MySQL: Introduction database design Shop

You can follow the tutorial via the website: https://wiki.ostrowski.net.pl/php_mysql/sklep.php

In this article, we will use a sample online shop database with three tables: customers, goods and orders. We will present the definitions of the tables in SQL and add 10 sample records to each table. We will then discuss the different types of queries on this data, including:

- basic SELECT queries with WHERE, ORDER BY and LIMIT clauses,
- different types of JOIN (INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN) and nested joins (joins of multiple tables),
- queries using GROUP BY and aggregate functions (COUNT, SUM, AVG, etc.),
- modifying INSERT, UPDATE and DELETE queries,
- an example of a subquery,
- a simple stored procedure.

Each type of query will be illustrated with an example SQL code running on our database. We will conclude the article with a brief summary of the issues discussed.

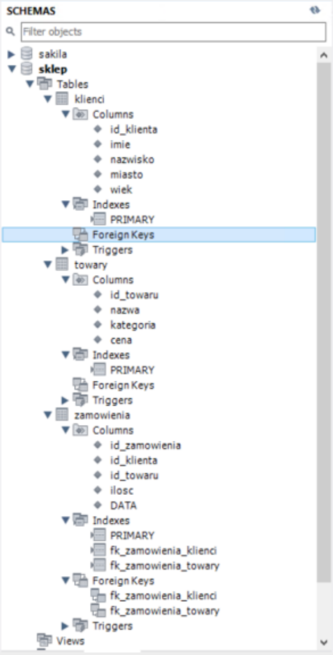
Tables and sample data

Let's start by defining the structure of the shop database. We will create three tables:

```
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)
```



After executing the script we should get something like this

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

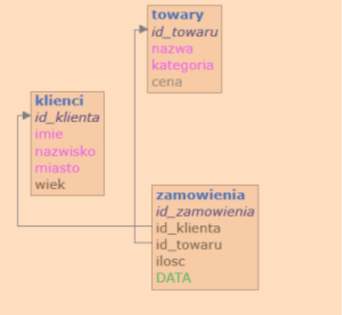
Explanations:

PRIMARY KEY next to the columns id_customer, id_goods and id_order uniquely identifies a row in each table.

In the orders table:

- fk_orders_customers is a foreign key pointing to customers(id_customer).

ON UPDATE CASCADE - changing the value of id_client in the customers table will automatically update all related rows in the orders.



Database scheme generated by the tool Adminer

ON DELETE CASCADE - deleting a customer will automatically delete their orders.

- fk_orders_goods is a foreign key indicating goods(id_goods).

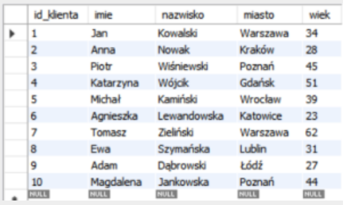
ON UPDATE CASCADE - changing the id_goods into goods will automatically update the references in the orders.

ON DELETE RESTRICT - will not allow deletion of goods that are associated with at least one order.

In the customers table we store customer data (first name, last name, city, age), in the goods table we store product information (name, category, price), and in the orders table we store order data (customer and goods relationship, quantity, date).

We will then add sample records to each table:

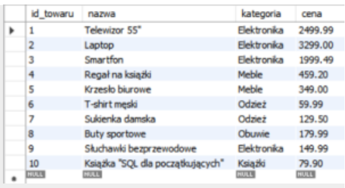
```
INSERT INTO klienci
(id_klienta, imie, nazwisko, miasto, wiek)
VALUES
(1,'Jan','Kowalski','Warszawa',34),
(2,'Anna','Nowak','Kraków',28),
(3,'Piotr','Wiśniewski','Poznań',45),
(4,'Katarzyna','Wójcik','Gdańsk',51),
(5,'Michał','Kamiński','Wrocław',39),
(6,'Agnieszka','Lewandowska','Katowice',23),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



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

When done, we should get something like this

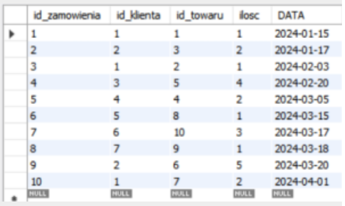
```
INSERT INTO towary
(id_towaru, nazwa, kategoria, cena)
VALUES
(1,'Telewizor 55\"','Elektronika',2499.99),
(2,'Laptop','Elektronika',3299.00),
(3,'Smartfon','Elektronika',1999.49),
(4,'Regał na książki','Meble',459.20),
(5,'Krzesło biurowe','Meble',349.00),
(6,'T-shirt męski','Odzież',59.99),
(7,'Sukienka 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
*	NULL	NULL	NULL	NULL

Once executed, we should get something like this

```
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
*	NULL	NULL	NULL	NULL	NULL

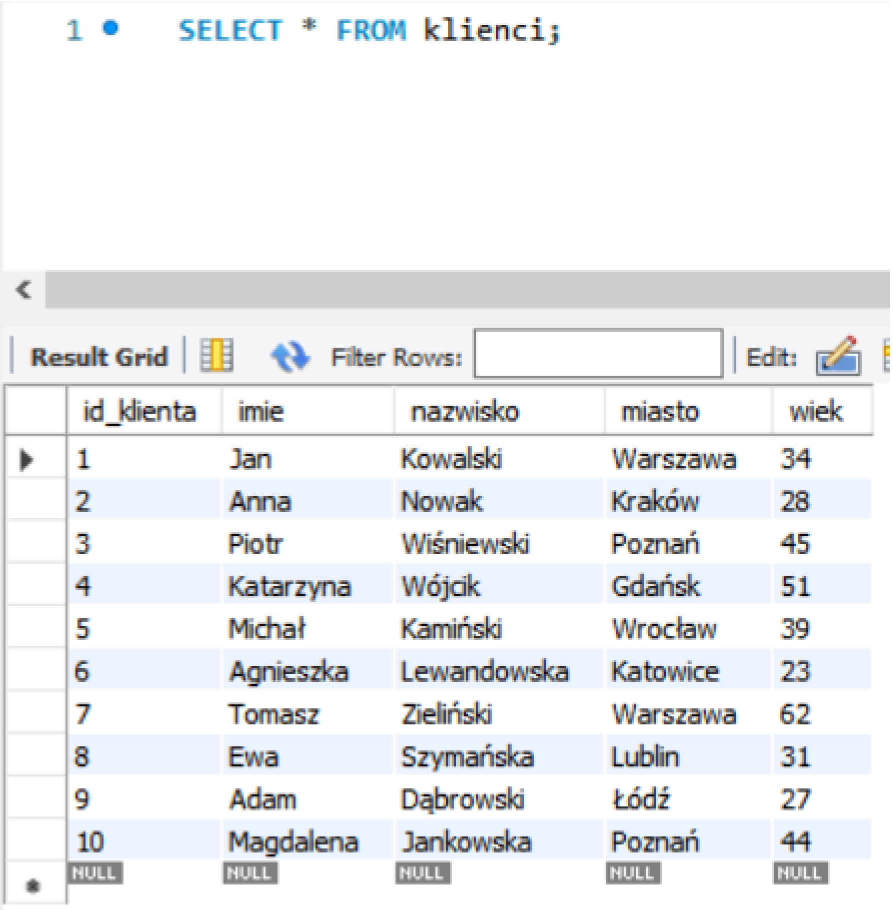
Once executed, we should get something like this

This configuration provides realistic data: several customers order different goods in different quantities and at different times.

Basic SELECT queries

The basic query used to retrieve data is SELECT. It allows you to select one or more columns from a table. The simplest SELECT query returns all the columns and rows of a given table. For example:

```
SELECT * FROM klienci;
```

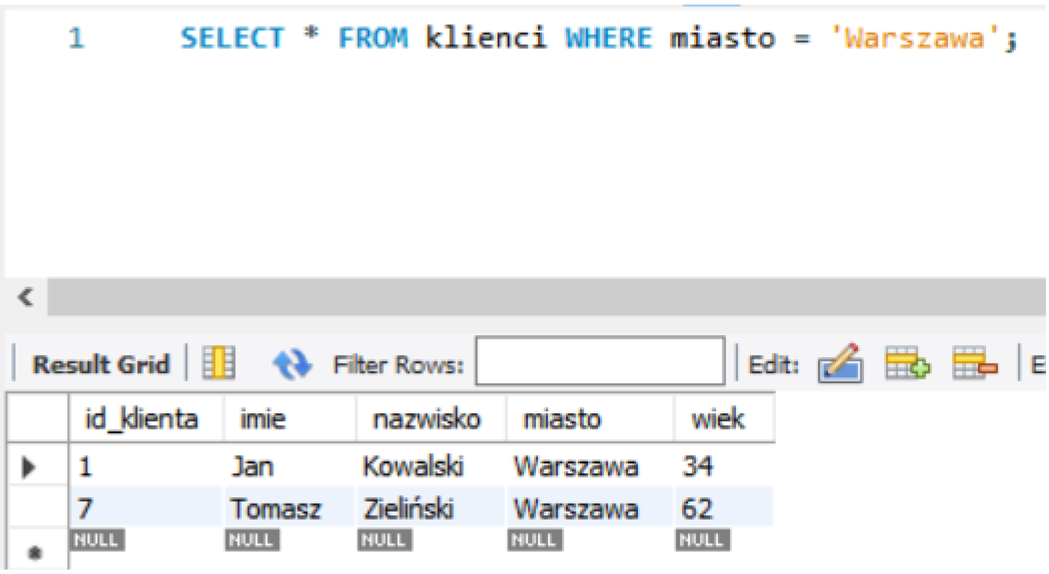


```
SELECT * FROM klienci;
```

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

this query retrieves all data from the customers table. However, we often need to filter the records according to certain criteria. The WHERE clause is used for this, e.g. to select customers from a specific city:

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



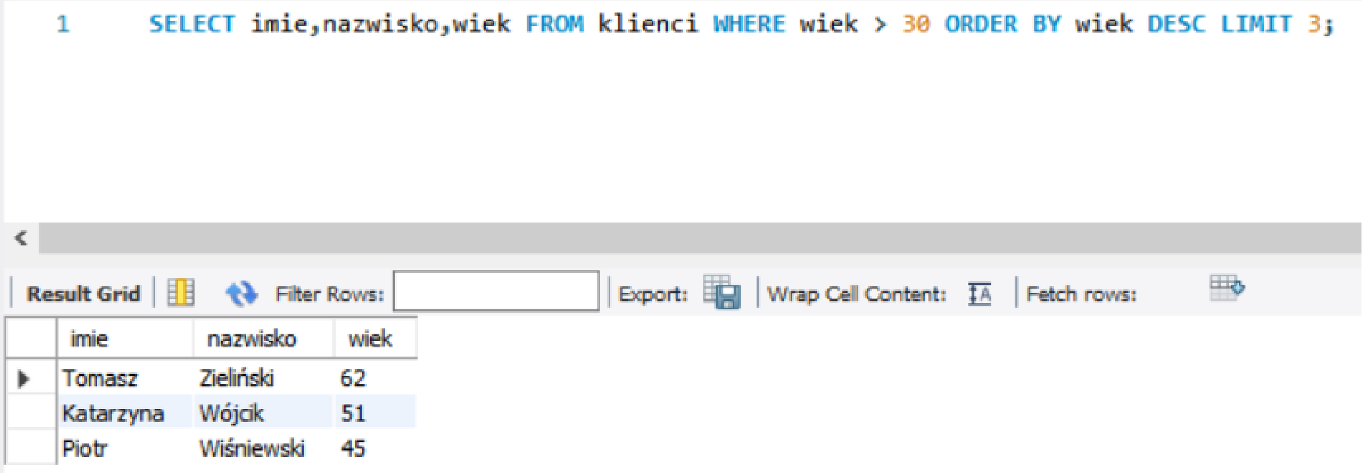
```
SELECT * FROM klienci WHERE miasto = 'Warszawa';
```

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

We can also limit the number of results and their order. The ORDER BY clause sorts the results against the given columns (ascending by default), while LIMIT limits the number of rows returned

Examples:

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



```
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

The above query selects the names, surnames and ages of clients older than 30, sorts them descending by age and limits the result to the first three records.

To summarise:

- SELECT is used to retrieve data from a database
- WHERE filters the results according to the condition.
- ORDER BY sorts the results (ascending by default).
- LIMIT limits the number of records returned

Joining tables (JOIN)

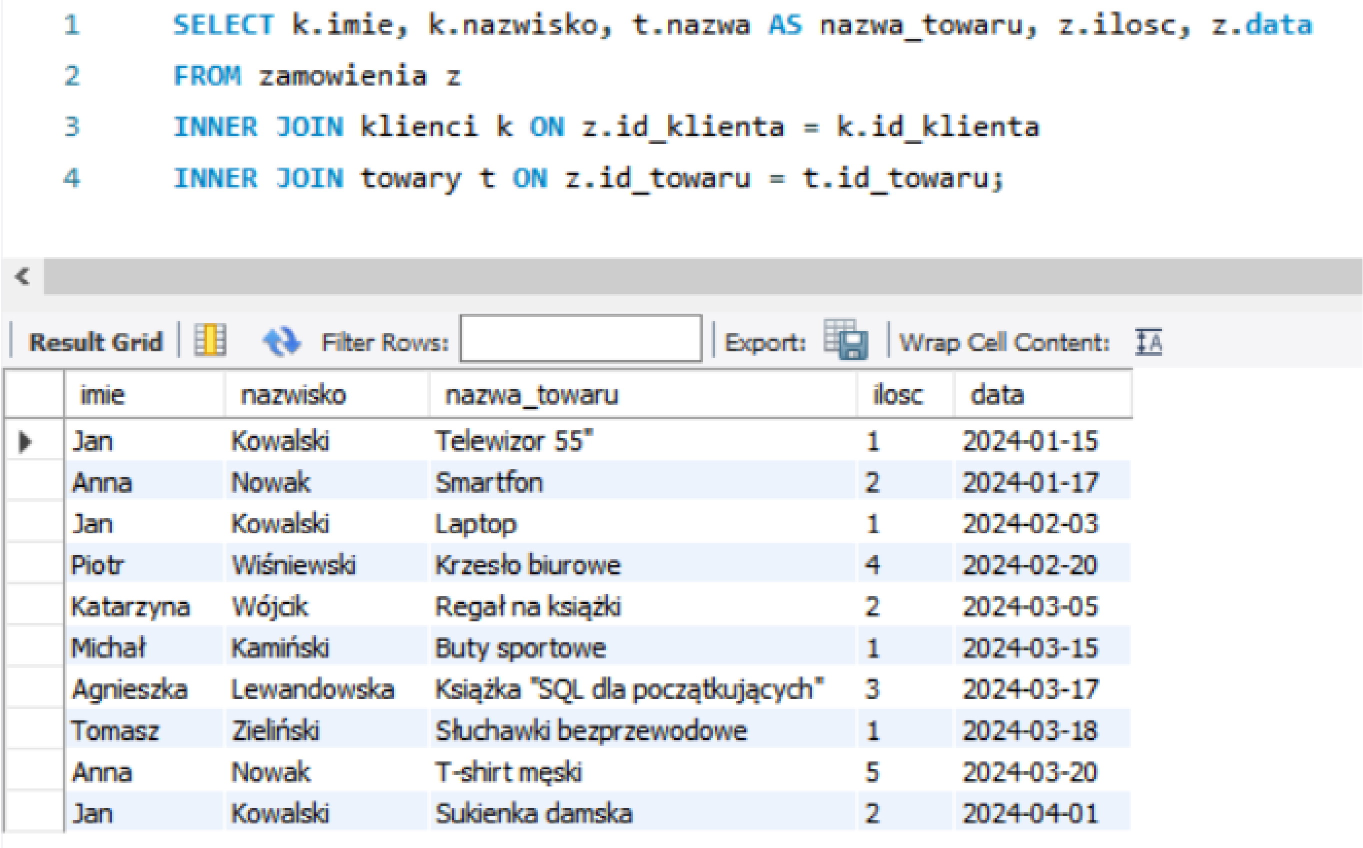
We often need to retrieve data from more than one table. Different types of table joins (JOIN) are used to do this. The most commonly used is INNER JOIN, which combines rows from two (or more) tables, returning only those rows for which there is a match based on a common field

Example syntax:

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

In our example, in order to get a list of orders including the customer data and the commodity name, we can join all three tables. Example of a nested join (joining multiple tables):

```
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;
```



```
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;
```

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

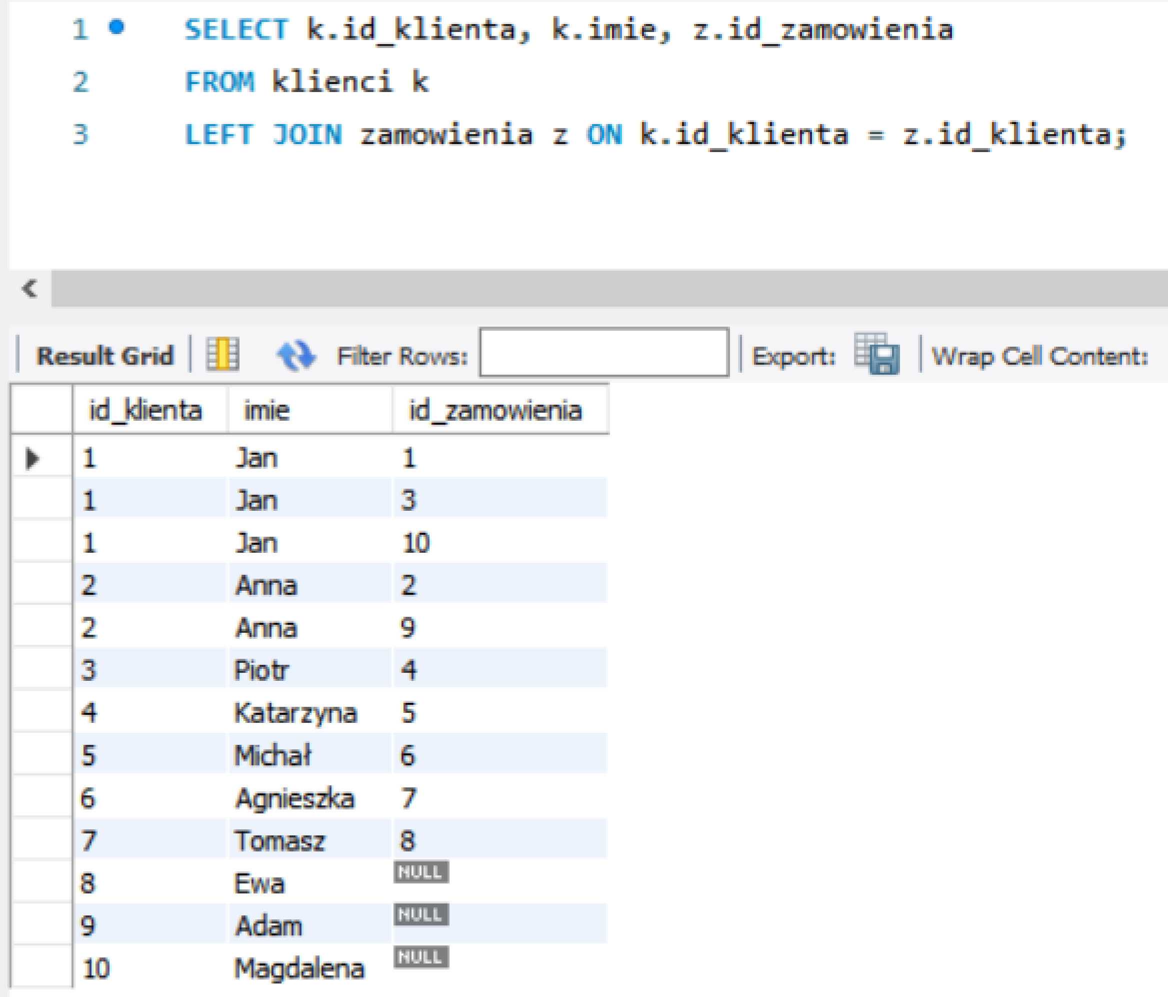
The result of this query will include the customer's name, the name of the goods ordered, the quantity and the date for each order.

In addition to INNER JOIN, there are other types of joins:

- LEFT JOIN: returns all rows from the left table and a match from the right table. If there is no match for a row in the left table, the fields on the right will be NULL
- RIGHT JOIN: returns all rows from the right table and matching rows from the left table. Works the opposite of LEFT JOIN (if there is no match, the fields on the left are NULL)
- FULL OUTER JOIN: theoretically returns all rows that have a match in the left or right table. In practice, MySQL does not support FULL JOIN directly, but it can be simulated with UNION SELECT, for example.

Example of use `LEFT JOIN` in our shop: we want to see all customers and possible information about their orders (even if the order does not exist).

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

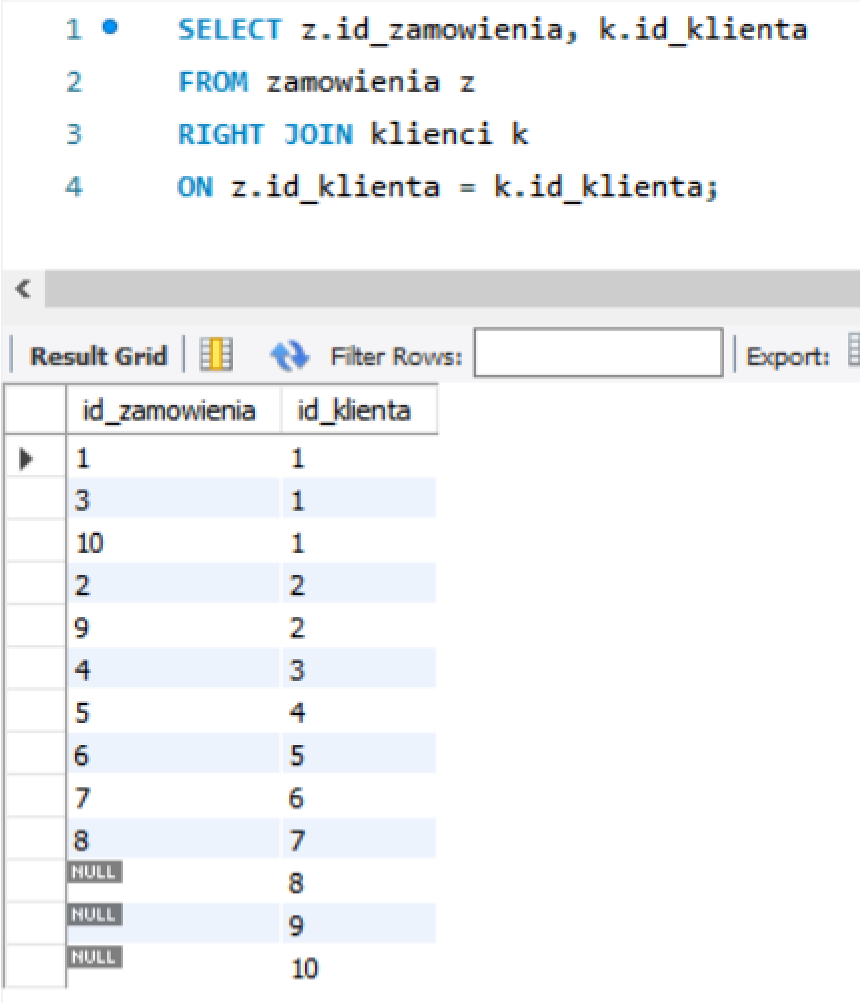


```sql
SELECT k.id_klienta, k.imie, z.id_zamowienia
FROM klienci k
LEFT JOIN zamowienia z ON k.id_klienta = z.id_klienta;
```

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

As a result, we will see that customers without orders (e.g. with id 8, 9, 10) will have NULL in the id_orders column. RIGHT JOIN works similarly, only relative to the right-hand table:

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



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

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

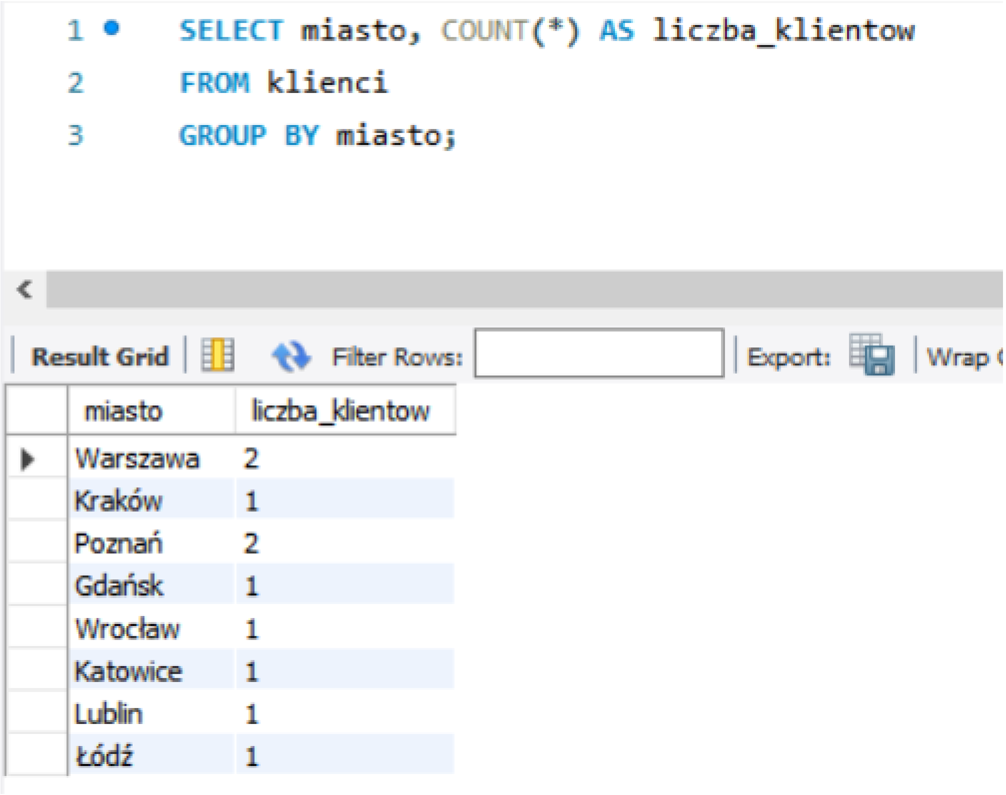
To summarise:

- INNER JOIN - merges rows when there is a match in both tables.
- LEFT JOIN - returns all from the left table and matches from the right table
- RIGHT JOIN - returns all from the right table and matched from the left table
- FULL OUTER JOIN - returns all from the left or right (NULL where there is no match)

Grouping and aggregation functions

We use the GROUP BY clause and aggregate functions to summarise the data. The GROUP BY clause groups rows with the same values in specific columns (usually in combination with aggregating functions like SUM or COUNT). For example, to count how many customers there are in each city:

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



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

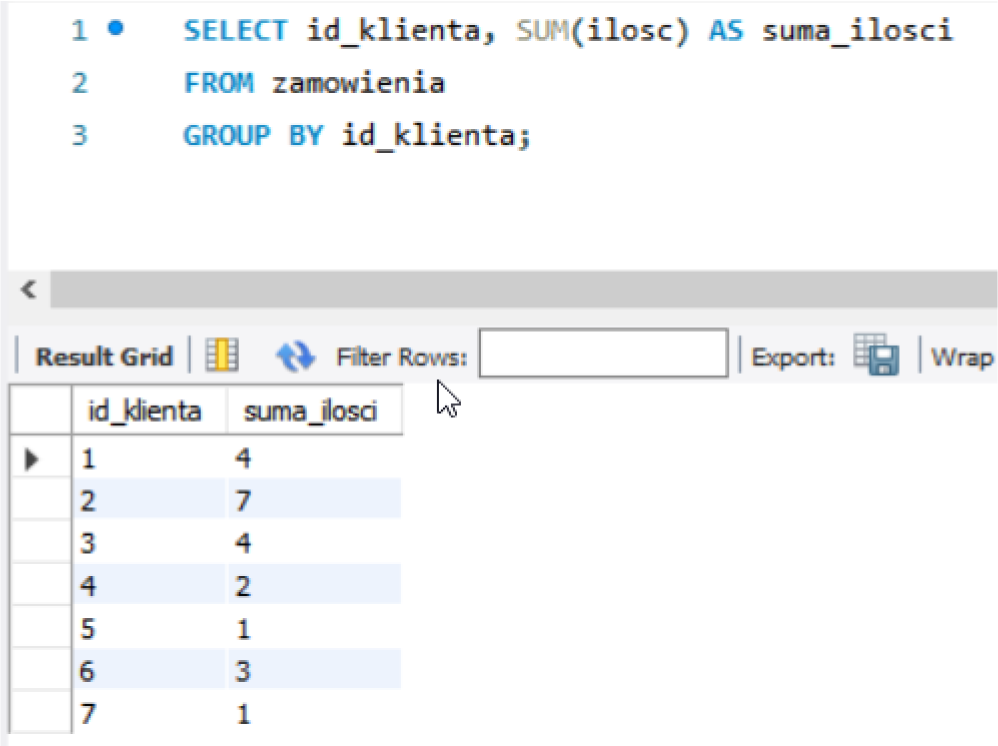
miasto	liczba_klientow
Warszawa	2
Kraków	1
Poznań	2
Gdańsk	1
Wrocław	1
Katowice	1
Lublin	1
Łódź	1

The result will show the number of customers from Warsaw, Krakow, etc. Aggregating functions:

- COUNT() - returns the number of rows
- SUM() - calculates the sum of values in a column
- AVG() - calculates the average value of a column
- (further: MIN(), MAX(), etc.).

For example, to calculate the total amount of goods ordered by each customer:

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

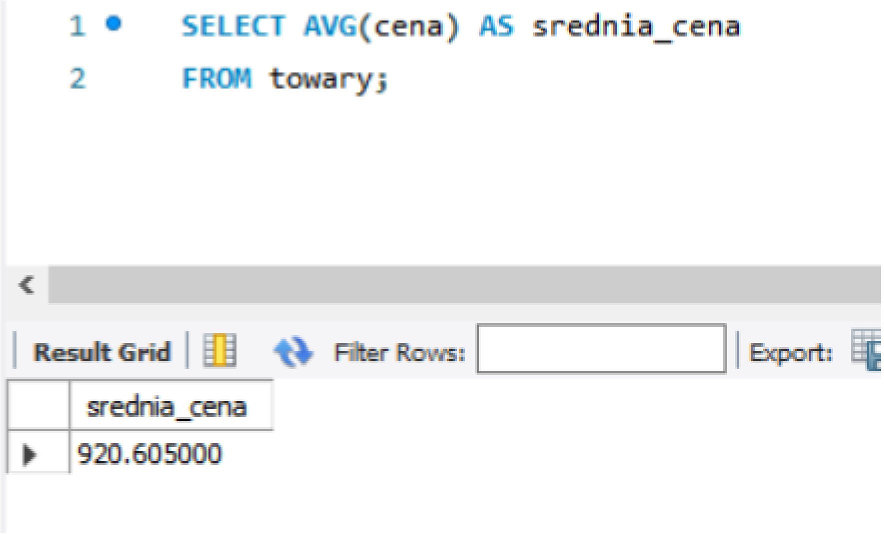


```
SELECT id_klienta, SUM(ilosc) AS suma_ilosci
FROM zamowienia
GROUP BY id_klienta;
```

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

Another example is to examine the average price of all products:

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



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

srednia_cena
920.605000

Aggregating functions ignore NULL values, and when used without a GROUP BY clause, they are treated as a single group including all rows.

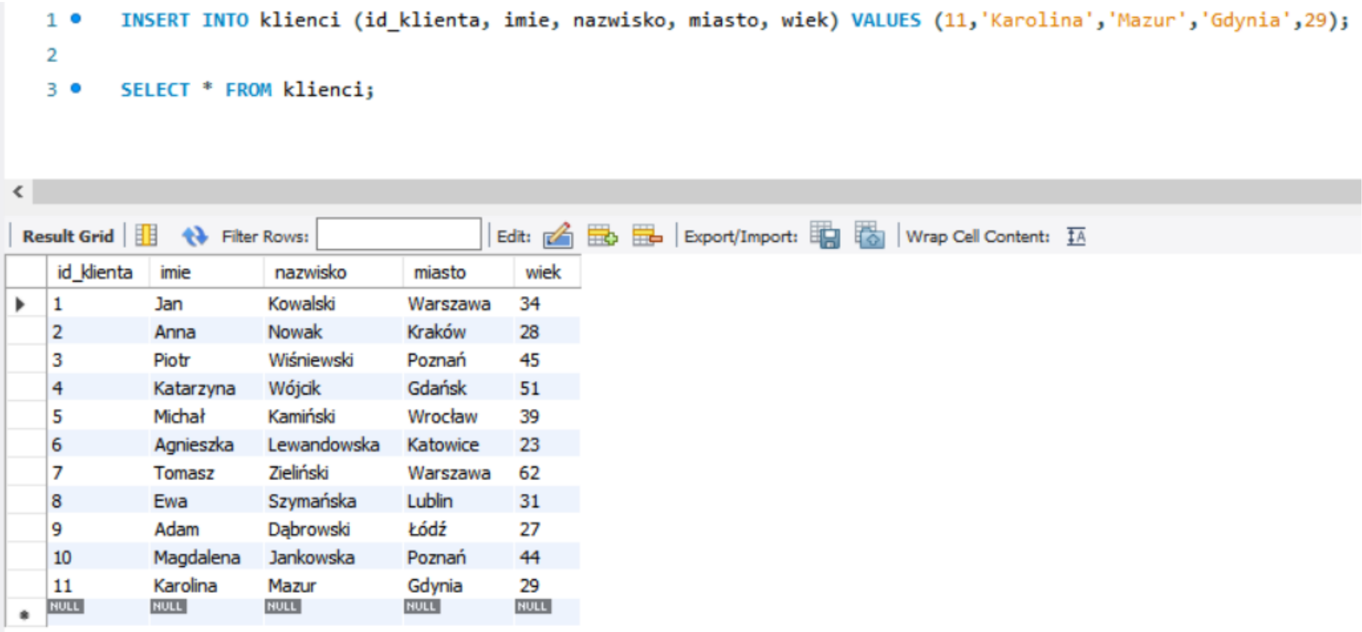
INSERT, UPDATE, DELETE

We use three basic commands to modify data in tables:

- INSERT - used to insert new records into a table.
- UPDATE - to modify existing records in the table
- DELETE - to delete existing records from the table

Examples of use:

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

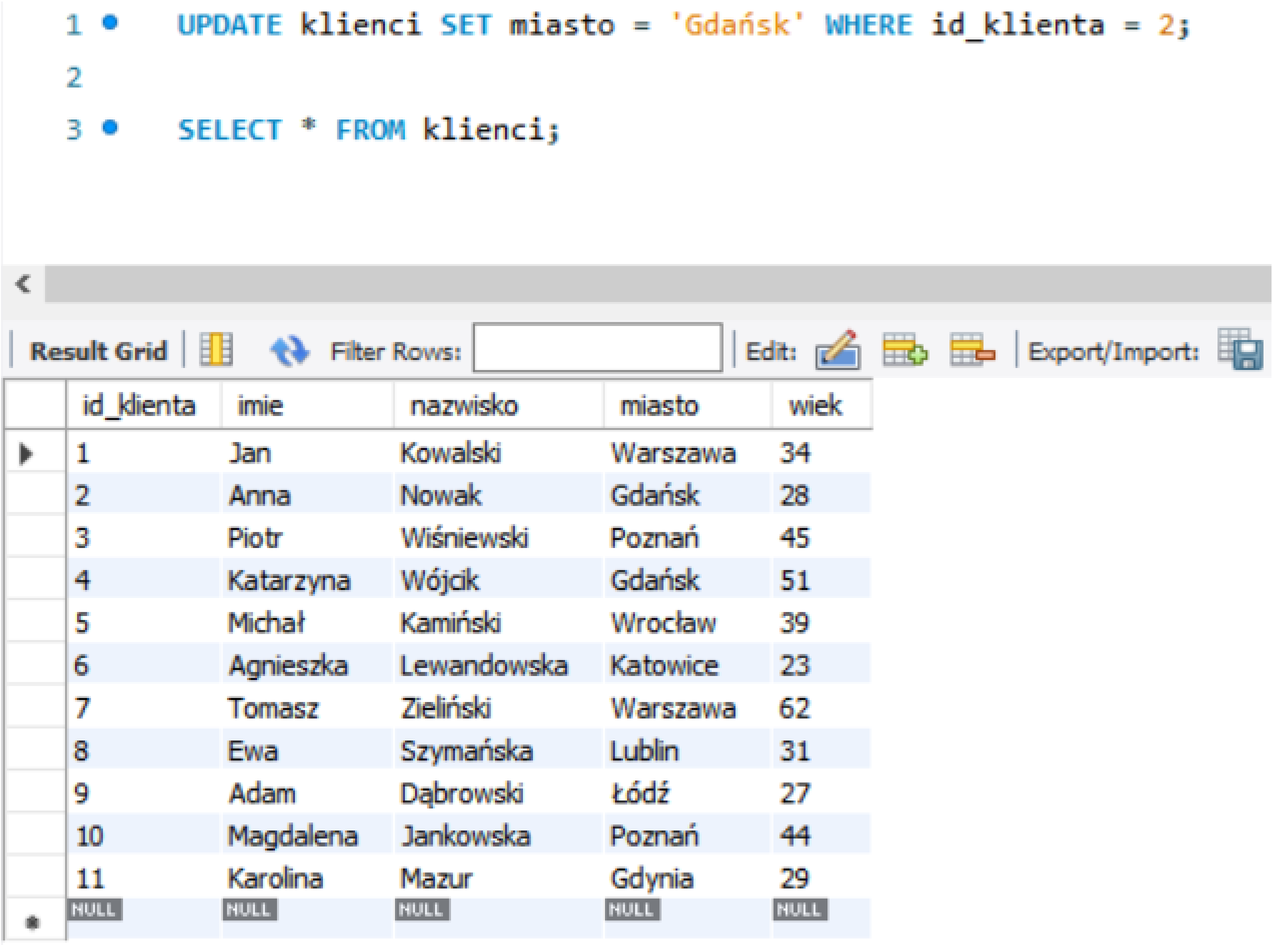


```
INSERT INTO klienci (id_klienta, imie, nazwisko, miasto, wiek) VALUES (11,'Karolina','Mazur','Gdynia',29);

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;
```

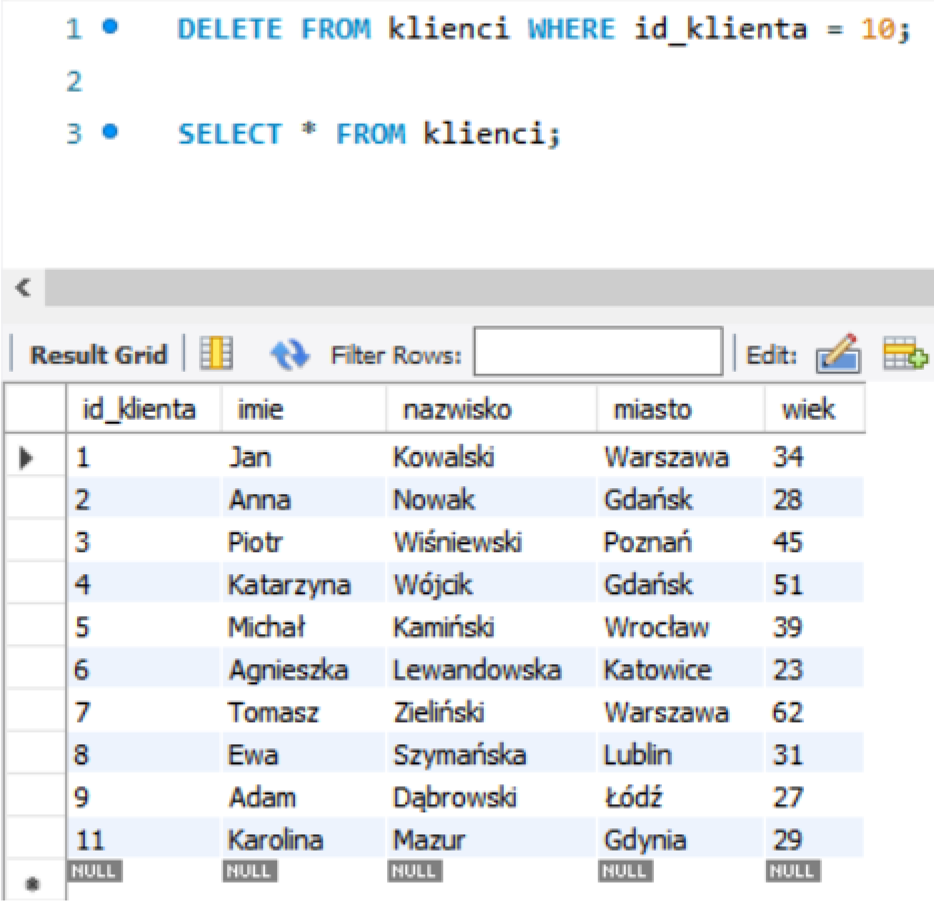


```
UPDATE klienci SET miasto = 'Gdańsk' WHERE id_klienta = 2;

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;
```



```
DELETE FROM klienci WHERE id_klienta = 10;

SELECT * FROM klienci;
```

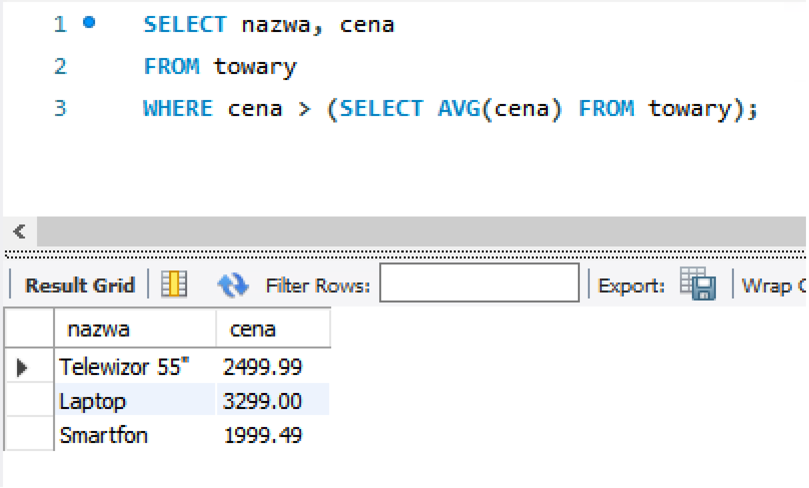
id_klienta	imie	nazwisko	miasto	wiek
1	Jan	Kowalski	Warszawa	34
2	Anna	Nowak	Gdańsk	28
3	Piotr	Wiśniewski	Poznań	45
4	Katarzyna	Wójcik	Gdańsk	51
5	Michał	Kamiński	Wrocław	39
6	Agnieszka	Lewandowska	Katowice	23
7	Tomasz	Zieliński	Warszawa	62
8	Ewa	Szymańska	Lublin	31
9	Adam	Dąbrowski	Łódź	27
11	Karolina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

Each of these queries has its own details: e.g. UPDATE and DELETE WITHOUT a WHERE clause will modify/delete all rows in the table, so always specify the condition.

Subqueries

A subquery is a query nested inside another query. In other words, it is a SELECT query that contains another SELECT in the WHERE clause (or FROM, HAVING, etc.). Subqueries allow you to compare a value with the result of another query. For example:

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



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

nazwa	cena
Telewizor 55"	2499.99
Laptop	3299.00
Smartfon	1999.49

This query returns products whose price is greater than the average price of all products. We first execute a sub-query (SELECT AVG(price) FROM goods), which calculates the average price, and then the main query selects the goods with a price greater than this value.

In summary, the sub-query is usually included in the WHERE clause of another SELECT and executes first, providing a value for comparison

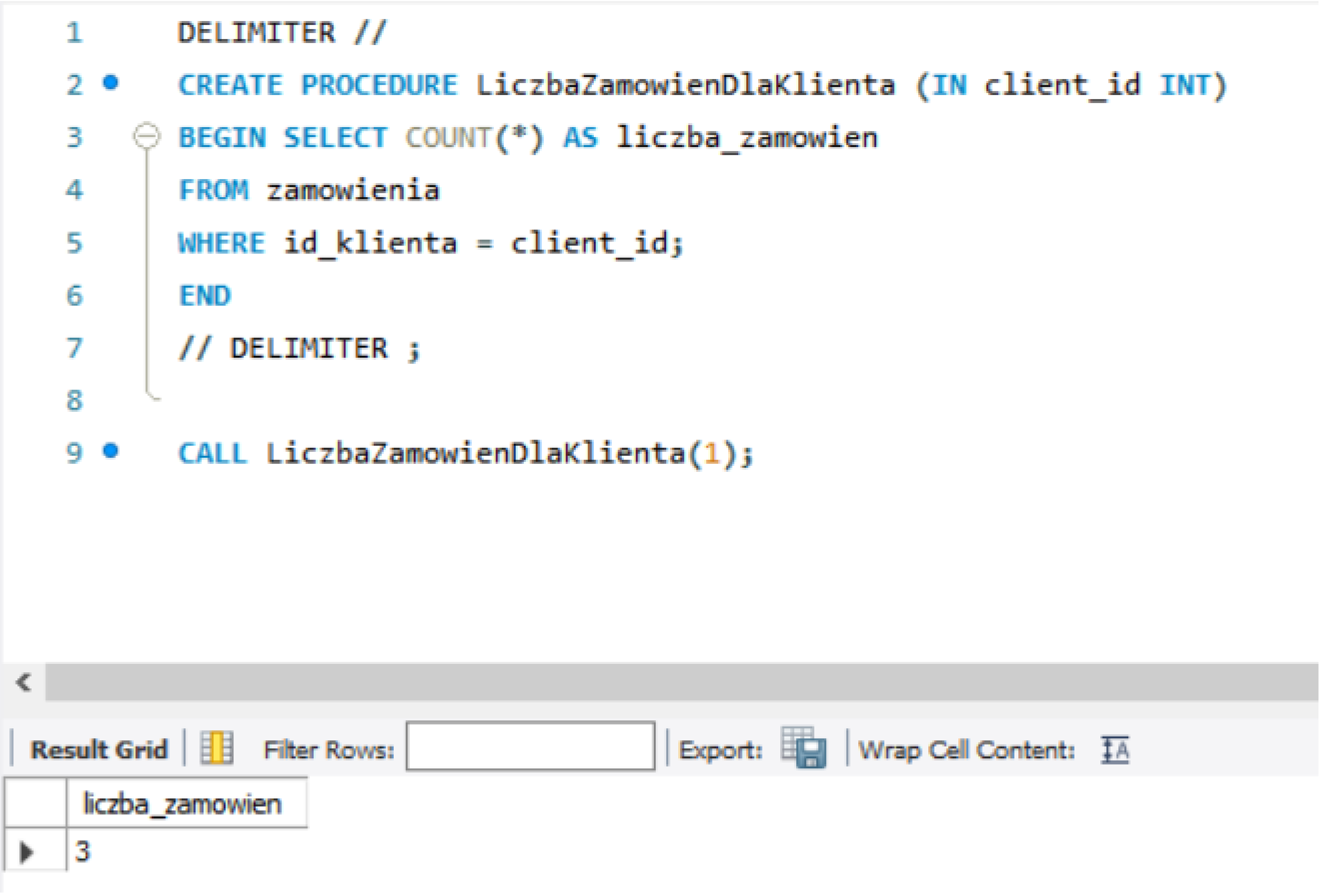
A simple stored procedure

A stored procedure (stored procedure) is a set of SQL statements stored in the database that can be executed multiple times. In MySQL, you create it with the CREATE PROCEDURE statement. For example, let's create a procedure to count the number of orders for a given customer:

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

The above procedure NumberOrdersCustomer takes an input parameter client_id and returns the number of orders of a given customer. To call it, we use the CALL command:

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



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

CALL LiczbaZamowienDlaKlienta(1);
```

liczba_zamowien
3

Stored procedures facilitate repeated execution of the same operations on the database and can return result sets or output.

Summary

In this tutorial, we have described the basic ways to formulate queries in MySQL using the example of a shop database with tables customers, orders and goods. We discussed retrieving data using SELECT with WHERE, ORDER BY and LIMIT clauses, joining tables using different JOIN types (INNER, LEFT, RIGHT, FULL) and nesting queries. We also showed how to group data using GROUP BY and how to use aggregating functions such as COUNT, SUM and AVG to summarise results. We presented the commands that modify data: INSERT (adding new records), UPDATE (updating existing records) and DELETE (deleting records). We also demonstrated an example of a sub-query (SELECT inside SELECT) and the creation of a simple stored procedure.

With these example queries, you can practice working with the database and gradually develop your SQL skills. Remember that practice is the key - it is worth experimenting with different data and tasks to better master MySQL syntax and capabilities.

Full database dump

```
-- Adminer 5.2.1 MySQL 8.0.42 dump

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

```sql
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
```