

# Lekcja 11

## Temat: Zaawansowane zapytania JOIN

```
DROP TABLE IF EXISTS zamowienia;
DROP TABLE IF EXISTS klienci;

CREATE TABLE klienci (
    id INT PRIMARY KEY,
    imie VARCHAR(50)
);

CREATE TABLE zamowienia (
    id INT PRIMARY KEY,
    id_klienta INT,
    produkt VARCHAR(50),
    FOREIGN KEY (id_klienta) REFERENCES klienci(id)
);

INSERT INTO klienci (id, imie) VALUES
(1, 'Anna'),
(2, 'Jan'),
(3, 'Ola'),
(4, 'Piotr');

INSERT INTO zamowienia (id, id_klienta, produkt) VALUES
(1, 1, 'Laptop'),
(2, 1, 'Myszka'),
(3, 2, 'Telefon'),
(4, null, 'Monitor'); -- ten klient (id=5) nie istnieje w tabeli klienci
```

### ● INNER JOIN

```
SELECT k.imie, z.produkt
FROM klienci k
INNER JOIN zamowienia z ON k.id = z.id_klienta;
```

imie	produkt
Anna	Laptop
Anna	Myszka
Jan	Telefon

### ● LEFT JOIN

```
SELECT k.imie, z.produkt  
FROM klienci k  
LEFT JOIN zamówienia z ON k.id = z.id_klienta;
```

imie	produkt
Anna	Laptop
Anna	Myszka
Jan	Telefon
Ola	NULL
Piotr	NULL

## ● RIGHT JOIN

```
SELECT k.imie, z.produkt  
FROM klienci k  
RIGHT JOIN zamówienia z ON k.id = z.id_klienta;
```

imie	produkt
Anna	Laptop
Anna	Myszka
Jan	Telefon
NULL	Monitor

## ● FULL JOIN (symulowany)

```
SELECT k.imie, z.produkt  
FROM klienci k  
LEFT JOIN zamówienia z ON k.id = z.id_klienta
```

```
UNION
```

```
SELECT k.imie, z.produkt  
FROM klienci k  
RIGHT JOIN zamówienia z ON k.id = z.id_klienta;
```

imie	produkt
Anna	Laptop

Anna	Myszka
Jan	Telefon
Ola	NULL
Piotr	NULL
NULL	Monitor

```

DROP TABLE IF EXISTS zamowienia;
DROP TABLE IF EXISTS produkty;
DROP TABLE IF EXISTS sklepy;

CREATE TABLE sklepy (
    id INT PRIMARY KEY,
    nazwa VARCHAR(50)
);

CREATE TABLE produkty (
    id INT PRIMARY KEY,
    nazwa VARCHAR(50),
    id_sklepu INT,
    FOREIGN KEY (id_sklepu) REFERENCES sklepy(id)
);

CREATE TABLE zamowienia (
    id INT PRIMARY KEY,
    id_produktu INT ,
    ilosc INT,
    FOREIGN KEY (id_produktu) REFERENCES produkty(id)
);

INSERT INTO sklepy (id, nazwa) VALUES
(1, 'Sklep A'),
(2, 'Sklep B'),
(3, 'Sklep C');

INSERT INTO produkty (id, nazwa, id_sklepu) VALUES
(1, 'Laptop', 1),
(2, 'Myszka', 1),
(3, 'Monitor', 2),
(4, 'Klawiatura', 3);

INSERT INTO zamowienia (id, id_produktu, ilosc) VALUES
(1, 1, 5),
(2, 1, 3),
(3, 2, 10),
(4, 3, 2);

```

**Zestawienie sklepów i produktów, łącznie z tymi, dla których nie odnotowano zamówień:**

```

SELECT s.nazwa AS sklep,
       p.nazwa AS produkt,
       COALESCE(SUM(z.ilosc), 0) AS sprzedane_sztuki
  FROM sklepy s
 LEFT JOIN produkty p ON p.id_sklepu = s.id
 LEFT JOIN zamowienia z ON z.id_produktu = p.id
 GROUP BY s.id, p.id
 ORDER BY s.id, sprzedane_sztuki DESC;

```

sklep	produkt	sprzedane_sztuki
Sklep A	Myszka	10
Sklep A	Laptop	8
Sklep B	Monitor	2
Sklep C	Klawiatura	0

#### Wyjaśnienie:

- LEFT JOIN produkty → bierzemy wszystkie sklepy, nawet jeśli nie mają produktów.
- LEFT JOIN zamowienia → bierzemy wszystkie produkty, nawet jeśli nie mają zamówień.
- SUM(z.ilosc) → sumujemy liczbę sprzedanych sztuk dla każdego produktu.
- COALESCE(..., 0) → jeśli produkt nie ma zamówień, pokazujemy 0 zamiast NULL.
- GROUP BY s.id, p.id → agregujemy dane po sklepie i produkcie.
- ORDER BY s.id, sprzedane\_sztuki DESC → sortujemy dane po sklepie i liczbie sprzedanych sztuk.

#### Pokazuje wszystkie zamówienia, nawet jeśli nie ma dopasowanego produktu lub sklepu:

```

SELECT s.nazwa AS sklep,
       p.nazwa AS produkt,
       z.ilosc AS sprzedane_sztuki
  FROM sklepy s
 RIGHT JOIN produkty p ON p.id_sklepu = s.id
 RIGHT JOIN zamowienia z ON z.id_produktu = p.id
 GROUP BY s.id, p.id;

```

sklep	produkt	sprzedane_sztuki
Sklep A	Laptop	3
Sklep A	Laptop	5
Sklep A	Myszka	10
Sklep B	Monitor	2

#### Wyjaśnienie:

- RIGHT JOIN produkty p ON p.id\_sklepu = s.id → bierzemy wszystkie produkty, nawet jeśli nie

- mają sklepu.
- RIGHT JOIN** zamówienia z ON z.id\_produktu = p.id → bierzemy wszystkie zamówienia, nawet jeśli nie mają przypisanego produktu.
- Jeśli w tabeli produkty lub sklepy brakuje dopasowania → kolumny będą **NULL**.

## Lekcja

### Temat: Group by

GROUP BY w **MySQL** służy do **grupowania rekordów**, które mają te same wartości w określonych kolumnach. Zazwyczaj używa się go **razem z funkcjami agregującymi**, takimi jak:

- COUNT() – zlicza ilość rekordów,
- SUM() – sumuje wartości,
- AVG() – liczy średnią,
- MAX() – zwraca wartość maksymalną,
- MIN() – zwraca wartość minimalną.

#### ◆ Przykład praktyczny

```
CREATE TABLE orders (
    id INT AUTO_INCREMENT PRIMARY KEY,
    customer_name VARCHAR(50),
    product_name VARCHAR(50),
    quantity INT,
    price DECIMAL(10, 2)
);
```

```
INSERT INTO orders (customer_name, product_name, quantity, price)
VALUES
('Jan Kowalski', 'Laptop', 1, 3500.00),
('Anna Nowak', 'Mysz', 2, 50.00),
('Jan Kowalski', 'Mysz', 1, 50.00),
('Piotr Wiśniewski', 'Klawiatura', 1, 120.00),
('Anna Nowak', 'Laptop', 1, 3400.00),
('Jan Kowalski', 'Laptop', 1, 3500.00);
```

#### ◆ Przykład 1 – Suma wartości zamówień dla każdego klienta

```
SELECT customer_name, SUM(price * quantity) AS total_spent  
FROM orders  
GROUP BY customer_name;
```

♦ **Przykład 2 – Ile produktów kupił każdy klient**

```
SELECT customer_name, COUNT(*) AS total_orders  
FROM orders  
GROUP BY customer_name;
```

♦ **Przykład 3 – Średnia cena produktów kupionych przez każdego klienta**

```
SELECT customer_name, AVG(price) AS avg_price  
FROM orders  
GROUP BY customer_name;
```

♦ **Przykład 4 – Grupowanie po dwóch kolumnach (klient + produkt)**

```
SELECT customer_name, product_name, SUM(quantity) AS total_quantity  
FROM orders  
GROUP BY customer_name, product_name;
```

♦ **Przykład 5 GROUP BY z HAVING**

Założymy, że chcemy zobaczyć **tylko tych klientów**, którzy **wydali łącznie więcej niż 1000 zł.**

```
SELECT customer_name, SUM(price * quantity) AS total_spent  
FROM orders  
GROUP BY customer_name  
HAVING SUM(price * quantity) > 1000;
```

♦ **Przykład 6 filtracja po liczbie zamówień**

Chcemy zobaczyć klientów, którzy złożyli więcej niż jedno zamówienie:

```
SELECT customer_name, COUNT(*) AS total_orders  
FROM orders  
GROUP BY customer_name  
HAVING COUNT(*) > 1;
```

♦ Różnica między WHERE a HAVING

- WHERE filzuje **pojedyncze rekordy przed grupowaniem**,
- HAVING filzuje **całe grupy po agregacji**.

📌 Przykład błędu:

-- ✗ Nie zadziała:

```
SELECT customer_name, SUM(price)
FROM orders
WHERE SUM(price) > 1000
GROUP BY customer_name;
```

📌 Poprawnie:

```
SELECT customer_name, SUM(price)
FROM orders
GROUP BY customer_name
HAVING SUM(price) > 1000;
```

#### ◆ Podsumowanie

- GROUP BY **grupuje** dane na podstawie wartości w kolumnach.
- Zazwyczaj używa się go z **funkcjami agregującymi** (SUM, COUNT, AVG, itp.).
- Może grupować po **jednej lub wielu kolumnach**.
- Często łączy się z HAVING, aby filtrować wyniki po agregacji (np. „pokaż tylko klientów, którzy wydali więcej niż 1000 zł”).

## Lekcja

**Temat:** Having, funkcje agregujące. Przykłady zapytań z datami, kwartałami i czasem

```
CREATE TABLE zamówienia (
    id INT AUTO_INCREMENT PRIMARY KEY,
    id_produktu INT NOT NULL,
    id_klienta INT NOT NULL,
    ilosc INT NOT NULL,
    kwota DECIMAL(10,2) NOT NULL,
    data_zamówienia DATE NOT NULL,
    status ENUM('oczekujące', 'zrealizowane', 'anulowane')
);
```

```
INSERT INTO zamowienia (id_produktu, id_klienta, ilosc, kwota, data_zamowienia, status)  
VALUES
```

```
(1, 1, 2, 200.00, '2025-04-01', 'zrealizowane'),  
(1, 1, 1, 200.00, '2025-05-01', 'zrealizowane'),  
(2, 1, 5, 300.00, '2025-10-05', 'oczekujące'),  
(3, 2, 3, 400.00, '2025-10-06', 'zrealizowane'),  
(3, 2, 1, 400.00, '2025-09-15', 'oczekujące'),  
(3, 2, 2, 400.50, '2025-11-05', 'anulowane'),  
(4, 3, 3, 600.00, '2025-10-07', 'zrealizowane'),  
(4, 3, 1, 250.00, '2025-11-02', 'anulowane');
```

HAVING to słowo kluczowe w MySQL, które często bywa mylone z WHERE.

W skrócie:

- WHERE filzuje pojedyncze wiersze przed grupowaniem,
- HAVING filzuje całe grupy po wykonaniu GROUP BY.

#### ♦ Składnia

```
SELECT kolumna, funkcja_agregująca(...)  
FROM tabela  
[WHERE warunek]  
GROUP BY kolumna  
HAVING warunek_na_grupie;
```

#### Różnica między WHERE a HAVING

Etap	Kiedy działa	Co filzuje
WHERE	Przed grupowaniem (GROUP BY)	Pojedyncze wiersze
HAVING	Po grupowaniu	Całe grupy wynikowe

#### Krok po kroku

1. Na początku chcesz zobaczyć sumę zamówień każdego klienta

```
SELECT id_klienta, SUM(kwota) AS suma_zamowien  
FROM zamowienia  
GROUP BY id_klienta;
```

id_klienta	suma_zamowien
1	700.00
2	1200.50
3	850.00

2. Teraz chcesz tylko klientów, którzy wydali więcej niż 300 zł

```
SELECT id_klienta, SUM(kwota) AS suma_zamowien
FROM zamowienia
GROUP BY id_klienta
HAVING SUM(kwota) > 300;
```

<b>id_klienta</b>	<b>suma_zamowien</b>
1	700.00
2	1200.50
3	850.00

**Można używać HAVING bez GROUP BY**

**Jeśli nie masz GROUP BY, HAVING może nadal działać, ale wtedy traktuje cały zestaw wyników jako jedną grupę.**

```
SELECT SUM(kwota) AS suma
FROM zamowienia
HAVING SUM(kwota) > 1000;
```

```
suma
2750.50
```

## Funkcje agregujące

1. **SUM()** z warunkiem i **GROUP BY**

**Suma wartości zamówień (ilość \* kwota) dla każdego klienta, tylko dla zamówień „zrealizowanych”.**

```
SELECT id_klienta,
      SUM(ilosc * kwota) AS laczna_kwota
FROM zamowienia
WHERE status = 'zrealizowane'
GROUP BY id_klienta;
```

<b>id_klienta</b>	<b>laczna_kwota</b>
1	600.00
2	1200.00

3	1800.00
---	---------

## 2. AVG() + ROUND()

**Średnia wartość pojedynczego zamówienia w zaokrągleniu do 2 miejsc po przecinku.**

```
SELECT id_klienta,  
       ROUND(AVG(ilosc * kwota),2) AS srednia_wartosc_zamowienia  
FROM zamowienia  
GROUP BY id_klienta;
```

	<b>id_klienta</b>	<b>srednia_wartosc_zamowienia</b>
	1	700.00
	2	800.33
	3	1025.00

## 3. COUNT(DISTINCT ...)

**Ile różnych produktów zamówił każdy klient.**

```
SELECT id_klienta,  
       COUNT(DISTINCT id_produktu) AS unikalne_produkty  
FROM zamowienia  
GROUP BY id_klienta;
```

	<b>id_klienta</b>	<b>unikalne_produkty</b>
	1	2
	2	1
	3	1

## 4. MIN() i MAX() z datami

**Najstarsze i najnowsze zamówienie dla każdego klienta.**

```
SELECT id_klienta,  
       MIN(data_zamowienia) AS pierwsze_zamowienie,  
       MAX(data_zamowienia) AS ostatnie_zamowienie  
FROM zamowienia  
GROUP BY id_klienta;
```

<b>id_klienta</b>	<b>pierwsze_zamowienie</b>	<b>ostatnie_zamowienie</b>
1	2025-04-01	2025-10-05
2	2025-09-15	2025-11-05
3	2025-10-07	2025-11-02

5. GROUP\_CONCAT()  (często pojawia się na egzaminie!)

**Wypisanie wszystkich statusów zamówień dla każdego klienta w jednej kolumnie.**

```
SELECT id_klienta,
       GROUP_CONCAT(DISTINCT status ORDER BY status SEPARATOR ', ') AS staty
  FROM zamowienia
 GROUP BY id_klienta;
```

<b>id_klienta</b>	<b>unikalne_produkty</b>
1	oczekujące, zrealizowane
2	oczekujące, zrealizowane, anulowane
3	zrealizowane, anulowane

### Podzapytanie z agregacją

Klient, który wydał najwięcej pieniędzy łącznie.

```
SELECT id_klienta, SUM(ilosc * kwota) AS suma
  FROM zamowienia
 GROUP BY id_klienta
 HAVING suma = (
    SELECT MAX(suma_kwot)
      FROM (
        SELECT SUM(ilosc * kwota) AS suma_kwot
          FROM zamowienia
         GROUP BY id_klienta
      ) AS t
);
```

<b>id_klienta</b>	<b>suma</b>
2	2401.00

**rozkładamy na czynniki**

```
SELECT SUM(ilosc * kwota) AS suma_kwot
FROM zamowienia
GROUP BY id_klienta
```

**Klient 1:**

$$\begin{aligned} & (2 * 200.00) + (1 * 200.00) + (5 * 300.00) \\ & = 400 + 200 + 1500 \\ & = 2100.00 \end{aligned}$$

**Klient 2:**

$$\begin{aligned} & (3 * 400.00) + (1 * 400.00) + (2 * 400.50) \\ & = 1200 + 400 + 801 \\ & = 2401.00 \end{aligned}$$

**Klient 3:**

$$\begin{aligned} & (3 * 600.00) + (1 * 250.00) \\ & = 1800 + 250 \\ & = 2050.00 \end{aligned}$$

<b>id_klienta</b>	<b>suma_kwot</b>
1	2100
2	2401
3	2050

Teraz wybieramy największą wartość:

```
SELECT MAX(suma_kwot)
FROM (
    SELECT SUM(ilosc * kwota) AS suma_kwot
    FROM zamowienia
    GROUP BY id_klienta
) AS t
```

Następnie pokaż tylko tych klientów, których łączna suma = największej sumie z całej tabeli.

 **Przykłady zapytań z datami, kwartałami i czasem**

## **1. Zamówienia z ostatniego miesiąca**

Pokazuje wszystkie zamówienia z ostatnich 30 dni względem bieżącej daty (**CURDATE()**):

```
SELECT *
FROM zamowienia
WHERE data_zamowienia >= DATE_SUB(CURDATE(), INTERVAL 1 MONTH);
```

## **2. Suma wartości zamówień w każdym kwartale**

To klasyczne zapytanie egzaminacyjne.

```
SELECT
YEAR(data_zamowienia) AS rok,
QUARTER(data_zamowienia) AS kwartal,
SUM(ilosc * kwota) AS suma_kwartału
FROM zamowienia
GROUP BY rok, kwartal
ORDER BY rok, kwartal;
```

## **3. Liczba zamówień według miesiąca**

Często spotykane na INF.03: raport miesięczny.

```
SELECT
YEAR(data_zamowienia) AS rok,
MONTH(data_zamowienia) AS miesiac,
COUNT(*) AS liczba_zamowien
FROM zamowienia
GROUP BY rok, miesiac
ORDER BY rok, miesiac;
```

## **4. Zamówienia, które miały miejsce więcej niż 2 miesiące temu**

Dobre na testy z **DATE\_SUB()**:

```
SELECT *
FROM zamowienia
WHERE data_zamowienia < DATE_SUB(CURDATE(), INTERVAL 2 MONTH);
```

## **5. Zamówienia z bieżącego kwartału**

Egzaminowe pytanie: „*Wyświetl wszystkie zamówienia z bieżącego kwartału*”

```
SELECT *
FROM zamowienia
WHERE QUARTER(data_zamowienia) = QUARTER(CURDATE())
AND YEAR(data_zamowienia) = YEAR(CURDATE());
```

## **6. Łączna wartość zamówień w każdym kwartale**

*„Podaj sumę wartości wszystkich zamówień w poszczególnych kwartałach 2025 roku.”*

```
SELECT
    QUARTER(data_zamowienia) AS kwartal,
    ROUND(SUM(ilosc * kwota), 2) AS wartosc_zamowien
FROM zamowienia
WHERE YEAR(data_zamowienia) = 2025
GROUP BY kwartal
ORDER BY kwartal;
```

## **7. Średnia wartość zamówienia w każdym miesiącu**

*„Wyznacz średnią wartość zamówienia dla każdego miesiąca 2025 roku.”*

```
SELECT
    DATE_FORMAT(data_zamowienia, '%Y-%m') AS miesiac,
    ROUND(AVG(ilosc * kwota), 2) AS srednia_kwota
FROM zamowienia
GROUP BY miesiac
ORDER BY miesiac;
✖ Funkcja DATE_FORMAT() formatuje datę — tutaj do postaci 2025-10 itd.
```

## **8. W którym kwartale było najwięcej zamówień?**

*„Znajdź kwartał, w którym złożono najwięcej zamówień.”*

```
SELECT
    QUARTER(data_zamowienia) AS kwartal,
    COUNT(*) AS liczba_zamowien
FROM zamowienia
GROUP BY kwartal
ORDER BY liczba_zamowien DESC
LIMIT 1;
```

## **9. Zamówienia złożone w weekendy**

*„Wyświetl zamówienia, które złożono w sobotę lub niedzielę.”*

```
SELECT *
FROM zamowienia
WHERE DAYOFWEEK(data_zamowienia) IN (1, 7);
```

✿ **DAYOFWEEK()** zwraca numer dnia tygodnia (**1 = niedziela, 7 = sobota**).

### Różnice między CURDATE() a innymi podobnymi funkcjami

Funkcja	Zwraca	Przykład wyniku
CURDATE()	Tylko datę (rok-miesiąc-dzień)	2025-11-05
CURRENT_DATE()	To samo co CURDATE()	2025-11-05
NOW()	Datę i czas	2025-11-05 14:32:11
SYSDATE()	Datę i czas w momencie <i>realnego wykonania</i>	2025-11-05 14:32:11
CURTIME()	Tylko czas	14:32:11

## Lekcja

**Temat:** Funkcje związane z czasem, datą, operatorami łańcuchowymi

### Funkcje daty i czasu

**Link do dokumentacji MySQL:**

<https://dev.mysql.com/doc/refman/8.4/en/date-and-time-functions.html>

Metoda	Wyjaśnienie	Przykład SQL	Wynik
<b>ADDDATE()</b>	Dodaje interwał do daty	SELECT ADDDATE('2024-01-01', INTERVAL 5 DAY);	2024-01-06
<b>ADDTIME()</b>	Dodaje czas	SELECT ADDTIME('10:00:00','02:30:00');	12:30:00
<b>CONVERT_TZ()</b>	Konwersja strefy czasowej	SELECT CONVERT_TZ('2024-01-01 12:00','UTC','Europe/Warsaw');	2024-01-01 13:00
<b>CURDATE()</b>	Bieżąca data	SELECT CURDATE();	2025-11-10
<b>CURTIME()</b>	Bieżący czas	SELECT CURTIME();	np. 14:22:01
<b>DATE()</b>	Zwraca część datową	SELECT DATE('2024-01-01 10:00:00');	2024-01-01
<b>DATE_ADD()</b>	Dodaje interwał do daty	SELECT DATE_ADD('2024-01-01', INTERVAL 1 MONTH);	2024-02-01
<b>DATE_FORMAT()</b>	Formatuje datę	SELECT DATE_FORMAT('2024-01-15','%d-%m-%Y');	15-01-2024
<b>DATE_SUB()</b>	Odejmuje interwał	SELECT DATE_SUB('2024-01-10', INTERVAL 3 DAY);	2024-01-07
<b>DATEDIFF()</b>	Różnica między datami	SELECT DATEDIFF('2024-02-01','2024-01-01');	31

<b>DAY()</b>	Dzień miesiąca	<code>SELECT DAY('2024-01-15');</code>	15
<b>DAYNAME()</b>	Nazwa dnia	<code>SELECT DAYNAME('2024-01-15');</code>	Tuesday
<b>DAYOFMONTH()</b>	Dzień miesiąca	<code>SELECT DAYOFMONTH('2024-01-15');</code>	15
<b>DAYOFWEEK()</b>	Numer dnia tyg. (1=nd)	<code>SELECT DAYOFWEEK('2024-01-15');</code>	3
<b>DAYOFYEAR()</b>	Dzień roku	<code>SELECT DAYOFYEAR('2024-01-15');</code>	15
<b>EXTRACT()</b>	Wyodrębnia część daty	<code>SELECT EXTRACT(YEAR FROM '2024-01-15');</code>	2024
<b>FROM_DAYS()</b>	Dni → data	<code>SELECT FROM_DAYS(750000);</code>	2044-01-22
<b>FROM_UNIXTIME()</b>	UNIX → data	<code>SELECT FROM_UNIXTIME(1700000000);</code>	2023-11-14 22:13:20
<b>HOUR()</b>	Pobiera godzinę	<code>SELECT HOUR('12:45:00');</code>	12
<b>LAST_DAY()</b>	Ostatni dzień miesiąca	<code>SELECT LAST_DAY('2024-02-10');</code>	2024-02-29
<b>MAKEDATE()</b>	Tworzy datę z dnia roku	<code>SELECT MAKEDATE(2024,32);</code>	2024-02-01
<b>MAKETIME()</b>	Tworzy czas	<code>SELECT MAKETIME(10,20,30);</code>	10:20:30
<b>MICROSECOND()</b>	Mikrosekundy	<code>SELECT MICROSECOND('10:00:00.123456');</code>	123456

<b>MINUTE()</b>	Minuta	SELECT MINUTE('12:45:30');	45
<b>MONTH()</b>	Numer miesiąca	SELECT MONTH('2024-05-10');	5
<b>MONTHNAME()</b>	Nazwa miesiąca	SELECT MONTHNAME('2024-05-10');	May
<b>NOW()</b>	Aktualny datetime	SELECT NOW();	2025-11-10 14:20:xx
<b>PERIOD_ADD()</b>	Dodaje miesiące do YYYYMM	SELECT PERIOD_ADD(202401,2);	202403
<b>PERIOD_DIFF()</b>	Ilość miesięcy między okresami	SELECT PERIOD_DIFF(202402,202401);	1
<b>QUARTER()</b>	Kwartał	SELECT QUARTER('2024-05-10');	2
<b>SEC_TO_TIME()</b>	Sekundy → czas	SELECT SEC_TO_TIME(3661);	01:01:01
<b>SECOND()</b>	Sekundy	SELECT SECOND('12:45:59');	59
<b>STR_TO_DATE()</b>	Tekst → data	SELECT STR_TO_DATE('31-01-2024','%d-%m-%Y');	2024-01-31
<b>SUBTIME()</b>	Odejmuje czas	SELECT SUBTIME('10:00:00','01:30:00');	08:30:00
<b>SYSDATE()</b>	Czas wykonania	SELECT SYSDATE();	2025-11-10...

<b>TIME()</b>	Czas z datetime	SELECT TIME('2024-01-01 12:30:45');	12:30:45
<b>TIME_FORMAT()</b>	Formatuje czas	SELECT TIME_FORMAT('12:30:45','%H:%i');	12:30
<b>TIME_TO_SEC()</b>	Czas → sekundy	SELECT TIME_TO_SEC('01:00:00');	3600
<b>TIMEDIFF()</b>	Różnica czasu	SELECT TIMEDIFF('12:00:00','10:00:00');	02:00:00
<b>TIMESTAMP()</b>	Tworzy datetime	SELECT TIMESTAMP('2024-01-01');	2024-01-01 00:00:00
<b>TIMESTAMPADD()</b>	Dodaje interwał	SELECT TIMESTAMPADD(HOUR,2,'2024-01-01 10:00');	2024-01-01 12:00
<b>TIMESTAMPDIFF()</b>	Różnica datetime	SELECT TIMESTAMPDIFF(DAY,'2024-01-01','2024-01-10');	9
<b>TO_DAYS()</b>	Data → dni od roku 0	SELECT TO_DAYS('2024-01-01');	739252
<b>TO_SECONDS()</b>	Data → sekundy od roku 0	SELECT TO_SECONDS('2024-01-01');	64092288000
<b>UNIX_TIMESTAMP()</b>	Aktualny UNIX time	SELECT UNIX_TIMESTAMP();	np. 1768060000
<b>UTC_DATE()</b>	Data UTC	SELECT UTC_DATE();	2025-11-10

<b>UTC_TIME()</b>	Czas UTC	SELECT UTC_TIME();	13:14:xx
<b>UTC_TIMESTAMP()</b>	Datetime UTC	SELECT UTC_TIMESTAMP();	2025-11-10 13:14:xx
<b>WEEK()</b>	Numer tygodnia	SELECT WEEK('2024-01-10');	1
<b>WEEKDAY()</b>	Dzień tyg. (0=pon)	SELECT WEEKDAY('2024-01-10');	3
<b>WEEKOFYEAR()</b>	Tydzień ISO	SELECT WEEKOFYEAR('2024-01-10');	2
<b>YEAR()</b>	Rok	SELECT YEAR('2024-01-10');	2024
<b>YEARWEEK()</b>	Rok + tydzień	SELECT YEARWEEK('2024-01-10');	202402

**UTC (Uniwersalny Czas Koordynowany) to światowy standard czasu atomowego, który służy jako podstawa do ustalania lokalnego czasu w różnych strefach czasowych.** Polska znajduje się w strefie czasowej UTC+1 (czas środkowoeuropejski, CET) zimą i UTC+2 (czas środkowoeuropejski letni, CEST) latem, a lokalny czas w Polsce jest o 1 lub 2 godziny późniejszy od czasu UTC.

- **Co to jest UTC:**
  - UTC to międzynarodowy standard czasu, który jest niezależny od ruchu obrotowego Ziemi i oparty na bardzo precyzyjnym czasie atomowym.

- Jest to punkt odniesienia, taki sam na całym świecie, do którego dodaje się lub od którego odejmuje się czas, aby uzyskać lokalny czas dla danej strefy czasowej.
- **UTC w Polsce:**
  - Polska leży w strefie czasowej UTC+1 (czas zimowy) lub UTC+2 (czas letni).
  - Czas zimowy (CET): Obowiązuje od ostatniej niedzieli października do ostatniej niedzieli marca. Czas lokalny w Polsce jest o 1 godzinę późniejszy niż UTC. (np. jeśli UTC to 12:00, w Polsce jest 13:00).
  - Czas letni (CEST): Obowiązuje od ostatniej niedzieli marca do ostatniej niedzieli października. Czas lokalny w Polsce jest o 2 godziny późniejszy niż UTC. (np. jeśli UTC to 12:00, w Polsce jest 14:00).

#### Zastosowania:

- Programowanie - przechowywanie dat i czasu w bazach danych
- Lotnictwo - koordynacja lotów międzynarodowych
- Internet - synchronizacja serwerów
- Telekomunikacja - koordynacja transmisji
- Nauka - precyzyjne pomiary czasu

**W praktyce:** Gdy widzisz znacznik czasu typu [2025-11-11T14:30:00Z](#), litera "Z" na końcu oznacza właśnie UTC (od "Zulu time" - wojskowego określenia UTC).

#### Przykłady:

- Polska: UTC+1 (zimą) lub UTC+2 (latem)
- Nowy Jork: UTC-5 (zimą) lub UTC-4 (latem)
- Tokio: UTC+9
- Londyn: UTC+0 (zimą) lub UTC+1 (latem)

## Funkcje i operatory łańcuchowe

### Link do dokumentacji MySQL:

<https://dev.mysql.com/doc/refman/8.4/en/string-functions.html>

Metoda	Wyjaśnienie	Przykład	Wynik
<b>ASCII()</b>	Zwraca kod ASCII pierwszego znaku	<code>SELECT ASCII('A');</code>	65
<b>BIN()</b>	Zwraca liczbę w postaci binarnej	<code>SELECT BIN(10);</code>	1010
<b>BIT_LENGTH()</b>	Zwraca długość napisu w bitach	<code>SELECT BIT_LENGTH('ABC');</code>	24
<b>CHAR()</b>	Zwraca znak odpowiadający podanemu kodowi ASCII	<code>SELECT CHAR(65);</code>	'A'
<b>CHAR_LENGTH()</b>	Liczba znaków (nie bajtów)	<code>SELECT CHAR_LENGTH('Łódź');</code>	4
<b>CHARACTER_LENGTH()</b>	To samo co CHAR_LENGTH()	<code>SELECT CHARACTER_LENGTH('Test');</code>	4
<b>CONCAT()</b>	Łączy napisy	<code>SELECT CONCAT('A', 'B', 'C');</code>	'ABC'
<b>CONCAT_WS()</b>	Łączy napisy z separatorem	<code>SELECT CONCAT_WS('-', 'A','B','C');</code>	'A-B-C'

<b>ELT()</b>	Zwraca element listy na indeksie (1-based)	<code>SELECT ELT(2,'jeden','dwa','trzy');</code>	'dwa'
<b>EXPORT_SET()</b>	Zamienia liczby bitowe na tekst ON/OFF	<code>SELECT EXPORT_SET(5, 'ON', 'OFF', '', 4);</code>	ON,OFF,O N,OFF
<b>FIELD()</b>	Zwraca pozycję pierwszego argumentu w liście	<code>SELECT FIELD('kot','pies','kot','mysz');</code>	2
<b>FIND_IN_SET()</b>	Pozycja elementu w liście CSV	<code>SELECT FIND_IN_SET('B', 'A,B,C');</code>	2
<b>FORMAT()</b>	Formatuje liczbę z przecinkami	<code>SELECT FORMAT(12345.678, 2);</code>	'12,345.68'
<b>FROM_BASE64()</b>	Dekoduje Base64	<code>SELECT FROM_BASE64('SGVsbG8=');</code>	'Hello'
<b>HEX()</b>	Zamienia liczbę lub tekst na hex	<code>SELECT HEX('ABC');</code>	414243
<b>INSERT()</b>	Wstawia podciąg w podaną pozycję, zastępując określoną liczbę znaków	<code>SELECT INSERT('abcdef', 3, 2, 'XYZ');</code>	'abXYZef'
<b>INSTR()</b>	Pozycja pierwszego wystąpienia podciągu	<code>SELECT INSTR('abcabc','ca');</code>	3

<b>LCASE()</b>	To samo co LOWER() – zamienia na małe litery	<code>SELECT LCASE('Test');</code>	'test'
<b>LEFT()</b>	Zwraca określoną liczbę znaków od lewej	<code>SELECT LEFT('abcdef', 3);</code>	'abc'
<b>LENGTH()</b>	Długość napisu w bajtach	<code>SELECT LENGTH('ABC');</code>	3
<b>LIKE</b>	Sprawdza dopasowanie wzorca	<code>SELECT 'Ala' LIKE 'A%';</code>	1
<b>LOAD_FILE()</b>	Wczytuje zawartość pliku (jeśli SQL ma dostęp)	<code>SELECT LOAD_FILE('/path/file.txt');</code>	<i>treść pliku</i>
<b>LOCATE()</b>	Pozycja podcięgu (jak INSTR, ale kolejność argumentów odwrotna)	<code>SELECT LOCATE('b','abc');</code>	2
<b>LOWER()</b>	Zamienia na małe litery	<code>SELECT LOWER('TEST');</code>	'test'
<b>LPAD()</b>	Uzupełnia z lewej do zadanej długości	<code>SELECT LPAD('7', 3, '0');</code>	'007'
<b>LTRIM()</b>	Usuwa spacje z lewej	<code>SELECT LTRIM(' test');</code>	'test'

<b>MAKE_SET()</b>	Zwraca listę elementów pasujących do bitów liczby	<code>SELECT MAKE_SET(5,'A','B','C');</code>	'A,C'
<b>MATCH() AGAINST()</b>	Pienotekstowe wyszukiwanie	<code>SELECT MATCH(text) AGAINST('kot');</code>	<i>ocena dopasowania</i>
<b>MID()</b>	Alias SUBSTRING()	<code>SELECT MID('abcdef', 2, 3);</code>	'bcd'
<b>NOT LIKE</b>	Odwrotność LIKE	<code>SELECT 'Ala' NOT LIKE 'K%';</code>	1
<b>NOT REGEXP</b>	Odwrotność REGEXP	<code>SELECT 'abc' NOT REGEXP '^[0-9]+\$';</code>	1
<b>OCT()</b>	Zamienia liczbę na system ósemkowy	<code>SELECT OCT(15);</code>	'17'
<b>OCTET_LENGTH()</b>	Alias LENGTH()	<code>SELECT OCTET_LENGTH('ABC');</code>	3
<b>ORD()</b>	Kod ASCII pierwszego znaku	<code>SELECT ORD('A');</code>	65
<b>POSITION()</b>	Alias LOCATE()	<code>SELECT POSITION('a' IN 'banan');</code>	2
<b>QUOTE()</b>	Zwraca tekst w bezpiecznej formie (escape)	<code>SELECT QUOTE("Ala's cat");</code>	'Ala\'s cat'

<b>REGEXP</b>	Dopasowanie wyrażenia regularnego	<code>SELECT 'abc123' REGEXP '[0-9]+';</code>	1
<b>REGEXP_INSTR()</b>	Pozycja dopasowania regsexu	<code>SELECT REGEXP_INSTR('abc123','[0-9]+');</code>	4
<b>REGEXP_LIKE()</b>	Czy pasuje regex	<code>SELECT REGEXP_LIKE('test123','[a-z]+');</code>	1
<b>REGEXP_REPLACE()</b>	Zamienia dopasowane fragmenty	<code>SELECT REGEXP_REPLACE('a1b2c3','[0-9]','X');</code>	'aXbXcX'
<b>REGEXP_SUBSTR()</b>	Zwraca fragment pasujący do regsexu	<code>SELECT REGEXP_SUBSTR('abc123','[0-9]+');</code>	'123'
<b>REPEAT()</b>	Powtarza tekst	<code>SELECT REPEAT('A',3);</code>	'AAA'
<b>REPLACE()</b>	Podmienia tekst	<code>SELECT REPLACE('ala ma kota','a','X');</code>	'XIX mX kotX'
<b>REVERSE()</b>	Odwraca napis	<code>SELECT REVERSE('kota');</code>	'atok'
<b>RIGHT()</b>	Znaki od prawej	<code>SELECT RIGHT('abcdef', 2);</code>	'ef'
<b>RLIKE</b>	Alias REGEXP	<code>SELECT 'abc' RLIKE '[a-z]+';</code>	1
<b>RPAD()</b>	Uzupełnia napis z prawej	<code>SELECT RPAD('A', 4, '.');</code>	'A...'

<b>RTRIM()</b>	Usuwa spacje z prawej	<code>SELECT RTRIM('test ');</code>	'test'
<b>SOUNDEX()</b>	Kod fonetyczny słów	<code>SELECT SOUNDEX('Robert');</code>	'R163'
<b>SOUNDS LIKE</b>	Porównanie brzmienia	<code>SELECT 'Robert' SOUNDS LIKE 'Rupert';</code>	1
<b>SPACE()</b>	Generuje spacje	<code>SELECT SPACE(5);</code>	' '
<b>STRCMP()</b>	Porównuje napisy	<code>SELECT STRCMP('abc','abd');</code>	-1
<b>SUBSTR()</b>	Podciąg (alias SUBSTRING)	<code>SELECT SUBSTR('abcdef',2,3);</code>	'bcd'
<b>SUBSTRING()</b>	Podciąg	<code>SELECT SUBSTRING('abcdef',3);</code>	'cdef'
<b>SUBSTRING_INDEX()</b>	Podciąg do N-tego separatora	<code>SELECT SUBSTRING_INDEX('a,b,c','','2');</code>	'a,b'
<b>TO_BASE64()</b>	Kodowanie Base64	<code>SELECT TO_BASE64('Hello');</code>	'SGVsbG8='
<b>TRIM()</b>	Usuwa spacje z obu stron	<code>SELECT TRIM(' test '');</code>	'test'
<b>UCASE()</b>	Alias UPPER()	<code>SELECT UCASE('abc');</code>	'ABC'
<b>UNHEX()</b>	Hex → tekst	<code>SELECT UNHEX('414243');</code>	'ABC'

<b>UPPER()</b>	Zamienia na wielkie litery	<code>SELECT UPPER('kot');</code>	'KOT'
<b>WEIGHT_STRING()</b>	Zwraca wewnętrzną wagę znaków (techniczne)	<code>SELECT WEIGHT_STRING('A');</code>	<i>(hex bajty)</i>

## Lekcja

**Temat:** Właściwości kolumn (pół) w MySQL: NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, DEFAULT, CHECK, AUTO\_INCREMENT, ENUM, COMMENT

W MySQL możesz nałożyć **wiele rodzajów właściwości (constraints)** na pojedynczą kolumnę albo na kilka kolumn naraz, żeby wymusić reguły zachowania danych.

## 1. NOT NULL

Kolumna **nie może przyjmować wartości NULL**.

Wymusza, że musisz zawsze podać wartość.

**Przykład:**

```
CREATE TABLE osoby (
    id INT NOT NULL,
    imie VARCHAR(100) NOT NULL
);
```

**Wyjaśnienie:**

- imię i id **musi** być podane.

## 2. UNIQUE

Wymusza **unikalne wartości** w kolumnie — nie mogą się powtarzać.

**Przykład:**

```
CREATE TABLE klienci (
    id INT PRIMARY KEY,
    email VARCHAR(255) UNIQUE
);
```

**Wyjaśnienie:**

Dwa takie same maile →  błęd.

Można też ustawić UNIQUE na **kilka kolumn naraz**:

UNIQUE (uczen\_id, kurs\_id)

**Dodanie UNIQUE do istniejącej tabeli**

```
ALTER TABLE klienci
ADD UNIQUE (email);
```

## Różnica: PRIMARY KEY vs UNIQUE

Cecha	PRIMARY KEY	UNIQUE
Musi być unikalne	✓ Tak	✓ Tak
Może być NULL	✗ Nie	✓ Tak
Można mieć więcej niż jeden?	✗ Nie (tylko jeden PK na tabelę)	✓ Tak (wiele UNIQUE)
Tworzy indeks	✓ Tak	✓ Tak

### ✓ 3. PRIMARY KEY

- jednoznacznie identyfikuje każdy wiersz (unikalny),
- automatycznie ma **UNIQUE + NOT NULL**.

#### Przykład:

```
CREATE TABLE produkty (
    produkt_id INT PRIMARY KEY,
    nazwa VARCHAR(100)
);
```

Możesz też zrobić klucz **złożony z kilku kolumn**:

PRIMARY KEY (zamowienie\_id, produkt\_id)

### ✓ 4. FOREIGN KEY

Łączy tabele — kolumna musi wskazywać na wartość z innej tabeli.

**Przykład:**

```
CREATE TABLE zamówienia (
```

```
    id INT PRIMARY KEY
```

```
);
```

```
CREATE TABLE produkty_w_zamowieniu (
```

```
    zamowienie_id INT,
```

```
    produkt_id INT,
```

```
    FOREIGN KEY (zamowienie_id) REFERENCES zamówienia(id)
```

```
);
```

Nie można dodać produktu do zamówienia, które nie istnieje.

 **5. DEFAULT**

Ustawia **wartość domyślną**, jeśli użytkownik nie poda swojej.

**Przykład:**

```
CREATE TABLE artykuły (
```

```
    id INT PRIMARY KEY,
```

```
    status VARCHAR(20) DEFAULT 'aktywny'
```

```
);
```

Jeśli nie podasz statusu → automatycznie będzie „aktywny”.

 **6. CHECK**

Wymusza spełnienie **logicznego warunku**.

**Przykład:**

```
CREATE TABLE pracownicy (
```

```
    id INT PRIMARY KEY,
```

```
    wiek INT CHECK (wiek >= 18 AND wiek <= 65)
```

);

Próba dodania wiek = 10 → ❌ błąd.

## ✓ 7. AUTO\_INCREMENT

Automatycznie zwiększa wartość w kolumnie liczbowej przy każdym INSERT.

**Przykład:**

```
CREATE TABLE logi (
    id INT PRIMARY KEY AUTO_INCREMENT,
    opis VARCHAR(255)
);
```

Dodajesz 5 logów → id będą: 1, 2, 3, 4, 5.

## ✓ 8. ENUM

Ogranicza wartości w kolumnie do zamkniętej listy dopuszczalnych opcji.

**Przykład:**

```
CREATE TABLE użytkownicy (
    id INT PRIMARY KEY,
    plec ENUM('M', 'K', 'INNE') DEFAULT 'INNE'
);
```

Próba zapisania plec = 'ABC' → ❌ błąd.

## ✓ 9. COMMENT

Pozwala dodać komentarz do kolumny — bardzo przydatne przy dokumentowaniu schematu.

**Przykład:**

```
CREATE TABLE produkty (
    id INT PRIMARY KEY,
    cena DECIMAL(10,2) COMMENT 'Cena brutto w zł'
);
```

W narzędziach typu phpMyAdmin, DBeaver zobaczysz komentarz przy kolumnie.

## Lekcja

**Temat:** ERD (Diagram związków encji ang. Entity Relationship Diagram). Polecenie DELETE i DROP

### ERD — diagram związków encji

To graficzny sposób przedstawienia struktury bazy danych:

- jakie **tabele (encje)** istnieją,
- jakie mają **atrybuty (kolumny)**,
- jakie występują **relacje** między tabelami:
  - **1:1**
  - **1:N**
  - **N:M**

ERD jest tworzony zanim powstanie baza danych, aby zaplanować jej strukturę.

**Encja (Entity)** = obiekt, który ma znaczenie w systemie i który chcesz zapisać w bazie.

**Inaczej mówiąc:**

- 👉 **Encja = tabela w bazie danych**
- 👉 **Atrybut = kolumna w tabeli**

Przykłady encji:

- **User (użytkownik)**
- **Product (produkt)**
- **Order (zamówienie)**
- **Invoice (faktura)**
- **Department (dział firmy)**

Każda encja ma klucz główny (Primary Key, PK) – unikalny identyfikator, np. id.

### **Tworzenie krok po kroku diagramu związków encji**

#### **Krok 1: Zidentyfikuj encje (tabele)**

#### **Krok 2: Określ atrybuty**

Dla każdej encji określasz pola.

Przykład:

Customer

- id
- first\_name
- last\_name
- email

#### **Krok 3: Ustal klucze główne**

Każda encja ma PK:

#### **Krok 4: Określ relacje między encjami**

1) Relacja 1:1 (One to One)

Jeden rekord odpowiada dokładnie jednemu rekordowi w drugiej tabeli.

2) Relacja 1:N (One to Many)

Jeden klient może mieć wiele zamówień.

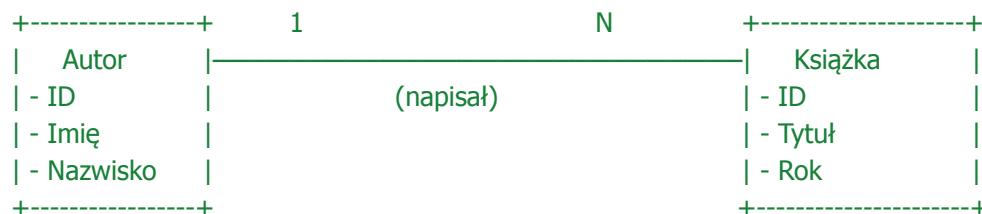
3) Relacja N:M (Many to Many)

Tworzy się tabelę pośredniczącą.

1. Relacja 1 : 1 (Osoba — Adres)

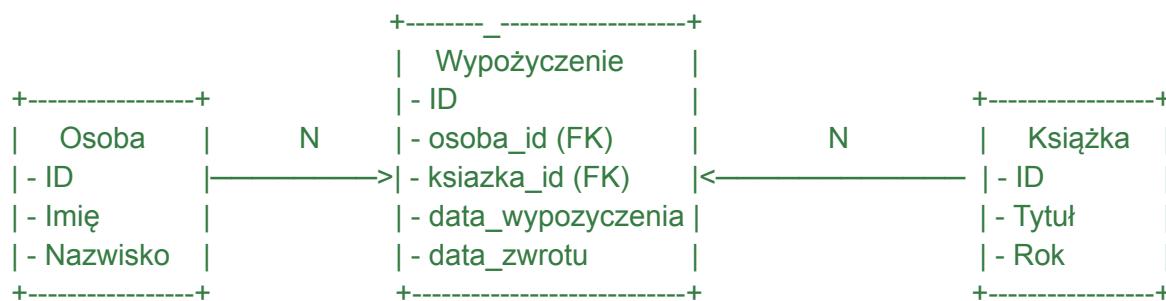


2. Relacja 1 : N (Autor — Książka)



3. Relacja N : N (Osoba — Książka) przez tabelę Wypożyczenie

W MySQL/SQL relacja N:N **zawsze wymaga tabeli pośredniej**.



N : N

(wiele osób wypożycza wiele książek)

Opis reprezentacji graficznej stopnia związku zostało pokazany na rysunku

**Rysunek**

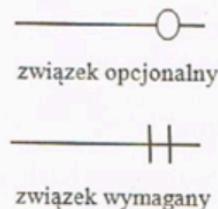
Graficzna reprezentacja  
związków zachodzą-  
cych między encjami



Opis reprezentacji graficznej opcjonalności związku zostało pokazany na rysunku

**Rysunek**

Graficzna reprezentacja  
opcjonalności związku



Diagramy ERD możemy tworzyć za pomocą różnych notacji. Najpopularniejsze są diagramy w zapisie według Martina i Chena.

## Polecenie **DELETE** i **DROP**

### ◆ **DELETE FROM**

Polecenie:

**DELETE FROM nazwa\_tabeli;**

**Usuwa rekordy (wiersze) z tabeli**, ale:

- nie usuwa struktury tabeli, kolumn ani jej definicji,
- nie resetuje auto\_increment (chyba że użyjesz TRUNCATE),
- może usuwać pojedyncze wiersze lub wszystkie — zależnie od warunku WHERE.

**Przykłady:**

Usuń wszystkie rekordy:

**DELETE FROM users;**

Usuń tylko wybrane:

**DELETE FROM users WHERE id = 5;**

### ◆ **DROP**

Polecenie:

**DROP TABLE nazwa\_tabeli;**

**Usuwa całą tabelę z bazy danych**, czyli:

- usuwa wszystkie dane,
- usuwa strukturę tabeli (kolumny, indeksy, klucze),
- usuwa definicję tabeli z katalogu bazy.

Po wykonaniu DROP tabela **przystaje istnieć**.

**Przykłady:**

Usuń tabelę:

```
DROP TABLE users;
```

Usuń całą bazę danych:

```
DROP DATABASE sklep;
```

## Lekcja

### Temat: Powtórzenie wiedzy na temat JOIN

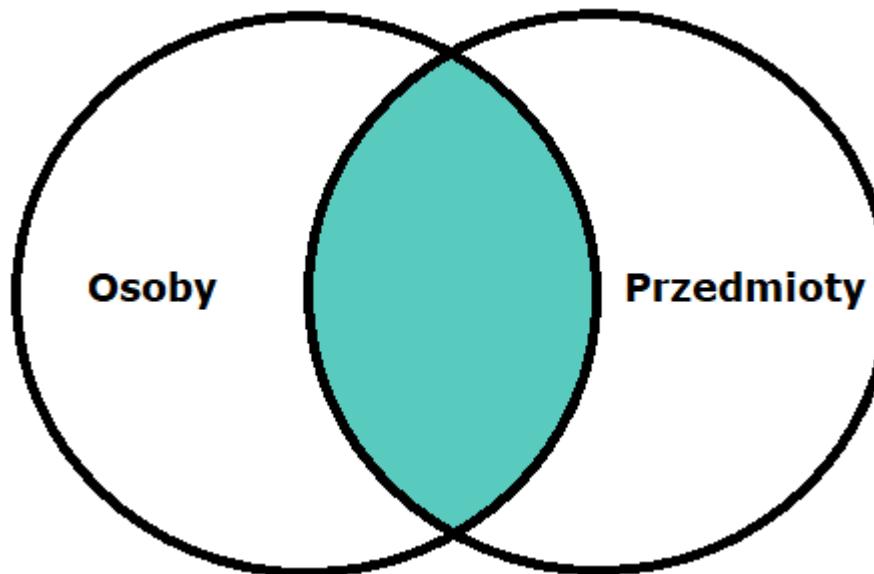
```
DROP TABLE IF EXISTS Przedmioty;
DROP TABLE IF EXISTS Osoby;

CREATE TABLE Osoby (
    osoba_id INT AUTO_INCREMENT PRIMARY KEY,
    imie VARCHAR(50) NOT NULL,
    nazwisko VARCHAR(50) NOT NULL
);
CREATE TABLE Przedmioty (
    przedmiot_id INT AUTO_INCREMENT PRIMARY KEY,
    nazwa VARCHAR(100) NOT NULL,
    osoba_id INT,
    CONSTRAINT fk_przedmiot_osoba FOREIGN KEY (osoba_id) REFERENCES Osoby(osoba_id)
);
INSERT INTO Osoby (imie, nazwisko) VALUES
('Jan', 'Kowalski'),
('Anna', 'Nowak'),
('Piotr', 'Zieliński'),
('Kasia', 'Wiśniewska'),
```

```
('Patryk', 'Nowakowski');

INSERT INTO Przedmioty (nazwa, osoba_id) VALUES
('Laptop', 1),
('Telefon', 1),
('Rower', 2),
('Książka', 3),
('Plecak', 4),
('Kubek', null);
```

- ◆ **INNER JOIN** - czyli wszystkie wspólne rekordy, bez NULL

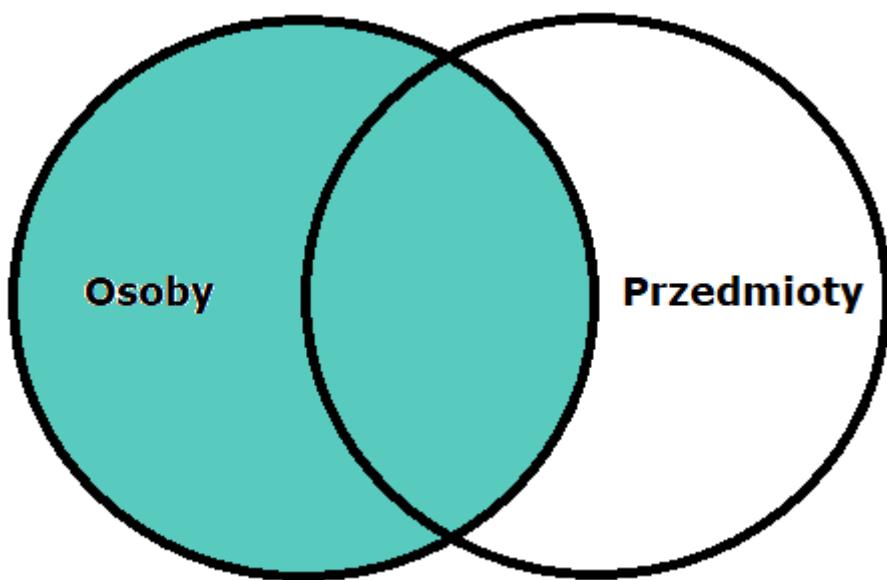


```
SELECT Osoby.imie, Osoby.nazwisko, Przedmioty.nazwa
FROM Osoby
INNER JOIN Przedmioty ON Osoby.osoba_id = Przedmioty.osoba_id;
```

Wynik:

<b>imie</b>	<b>nazwisko</b>	<b>nazwa</b>
Jan	Kowalski	Laptop
Jan	Kowalski	Telefon
Anna	Nowak	Rower
Piotr	Zieliński	Książka
Kasia	Wiśniewska	Plecak

- ♦ **LEFT JOIN** - czyli wszystkie rekordy z lewej tabeli. W naszym przypadku lewa tabela to Osoby. Jeśli Osoba jest a nie ma dopasowania w tabeli Przedmioty również się wyświetli.

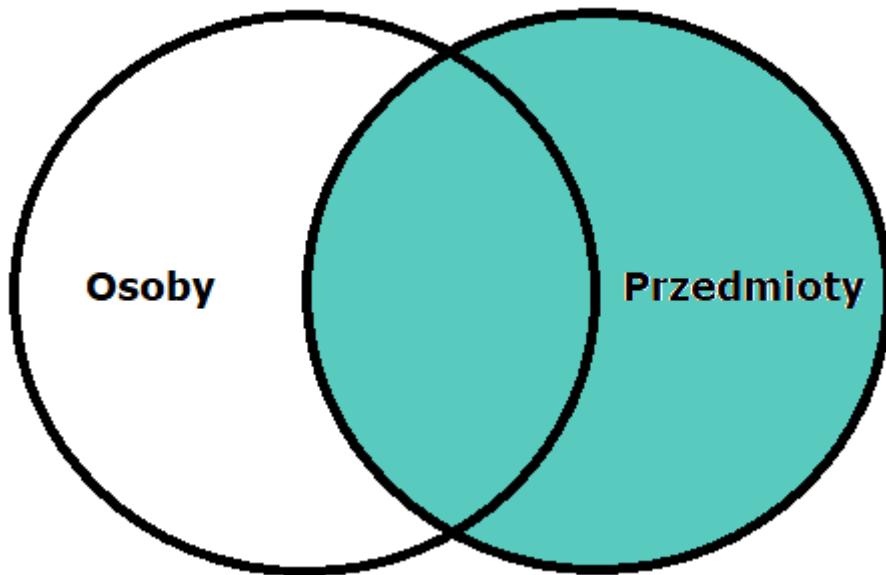


```
SELECT Osoby.imie, Osoby.nazwisko, Przedmioty.nazwa  
FROM Osoby  
LEFT JOIN Przedmioty ON Osoby.osoba_id = Przedmioty.osoba_id;
```

Wynik:

imie	nazwisko	nazwa
Jan	Kowalski	Laptop
Jan	Kowalski	Telefon
Anna	Nowak	Rower
Piotr	Zieliński	Książka
Kasia	Wiśniewska	Plecak
Patryk	Nowakowski	NULL

- ♦ **RIGHT JOIN** - czyli wszystkie rekordy z prawej tabeli. W naszym przypadku prawa tabela to Przedmioty. Jeśli Przedmiot nie ma dopasowania w tabeli Osoby również się wyświetli.



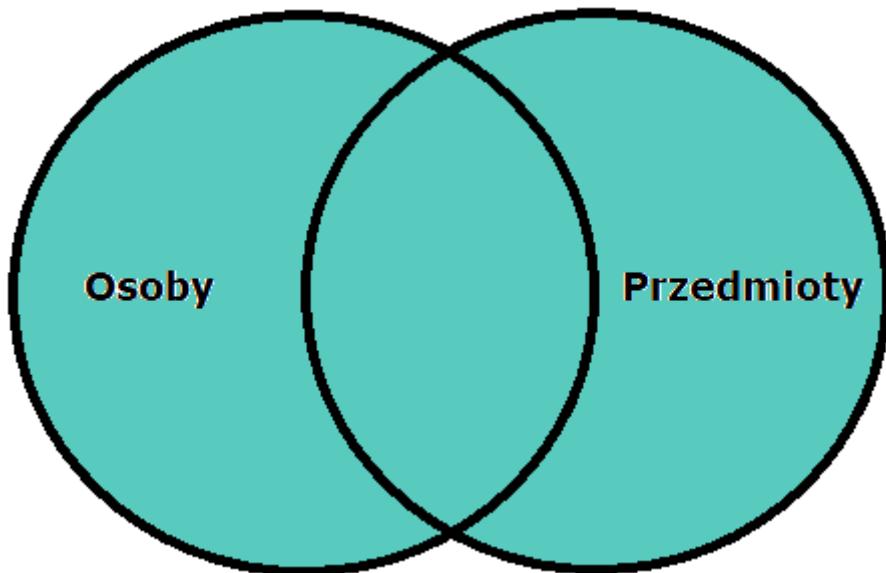
```
SELECT Osoby.imie, Osoby.nazwisko, Przedmioty.nazwa  
FROM Osoby  
RIGHT JOIN Przedmioty ON Osoby.osoba_id = Przedmioty.osoba_id;
```

Wynik:

imie	nazwisko	nazwa
Jan	Kowalski	Laptop
Jan	Kowalski	Telefon
Anna	Nowak	Rower
Piotr	Zieliński	Książka
Kasia	Wiśniewska	Plecak
NULL	NULL	Kubek

- ♦ **FULL OUTER JOIN (LEFT JOIN, UNION, RIGHT JOIN )** - czyli wszystkie rekordy z prawej i lewej tabeli połączone.

**W MySQL nie ma instrukcji FULL OUTER JOIN.** Jednak można wykonać ten mechanizm za pomocą połączenia poleceń right join, left join i UNION.



```
SELECT o.imie, o.nazwisko, p.nazwa
FROM Osoby o
LEFT JOIN Przedmioty p ON o.osoba_id = p.osoba_id
```

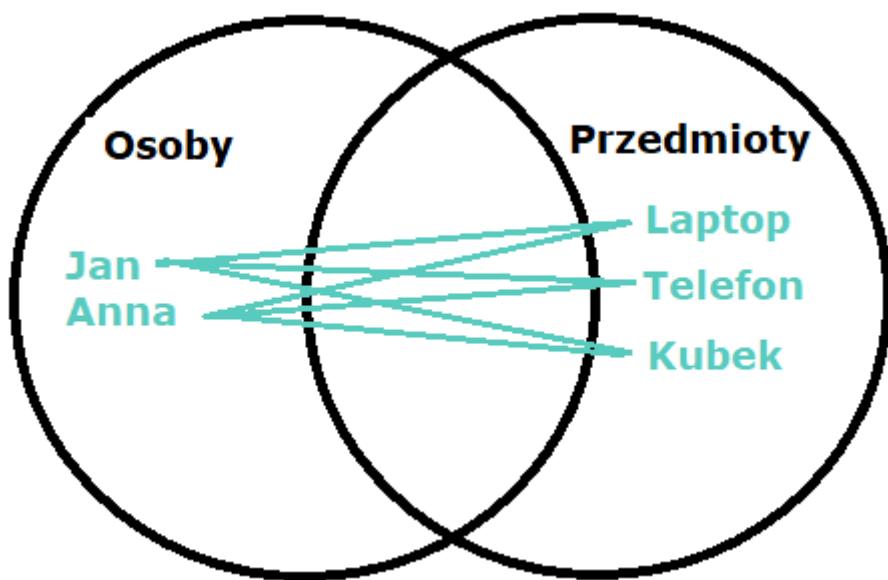
UNION

```
SELECT o.imie, o.nazwisko, p.nazwa
FROM Osoby o
RIGHT JOIN Przedmioty p ON o.osoba_id = p.osoba_id;
```

Wynik:

imie	nazwisko	nazwa
Jan	Kowalski	Laptop
Jan	Kowalski	Telefon
Anna	Nowak	Rower
Piotr	Zieliński	Książka
Kasia	Wiśniewska	Plecak
Patryk	Nowakowski	NULL
NULL	NULL	Kubek

- **CROSS JOIN** - łączy każdy wiersz z pierwszej tabeli z każdym wierszem z drugiej tabeli.



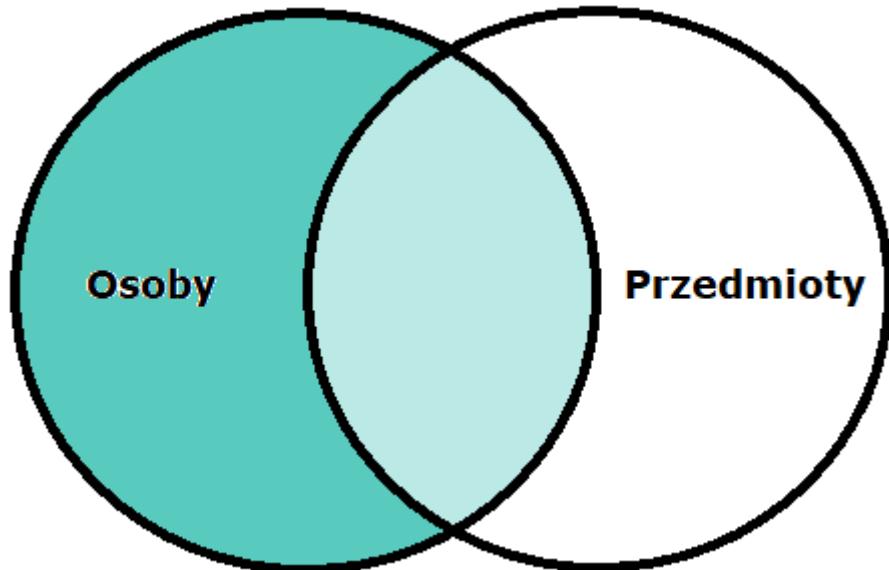
```
SELECT o.imie, p.nazwa
FROM Osoby o
CROSS JOIN Przedmioty p;
```

Wynik:

<b>imie</b>	<b>nazwa</b>
Jan	Laptop
Anna	Laptop
Piotr	Laptop
Kasia	Laptop
Patryk	Laptop
Jan	Telefon
Anna	Telefon
Piotr	Telefon
Kasia	Telefon
Patryk	Telefon
Jan	Rower
Anna	Rower
Piotr	Rower
Kasia	Rower

Patryk	Rower
Jan	Książka
Anna	Książka
Piotr	Książka
Kasia	Książka
Patryk	Książka
Jan	Plecak
Anna	Plecak
Piotr	Plecak
Kasia	Plecak
Patryk	Plecak
Jan	Kubek
Anna	Kubek
Piotr	Kubek
Kasia	Kubek
Patryk	Kubek

- ♦ **LEFT JOIN excluding INNER JOIN (LEFT JOIN wykluczający wiersze dopasowane)** - na początku wykonuje zapytanie LEFT JOIN. Następnie filtry wynik wyświetlając z lewej tabeli wartości nie mających dopasowania w tabeli prawej. Czyli w naszym przypadku z tabeli Osoby wyświetli wartości, które nie mają dopasowania

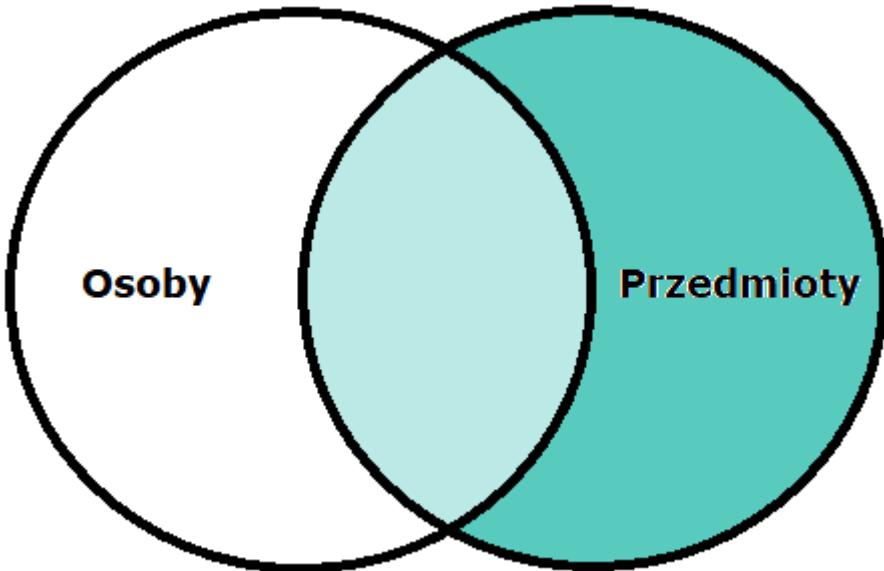


```
SELECT o.imie, o.nazwisko, p.nazwa
FROM Osoby o
LEFT JOIN Przedmioty p ON o.osoba_id = p.osoba_id
WHERE p.osoba_id IS NULL;
```

Wynik:

imie	nazwisko	nazwa
Patryk	Nowakowski	NULL

- **RIGHT JOIN excluding INNER JOIN (RIGHT JOIN wykluczający wiersze dopasowane)**  
- na początku wykonuje zapytanie **RIGHT JOIN**. Następnie filtry wynik wyświetlając z prawej tabeli wartości nie mających dopasowania w tabeli lewej. Czyli w naszym przypadku z tabeli Przedmioty wyświetli wartości, które nie mają dopasowania



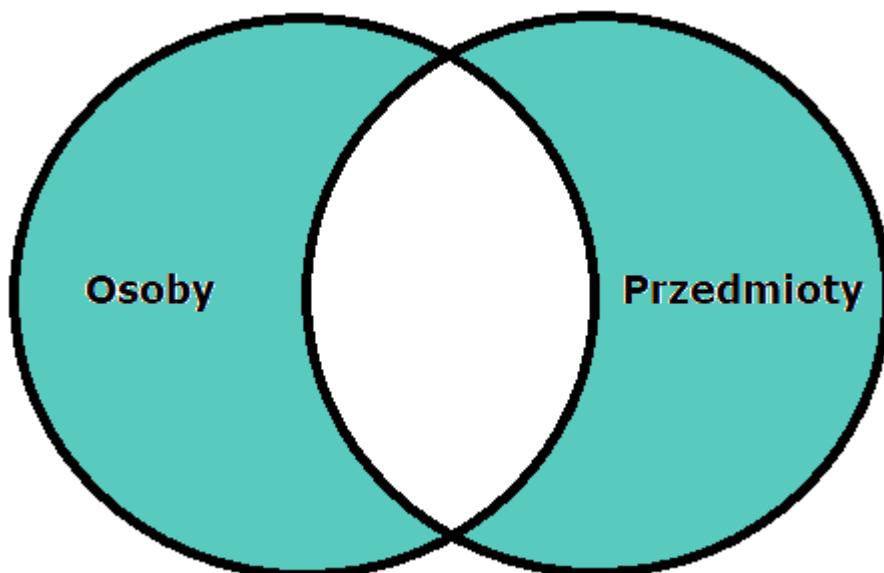
```
SELECT o.imie, o.nazwisko, p.nazwa
FROM Osoby o
RIGHT JOIN Przedmioty p ON o.osoba_id = p.osoba_id
WHERE o.osoba_id IS NULL;
```

Wynik:

imie	nazwisko	nazwa
NULL	NULL	Kubek

- **FULL OUTER JOIN excluding INNER JOIN (LEFT JOIN wykluczający wiersze dopasowane, UNION, RIGHT JOIN wykluczający wiersze dopasowane )** - czyli wszystkie rekordy z prawej i lewej tabeli połączone. Następnie odrzucamy te wiersze, które mają dopasowanie w obu tabelach.

**W MySQL nie ma instrukcji FULL OUTER JOIN.** Dla MySQL należy zastosować UNION. Czyli left join z wartościami nie mających dopasowania oraz right join z wartościami nie mających dopasowania łączymy z UNION.



```
SELECT o.imie, o.nazwisko, p.nazwa
FROM Osoby o
LEFT JOIN Przedmioty p ON o.osoba_id = p.osoba_id
WHERE p.osoba_id IS NULL
```

UNION

```
SELECT o.imie, o.nazwisko, p.nazwa
FROM Osoby o
RIGHT JOIN Przedmioty p ON o.osoba_id = p.osoba_id
WHERE o.osoba_id IS NULL;
```

Wynik:

imie	nazwisko	nazwa
Patryk	Nowakowski	NULL
NULL	NULL	Kubek

# Lekcja

## Temat: Normalizacja danych (1NF, 2NF, 3NF)

Normalizacja danych to **proces porządkowania struktury tabel w bazie danych**, tak aby:

- **nie było nadmiarowych danych**,
- **uniknąć anomalii** podczas wstawiania, usuwania i aktualizacji,
- zwiększyć spójność i logiczność danych.**

Najczęściej stosuje się trzy podstawowe formy normalizacji: **1NF, 2NF i 3NF**.

### 1NF — Pierwsza postać normalna

Tabela jest w **1NF**, jeśli:

- ✓ Każda kolumna ma **wartości atomowe** (niepodzielne).
- ✓ Nie ma list, wielu wartości w jednej komórce ani powtarzających się grup kolumn.
- ✓ Każdy wiersz jest unikalny (najczęściej dzięki kluczowi głównemu).

#### ✗ Przykład złamania 1NF:

<b>id</b>	<b>imię</b>	<b>hobby</b>
1	Ola	pływanie, bieganie

(— w jednej komórce są **dwie wartości**)

#### ✓ Wersja poprawna (1NF):

<b>id</b>	<b>imię</b>	<b>hobby</b>
1	Ola	pływanie
1	Ola	bieganie

lub tabela osobna **Hobby**.

### 2NF — Druga postać normalna

Tabela jest w **2NF**, jeśli:

- ✓ Jest w **1NF**
- ✓ **Każda kolumna niekluczowa zależy w pełni od całego klucza głównego**

Dotyczy to głównie tabel z **kluczem złożonym** (wiele kolumn jako PK).

### **✗ Przykład złamania 2NF:**

#### **Tabela Zamówienia\_Produkty**

<b>zamówienie_id (PK)</b>	<b>produkt_id (PK)</b>	<b>produkt_nazwa</b>
10	5	Młotek

**produkt\_nazwa** zależy **tylko od produkt\_id**, a nie od całego klucza złożonego.

### **✓ Wersja poprawna:**

Rozdzielimy dane:

#### **Tabela: Zamówienia\_Produkty**

zamówienie_id	produkt_id
---------------	------------

#### **Tabela: Produkty**

produkt_id	produkt_nazwa
------------	---------------

## **3NF — Trzecia postać normalna**

Tabela jest w **3NF**, jeśli:

- ✓ Jest w **2NF**
- ✓ Kolumny niekluczowe **nie zależą od innych kolumn niekluczowych** (czyli **brak zależności przechodnich**).

### **✗ Przykład złamania 3NF:**

<b>pracownik_id</b>	<b>dział_id</b>	<b>dział_nazwa</b>
---------------------	-----------------	--------------------

**dział\_nazwa** zależy od **dział\_id**, a nie od **pracownik\_id**.

To zależność przechodnia → złamanie 3NF.

### **✓ Wersja poprawna:**

#### **Tabela Pracownicy**

pracownik_id	dział_id
--------------	----------

#### **Tabela Działy**

dział_id	dział_nazwa
----------	-------------

# Lekcja

**Temat:** Kopie zapasowe i przywracanie w MySQL. Rodzaje indeksów w MySQL / MariaDB

**Kopia zapasowa** to zapis danych z bazy MySQL do pliku, aby można je było **odzyskać w razie**:

- awarii serwera,
- usunięcia danych,
- ataku (np. ransomware),
- błędu użytkownika.

## Backupy w MySQL – metody i strategie

Wykonanie kopii zapasowej wykonuje się poleceniem mysqldump

### 1 mysqldump – backup logiczny

- odczytuje **strukturę bazy** (bazy, tabele, indeksy),
- odczytuje **dane z tabel**,
- zapisuje wszystko do **pliku tekstowego .sql**,
- w pliku są polecenia SQL (CREATE, INSERT).

#### ♦ Kopia jednej bazy danych

**mysqldump -u root -p nazwa\_bazy > backup.sql**

Co to znaczy:

- **-u root** → użytkownik MySQL
- **-p** → zapyta o hasło
- **nazwa\_bazy** → baza do skopiowania
- **backup.sql** → plik kopii zapasowej

📌 Po wykonaniu masz plik, który zawiera **całą bazę**.

#### ♦ Kopia wszystkich baz

**mysqldump -u root -p --all-databases > all\_backup.sql**

### **Przywracanie bazy**

- ◆ **Przywracanie do istniejącej bazy**

**mysql -u root -p nazwa\_bazy < backup.sql**

- ◆ **Przywracanie nowej bazy**

**mysql -u root -p < backup.sql**

(MySQL sam utworzy bazę, jeśli polecenia CREATE DATABASE są w pliku)

### **Backup a cyberbezpieczeństwo**

Kopie zapasowe:

- chronią przed **utrataą danych**,
- pozwalają odzyskać dane po **atakowi ransomware**,
- są elementem **polityki bezpieczeństwa**.

📌 Bez backupu = utrata danych na stałe.

- ◆ **mysqldump – Zalety**

- ✓ prosty
- ✓ przenośny
- ✓ idealny do nauki i małych baz

- ◆ **mysqldump – Wady**

- ✗ wolny przy dużych bazach
- ✗ brak prawdziwych backupów inkrementalnych

## **2 Percona XtraBackup – backup fizyczny**

**Percona XtraBackup to zaawansowane narzędzie do fizycznych kopii zapasowych MySQL/MariaDB.**

- kopiuje pliki danych (InnoDB)
- działa bez zatrzymywania serwera
- używane w firmach i produkcji

### **Percona XtraBackup:**

- kopiuje pliki **.ibd**, **.frm**, logi transakcji
- zapisuje je do katalogu backupu

- backup jest spójny (consistent)

**Percona XtraBackup wymaga pełnej instalacji MySQL (nie XAMPP)**

## Rodzaje indeksów w MySQL / MariaDB

### 1. PRIMARY KEY (klucz główny)

- Główny, **unikalny identyfikator wiersza**
- **Nie może być NULL**
- W InnoDB to **klastrowany indeks**

Przykład

```
CREATE TABLE klienci (
    id_klienta INT PRIMARY KEY,
    nazwisko VARCHAR(100)
);
```

Kiedy używać?

zawsze  
do WHERE id = ?  
do JOIN

### 2. UNIQUE INDEX

- Gwarantuje **unikalność wartości**
- Może zawierać NULL

Przykład

```
CREATE UNIQUE INDEX idx_email
ON klienci(email);
```

Kiedy używać?

e-mail, login  
dane biznesowo unikalne  
poprawia wydajność = i JOIN

### 3. INDEX (zwykły / NON-UNIQUE)

- Najczęściej używany indeks
- Może zawierać duplikaty

Przykład

```
CREATE INDEX idx_id_klienta
ON zamówienia(id_klienta);
```

Kiedy używać?

WHERE, JOIN, ORDER BY, GROUP BY

### 4. INDEX ZŁOŻONY (composite index)

- Indeks na **kilku kolumnach**

- Kolejność kolumn ma znaczenie

Przykład

```
CREATE INDEX idx_klient_kwota  
ON zamówienia(id_klienta, kwota);
```

Kiedy używać?  
kilka warunków w WHERE  
= + > / <  
dokładnie Twój przypadek

## 5. FULLTEXT INDEX

- Do **wyszukiwania tekstowego**
- MATCH ... AGAINST

Przykład

```
CREATE FULLTEXT INDEX idx_opis  
ON produkty(opis);
```

Kiedy używać?  
wyszukiwarki  
duże teksty  
NIE do LIKE '%tekst%'

## 6. SPATIAL INDEX

- Indeks dla danych przestrzennych (GEOMETRY)
- GIS, mapy

Przykład

```
CREATE SPATIAL INDEX idx_lokalizacja  
ON miejsca(lokalizacja);
```

## 7. INDEX PREFIX (częściowy)

- Indeks na **początku kolumny**
- Zmniejsza rozmiar indeksu

Przykład

```
CREATE INDEX idx_nazwisko_prefix  
ON klienci(nazwisko(10));
```

Kiedy używać?  
długie VARCHAR  
LIKE 'ABC%'

## 8. COVERING INDEX (indeks pokrywający)

- Indeks zawiera **wszystkie kolumny z SELECT**
- MySQL **nie czyta tabeli**

Przykład

```
CREATE INDEX idx_cover  
ON zamówienia(id_klienta, kwota, id_zamowienia);
```

# Lekcja

## Temat: Widoki (VIEW), tabele tymczasowe (temporary tables) w MySQL.

```
-- Tabela klientów
CREATE TABLE klienci (
    id_klienta INT PRIMARY KEY AUTO_INCREMENT,
    imie VARCHAR(50) NOT NULL,
    nazwisko VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL
);

-- Tabela zamówień
CREATE TABLE zamówienia (
    id_zamówienia INT PRIMARY KEY AUTO_INCREMENT,
    id_klienta INT NOT NULL,
    data DATE NOT NULL,
    kwota DECIMAL(10, 2) NOT NULL,
    FOREIGN KEY (id_klienta) REFERENCES klienci(id_klienta)
);

-- Wstawianie klientów
INSERT INTO klienci (imie, nazwisko, email) VALUES
('Jan', 'Kowalski', 'jan.kowalski@example.com'),
('Anna', 'Nowak', 'anna.nowak@example.com'),
('Piotr', 'Wiśniewski', 'piotr.wisniewski@example.com');

-- Wstawianie zamówień (mieszanka dat: nowe i stare)
INSERT INTO zamówienia (id_klienta, data, kwota) VALUES
(1, '2024-12-01', 150.00),
(1, '2025-06-15', 200.00),
(2, '2025-11-20', 300.00),
(2, '2024-05-10', 100.00),
(3, '2026-01-05', 250.00);
```

Widoki (**VIEW**) w MySQL to **wirtualne tabele**, które **nie przechowują danych**, tylko **zapamiętują zapytanie SQL**.

Za każdym razem, gdy odwołujesz się do widoku, MySQL wykonuje to zapytanie i zwraca wynik.

### Zastosowanie

Widoki są używane w różnych scenariuszach:

- **Uproszczenie zapytań:** Zamiast pisać skomplikowane JOIN-y za każdym razem, użytkownik może odwołać się do widoku.
- **Bezpieczeństwo:** Ograniczają dostęp do wrażliwych danych – np. widok pokazuje tylko wybrane kolumny, ukrywając resztę tabeli.

- **Abstrakcja danych:** Ukrywają strukturę bazy danych przed użytkownikami końcowymi, np. w aplikacjach webowych lub raportach.
- **Integracja systemów:** Ułatwiają migrację lub integrację z innymi narzędziami, prezentując dane w spójny sposób.
- **Optymalizacja:** W niektórych przypadkach (z materializowanymi widokami w nowszych wersjach MySQL lub InnoDB) mogą cache'ować wyniki dla szyszego dostępu.

## Znaczenie

Widoki mają kluczowe znaczenie w zarządzaniu bazami danych, ponieważ promują zasadę DRY (Don't Repeat Yourself) – unikają duplikowania kodu SQL. Poprawiają czytelność i utrzymywalność kodu, ułatwiają kontrolę dostępu (np. via GRANT na widokach) i wspierają modularność w dużych systemach. W kontekście MySQL, widoki są standardową funkcją od wersji 5.0, co czyni je istotnym narzędziem w projektowaniu skalowań aplikacji bazodanowych. Pomagają też w compliance z regulacjami jak GDPR, ograniczając ekspozycję danych osobowych.

## Zalety

- **Bezpieczeństwo i kontrola dostępu:** Można nadawać uprawnienia tylko do widoku, bez dostępu do tabel bazowych.
- **Uproszczenie:** Redukują złożoność zapytań dla użytkowników i deweloperów.
- **Aktualność danych:** Widoki zawsze pokazują bieżące dane, bez potrzeby ręcznego odświeżania.
- **Efektywność:** W dużych bazach ułatwiają optymalizację, np. poprzez indeksy na widokach (w MySQL 8.0+).
- **Łatwość utrzymania:** Zmiana w widoku wpływa na wszystkie zapytania go używające, bez modyfikacji kodu aplikacji.

## Wady

- **Wydajność:** Ponieważ dane są obliczane dynamicznie, skomplikowane widoki mogą spowalniać zapytania, zwłaszcza przy dużych zbiorach danych (brak materializacji w standardowych widokach MySQL).
- **Brak modyfikacji:** Widoki są tylko do odczytu (nie można INSERT/UPDATE/DELETE bezpośrednio, chyba że przez wyzwalacze lub updatable views w prostych przypadkach).
- **Zależności:** Zmiana struktury tabel bazowych może złamać widok, wymagając jego odtworzenia.
- **Ograniczona funkcjonalność:** Nie obsługują wszystkich operacji (np. ORDER BY w definicji widoku bez TOP/LIMIT), a w MySQL nie ma natywnej materializacji (trzeba używać tabel tymczasowych lub zewnętrznych narzędzi).
- **Złożoność debugowania:** Błędy w widokach mogą być trudne do namierzenia, bo są "czarną skrzynką" dla użytkownika.

## Tworzenie widoku

```
CREATE VIEW aktywni_klienci AS
SELECT k.imie, k.nazwisko, k.email, z.data, z.kwota
FROM klienci k
JOIN zamówienia z ON k.id_klienta = z.id_klienta
WHERE z.data > DATE_SUB(NOW(), INTERVAL 1 YEAR);
```

## Użycie widoku

```
SELECT * FROM aktywni_klienci;
```

**Tabele tymczasowe** (ang. *temporary tables*) w MySQL to specjalne tabele, które **istnieją tylko w ramach bieżącej sesji połączenia z bazą danych**. Jest ona automatycznie usuwana po zakończeniu sesji (np. po rozłączeniu się z serwerem MySQL). Tabele tymczasowe są **przydatne do przechowywania pośrednich wyników** zapytań,

**przetwarzania danych tymczasowo lub unikania konfliktów** z trwałymi tabelami. Są widoczne tylko dla użytkownika, który je utworzył, i nie wpływają na inne sesje.

```
-- Tworzenie tabeli tymczasowej  
CREATE TEMPORARY TABLE temp_suma_zamowien AS  
SELECT id_klienta, SUM(kwota) AS calkowita_kwota  
FROM zamowienia  
GROUP BY id_klienta;  
  
-- Użycie tabeli tymczasowej  
SELECT * FROM temp_suma_zamowien WHERE calkowita_kwota > 200;  
  
-- Usunięcie (opcjonalne, bo zniknie po sesji)  
DROP TEMPORARY TABLE IF EXISTS temp_suma_zamowien;
```

## Zastosowanie

Tabele tymczasowe są używane w scenariuszach, gdzie potrzebne jest tymczasowe przechowywanie pośrednich wyników:

- **Przetwarzanie złożonych zapytań:** Podzielenie skomplikowanego query na etapy, np. agregacja danych przed JOIN-em.
- **Optymalizacja wydajności:** Przechowywanie wyników podzapytań, aby uniknąć wielokrotnego obliczania (np. w raportach).
- **Testowanie i debugowanie:** Tworzenie mock danych bez modyfikacji stałych tabel.
- **Procedury przechowywane i wyzwalacze:** Przechowywanie tymczasowych danych w skryptach SQL.
- **Integracja z aplikacjami:** W sesjach użytkownika, np. w web app, do personalizowanych obliczeń bez obciążania głównej bazy.

## Znaczenie

Tabele tymczasowe mają istotne znaczenie w optymalizacji i zarządzaniu zasobami w MySQL, szczególnie w środowiskach o wysokim obciążeniu. Pozwalają na efektywne zarządzanie pamięcią (mogą być w pamięci lub na dysku, w zależności od silnika jak InnoDB lub MEMORY), co poprawia wydajność zapytań. W kontekście MySQL (od wersji 3.23, ale ulepszone w nowszych), wspierają skalowalność aplikacji, zapobiegając blokadom i konfliktom w wielosesyjnych środowiskach. Są kluczowe w compliance, bo nie pozostawiają śladów po sesji, co pomaga w ochronie danych tymczasowych.

## Zalety

- **Izolacja sesji:** Widoczne tylko dla bieżącego połączenia, co zapobiega konfliktom w aplikacjach wielu użytkowników.
- **Automatyczne czyszczenie:** Usuwane po zakończeniu sesji, co oszczędza miejsce i upraszcza zarządzanie.
- **Wydajność:** Mogą być szybsze niż podzapytania, zwłaszcza przy dużych zbiorach danych (np. używając ENGINE=MEMORY dla tabel w RAM).
- **Elastyczność:** Wspierają indeksy, klucze obce i większość operacji jak zwykłe tabele.
- **Bezpieczeństwo:** Nie wpływają na trwałe dane, idealne do testów lub tymczasowych obliczeń.

## Wady

- **Ograniczony zakres:** Niewidoczne poza sesją, co uniemożliwia udostępnianie danych między połączonymi.
- **Zużycie zasobów:** Przy dużych tabelach mogą zużywać dużo pamięci/dysku, prowadząc do błędów (np. "Out of memory").
- **Brak trwałości:** Dane giną po awarii sesji lub serwera, co wymaga ponownego tworzenia.
- **Złożoność w transakcjach:** W InnoDB, mogą być rollback'owane, ale nie zawsze zachowują się jak trwałe tabele.

- **Ograniczona funkcjonalność:** Nie można ich replikować w klastrach MySQL, a w starszych wersjach miały problemy z LOCK-ami.

**Uwaga:** jeśli **utworzysz tymczasową tabelę o takiej samej nazwie jak istniejąca stała tabela, MySQL będzie używać wersji tymczasowej w danej sesji**. Po jej usunięciu znów zobaczysz oryginalną tabelę.

#### Indeksy i klucze

**Możesz w tabelach tymczasowych: definiować PRIMARY KEY, UNIQUE, INDEX itp.**

```
CREATE TEMPORARY TABLE produkty_tmp (
    id INT PRIMARY KEY AUTO_INCREMENT,
    nazwa VARCHAR(100),
    cena DECIMAL(10,2),
    INDEX (cena)
);
```

- Jednak nie możesz tworzyć indeksów **FULLTEXT** ani **SPATIAL** w tabelach tymczasowych.