

ZBIÓR ZADAŃ

# Zapytania SQL

Treści zadań z rozwiązaniami i wyjaśnieniami

Kwalifikacja INF.03 — Tworzenie i administrowanie stronami i aplikacjami internetowymi

**SELECT • JOIN • GROUP BY • INSERT • UPDATE • DELETE • ALTER • GRANT**

## Spis treści

---

### Wstęp — jak korzystać z materiału

### Część I — Zadania praktyczne (zapytania do bazy danych)

Zestaw 1. Wędkarstwo

Zestaw 2. Biuro podróży

Zestaw 3. Prognoza pogody

Zestaw 4. Kwiaciarnia

Zestaw 5. Komis samochodowy

Zestaw 6. Biblioteka

Zestaw 7. Przychodnia lekarska

Zestaw 8. Kino

Zestaw 9. Hurtownia

Zestaw 10. Portal ogłoszeniowy

Zestaw 11. Dziennik ocen

Zestaw 12. Tworzenie bazy i administrowanie

### Część II — Pytania testowe z SQL (część pisemna)

### Część III — Ściąga: najważniejsze polecenia SQL

## Wstęp — jak korzystać z materiału

W części praktycznej egzaminu INF.03 jedno z zadań polega zawsze na zaprojektowaniu i wykonaniu kilku (zwykle czterech) zapytań SQL do przygotowanej bazy danych, a następnie zapisaniu ich w pliku kwerendy.txt i zrobieniu zrzutów ekranu z wynikami. Zapytania SQL pojawiają się również w części pisemnej jako pytania testowe.

Niniejszy zbiór zawiera zadania ułożone według schematu arkuszy CKE: opis struktury bazy danych, polecenia do wykonania oraz wzorcowe rozwiązania z krótkim wyjaśnieniem. Tematy baz (wędkarstwo, biuro podróży, prognoza pogody, kwiaciarnia, komis, biblioteka i inne) odpowiadają motywom, które regularnie pojawiają się na egzaminie, a polecenia obejmują wszystkie typy zapytań sprawdzanych w tej kwalifikacji.

### Ważne

Zadania są wzorowane na arkuszach egzaminacyjnych, ale nie są ich dosłownymi kopiami. Oryginalne arkusze wraz z zasadami oceniania są publikowane na stronie CKE ([cke.gov.pl](http://cke.gov.pl)) — warto przerobić także je, porównując swoje rozwiązania z kryteriami oceniania.

### Jak ćwiczyć:

- Przeczytaj treść polecenia i spróbuj samodzielnie napisać zapytanie, zanim spojrzysz na rozwiązanie.
- Porównaj swoje zapytanie z wzorcem — pamiętaj, że często istnieje kilka poprawnych wersji (np. z aliasami lub bez, JOIN lub warunek w WHERE).
- Najlepiej utwórz opisane tabele w phpMyAdmin (XAMPP) i sprawdź działanie zapytań na własnych danych.
- Na egzaminie wybieraj dokładnie te kolumny, o które prosi polecenie — dodatkowe kolumny lub `SELECT *` zamiast wskazanych pól mogą kosztować punkt.

## Część I — Zestaw 1. Wędkarstwo

### Baza danych: wedkowanie

Baza danych portalu dla wędkarzy przechowuje informacje o rybach, ich okresach ochronnych oraz łowiskach, w których występują.

| Tabela: Ryby |             |                                    |
|--------------|-------------|------------------------------------|
| Kolumna      | Typ         | Opis                               |
| id           | INT, PK     | identyfikator ryby                 |
| nazwa        | VARCHAR(20) | nazwa gatunku                      |
| wystepowanie | TEXT        | miejsca występowania               |
| styl_zycia   | INT         | 1 – drapieżna, 2 – spokojnego żeru |

| Tabela: Okres_ochronny |         |                            |
|------------------------|---------|----------------------------|
| Kolumna                | Typ     | Opis                       |
| id                     | INT, PK | identyfikator              |
| Ryby_id                | INT, FK | odwołanie do Ryby.id       |
| od_miesiaca            | INT     | początek okresu ochronnego |
| do_miesiaca            | INT     | koniec okresu ochronnego   |
| wymiar_ochronny        | INT     | wymiar ochronny w cm       |

| Tabela: Lowisko |         |                                             |
|-----------------|---------|---------------------------------------------|
| Kolumna         | Typ     | Opis                                        |
| id              | INT, PK | identyfikator łowiska                       |
| Ryby_id         | INT, FK | odwołanie do Ryby.id                        |
| akwen           | TEXT    | nazwa akwenu                                |
| wojewodztwo     | TEXT    | województwo                                 |
| rodzaj          | INT     | 1 – jezioro, 2 – morze, 3 – rzeka, 4 – staw |

### Zapytanie 1

Wybierz nazwy i miejsca występowania wszystkich ryb drapieżnych.

#### SQL

```
SELECT nazwa, wystepowanie
FROM Ryby
WHERE styl_zycia = 1;
```

**Wyjaśnienie:** Filtrowanie klauzulą WHERE — ryby drapieżne mają styl\_zycia równy 1.

## Zapytanie 2

Z tabeli Okres\_ochronny wybierz identyfikator ryby oraz wymiar ochronny dla ryb, których wymiar ochronny jest mniejszy niż 30 cm.

### SQL

```
SELECT Ryby_id, wymiar_ochronny
FROM Okres_ochronny
WHERE wymiar_ochronny < 30;
```

**Wyjaśnienie:** Operator porównania < zwraca wyłącznie rekordy o wartości mniejszej od podanej (30 nie jest uwzględniane).

## Zapytanie 3

Wybierz nazwę ryby oraz akwen i województwo łowisk będących rzekami (rodzaj = 3).

### SQL

```
SELECT Ryby.nazwa, Lowisko.akwen, Lowisko.wojewodztwo
FROM Ryby
JOIN Lowisko ON Ryby.id = Lowisko.Ryby_id
WHERE Lowisko.rodzaj = 3;
```

**Wyjaśnienie:** Dane pochodzą z dwóch tabel, dlatego łączymy je relacją klucz główny – klucz obcy (Ryby.id = Lowisko.Ryby\_id).

## Zapytanie 4

Do tabeli Ryby dodaj kolumnę dobowy\_limit typu liczbowego, przyjmującą tylko wartości nieujemne z zakresu 0–255.

### SQL

```
ALTER TABLE Ryby
ADD dobowy_limit TINYINT UNSIGNED;
```

**Wyjaśnienie:** TINYINT UNSIGNED przechowuje liczby 0–255. ALTER TABLE ... ADD dodaje nową kolumnę do istniejącej tabeli.

## Zestaw 2. Biuro podróży

### Baza danych: biuro

Baza biura podróży zawiera oferowane wycieczki oraz zdjęcia prezentowane na stronie WWW.

| Tabela: <b>zdjecia</b> |              |                         |
|------------------------|--------------|-------------------------|
| Kolumna                | Typ          | Opis                    |
| id                     | INT, PK      | identyfikator zdjęcia   |
| nazwaPliku             | VARCHAR(50)  | nazwa pliku graficznego |
| podpis                 | VARCHAR(100) | podpis pod zdjęciem     |

| Tabela: <b>wycieczki</b> |             |                               |
|--------------------------|-------------|-------------------------------|
| Kolumna                  | Typ         | Opis                          |
| id                       | INT, PK     | identyfikator wycieczki       |
| zdjecia_id               | INT, FK     | odwołanie do zdjecia.id       |
| dataWjazdu               | DATE        | data wyjazdu                  |
| cel                      | VARCHAR(50) | miejsce docelowe              |
| cena                     | DOUBLE      | cena w zł                     |
| dostepna                 | BOOLEAN     | 1 – dostępna, 0 – niedostępna |

### Zapytanie 1

Wybierz nazwy plików i podpisy wszystkich zdjęć, posortowane malejąco według podpisu.

#### SQL

```
SELECT nazwaPliku, podpis
FROM zdjecia
ORDER BY podpis DESC;
```

**Wyjaśnienie:** DESC oznacza sortowanie malejące (Z → A). Domyślny kierunek ORDER BY to ASC (rosnąco).

### Zapytanie 2

Wybierz cel, cenę oraz nazwę pliku zdjęcia dla wszystkich dostępnych wycieczek.

#### SQL

```
SELECT wycieczki.cel, wycieczki.cena, zdjecia.nazwaPliku
FROM wycieczki
JOIN zdjecia ON wycieczki.zdjecia_id = zdjecia.id
WHERE wycieczki.dostepna = 1;
```

**Wyjaśnienie:** Połączenie tabel przez klucz obcy zdjecia\_id oraz warunek na kolumnę logiczną dostepna.

### Zapytanie 3

Policz, ile jest wycieczek w cenie powyżej 1300 zł. Wynik wyświetl w kolumnie o nazwie liczba.

#### SQL

```
SELECT COUNT(*) AS liczba
FROM wycieczki
WHERE cena > 1300;
```

**Wyjaśnienie:** COUNT(\*) zlicza rekordy spełniające warunek, a AS nadaje kolumnie wynikowej alias.

## Zapytanie 4

Oznacz jako niedostępne wszystkie wycieczki z datą wyjazdu wcześniejszą niż 1 maja 2024 r.

### SQL

```
UPDATE wycieczki  
SET dostępna = 0  
WHERE dataWyjazdu < '2024-05-01';
```

**Wyjaśnienie:** Daty w MySQL zapisujemy w formacie 'RRRR-MM-DD' w apostrofach. UPDATE bez WHERE zmieniłby wszystkie rekordy!

## Zestaw 3. Prognoza pogody

### Baza danych: prognoza

Baza serwisu pogodowego przechowuje prognozy dla wybranych miast.

| Tabela: miasta |             |                      |
|----------------|-------------|----------------------|
| Kolumna        | Typ         | Opis                 |
| id             | INT, PK     | identyfikator miasta |
| nazwa          | VARCHAR(50) | nazwa miasta         |

| Tabela: pogoda    |         |                               |
|-------------------|---------|-------------------------------|
| Kolumna           | Typ     | Opis                          |
| id                | INT, PK | identyfikator prognozy        |
| miasta_id         | INT, FK | odwołanie do miasta.id        |
| data_prognozy     | DATE    | dzień prognozy                |
| temperatura_noc   | INT     | temperatura w nocy [°C]       |
| temperatura_dzien | INT     | temperatura w dzień [°C]      |
| opady             | INT     | prawdopodobieństwo opadów [%] |
| cisnienie         | INT     | ciśnienie [hPa]               |

### Zapytanie 1

Wybierz wszystkie dane prognoz dla miasta o identyfikatorze 1, posortowane rosnąco według daty prognozy.

```
SQL
SELECT *
FROM pogoda
WHERE miasta_id = 1
ORDER BY data_prognozy ASC;
```

**Wyjaśnienie:** Gwiazdka \* oznacza wszystkie kolumny tabeli. Słowo ASC można pominąć, bo jest domyślne.

### Zapytanie 2

Wybierz identyfikator i datę prognoz, w których prawdopodobieństwo opadów przekracza 30%.

```
SQL
SELECT id, data_prognozy
FROM pogoda
WHERE opady > 30;
```

**Wyjaśnienie:** Proste filtrowanie operatorem >.

### Zapytanie 3

Wybierz datę prognozy, temperaturę nocną oraz nazwę miasta dla prognoz, w których temperatura w nocy jest niższa niż 6°C.

#### SQL

```
SELECT p.data_prognozy, p.temperatura_noc, m.nazwa
FROM pogoda AS p
JOIN miasta AS m ON p.miasta_id = m.id
WHERE p.temperatura_noc < 6;
```

**Wyjaśnienie:** Aliasy tabel (p, m) skracają zapis. Słowo AS przy aliasie tabeli jest opcjonalne.

### Zapytanie 4

Dla każdego miasta oblicz średnią temperaturę dzienną zaokrągloną do jednego miejsca po przecinku. Kolumnę wyniku nazwij srednia.

#### SQL

```
SELECT m.nazwa, ROUND(AVG(p.temperatura_dzien), 1) AS srednia
FROM pogoda p
JOIN miasta m ON p.miasta_id = m.id
GROUP BY m.nazwa;
```

**Wyjaśnienie:** AVG() liczy średnią w obrębie grupy utworzonej przez GROUP BY, a ROUND(x, 1) zaokrągla do 1 miejsca.

### Zapytanie 5

Usuń z bazy wszystkie prognozy sporządzone dla dni wcześniejszych niż 1 stycznia 2024 r.

#### SQL

```
DELETE FROM pogoda
WHERE data_prognozy < '2024-01-01';
```

**Wyjaśnienie:** DELETE usuwa rekordy (strukturę tabeli pozostawia). Bez WHERE usunięte zostałyby wszystkie dane.

## Zestaw 4. Kwiaciarnia

### Baza danych: kwiaciarnia

Baza sieci kwiaciarni gromadzi dane o punktach sprzedaży oraz złożonych zamówieniach.

| Tabela: kwiaciarnie |             |                          |
|---------------------|-------------|--------------------------|
| Kolumna             | Typ         | Opis                     |
| id_kwiaciarni       | INT, PK     | identyfikator kwiaciarni |
| nazwa               | VARCHAR(50) | nazwa kwiaciarni         |
| ulica               | VARCHAR(50) | ulica                    |
| kod                 | VARCHAR(6)  | kod pocztowy             |
| miasto              | VARCHAR(50) | miasto                   |

| Tabela: zamowienia |              |                          |
|--------------------|--------------|--------------------------|
| Kolumna            | Typ          | Opis                     |
| id                 | INT, PK      | identyfikator zamówienia |
| id_kwiaciarni      | INT, FK      | odwołanie do kwiaciarni  |
| data               | DATE         | data zamówienia          |
| kwiaty             | VARCHAR(100) | zamówione kwiaty         |
| cena               | DOUBLE       | cena zamówienia          |

### Zapytanie 1

Wybierz nazwę i ulicę kwiaciarni znajdujących się w Malborku.

#### SQL

```
SELECT nazwa, ulica
FROM kwiaciarnie
WHERE miasto = 'Malbork';
```

**Wyjaśnienie:** Wartości tekstowe w warunku zapisujemy w apostrofach.

### Zapytanie 2

Dla każdej kwiaciarni wyświetl jej identyfikator oraz liczbę zamówień.

#### SQL

```
SELECT id_kwiaciarni, COUNT(*) AS liczba_zamowien
FROM zamowienia
GROUP BY id_kwiaciarni;
```

**Wyjaśnienie:** GROUP BY tworzy grupę dla każdej kwiaciarni, a COUNT(\*) zlicza zamówienia w grupie.

### Zapytanie 3

Wybierz nazwę kwiaciarni, miasto oraz zamówione kwiaty dla zamówień złożonych 7 stycznia 2017 r.

#### SQL

```
SELECT k.nazwa, k.miasto, z.kwiaty
FROM zamowienia z
JOIN kwiaciarnie k ON z.id_kwiaciarni = k.id_kwiaciarni
WHERE z.data = '2017-01-07';
```

**Wyjaśnienie:** Łączymy tabele po wspólnej kolumnie id\_kwiaciarni i filtrujemy po dokładnej dacie.

### Zapytanie 4

Utwórz użytkownika Anna na serwerze localhost z hasłem @nna23.

**SQL**

```
CREATE USER 'Anna'@'localhost' IDENTIFIED BY '@nna23';
```

**Wyjaśnienie:** Nazwa użytkownika i host są w osobnych apostrofach połączonych znakiem @.

### Zapytanie 5

Nadaj użytkownikowi Anna prawa do przeglądania, dodawania i modyfikowania danych w tabeli zamówienia bazy kwiaciarnia.

**SQL**

```
GRANT SELECT, INSERT, UPDATE  
ON kwiaciarnia.zamowienia  
TO 'Anna'@'localhost';
```

**Wyjaśnienie:** Zapis baza.tabela ogranicza uprawnienia do jednej tabeli; baza.\* oznaczałoby wszystkie tabele bazy.

## Zestaw 5. Komis samochodowy

### Baza danych: komis

Baza komis przechowuje ofertę samochodów oraz zamówienia klientów.

| Tabela: samochody |             |                         |
|-------------------|-------------|-------------------------|
| Kolumna           | Typ         | Opis                    |
| id                | INT, PK, AI | identyfikator samochodu |
| marka             | VARCHAR(30) | marka                   |
| model             | VARCHAR(30) | model                   |
| rocznik           | INT         | rok produkcji           |
| kolor             | VARCHAR(20) | kolor                   |
| stan              | VARCHAR(20) | stan techniczny         |
| cena              | INT         | cena w zł               |

| Tabela: zamowienia |             |                           |
|--------------------|-------------|---------------------------|
| Kolumna            | Typ         | Opis                      |
| id                 | INT, PK     | identyfikator zamówienia  |
| samochody_id       | INT, FK     | odwołanie do samochody.id |
| klient             | VARCHAR(50) | imię i nazwisko klienta   |
| telefon            | VARCHAR(15) | telefon klienta           |

### Zapytanie 1

Wybierz markę i model czerwonych samochodów wyprodukowanych po 2015 roku.

#### SQL

```
SELECT marka, model
FROM samochody
WHERE kolor = 'czerwony' AND rocznik > 2015;
```

**Wyjaśnienie:** AND wymaga spełnienia obu warunków jednocześnie.

### Zapytanie 2

Wybierz wszystkie dane samochodów marki Toyota lub Fiat w cenie od 20 000 do 40 000 zł.

#### SQL

```
SELECT *
FROM samochody
WHERE marka IN ('Toyota', 'Fiat')
AND cena BETWEEN 20000 AND 40000;
```

**Wyjaśnienie:** IN zastępuje kilka warunków połączonych OR. BETWEEN obejmuje obie wartości graniczne.

### Zapytanie 3

Dla każdej marki oblicz średnią cenę samochodu (alias srednia\_cena). Wynik posortuj malejąco według średniej.

**SQL**

```
SELECT marka, AVG(cena) AS srednia_cena
FROM samochody
GROUP BY marka
ORDER BY srednia_cena DESC;
```

**Wyjaśnienie:** W ORDER BY można użyć aliasu zdefiniowanego w SELECT.

### Zapytanie 4

Wyświetl klienta, jego telefon oraz markę i model zamówionego samochodu dla wszystkich zamówień.

**SQL**

```
SELECT z.klient, z.telefon, s.marka, s.model
FROM zamowienia z
JOIN samochody s ON z.samochody_id = s.id;
```

**Wyjaśnienie:** Klasyczne złączenie INNER JOIN (samo JOIN oznacza INNER JOIN).

### Zapytanie 5

Dodaj do tabeli samochody auto: Opel Astra, rocznik 2018, kolor srebrny, stan dobry, cena 35 000 zł.

**SQL**

```
INSERT INTO samochody (marka, model, rocznik, kolor, stan, cena)
VALUES ('Opel', 'Astra', 2018, 'srebrny', 'dobry', 35000);
```

**Wyjaśnienie:** Kolumny id nie podajemy — AUTO\_INCREMENT nada ją automatycznie. Liczby zapisujemy bez apostrofów.

## Zestaw 6. Biblioteka

### Baza danych: biblioteka

Baza biblioteki szkolnej przechowuje informacje o autorach, książkach, czytelnikach i wypożyczeniach.

| Tabela: autorzy |             |                      |
|-----------------|-------------|----------------------|
| Kolumna         | Typ         | Opis                 |
| id              | INT, PK     | identyfikator autora |
| imie            | VARCHAR(30) | imię                 |
| nazwisko        | VARCHAR(50) | nazwisko             |

| Tabela: ksiazki |              |                         |
|-----------------|--------------|-------------------------|
| Kolumna         | Typ          | Opis                    |
| id              | INT, PK      | identyfikator książki   |
| tytul           | VARCHAR(150) | tytuł                   |
| id_autor        | INT, FK      | odwołanie do autorzy.id |
| rok_wydania     | INT          | rok wydania             |

| Tabela: czytelnicy |             |                          |
|--------------------|-------------|--------------------------|
| Kolumna            | Typ         | Opis                     |
| id                 | INT, PK, AI | identyfikator czytelnika |
| imie               | VARCHAR(30) | imię                     |
| nazwisko           | VARCHAR(50) | nazwisko                 |
| kod                | VARCHAR(10) | kod karty                |

| Tabela: wypozyczenia |         |                                   |
|----------------------|---------|-----------------------------------|
| Kolumna              | Typ     | Opis                              |
| id                   | INT, PK | identyfikator                     |
| id_czytelnik         | INT, FK | odwołanie do czytelnicy.id        |
| id_ksiazka           | INT, FK | odwołanie do ksiazki.id           |
| data_wyp             | DATE    | data wypożyczenia                 |
| data_zwr             | DATE    | data zwrotu (NULL – nie zwrócono) |

### Zapytanie 1

Dodaj czytelnika Jan Kowalski o kodzie karty JK123.

#### SQL

```
INSERT INTO czytelnicy (imie, nazwisko, kod)
VALUES ('Jan', 'Kowalski', 'JK123');
```

**Wyjaśnienie:** Kolejność wartości w VALUES musi odpowiadać kolejności kolumn podanych w nawiasie.

## Zapytanie 2

Wybierz tytuły książek autora o nazwisku Sienkiewicz, posortowane alfabetycznie.

### SQL

```
SELECT k.tytul
FROM książki k
JOIN autorzy a ON k.id_autor = a.id
WHERE a.nazwisko = 'Sienkiewicz'
ORDER BY k.tytul;
```

**Wyjaśnienie:** Warunek dotyczy tabeli autorzy, a wynik tabeli książki — potrzebne złączenie.

## Zapytanie 3

Wybierz imiona i nazwiska czytelników, którzy mają niezwrócone książki. Każdy czytelnik ma pojawić się tylko raz.

### SQL

```
SELECT DISTINCT c.imie, c.nazwisko
FROM czytelnicy c
JOIN wypozyczenia w ON c.id = w.id_czytelnik
WHERE w.data_zwr IS NULL;
```

**Wyjaśnienie:** Wartość NULL sprawdzamy przez IS NULL (nie = NULL). DISTINCT usuwa powtórzenia.

## Zapytanie 4

Wyświetl nazwiska czytelników i liczbę ich wypożyczeń, ale tylko tych, którzy wypożyczyli więcej niż 2 książki.

### SQL

```
SELECT c.nazwisko, COUNT(w.id) AS ile
FROM czytelnicy c
JOIN wypozyczenia w ON c.id = w.id_czytelnik
GROUP BY c.id, c.nazwisko
HAVING COUNT(w.id) > 2;
```

**Wyjaśnienie:** HAVING filtruje grupy po agregacji. WHERE działa przed grupowaniem i nie może zawierać COUNT().

## Zapytanie 5

Wypożyczeniu o identyfikatorze 5 ustaw datę zwrotu na dzień bieżący.

### SQL

```
UPDATE wypozyczenia
SET data_zwr = CURDATE()
WHERE id = 5;
```

**Wyjaśnienie:** CURDATE() zwraca bieżącą datę, NOW() — bieżącą datę i godzinę.

## Zestaw 7. Przychodnia lekarska

### Baza danych: przychodnia

Baza przychodni gromadzi dane pacjentów, lekarzy i umówionych wizyt.

| Tabela: pacjenci |             |                        |
|------------------|-------------|------------------------|
| Kolumna          | Typ         | Opis                   |
| id               | INT, PK     | identyfikator pacjenta |
| imie             | VARCHAR(30) | imię                   |
| nazwisko         | VARCHAR(50) | nazwisko               |
| pesel            | CHAR(11)    | numer PESEL            |
| miasto           | VARCHAR(50) | miejsce zamieszkania   |

| Tabela: lekarze |             |                       |
|-----------------|-------------|-----------------------|
| Kolumna         | Typ         | Opis                  |
| id              | INT, PK     | identyfikator lekarza |
| imie            | VARCHAR(30) | imię                  |
| nazwisko        | VARCHAR(50) | nazwisko              |
| specjalizacja   | VARCHAR(50) | specjalizacja         |

| Tabela: wizyty |         |                          |
|----------------|---------|--------------------------|
| Kolumna        | Typ     | Opis                     |
| id             | INT, PK | identyfikator wizyty     |
| pacjent_id     | INT, FK | odwołanie do pacjenci.id |
| lekarz_id      | INT, FK | odwołanie do lekarze.id  |
| data_wizyty    | DATE    | data                     |
| godzina        | TIME    | godzina                  |

### Zapytanie 1

Wybierz imiona i nazwiska lekarzy o specjalizacji kardiolog.

#### SQL

```
SELECT imie, nazwisko
FROM lekarze
WHERE specjalizacja = 'kardiolog';
```

**Wyjaśnienie:** Proste zapytanie z warunkiem tekstowym.

### Zapytanie 2

Wybierz wszystkie wizyty zaplanowane w marcu 2024 r.

#### SQL

```
SELECT *
FROM wizyty
WHERE data_wizyty BETWEEN '2024-03-01' AND '2024-03-31';
```

**Wyjaśnienie:** BETWEEN obejmuje obie daty graniczne. Alternatywa: WHERE MONTH(data\_wizyty) = 3 AND YEAR(data\_wizyty) = 2024.

### Zapytanie 3

Wyświetl nazwisko pacjenta, nazwisko lekarza i datę wizyty dla wszystkich wizyt, posortowane według daty.

#### SQL

```
SELECT p.nazwisko AS pacjent, l.nazwisko AS lekarz, w.data_wizyty
FROM wizyty w
JOIN pacjenci p ON w.pacjent_id = p.id
JOIN lekarze l ON w.lekarz_id = l.id
ORDER BY w.data_wizyty;
```

**Wyjaśnienie:** Złączenie trzech tabel — tabela wizyty łączy się z obiema pozostałymi. Aliasy kolumn odróżniają dwa nazwiska.

### Zapytanie 4

Policz pacjentów z każdego miasta.

#### SQL

```
SELECT miasto, COUNT(*) AS liczba_pacjentow
FROM pacjenci
GROUP BY miasto;
```

**Wyjaśnienie:** Grupowanie po kolumnie miasto.

### Zapytanie 5

Wybierz dane pacjentów, których nazwisko zaczyna się na literę K.

#### SQL

```
SELECT *
FROM pacjenci
WHERE nazwisko LIKE 'K%';
```

**Wyjaśnienie:** W LIKE znak % oznacza dowolny ciąg znaków (także pusty), a \_ dokładnie jeden znak.

## Zestaw 8. Kino

### Baza danych: kino

Baza kina przechowuje repertuar filmów wraz z gatunkami.

| Tabela: gatunki |             |                       |
|-----------------|-------------|-----------------------|
| Kolumna         | Typ         | Opis                  |
| id              | INT, PK     | identyfikator gatunku |
| nazwa           | VARCHAR(30) | nazwa gatunku         |

| Tabela: filmy |              |                         |
|---------------|--------------|-------------------------|
| Kolumna       | Typ          | Opis                    |
| id            | INT, PK      | identyfikator filmu     |
| tytuł         | VARCHAR(100) | tytuł                   |
| gatunek_id    | INT, FK      | odwołanie do gatunki.id |
| rok_produkcji | INT          | rok produkcji           |
| czas_trwania  | INT          | czas w minutach         |
| ocena         | DECIMAL(3,1) | ocena widzów            |

### Zapytanie 1

Wybierz tytuły filmów, które zawierają w tytule słowo miłość.

#### SQL

```
SELECT tytuł
FROM filmy
WHERE tytuł LIKE '%miłość%';
```

**Wyjaśnienie:** % po obu stronach oznacza, że słowo może wystąpić w dowolnym miejscu tytułu.

### Zapytanie 2

Wybierz tytuły i oceny trzech najwyżej ocenianych filmów.

#### SQL

```
SELECT tytuł, ocena
FROM filmy
ORDER BY ocena DESC
LIMIT 3;
```

**Wyjaśnienie:** LIMIT ogranicza liczbę zwróconych wierszy — najpierw sortujemy, potem obcinamy.

### Zapytanie 3

Wybierz tytuł i czas trwania najdłuższego filmu.

#### SQL

```
SELECT tytuł, czas_trwania
FROM filmy
WHERE czas_trwania = (SELECT MAX(czas_trwania) FROM filmy);
```

**Wyjaśnienie:** Podzapytanie zwraca maksymalny czas, a zapytanie główne szuka filmu z tą wartością.

### Zapytanie 4

Wyświetl listę lat produkcji filmów bez powtórzeń, posortowaną rosnąco.

**SQL**

```
SELECT DISTINCT rok_produkcji
FROM filmy
ORDER BY rok_produkcji;
```

**Wyjaśnienie:** DISTINCT eliminuje powtarzające się wartości.

### Zapytanie 5

Dla każdego gatunku wyświetl jego nazwę i liczbę filmów (alias ilosc\_filmow).

**SQL**

```
SELECT g.nazwa, COUNT(f.id) AS ilosc_filmow
FROM gatunki g
JOIN filmy f ON f.gatunek_id = g.id
GROUP BY g.nazwa;
```

**Wyjaśnienie:** Złączenie z grupowaniem — typowe zadanie łączące JOIN i funkcję agregującą.

## Zestaw 9. Hurtownia

### Baza danych: hurtownia

Baza hurtowni przechowuje produkty przypisane do kategorii wraz ze stanem magazynowym.

| Tabela: <b>kategorie</b> |             |                         |
|--------------------------|-------------|-------------------------|
| Kolumna                  | Typ         | Opis                    |
| id                       | INT, PK     | identyfikator kategorii |
| nazwa                    | VARCHAR(50) | nazwa kategorii         |

| Tabela: <b>produkty</b> |              |                           |
|-------------------------|--------------|---------------------------|
| Kolumna                 | Typ          | Opis                      |
| id                      | INT, PK      | identyfikator produktu    |
| nazwa                   | VARCHAR(50)  | nazwa produktu            |
| kategoria_id            | INT, FK      | odwołanie do kategorie.id |
| cena                    | DECIMAL(8,2) | cena jednostkowa          |
| ilosc                   | INT          | liczba sztuk w magazynie  |

### Zapytanie 1

Podnieś o 10% ceny wszystkich produktów z kategorii o identyfikatorze 2.

#### SQL

```
UPDATE produkty
SET cena = cena * 1.1
WHERE kategoria_id = 2;
```

**Wyjaśnienie:** Nowa wartość może być wyliczana na podstawie starej wartości tej samej kolumny.

### Zapytanie 2

Oblicz łączną wartość towaru w magazynie (cena × ilość). Kolumnę nazwij wartosc\_magazynu.

#### SQL

```
SELECT SUM(cena * ilosc) AS wartosc_magazynu
FROM produkty;
```

**Wyjaśnienie:** SUM() sumuje wyrażenie obliczone dla każdego wiersza.

### Zapytanie 3

Wyświetl cenę najtańszego i najdroższego produktu.

#### SQL

```
SELECT MIN(cena) AS najtanszy, MAX(cena) AS najdrozszy
FROM produkty;
```

**Wyjaśnienie:** W jednym zapytaniu można użyć kilku funkcji agregujących.

### Zapytanie 4

Wybierz nazwy i ceny produktów należących do kategorii 1, 3 lub 5.

**SQL**

```
SELECT nazwa, cena  
FROM produkty  
WHERE kategoria_id IN (1, 3, 5);
```

**Wyjaśnienie:** Równoważnie: WHERE kategoria\_id = 1 OR kategoria\_id = 3 OR kategoria\_id = 5.

### Zapytanie 5

Dodaj do tabeli produkty kolumnę opis typu tekstowego.

**SQL**

```
ALTER TABLE produkty  
ADD COLUMN opis TEXT;
```

**Wyjaśnienie:** Słowo COLUMN jest opcjonalne.

### Zapytanie 6

Zmień typ kolumny nazwa tak, aby mogła przechowywać do 100 znaków, a następnie usuń kolumnę opis.

**SQL**

```
ALTER TABLE produkty MODIFY nazwa VARCHAR(100);  
ALTER TABLE produkty DROP COLUMN opis;
```

**Wyjaśnienie:** MODIFY zmienia typ kolumny, CHANGE zmienia nazwę i typ, DROP COLUMN usuwa kolumnę.

## Zestaw 10. Portal ogłoszeniowy

### Baza danych: ogłoszenia

Baza portalu przechowuje dane zarejestrowanych użytkowników oraz dodane przez nich ogłoszenia.

| Tabela: <b>uzytkownik</b> |              |                           |
|---------------------------|--------------|---------------------------|
| Kolumna                   | Typ          | Opis                      |
| id                        | INT, PK      | identyfikator użytkownika |
| imie                      | VARCHAR(30)  | imię                      |
| nazwisko                  | VARCHAR(50)  | nazwisko                  |
| telefon                   | VARCHAR(15)  | telefon                   |
| email                     | VARCHAR(100) | adres e-mail              |

| Tabela: <b>ogloszenie</b> |              |                            |
|---------------------------|--------------|----------------------------|
| Kolumna                   | Typ          | Opis                       |
| id                        | INT, PK      | identyfikator ogłoszenia   |
| uzytkownik_id             | INT, FK      | odwołanie do uzytkownik.id |
| kategoria                 | INT          | numer kategorii            |
| tytul                     | VARCHAR(100) | tytuł ogłoszenia           |
| tresc                     | TEXT         | treść ogłoszenia           |

### Zapytanie 1

Wybierz identyfikator, tytuł i treść ogłoszeń z kategorii 1.

#### SQL

```
SELECT id, tytul, tresc
FROM ogloszenie
WHERE kategoria = 1;
```

**Wyjaśnienie:** Proste zapytanie wybierające.

### Zapytanie 2

Wybierz numer telefonu użytkownika, który dodał ogłoszenie o identyfikatorze 10.

#### SQL

```
SELECT u.telefon
FROM uzytkownik u
JOIN ogloszenie o ON u.id = o.uzytkownik_id
WHERE o.id = 10;
```

**Wyjaśnienie:** Warunek dotyczy ogłoszenia, a wynik — użytkownika, stąd złączenie.

### Zapytanie 3

Utwórz użytkownika moderator na serwerze localhost z hasłem qwerty.

#### SQL

```
CREATE USER 'moderator'@'localhost' IDENTIFIED BY 'qwerty';
```

**Wyjaśnienie:** Polecenie z grupy DCL (zarządzanie dostępem).

### Zapytanie 4

Nadaj użytkownikowi moderator prawo przeglądania i usuwania rekordów z tabeli ogłoszenie.

**SQL**

```
GRANT SELECT, DELETE ON ogłoszenia.ogloszenie TO 'moderator'@'localhost';
```

**Wyjaśnienie:** Wymieniamy tylko te uprawnienia, o które prosi zadanie — nadanie ALL PRIVILEGES byłoby błędem.

### Zapytanie 5

Odbierz użytkownikowi moderator prawo usuwania rekordów z tabeli ogłoszenie.

**SQL**

```
REVOKE DELETE ON ogłoszenia.ogloszenie FROM 'moderator'@'localhost';
```

**Wyjaśnienie:** GRANT ... TO nadaje uprawnienia, REVOKE ... FROM je odbiera.

## Zestaw 11. Dziennik ocen

### Baza danych: szkoła

Baza dziennika elektronicznego przechowuje klasy, uczniów oraz wystawione oceny.

| Tabela: klasy |             |                     |
|---------------|-------------|---------------------|
| Kolumna       | Typ         | Opis                |
| id            | INT, PK     | identyfikator klasy |
| nazwa         | VARCHAR(10) | np. 3TI             |

| Tabela: uczniowie |             |                       |
|-------------------|-------------|-----------------------|
| Kolumna           | Typ         | Opis                  |
| id                | INT, PK     | identyfikator ucznia  |
| imie              | VARCHAR(30) | imię                  |
| nazwisko          | VARCHAR(50) | nazwisko              |
| klasa_id          | INT, FK     | odwołanie do klasy.id |

| Tabela: oceny |             |                           |
|---------------|-------------|---------------------------|
| Kolumna       | Typ         | Opis                      |
| id            | INT, PK     | identyfikator oceny       |
| uczen_id      | INT, FK     | odwołanie do uczniowie.id |
| przedmiot     | VARCHAR(30) | nazwa przedmiotu          |
| ocena         | INT         | ocena 1–6                 |
| data          | DATE        | data wystawienia          |

### Zapytanie 1

Dla każdego ucznia wyświetl nazwisko i średnią ocen zaokrągloną do dwóch miejsc po przecinku. Posortuj malejąco według średniej.

#### SQL

```
SELECT u.nazwisko, ROUND(AVG(o.ocena), 2) AS srednia
FROM uczniowie u
JOIN oceny o ON u.id = o.uczen_id
GROUP BY u.id, u.nazwisko
ORDER BY srednia DESC;
```

**Wyjaśnienie:** Grupujemy po id, aby uczniowie o tym samym nazwisku nie zostali połączeni w jedną grupę.

### Zapytanie 2

Wyświetl nazwiska uczniów, których średnia ocen jest wyższa niż 4,5.

#### SQL

```
SELECT u.nazwisko, AVG(o.ocena) AS srednia
FROM uczniowie u
JOIN oceny o ON u.id = o.uczen_id
GROUP BY u.id, u.nazwisko
HAVING AVG(o.ocena) > 4.5;
```

**Wyjaśnienie:** W SQL separatorem dziesiętnym jest kropka, nie przecinek.

### Zapytanie 3

Wyświetl nazwę każdej klasy i liczbę jej uczniów — również dla klas, do których nie przypisano jeszcze żadnego ucznia.

**SQL**

```
SELECT k.nazwa, COUNT(u.id) AS liczba_uczniow
FROM klasy k
LEFT JOIN uczniowie u ON u.klasa_id = k.id
GROUP BY k.id, k.nazwa;
```

**Wyjaśnienie:** LEFT JOIN zwraca wszystkie rekordy z lewej tabeli, nawet bez dopasowania. COUNT(u.id) nie liczy wartości NULL, więc pusta klasa da 0.

### Zapytanie 4

Wybierz wszystkie oceny niedostateczne z matematyki.

**SQL**

```
SELECT *
FROM oceny
WHERE ocena = 1 AND przedmiot = 'matematyka';
```

**Wyjaśnienie:** Dwa warunki połączone operatorem AND.

### Zapytanie 5

Wybierz imiona i nazwiska uczniów, którzy nie mają jeszcze żadnej oceny.

**SQL**

```
SELECT imie, nazwisko
FROM uczniowie
WHERE id NOT IN (SELECT uczen_id FROM oceny);
```

**Wyjaśnienie:** Podzapytanie zwraca uczniów z ocenami; NOT IN wybiera pozostałych. Alternatywa: LEFT JOIN oceny ... WHERE oceny.id IS NULL.

## Zestaw 12. Tworzenie bazy i administrowanie

### Baza danych: sklep

Zestaw poleceń DDL i DCL: tworzenie bazy i tabel, klucze obce oraz zarządzanie użytkownikami. Takie polecenia pojawiają się zarówno w części praktycznej, jak i w teście.

#### Zapytanie 1

Utwórz bazę danych sklep z kodowaniem utf8mb4 i porównywaniem znaków dla języka polskiego.

##### SQL

```
CREATE DATABASE sklep
CHARACTER SET utf8mb4
COLLATE utf8mb4_polish_ci;
```

**Wyjaśnienie:** utf8mb4 poprawnie zapisuje polskie znaki diakrytyczne; \_polish\_ci sortuje zgodnie z polskim alfabetem.

#### Zapytanie 2

Utwórz tabelę klienci z kolumnami: id (liczba całkowita, klucz główny, auto-numeracja), imie (do 30 znaków, wymagane), nazwisko (do 50 znaków, wymagane), email (do 100 znaków, unikalny).

##### SQL

```
CREATE TABLE klienci (
  id INT AUTO_INCREMENT PRIMARY KEY,
  imie VARCHAR(30) NOT NULL,
  nazwisko VARCHAR(50) NOT NULL,
  email VARCHAR(100) UNIQUE
);
```

**Wyjaśnienie:** NOT NULL wymusza podanie wartości, UNIQUE blokuje powtórzenia.

#### Zapytanie 3

Utwórz tabelę zamówienia (id, klient\_id, data, kwota) z kluczem obcym klient\_id wskazującym na tabelę klienci.

##### SQL

```
CREATE TABLE zamówienia (
  id INT AUTO_INCREMENT PRIMARY KEY,
  klient_id INT NOT NULL,
  data DATE,
  kwota DECIMAL(10,2),
  FOREIGN KEY (klient_id) REFERENCES klienci(id)
);
```

**Wyjaśnienie:** Klucz obcy zapewnia integralność referencyjną — nie da się dodać zamówienia dla nieistniejącego klienta.

#### Zapytanie 4

Dodaj klucz obcy do istniejącej już tabeli zamówienia (gdyby nie został utworzony przy CREATE TABLE).

##### SQL

```
ALTER TABLE zamówienia
ADD FOREIGN KEY (klient_id) REFERENCES klienci(id);
```

**Wyjaśnienie:** Klucz obcy można dodać także po utworzeniu tabeli.

### Zapytanie 5

Utwórz użytkownika admin z hasłem Adm1n! i nadaj mu wszystkie uprawnienia do bazy sklep.

#### SQL

```
CREATE USER 'admin'@'localhost' IDENTIFIED BY 'Adm1n!';  
GRANT ALL PRIVILEGES ON sklep.* TO 'admin'@'localhost';
```

**Wyjaśnienie:** sklep.\* oznacza wszystkie tabele w bazie sklep; \*.\* oznaczałoby wszystkie bazy na serwerze.

### Zapytanie 6

Wyświetl uprawnienia użytkownika admin.

#### SQL

```
SHOW GRANTS FOR 'admin'@'localhost';
```

**Wyjaśnienie:** Polecenie pozwala sprawdzić, czy uprawnienia zostały nadane poprawnie.

### Zapytanie 7

Usuń wszystkie rekordy z tabeli zamowienia, zachowując jej strukturę, a następnie usuń użytkownika admin.

#### SQL

```
TRUNCATE TABLE zamowienia;  
DROP USER 'admin'@'localhost';
```

**Wyjaśnienie:** TRUNCATE czyści tabelę i zeruje licznik AUTO\_INCREMENT; DROP TABLE usunąłby tabelę razem ze strukturą.

## Część II — Pytania testowe z SQL (część pisemna)

Pytania jednokrotnego wyboru w formie stosowanej w części pisemnej egzaminu. Poprawna odpowiedź jest zaznaczona pod każdym pytaniem wraz z krótkim uzasadnieniem.

**Pytanie 1.** Która klauzula służy do filtrowania grup utworzonych przez GROUP BY?

- A. WHERE
- B. HAVING
- C. ORDER BY
- D. LIMIT

**Odpowiedź: B** — HAVING działa po grupowaniu i może zawierać funkcje agregujące; WHERE filtruje pojedyncze rekordy przed grupowaniem.

**Pytanie 2.** Aby posortować wyniki zapytania, należy użyć klauzuli:

- A. GROUP BY
- B. SORT
- C. ORDER BY
- D. ARRANGE

**Odpowiedź: C** — ORDER BY sortuje wyniki; domyślnie rosnąco (ASC).

**Pytanie 3.** Które polecenie usuwa tabelę wraz z jej strukturą?

- A. DELETE FROM
- B. TRUNCATE TABLE
- C. DROP TABLE
- D. REMOVE TABLE

**Odpowiedź: C** — DROP TABLE usuwa całą tabelę. DELETE i TRUNCATE usuwają tylko dane.

**Pytanie 4.** Słowo kluczowe eliminujące powtarzające się wiersze w wyniku to:

- A. UNIQUE
- B. DISTINCT
- C. SINGLE
- D. DIFFERENT

**Odpowiedź: B** — SELECT DISTINCT zwraca tylko unikalne wiersze. UNIQUE to ograniczenie kolumny w CREATE TABLE.

**Pytanie 5.** Warunek WHERE nazwisko LIKE 'A%' wybierze nazwiska:

- A. kończące się na A
- B. zawierające literę A
- C. zaczynające się na A
- D. składające się z dwóch znaków

**Odpowiedź: C** — % oznacza dowolny ciąg znaków po literze A.

**Pytanie 6. W operatorze LIKE znak podkreślenia \_ oznacza:**

- A. dowolny ciąg znaków
- B. dokładnie jeden dowolny znak
- C. spację
- D. brak znaku

**Odpowiedź: B** — Np. LIKE '\_a%' wybierze wartości, w których druga litera to a.

**Pytanie 7. Funkcja agregująca zliczająca rekordy to:**

- A. SUM()
- B. COUNT()
- C. NUMBER()
- D. TOTAL()

**Odpowiedź: B** — COUNT(\*) liczy wszystkie wiersze, COUNT(kolumna) pomija wartości NULL.

**Pytanie 8. Klucz obcy (FOREIGN KEY) służy do:**

- A. szyfrowania danych
- B. zapewnienia integralności referencyjnej między tabelami
- C. automatycznej numeracji rekordów
- D. przyspieszenia sortowania

**Odpowiedź: B** — Klucz obcy musi wskazywać na istniejący klucz w tabeli powiązanej.

**Pytanie 9. Klucz podstawowy (PRIMARY KEY) musi być:**

- A. unikalny i nie może przyjmować wartości NULL
- B. typu tekstowego
- C. unikalny, może przyjmować NULL
- D. zawsze typu AUTO\_INCREMENT

**Odpowiedź: A** — PRIMARY KEY jednoznacznie identyfikuje rekord, więc musi być unikalny i niepusty.

**Pytanie 10. Atrybut AUTO\_INCREMENT powoduje, że:**

- A. kolumna przyjmuje wartości losowe
- B. wartość kolumny jest automatycznie zwiększana dla każdego nowego rekordu
- C. kolumna jest kopiowana do innej tabeli
- D. rekord jest automatycznie usuwany

**Odpowiedź: B** — Stosowany zwykle dla kolumny id będącej kluczem głównym.

**Pytanie 11. Które polecenie należy do języka DCL (Data Control Language)?**

- A. SELECT
- B. CREATE
- C. GRANT
- D. UPDATE

**Odpowiedź: C** — DCL: GRANT, REVOKE. DDL: CREATE, ALTER, DROP. DML: INSERT, UPDATE, DELETE (i SELECT).

**Pytanie 12.** Które polecenie należy do języka DDL (Data Definition Language)?

- A. INSERT
- B. ALTER
- C. DELETE
- D. REVOKE

**Odpowiedź: B** — DDL definiuje strukturę bazy: CREATE, ALTER, DROP, TRUNCATE.

**Pytanie 13.** Polecenie odbierające użytkownikowi uprawnienia to:

- A. DENY
- B. REMOVE
- C. REVOKE
- D. DROP GRANT

**Odpowiedź: C** — Składnia: REVOKE uprawnienie ON baza.tabela FROM 'user'@'host';

**Pytanie 14.** Złączenie LEFT JOIN zwraca:

- A. tylko rekordy mające dopasowanie w obu tabelach
- B. wszystkie rekordy lewej tabeli oraz dopasowane rekordy prawej
- C. wszystkie rekordy prawej tabeli
- D. iloczyn kartezjański tabel

**Odpowiedź: B** — Brak dopasowania w prawej tabeli daje wartości NULL w jej kolumnach.

**Pytanie 15.** Aby sprawdzić, czy kolumna nie zawiera wartości, należy użyć warunku:

- A. kolumna = NULL
- B. kolumna IS NULL
- C. kolumna == ''
- D. kolumna EMPTY

**Odpowiedź: B** — NULL nie jest wartością, więc nie można go porównać operatorem =.

**Pytanie 16.** Warunek WHERE cena BETWEEN 10 AND 20 wybierze ceny:

- A. większe od 10 i mniejsze od 20
- B. od 10 do 20 włącznie
- C. tylko 10 i 20
- D. mniejsze od 10 lub większe od 20

**Odpowiedź: B** — BETWEEN obejmuje wartości graniczne.

**Pytanie 17.** Polecenie UPDATE produkty SET cena = 0; (bez klauzuli WHERE) spowoduje:

- A. błąd składni
- B. zmianę ceny tylko pierwszego rekordu
- C. zmianę ceny we wszystkich rekordach
- D. usunięcie kolumny cena

**Odpowiedź: C** — Brak WHERE oznacza, że operacja dotyczy całej tabeli.

**Pytanie 18.** Do dodania nowej kolumny w istniejącej tabeli służy polecenie:

- A. UPDATE TABLE ... ADD
- B. ALTER TABLE ... ADD
- C. INSERT COLUMN
- D. MODIFY TABLE ... NEW

**Odpowiedź: B** — ALTER TABLE nazwa ADD kolumna typ;

**Pytanie 19.** Relacja wiele-do-wielu między dwiema tabelami wymaga:

- A. jednego klucza obcego
- B. tabeli pośredniczącej
- C. indeksu pełnotekstowego
- D. widoku

**Odpowiedź: B** — Np. tabela wypożyczenia łączy czytelników z książkami.

**Pytanie 20.** Pierwsza postać normalna (1NF) wymaga, aby:

- A. każda tabela miała klucz obcy
- B. wartości w kolumnach były atomowe (niepodzielne)
- C. tabela miała co najmniej trzy kolumny
- D. nie było indeksów

**Odpowiedź: B** — Jedna komórka = jedna wartość, bez list i powtarzających się grup.

**Pytanie 21.** Jaki typ danych jest najwłaściwszy do przechowywania ceny produktu?

- A. VARCHAR
- B. DECIMAL
- C. DATE
- D. BOOLEAN

**Odpowiedź: B** — DECIMAL przechowuje liczby dokładnie, bez błędów zaokrągleń typowych dla FLOAT.

**Pytanie 22.** Funkcja AVG() zwraca:

- A. największą wartość
- B. sumę wartości
- C. średnią arytmetyczną
- D. liczbę wartości

**Odpowiedź: C** — MAX – największa, SUM – suma, COUNT – liczba.

**Pytanie 23. Polecenia COMMIT i ROLLBACK są związane z:**

- A. tworzeniem kopii zapasowej
- B. transakcjami
- C. indeksami
- D. uprawnieniami

**Odpowiedź: B** — COMMIT zatwierdza transakcję, ROLLBACK ją wycofuje.

**Pytanie 24. Polecenie CREATE VIEW służy do utworzenia:**

- A. nowej bazy danych
- B. widoku (wirtualnej tabeli opartej na zapytaniu)
- C. użytkownika
- D. indeksu

**Odpowiedź: B** — Widok przechowuje zapytanie, a nie dane.

**Pytanie 25. Indeks w tabeli bazy danych służy przede wszystkim do:**

- A. przyspieszenia wyszukiwania danych
- B. szyfrowania rekordów
- C. usuwania duplikatów
- D. nadawania uprawnień

**Odpowiedź: A** — Indeks przyspiesza SELECT, ale spowalnia nieco INSERT i UPDATE.

**Pytanie 26. Najskuteczniejszą ochroną przed atakiem SQL Injection w PHP jest:**

- A. używanie metody GET
- B. zapytania przygotowane (prepared statements)
- C. wyłączenie JavaScript
- D. szyfrowanie hasła MD5

**Odpowiedź: B** — Parametry (?) są przekazywane oddzielnie od treści zapytania, więc nie mogą zmienić jego struktury.

**Pytanie 27. Funkcja PHP mysqli\_num\_rows() zwraca:**

- A. liczbę kolumn wyniku
- B. liczbę wierszy w wyniku zapytania SELECT
- C. identyfikator ostatniego rekordu
- D. liczbę baz danych

**Odpowiedź: B** — Liczba kolumn to mysqli\_num\_fields(), ostatnie id to mysqli\_insert\_id().

**Pytanie 28.** Funkcja PHP `mysqli_fetch_row()` zwraca wiersz wyniku jako:

- A. tablicę asocjacyjną
- B. tablicę indeksowaną numerycznie
- C. obiekt
- D. ciąg znaków

**Odpowiedź: B** — `mysqli_fetch_assoc()` – tablica asocjacyjna, `mysqli_fetch_array()` – obie formy naraz.

**Pytanie 29.** Zapis `'jan'@'localhost'` w poleceniu `CREATE USER` oznacza użytkownika jan:

- A. logującego się z dowolnego komputera
- B. logującego się tylko z komputera lokalnego
- C. bez hasła
- D. z uprawnieniami administratora

**Odpowiedź: B** — `'jan'@'%'` oznaczałby dostęp z dowolnego hosta.

**Pytanie 30.** Domyślny kierunek sortowania w klauzuli `ORDER BY` to:

- A. DESC
- B. ASC
- C. RAND
- D. brak sortowania

**Odpowiedź: B** — ASC – rosnąco (A→Z, 1→9).

## Część III — Ściąga: najważniejsze polecenia SQL

Zestawienie składni wykorzystanej w zadaniach. Warto nauczyć się jej na pamięć przed egzaminem.

| Polecenie / klauzula                       | Zastosowanie                                             | Przykład                                             |
|--------------------------------------------|----------------------------------------------------------|------------------------------------------------------|
| <b>SELECT ... FROM ... WHERE</b>           | wybieranie danych z warunkiem                            | SELECT nazwa FROM ryby WHERE id = 1;                 |
| <b>ORDER BY ... ASC/DESC</b>               | sortowanie wyników                                       | ORDER BY cena DESC                                   |
| <b>LIMIT n</b>                             | ograniczenie liczby wierszy                              | LIMIT 3                                              |
| <b>DISTINCT</b>                            | usunięcie duplikatów                                     | SELECT DISTINCT miasto FROM klienci;                 |
| <b>LIKE, %, _</b>                          | wyszukiwanie wzorca                                      | WHERE tytuł LIKE '%kot%'                             |
| <b>BETWEEN ... AND</b>                     | zakres wartości (włącznie)                               | WHERE rok BETWEEN 2000 AND 2010                      |
| <b>IN (...)</b>                            | wartość z listy                                          | WHERE id IN (1, 3, 5)                                |
| <b>IS NULL / IS NOT NULL</b>               | brak wartości                                            | WHERE data_zwr IS NULL                               |
| <b>COUNT, SUM, AVG, MIN, MAX</b>           | funkcje agregujące                                       | SELECT AVG(cena) FROM produkty;                      |
| <b>GROUP BY ... HAVING</b>                 | grupowanie i filtrowanie grup                            | GROUP BY marka HAVING COUNT(*) > 2                   |
| <b>JOIN ... ON</b>                         | złączenie tabel                                          | JOIN autorzy ON książki.id_autor = autorzy.id        |
| <b>LEFT JOIN</b>                           | złączenie z zachowaniem wszystkich rekordów lewej tabeli | FROM klasy LEFT JOIN uczniowie ON ...                |
| <b>INSERT INTO ... VALUES</b>              | dodanie rekordu                                          | INSERT INTO t (a, b) VALUES (1, 'x');                |
| <b>UPDATE ... SET ... WHERE</b>            | modyfikacja rekordów                                     | UPDATE t SET cena = 10 WHERE id = 2;                 |
| <b>DELETE FROM ... WHERE</b>               | usunięcie rekordów                                       | DELETE FROM t WHERE id = 2;                          |
| <b>CREATE DATABASE / TABLE</b>             | tworzenie bazy / tabeli                                  | CREATE TABLE t (id INT PRIMARY KEY);                 |
| <b>ALTER TABLE ... ADD / MODIFY / DROP</b> | zmiana struktury tabeli                                  | ALTER TABLE t ADD opis TEXT;                         |
| <b>DROP TABLE / TRUNCATE TABLE</b>         | usunięcie tabeli / wyczyszczenie danych                  | TRUNCATE TABLE t;                                    |
| <b>CREATE USER</b>                         | utworzenie użytkownika                                   | CREATE USER 'jan'@'localhost' IDENTIFIED BY 'hasło'; |
| <b>GRANT ... ON ... TO</b>                 | nadanie uprawnień                                        | GRANT SELECT ON baza.* TO 'jan'@'localhost';         |
| <b>REVOKE ... ON ... FROM</b>              | odebranie uprawnień                                      | REVOKE SELECT ON baza.* FROM 'jan'@'localhost';      |

### Kolejność klauzul w zapytaniu SELECT

**SELECT → FROM → JOIN ... ON → WHERE → GROUP BY → HAVING → ORDER BY → LIMIT**

Zamiana kolejności (np. WHERE po GROUP BY albo ORDER BY przed WHERE) powoduje błąd składni.