

Wstęp do MySQL: projekt bazy Sklep

Możesz podążać za tutorialiem poprzez stronę: https://wiki.ostrowski.net.pl/php_mysql/sklep.php

W tym artykule wykorzystamy przykładową bazę danych sklepu internetowego z trzema tabelami: klienci, towary i zamówienia. Przedstawimy definicje tabel w SQL oraz dodamy po 10 przykładowych rekordów do każdej z nich. Następnie omówimy różne rodzaje zapytań na tych danych, w tym:

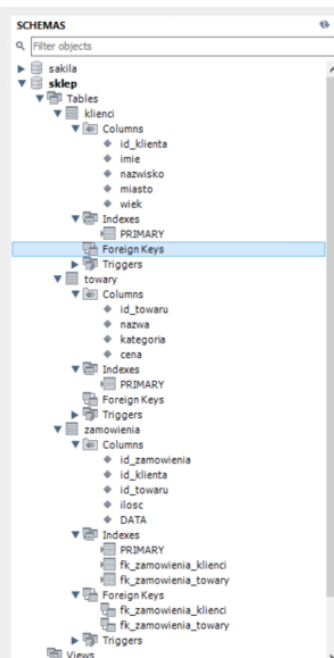
- podstawowe zapytania SELECT z klauzulami WHERE, ORDER BY i LIMIT,
- różne rodzaje JOIN (INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN) oraz zagnieżdżone łączenia (łączenia wielu tabel),
- zapytania z użyciem GROUP BY i funkcji agregujących (COUNT, SUM, AVG itp.),
- modyfikujące zapytania INSERT, UPDATE i DELETE,
- przykład zapytania z podzapytaniem (subquery),
- prostą procedurę składowaną.

Każdy rodzaj zapytania zostanie zilustrowany przykładowym kodem SQL działającym na naszej bazie. Na zakończenie artykułu zamieścimy krótkie podsumowanie omawianych zagadnień.

Tabele i przykładowe dane

Zacznijmy od zdefiniowania struktury bazy danych sklepu. Utworzymy trzy tabele:

```
CREATE TABLE klienci (  
  id_klienta INT PRIMARY KEY,  
  imie VARCHAR(50),  
  nazwisko VARCHAR(50),  
  miasto VARCHAR(50),  
  wiek INT  
);  
  
CREATE TABLE towary (  
  id_towaru INT PRIMARY KEY,  
  nazwa VARCHAR(100),  
  kategoria VARCHAR(50),  
  cena DECIMAL(10,2)  
);  
  
CREATE TABLE zamowienia (  
  id_zamowienia INT PRIMARY KEY,  
  id_klienta INT,  
  id_towaru INT,  
  ilosc INT,  
  DATA DATE,  
  CONSTRAINT fk_zamowienia_klienci  
    FOREIGN KEY (id_klienta)  
    REFERENCES klienci(id_klienta)
```



Po wykonaniu skryptu powinniśmy uzyskać coś takiego

```

ON UPDATE CASCADE
ON DELETE CASCADE,
CONSTRAINT fk_zamowienia_towary
FOREIGN KEY (id_towaru)
REFERENCES towary(id_towaru)
ON UPDATE CASCADE
ON DELETE RESTRICT
);

```

Wyjaśnienia:

PRIMARY KEY przy kolumnach id_klienta, id_towaru i id_zamowienia jednoznacznie identyfikuje wiersz w każdej z tabel.

W tabeli zamowienia:

- fk_zamowienia_klienci to klucz obcy wskazujący na klienci(id_klienta).

ON UPDATE CASCADE – zmiana wartości id_klienta w tabeli klienci automatycznie zaktualizuje wszystkie powiązane wiersze w zamowienia.

ON DELETE CASCADE – usunięcie klienta spowoduje automatyczne skasowanie jego zamówień.

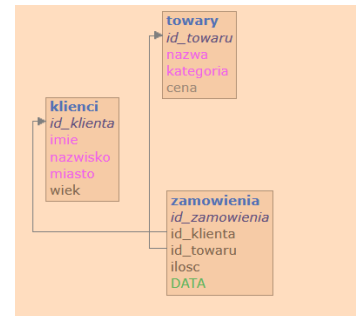
- fk_zamowienia_towary to klucz obcy wskazujący na towary(id_towaru).

ON UPDATE CASCADE – zmiana id_towaru w towary automatycznie zaktualizuje referencje w zamowienia.

ON DELETE RESTRICT – nie pozwoli usunąć towaru, który jest powiązany z przynajmniej jednym zamówieniem.

W tabeli klienci przechowujemy dane klientów (imię, nazwisko, miasto, wiek), w tabeli towary – informacje o produktach (nazwa, kategoria, cena), a w tabeli zamowienia – dane dotyczące zamówień (powiązanie z klientem i towarem, ilość, data).

Następnie dodamy przykładowe rekordy do każdej tabeli:



Schemat bazy wygenerowany za pomocą narzędzia [Adminer](#)

```

INSERT INTO klienci
(id_klienta, imie, nazwisko, miasto, wiek)
VALUES
(1, 'Jan', 'Kowalski', 'Warszawa', 34),
(2, 'Anna', 'Nowak', 'Kraków', 28),
(3, 'Piotr', 'Wiśniewski', 'Poznań', 45),
(4, 'Katarzyna', 'Wójcik', 'Gdańsk', 51),
(5, 'Michał', 'Kamiński', 'Wrocław', 39),
(6, 'Agnieszka', 'Lewandowska', 'Katowice', 23),
(7, 'Tomasz', 'Zieliński', 'Warszawa', 62),
(8, 'Ewa', 'Szymańska', 'Lublin', 31),
(9, 'Adam', 'Dąbrowski', 'Łódź', 27),

```

	id_klienta	imie	nazwisko	miasto	wiek
▶	1	Jan	Kowalski	Warszawa	34
	2	Anna	Nowak	Kraków	28
	3	Piotr	Wiśniewski	Poznań	45
	4	Katarzyna	Wójcik	Gdańsk	51
	5	Michał	Kamiński	Wrocław	39
	6	Agnieszka	Lewandowska	Katowice	23
	7	Tomasz	Zieliński	Warszawa	62
	8	Ewa	Szymańska	Lublin	31
	9	Adam	Dąbrowski	Łódź	27
	10	Magdalena	Janikowska	Poznań	44

Po wykonaniu powinniśmy uzyskać coś takiego

```
(10, 'Magdalena', 'Jankowska', 'Poznań', 44);
```

```
INSERT INTO towary
(id_towaru, nazwa, kategoria, cena)
VALUES
(1, 'Telewizor 55"', 'Elektronika', 2499.99),
(2, 'Laptop', 'Elektronika', 3299.00),
(3, 'Smartfon', 'Elektronika', 1999.49),
(4, 'Regał na książki', 'Meble', 459.20),
(5, 'Krzesło biurowe', 'Meble', 349.00),
(6, 'T-shirt męski', 'Odzież', 59.99),
(7, 'Sukienka damska', 'Odzież', 129.50),
(8, 'Buty sportowe', 'Obuwie', 179.99),
(9, 'Słuchawki bezprzewodowe', 'Elektronika', 149.99),
(10, 'Książka "SQL dla początkujących"', 'Książki', 79.90);
```

id_towaru	nazwa	kategoria	cena
1	Telewizor 55"	Elektronika	2499.99
2	Laptop	Elektronika	3299.00
3	Smartfon	Elektronika	1999.49
4	Regał na książki	Meble	459.20
5	Krzesło biurowe	Meble	349.00
6	T-shirt męski	Odzież	59.99
7	Sukienka damska	Odzież	129.50
8	Buty sportowe	Obuwie	179.99
9	Słuchawki bezprzewodowe	Elektronika	149.99
10	Książka "SQL dla początkujących"	Książki	79.90

Po wykonaniu
powinniśmy uzyskać coś
takiego

```
INSERT INTO zamowienia
(id_zamowienia, id_klienta, id_towaru, ilosc, DATA)
VALUES
(1, 1, 1, 1, '2024-01-15'),
(2, 2, 3, 2, '2024-01-17'),
(3, 1, 2, 1, '2024-02-03'),
(4, 3, 5, 4, '2024-02-20'),
(5, 4, 4, 2, '2024-03-05'),
(6, 5, 8, 1, '2024-03-15'),
(7, 6, 10, 3, '2024-03-17'),
(8, 7, 9, 1, '2024-03-18'),
(9, 2, 6, 5, '2024-03-20'),
(10, 1, 7, 2, '2024-04-01');
```

id_zamowienia	id_klienta	id_towaru	ilosc	DATA
1	1	1	1	2024-01-15
2	2	3	2	2024-01-17
3	1	2	1	2024-02-03
4	3	5	4	2024-02-20
5	4	4	2	2024-03-05
6	5	8	1	2024-03-15
7	6	10	3	2024-03-17
8	7	9	1	2024-03-18
9	2	6	5	2024-03-20
10	1	7	2	2024-04-01

Po wykonaniu
powinniśmy uzyskać coś
takiego

Taka konfiguracja zapewnia realistyczne dane: kilku klientów zamawia różne towary w różnych ilościach i terminach.

Podstawowe zapytania SELECT

Podstawowym zapytaniem służącym do pobierania danych jest SELECT. Pozwala ono wybrać jedną lub więcej kolumn z tabeli. Najprostsze zapytanie SELECT zwraca wszystkie kolumny i wiersze danej tabeli. Przykładowo:

```
SELECT * FROM klienci;
```

```
1 • SELECT * FROM klienci;
```

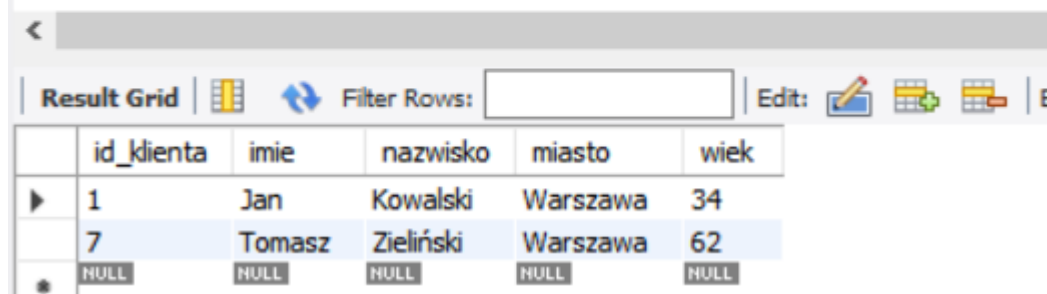


	id_klienta	imie	nazwisko	miasto	wiek
▶	1	Jan	Kowalski	Warszawa	34
	2	Anna	Nowak	Kraków	28
	3	Piotr	Wiśniewski	Poznań	45
	4	Katarzyna	Wójcik	Gdańsk	51
	5	Michał	Kamiński	Wrocław	39
	6	Agnieszka	Lewandowska	Katowice	23
	7	Tomasz	Zieliński	Warszawa	62
	8	Ewa	Szymańska	Lublin	31
	9	Adam	Dąbrowski	Łódź	27
	10	Magdalena	Jankowska	Poznań	44
•	NULL	NULL	NULL	NULL	NULL

to zapytanie pobiera wszystkie dane z tabeli klienci. Często potrzebujemy jednak filtrować rekordy według pewnych kryteriów. Służy do tego klauzula WHERE, np. aby wybrać klientów z konkretnego miasta:

```
SELECT * FROM klienci WHERE miasto = 'Warszawa';
```

```
1 SELECT * FROM klienci WHERE miasto = 'Warszawa';
```



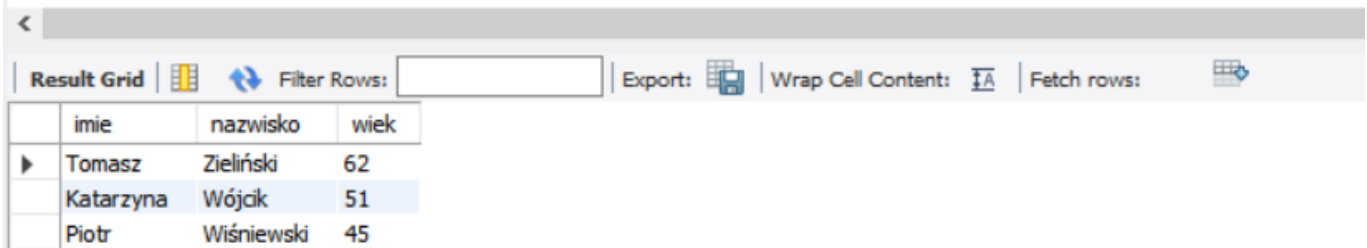
	id_klienta	imie	nazwisko	miasto	wiek
▶	1	Jan	Kowalski	Warszawa	34
	7	Tomasz	Zieliński	Warszawa	62
•	NULL	NULL	NULL	NULL	NULL

Możemy też ograniczać liczbę wyników i ich kolejność. Klauzula ORDER BY sortuje wyniki względem podanych kolumn (domyślnie rosnąco), natomiast LIMIT ogranicza liczbę zwracanych wierszy

Przykłady:

```
SELECT imie,nazwisko,wiek FROM klienci WHERE wiek > 30 ORDER BY wiek DESC  
LIMIT 3;
```

```
1 SELECT imie,nazwisko,wiek FROM klienci WHERE wiek > 30 ORDER BY wiek DESC LIMIT 3;
```



	imie	nazwisko	wiek
▶	Tomasz	Zieliński	62
	Katarzyna	Wójcik	51
	Piotr	Wiśniewski	45

Powyższe zapytanie wybiera imiona, nazwiska i wiek klientów starszych niż 30 lat, sortuje je malejąco według wieku i ogranicza wynik do trzech pierwszych rekordów.

Podsumowując:

- SELECT służy do pobierania danych z bazy
- WHERE filtruje wyniki wg warunku.
- ORDER BY sortuje wyniki (rosnąco domyślnie)
- LIMIT ogranicza liczbę zwróconych rekordów

Łączenie tabel (JOIN)

Często musimy pobrać dane z więcej niż jednej tabeli. Służą do tego różne rodzaje łączeń tabel (JOIN). Najczęściej używanym jest INNER JOIN, który łączy wiersze z dwóch (lub więcej) tabel, zwracając tylko te wiersze, dla których istnieje dopasowanie na podstawie wspólnego pola

Przykładowa składnia:

```
SELECT kolumna1, kolumna2, ...  
FROM tabela1  
INNER JOIN tabela2 ON tabela1.klucz = tabela2.klucz;
```

W naszym przykładzie, aby uzyskać listę zamówień wraz z danymi klienta i nazwy towaru, możemy połączyć wszystkie trzy tabele. Przykład zagnieżdżonego łączenia (łączenie wielu tabel):

```
SELECT k.imie, k.nazwisko, t.nazwa AS nazwa_towaru, z.ilosc, z.data  
FROM zamowienia z  
INNER JOIN klienci k ON z.id_klienta = k.id_klienta  
INNER JOIN towary t ON z.id_towaru = t.id_towaru;
```

```
1 SELECT k.imie, k.nazwisko, t.nazwa AS nazwa_towaru, z.ilosc, z.data
2 FROM zamowienia z
3 INNER JOIN klienci k ON z.id_klienta = k.id_klienta
4 INNER JOIN towary t ON z.id_towaru = t.id_towaru;
```

	imie	nazwisko	nazwa_towaru	ilosc	data
▶	Jan	Kowalski	Telewizor 55"	1	2024-01-15
	Anna	Nowak	Smartfon	2	2024-01-17
	Jan	Kowalski	Laptop	1	2024-02-03
	Piotr	Wiśniewski	Krzesło biurowe	4	2024-02-20
	Katarzyna	Wójcik	Regał na książki	2	2024-03-05
	Michał	Kamiński	Buty sportowe	1	2024-03-15
	Agnieszka	Lewandowska	Książka "SQL dla początkujących"	3	2024-03-17
	Tomasz	Zieliński	Słuchawki bezprzewodowe	1	2024-03-18
	Anna	Nowak	T-shirt męski	5	2024-03-20
	Jan	Kowalski	Sukienka damska	2	2024-04-01

Wynik tego zapytania będzie zawierał imię i nazwisko klienta, nazwę zamówionego towaru, ilość i datę dla każdego zamówienia.

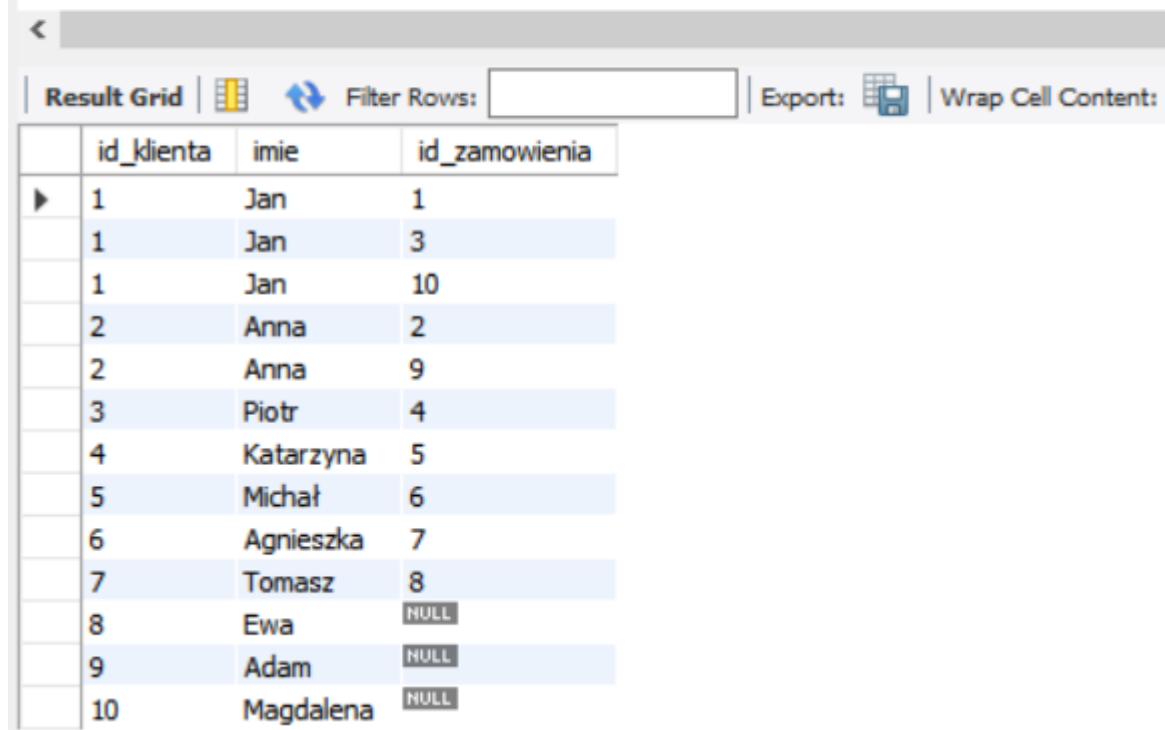
Oprócz INNER JOIN istnieją również inne typy łączy:

- LEFT JOIN: zwraca wszystkie wiersze z lewej tabeli oraz pasujące z prawej. Jeśli dla wiersza z lewej tabeli nie ma dopasowania, pola z prawej będą miały wartość NULL
- RIGHT JOIN: zwraca wszystkie wiersze z prawej tabeli oraz pasujące z lewej. Działa odwrotnie niż LEFT JOIN (przy braku dopasowania pola z lewej strony są NULL)
- FULL OUTER JOIN: teoretycznie zwraca wszystkie wiersze, które mają dopasowanie w lewej lub prawej tabeli. W praktyce MySQL nie obsługuje bezpośrednio FULL JOIN, ale można go zasymulować np. za pomocą UNION SELECT.

Przykład użycia LEFT JOIN w naszym sklepie: chcemy zobaczyć wszystkich klientów i ewentualne informacje o ich zamówieniach (nawet jeśli zamówienie nie istnieje).

```
SELECT k.id_klienta, k.imie, z.id_zamowienia
FROM klienci k
LEFT JOIN zamowienia z ON k.id_klienta = z.id_klienta;
```

```
1 • SELECT k.id_klienta, k.imie, z.id_zamowienia
2 FROM klienci k
3 LEFT JOIN zamowienia z ON k.id_klienta = z.id_klienta;
```

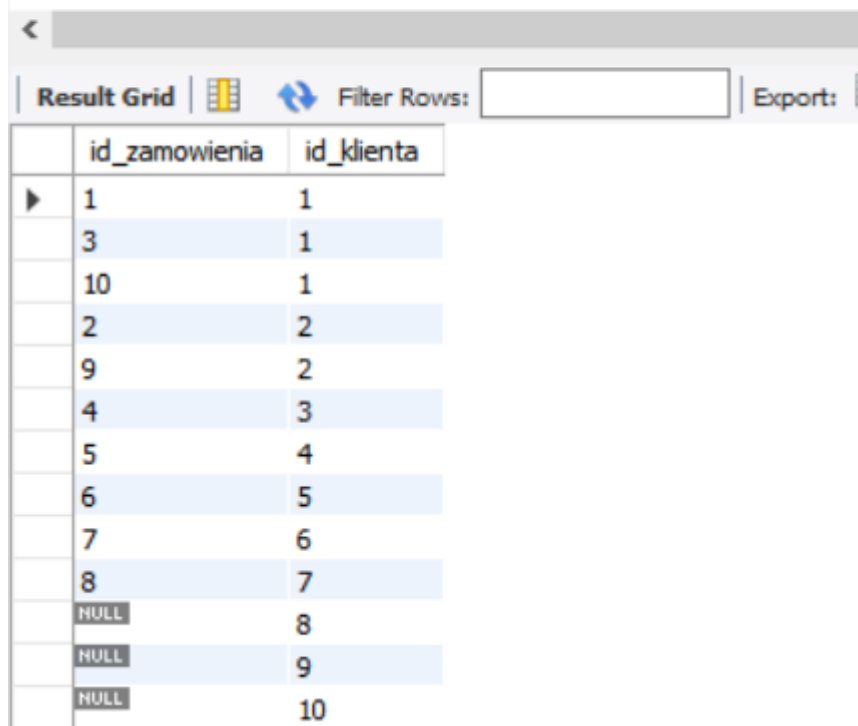


	id_klienta	imie	id_zamowienia
▶	1	Jan	1
	1	Jan	3
	1	Jan	10
	2	Anna	2
	2	Anna	9
	3	Piotr	4
	4	Katarzyna	5
	5	Michał	6
	6	Agnieszka	7
	7	Tomasz	8
	8	Ewa	NULL
	9	Adam	NULL
	10	Magdalena	NULL

W efekcie zobaczymy, że klienci bez zamówień (np. o id 8, 9, 10) będą mieli NULL w kolumnie id_zamowienia. RIGHT JOIN działa podobnie, tylko względem prawej tabeli:

```
SELECT z.id_zamowienia, k.id_klienta
FROM zamowienia z
RIGHT JOIN klienci k
ON z.id_klienta = k.id_klienta;
```

```
1 • SELECT z.id_zamowienia, k.id_klienta
2 FROM zamowienia z
3 RIGHT JOIN klienci k
4 ON z.id_klienta = k.id_klienta;
```



The screenshot shows a MySQL query result grid. The query is a RIGHT JOIN between the 'zamowienia' table (aliased as 'z') and the 'klienci' table (aliased as 'k') on the condition 'z.id_klienta = k.id_klienta'. The result grid has two columns: 'id_zamowienia' and 'id_klienta'. The data is as follows:

	id_zamowienia	id_klienta
▶	1	1
	3	1
	10	1
	2	2
	9	2
	4	3
	5	4
	6	5
	7	6
	8	7
	NULL	8
	NULL	9
	NULL	10

Podsumowując:

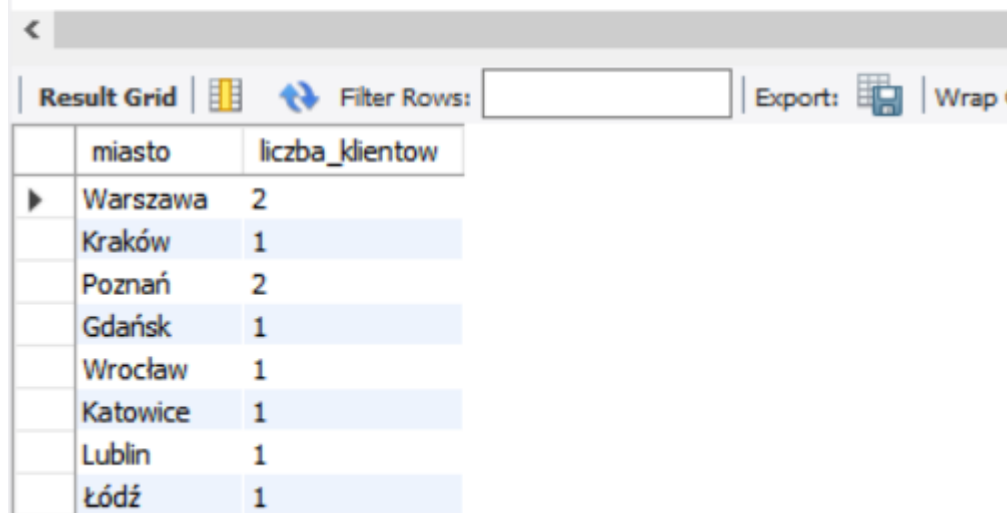
- INNER JOIN – łączy wiersze, gdy istnieje dopasowanie w obu tabelach
- LEFT JOIN – zwraca wszystkie z lewej tabeli i dopasowane z prawej
- RIGHT JOIN – zwraca wszystkie z prawej tabeli i dopasowane z lewej
- FULL OUTER JOIN – zwraca wszystkie z lewej lub prawej (NULL tam, gdzie brak dopasowania)

Grupowanie i funkcje agregujące

Do podsumowywania danych używamy klauzuli GROUP BY oraz funkcji agregujących. Klauzula GROUP BY grupuje wiersze o tych samych wartościach w określonych kolumnach (zwykle w połączeniu z funkcjami agregującymi jak SUM czy COUNT). Przykładowo, aby policzyć ilu jest klientów w każdym mieście:

```
SELECT miasto, COUNT(*) AS liczba_klientow
FROM klienci
GROUP BY miasto;
```

```
1 • SELECT miasto, COUNT(*) AS liczba_klientow
2 FROM klienci
3 GROUP BY miasto;
```



	miasto	liczba_klientow
▶	Warszawa	2
	Kraków	1
	Poznań	2
	Gdańsk	1
	Wrocław	1
	Katowice	1
	Lublin	1
	Łódź	1

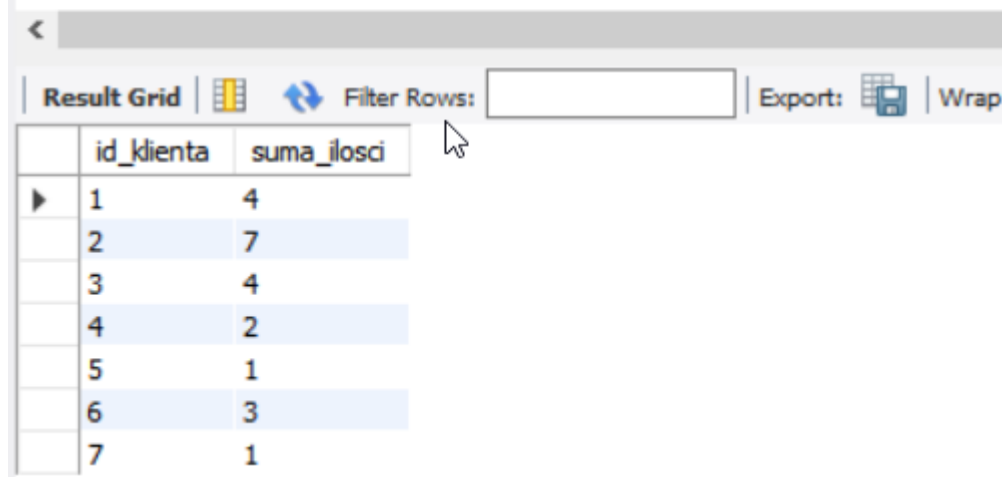
Wynik pokaże liczbę klientów z Warszawy, Krakowa, itp. Funkcje agregujące:

- COUNT() - zwraca liczbę wierszy
- SUM() - oblicza sumę wartości w kolumnie
- AVG() - oblicza średnią wartość kolumny
- (dalej: MIN(), MAX() itd.).

Na przykład, by obliczyć łączną ilość zamówionych towarów przez każdego klienta:

```
SELECT id_klienta, SUM(ilosc) AS suma_ilosci
FROM zamowienia
GROUP BY id_klienta;
```

```
1 • SELECT id_klienta, SUM(ilosc) AS suma_ilosci
2 FROM zamowienia
3 GROUP BY id_klienta;
```



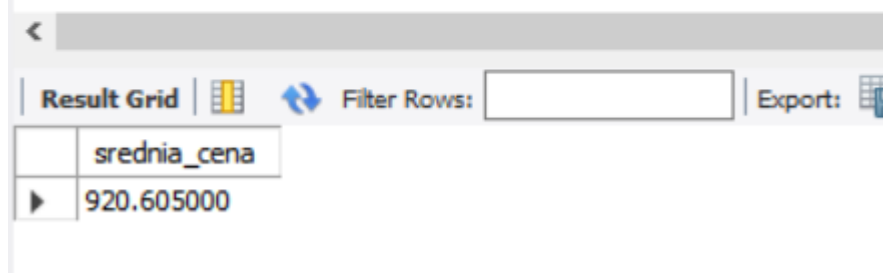
The screenshot shows a MySQL query result grid. The toolbar includes a back arrow, a 'Result Grid' button, a 'Filter Rows' input field, an 'Export' button, and a 'Wrap' button. The result table has two columns: 'id_klienta' and 'suma_ilosci'. It contains seven rows of data.

	id_klienta	suma_ilosci
▶	1	4
	2	7
	3	4
	4	2
	5	1
	6	3
	7	1

Innym przykładem jest zbadanie średniej ceny wszystkich produktów:

```
SELECT AVG(cena) AS srednia_cena
FROM towary;
```

```
1 • SELECT AVG(cena) AS srednia_cena
2 FROM towary;
```



The screenshot shows a MySQL query result grid. The toolbar includes a back arrow, a 'Result Grid' button, a 'Filter Rows' input field, and an 'Export' button. The result table has one column: 'srednia_cena'. It contains one row with the value 920.605000.

	srednia_cena
▶	920.605000

Funkcje agregujące ignorują wartości NULL, a gdy użyjemy ich bez klauzuli GROUP BY, traktowane są jak pojedyncza grupa obejmująca wszystkie wiersze.

INSERT, UPDATE, DELETE

Do modyfikacji danych w tabelach używamy trzech podstawowych poleceń:

- INSERT – służy do wstawiania nowych rekordów do tabeli
- UPDATE – służy do modyfikacji istniejących rekordów w tabeli
- DELETE – służy do usuwania istniejących rekordów z tabeli

Przykłady użycia:

```
-- Dodanie nowego klienta
```

```
INSERT INTO klienci (id_klienta, imie, nazwisko, miasto, wiek) VALUES  
(11, 'Karolina', 'Mazur', 'Gdynia', 29);
```

```
1 • INSERT INTO klienci (id_klienta, imie, nazwisko, miasto, wiek) VALUES (11, 'Karolina', 'Mazur', 'Gdynia', 29);  
2  
3 • SELECT * FROM klienci;
```

	id_klienta	imie	nazwisko	miasto	wiek
▶	1	Jan	Kowalski	Warszawa	34
	2	Anna	Nowak	Kraków	28
	3	Piotr	Wiśniewski	Poznań	45
	4	Katarzyna	Wójcik	Gdańsk	51
	5	Michał	Kamiński	Wrocław	39
	6	Agnieszka	Lewandowska	Katowice	23
	7	Tomasz	Zieliński	Warszawa	62
	8	Ewa	Szymańska	Lublin	31
	9	Adam	Dąbrowski	Łódź	27
	10	Magdalena	Jankowska	Poznań	44
	11	Karolina	Mazur	Gdynia	29
*	NULL	NULL	NULL	NULL	NULL

```
-- Zmiana miasta klienta o id 2
```

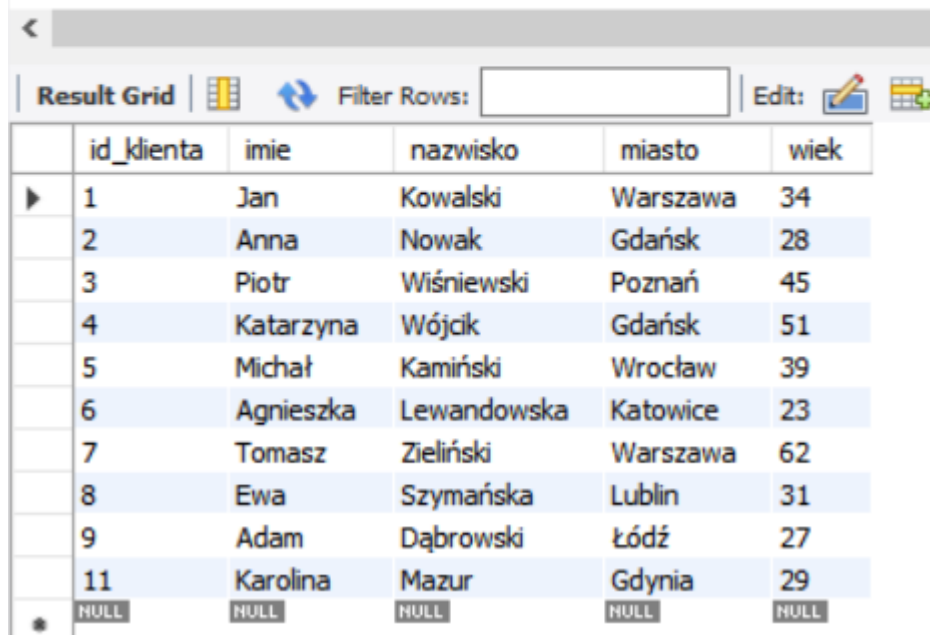
```
UPDATE klienci SET miasto = 'Gdańsk' WHERE id_klienta = 2;
```

```
1 • UPDATE klienci SET miasto = 'Gdańsk' WHERE id_klienta = 2;  
2  
3 • SELECT * FROM klienci;
```

	id_klienta	imie	nazwisko	miasto	wiek
▶	1	Jan	Kowalski	Warszawa	34
	2	Anna	Nowak	Gdańsk	28
	3	Piotr	Wiśniewski	Poznań	45
	4	Katarzyna	Wójcik	Gdańsk	51
	5	Michał	Kamiński	Wrocław	39
	6	Agnieszka	Lewandowska	Katowice	23
	7	Tomasz	Zieliński	Warszawa	62
	8	Ewa	Szymańska	Lublin	31
	9	Adam	Dąbrowski	Łódź	27
	10	Magdalena	Jankowska	Poznań	44
	11	Karolina	Mazur	Gdynia	29
*	NULL	NULL	NULL	NULL	NULL

```
-- Usunięcie klienta o id 10  
DELETE FROM klienci WHERE id_klienta = 10;
```

```
1 • DELETE FROM klienci WHERE id_klienta = 10;  
2  
3 • SELECT * FROM klienci;
```



The screenshot shows a database client interface with a 'Result Grid' tab selected. The grid displays the results of the query 'SELECT * FROM klienci;'. The columns are 'id_klienta', 'imie', 'nazwisko', 'miasto', and 'wiek'. The data is as follows:

	id_klienta	imie	nazwisko	miasto	wiek
▶	1	Jan	Kowalski	Warszawa	34
	2	Anna	Nowak	Gdańsk	28
	3	Piotr	Wiśniewski	Poznań	45
	4	Katarzyna	Wójcik	Gdańsk	51
	5	Michał	Kamiński	Wrocław	39
	6	Agnieszka	Lewandowska	Katowice	23
	7	Tomasz	Zieliński	Warszawa	62
	8	Ewa	Szymańska	Lublin	31
	9	Adam	Dąbrowski	Łódź	27
	11	Karolina	Mazur	Gdynia	29
•	NULL	NULL	NULL	NULL	NULL

Każde z tych zapytań ma swoje szczegóły: np. UPDATE i DELETE BEZ klauzuli WHERE zmodyfikują/usuną wszystkie wiersze tabeli, dlatego zawsze należy doprecyzować warunek.

Podzapytania (subqueries)

Podzapytanie to zapytanie zagnieżdżone wewnątrz innego zapytania. Innymi słowy, to zapytanie SELECT zawierające w klauzuli WHERE (lub FROM, HAVING itp.) inny SELECT. Podzapytania pozwalają porównywać wartość z wynikiem innego zapytania. Na przykład:

```
SELECT nazwa, cena  
FROM towary  
WHERE cena > (SELECT AVG(cena) FROM towary);
```

```
1 • SELECT nazwa, cena
2 FROM towary
3 WHERE cena > (SELECT AVG(cena) FROM towary);
4
5 • SELECT * FROM klienci;
```

< Result Grid | Filter Rows: | Edit:

	id_klienta	imie	nazwisko	miasto	wiek
▶	1	Jan	Kowalski	Warszawa	34
	2	Anna	Nowak	Gdańsk	28
	3	Piotr	Wiśniewski	Poznań	45
	4	Katarzyna	Wójcik	Gdańsk	51
	5	Michał	Kamiński	Wrocław	39
	6	Agnieszka	Lewandowska	Katowice	23
	7	Tomasz	Zieliński	Warszawa	62
	8	Ewa	Szymańska	Lublin	31
	9	Adam	Dąbrowski	Łódź	27
	11	Karolina	Mazur	Gdynia	29
•	NULL	NULL	NULL	NULL	NULL

To zapytanie zwraca produkty, których cena jest większa niż średnia cena wszystkich produktów. Najpierw wykonujemy podzapytanie (SELECT AVG(cena) FROM towary), które oblicza średnią cenę, a następnie główne zapytanie wybiera towary o cenie większej od tej wartości.

Podsumowując, podzapytanie jest zwykle zawarte w klauzuli WHERE innego SELECT i wykonuje się najpierw, dostarczając wartość do porównania

Prosta procedura składowana

Procedura składowana (stored procedure) to zapisany w bazie zestaw instrukcji SQL, który można wykonywać wielokrotnie. W MySQL tworzymy ją za pomocą polecenia CREATE PROCEDURE. Przykładowo, utwórzmy procedurę liczącą liczbę zamówień dla zadanego klienta:

```
DELIMITER //
CREATE PROCEDURE LiczbaZamowienDlaKlienta (IN client_id INT)
BEGIN SELECT COUNT(*) AS liczba_zamowien
FROM zamowienia
WHERE id_klienta = client_id;
END
// DELIMITER ;
```

Powyższa procedura LiczbaZamowienDlaKlienta przyjmuje parametr wejściowy client_id i zwraca liczbę zamówień danego klienta. Aby ją wywołać, używamy polecenia CALL:

```
CALL LiczbaZamowienDlaKlienta(1);
```

```
1 DELIMITER //
2 • CREATE PROCEDURE LiczbaZamowienDlaKlienta (IN client_id INT)
3   BEGIN SELECT COUNT(*) AS liczba_zamowien
4     FROM zamowienia
5     WHERE id_klienta = client_id;
6   END
7   // DELIMITER ;
8
9 • CALL LiczbaZamowienDlaKlienta(1);
```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
	liczba_zamowien			
▶	3			

Procedury składowane ułatwiają wielokrotne wykonywanie tych samych operacji na bazie i mogą zwracać zestawy wyników lub dane wyjściowe.

Podsumowanie

W tym tutorialu opisaliśmy podstawowe sposoby formułowania zapytań w MySQL na przykładzie bazy sklepu z tabelami klienci, zamowienia i towary. Omówiliśmy pobieranie danych przy pomocy SELECT z klauzulami WHERE, ORDER BY i LIMIT, łączenie tabel za pomocą różnych typów JOIN (INNER, LEFT, RIGHT, FULL) oraz zagnieżdżanie zapytań. Pokazaliśmy także, jak grupować dane używając GROUP BY oraz jak stosować funkcje agregujące takie jak COUNT, SUM i AVG do podsumowywania wyników. Przedstawiliśmy polecenia modyfikujące dane: INSERT (dodawanie nowych rekordów), UPDATE (aktualizacja istniejących) i DELETE (usuwanie rekordów). Zademonstrowaliśmy także przykład podzapytania (SELECT wewnątrz SELECT) oraz stworzenie prostej procedury składowanej.

Dzięki tym przykładowym zapytaniom możesz ćwiczyć pracę z bazą danych i stopniowo rozwijać swoje umiejętności SQL. Pamiętaj, że kluczem jest praktyka – warto eksperymentować na różnych danych i zadaniach, aby lepiej opanować składnię i możliwości MySQL.

Pełen dump bazy danych

```
-- Adminer 5.2.1 MySQL 8.0.42 dump

SET NAMES utf8;
SET time_zone = '+00:00';
SET foreign_key_checks = 0;
```

```
SET sql_mode = 'NO_AUTO_VALUE_ON_ZERO';

DROP DATABASE IF EXISTS `sklep`;
CREATE DATABASE `sklep` /*!40100 DEFAULT CHARACTER SET utf8mb3 COLLATE
utf8mb3_polish_ci */ /*!80016 DEFAULT ENCRYPTION='N' */;
USE `sklep`;

DELIMITER ;;

CREATE PROCEDURE `LiczbaZamowienDlaKlienta` (IN `client_id` int)
BEGIN SELECT COUNT(*) AS liczba_zamowien
FROM zamowienia
WHERE id_klienta = client_id;
END;;

DELIMITER ;

DROP TABLE IF EXISTS `klienci`;
CREATE TABLE `klienci` (
  `id_klienta` int NOT NULL,
  `imie` varchar(50) COLLATE utf8mb3_polish_ci DEFAULT NULL,
  `nazwisko` varchar(50) COLLATE utf8mb3_polish_ci DEFAULT NULL,
  `miasto` varchar(50) COLLATE utf8mb3_polish_ci DEFAULT NULL,
  `wiek` int DEFAULT NULL,
  PRIMARY KEY (`id_klienta`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3 COLLATE=utf8mb3_polish_ci;

INSERT INTO `klienci` (`id_klienta`, `imie`, `nazwisko`, `miasto`, `wiek`)
VALUES
(1, 'Jan', 'Kowalski', 'Warszawa', 34),
(2, 'Anna', 'Nowak', 'Gdańsk', 28),
(3, 'Piotr', 'Wiśniewski', 'Poznań', 45),
(4, 'Katarzyna', 'Wójcik', 'Gdańsk', 51),
(5, 'Michał', 'Kamiński', 'Wrocław', 39),
(6, 'Agnieszka', 'Lewandowska', 'Katowice', 23),
(7, 'Tomasz', 'Zieliński', 'Warszawa', 62),
(8, 'Ewa', 'Szymańska', 'Lublin', 31),
(9, 'Adam', 'Dąbrowski', 'Łódź', 27),
(11, 'Karolina', 'Mazur', 'Gdynia', 29);

DROP TABLE IF EXISTS `towary`;
CREATE TABLE `towary` (
  `id_towaru` int NOT NULL,
  `nazwa` varchar(100) COLLATE utf8mb3_polish_ci DEFAULT NULL,
  `kategoria` varchar(50) COLLATE utf8mb3_polish_ci DEFAULT NULL,
  `cena` decimal(10,2) DEFAULT NULL,
  PRIMARY KEY (`id_towaru`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3 COLLATE=utf8mb3_polish_ci;

INSERT INTO `towary` (`id_towaru`, `nazwa`, `kategoria`, `cena`) VALUES
(1, 'Telewizor 55"', 'Elektronika', 2499.99),
```

```
(2, 'Laptop', 'Elektronika', 3299.00),
(3, 'Smartfon', 'Elektronika', 1999.49),
(4, 'Regał na książki', 'Meble', 459.20),
(5, 'Krzesło biurowe', 'Meble', 349.00),
(6, 'T-shirt męski', 'Odzież', 59.99),
(7, 'Sukienka damska', 'Odzież', 129.50),
(8, 'Buty sportowe', 'Obuwie', 179.99),
(9, 'Słuchawki bezprzewodowe', 'Elektronika', 149.99),
(10, 'Książka "SQL dla początkujących"', 'Książki', 79.90);

DROP TABLE IF EXISTS `zamowienia`;
CREATE TABLE `zamowienia` (
  `id_zamowienia` int NOT NULL,
  `id_klienta` int DEFAULT NULL,
  `id_towaru` int DEFAULT NULL,
  `ilosc` int DEFAULT NULL,
  `DATA` date DEFAULT NULL,
  PRIMARY KEY (`id_zamowienia`),
  KEY `fk_zamowienia_klienci` (`id_klienta`),
  KEY `fk_zamowienia_towary` (`id_towaru`),
  CONSTRAINT `fk_zamowienia_klienci` FOREIGN KEY (`id_klienta`) REFERENCES
`klienci` (`id_klienta`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `fk_zamowienia_towary` FOREIGN KEY (`id_towaru`) REFERENCES
`towary` (`id_towaru`) ON DELETE RESTRICT ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3 COLLATE=utf8mb3_polish_ci;

INSERT INTO `zamowienia` (`id_zamowienia`, `id_klienta`, `id_towaru`,
`ilosc`, `DATA`) VALUES
(1, 1, 1, 1, '2024-01-15'),
(2, 2, 3, 2, '2024-01-17'),
(3, 1, 2, 1, '2024-02-03'),
(4, 3, 5, 4, '2024-02-20'),
(5, 4, 4, 2, '2024-03-05'),
(6, 5, 8, 1, '2024-03-15'),
(7, 6, 10, 3, '2024-03-17'),
(8, 7, 9, 1, '2024-03-18'),
(9, 2, 6, 5, '2024-03-20'),
(10, 1, 7, 2, '2024-04-01');

-- 2025-05-13 10:05:07 UTC
```

From:

<https://wiki.ostrowski.net.pl/> - Kacper's Wiki

Permanent link:

https://wiki.ostrowski.net.pl/doku.php?id=notatki:bazy_danych&rev=1747145635

Last update: **2025/05/13 16:13**