

Lekcja

Temat: Oddanie i omówienie sprawdzianu

Lekcja

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

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

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

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

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

UTC_TIME()	Czas UTC	SELECT UTC_TIME();	13:14:xx
UTC_TIMESTAMP()	Datetime UTC	SELECT UTC_TIMESTAMP();	2025-11-10 13:14:xx
WEEK()	Numer tygodnia	SELECT WEEK('2024-01-10');	1
WEEKDAY()	Dzień tyg. (0=pon)	SELECT WEEKDAY('2024-01-10');	3
WEEKOFYEAR()	Tydzień ISO	SELECT WEEKOFYEAR('2024-01-10');	2
YEAR()	Rok	SELECT YEAR('2024-01-10');	2024
YEARWEEK()	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
ASCII()	Zwraca kod ASCII pierwszego znaku	<code>SELECT ASCII('A');</code>	65
BIN()	Zwraca liczbę w postaci binarnej	<code>SELECT BIN(10);</code>	1010
BIT_LENGTH()	Zwraca długość napisu w bitach	<code>SELECT BIT_LENGTH('ABC');</code>	24
CHAR()	Zwraca znak odpowiadający podanemu kodowi ASCII	<code>SELECT CHAR(65);</code>	'A'
CHAR_LENGTH()	Liczba znaków (nie bajtów)	<code>SELECT CHAR_LENGTH('Łódź');</code>	4
CHARACTER_LENGTH()	To samo co CHAR_LENGTH()	<code>SELECT CHARACTER_LENGTH('Test');</code>	4
CONCAT()	Łączy napisy	<code>SELECT CONCAT('A', 'B', 'C');</code>	'ABC'
CONCAT_WS()	Łączy napisy z separatorem	<code>SELECT CONCAT_WS('-', 'A', 'B', 'C');</code>	'A-B-C'

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

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

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

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

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

Lekcja

Temat: ERD (Diagram związków encji ang. Entity Relationship Diagram)

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.

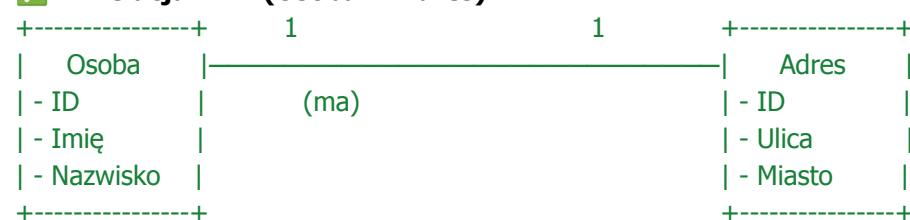
2) Relacja 1:N (One to Many)

Jeden klient może mieć wiele zamówień.

3) Relacja N:M (Many to Many)

Tworzy się tabelę pośredniczącą.

1. Relacja 1 : 1 (Osoba — Adres)

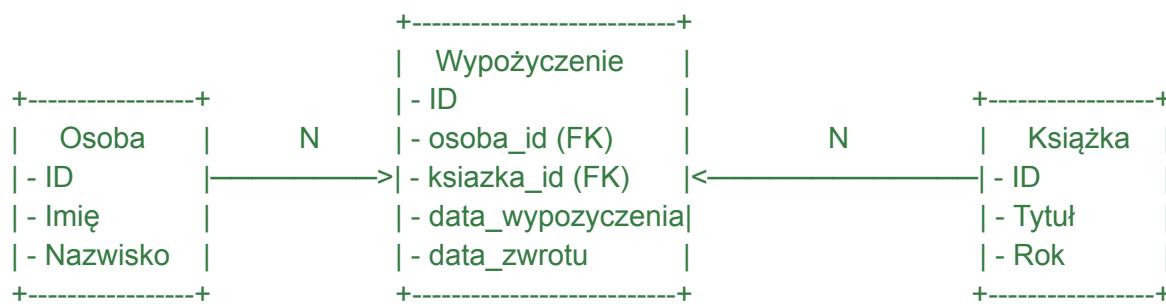


2. Relacja 1 : N (Autor — Książka)



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

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



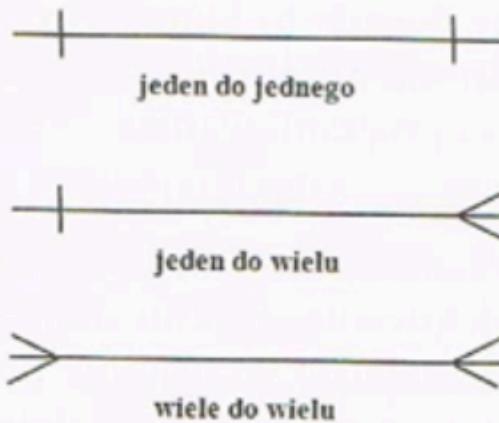
N : N

(wiele osób wypożycza wiele książek)

Opis reprezentacji graficznej stopnia związku został pokazany na rysunku

Rysunek

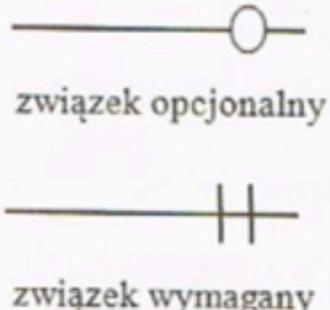
Graficzna reprezentacja związków zachodzących między encjami



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.

Lekcja

Temat: Właściwości kolumn (pola) w MySQL: NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, DEFAULT, CHECK, AUTO_INCREMENT, ENUM, COMMENT. Replair. Akrónim ACID, kategorie poleceń w SQL. Polecenie **DELETE i DROP.** System Zarządzania Bazą Danych (DBMS – DataBase Management System)

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 →  błąd.

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

UNIQUE (uczen_id, kurs_id)

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 użytkownicy (
    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 **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 tabeli).
- **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).

◆ **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;
```

◆ Polecenie ustawiające określoną wartość dla kolumny dla wszystkich rekordów:

```
UPDATE nazwa_tabeli
```

```
SET nazwa_kolumny = WARTOSC;
```

◆ 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.

Lekcja

Temat: Kopie zapasowe i przywracanie w MySQL

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 tabeli**,
- 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 **utratą danych**,
- pozwalają odzyskać dane po **ataku 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)

3 osób na następnej lekcji będzie pisało sprawdzian

8 osób będzie pisało kartkówkę z funkcji dat, czasu, ciągów znaków

3 osoby może poprawiać kartkówkę z funkcji dat, czasu, ciągów znaków