helion.pl
KSIĘGARNIA INTERNETOWA · WYDAWNICTWO

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

SQL. Ćwiczenia praktyczne

Autor: Marcin Lis
ISBN: 83-246-0621-1
Format: A5, stron: 152

Poznaj zasady pracy z bazami danych

- Projektowanie baz i tabel
- Wprowadzanie i wybieranie danych
- Konstruowanie złożonych zapytań

Bazy danych są „kręgosłupem" niemal każdej aplikacji. Rozbudowane systemy finansowe, aplikacje korporacyjne, portale i sklepy internetowe, a nawet proste programy do fakturowania opierają się na bazach danych. Rynek systemów zarządzania bazami danych jest bardzo zróżnicowany – można znaleźć zarówno ogromne aplikacje komercyjne, jak i systemy dostępne nieodpłatnie. Na szczęście dla programistów i użytkowników z każdym z takich systemów można „porozumieć się" za pomocą języka o nazwie SQL. Oczywiście, każda z baz danych ma specyficzne dla siebie instrukcje, jednak rdzeń języka jest wspólny.

Dzięki książce „SQL. Ćwiczenia praktyczne" na podstawie prostych i gotowych do rozwiązania przykładów poznasz podstawy tego języka. Nauczysz się planować i projektować tabele, umieszczać w nich dane oraz przetwarzać je. Dowiesz się, w jaki sposób formułować zapytania języka SQL, za pomocą których można wprowadzać dane, wybierać je według określonych kryteriów i przeprowadzać obliczenia. Poznasz również sposoby pobierania danych z wielu tabel za pomocą złączeń oraz przeczytasz o transakcjach i więzach integralności.

- Zasady projektowania tabel baz danych
- Typy danych
- Wprowadzanie danych do bazy
- Pobieranie danych
- Modyfikowanie i usuwanie danych
- Złączenia
- Transakcje w systemach baz danych

Po przeczytaniu tej książki będziesz w stanie sprawnie posługiwać się systemami baz danych opartymi na SQL

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

Spis treści

Pobieranie danych z tabel

Podstawy instrukcji SELECT

Dane zapisane w tabelach bazy danych można pobierać za pomocą instrukcji `SELECT`. Jej podstawowa postać ogólnie wygląda tak:

```
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 nawiasy kwadratowe są opcjonalne.

Aby przećwiczyć działanie tej wersji instrukcji `SELECT`, utworzymy przykładową tabelę `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;

- `placa` — typu `DECIMAL(7, 2)`, z atrybutem `NOT NULL`, zawierająca miesięczne wynagrodzenie pracownika;
- `stanowisko` — typu `VARCHAR(20)`, z atrybutem `NOT NULL`, zawierająca stanowisko pracownika[1];
- `pesel` — typu `CHAR(11)`, zawierająca PESEL pracownika.

Taka tabela zostanie utworzona za pomocą instrukcji:

```
CREATE TABLE pracownicy
(
  id INTEGER PRIMARY KEY,
  imie VARCHAR(20) NOT NULL,
  nazwisko VARCHAR(30) NOT NULL,
  placa DECIMAL(7, 2) NOT NULL,
  stanowisko VARCHAR(20),
  pesel CHAR(11)
)
```

Wypełnimy ją przykładowymi danymi:

```
INSERT INTO pracownicy VALUES (1, 'Adam', 'Kowalski', 1624.50,
'magazynier', '12345678901');
INSERT INTO pracownicy VALUES (2, 'Adam', 'Nowak', 3760.00, 'kierownik',
'92345678901');
INSERT INTO pracownicy VALUES (3, 'Andrzej', 'Kowalski', 4200.00,
'kierownik', '72345678901');
INSERT INTO pracownicy VALUES (4, 'Arkadiusz', 'Malinowski', 1600.00,
'kierowca', '92345678909');
INSERT INTO pracownicy VALUES (5, 'Andrzej', 'Malinowski', 1450.00,
'sprzedawca', NULL);
INSERT INTO pracownicy VALUES (6, 'Krzysztof', 'Nowicki', 1300.00,
'sprzedawca', NULL);
INSERT INTO pracownicy VALUES (7, 'Kacper', 'Adamczyk', 1610.50,
'serwisant', '92341678903');
INSERT INTO pracownicy VALUES (8, 'Kamil', 'Andrzejczak', 1200.00,
'asystent', NULL);
INSERT INTO pracownicy VALUES (9, 'Krzysztof', 'Arkuszewski', 1500,
'magazynier', '02343678913');
INSERT INTO pracownicy VALUES (10, 'Kamil', 'Borowski', 1600.00,
'sprzedawca', '32349678913');
```

[1] Jak wiesz z rozdziału 1., w realnie działającej bazie nazwy stanowisk powinny raczej znajdować się w oddzielnej tabeli, jednak umieszczenie ich w tabeli `pracownicy` ułatwi nam wykonywanie dalszych ćwiczeń.

ĆWICZENIE

4.1 Wyświetlenie zawartości wybranej tabeli

Użyj instrukcji SELECT do wyświetlenia zawartości tabeli pracownicy.

Instrukcja SELECT, która pobierze wszystkie wiersze z tabeli pracownicy, ma postać:

```
SELECT * FROM pracownicy;
```

Symbol * oznacza tu, że interesuje nas zawartość wszystkich kolumn. Przykładowy efekt działania tego polecenia został zaprezentowany na rysunku 4.1. Widać na nim, że faktycznie wyświetlone zostały wszystkie dane wprowadzone uprzednio do tabeli pracownicy, jak również to, że kolejność wierszy jest taka, w jakiej były one wprowadzane do bazy.

```
+----+-----------+-------------+---------+------------+-------------+
| id | imie      | nazwisko    | placa   | stanowisko | pesel       |
+----+-----------+-------------+---------+------------+-------------+
|  1 | Adam      | Kowalski    | 1624.50 | magazynier | 12345678901 |
|  2 | Adam      | Nowak       | 3760.00 | kierownik  | 92345678901 |
|  3 | Andrzej   | Kowalski    | 4200.00 | kierownik  | 72345678901 |
|  4 | Arkadiusz | Malinowski  | 1600.00 | kierowca   | 92345678909 |
|  5 | Andrzej   | Malinowski  | 1450.00 | sprzedawca | NULL        |
|  6 | Krzysztof | Nowicki     | 1300.00 | sprzedawca | NULL        |
|  7 | Kacper    | Adamczyk    | 1610.50 | serwisant  | 92341678903 |
|  8 | Kamil     | Andrzejczak | 1200.00 | asystent   | NULL        |
|  9 | Krzysztof | Arkuszewski | 1500.00 | magazynier | 02343678913 |
| 10 | Kamil     | Borowski    | 1600.00 | sprzedawca | 32349678913 |
+----+-----------+-------------+---------+------------+-------------+
10 rows in set (0.01 sec)
```

Rysunek 4.1. *Efekt działania instrukcji SELECT pobierającej wszystkie dane z tabeli pracownicy*

Aby wyświetlić zawartość tylko niektórych kolumn z wybranej tabeli, nazwy tych kolumn należy umieścić za słowem SELECT, oddzielając poszczególne nazwy znakami przecinka. Przykładowo mogą nas interesować jedynie imiona, nazwiska i stanowiska pracowników.

ĆWICZENIE

4.2 Pobieranie danych z wybranych kolumn

Pobierz z tabeli pracownicy dane o imionach, nazwiskach i stanowiskach.

Wykonanie ćwiczenia zapewni nam instrukcja:

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

Efekt działania został zaprezentowany na rysunku 4.2.

Rysunek 4.2. *Wynik pobrania danych z trzech wybranych kolumn*

```
+-----------+-------------+------------+
| imie      | nazwisko    | stanowisko |
+-----------+-------------+------------+
| Adam      | Kowalski    | magazynier |
| Adam      | Nowak       | kierownik  |
| Andrzej   | Kowalski    | kierownik  |
| Arkadiusz | Malinowski  | kierowca   |
| Andrzej   | Malinowski  | sprzedawca |
| Krzysztof | Nowicki     | sprzedawca |
| Kacper    | Adamczyk    | serwisant  |
| Kamil     | Andrzejczak | asystent   |
| Krzysztof | Arkuszewski | magazynier |
| Kamil     | Borowski    | sprzedawca |
+-----------+-------------+------------+
10 rows in set (0.05 sec)
```

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

```
nazwa_kolumny AS alias
```

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

ĆWICZENIE

4.3 Zmiana nazw kolumn w wynikach zapytania

Pobierz z tabeli `pracownicy` dane o imionach, nazwiskach i stanowiskach, tak aby kolumna `placa` miała nazwę `wynagrodzenie`.

Wykonanie ćwiczenia zapewni nam instrukcja:

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

Efekt wykonania ćwiczenia został zaprezentowany na rysunku 4.3.

Rysunek 4.3. *Nazwy kolumn w wynikach zapytania zostały zmienione*

```
+-----------+-------------+---------------+
| imie      | nazwisko    | wynagrodzenie |
+-----------+-------------+---------------+
| Adam      | Kowalski    | 1624.50       |
| Adam      | Nowak       | 3760.00       |
| Andrzej   | Kowalski    | 4200.00       |
| Arkadiusz | Malinowski  | 1600.00       |
| Andrzej   | Malinowski  | 1450.00       |
| Krzysztof | Nowicki     | 1300.00       |
| Kacper    | Adamczyk    | 1610.50       |
| Kamil     | Andrzejczak | 1200.00       |
| Krzysztof | Arkuszewski | 1500.00       |
| Kamil     | Borowski    | 1600.00       |
+-----------+-------------+---------------+
10 rows in set (0.02 sec)
```

Sortowanie wyników zapytań

Wyniki zapytania typu SELECT mogą być sortowane, co umożliwia klauzula ORDER BY. Sortowanie może odbywać się w porządku rosnącym bądź malejącym względem jednej bądź kilku kolumn.

ĆWICZENIE

4.4 Sortowanie w porządku rosnącym

Wyświetl zawartość tablicy pracownicy posortowaną względem kolumny nazwisko w porządku rosnącym.

Zatem aby wyświetlić wszystkie wiersze tabeli posortowane względem kolumny nazwisko rosnąco w porządku alfabetycznym, należy zastosować konstrukcję:

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

lub prościej:

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

opcja ASC jest bowiem opcją domyślną. Wynik działania takiego zapytania został zaprezentowany na rysunku 4.4.

```
+----+-----------+-------------+---------+------------+-------------+
| id | imie      | nazwisko    | placa   | stanowisko | pesel       |
+----+-----------+-------------+---------+------------+-------------+
|  7 | Kacper    | Adamczyk    | 1610.50 | serwisant  | 92341678903 |
|  8 | Kamil     | Andrzejczak | 1200.00 | asystent   | NULL        |
|  9 | Krzysztof | Arkuszewski | 1500.00 | magazynier | 02343678913 |
| 10 | Kamil     | Borowski    | 1600.00 | sprzedawca | 32349678913 |
|  1 | Adam      | Kowalski    | 1624.50 | magazynier | 12345678901 |
|  3 | Andrzej   | Kowalski    | 4200.00 | kierownik  | 72345678901 |
|  5 | Andrzej   | Malinowski  | 1450.00 | sprzedawca | NULL        |
|  4 | Arkadiusz | Malinowski  | 1600.00 | kierowca   | 92345678909 |
|  2 | Adam      | Nowak       | 3760.00 | kierownik  | 92345678901 |
|  6 | Krzysztof | Nowicki     | 1300.00 | sprzedawca | NULL        |
+----+-----------+-------------+---------+------------+-------------+
10 rows in set (0.05 sec)
```

Rysunek 4.4. *Wynik sortowania tabeli pracownicy względem kolumny nazwisko w porządku rosnącym*

ĆWICZENIE

4.5 Sortowanie w porządku malejącym

Wyświetl zawartość tablicy `pracownicy` posortowaną względem kolumny `nazwisko` w porządku malejącym.

Aby wyświetlić wszystkie wiersze tabeli posortowane względem kolumny `nazwisko` malejąco w porządku alfabetycznym, należy zastosować konstrukcję:

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

Wynik działania tego zapytania jest widoczny na rysunku 4.5.

```
+----+-----------+-------------+---------+------------+-------------+
| id | imie      | nazwisko    | placa   | stanowisko | pesel       |
+----+-----------+-------------+---------+------------+-------------+
|  6 | Krzysztof | Nowicki     | 1300.00 | sprzedawca | NULL        |
|  2 | Adam      | Nowak       | 3760.00 | kierownik  | 92345678901 |
|  4 | Arkadiusz | Malinowski  | 1600.00 | kierowca   | 92345678909 |
|  5 | Andrzej   | Malinowski  | 1450.00 | sprzedawca | NULL        |
|  1 | Adam      | Kowalski    | 1624.50 | magazynier | 12345678901 |
|  3 | Andrzej   | Kowalski    | 4200.00 | kierownik  | 72345678901 |
| 10 | Kamil     | Borowski    | 1600.00 | sprzedawca | 32349678913 |
|  9 | Krzysztof | Arkuszewski | 1500.00 | magazynier | 02343678913 |
|  8 | Kamil     | Andrzejczak | 1200.00 | asystent   | NULL        |
|  7 | Kacper    | Adamczyk    | 1610.50 | serwisant  | 92341678903 |
+----+-----------+-------------+---------+------------+-------------+
10 rows in set (0.00 sec)
```

Rysunek 4.5. *Wynik sortowania tabeli pracownicy względem kolumny nazwisko w porządku malejącym*

Sortowanie może się również odbywać względem większej liczby kolumn. Możemy sobie na przykład zażyczyć, aby tablica została 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.

ĆWICZENIE

4.6 Sortowanie względem kilku kolumn

Wyświetl zawartość tablicy `pracownicy` posortowaną względem kolumn `nazwisko` (w porządku rosnącym) i `placa` (w porządku malejącym).

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

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

Efekt jego działania został przedstawiony na rysunku 4.6.

```
+----+-----------+-------------+---------+------------+-------------+
| id | imie      | nazwisko    | placa   | stanowisko | pesel       |
+----+-----------+-------------+---------+------------+-------------+
|  7 | Kacper    | Adamczyk    | 1610.50 | serwisant  | 92341678903 |
|  8 | Kamil     | Andrzejczak | 1200.00 | asystent   | NULL        |
|  9 | Krzysztof | Arkuszewski | 1500.00 | magazynier | 02343678913 |
| 10 | Kamil     | Borowski    | 1600.00 | sprzedawca | 32349678913 |
|  3 | Andrzej   | Kowalski    | 4200.00 | kierownik  | 72345678901 |
|  1 | Adam      | Kowalski    | 1624.50 | magazynier | 12345678901 |
|  4 | Arkadiusz | Malinowski  | 1600.00 | kierowca   | 92345678909 |
|  5 | Andrzej   | Malinowski  | 1450.00 | sprzedawca | NULL        |
|  2 | Adam      | Nowak       | 3760.00 | kierownik  | 92345678901 |
|  6 | Krzysztof | Nowicki     | 1300.00 | sprzedawca | NULL        |
+----+-----------+-------------+---------+------------+-------------+
10 rows in set (0.00 sec)
```

Rysunek 4.6. *Wynik sortowania tabeli pracownicy względem dwóch kolumn*

Kryteria pobierania danych

Możliwości pobierania danych z tabeli nie ograniczają się, rzecz jasna, do wszystkich zapisanych w niej wierszy. Najczęściej interesuje nas przecież tylko pewien podzbiór danych, spełniających zadane kryteria. Otrzymanie określonego zestawu wierszy zapewnia 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*

Operator	Opis	Przykład
=	Operator równości. Zwraca wartość `TRUE`, jeśli argument znajdujący się z lewej strony jest równy argumentowi znajdującemu się z prawej strony, w przeciwnym razie zwraca `FALSE`.	`id=10, nazwisko= 'Kowalski'`

[2] Poszczególne dialekty SQL mogą również zawierać inne operatory.

Tabela 5.1. *Operatory relacyjne (ciąg dalszy)*

Operator	Opis	Przykład
<>	Zwraca wartość TRUE, jeśli argument znajdujący się z lewej strony jest różny od argumentu znajdującego się z prawej strony, w przeciwnym razie zwraca FALSE.	id<>2, nazwisko <>'Kowalski'
!=	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, w przeciwnym razie zwraca FALSE.	id<10
>	Zwraca wartość TRUE, jeśli argument znajdujący się z lewej strony jest większy od argumentu znajdującego się z prawej strony, w przeciwnym razie zwraca FALSE.	id>10
<=	Zwraca wartość TRUE, jeśli argument znajdujący się z lewej strony jest mniejszy lub równy argumentowi znajdującemu się z prawej strony, w przeciwnym razie zwraca FALSE.	id<=10
>=	Zwraca wartość TRUE, jeśli argument znajdujący się z lewej strony jest większy lub równy argumentowi znajdującemu się z prawej strony, w przeciwnym razie zwraca FALSE.	id>=10
IS NULL	Zwraca wartość TRUE, jeśli argument znajdujący się z lewej strony jest równy NULL, w przeciwnym razie zwraca FALSE.	adres IS NULL, id IS NULL
IS NOT NULL	Zwraca wartość TRUE, jeśli argument znajdujący się z lewej strony jest różny od NULL, w przeciwnym razie zwraca FALSE.	adres IS NOT NULL, id IS NOT NULL
BETWEEN N AND M	Zwraca wartość TRUE, jeśli argument znajdujący się z lewej strony ma wartość z przedziału od N do M, w przeciwnym razie zwraca FALSE.	id BETWEEN 10 AND 20

Tabela 5.1. *Operatory relacyjne (ciąg dalszy)*

Operator	Opis	Przykład
`NOT BETWEEN N AND M`	Zwraca wartość `TRUE`, jeśli argument znajdujący się z lewej strony nie ma wartości z przedziału od `N` do `M`, w przeciwnym razie zwraca `FALSE`.	`id NOT BETWEEN 10 AND 20`
`IN`	Zwraca wartość `TRUE`, jeśli argument znajdujący się z lewej strony jest równy jednej z wartości wymienionych w nawiasie okrągłym za operatorem, w przeciwnym razie zwraca `FALSE`.	`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, w przeciwnym razie zwraca `FALSE`.	`id NOT IN(1, 3, 5), nazwisko NOT IN('Kowalski', 'Nowak')`

Tabela 5.2. *Operatory logiczne*

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'`
`XOR`	Logiczna różnica symetryczna (logiczna alternatywa wykluczająca). Zwraca wartość `TRUE`, kiedy oba argumenty mają różne wartości logiczne, oraz wartość `FALSE`, kiedy oba argumenty mają takie same wartości logiczne.	`kolumna1 XOR kolumna2, pole XOR 64`
`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`

Oprócz przedstawionych w powyższych tabelach operatorów relacyjnych i logicznych stosunkowo często wykorzystywane są także dwa wyrażenia operujące na ciągach znaków. Są to LIKE i NOT LIKE[3]. Wywołanie funkcji LIKE ma postać:

```
wyrażenie LIKE wzorzec
```

Zwraca ona wartość TRUE, jeśli wyrażenie pasuje do wzorca, w przeciwnym razie zwraca FALSE. Jako wyrażenie zazwyczaj jest stosowana nazwa kolumny. Argument *wzorzec* może zawierać dwa znaki specjalne. Pierwszy z nich to %, który zastępuje dowolną liczbę znaków, drugi to _ (podkreślenie), który zastępuje 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.

Funkcja NOT LIKE ma postać:

```
wyrażenie NOT LIKE wzorzec
```

i działa odwrotnie do LIKE, czyli zwraca wartość TRUE, jeśli wyrażenie nie jest zgodne ze wzorcem, lub wartość FALSE, kiedy jest zgodne.

Wykonajmy zatem serię ćwiczeń, które w praktyce pokażą, jak wykorzystywać warunki i wyrażenia w klauzuli WHERE. Operować będziemy na znanej Ci z poprzednich przykładów tabeli pracownicy.

ĆWICZENIE

4.7 Kryteria dla pojedynczej kolumny

Wyświetl dane pracowników o nazwisku Kowalski.

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

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

Wynik jego działania został przedstawiony na rysunku 4.7.

[3] W zależności od systemu bazy danych określa się je również mianem operatorów lub funkcji.

```
+----+---------+----------+---------+------------+-------------+
| id | imie    | nazwisko | placa   | stanowisko | pesel       |
+----+---------+----------+---------+------------+-------------+
|  1 | Adam    | Kowalski | 1624.50 | magazynier | 12345678901 |
|  3 | Andrzej | Kowalski | 4200.00 | kierownik  | 72345678901 |
+----+---------+----------+---------+------------+-------------+
2 rows in set (0.03 sec)
```

Rysunek 4.7. *Wyszukiwanie ze względu na nazwisko*

ĆWICZENIE

4.8 Użycie operatora mniejszości

Wykorzystaj operator mniejszości do pobrania listy osób o zarobkach poniżej 1600 zł.

Zapytanie SQL będzie miało postać:

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

Efekt jego działania został przedstawiony na rysunku 4.8.

```
+----+-----------+-------------+---------+------------+-------------+
| id | imie      | nazwisko    | placa   | stanowisko | pesel       |
+----+-----------+-------------+---------+------------+-------------+
|  5 | Andrzej   | Malinowski  | 1450.00 | sprzedawca | NULL        |
|  6 | Krzysztof | Nowicki     | 1300.00 | sprzedawca | NULL        |
|  8 | Kamil     | Andrzejczak | 1200.00 | asystent   | NULL        |
|  9 | Krzysztof | Arkuszewski | 1500.00 | magazynier | 02343678913 |
+----+-----------+-------------+---------+------------+-------------+
4 rows in set (0.02 sec)
```

Rysunek 4.8. *Dane osób o zarobkach poniżej 1600 zł.*

Często do uzyskania pożądanych danych niezbędne jest użycie kilku warunków połączonych operatorem logicznym. Taki przykład przedstawia następne ćwiczenie.

ĆWICZENIE

4.9 Wykorzystanie operatorów relacyjnych i iloczynu logicznego

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 o postaci:

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

co należy rozumieć jako: wyświetl takie wiersze z tabeli pracownicy, których wartość w kolumnie id jest większa lub równa 3 i jednocześnie mniejsza lub równa 6. Efekt jej działania został przedstawiony na rysunku 4.9.

```
+----+-----------+------------+---------+------------+-------------+
| id | imie      | nazwisko   | placa   | stanowisko | pesel       |
+----+-----------+------------+---------+------------+-------------+
|  3 | Andrzej   | Kowalski   | 4200.00 | kierownik  | 72345678901 |
|  4 | Arkadiusz | Malinowski | 1600.00 | kierowca   | 92345678909 |
|  5 | Andrzej   | Malinowski | 1450.00 | sprzedawca | NULL        |
|  6 | Krzysztof | Nowicki    | 1300.00 | sprzedawca | NULL        |
+----+-----------+------------+---------+------------+-------------+
4 rows in set (0.03 sec)
```

Rysunek 4.9. *Działanie iloczynu logicznego*

Jeśli chcielibyśmy w prosty sposób wybrać dane z pewnego przedziału, zamiast z dwóch warunków połączonych operatorem AND możemy skorzystać z operatora BETWEEN. Wtedy zamiast pisać:

kolumna >= *początek_zakresu* AND *kolumna* <= *koniec_zakresu*

jak w poprzednim ćwiczeniu, możemy zastosować konstrukcję:

kolumna BETWEEN *początek_zakresu* AND *koniec_zakresu*

ĆWICZENIE

4.10 Operator BETWEEN zamiast warunku złożonego

Pobierz listę pracowników o płacach od 1400 do 1600 złotych. Użyj operatora BETWEEN. Posortuj dane względem płacy rosnąco.

Jeśli do pobrania listy osób o płacach z zakresu 1400 – 1600 ma zostać wykorzystany operator BETWEEN, należy użyć instrukcji:

```
SELECT * FROM pracownicy WHERE placa BETWEEN 1400 AND 1600 ORDER BY placa;
```

Niekiedy konieczne jest pobranie danych o wartościach należących do pewnego zbioru, a nie przedziału. W takiej sytuacji można użyć zarówno serii instrukcji warunkowych połączonych operatorami logicznymi, jak i operatora IN. Działanie będzie takie samo, choć ta druga możliwość pozwala na prostszy i dużo czytelniejszy zapis instrukcji. Operator IN ma ogólną postać:

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

Oba sposoby zostaną wykorzystane w dwóch następnych ćwiczeniach.

ĆWICZENIE

4.11 Wybranie wierszy o identyfikatorach z określonego zbioru

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

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

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

Co oznacza: wyświetl takie wiersze z tabeli `pracownicy`, których wartość w kolumnie `id` jest równa 3 lub równa 5 lub równa 7. Po wykonaniu tej instrukcji na ekranie ujrzymy widok taki jak zaprezentowany na rysunku 4.10.

```
+----+---------+------------+---------+------------+-------------+
| id | imie    | nazwisko   | placa   | stanowisko | pesel       |
+----+---------+------------+---------+------------+-------------+
|  3 | Andrzej | Kowalski   | 4200.00 | kierownik  | 72345678901 |
|  5 | Andrzej | Malinowski | 1450.00 | sprzedawca | NULL        |
|  7 | Kacper  | Adamczyk   | 1610.50 | serwisant  | 92341678903 |
+----+---------+------------+---------+------------+-------------+
3 rows in set (0.00 sec)
```

Rysunek 4.10. *Wiersze o identyfikatorach z określonego zbioru*

ĆWICZENIE

4.12 Użycie operatora IN zamiast operatorów relacyjnych

Wykonaj zadanie z ćwiczenia 4.11, wykorzystując operator `IN`.

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

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

Efekt jej wykonania będzie taki sam jak zaprezentowany na rysunku 4.10.

Wybranie z tabeli danych, które pasują do określonego wzorca, umożliwi opisany na początku rozdziału operator `LIKE`.

ĆWICZENIE

4.13 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 pracownicy WHERE imie LIKE 'Ka%';
```

Efekt działania tego polecenia jest widoczny na rysunku 4.11.

```
+----+--------+-------------+---------+------------+-------------+
| id | imie   | nazwisko    | placa   | stanowisko | pesel       |
+----+--------+-------------+---------+------------+-------------+
|  7 | Kacper | Adamczyk    | 1610.50 | serwisant  | 92341678903 |
|  8 | Kamil  | Andrzejczak | 1200.00 | asystent   | NULL        |
| 10 | Kamil  | Borowski    | 1600.00 | sprzedawca | 32349678913 |
+----+--------+-------------+---------+------------+-------------+
3 rows in set (0.20 sec)
```

Rysunek 4.11. *Wyświetlenie danych pasujących do wybranego wzorca*

ĆWICZENIE

4.14 Wyszukiwanie wartości pustych

Wyszukaj w tabeli identyfikatory oraz imiona i nazwiska pracowników, dla których baza nie zawiera numerów PESEL.

Wyszukanie wszystkich pracowników, dla których nie został wprowadzony numer PESEL, zapewni nam operator `IS NULL`. Zapytanie będzie miało postać:

```
SELECT id, imie, nazwisko FROM pracownicy WHERE pesel IS NULL;
```

a jego wynik został przedstawiony na rysunku 4.12.

Rysunek 4.12. *Dane osób, dla których brakuje numerów PESEL*

```
+----+-----------+-------------+
| id | imie      | nazwisko    |
+----+-----------+-------------+
|  5 | Andrzej   | Malinowski  |
|  6 | Krzysztof | Nowicki     |
|  8 | Kamil     | Andrzejczak |
+----+-----------+-------------+
3 rows in set (0.00 sec)
```

Warunek w klauzuli WHERE nie musi ograniczać się do danych pobieranych z jednej kolumny; można stosować warunki złożone połączone operatorami logicznymi.

ĆWICZENIE

4.15 Wiele kolumn w klauzuli WHERE

Wyświetl znajdujące się w tabeli pracownicy dane osób, których płaca jest większa niż 1400 zł, pracujących na stanowiskach innych niż kierownik, dla których znany jest numer PESEL.

Aby wykonać ćwiczenie, należy zastosować instrukcję:

```
SELECT * FROM pracownicy WHERE placa > 1400 AND stanowisko <>
'kierownik' AND pesel IS NOT NULL;
```

Efekt działania tego zapytania został zaprezentowany na rysunku 4.13.

```
+----+-----------+-------------+---------+------------+-------------+
| id | imie      | nazwisko    | placa   | stanowisko | pesel       |
+----+-----------+-------------+---------+------------+-------------+
|  1 | Adam      | Kowalski    | 1624.50 | magazynier | 12345678901 |
|  4 | Arkadiusz | Malinowski  | 1600.00 | kierowca   | 92345678909 |
|  7 | Kacper    | Adamczyk    | 1610.50 | serwisant  | 92341678903 |
|  9 | Krzysztof | Arkuszewski | 1500.00 | magazynier | 02343678913 |
| 10 | Kamil     | Borowski    | 1600.00 | sprzedawca | 32349678913 |
+----+-----------+-------------+---------+------------+-------------+
5 rows in set (0.00 sec)
```

Rysunek 4.13. *Wyświetlenie danych spełniających kilka warunków*

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 4.14). Nie o to nam jednak chodziło. Do uzyskania prawidłowych wyników niezbędne będzie więc użycie słowa DISTINCT.

Rysunek 4.14. *W wynikach zapytania pojawiły się duplikaty danych*

```
+--------------+
| nazwisko     |
+--------------+
| Adamczyk     |
| Andrzejewski |
| Arkuszewski  |
| Borowski     |
| Kowalski     |
| Kowalski     |
| Malinowski   |
| Malinowski   |
| Nowak        |
| Nowicki      |
+--------------+
10 rows in set (0.00 sec)
```

ĆWICZENIE

4.16 Użycie klauzuli DISTINCT

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

Wyeliminowanie duplikatów danych uzyskamy, umieszczając za słowem `SELECT` słowo `DISTINCT`. Instrukcja będzie więc miała postać:

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

Wynik jej działania został zaprezentowany na rysunku 4.15.

Rysunek 4.15. *Podwójne dane zostały usunięte z wyników zapytania*

```
+--------------+
| nazwisko     |
+--------------+
| Adamczyk     |
| Andrzejewski |
| Arkuszewski  |
| Borowski     |
| Kowalski     |
| Malinowski   |
| Nowak        |
| Nowicki      |
+--------------+
8 rows in set (0.00 sec)
```