | Kacper    | Adamczyk    |          1971 | Kielce
  8 | Kamil     | Andrzejczak |          1971 | Radom
  9 | Krzysztof | Arkuszewski |          1989 | Szczecin
 10 | Kamil     | Borowski    |          1976 | Skierniewice
  1 | Adam      | Kowalski    |          1964 | Bydgoszcz
  3 | Andrzej   | Kowalski    |          1986 | Nidzica
  5 | Andrzej   | Malinowski  |          1989 | Kielce
  4 | Arkadiusz | Malinowski  |          1986 | Kielce
  2 | Adam      | Nowak       |          1972 | Szczecin
  6 | Krzysztof | Nowicki     |          1986 | Bydgoszcz
(10 rows)
```

Rysunek 5.5. *Wynik sortowania względem kolumny nazwisko w porządku rosnącym*

ĆWICZENIE

5.13 Sortowanie w porządku malejącym

Wyświetl wszystkie dane z tabeli `osoba` posortowane według kolumny `nazwisko` w porządku malejącym.

W sytuacji, kiedy chcemy wyświetlić wszystkie wiersze tabeli posortowane względem kolumny `nazwisko` w porządku alfabetycznym malejącym, powinniśmy zastosować konstrukcję:

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

Wynik działania tego zapytania jest widoczny na rysunku 5.6.

```
test=> SELECT * FROM osoba ORDER BY Nazwisko DESC;
 id |   imie    |  nazwisko   | rok_urodzenia | miejsce_urodzenia
----+-----------+-------------+---------------+-------------------
  6 | Krzysztof | Nowicki     |          1986 | Bydgoszcz
  2 | Adam      | Nowak       |          1972 | Szczecin
  5 | Andrzej   | Malinowski  |          1989 | Kielce
  4 | Arkadiusz | Malinowski  |          1986 | Kielce
  1 | Adam      | Kowalski    |          1964 | Bydgoszcz
  3 | Andrzej   | Kowalski    |          1986 | Nidzica
 10 | Kamil     | Borowski    |          1976 | Skierniewice
  9 | Krzysztof | Arkuszewski |          1989 | Szczecin
  8 | Kamil     | Andrzejczak |          1971 | Radom
  7 | Kacper    | Adamczyk    |          1971 | Kielce
(10 rows)
```

Rysunek 5.6. *Wynik sortowania względem kolumny nazwisko w porządku malejącym*

Sortowanie może się również odbywać względem większej liczby kolumn. Możemy sobie na przykład zażyczyć, żeby tablica została posortowana najpierw względem nazwiska, a następnie względem roku urodzenia.

ĆWICZENIE

5.14 Sortowanie względem kilku kolumn

Wyświetl wszystkie dane z tabeli `osoba` posortowane według kolumn `nazwisko` i `miejsce_urodzenia` w porządku malejącym.

Zadanie takie zostanie zrealizowane przez instrukcję `SELECT` w postaci:

```
SELECT * FROM osoba ORDER BY nazwisko, rok_urodzenia DESC;
```

Efekt jego działania został przedstawiony na rysunku 5.7.

Pobieranie danych z wybranych kolumn

Jeżeli chcemy wyświetlić zawartość jedynie niektórych kolumn z wybranej tabeli, ich nazwy należy umieścić za słowem `SELECT`, oddzielając je od siebie znakami przecinka.

```
test=> SELECT * FROM osoba ORDER BY nazwisko, rok_urodzenia DESC;
 id |   imie    |  nazwisko   | rok_urodzenia | miejsce_urodzenia
----+-----------+-------------+---------------+-------------------
  7 | Kacper    | Adamczyk    |          1971 | Kielce
  8 | Kamil     | Andrzejczak |          1971 | Radom
  9 | Krzysztof | Arkuszewski |          1989 | Szczecin
 10 | Kamil     | Borowski    |          1976 | Skierniewice
  3 | Andrzej   | Kowalski    |          1986 | Nidzica
  1 | Adam      | Kowalski    |          1964 | Bydgoszcz
  5 | Andrzej   | Malinowski  |          1989 | Kielce
  4 | Arkadiusz | Malinowski  |          1986 | Kielce
  2 | Adam      | Nowak       |          1972 | Szczecin
  6 | Krzysztof | Nowicki     |          1986 | Bydgoszcz
(10 rows)
```

Rysunek 5.7. *Wynik sortowania względem dwóch kolumn*

ĆWICZENIE

5.15 Pobranie danych z dwu wybranych kolumn

Wyświetl zawartość kolumn `imie` i `nazwisko` z tabeli `osoba`.

Jeśli interesują nas jedynie imiona i nazwiska osób, należy wykonać polecenie:

```
SELECT imie, nazwisko FROM osoba;
```

Uzyskamy wtedy efekt widoczny na rysunku 5.8.

Rysunek 5.8. *Wynik pobrania danych z dwóch wybranych kolumn*

```
test=> SELECT imie, nazwisko FROM osoba;
   imie    |  nazwisko
-----------+-------------
 Adam      | Kowalski
 Adam      | Nowak
 Andrzej   | Kowalski
 Arkadiusz | Malinowski
 Andrzej   | Malinowski
 Krzysztof | Nowicki
 Kacper    | Adamczyk
 Kamil     | Andrzejczak
 Krzysztof | Arkuszewski
 Kamil     | Borowski
(10 rows)
```

Oczywiście pobierana w przedstawiony wyżej sposób zawartość kolumn może być również sortowana na takich samych zasadach jak przedstawione w poprzednim podrozdziale.

ĆWICZENIE

5.16 Pobranie danych z kilku kolumn z uwzględnieniem sortowania

Wyświetl zawartość kolumn `nazwisko` i `miejsce_urodzenia` z tabeli `osoba` posortowaną względem nazwiska.

Wykonanie zadania zapewni nam instrukcja:

```
SELECT nazwisko, miejsce_urodzenia FROM osoba ORDER BY nazwisko;
```

Zmiana nazw kolumn w wynikach zapytania

W pewnych sytuacjach, przy pobieraniu danych, oryginalne nazwy kolumn tabeli mogą być dla nas niewygodne i chcielibyśmy je zmienić. Taka zamiana może zostać wykonana w bardzo prosty sposób, jeśli występujące w zapytaniu `SELECT` nazwy kolumn zastąpimy 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.

ĆWICZENIE

5.17 Wykorzystanie aliasów

Zmień nazwy kolumn w wynikach zapytania pobierającego dane z tabeli `osoba`.

Jeśli zechcemy zmienić nazwę kolumny `imie` na `Imię`, `rok_urodzenia` na `Rok urodzenia`, `miejsce_urodzenia` na `Miasto`, a `nazwisko` pozostawić bez zmian, powinniśmy wykonać instrukcję:

```
SELECT imie AS "Imię", nazwisko, rok_urodzenia AS "Rok urodzenia",
↪miejsce_urodzenia AS "Miasto" FROM osoba;
```

Efekt jej działania został zaprezentowany na rysunku 5.9.

```
test=> SELECT imie AS "Imię", nazwisko, rok_urodzenia AS "Rok urodzenia", miejsc
e_urodzenia AS "Miasto" FROM osoba;
   Imię    |  nazwisko   | Rok urodzenia |    Miasto
-----------+-------------+---------------+--------------
 Adam      | Kowalski    |          1964 | Bydgoszcz
 Adam      | Nowak       |          1972 | Szczecin
 Andrzej   | Kowalski    |          1986 | Nidzica
 Arkadiusz | Malinowski  |          1986 | Kielce
 Andrzej   | Malinowski  |          1989 | Kielce
 Krzysztof | Nowicki     |          1986 | Bydgoszcz
 Kacper    | Adamczyk    |          1971 | Kielce
 Kamil     | Andrzejczak |          1971 | Radom
 Krzysztof | Arkuszewski |          1989 | Szczecin
 Kamil     | Borowski    |          1976 | Skierniewice
(10 rows)
```

Rysunek 5.9. *Nazwy kolumn w wynikach zapytania zostały zmienione*

Kryteria pobierania danych

Gdyby możliwości pobierania danych z tabeli ograniczały się do wszystkich zapisanych w niej wierszy, użyteczność baz danych byłaby bardzo ograniczona. W rzeczywistości najczęściej interesuje nas przecież wybrany podzbiór danych spełniających pewne kryteria. Otrzymanie określonego zestawu wierszy zapewni nam 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 operatory relacyjne przedstawione w tabeli 5.1 oraz operatory logiczne przedstawione w tabeli 5.2[2].

Tabela 5.1. *Operatory relacyjne w PostgreSQL*

Operator	Opis	Przykład
=	Zwraca wartość `true`, jeśli argument znajdujący się z lewej strony jest równy argumentowi znajdującemu się z prawej strony, a `false` — w przeciwnym razie.	`id=10`, `nazwisko='Kowalski'`
<>	Zwraca wartość `true`, jeśli argument znajdujący się z lewej strony jest różny od argumentu znajdującego się z prawej strony, a `false` — w przeciwnym razie.	`id<>2`, `nazwisko<>'Kowalski'`

[2] W PostgreSQL występują również inne typy operatorów. Ich listę wraz z wyjaśnieniami można znaleźć m.in. w publikacji *PostgreSQL. Leksykon kieszonkowy* (*http://helion.pl/ksiazki/psqllk.htm*).

Tabela 5.1. *Operatory relacyjne w PostgreSQL — ciąg dalszy*

Operator	Opis	Przykład
`!=`	Takie samo znaczenie jak <>.	`id!=2, nazwisko!='Kowalski'`
`<`	Zwraca wartość `true`, jeśli argument znajdujący się z lewej strony jest mniejszy od argumentu znajdującego się z prawej strony, a `false` — w przeciwnym razie.	`id<10`
`>`	Zwraca wartość `true`, jeśli argument znajdujący się z lewej strony jest większy od argumentu znajdującego się z prawej strony, a `false` — w przeciwnym razie.	`id>10`
`<=`	Zwraca wartość `true`, jeśli argument znajdujący się z lewej strony jest mniejszy lub równy argumentowi znajdującemu się z prawej strony, a `false` — w przeciwnym razie.	`id<=10`
`>=`	Zwraca wartość `true`, jeśli argument znajdujący się z lewej strony jest większy lub równy argumentowi znajdującemu się z prawej strony, a `false` — w przeciwnym razie.	`id>=10`
`IS NULL`	Zwraca wartość `true`, jeśli argument znajdujący się z lewej strony jest równy `NULL`, lub `false` — w przeciwnym przypadku.	`adres IS NULL, id IS NULL`
`IS NOT NULL`	Zwraca wartość `true`, jeśli argument znajdujący się z lewej strony jest różny od `NULL`, a `false` — w przeciwnym razie.	`adres IS NOT NULL, id IS NOT NULL`
`ISNULL`	Takie samo znaczenie jak `IS NULL`.	`adres ISNULL, id ISNULL`
`NOTNULL`	Takie samo znaczenie jak `IS NOT NULL`.	`adres NOTNULL, id NOTNULL`

Tabela 5.1. *Operatory relacyjne w PostgreSQL — ciąg dalszy*

Operator	Opis	Przykład
`IS TRUE`	Zwraca wartość `true`, jeśli argument znajdujący się z lewej strony ma wartość `true`, a `false` — w przeciwnym razie.	`pole IS TRUE`
`IS NOT TRUE`	Zwraca wartość `true`, jeśli argument znajdujący się z lewej strony ma wartość różną od `true`, a `false` — w przeciwnym razie.	`pole IS NOT TRUE`
`IS FALSE`	Zwraca wartość `true`, jeśli argument znajdujący się z lewej strony ma wartość `false`, a `false` — w przeciwnym razie.	`pole IS FALSE`
`IS NOT FALSE`	Zwraca wartość `true`, jeśli argument znajdujący się z lewej strony ma wartość różną od `false`, a `false` w przeciwnym razie.	`pole IS NOT FALSE`
`IS UNKNOWN`	Zwraca wartość `true`, jeśli argument znajdujący się z lewej strony ma wartość nieokreśloną, a `false` — w przeciwnym razie.	`pole IS UNKNOWN`
`IS NOT UNKNOWN`	Zwraca wartość `true`, jeśli argument znajdujący się z lewej strony nie ma wartości nieokreślonej, a `false` — w przeciwnym razie.	`pole IS NOT UNKNOWN`
`BETWEEN` *N* `AND` *M*	Zwraca wartość `true`, jeśli argument znajdujący się z lewej strony ma wartość z przedziału od *N* do *M*, a `false` — w przeciwnym razie.	`id BETWEEN 10 AND 20`
`NOT BETWEEN` *N* `AND` *M*	Zwraca wartość `true`, jeśli argument znajdujący się z lewej strony nie ma wartości z przedziału od *N* do *M*, a `false` — w przeciwnym razie.	`id NOT BETWEEN 10 AND 20`

Tabela 5.1. *Operatory relacyjne w PostgreSQL — ciąg dalszy*

Operator	Opis	Przykład
IN	Zwraca wartość true, jeśli argument znajdujący się z lewej strony jest równy jednej z wartości wymienionych w nawiasie okrągłym za operatorem, a false — w przeciwnym razie.	id IN(1, 3, 5), nazwisko IN('Kowalski ', 'Nowak')
NOT IN	Zwraca wartość true, jeśli argument znajdujący się z lewej strony nie jest równy jednej z wartości wymienionych w nawiasie okrągłym za operatorem, a false — w przeciwnym razie.	id NOT IN(1, 3, 5), nazwisko NOT IN('Kowalski ', 'Nowak')

Tabela 5.2. *Operatory logiczne w PostgreSQL*

Operator	Opis	Przykład
AND	Logiczny iloczyn. Zwraca wartość TRUE wtedy i tylko wtedy, gdy oba argumenty mają wartość TRUE. W każdym innym przypadku zwraca wartość FALSE.	imie='Jan' AND Nazwisko='Kowalski'
OR	Logiczna suma. Zwraca wartość TRUE, kiedy przynajmniej jeden z argumentów ma wartość TRUE. W każdym innym przypadku zwraca wartość FALSE.	imie='Jan' OR imie='Andrzej'
NOT	Logiczna negacja. Zmienia wartość argumentu na przeciwną. Jeśli wartością argumentu było TRUE, wynikiem będzie FALSE, a jeśli wartością argumentu było FALSE, wynikiem będzie TRUE.	NOT Aktywny

Porównywanie danych ze zdefiniowanym wzorcem może być również wykonywane za pomocą operatorów: LIKE, NOT LIKE, SIMILAR TO i NOT SIMILAR TO, a także wyrażeń regularnych wg standardu POSIX. Wyrażenie z operatorem LIKE będzie miało ogólną postać:

```
ciąg LIKE wzorzec
```

Jego wynikiem będzie wartość `true`, jeśli `ciąg` pasuje do wzorca, a `false` — w przeciwnym razie. Argument *wzorzec* może zawierać dwa znaki specjalne. Pierwszy z nich to `%`, który zastępuje dowolną liczbę znaków, drugi to _ (podkreślenie), zastępujący dokładnie jeden znak. Oznacza to, że do przykładowego wzorca `Jan%` będą pasowały ciągi `Jan`, `Janusz`, `Janek`, `Janowski` itp., a do wzorca `Warszaw_` będą pasowały ciągi `Warszawa`, `Warszawy`, `Warszawo` itp.

Wyrażenie zawierające operator `NOT LIKE` ma postać:

```
ciąg NOT LIKE wzorzec
```

i działa odwrotnie do `LIKE`, czyli zwraca wartość `true`, jeśli ciąg nie jest zgodny ze wzorcem, a wartość `false`, jeżeli jest zgodny.

Operatory `SIMILAR TO` i `NOT SIMILAR TO` działają analogicznie do `LIKE` i `NOT LIKE`, z tą różnicą, że wzorzec jest interpretowany jako wyrażenie regularne zgodne ze składnią tego typu wyrażeń stosowanych w SQL.

ĆWICZENIE

5.18 Kryteria dla pojedynczej kolumny

Pobierz wszystkie wiersze tabeli `osoba`, które w polu `nazwisko` mają zapisaną wartość `Kowalski`.

Skoro interesują nas wiersze tabeli, które w kolumnie `nazwisko` zawierają wartość `Kowalski`, powinniśmy zastosować warunek `nazwisko='Kowalski'`, a więc pełne zapytanie będzie miało postać:

```
SELECT * FROM osoba WHERE nazwisko='Kowalski';
```

Wynik jego działania został przedstawiony na rysunku 5.10.

```
test=> SELECT * FROM osoba WHERE Nazwisko='Kowalski';
 id |  imie   | nazwisko | rok_urodzenia | miejsce_urodzenia
----+---------+----------+---------------+-------------------
  1 | Adam    | Kowalski |          1964 | Bydgoszcz
  3 | Andrzej | Kowalski |          1986 | Nidzica
(2 rows)
```

Rysunek 5.10. *Wyszukiwanie ze względu na nazwisko*

ĆWICZENIE

5.19 Użycie operatora większości

Wykorzystaj operator większości do pobrania listy osób urodzonych po roku 1985.

Zapytanie SQL będzie miało postać:

```
SELECT * FROM osoba WHERE rok_urodzenia > 1985;
```

Efekt jego działania został przedstawiony na rysunku 5.11. Analogiczny efekt moglibyśmy również osiągnąć, wykorzystując operator >= w postaci:

```
SELECT * FROM osoba WHERE rok_urodzenia >= 1986;
```

```
test=> SELECT * FROM osoba WHERE Rok_Urodzenia > 1985;
 id |   imie    |  nazwisko   | rok_urodzenia | miejsce_urodzenia
----+-----------+-------------+---------------+-------------------
  3 | Andrzej   | Kowalski    |          1986 | Nidzica
  4 | Arkadiusz | Malinowski  |          1986 | Kielce
  5 | Andrzej   | Malinowski  |          1989 | Kielce
  6 | Krzysztof | Nowicki     |          1986 | Bydgoszcz
  9 | Krzysztof | Arkuszewski |          1989 | Szczecin
(5 rows)
```

Rysunek 5.11. Wynik zapytania korzystającego z operatora większości

ĆWICZENIE

5.20 Użycie operatora logicznego AND

Użyj operatora logicznego `AND` do uzyskania listy osób o identyfikatorach z przedziału `3 – 6`.

Aby uzyskać w wyniku zapytania wartości pól z podanego zakresu, należy użyć dwóch warunków — `id >= 3` i `id <= 6` — połączonych operatorem `AND`, a więc konstrukcji w postaci:

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

Należy ją rozumieć jako: wyświetl takie wiersze z tabeli `osoba`, których wartość w kolumnie `id` jest większa od 3 lub równa oraz mniejsza od 6 lub równa. Efekt jej działania został przedstawiony na rysunku 5.12.

Jeśli chcemy w prosty sposób wybrać dane z pewnego przedziału, możemy skorzystać z operatora `BETWEEN`, a nie z dwóch warunków połączonych operatorem `AND`. Zamiast wtedy pisać:

```
test=> SELECT * FROM osoba WHERE id >= 3 AND id <= 6;
 id |   imie    |  nazwisko  | rok_urodzenia | miejsce_urodzenia
----+-----------+------------+---------------+-------------------
  3 | Andrzej   | Kowalski   |          1986 | Nidzica
  4 | Arkadiusz | Malinowski |          1986 | Kielce
  5 | Andrzej   | Malinowski |          1989 | Kielce
  6 | Krzysztof | Nowicki    |          1986 | Bydgoszcz
(4 rows)
```

Rysunek 5.12. *Działanie operatora AND*

```
kolumna >= początek_zakresu AND kolumna <= koniec_zakresu
```

tak jak to miało miejsce w poprzednim ćwiczeniu, możemy zastosować konstrukcję:

```
kolumna BETWEEN początek_zakresu AND koniec_zakresu
```

ĆWICZENIE

5.21 Użycie operatora BETWEEN

Użyj operatora `BETWEEN` do uzyskania listy osób o identyfikatorach z przedziału `3 – 6`.

Jeśli do pobrania wierszy o identyfikatorach z zakresu `3 – 6` ma zostać wykorzystany operator `BETWEEN`, należy użyć instrukcji:

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

Efekt działania będzie taki sam jak na rysunku 5.12.

Jeśli chcielibyśmy, aby w wynikach zostały uwzględnione wartości z pewnego zbioru, a nie przedziału, musielibyśmy użyć zarówno serii instrukcji warunkowych połączonych operatorami logicznymi, jak i operatora `IN`. Działanie będzie takie samo, jednak ta druga możliwość pozwala na prostszy i dużo czytelniejszy zapis instrukcji. Operator `IN` ma ogólną postać:

```
wartość IN (wartość1, wartość2, ..., wartośćN)
```

ĆWICZENIE

5.22 Wybranie wierszy o identyfikatorach z określonego zbioru

Wyświetl dane osób o identyfikatorach `3`, `5` i `7`, wykorzystując instrukcje warunkowe połączone wybranym operatorem logicznym.

Jeśli chcemy uzyskać dane osób o identyfikatorach 3, 5 i 7, powinniśmy zastosować trzy instrukcje warunkowe: `id = 3`, `id = 5`, `id = 7` połączone za pomocą operatora `OR` (czyli sumy logicznej). Instrukcja taka będzie miała postać:

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

Oznacza ona: wyświetl wiersze z tabeli `osoba`, których wartość w kolumnie `id` jest równa 3 lub równa 5, lub równa 7. Po jej wykonaniu na ekranie ujrzymy widok taki, jak zaprezentowany na rysunku 5.13.

```
test=> SELECT * FROM osoba WHERE id=3 OR id=5 OR id=7;
 id |  imie   |  nazwisko  | rok_urodzenia | miejsce_urodzenia
----+---------+------------+---------------+-------------------
  3 | Andrzej | Kowalski   |          1986 | Nidzica
  5 | Andrzej | Malinowski |          1989 | Kielce
  7 | Kacper  | Adamczyk   |          1971 | Kielce
(3 rows)
```

Rysunek 5.13. *Wiersze o identyfikatorach z określonego zbioru*

ĆWICZENIE

5.23 Użycie operatora IN

Wyświetl dane osób o identyfikatorach 3, 5 i 7, wykorzystując operator `IN`.

Jeśli do wyświetlenia rekordów o identyfikatorach 3, 5 i 7 ma zostać użyty operator `IN`, należy zastosować instrukcję:

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

Efekt jej wykonania będzie taki sam jak zaprezentowany na rysunku 5.13.

Odwrotnością `IN` jest `NOT IN`, które pozwala na pobranie danych nienależących do wymienionego zbioru. Również i w tym wypadku możliwe jest użycie ekwiwalentu w postaci serii instrukcji warunkowych połączonych operatorami logicznymi. Sam operator `NOT IN` ma ogólną postać:

```
wartość NOT IN (wartość1, wartość2, ..., wartośćN)
```

ĆWICZENIE

5.24 Użycie operatora różności i operatorów logicznych

Wyświetl dane osób o identyfikatorach różnych od 1, 3, 5, 7 i 9. Użyj operatorów warunkowych i logicznych.

Niezbędne będzie tu użycie 5 warunków w ogólnej postaci `id <> wartość` połączonych ze sobą operatorem logicznym `AND`. Cała instrukcja będzie więc miała postać:

```
SELECT * FROM osoba
WHERE id <> 1 AND id <> 3 AND id <> 5 AND id <> 7 AND id <> 9;
```

a efekt jej wykonania został przedstawiony na rysunku 5.14.

```
test=> SELECT * FROM osoba WHERE id <> 1 AND id <> 3 AND id <> 5 AND id <> 7 AND
 id <> 9;
 id |   imie    |  nazwisko   | rok_urodzenia | miejsce_urodzenia
----+-----------+-------------+---------------+-------------------
  2 | Adam      | Nowak       |          1972 | Szczecin
  4 | Arkadiusz | Malinowski  |          1986 | Kielce
  6 | Krzysztof | Nowicki     |          1986 | Bydgoszcz
  8 | Kamil     | Andrzejczak |          1971 | Radom
 10 | Kamil     | Borowski    |          1976 | Skierniewice
(5 rows)
```

Rysunek 5.14. *Wynik działania zapytania z ćwiczenia 5.24*

ĆWICZENIE

5.25 Użycie operatora NOT IN

Wyświetl dane osób o identyfikatorach różnych od 1, 3, 5, 7 i 9. Użyj operatora `NOT IN`.

Użycie operatora `NOT IN` uprości instrukcję przedstawioną w ćwiczeniu 5.24. W tym przypadku będzie ona bowiem miała postać:

```
SELECT * FROM osoba WHERE id NOT IN(1, 3, 5, 7, 9);
```

Wynik jej działania będzie taki sam jak widoczny na rysunku 5.14.

Operator `LIKE` pozwala na pobranie z tabeli wierszy, których wybrane pola pasują do zdefiniowanego przez nas wzorca.

ĆWICZENIE

5.26 Dane pasujące do określonego wzorca

Wyświetl dane wszystkich osób, których imiona zaczynają się od ciągu `Ka`.

Jeśli chcemy poznać dane wszystkich osób, których imiona zaczynają się od ciągu `Ka`, powinniśmy zastosować instrukcję:

```
SELECT * FROM osoba WHERE imie LIKE 'Ka%';
```

Efekt działania tego polecenia jest widoczny na rysunku 5.15.

```
test=> SELECT * FROM osoba WHERE imie LIKE 'Ka%';
 id |  imie  |  nazwisko   | rok_urodzenia | miejsce_urodzenia
----+--------+-------------+---------------+-------------------
  7 | Kacper | Adamczyk    |          1971 | Kielce
  8 | Kamil  | Andrzejczak |          1971 | Radom
 10 | Kamil  | Borowski    |          1976 | Skierniewice
(3 rows)
```

Rysunek 5.15. *Wyświetlenie danych pasujących do zdefiniowanego wzorca*

Oczywiście warunek w klauzuli `WHERE` nie musi ograniczać się do danych pobieranych z jednej kolumny, można stosować warunki złożone połączone operatorami logicznymi.

ĆWICZENIE

5.27 Wiele kolumn w klauzuli WHERE

Wyświetl znajdujące się w tabeli `osoba` dane osób, których imiona zaczynają się na literę A, urodzonych po roku 1970 w Kielcach lub w Szczecinie.

W celu wykonania ćwiczenia należy zastosować instrukcję:

```
SELECT * FROM osoba WHERE imie LIKE 'A%' AND rok_urodzenia > 1970 AND
↪miejsce_urodzenia IN ('Kielce', 'Szczecin');
```

Efekt działania tego zapytania został zaprezentowany na rysunku 5.16.

```
test=> SELECT * FROM osoba WHERE imie LIKE 'A%' AND rok_urodzenia > 1970 AND mie
jsce_urodzenia IN ('Kielce', 'Szczecin');
 id |   imie    |  nazwisko  | rok_urodzenia | miejsce_urodzenia
----+-----------+------------+---------------+-------------------
  2 | Adam      | Nowak      |          1972 | Szczecin
  4 | Arkadiusz | Malinowski |          1986 | Kielce
  5 | Andrzej   | Malinowski |          1989 | Kielce
(3 rows)
```

Rysunek 5.16. *Wyniki zapytania z warunkami dotyczącymi wielu kolumn*

Niepowtarzalność wierszy

Instrukcja `SELECT` może być również uzupełniona o klauzulę `DISTINCT`, która gwarantuje niepowtarzalność wierszy wynikowych, innymi słowy, eliminuje duplikaty z wyników zapytania. Załóżmy, że chcemy się dowiedzieć, jakie różne nazwiska noszą osoby, których dane są zapisane w tabeli `pracownicy`. Jeśli zastosujemy typową instrukcję:

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

w wynikach znajdą się podwójne dane dla nazwisk Kowalski i Malinowski (rysunek 5.17). Nie o to nam jednak chodziło. Do uzyskania prawidłowych wyników niezbędne będzie więc użycie słowa `DISTINCT`.

Rysunek 5.17. *W wynikach zapytania pojawiły się duplikaty danych*

```
test=> SELECT nazwisko FROM osoba ORDER BY nazwisko;
  nazwisko
-------------
 Adamczyk
 Andrzejczak
 Arkuszewski
 Borowski
 Kowalski
 Kowalski
 Malinowski
 Malinowski
 Nowak
 Nowicki
(10 rows)
```

ĆWICZENIE

5.28 Użycie klauzuli DISTINCT

Napisz zapytanie, które pobierze z tabeli `osoba` listę nazwisk. W wynikach nie mogą się pojawić duplikaty danych.

Duplikaty danych wyeliminujemy, umieszczając za słowem `SELECT` słowo `DISTINCT`. Instrukcja będzie więc miała postać:

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

Wynik jej działania został zaprezentowany na rysunku 5.18.

```
test=> SELECT DISTINCT nazwisko FROM osoba ORDER BY nazwisko;
  nazwisko
-------------
 Adamczyk
 Andrzejczak
 Arkuszewski
 Borowski
 Kowalski
 Malinowski
 Nowak
 Nowicki
(8 rows)
```

Rysunek 5.18. *Brak duplikatów w wynikach zapytania*

Ograniczanie wyników zapytań

W wyniku wykonania zapytania pobierającego dane możemy otrzymać bardzo wiele wierszy wynikowych. Jednak liczba ta może zostać ograniczona za pomocą klauzul `LIMIT` i `OFFSET` umieszczonych na samym końcu zapytania. Instrukcja `SELECT` przyjmie wtedy 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`.

ĆWICZENIE

5.29 Ograniczenie liczby wierszy wynikowych

Wykonaj instrukcję pobierającą wszystkie dane z tabeli `osoba` posortowane względem kolumny `nazwisko`. Ogranicz liczbę wyświetlanych wierszy do `5`.

Ograniczenie do pięciu liczby wierszy pobieranych z tabeli `osoba` zapewni nam instrukcja:

```
SELECT * FROM osoba ORDER BY nazwisko LIMIT 5;
```

Efekt jej działania został zaprezentowany na rysunku 5.19.

```
test=> SELECT * FROM osoba ORDER BY nazwisko LIMIT 5;
 id |   imie    |  nazwisko    | rok_urodzenia | miejsce_urodzenia
----+-----------+--------------+---------------+-------------------
  7 | Kacper    | Adamczyk     |          1971 | Kielce
  8 | Kamil     | Andrzejczak  |          1971 | Radom
  9 | Krzysztof | Arkuszewski  |          1989 | Szczecin
 10 | Kamil     | Borowski     |          1976 | Skierniewice
  1 | Adam      | Kowalski     |          1964 | Bydgoszcz
(5 rows)
```

Rysunek 5.19. *Ograniczenie liczby wierszy pobieranych z tabeli osoba*

ĆWICZENIE

5.30 Pobieranie wierszy, począwszy od konkretnej pozycji

Wykonaj instrukcję pobierającą wszystkie dane z tabeli `osoba` posortowane względem kolumny `nazwisko` tak, aby wyświetlone zostały 3 wiersze, poczynając od 4.

Jeśli chcemy, aby wyświetlanie rozpoczęło się od 4 wiersza i zostały w sumie wyświetlone 3 wiersze, powinniśmy zastosować instrukcję:

```
SELECT * FROM osoba ORDER BY nazwisko LIMIT 3 OFFSET 3;
```

Wynik jej działania jest widoczny na rysunku 5.20.

```
test=> SELECT * FROM osoba ORDER BY Nazwisko LIMIT 3 OFFSET 3;
 id |  imie   | nazwisko | rok_urodzenia | miejsce_urodzenia
----+---------+----------+---------------+-------------------
 10 | Kamil   | Borowski |          1976 | Skierniewice
  1 | Adam    | Kowalski |          1964 | Bydgoszcz
  3 | Andrzej | Kowalski |          1986 | Nidzica
(3 rows)
```

Rysunek 5.20. *Wynik działania instrukcji z ćwiczenia 5.30*

Modyfikacja danych

Dane zapisane w tabelach mogą być zmieniane i modyfikowane. Służy do tego instrukcja `UPDATE`, która ma ogólną postać:

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

Oznacza ona: zmień w tabeli *tabela* — w wierszach spełniających warunek *warunek* — pole *kolumna1* na *wartość1*, pole *kolumna2* na *wartość2* itd. Klauzula WHERE jest opcjonalna i może zostać pominięta; w takiej sytuacji zmianie ulegną wszystkie wskazane wiersze w tabeli.

ĆWICZENIE

5.31 Zmiana wszystkich wartości we wskazanej kolumnie

Zmień zawartość kolumny miejsce_urodzenia w tabeli osoba tak, aby wszystkie wiersze zawierały ciąg Krosno.

Wykonanie ćwiczenia zmodyfikuje zawartość WSZYSTKICH wierszy w tabeli osoba. Nie pojawi się przy tym żadne ostrzeżenie czy też pytanie o potwierdzenie chęci wykonania instrukcji. Przywrócenie oryginalnej zawartości tabeli będzie wymagało ponownego wprowadzenia danych.

Zmianę wszystkich wartości w kolumnie miejsce_urodzenia zapewni instrukcja:

```
UPDATE osoba SET miejsce_urodzenia='Krosno';
```

Kiedy ją wykonamy, zobaczymy komunikat, że miała ona wpływ na 10 wierszy tabeli (rysunek 5.21). Jeśli teraz wykonamy instrukcję SELECT pobierającą wszystkie dane z tabeli osoba, zobaczymy, że faktycznie wszystkie wiersze kolumny miejsce_urodzenia zostały zmienione. Jest to widoczne na rysunku 5.22.

```
test=> UPDATE osoba SET miejsce_urodzenia='Krosno';
UPDATE 10
test=>
```

Rysunek 5.21. *Wykonanie instrukcji aktualizującej dane w tabeli*

Zazwyczaj jednak aktualizuje się tylko jeden lub kilka wierszy, niezbędne jest więc zastosowanie klauzuli WHERE. Przykładowo mogłoby się okazać, że w tablicy zapisaliśmy błędny rok urodzenia Andrzeja Malinowskiego, któremu w tabeli został przypisany identyfikator 5, i należy zamienić wartość 1989 na 1988.

```
test=> SELECT * FROM osoba;
 id |   imie    |  nazwisko   | rok_urodzenia | miejsce_urodzenia
----+-----------+-------------+---------------+-------------------
  1 | Adam      | Kowalski    |          1964 | Krosno
  2 | Adam      | Nowak       |          1972 | Krosno
  3 | Andrzej   | Kowalski    |          1986 | Krosno
  4 | Arkadiusz | Malinowski  |          1986 | Krosno
  5 | Andrzej   | Malinowski  |          1989 | Krosno
  6 | Krzysztof | Nowicki     |          1986 | Krosno
  7 | Kacper    | Adamczyk    |          1971 | Krosno
  8 | Kamil     | Andrzejczak |          1971 | Krosno
  9 | Krzysztof | Arkuszewski |          1989 | Krosno
 10 | Kamil     | Borowski    |          1976 | Krosno
(10 rows)
```

Rysunek 5.22. *Wszystkie wiersze w kolumnie miejsce_urodzenia zostały zmodyfikowane*

ĆWICZENIE

5.32 Uaktualnienie wybranego wiersza w tabeli

Zmień rok urodzenia osoby o identyfikatorze `5` na `1998`.

W celu wykonania ćwiczenia należy zastosować instrukcję:

```
UPDATE osoba SET rok_urodzenia=1988 WHERE id=5;
```

Warunek `id=5` został zastosowany, bowiem pole `id` jest kluczem podstawowym jednoznacznie identyfikującym każdy rekord. Po wykonaniu tego zapytania, podobnie jak w przypadku ćwiczenia 5.31, otrzymamy informację, ile rekordów zostało zmodyfikowanych (`UPDATE`).

Nic nie stoi również na przeszkodzie, aby jednocześnie zmodyfikować kilka pól w danym wierszu. Moglibyśmy na przykład zmienić od razu imię, rok i miejsce urodzenia danej osoby.

ĆWICZENIE

5.33 Jednoczesna modyfikacja kilku pól

Zmodyfikuj dane postaci o identyfikatorze `8` tak, aby opisywały inną osobę.

Aby wykonać to zadanie, możemy wykonać instrukcję:

```
UPDATE osoba SET nazwisko='Andrzejewski', rok_urodzenia=1990,
↪miejsce_urodzenia='Kielce' WHERE id=8;
```

Tym samym Kamil Andrzejczak urodzony w 1971 roku w Radomiu stanie się Kamilem Adrzejewskim urodzonym w 1990 roku w Kielcach.

Warunek w klauzuli WHERE może wykorzystywać operatory opisane w podrozdziale „Kryteria pobierania danych", może więc jednocześnie wskazywać więcej niż jeden rekord do modyfikacji. Jeśli zatem wykryjemy, że osoby o identyfikatorach 4 i 5 mają błędnie przypisane miejsce urodzenia, którym nie jest Kielce, ale Radom, możemy to szybko naprawić.

ĆWICZENIE

5.34 Modyfikacja kilku wybranych rekordów

Zmień miejsce urodzenia osób o identyfikatorach 4 i 5 na Radom.

Zmiana ta będzie wymagała instrukcji:

```
UPDATE osoba SET miejsce_urodzenia='Radom' WHERE id=4 OR id=5;
```

lub

```
UPDATE osoba SET miejsce_urodzenia='Radom' WHERE id IN(4,5);
```

Usuwanie danych

Wiemy już, jak dodawać dane do tabeli, jak je pobierać i modyfikować. Do omówienia została jeszcze równie ważna kwestia usuwania wierszy z tabel. 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 warunek zostanie pominięty, zostaną usunięte wszystkie dane (podobnie jak w przypadku instrukcji UPDATE, gdzie pominięcie warunku powodowało modyfikację wszystkich wierszy tabeli).

ĆWICZENIE

5.35 Usunięcie wszystkich danych z tabeli

Usuń wszystkie dane z tabeli osoba.

Aby usunąć wszystkie dane z tabeli osoba, trzeba wykonać instrukcję:

```
DELETE FROM osoba;
```

Po jej wykonaniu tabela osoba nie będzie zawierała żadnych danych. Taką konstrukcję należy stosować z uwagą, gdyż serwer nie wygeneruje żadnego ostrzeżenia czy dodatkowego pytania. Wpisanie powyższej konstrukcji i zatwierdzenie jej klawiszem *Enter* spowoduje natychmiastowe skasowanie danych!

Najczęściej więc stosuje się instrukcję DELETE zawierającą warunek w klauzuli WHERE, który ma taką samą postać jak opisywana podczas omawiania instrukcji SELECT. Wykonajmy kilka ćwiczeń obrazujących to zagadnienie.

ĆWICZENIE

5.36 Usuwanie konkretnego wiersza tabeli

Usuń z tabeli osoba dane osoby o identyfikatorze 5.

Jeśli chcemy usunąć z tabeli osoba dane osoby o identyfikatorze 5, zastosujemy instrukcję:

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

Serwer odpowie komunikatem:

```
DELETE 1
```

Komunikat ten wskazuje, że wykonanie zapytania zakończyło się sukcesem oraz że został usunięty 1 wiersz.

ĆWICZENIE

5.37 Usuwanie kilku wybranych rekordów

Usuń dane osób o identyfikatorach równych 3, 5 i 7.

Aby najprościej usunąć dane osób o identyfikatorach 3, 5 i 7, należy wykonać instrukcję:

```
DELETE FROM osoba WHERE id IN (3, 5, 7);
```

Można również zastosować serię warunków połączonych operatorem OR:

```
DELETE FROM osoba WHERE id=3 OR id=5 OR id=7;
```

ĆWICZENIE

5.38 Usuwanie rekordów z określonego przedziału

Usuń dane osób o identyfikatorach z przedziału 3 – 5.

W celu usunięcia danych osób o identyfikatorach z przedziału 3 – 5 użyjemy instrukcji:

```
DELETE FROM osoba WHERE id BETWEEN 3 AND 5;
```

lub:

```
DELETE FROM osoba WHERE id >= 4 AND id <= 5;
```

Oczywiście kryteria usuwania nie muszą dotyczyć tylko kolumn liczbowych. Warunek klauzuli WHERE może dotyczyć dowolnych kolumn, dowolnych typów.

ĆWICZENIE

5.39 Usuwanie rekordów ze względu na ciąg znaków

Usuń z tabeli osoba dane osób o nazwisku Kowalski.

Wykonanie ćwiczenia zapewni nam instrukcja:

```
DELETE FROM osoba WHERE nazwisko='Kowalski';
```