

# 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	SELECT DAY('2024-01-15');	15
<b>DAYNAME()</b>	Nazwa dnia	SELECT DAYNAME('2024-01-15');	Tuesday
<b>DAYOFMONTH()</b>	Dzień miesiąca	SELECT DAYOFMONTH('2024-01-15');	15
<b>DAYOFWEEK()</b>	Numer dnia tyg. (1=nd)	SELECT DAYOFWEEK('2024-01-15');	3
<b>DAYOFYEAR()</b>	Dzień roku	SELECT DAYOFYEAR('2024-01-15');	15
<b>EXTRACT()</b>	Wyodrębnia część daty	SELECT EXTRACT(YEAR FROM '2024-01-15');	2024
<b>FROM_DAYS()</b>	Dni → data	SELECT FROM_DAYS(750000);	2044-01-22
<b>FROM_UNIXTIME() )</b>	UNIX → data	SELECT FROM_UNIXTIME(1700000000);	2023-11-14 22:13:20
<b>HOUR()</b>	Pobiera godzinę	SELECT HOUR('12:45:00');	12
<b>LAST_DAY()</b>	Ostatni dzień miesiąca	SELECT LAST_DAY('2024-02-10');	2024-02-29
<b>MAKEDATE()</b>	Tworzy datę z dnia roku	SELECT MAKEDATE(2024,32);	2024-02-01
<b>MAKETIME()</b>	Tworzy czas	SELECT MAKETIME(10,20,30);	10:20:30
<b>MICROSECOND()</b>	Mikrosekundy	SELECT MICROSECOND('10:00:00.123456');	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	SELECT ASCII('A');	65
<b>BIN()</b>	Zwraca liczbę w postaci binarnej	SELECT BIN(10);	1010
<b>BIT_LENGTH()</b>	Zwraca długość napisu w bitach	SELECT BIT_LENGTH('ABC');	24
<b>CHAR()</b>	Zwraca znak odpowiadający podanemu kodowi ASCII	SELECT CHAR(65);	'A'
<b>CHAR_LENGTH()</b>	Liczba znaków (nie bajtów)	SELECT CHAR_LENGTH('Łódź');	4
<b>CHARACTER_LENGTH()</b>	To samo co CHAR_LENGTH()	SELECT CHARACTER_LENGTH('Test');	4
<b>CONCAT()</b>	Łączy napisy	SELECT CONCAT('A', 'B', 'C');	'ABC'
<b>CONCAT_WS()</b>	Łączy napisy z separatorem	SELECT CONCAT_WS('-', 'A','B','C');	'A-B-C'



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

<b>LEFT()</b>	Zwraca określoną liczbę znaków od lewej	SELECT LEFT('abcdef', 3);	'abc'
<b>LENGTH()</b>	Długość napisu w bajtach	SELECT LENGTH('ABC');	3
<b>LIKE</b>	Sprawdza dopasowanie wzorca	SELECT 'Ala' LIKE 'A%';	1
<b>LOAD_FILE()</b>	Wczytuje zawartość pliku (jeśli SQL ma dostęp)	SELECT LOAD_FILE('/path/file.txt');	<i>treść pliku</i>
<b>LOCATE()</b>	Pozycja podciągu (jak INSTR, ale kolejność argumentów odwrotna)	SELECT LOCATE('b','abc');	2
<b>LOWER()</b>	Zamienia na małe litery	SELECT LOWER('TEST');	'test'
<b>LPAD()</b>	Uzupełnia z lewej do zadanej długości	SELECT LPAD('7', 3, '0');	'007'
<b>LTRIM()</b>	Usuwa spacje z lewej	SELECT LTRIM(' test');	'test'
<b>MAKE_SET()</b>	Zwraca listę elementów pasujących do bitów liczby	SELECT MAKE_SET(5,'A','B','C');	'A,C'
<b>MATCH() AGAINST()</b>	Pełnotekstowe wyszukiwanie	SELECT MATCH(text) AGAINST('kot');	<i>ocena dopasowania</i>

<b>MID()</b>	Alias SUBSTRING()	SELECT MID('abcdef', 2, 3);	'bcd'
<b>NOT LIKE</b>	Odwrotność LIKE	SELECT 'Ala' NOT LIKE 'K%';	1
<b>NOT REGEXP</b>	Odwrotność REGEXP	SELECT 'abc' NOT REGEXP '^[0-9]+\$';	1
<b>OCT()</b>	Zamienia liczbę na system ósemkowy	SELECT OCT(15);	'17'
<b>OCTET_LENGTH()</b>	Alias LENGTH()	SELECT OCTET_LENGTH('ABC');	3
<b>ORD()</b>	Kod ASCII pierwszego znaku	SELECT ORD('A');	65
<b>POSITION()</b>	Alias LOCATE()	SELECT POSITION('a' IN 'banan');	2
<b>QUOTE()</b>	Zwraca tekst w bezpiecznej formie (escape)	SELECT QUOTE("Ala's cat");	'Ala\'s cat'
<b>REGEXP</b>	Dopasowanie wyrażenia regularnego	SELECT 'abc123' REGEXP '[0-9]+';	1
<b>REGEXP_INSTR()</b>	Pozycja dopasowania regexu	SELECT REGEXP_INSTR('abc123','[0-9]+');	4
<b>REGEXP_LIKE()</b>	Czy pasuje regex	SELECT REGEXP_LIKE('test123','[a-z]+');	1

<b>REGEXP_REPLACE()</b>	Zamienia dopasowane fragmenty	SELECT REGEXP_REPLACE('a1b2c3','[0-9]',' X');	'aXbXcX'
<b>REGEXP_SUBSTR()</b>	Zwraca fragment pasujący do regexu	SELECT REGEXP_SUBSTR('abc123','[0-9]+' );	'123'
<b>REPEAT()</b>	Powtarza tekst	SELECT REPEAT('A',3);	'AAA'
<b>REPLACE()</b>	Podmienia tekst	SELECT REPLACE('ala ma kota','a','X');	'XIX mX kotX'
<b>REVERSE()</b>	Odwraca napis	SELECT REVERSE('kota');	'atok'
<b>RIGHT()</b>	Znaki od prawej	SELECT RIGHT('abcdef', 2);	'ef'
<b>RLIKE</b>	Alias REGEXP	SELECT 'abc' RLIKE '[a-z]+';	1
<b>RPAD()</b>	Uzupełnia napis z prawej	SELECT RPAD('A', 4, '.');	'A...'
<b>RTRIM()</b>	Usuwa spacje z prawej	SELECT RTRIM('test ');	'test'
<b>SOUNDEX()</b>	Kod fonetyczny słów	SELECT SOUNDEX('Robert');	'R163'
<b>SOUNDS LIKE</b>	Porównanie brzmienia	SELECT 'Robert' SOUNDS LIKE 'Rupert';	1
<b>SPACE()</b>	Generuje spacje	SELECT SPACE(5);	' '

<b>STRCMP()</b>	Porównuje napisy	SELECT STRCMP('abc','abd');	-1
<b>SUBSTR()</b>	Podciąg (alias SUBSTRING)	SELECT SUBSTR('abcdef',2,3);	'bcd'
<b>SUBSTRING()</b>	Podciąg	SELECT SUBSTRING('abcdef',3);	'cdef'
<b>SUBSTRING_INDEX()</b>	Podciąg do N-tego separatora	SELECT SUBSTRING_INDEX('a,b,c',' ',2);	'a,b'
<b>TO_BASE64()</b>	Kodowanie Base64	SELECT TO_BASE64('Hello');	'SGVsbG8='
<b>TRIM()</b>	Usuwa spacje z obu stron	SELECT TRIM(' test ');	'test'
<b>UCASE()</b>	Alias UPPER()	SELECT UCASE('abc');	'ABC'
<b>UNHEX()</b>	Hex → tekst	SELECT UNHEX('414243');	'ABC'
<b>UPPER()</b>	Zamienia na wielkie litery	SELECT UPPER('kot');	'KOT'
<b>WEIGHT_STRING()</b>	Zwraca wewnętrzną wagę znaków (techniczne)	SELECT WEIGHT_STRING('A');	(hex bajty)

# Lekcja

**Temat:** ERD (Diagram związków encji ang. Entity Relationship Diagram). Właściwości kolumn (pól) w MySQL: NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, DEFAULT, CHECK, AUTO\_INCREMENT, ENUM, COMMENT. 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.

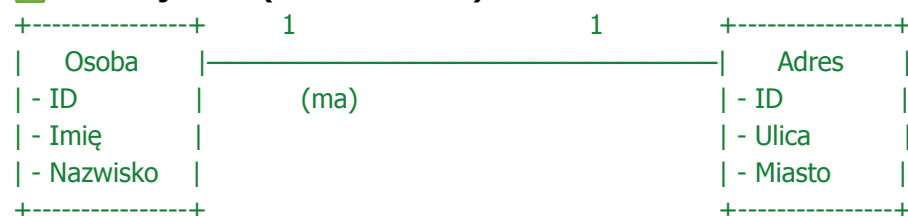
2) Relacja 1:N (One to Many)

Jeden klient może mieć wiele zamówień.

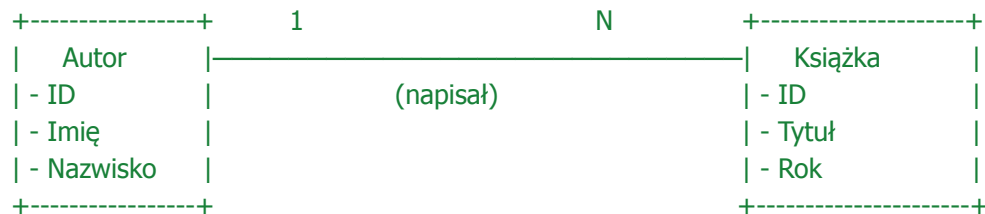
3) Relacja N:M (Many to Many)

Tworzy się tabelę pośredniczącą.

### ✓ 1. Relacja 1 : 1 (Osoba — Adres)

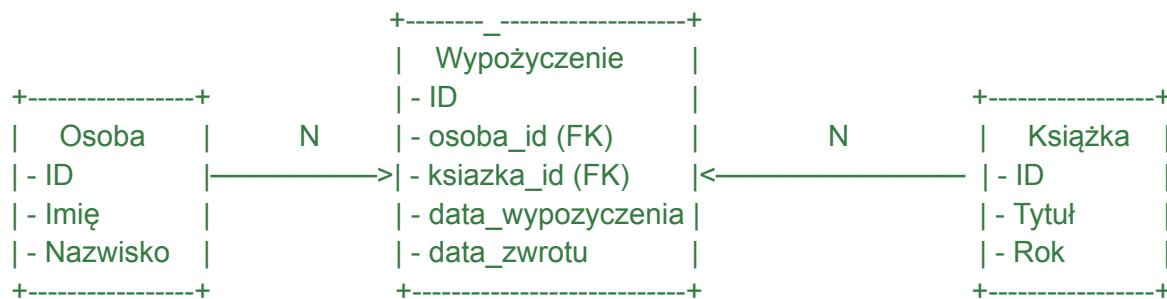


## ✓ 2. Relacja 1 : N (Autor — Książka)



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

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



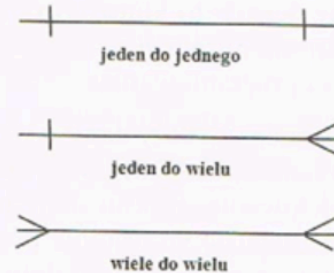
N : N  
(wiele osób wypożycza wiele książek)



Opis reprezentacji graficznej stopnia związku został pokazany na rysunku

**Rysunek**

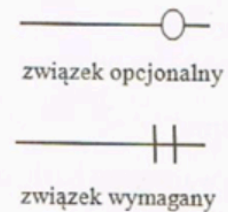
Graficzna reprezentacja związków zachodzących między encjami



Opis reprezentacji graficznej opcjonalności związku został 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.

Właściwości kolumn (pól) 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:

- imie 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 → **✗** błąd.

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 zamowienia (  
  id INT PRIMARY KEY  
);
```

```
CREATE TABLE produkty_w_zamowieniu (  
  zamowienie_id INT,  
  produkt_id INT,  
  FOREIGN KEY (zamowienie_id) REFERENCES zamowienia(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 uzytkownicy (  
  id INT PRIMARY KEY,  
  plec ENUM('M', 'K', 'INNE') DEFAULT 'INNE'  
);
```

Próba zapisania `plec = 'ABC'` → ❌ błąd.

## ✓ 9. COMMENT

Pozwala dopisać **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.

## 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 **przestaje istnieć**.

**Przykłady:**

Usuń tabelę:

**DROP TABLE** users;

Usuń całą bazę danych:

**DROP DATABASE** sklep;

## Lekcja

**Temat:** Replair. Akronym ACID, kategorie poleceń w SQL. System Zarządzania Bazą Danych (DBMS – DataBase Management System). Having, funkcje agregujące. Przykłady zapytań z datami, kwartałami i czasem

♦ Polecenie **REPAIR TABLE** w MySQL służy do **naprawy uszkodzonych tabel** oraz do **optymalizacji** pewnych typów tabel. Działa jednak tylko dla wybranych silników — głównie **MyISAM** oraz **ARCHIVE**.

Jeśli tabela MyISAM została uszkodzona (np. po awarii serwera), **REPAIR TABLE** próbuje:

- odbudować indeksy,
- odtworzyć strukturę danych,
- odzyskać jak najwięcej wierszy.

#### **Składnia**

```
REPAIR TABLE nazwa_tabeli;
```

Dodatkowe opcje:

- **QUICK** – naprawia tylko indeksy, bez skanowania danych
- **EXTENDED** – dogłębna naprawa, rekonstruuje plik danych (najwolniejsza)
- **USE\_FRM** – odbudowuje indeksy na podstawie pliku .frm (tylko MyISAM)

**REPAIR TABLE *nie naprawia* tabel InnoDB.**

♦ **Akrónim ACID w SQL oznacza cztery kluczowe właściwości transakcji w systemach baz danych:**

**A – Atomicity (Atomowość)**

Transakcja jest niepodzielna: albo wykonuje się w całości, albo wcale.

**C – Consistency (Spójność)**

Transakcja musi pozostawić bazę danych w stanie zgodnym z regułami i ograniczeniami (constraints).

**I – Isolation (Izolacja)**

Równocześnie wykonywane transakcje nie powinny wzajemnie sobie przeszkadzać — każda działa tak, jakby była wykonywana osobno.

**D – Durability (Trwałość)**

Po zatwierdzeniu transakcji (COMMIT) jej skutki są trwałe i nie zostaną utracone, nawet w przypadku awarii.



♦ **Podstawowe kategorie poleceń w SQL to:**

- **DDL (Data Definition Language)** – definiowanie struktury bazy danych (np. tworzenie tabel).
- **DML (Data Manipulation Language)** – manipulacja danymi (np. wstawianie, aktualizacja, usuwanie).
- **DCL (Data Control Language)** – zarządzanie uprawnieniami (np. GRANT, REVOKE).
- **DQL (Data Query Language)** – pobieranie danych (np. SELECT).
- **TCL (Transaction Control Language)** – zarządzanie transakcjami (np. COMMIT, ROLLBACK).

♦ **System Zarządzania Bazą Danych (DBMS – DataBase Management System)**

**to oprogramowanie, które umożliwia:**

- tworzenie baz danych,
- zapisywanie, modyfikowanie i usuwanie danych,
- zarządzanie dostępem użytkowników,
- zapewnianie bezpieczeństwa i integralności danych,
- wykonywanie zapytań (np. SQL),
- jednoczesny dostęp wielu użytkowników.

**Prościej:**

👉 **DBMS to program do zarządzania danymi w bazie – np. MySQL, PostgreSQL, Oracle, SQL Server.**

✅ **Jakie mechanizmy są NIEZBĘDNE dla Systemu Zarządzania Bazą Danych?**

**Wszystkie SZBD muszą mieć pewne podstawowe mechanizmy — zwykle wymienia się:**

1. Mechanizm składowania danych

**Przechowywanie danych na dysku, w tabelach, indeksach itp.**

2. Mechanizm dostępu do danych / język zapytań (np. SQL)

**Możliwość pobierania, wstawiania, usuwania, aktualizowania danych.**

3. Mechanizmy bezpieczeństwa

- **autoryzacja i autentykacja,**
- **role, użytkownicy,**
- **uprawnienia.**

4. Mechanizmy kontroli współbieżności (concurrency control)

**Zapewniają poprawną pracę wielu użytkowników *jednocześnie*.**

5. Mechanizmy zapewnienia integralności danych

- **klucze główne,**
- **klucze obce,**
- **ograniczenia (NOT NULL, UNIQUE, CHECK).**

**Chronią przed niepoprawnymi danymi.**

6. Mechanizmy odtwarzania po awarii (recovery)

**Przywracają działanie po:**

- **awarii systemu,**
- **utracie zasilania,**
- **błędach sprzętu.**

**Zapisywanie logów transakcyjnych, backupy itp.**

7. Mechanizmy zarządzania transakcjami (ACID)

**Każdy SZBD musi obsługiwać transakcje zgodnie z zasadą:**

- **A *atomicity* – niepodzielność**
- **C *consistency* – spójność**
- **I *isolation* – izolacja**

- **D durability – trwałość**

To fundament poprawnej pracy.

```
CREATE TABLE zamowienia (  
  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_zamowienia 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** filtruje pojedyncze wiersze przed grupowaniem,

→ **HAVING** filtruje 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 filtruje
<b>WHERE</b>	Przed grupowaniem ( <b>GROUP BY</b> )	Pojedyncze wiersze
<b>HAVING</b>	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;
```

id_klienta	suma_zamowien
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(ileosc * kwota) AS laczna_kwota  
FROM zamowienia  
WHERE status = 'zrealizowane'  
GROUP BY id_klienta;
```

id_klienta	laczna_kwota
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(ileosc * kwota), 2) AS srednia_wartosc_zamowienia  
FROM zamowienia  
GROUP BY id_klienta;
```

id_klienta	srednia_wartosc_zamowienia
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;
```

id_klienta	unikalne_produkty
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;
```

id_klienta	pierwsze_zamowienie	ostatnie_zamowienie
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 statusy
FROM zamowienia
GROUP BY id_klienta;
```

id_klienta	unikalne_produkty
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
);

```

id_klienta	suma
2	2401.00

rozkładamy na czynniki

```

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

```

**Klient 1:**

$(2 * 200.00) + (1 * 200.00) + (5 * 300.00)$   
 $= 400 + 200 + 1500$   
 $= 2100.00$

**Klient 2:**

$(3 * 400.00) + (1 * 400.00) + (2 * 400.50)$

= 1200 + 400 + 801  
= 2401.00

**Klient 3:**

(3 \* 600.00) + (1 \* 250.00)  
= 1800 + 250  
= 2050.00

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

## Lekcja

**Temat:** Właściwości kolumn (pól) w MySQL: NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, DEFAULT, CHECK, AUTO\_INCREMENT, ENUM, COMMENT. ERD (Diagram związków encji ang. Entity Relationship Diagram)

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:

- imie 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 → **✗** błąd.

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 zamowienia (  
  id INT PRIMARY KEY  
);
```

```
CREATE TABLE produkty_w_zamowieniu (  
  zamowienie_id INT,  
  produkt_id INT,  
  FOREIGN KEY (zamowienie_id) REFERENCES zamowienia(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 uzytkownicy (  
  id INT PRIMARY KEY,  
  plec ENUM('M', 'K', 'INNE') DEFAULT 'INNE'  
);
```

Próba zapisania `plec = 'ABC'` → ❌ błąd.

## ✅ 9. COMMENT

Pozwala dopisać **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.

## 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)

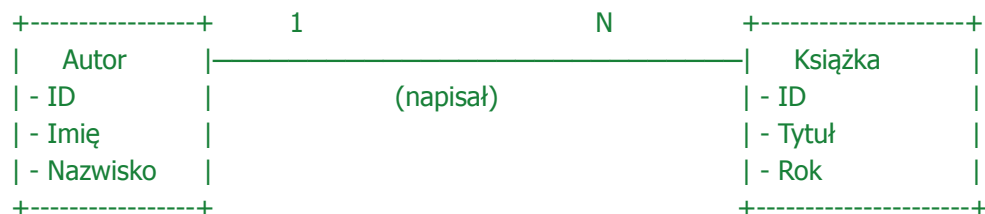
Tworzy się tabelę pośredniczącą.

### ✓ 1. Relacja 1 : 1 (Osoba — Adres)



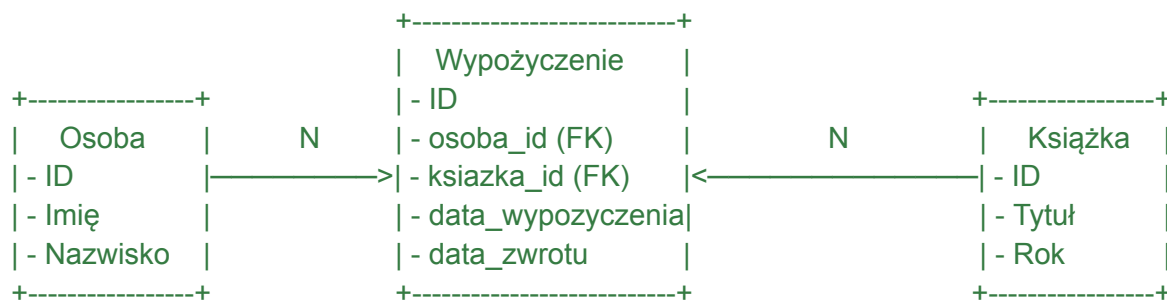
- ID	(ma)	- ID
- Imię		- Ulica
- Nazwisko		- Miasto
+-----+		+-----+

## ✓ 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

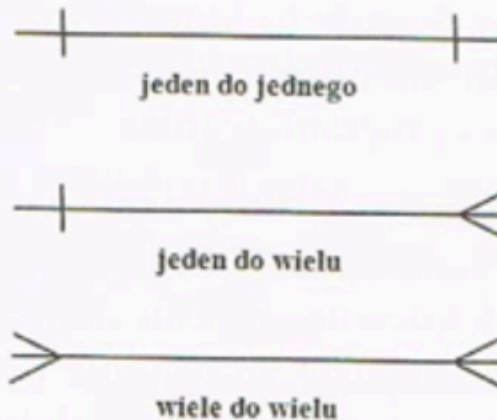
(wiele osób wypożycza wiele książek)



Opis reprezentacji graficznej stopnia związku został pokazany na rysunku

#### Rysunek

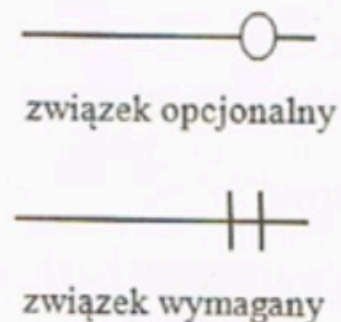
Graficzna reprezentacja związków zachodzących między encjami



Opis reprezentacji graficznej opcjonalności związku został 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.

**Na lekcji kartkówka** (właściwości kolumn).

kilka osob pisze kartkówkę z funkcji dat, czasu, ciągów znaków