SQL

PRAKTYCZNY PRZEWODNIK

Java + MySQL w praktyce

50 map myśli



Od modelu danych do bezpiecznych transakcji

SQL • JDBC • TRANSAKCJE • TESTY

JAVA + MYSQL W PRAKTYCE

SQL

SPIS TREŚCI

Numery odnoszą się do map 01–50.



01 Fundamenty i model danych

01 Jak Java komunikuje się z MySQL
02 Środowisko pracy i konfiguracja projektu
03 Od wymagań do modelu danych
04 Relacje i normalizacja
05 Typy danych w MySQL

02 Odczyt danych w SQL

06 SELECT i aliasy
07 WHERE i logika NULL
08 ORDER BY i LIMIT
09 INNER JOIN i LEFT JOIN
10 GROUP BY, HAVING i agregacje

03 Zapis danych i spójność

11 CREATE TABLE i ograniczenia
12 INSERT i AUTO_INCREMENT
13 UPDATE z kontrolą zakresu
14 DELETE i reguły usuwania
15 ALTER TABLE i migracje

04 Zaawansowane zapytania i indeksy

16 Podzapytania i EXISTS
17 CTE: zapytania wieloetapowe
18 Funkcje okna
19 Indeksy pojedyncze i złożone
20 EXPLAIN i EXPLAIN ANALYZE

05 JDBC krok po kroku

21 Connection i DataSource
22 PreparedStatement i parametry
23 executeQuery, executeUpdate i execute
24 ResultSet
25 try-with-resources

SQL

SPIS TREŚCI

Numery odnoszą się do map 01–50.



06 Mapowanie danych i warstwa dostępu

26 | SQL NULL a Java null
27 | DECIMAL i BigDecimal
28 | Daty, czas i strefy czasowe
29 | Klucze generowane i operacje batch
30 | DAO, repozytorium i warstwa usług

07 Transakcje i współbieżność

31 | ACID na przykładzie zamówienia
32 | Auto-commit, commit i rollback
33 | Jedna transakcja, jedno połączenie
34 | Poziomy izolacji i anomalie
35 | Blokady i deadlocki

08 Bezpieczeństwo i wydajność aplikacji

36 | SQL injection i dynamiczne zapytania
37 | Uprawnienia, sekrety i TLS
38 | Pula połączeń i timeouty
39 | Stronicowanie i duże zbiory danych
40 | Problem N+1 i pomiar wydajności

09 Testowanie i diagnostyka

41 | SQLException i SQLState
42 | Logowanie bez ujawniania danych
43 | Testy integracyjne z MySQL i Testcontainers
44 | Testowanie ograniczeń i rollbacku
45 | Dane brzegowe i powtarzalność testów

10 Projekt praktyczny: system zamówień

46 | Schemat bazy i reguły biznesowe
47 | Wyszukiwanie produktów
48 | Zapis zamówienia i jego pozycji
49 | Aktualizacja stanu magazynowego
50 | Test całego procesu i checklista wdrożenia

SQL

JAK JAVA KOMUNIKUJE SIĘ Z MYSQL

JAVA + MYSQL W PRAKTYCE

Prześledź drogę jednego zapytania: od metody w Javie do wiersza wyniku.

01 1. Java: intencja

Aplikacja chce znaleźć produkt. Tworzy tekst SQL i przekazuje wartości parametrów. MySQL nie wykonuje kodu Java; otrzymuje polecenia przez sterownik.



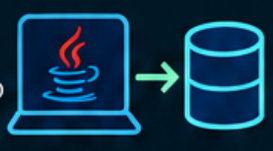
02 2. JDBC: wspólny kontrakt

Interfejsy Connection, PreparedStatement i ResultSet należą do java.sql. Kod korzysta z tego API, a Connector/J dostarcza implementację dla MySQL.



03 3. Połączenie: sesja

Connection reprezentuje sesję z bazą. Adres jdbc:mysql://localhost:3306/shop określa protokół JDBC, host, port i bazę; login ma własne uprawnienia.



jdbc:mysql://
localhost:3306/shop

04 4. Parametry: osobno

Znak ? oznacza wartość, a numeracja zaczyna się od 1. Fragment zakłada otwarte połączenie con; zasoby należy zamknąć przez try-with-resources.

```
var ps = con.prepareStatement(
    "SELECT name FROM products WHERE id = ?");
ps.setLong(1, 7L);
```

05 5. Serwer: wykonanie

MySQL sprawdza składnię i uprawnienia, wybiera plan oraz czyta dane. executeQuery() zwraca ResultSet; przed odczytem pierwszego wiersza wywołaj next().



06 6. Wynik albo błąd

Wynik mapujesz na wartości Javy. Brak pasującego produktu oznacza pusty wynik, nie SQLException. Błąd połączenia lub SQL wymaga osobnej obsługi.



07 Sprawdź się

Zapytanie o id = 7 nic nie znalazło. Co zwróci pierwsze rs.next()?

Odpowiedź | `false`. Nie wolno wtedy odczytywać `rs.getString("name")`.

PUŁAPKA

Nie sklejaj wartości użytkownika z SQL. PreparedStatement chroni strukturę zapytania, gdy dane wiążesz parametrami.

ZAPAMIĘTAJ

- Połączenie → parametry → wykonanie → odczyt → zamknięcie.

Potrafisz wskazać odpowiedzialność każdej warstwy.

SQL

ŚRODOWISKO PRACY i KONFIGURACJA PROJEKTU

JAVA + MYSQL W PRAKTYCE

Zbuduj powtarzalne laboratorium: Java 25, MySQL 8.4, Maven i Connector/J 26.7.0.



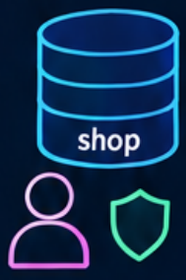
01 Sprawdź narzędzia

- Uruchom java --version i mvn --version.
- Maven powinien korzystać z JDK 25.
- W kliencie SQL sprawdź serwer poleceniem `SELECT VERSION()`.

```
$ java --version
$ mvn --version
mysql> SELECT
       VERSION();
```

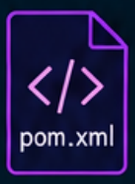
02 Odizoluj ćwiczenia

- Używaj lokalnej bazy shop z fikcyjnymi danymi.
- Utwórz osobnego użytkownika aplikacji; nie pracuj jako root i nie kopiuj sekretów produkcyjnych.



03 Dodaj sterownik

- Fragment pom.xml umieść w dependencies.
- Connector/J 26.7 obsługuje MySQL 8.4+ i JRE 8+; implementuje JDBC 4.2.




```
<dependency>
  <groupId>com.mysql</groupId>
  <artifactId>mysql-connector-j</artifactId>
  <version>26.7.0</version>
</dependency>
```

04 Ustal kompilację

- W konfiguracji Maven Compiler Plugin ustaw release na 25.
- Nie włączaj funkcji preview.
- Kod zapisuj w src/main/java, testy w src/test/java.



05 Oddziel konfigurację

- Adres jdbc:mysql://localhost:3306/shop, użytkownika i hasło wczytuj z konfiguracji lokalnej lub zmiennych środowiskowych.
- Hasła nie zapisuj w repozytorium ani logach.



06 Testuj warstwami

- Najpierw połącz się klientem SQL tym samym kontem.
- Potem uruchom w Javie `SELECT 1`.
- Sprawdź zależność także na ścieżce uruchomieniowej.



07 Sprawdź się

Klient SQL działa, Java zgłasza „No suitable driver”. Co sprawdzisz najpierw?

Odpowiedź

Obecność Connector/J w zależnościach uruchomienia i poprawny prefiks jdbc:mysql:.

PUŁAPKA

JDK 25 nie oznacza, że sterownik implementuje wszystkie nowsze metody JDBC. Sprawdzaj jego deklarowane możliwości.

ZAPAMIĘTAJ

- Wersje + izolacja + konfiguracja + prosty test.



Masz bazę ćwiczeniową i weryfikowalny punkt startu.

SQL

OD WYMAGAŃ DO MODELU DANYCH

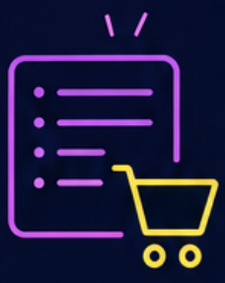
JAVA + MYSQL W PRAKTYCE

Zamień opis sklepu w tabele o jasno określonym znaczeniu każdego wiersza.



01 1. Zapisz reguły

- Klient składa wiele zamówień. Zamówienie zawiera produkty oraz ilości.
- Produkt ma bieżącą cenę i stan magazynowy.
- Cena sprzedaży musi zachować historię.



02 2. Ustal ziarnistość

Jeden wiersz customers opisuje klienta; products produkt; orders jedno zamówienie. order_items opisuje jeden produkt w jednym zamówieniu, przy przyjętej regule agregowania ilości.

- customers
- products
- orders
- order_items

03 3. Wybierz identyfikatory

- Każdy klient, produkt i zamówienie otrzymuje stabilne id.
- W order_items para (order_id, product_id) identyfikuje pozycję, jeśli produkt występuje w zamówieniu najwyżej raz.



order_id	product_id
21	7

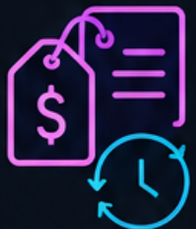
04 4. Rozdziel fakty

`customers`: id, name, email.
`products`: id, name, price, stock.
`orders`: id, customer_id, created_at.
`order_items`: order_id, product_id, quantity, unit_price.

- customers
- products
- orders
- order_items

05 5. Zachowaj historię

- `products.price` to cena aktualna.
- `order_items.unit_price` to cena uzgodniona przy zakupie. Ich późniejsza różnica jest poprawna, ponieważ opisują inne fakty.



06 6. Sprawdź scenariusz

Zamówienie 21; produkt 7, quantity = 2, unit_price = 39.90.
Wartość pozycji wynosi 79.80, nawet gdy products.price wzrośnie do 44.90.



`2 × 39.90 = 79.80`

07 Sprawdź się

Klient zmienia email. Ile miejsc powinno wymagać aktualizacji w naszym modelu?

Odpowiedź:
Jeden wiersz customers. orders wskazuje klienta przez customer_id, zamiast powielać email.

✕ PUŁAPKA

Model wymaga jawnych założeń. Jeśli ten sam produkt ma występować jako osobne pozycje z różnymi cenami, zmień klucz pozycji.

✓ ZAPAMIĘTAJ

- Najpierw znaczenie wiersza i reguły, dopiero potem CREATE TABLE.



Potrafisz wyjaśnić, dlaczego każda kolumna istnieje.

SQL

RELACJE i NORMALIZACJA

JAVA + MYSQL W PRAKTYCE

Usuń przypadkowe powtórzenia i zabezpiecz powiązania między tabelami.



01 Relacja jeden do wielu

- customers.id → orders.customer_id.
- Jeden klient może mieć wiele zamówień.
- Przy obowiązkowym przypisaniu customer_id otrzymuje NOT NULL i klucz obcy.

customers	
id	PK
name	
email	

1 →

orders	
id	PK
customer_id	FK
created_at	

02 Relacja wiele do wielu

- Zamówienie zawiera wiele produktów;
- produkt trafia do wielu zamówień.
- Tabela order_items rozbija tę relację na dwie relacje jeden do wielu.

orders	
id	PK
created_at	

products	
id	PK

1 ∞ ∞ 1

order_items	
order_id	PK, FK
product_id	PK, FK
quantity	

03 Pierwsza postać normalna

- Nie przechowuj listy identyfikatorów „7,8,9" w jednej kolumnie zamówienia.
- Każda pozycja dostaje własny wiersz, dzięki czemu można ją filtrować i łączyć.

✗ orders	
id	
1	7,8,9

Źle: lista w jednej kolumnie.

✓ order_items	
order_id	product_id
1	7
1	8
1	9

Dobrze: każda pozycja to osobny wiersz.

04 Druga postać normalna

- Przy kluczu (order_id, product_id) nazwa produktu zależy tylko od product_id.
- Umieść ją w products; quantity należy do konkretnej pary.

products	
id	PK
name	

order_items	
order_id	PK, FK
product_id	PK, FK
quantity	

05 Trzecia postać normalna

- Email klienta nie należy do orders; zależy od klienta wskazanego przez customer_id.
- Przechowywanie go w customers ogranicza niespójne kopie przy aktualizacji.

customers	
id	PK
name	
email	

orders	
id	PK
customer_id	FK
created_at	

06 Integralność w bazie

- FOREIGN KEY pilnuje istnienia wskazanego rodzica.
- Typy powiązanych identyfikatorów muszą być zgodne.
- Fragment definicji tabeli orders:

```
FOREIGN KEY (customer_id)
REFERENCES customers(id)
```

07 Sprawdź się

Czy unit_price w order_items narusza normalizację, skoro istnieje products.price?

Odpowiedź

Nie. Cena sprzedaży jest faktem dotyczącym konkretnej pozycji; cena katalogowa opisuje aktualny produkt.

✗ PUŁAPKA

Klucz obcy nie gwarantuje, że zamówienie ma choć jedną pozycję. Tę regułę realizuj w procesie zapisu zamówienia.

✓ ZAPAMIĘTAJ

- Jedna informacja ma jedno właściwe miejsce, chyba że zapisujesz odrębny fakt historyczny.



Potrafisz odróżnić duplikację od świadomego zapisu historii.

SQL

TYPY DANYCH W MySQL

Dobieraj typ do znaczenia wartości, jej zakresu i sposobu odczytu w Javie.



01 Identyfikatory i ilości

BIGINT dla id mapuje się wygodnie na long, jeśli używasz zakresu ze znakiem. **INT** wystarcza dla ilości; nieujemność stock wymaga osobnej reguły.



02 Pieniądze: dokładność

DECIMAL(10,2) przechowuje dziesięć cyfr łącznie, w tym dwie po przecinku. W Javie używaj **BigDecimal**; **FLOAT** i **DOUBLE** są przybliżone.



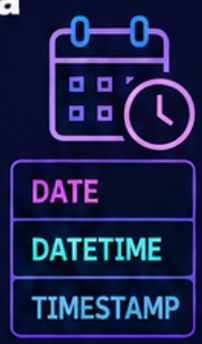
03 Tekst i Unicode

VARCHAR(120) ogranicza nazwę do 120 znaków, nie bajtów. **utf8mb4** obsługuje pełny zakres Unicode. **Collation** określa zasady porównywania, w tym rozróżnianie wielkości liter.



04 Data albo chwila

DATE przechowuje datę. **DATETIME** nie wykonuje konwersji strefy jak **TIMESTAMP**. **TIMESTAMP** jest przeliczany między strefą sesji a UTC; ma krótszy zakres dat.



05 Brak wartości

NULL oznacza brak znanej wartości; nie zastępuje zera ani pustego tekstu. **NOT NULL** wymusza obecność, **DEFAULT** dostarcza wartość przy pominięciu kolumny.



06 Cena w praktyce

Obliczenie korzysta z dokładnych literałów dziesiętnych. Wynik matematyczny wynosi 79.80; format prezentacji ustala klient.

```
SELECT CAST(39.90 AS DECIMAL(10,2)) * 2
       AS line_total;
```

07 Sprawdź się

Ile cyfr przed przecinkiem mieści DECIMAL(10,2)?

Odpowiedź

Osiem. Największa dodatnia wartość to 99999999.99.

PUŁAPKA

BIGINT UNSIGNED może przekroczyć zakres `long`. Nie wybieraj **UNSIGNED** dla kluczy automatycznie, bez sprawdzenia mapowania JDBC.

ZAPAMIĘTAJ

- Typ to kontrakt: zakres, precyzja, brak wartości i interpretacja.



Potrafisz dobrać typ ceny, ilości, identyfikatora oraz daty.

SQL

SELECT i ALIASY

JAVA + MYSQL W PRAKTYCE

Wybierz dokładnie te dane, których potrzebuje aplikacja, i nadaj im czytelne nazwy.

01 Lista wynikowa

- SELECT określa kolumny i wyrażenia wyniku.
- FROM wskazuje źródło danych.
- Kolejność kolumn wynika z listy SELECT, a nie z ich położenia w tabeli.

02 Jawne kolumny

- SELECT id, name jest stabilniejsze niż SELECT *.
- Nowa kolumna tabeli nie rozszerzy przypadkiem odpowiedzi aplikacji ani nie ujawni niepotrzebnych danych.

03 Alias kolumny

- AS nadaje etykietę tylko w wyniku zapytania.
- Nie zmienia nazwy w tabeli.
- Używaj prostych nazw angielskich, które łatwo odczytać w JDBC.

price	stock_value
39.90	119.70
12.50	25.00

04 Alias tabeli

- products AS p pozwala pisać p.id i p.name.
- Przy połączeniach tabel kwalifikowanie kolumn zapobiega niejednoznaczności nazw takich jak id.

products AS p

id
name
price
stock

05 Wyrażenie w wyniku

- Zapytanie oblicza wartość zapasu według bieżącej ceny.
- Dla price = 39.90 i stock = 3 otrzymasz 119.70.

```
SELECT p.id, p.name,
       p.price * p.stock AS stock_value
FROM products AS p;
```

06 Odczyt i duplikaty

- W JDBC użyj rs.getBigDecimal("stock_value").
- DISTINCT usuwa powtórzenia całych wybranych wierszy; nie wybiera automatycznie „najnowszego” rekordu produktu.

DISTINCT

07 Sprawdź się

Czy SELECT name AS product_name zmienia kolumnę name w tabeli products?

Odpowiedź

Nie. product_name jest etykietą kolumny wyniku tego zapytania.

PUŁAPKA

Alias z SELECT nie jest dostępny w WHERE tego samego poziomu. Użyj tam wyrażenia albo zewnętrznego zapytania.

ZAPAMIĘTAJ

- Lista SELECT to świadomy kontrakt danych zwracanych aplikacji.

Potrafisz wybrać, obliczyć i nazwać każdą kolumnę odpowiedzi.

SQL

WHERE i logika NULL

JAVA + MYSQL W PRAKTYCE

Zbuduj warunek, który wybiera dokładnie oczekiwane wiersze, również przy brakujących danych.



01 Warunek przyjęcia

`WHERE` przepuszcza wyłącznie wiersze, dla których warunek ma wartość `TRUE`. `FALSE` oraz `UNKNOWN` odpadają. To klucz do poprawnego rozumienia NULL.



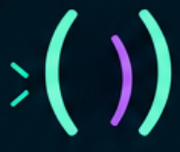
02 Porównania wartości

- Używaj =, <>, <, <=, > oraz >=.
- `BETWEEN` obejmuje obie granice; `IN` sprawdza przynależność do listy wartości.
- Tekst zapisuj w apostrofach.



03 AND, OR i nawiasy

- `AND` ma wyższy priorytet niż `OR`.
- Nawiasy pokazują intencję: `stock > 0 AND (price < 50 OR id = 7)`.



04 Brak wartości

- `email = NULL` nie sprawdza braku adresu.
- Użyj `email IS NULL` albo `IS NOT NULL`.
- Pusty tekst jest wartością różną od `NULL`.



05 Poprawny filtr

- Szukamy klientów bez adresu lub z adresem pustym.
- Warunek celowo obejmuje dwa różne przypadki jakości danych.

```
SELECT id, name FROM customers
WHERE email IS NULL OR email = '';
```

06 Porównanie bezpieczne dla NULL

- Operator <=> traktuje dwa NULL jako równe.
- Zwykłe = zwraca dla takiego porównania `UNKNOWN`.
- Wybierz operator zgodny z regułą biznesową.



07 Sprawdź się

Email klienta ma `NULL`. Czy warunek `email <> 'x@example.com'` go wybierze?

Odpowiedź

Nie. Wynik porównania to `UNKNOWN`; jeśli chcesz go uwzględnić, dodaj `OR email IS NULL`.

PUŁAPKA

`NOT IN` z `NULL` w zbiorze może dać `UNKNOWN` zamiast oczekiwanego `TRUE`. Dla sprawdzania nieistnienia rozważ `NOT EXISTS`.

ZAPAMIĘTAJ

- `NULL` wymaga świadomej obsługi, także w zaprzeczeniach.



Potrafisz przewidzieć wynik filtra przed jego uruchomieniem.

SQL

ORDER BY i LIMIT

Nadaj wynikom jednoznaczny porządek i pobieraj tylko potrzebny fragment.



JAVA + MYSQL W PRAKTYCE

01 Kolejność jest jawna

Bez ORDER BY kolejność nie jest gwarantowana. Obecność klucza głównego ani kolejność wcześniejszego INSERT nie stanowią kontraktu kolejności wyniku.



02 Kierunek sortowania

ASC oznacza rosnąco i jest domyślne; DESC malejąco. Każda kolumna ma własny kierunek: *price* DESC, id ASC.

A → Z ↑ ASC

Z → A ↓ DESC

03 Rozstrzygaj remisy

Dwa produkty mogą mieć tę samą cenę. Dopisz unikalne id jako kolejny klucz sortowania, aby LIMIT nie wybierał dowolnego elementu remisu.



04 Najdroższa trójka

Najpierw sortujemy po cenie malejąco, potem po id rosnąco. LIMIT 3 ogranicza liczbę zwróconych wierszy.

```
SELECT id, name, price FROM products
ORDER BY price DESC, id ASC
LIMIT 3;
```

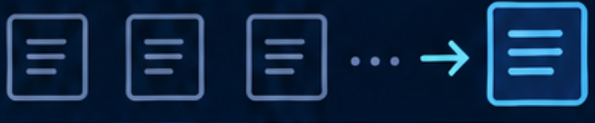
05 Strony z OFFSET

LIMIT 10 OFFSET 20 pomija dwadzieścia wierszy i pobiera następnych dziesięć. Duży OFFSET może być kosztowny; równoległe zmiany przesuwają granice stron.



06 Strona po kluczu

Dla ORDER BY id ASC kolejną stronę pobieraj warunkiem id > ostatnie_id i LIMIT 10. Przy sortowaniu wielokolumnowym kursor musi uwzględniać wszystkie klucze.



07 Sprawdź się

Produkty mają (id, price): (7, 50), (8, 50), (9, 40). Co zwróci przykład z LIMIT 2?

Odpowiedź Najpierw id 7, potem id 8. Rosnące id rozstrzyga remis ceny.

✕ PUŁAPKA

LIMIT bez jednoznacznego ORDER BY nie oznacza „pierwszych” ani „najnowszych” rekordów.

✓ ZAPAMIĘTAJ

- Stabilne stronicowanie zaczyna się od całkowitego porządku.



Potrafisz wyjaśnić, który rekord trafi na granicę strony.

SQL

INNER JOIN i LEFT JOIN

Których klientów pokaże zapytanie?

JAVA + MYSQL W PRAKTYCE

01 Relacja tabel

- customers: klienci
- orders: zamówienia
- customers.id → orders.customer_id
- Aliasy c i o skracają zapis.

customers	
id	PK
name	
email	
...	

1 → ∞

orders	
id	PK
customer_id	FK
order_date	
...	

02 INNER JOIN

- Tylko pasujące pary.
- Klient bez zamówienia znika.
- Dwa zamówienia = dwa wiersze klienta.

```
FROM customers c
INNER JOIN orders o
ON o.customer_id = c.id
```

03 LEFT JOIN

- Zachowuje wszystkich klientów.
- Brak dopasowania: NULL w kolumnach zamówienia.
- NULL nie oznacza numeru zero.



Wszyscy klienci (po lewej), nawet bez zamówień.

04 ON i WHERE

- ON: warunek dopasowania.
- WHERE: filtr wyniku.
- Kolejność tabel ma znaczenie.
- Po lewej: zbiór do zachowania.

05 Przykład SQL

```
SELECT c.name, o.id AS order_id
FROM customers c
LEFT JOIN orders o
  ON o.customer_id = c.id;
```

06 Wynik w praktyce

Anna ma 2 zamówienia, Jan żadnego.

name	order_id
Anna	101
Anna	102
Jan	NULL

07 Sprawdź się

Ile wierszy zwróci każde złączenie?

INNER JOIN: 2 • **LEFT JOIN: 3**

PUŁAPKA

`WHERE o.id IS NOT NULL` usuwa klientów bez zamówień.

ZAPAMIĘTAJ

- LEFT zachowuje lewą stronę.
- INNER wymaga dopasowania.

Najpierw określ, które wiersze chcesz zachować.

SQL

GROUP BY, HAVING i agregacje

JAVA + MYSQL W PRAKTYCE

Zamień wiele pozycji zamówienia w jedną liczbę, zachowując właściwy poziom danych.

01 Wybierz grupę

GROUP BY order_id tworzy grupę pozycji dla każdego zamówienia. Wynik zawiera jeden wiersz na grupę, a nie jeden na produkt.

order_id
101
101
101
102
102

02 Dobierz agregację

SUM liczy sumę, AVG średnią, MIN i MAX skrajne wartości. SUM(quantity * unit_price) uwzględnia liczbę sztuk oraz historyczną cenę sprzedaży.

SUM
AVG
MIN
MAX

03 Policz właściwą rzecz

- COUNT(*) liczy wiersze.
- COUNT(email) liczy wartości różne od NULL.
- COUNT(DISTINCT product_id) liczy różne niepuste identyfikatory produktów.

04 Filtruj w dobrym miejscu

- WHERE odrzuca wiersze przed grupowaniem.
- HAVING odrzuca gotowe grupy, na przykład zamówienia o zbyt małej wartości.
- Nie zastępuj jednego drugim bez sprawdzenia znaczenia.

05 Zamówienia powyżej 100

Dwie pozycje o wartościach 79.80 i 25.00 dadzą total = 104.80. Grupa spełni warunek HAVING.

```sql
SELECT order_id,
       SUM(quantity * unit_price) AS total
FROM order_items
GROUP BY order_id
HAVING SUM(quantity * unit_price) > 100;
```

06 Kolumny bez agregacji

Przy domyślnym ONLY_FULL_GROUP_BY nie wybieraj dowolnej nazwy z grupy. Kolumna musi spełniać reguły grupowania lub zależności funkcyjnej.

07 Sprawdź się

Trzy wiersze mają email: NULL, a@example.com, a@example.com. Ile wynoszą COUNT(*), COUNT(email), COUNT(DISTINCT email)?

Odpowiedź
Odpowiednio 3, 2 i 1.

PUŁAPKA

Po JOIN jeden rekord może wystąpić wielokrotnie. Sprawdź ziarnistość przed SUM, aby nie zawyżyć wartości zamówienia.

ZAPAMIĘTAJ

- Najpierw określ, co reprezentuje jeden wiersz raportu.

Potrafisz odróżnić filtr wierszy od filtra grup.

SQL

CREATE TABLE i ograniczenia

JAVA + MYSQL W PRAKTYCE

Przenieś reguły modelu do bazy, aby obowiązywały każdego klienta zapisującego dane.



01 Struktura tabeli

- `CREATE TABLE` definiuje kolumny, typy i ograniczenia.
- W ćwiczeniach używaj InnoDB.
- Przed wykonaniem sprawdź aktywną bazę przez `SELECT DATABASE()`.

02 Klucz główny

- `PRIMARY KEY` identyfikuje wiersz i nie dopuszcza `NULL` ani duplikatów.
- Dla order_items klucz (order_id, product_id) realizuje przyjętą regułę jednej pozycji produktu.

03 Obecność i unikalność

- `NOT NULL` wymusza wartość.
- `UNIQUE(email)` chroni przed powtórzeniem niepustego adresu według collation; sama unikalność dopuszcza wiele NULL.

04 Reguły liczbowe

- `CHECK(stock >= 0)` odrzuca ujemny stan.
- `CHECK` przyjmuje TRUE lub UNKNOWN, więc połącz go z `NOT NULL`, jeśli brak wartości jest niedozwolony.

05 Minimalna tabela produktu

- Uruchom w pustej bazie ćwiczeniowej.
- Cena i stan muszą istnieć oraz być nieujemne.

```
CREATE TABLE products (
  id BIGINT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(120) NOT NULL,
  price DECIMAL(10,2) NOT NULL, stock INT NOT NULL,
  CHECK (price >= 0), CHECK (stock >= 0))
  ENGINE=InnoDB;
```

06 Powiązania i kontrola

- `FOREIGN KEY` wiąże pozycje z istniejącym zamówieniem i produktem.
- Po utworzeniu tabeli odczytaj `SHOW CREATE TABLE products`, aby sprawdzić rzeczywistą definicję.

07 Sprawdź się

Czy `CHECK(stock >= 0)`, bez `NOT NULL`, odrzuci stock = NULL?

Odpowiedź Nie. Porównanie daje UNKNOWN, które CHECK dopuszcza. Potrzebujesz także `NOT NULL`.

PUŁAPKA

`IF NOT EXISTS` nie potwierdza zgodności istniejącej tabeli z oczekiwaną definicją. Nie zastępuje migracji.

ZAPAMIĘTAJ

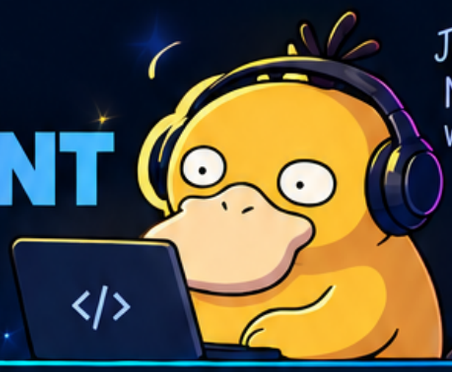
- Waliduj w aplikacji dla wygody, w bazie dla integralności.

Potrafisz zapisać i sprawdzić najważniejsze reguły danych.

SQL

INSERT i AUTO_INCREMENT

Dodaj wiersz, odbierz jego identyfikator i sprawdź, co naprawdę zostało zapisane.



JAVA + MYSQL W PRAKTYCE

01 Wymień kolumny

`INSERT` z jawną listą kolumn pozostaje czytelny po zmianie schematu. Kolejność wartości musi odpowiadać tej liście; tekst wymaga apostrofów.

02 Wygeneruj klucz

Gdy `id` ma `AUTO_INCREMENT`, pomiń je w `INSERT`. Serwer przydzieli identyfikator. Nie licz samodzielnie `MAX(id) + 1`, szczególnie przy równoległych zapisach.

03 Dodaj produkt

Przykład zakłada tabelę `products` ze strony 11. Powstaje produkt z ceną 39.90 i stanem 10 sztuk.

```
INSERT INTO products (name, price, stock)
VALUES ('Kubek', 39.90, 10);
SELECT LAST_INSERT_ID();
```

04 Odbierz właściwy numer

`LAST_INSERT_ID()` dotyczy bieżącego połączenia. W JDBC przygotuj polecenie z `RETURN_GENERATED_KEYS`, wykonaj je i odczytaj `getGeneratedKeys()`. Nie odpytuj innej sesji.

05 Domyślne i wymagane

Pominięta kolumna korzysta z DEFAULT lub reguł typu i trybu SQL. Wymagane dane podawaj jawnie. Nie wyłączaj trybu ścisłego, aby ukryć błędy.

06 Kilka wierszy

Jedno `INSERT` może zawierać wiele zestawów `VALUES`. `LAST_INSERT_ID()` zwróci pierwszy wygenerowany identyfikator takiego polecenia; nie zakładaj bezwarunkowo ciągłości numeracji.

07 Sprawdź się

`INSERT` został wycofany. Czy kolejny zapis musi wykorzystać utracone id?

Odpowiedź

Nie. `AUTO_INCREMENT` może mieć luki, także po `ROLLBACK`. Nie służy do bezlukowej numeracji dokumentów.

PUŁAPKA

`INSERT IGNORE` może zamienić część błędów w ostrzeżenia. Nie używaj go jako uniwersalnej naprawy niepoprawnych danych.

ZAPAMIĘTAJ

- Dodanie rekordu i poznanie jego klucza to jeden kontrolowany przepływ.

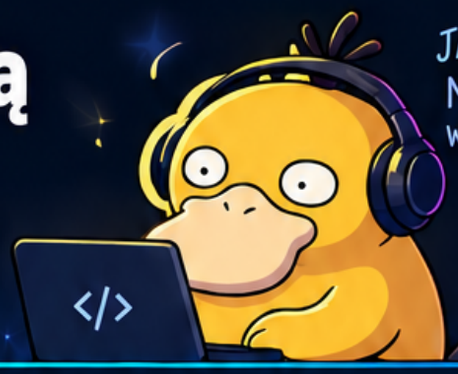


Potrafisz zapisać produkt bez zgadywania jego id.

SQL

UPDATE z kontrolą zakresu

Zmieniaj dokładnie wskazane rekordy i łącz warunek biznesowy z samą aktualizacją.



01 Cel i zakres

SET opisuje zmianę, WHERE wybiera wiersze. Bez WHERE aktualizacja obejmuje wszystkie wiersze tabeli. Przy pojedynczym produkcie filtruj po kluczu głównym.

02 Podgląd pomaga

Przed ręczną zmianą uruchom SELECT z identycznym WHERE. To kontrola zakresu, ale nie blokada: inna transakcja może zmienić dane między poleceniami.

03 Warunek atomowy

Odejmij dwie sztuki tylko wtedy, gdy są dostępne. Warunek stock >= 2 jest sprawdzany jako część aktualizacji, bez osobnego odczytu stanu w aplikacji.

```
UPDATE products
SET stock = stock - 2
WHERE id = 7 AND stock >= 2;
```

04 Oceń rezultat

Dla tego polecenia liczba 1 oznacza udane zmniejszenie stanu. Zero oznacza brak produktu albo niedostateczny stan. Obsłuż tę sytuację jako oczekiwany wynik biznesowy.

05 Próba w transakcji

W lokalnym InnoDB rozpocznij transakcję, wykonaj zmianę, sprawdź dane i zakończ ROLLBACK. Zapis produkcyjny zatwierdzaj dopiero po kontroli całej operacji.

06 Licznik wymaga kontekstu

Aktualizacja ceny na identyczną wartość nie zmienia danych. Connector/J domyślnie raportuje dopasowane wiersze; useAffectedRows zmienia interpretację licznika. Nie myl obu pojęć.

07 Sprawdź się

Produkt 7 ma stock = 1.
Co zrobi pokazane UPDATE?

Odpowiedź Nie zmieni wiersza. Warunek stock >= 2 jest fałszywy; stan pozostanie równy 1.

PUŁAPKA

`LIMIT 1` nie naprawia błędnego WHERE. Może jedynie ograniczyć szkodę do przypadkowego rekordu.

ZAPAMIĘTAJ

- Bezpieczna zmiana = dokładny zakres + warunek + kontrola wyniku.



Potrafisz odjąć stan bez wyścigu odczytaj-zapisz.

SQL

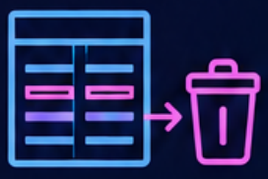
DELETE i reguły usuwania

JAVA + MYSQL W PRAKTYCE

Usuwaj dane świadomie: sprawdź zależności, zakres i możliwość wycofania operacji.

01 Co znika

`DELETE` usuwa wiersze, pozostawiając tabelę. `WHERE` określa zakres. Najpierw policz i obejrzyj kandydatów przez `SELECT` z tym samym warunkiem.



02 Rodzic i dziecko

`order_items` wskazuje `orders` oraz `products`. `RESTRICT` blokuje usunięcie rodzica, który nadal ma zależne rekordy. To zabezpieczenie integralności, nie przypadkowy błąd.



03 Kaskada ma znaczenie

`ON DELETE CASCADE` usuwa zależne wiersze. Ustawiaj ją tam, gdzie dziecko nie ma samodzielnego sensu, np. pozycja usuwanego zamówienia testowego.



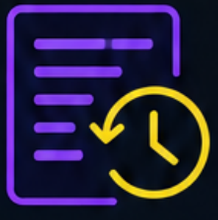
04 Usuwanie kontrolowane

Próba dla lokalnych tabel InnoDB z regułą `RESTRICT`. Najpierw usuwamy pozycje, potem zamówienie; `ROLLBACK` przywraca oba zbiory.

```
START TRANSACTION;
DELETE FROM order_items WHERE order_id = 21;
DELETE FROM orders WHERE id = 21;
ROLLBACK;
```

05 Zachowaj historię

W realnym sklepie usunięcie produktu może zniszczyć kontekst sprzedaży. Często potrzebujesz dezaktywacji zamiast kasowania; regułę retencji ustal przed projektem operacji.



06 DELETE a TRUNCATE

`TRUNCATE TABLE` opróżnia całą tabelę jako DDL, powoduje niejawne zatwierdzenie i nie podlega zwykłemu `ROLLBACK`. Nie zamieniaj nim mechanicznie `DELETE`.



07 Sprawdź się

Zamówienie 21 ma pozycje, a klucz obcy używa `RESTRICT`. Czy usunięcie samego orders.id = 21 powiedzie się?

Odpowiedź

Nie, dopóki istnieją odwołujące się pozycje. Usuń zależności zgodnie z zatwierdzoną regułą lub zachowaj zamówienie.

PUŁAPKA

Wyłączenie foreign_key_checks nie jest naprawą modelu. Ponowne włączenie nie sprawdza automatycznie już wstawionych niespójnych danych.

ZAPAMIĘTAJ

- Usunięcie rekordu jest również decyzją o jego powiązaniach.



Potrafisz przewidzieć skutki kasowania przed wykonaniem polecenia.

SQL

ALTER TABLE i migracje

Zmieniaj schemat małych, sprawdzalnych krokach zgodnych z działającą aplikacją.



01 Schemat też ewoluuje

`ALTER TABLE` dodaje i zmienia kolumny, indeksy oraz ograniczenia. Każda zmiana powinna wynikać z nowej reguły lub potrzeby aplikacji.

02 Rozszerz bez zaskoczeń

Dodaj opcjonalny opis produktu.

- Stare zapisy z jawną listą kolumn nadal działają, a dotychczasowe rekordy mogą mieć NULL.

```
ALTER TABLE products
ADD COLUMN description VARCHAR(500) NULL;
```

03 Wersjonuj zmianę

- Zapisz polecenie w kolejnym, niezmiennym pliku migracji.
- Rejestruj wykonane wersje.
- Odtworzenie pustej bazy powinno prowadzić do tego samego schematu.

04 Najpierw dane

- Zanim ustawisz NOT NULL albo UNIQUE, sprawdź braki i duplikaty.
- Dla wymaganej nowej kolumny: dodaj ją, uzupełnij dane, dopiero potem zaostrz ograniczenie.

05 Oceń koszt

- Nie każda operacja jest natychmiastowa.
- Algorytm zależy od zmiany; przebudowa może zużyć miejsce i blokować pracę.
- Przetestuj na kopii o realistycznym rozmiarze.

06 Zaplanuj odwrót

- ALTER TABLE powoduje niejawny COMMIT.
- ROLLBACK nie jest planem wycofania migracji.
- Przygotuj kopię, migrację naprawczą i zgodność wersji aplikacji.

07 Sprawdź się

Dodajesz `UNIQUE(email)`, lecz dwa rekordy mają ten sam niepusty email. Jaki krok powinien poprzedzić migrację?

Odpowiedź

Wykrycie duplikatów i ustalenie poprawnej korekty danych. Nie usuwaj ich automatycznie bez reguły biznesowej.

PUŁAPKA

Atomowe DDL chroni spójność pojedynczej operacji przy awarii; nie oznacza dowolnego cofania schematu przez ROLLBACK.

ZAPAMIĘTAJ

- Rozszerz → uzupełnij → przełącz aplikację → uporządkuj.



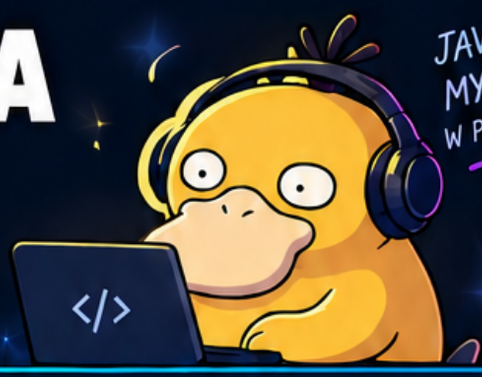
Potrafisz zaplanować zmianę tabeli, danych i kodu jako całość.

SQL

PODZAPYTANIA i EXISTS

JAVA + MYSQL W PRAKTYCE

Wybierz konstrukcję według pytania: jedna wartość, zbiór wartości czy samo istnienie.



01 Zapytanie w zapytaniu

Podzapytanie jest SELECT-em użytym wewnątrz innego polecenia. Pomaga wyrazić zależność bez przenoszenia pośredniej listy identyfikatorów do aplikacji.



02 Jedna wartość

Podzapytanie skalarne zwraca jedną kolumnę i najwyżej jeden wiersz. Więcej wierszy oznacza błąd; pusty wynik daje NULL. Średnia ceny jest dobrym przykładem.



03 Przynależność: IN

IN porównuje wartość ze zbiorem wynikowym. Przy pytaniu „czy klient ma zamówienie?" EXISTS często czytelniej wyraża zamiar niż pobieranie wszystkich identyfikatorów.



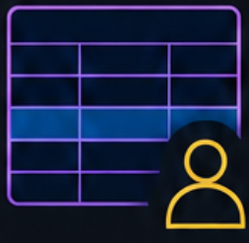
04 Klienci z zamówieniem

Podzapytanie jest skorelowane: o.customer_id odnośi się do aktualnego c.id. SELECT 1 sygnalizuje, że interesuje nas istnienie wiersza.

```sql
SELECT c.id, c.name FROM customers AS c
WHERE EXISTS (
    SELECT 1 FROM orders AS o
    WHERE o.customer_id = c.id
);
```

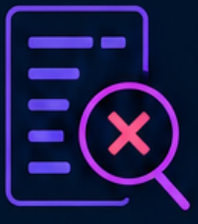
05 Wynik bez mnożenia

Klient z trzema zamówieniami pojawi się raz, ponieważ wybieramy wiersze customers. EXISTS nie dokleja do wyniku każdego pasującego zamówienia.



06 Brak powiązań

NOT EXISTS wybiera klientów bez pasującego zamówienia. Optymalizator może przekształcić zapytanie; nie zakładaj dosłownego osobnego wykonania podzapytania dla każdego klienta.



07 Sprawdź się

Klient 7 ma trzy zamówienia, klient 8 nie ma żadnego. Kogo zwróci przykład?

Odpowiedź

Klienta 7 dokładnie raz. Klient 8 nie spełnia EXISTS.

PUŁAPKA

NOT IN i NOT EXISTS nie są zawsze równoważne przy NULL. Najpierw ustal semantykę brakujących wartości.

ZAPAMIĘTAJ

- Jeśli pytasz o istnienie, zapisz istnienie wprost.



Potrafisz uniknąć duplikatów bez maskowania problemu przez DISTINCT.

SQL

CTE: zapytanie z nazwanymi etapami

Rozbij złożony SQL na czytelne części, które istnieją tylko w obrębie jednego polecenia.



01 Nazwany wynik

`WITH` definiuje CTE, czyli nazwany wynik pomocniczy. Zapytanie końcowe używa go podobnie jak tabeli. Nie powstaje trwała tabela ani widok.



02 Etap agregacji

Najpierw oblicz liczbę zamówień każdego klienta, potem wybierz klientów z przynajmniej dwoma zamówieniami. Nazwa `order_counts` opisuje zawartość etapu.

```
WITH order_counts AS (
   SELECT customer_id, COUNT(*) AS total
   FROM orders GROUP BY customer_id
)
SELECT * FROM order_counts WHERE total >= 2;
```

03 Zakres nazwy

CTE działa tylko w tym jednym poleceniu. Następny, osobny SELECT nie zna `order_counts`. W ramach polecenia można odwołać się do CTE wielokrotnie.



04 Kilka etapów

Po `WITH` oddziel kolejne definicje przecinkami. Późniejszy etap może używać wcześniejszego. Każdy etap powinien mieć czytelną nazwę i jasno określoną ziarnistość.



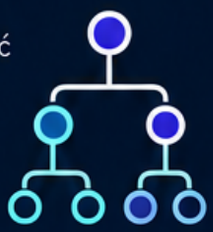
05 Czytelność a wydajność

CTE nie gwarantuje przyspieszenia ani materializacji. Optymalizator może scalić definicję z zapytaniem lub utworzyć wynik pośredni; sprawdzaj plan wykonania.



06 Rekurencja osobno

`WITH RECURSIVE` pozwala budować kolejne poziomy, np. hierarchię. Wymaga części startowej i kroku rekurencyjnego z warunkiem zakończenia; limit głębokości jest zabezpieczeniem.



07 Sprawdź się

Klient 7 ma trzy zamówienia, klient 8 jedno. Jakie dane zwróci przykład?

Odpowiedź

Jeden wiersz: customer_id = 7, total = 3. Klient 8 odpadnie po agregacji.

PUŁAPKA

`CTE` nie jest pamięcią podręczną między zapytaniami. Jeśli potrzebujesz trwałego wyniku, wybierz inny mechanizm.

ZAPAMIĘTAJ

- Dobra nazwa etapu odsłania sens złożonego SQL.

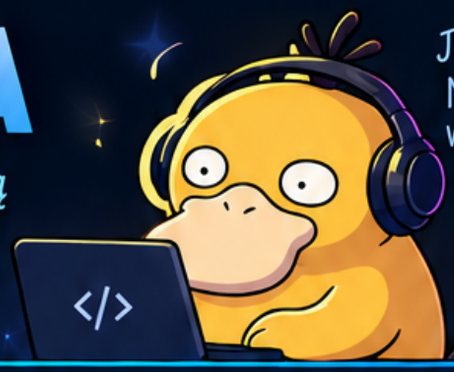


Potrafisz oddzielić obliczenie pośrednie od końcowego filtrowania.

SQL

FUNKCJE OKNA

Oblicz ranking lub wartość porównawczą bez zwijania wielu rekordów w jeden wiersz.



01 Zachowaj szczegóły

`GROUP BY` redukuje wiersze do grup. Funkcja okna dodaje obliczenie do poszczególnych wierszy wyniku, np. numer zamówienia danego klienta.

02 Podział na partycje

`PARTITION BY customer_id` wyznacza osobne okno dla każdego klienta. Bez `PARTITION BY` funkcja traktuje wszystkie wiersze jako jedną partycję.

03 Kolejność wewnątrz

- `ORDER BY` w `OVER` ustala porządek obliczenia.

Unikalne `id` rozstrzyga remis `created_at`. Końcową kolejność prezentacji ustala osobne `ORDER BY` zapytania.

04 Numer zamówienia

`ROW_NUMBER` nadaje kolejne numery w obrębie klienta. Pierwsze chronologicznie zamówienie klienta ma `sequence_no = 1`.

```
SELECT id, customer_id,
  ROW_NUMBER() OVER (PARTITION BY customer_id
  ORDER BY created_at, id) AS sequence_no
FROM orders
ORDER BY customer_id, sequence_no;
```

05 Wybierz funkcję

`ROW_NUMBER` numeruje każdy wiersz osobno.
`RANK` zostawia luki po remisach, `DENSE_RANK` ich nie zostawia.
`LAG` pozwala odczytać wartość z wcześniejszego wiersza.

06 Filtr po obliczeniu

Nie używaj wyniku funkcji okna w `WHERE` tego samego poziomu. Oblicz go w CTE albo podzapytaniu, a dopiero na zewnątrz filtruj `sequence_no = 1`.

07 Sprawdź się

Dla wartości 100, 100, 80 ranking malejący używa tylko wartości. Jakie numery dadzą `RANK` i `DENSE_RANK`?

Odpowiedź `RANK`: 1, 1, 3. `DENSE_RANK`: 1, 1, 2.

✕ PUŁAPKA

Przy `SUM` z `ORDER BY` domyślna ramka obejmuje także remisy. Dla sumy narastającej po wierszach świadomie wybierz `ROWS` i jednoznaczny porządek.

✓ ZAPAMIĘTAJ

`PARTITION` dzieli,
`ORDER` porządkuje,
funkcja oblicza.



Potrafisz dodać ranking bez utraty szczegółowych rekordów.

SQL

INDEKSY
POJEDYNCZE I ZŁOŻONE

JAVA + MYSQL W PRAKTYCE

Projektuj ścieżkę dostępu pod konkretne zapytania, a koszt oceniaj na rzeczywistych danych.



01 Cel indeksu

Indeks pomaga znaleźć potrzebne rekordy bez przeglądania całej tabeli. Może też wspierać sortowanie lub pokryć zapytanie, ale zajmuje miejsce.

02 Indeks pojedynczy

Indeks na `products.price` może pomóc przy selektywnym zakresie cen. Dla małej tabeli lub bardzo szerokiego zakresu skan bywa tańszy.

03 Indeks złożony

Dla historii klienta filtrujesz `customer_id` i zakres `created_at`. Indeks porządkuje najpierw klientów, a następnie daty wewnątrz każdego klienta.

```
CREATE INDEX idx_orders_customer_date
ON orders (customer_id, created_at);
```

04 Lewy prefiks

Indeks `(customer_id, created_at)` naturalnie wspiera wyszukiwanie po `customer_id` oraz po obu kolumnach. Sam warunek daty nie wykorzystuje takiego samego bezpośredniego dostępu.

05 Cena za zapis

INSERT, DELETE i zmiany indeksowanych wartości aktualizują indeksy. Więcej indeksów nie oznacza automatycznie lepszej bazy; sprawdzaj istniejące definicje przez `SHOW INDEX`.

06 Sprawdź hipotezę

Porównaj `EXPLAIN` przed i po dodaniu indeksu. Zbadaj liczbę czytanych wierszy i czas na reprezentatywnych danych; samo pojawienie się indeksu nie dowodzi poprawy.

07 Sprawdź się

Które zapytanie lepiej pasuje do pokazanego indeksu: `customer_id = 7` czy tylko `created_at >= '2026-01-01'`?

Odpowiedź

Pierwsze wykorzystuje lewy prefiks. Dla samej daty trzeba ocenić inny indeks lub plan dostępu.

✕ PUŁAPKA

Dwa indeksy pojedyncze nie są odpowiednikiem indeksu złożonego. Optymalizator może je łączyć, ale koszt i możliwości są inne.

✓ ZAPAMIĘTAJ

- Indeks dobierasz do wzorca zapytania i rozkładu danych.

Potrafisz uzasadnić kolejność kolumn zamiast indeksować wszystko.

SQL

EXPLAIN i EXPLAIN ANALYZE

JAVA + MYSQL W PRAKTYCE

Odróżnij przewidywany plan od pomiaru wykonania i znajdź miejsce zbędnej pracy.



01 Zacznij od pytania

Zapisz konkretne zapytanie, typowe parametry i oczekiwany wynik. Plan ma sens w kontekście rozmiaru tabel, selektywności filtrów oraz dostępnych indeksów.

02 EXPLAIN: przewidywanie

EXPLAIN FORMAT=TRADITIONAL pokazuje m.in. type, possible_keys, key i rows. To plan i szacunki, nie zmierzony czas ani gwarantowana liczba odczytanych rekordów.

03 ANALYZE: wykonanie

EXPLAIN ANALYZE faktycznie uruchamia zapytanie.
W MySQL 8.4 używa formatu TREE i dodaje rzeczywiste czasy, liczbę wierszy oraz pętli.

04 Bezpieczny pomiar

Uruchom SELECT na lokalnej kopii danych. Porównaj działanie z indeksem (customer_id, created_at) ze strony 19.

```
EXPLAIN ANALYZE FORMAT=TREE
SELECT id, created_at FROM orders
WHERE customer_id = 7
ORDER BY created_at;
```

05 Czytaj od źródeł

Śledź skany, filtry i połączenia od najgłębszych węzłów.
Duża różnica między szacowanymi i rzeczywistymi wierszami wskazuje miejsce wymagające zbadania.

06 Interpretuj liczby

Czasy są w milisekundach; przy wielu pętlach czas jest średnią na pętlę. Uwzględnia dzieci węzła, więc nie sumuj bezmyślnie wszystkich czasów drzewa.

07 Sprawdź się

Plan przewiduje 10 wierszy, a pomiar pokazuje 100 000. Czy to dowód, że indeks jest uszkodzony?

Odpowiedź

Nie. Sprawdź rozkład danych, statystyki i warunki zapytania. To sygnał błędnego oszacowania, nie gotowa diagnoza.

PUŁAPKA

Using filesort nie oznacza automatycznie sortowania na dysku. Nie oceniaj planu na podstawie pojedynczej etykiety.

ZAPAMIĘTAJ

- Hipoteza → plan → pomiar → jedna zmiana → ponowny pomiar.

Potrafisz wyjaśnić poprawę za pomocą dowodów, zamiast zgadywać.

SQL

CONNECTION i DATASOURCE

JAVA + MYSQL W PRAKTYCE

Fabryka połączeń → sesja → praca → zwolnienie



01 Dwie odpowiedzialności

DataSource udostępnia połączenia. Connection reprezentuje konkretną sesję bazy: wykonuje SQL i przechowuje stan transakcji, izolacji oraz ustawień sesyjnych.



02 Konfiguruj raz

Wstrzyknij skonfigurowany DataSource do serwisu. Adres, użytkownika i sekrety dostarczaj z konfiguracji środowiska; nie umieszczaj hasła w repozytorium ani logach.



03 Pożycz na czas pracy

Fragment zakłada skonfigurowane ds typu DataSource i obsługę SQLLeception. Zakończenie bloku zawsze wywoła close(), także po błędzie.

```
try (Connection c = ds.getConnection()) {
   boolean valid = c.isValid(2);
   System.out.println(valid);
}
```

04 Pula jest implementacją

Sam interfejs DataSource nie gwarantuje puli. Przy puli close() zwykle oddaje logiczne połączenie do ponownego użycia; prosta implementacja zamyka połączenie fizyczne.



05 Ustal właściciela

Warstwa rozpoczynająca jednostkę pracy odpowiada za jej zakończenie. DAO korzystające z przekazanego Connection nie powinno go zamykać ani samodzielnie zatwierdzać transakcji.



06 Nie współdziel sesji

Nie trzymaj jednego globalnego Connection dla równoległych żądań. Współdziel źródło, a osobne jednostki pracy obsługuj osobnymi wypożyczeniami. Sprawdzaj timeouty i limity puli.



07 Sprawdź się

Czy DataSource zawsze przechowuje pulę gotowych połączeń?

Odpowiedź: Nie. To interfejs; zachowanie zależy od wybranej implementacji.

PUŁAPKA

Pozostawione otwarte połączenia wyczerpują pulę, nawet gdy baza nadal działa poprawnie.

ZAPAMIĘTAJ

- Źródło jest długowieczne;
- wypożyczenie powinno być krótkie i mieć właściciela.

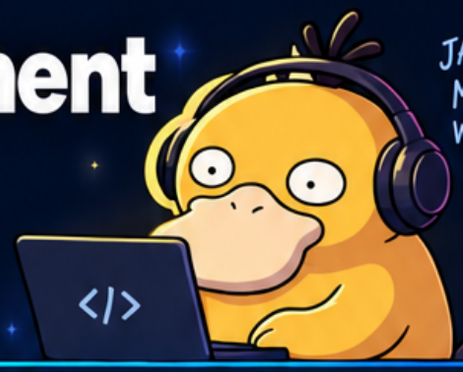


Połącz pozyskanie zasobu z jego zwolnieniem.

SQL

PreparedStatement i parametry

Wartości przekazuj oddzielnie od SQL



01 Szablon SQL

- Stała struktura zapytania.
- Wartości zastępuj znakiem ?.
- Nie doklejaj danych użytkownika.

```
UPDATE products
SET price = ?
WHERE id = ?
```

02 Wartości, nie nazwy

- ?: cena, ID lub tekst.
- Nie: nazwa tabeli czy kolumny.
- Nie otaczaj ? apostrofami.
- Dynamiczne nazwy: lista dozwolonych.

03 Typ i kolejność

- Numeracja parametrów od 1.
- setInt → liczba całkowita
- setString → tekst
- setBigDecimal → kwota

```
?1 → cena
UPDATE products
SET price = ?
WHERE id = ?
?2 → id
```

04 Wykonanie i zasoby

- UPDATE → executeUpdate()
- SELECT → executeQuery()
- try-with-resources zamyka komendę także przy wyjątku.
- Najpierw ustaw parametry.

05 Fragment Java

con: otwarte połączenie

```
String sql =
  "UPDATE products SET price=? WHERE idc=?";
try (PreparedStatement ps = con.prepareStatement(sql)) {
    ps.setBigDecimal(1, new BigDecimal("19.90"));
    ps.setInt(2, 7);
    ps.executeUpdate();
}
```

06 Efekt

- Produkt o ID 7 otrzymuje cenę 19,90 PLN.
- Cena i ID są parametrami, nie fragmentami składni SQL.

id	price
7	19.90

07 Sprawdź się

Który parametr wskazuje produkt?

```
UPDATE products SET price=? WHERE id=?
```

Odpowiedź: drugi — setInt(2, 7).

PUŁAPKA

prepareStatement nie chroni doklejanego, niezaufanego SQL.

ZAPAMIĘTAJ

Parametryzuj wartości.
Kontroluj dynamiczne nazwy.

Dane użytkownika mają pozostać danymi.

SQL

executeQuery, executeUpdate i execute

JAVA + MYSQL W PRAKTYCE

Wybierz metodę według rodzaju wyniku SQL



01 Odczyt wierszy

- executeQuery() zwraca ResultSet.
- Używaj dla zapytań dających jeden zbiór wyników, na przykład SELECT id, name FROM products.
- Pusty wynik nadal jest ResultSet.



02 Licznik operacji

- executeUpdate() zwraca int dla INSERT, UPDATE i DELETE.
- DDL może zwrócić zero.
- Zero nie oznacza błędu; znaczenie licznika UPDATE zależy także od konfiguracji sterownika.



03 Warunkowa rezerwacja

- Fragment: ps zawiera UPDATE products SET stock=stock-? WHERE id=? AND stock>=?, z ustawionymi parametrami: dodatnią ilością, identyfikatorem i tą samą ilością.

```
int count = ps.executeUpdate();
if (count != 1) {
    throw new SQLException(
      "Brak produktu lub zapasu");
}
```

04 Rodzaj wyniku

- execute() zwraca true, gdy pierwszy wynik jest ResultSet.
- false oznacza licznik aktualizacji albo brak wyniku, a nie niepowodzenie wykonania.



05 Odbierz całą sekwencję

- Po execute() odczytaj getResultSet() albo getUpdateCount().
- Następnie używaj getMoreResults().
- Koniec następuje dopiero, gdy nie ma ResultSet i licznik wynosi -1.



06 Nie mieszaj interfejsów

- PreparedStatement ma już przypisany SQL, więc wywołuj metody bez argumentu tekstowego.
- Nie przekazuj ponownie SQL do executeQuery(sql) na obiekcie PreparedStatement.



07 Sprawdź się

execute() zwróciło false, a getUpdateCount() zwróciło 3. Co się stało?

Odpowiedź

Pierwszy wynik jest licznikiem równym 3; false nie sygnalizuje błędu.

PUŁAPKA

Nie pokazuj sukcesu biznesowego wyłącznie na podstawie braku SQLException; sprawdź oczekiwany licznik.

ZAPAMIĘTAJ

- SELECT → zbiór, DML → licznik, nieznany wynik → execute().



Wynik techniczny musi spełniać warunek biznesowy.

SQL

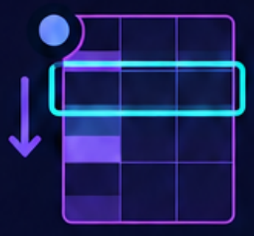
ResultSet

Kursor → bieżący wiersz → typowane wartości → obiekt



01 Kursor przed początkiem

Nowy ResultSet nie wskazuje pierwszego wiersza. next() przesuwa kursor i zwraca true, gdy wiersz istnieje. Najpierw przesunięcie, potem getLong(), getString() i inne odczyty.



02 Czytaj jawne kolumny

Zamiast SELECT * wybierz potrzebne pola. Nazwy lub aliasy zwiększają czytelność mapowania; numery kolumn zaczynają się od 1, nie od 0.



03 Zmapuj jeden wiersz

Fragment: otwarty rs pochodzi z SELECT id, name, price FROM products. Product jest rekordem z polami long, String, BigDecimal; products to lista.

```
while (rs.next()) {
    products.add(new Product(rs.getLong("id"),
                             rs.getString("name"),
                             rs.getBigDecimal("price")));
}
```

04 Wybieraj właściwy typ

Kwoty DECIMAL odczytuj jako BigDecimal. Dla wartości dopuszczających NULL wybierz obiektowy typ Java lub natychmiast sprawdzaj wasNull() po getterze typu prostego.



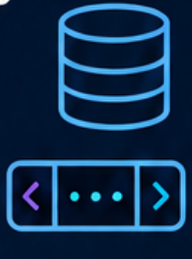
05 Respektuj czas życia

Wynik należy do instrukcji. Zamknięcie Statement zamyka jego bieżący ResultSet; ponowne wykonanie instrukcji też może unieważnić poprzedni wynik. Mapuj dane wewnątrz zakresu zasobów.



06 Ogranicz objętość

Domyślnie licz na kursor jednokierunkowy i tylko do odczytu. Ogranicz wynik SQL, stosuj stronicowanie; samo setFetchSize() nie gwarantuje strumieniowania w Connector/J.



07 Sprawdź się

Zapytanie zwróciło zero rekordów. Czy next() rzuci wyjątek?

Odpowedź Nie. Zwróci false, więc pętla nie wykona ani jednej iteracji.

PUŁAPKA

Zwracanie ResultSet z DAO po zamknięciu jego zasobów oddaje odbiorcy bezużyteczny uchwyt.

ZAPAMIĘTAJ

- Zwracaj z DAO dane, których życie
- nie zależy od kursora.

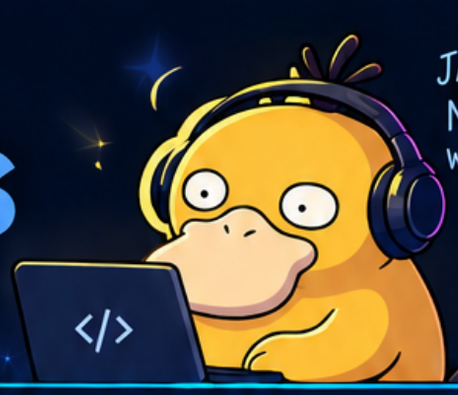


Każdy wiersz zamień w świadomie dobrany typ.

SQL

TRY-WITH-RESOURCES

Pozyskanie i zamknięcie zasobów w jednym zakresie



01 Zasoby wymagają końca

Connection, Statement i ResultSet implementują AutoCloseable. try-with-resources zamyka je przy normalnym wyjściu, return oraz wyjątku, bez powtarzania rozbudowanych bloków finally.

02 Zapisz zależności

Zadeklaruj zasoby od nadrzędnego do zależnego. Poniższy fragment zakłada skonfigurowane ds i obsługę SQLException. Odczyt nie wymaga ręcznego sterowania transakcją.

```
try (Connection c = ds.getConnection();
    var ps = c.prepareStatement("SELECT id FROM customers");
    var rs = ps.executeQuery()) {
  while (rs.next()) System.out.println(rs.getLong(1));
}
```

03 Odwrotna kolejność

Zamykanie nastąpi w kolejności rs → ps → c. Gdy utworzenie późniejszego zasobu zawiedzie, zasoby utworzone wcześniej także zostaną zamknięte.

rs → ps → c

04 Zachowaj pierwotny błąd

Jeżeli ciało try i close() rzucą wyjątki, pierwotny błąd pozostaje głównym. Błąd zamykania trafia do wyjątków stłumionych, dostępnych przez getSuppressed().

05 Domknij transakcję osobno

Automatyczne close() nie zastępuje świadomego commit() ani rollback(). Aktywną transakcję zakończ przed zwolnieniem połączenia; nie polegaj na domyślnym zachowaniu sterownika.

06 Szanuj cudzą własność

DAO zamyka instrukcje i wyniki, które utworzyło. Nie obejmuj przekazanego przez serwis Connection własnym try-with-resources, jeśli jego życie obejmuje kolejne operacje.

07 Sprawdź się

Tworzenie ResultSet nie powiodło się. Co z wcześniej utworzonym ps?

Odpowiedź

Zostanie automatycznie zamknięty, podobnie jak wcześniej pozyskane połączenie.

PUŁAPKA

Nie łap wyjątku tylko po to, aby go przemilczeć; zachowaj przyczynę i kontekst operacji.

ZAPAMIĘTAJ

- Automatyczne sprzątanie zasobów i decyzja o transakcji to dwa osobne obowiązki.

Krótki zakres ułatwia kontrolę zasobów.

SQL

SQL NULL a Java null

Brak wartości to osobny stan, nie zero ani pusty tekst



01 Dwa światy

- SQL NULL oznacza brak lub nieznaną wartość.
- Java null oznacza brak referencji.
- Pusty napis, liczba 0 i false pozostają zwykłymi wartościami.



02 Porównuj poprawnie

- W SQL używaj IS NULL albo IS NOT NULL.
- Warunek email = NULL nie wybiera brakujących adresów, ponieważ porównanie z NULL daje UNKNOWN.

03 Parametr z typem

- Fragment: ps aktualizuje customers.email i ma parametr adresu pod indeksem 1.
- Gdy model dopuszcza brak adresu, wyślij typowany SQL NULL.

```
if (email == null) {
    ps.setNull(1, Types.VARCHAR);
} else {
    ps.setString(1, email);
}
```

04 Odczyt typu prostego

- getLong() zwraca 0 dla SQL NULL.
- Natychmiastowe wasNull() pozwala odróżnić brak od zera; każdy kolejny getter zmienia to, do czego odnosi się wasNull().



05 Wariant obiektowy

- Po LEFT JOIN kolumna orders.id może być NULL.
- Dla aliasu order_id typu BIGINT SIGNED pobierz Long: getObject("order_id", Long.class).
- Brak dopasowania daje null.

06 Modeluj znaczenie

- Ustal, co oznacza brak email w domenie.
- Stosuj NOT NULL dla obowiązkowych pól.
- COALESCE dobieraj świadomie, ponieważ zastąpienie NULL ukrywa informację o braku.



07 Sprawdź się

getLong("order_id") zwróciło 0. Czy zamówienie ma identyfikator 0?

Odpowiedź

Niekoniecznie. Sprawdź wasNull() natychmiast po odczycie albo użyj typu Long.

PUŁAPKA

Automatyczny unboxing null do long kończy się NullPointerException; typ obiektowy wymaga obsługi braku.

ZAPAMIĘTAJ

- Nie zamieniaj brakującej wartości w fałszywą informację biznesową.



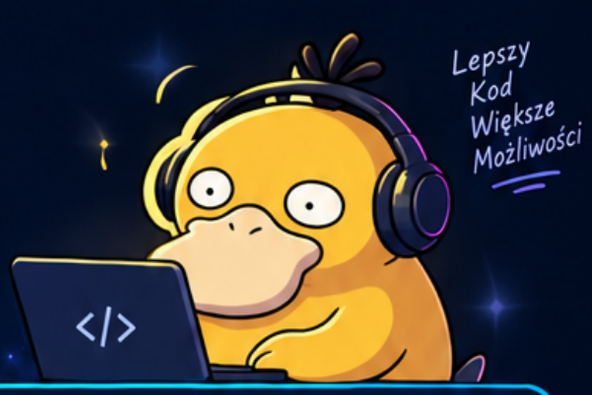
Brak danych także jest informacją.

JAVA + MYSQL W PRAKTYCE

SQL

DECIMAL i BigDecimal

Dokładne kwoty od MySQL do Javy



01 DECIMAL(10,2)

- 10 cyfr łącznie.
- 8 przed przecinkiem.
- 2 po przecinku.
- Dokładne wartości dziesiętne.

99999999,99

8 cyfr | 2 cyfry

02 BigDecimal

- Twórz kwoty z tekstu.
- Używaj kropki w kodzie.
- Obiekt jest niezmienny.
- Zachowaj wynik operacji.

```
BigDecimal kwota =
new BigDecimal("19.90");
```

03 Mapowanie JDBC

- PreparedStatement: setBigDecimal(...) → zapis
- ResultSet: getBigDecimal(...) → odczyt
- Bez konwersji przez double.

```
PreparedStatement ps = ...;
ps.setBigDecimal(1, kwota);

ResultSet rs = ...;
BigDecimal cena =
rs.getBigDecimal("cena");
```

04 Zaokrąglanie

- Skala: miejsca po przecinku.
- Reguła wynika z wymagań.
- Przykład: rabat 15%, skala 2, HALF_UP.

16,915 → 16,92

05 Fragment Java

```
BigDecimal price = new BigDecimal("19.90");
BigDecimal factor = new BimDecimal("0.85");
BigDecimal result = price.multiply(factor);
result = result.setScale(2, RoundingMode.HALF_UP);
```

Pominięto importy.

06 Efekt

- Cena: 19,90 PLN
- Rabat: 15%
- Przed zaokrągleniem: 16,915 PLN
- Do zapłaty: 16,92 PLN

07 Sprawdź się

Ile cyfr przed przecinkiem ma DECIMAL(10,2)?

Odpowiedź: 8.
Znak minus nie wlicza się do precyzji.

PUŁAPKA

`new BigDecimal(0.1)` przejmuje przybliżenie *double*.



ZAPAMIĘTAJ

Używaj tekstu: `new BigDecimal("0.1")`.



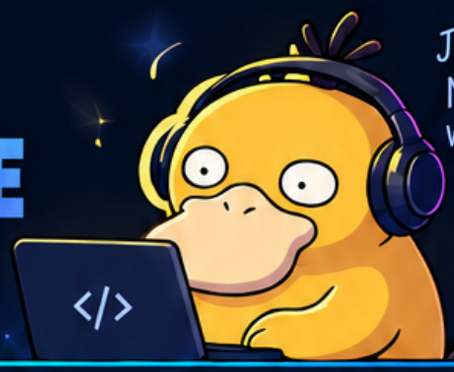
Precyzja ma znaczenie, gdy liczysz pieniądze.

SQL

DATY, CZAS i STREFY CZASOWE

JAVA + MYSQL W PRAKTYCE

Najpierw znaczenie czasu, potem typ kolumny.



01 Dzień kalendarzowy

DATE opisuje dzień bez godziny i strefy.
W Javie używaj LocalDate.
Data urodzenia nie powinna przesuwać się po zmianie strefy użytkownika.



02 Lokalna data i godzina

DATETIME przechowuje lokalne pola daty i czasu bez automatycznej konwersji strefowej jak TIMESTAMP.
Naturalnym typem Java jest LocalDateTime, który sam nie identyfikuje chwili.



03 Chwila na osi czasu

Dla zdarzenia orders.created_at można przyjąć TIMESTAMP(6).
MySQL konwertuje między strefą sesji a UTC.
Oryginalna strefa geograficzna nie zostaje zapisana w kolumnie.



04 Ustal kontrakt UTC

Przykładowe parametry Connector/J:
`connectionTimeZone=UTC`, `forceConnectionTimeZoneToSession=true`, `preserveInstants=true`.
Konfiguracja sterownika i rzeczywista strefa sesji muszą być zgodne; samo ustawienie JVM nie wystarcza.

05 Odczytaj jako chwilę

Fragment: rs wskazuje wiersz, created_at jest niepustym TIMESTAMP, a połączenie ma opisany kontrakt UTC. Timestamp służy tu jako pomost JDBC.

```
Instant createdAt =
   rs.getTimestamp("created_at").toInstant();
ZonedDateTime local = createdAt.atZone(
    ZoneId.of("Europe/Warsaw"));
```

06 Sprawdź granice

Testuj zmianę czasu letniego, różne strefy JVM i serwera oraz precyzję mikrosekund.
TIMESTAMP w MySQL 8.4 ma ograniczony zakres sięgający roku 2038.



07 Sprawdź się

Czy samo LocalDateTime wystarczy, aby wskazać dokładną chwilę na świecie?

Odpowiedź

Nie. Potrzebna jest strefa lub offset oraz rozstrzygnięcie ewentualnej niejednoznaczności.

PUŁAPKA

Nie traktuj **DATETIME** jako automatycznie zapisanej chwili UTC bez jawnej umowy aplikacji.

ZAPAMIĘTAJ

- Typ opisuje znaczenie; konfiguracja gwarantuje zgodną interpretację.



Wyświetlanie lokalizuj dopiero na brzegu aplikacji.

SQL

KLUCZE GENEROWANE i OPERACJE BATCH

JAVA + MYSQL W PRAKTYCE

Identyfikator po INSERT; zbiór operacji pod kontrolą



01 Poproś o klucz

Dla orders.id z AUTO_INCREMENT przygotuj INSERT przez c.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS). Parametr customer_id ustaw przed wykonaniem; created_at może przyjąć CURRENT_TIMESTAMP(6).

02 Odbierz wynik INSERT

Fragment zakłada otwarte ps po udanym executeUpdate(), przygotowane z żądaniem kluczy. Wynik kluczy również trzeba zamknąć; nie zakładaj, że zawsze zawiera wiersz.

```
try (var keys = ps.getGeneratedKeys()) {
  if (!keys.next()) throw new SQLException("Brak ID");
  long orderId = keys.getLong(1);
}
```

03 Zbuduj batch

Dla kolejnych order_items ustaw wszystkie parametry tego samego PreparedStatement i wywołaj addBatch(). executeBatch() wysyła zebrane operacje. Dobieraj wielkość partii do pomiarów i limitów.

04 Sprawdź liczniki

Wyniki odpowiadają kolejności dodania. Wartość nieujemna oznacza licznik, SUCCESS_NO_INFO sukces bez dokładnej liczby, a EXECUTE_FAILED niepowodzenie przy kontynuowaniu partii.

05 Obsłuż częściową awarię

BatchUpdateException może zawierać wyniki już wykonanych poleceń. Batch sam nie zapewnia atomowości: użyj transakcji i wycofaj całą jednostkę pracy, gdy część zawiedzie.

06 Optymalizuj po pomiarze

rewriteBatchedStatements może zmieniać sposób wysyłania batcha przez Connector/J. Sprawdź liczniki i klucze dla użytego SQL; nie wyliczaj kolejnych identyfikatorów z pierwszego.

07 Sprawdź się

Czy executeBatch() automatycznie gwarantuje „wszystko albo nic”?

Odpowiedź

Nie. Taką granicę zapewnia poprawnie obsłużona transakcja na tabelach transakcyjnych.

PUŁAPKA

SELECT MAX(id) może zwrócić rekord innego klienta; nie służy do pobierania właśnie nadanego klucza.

ZAPAMIĘTAJ

- Pobieraj klucz z wykonanej instrukcji i jawnie kontroluj atomowość partii.

Szybsza wysyłka nie zastępuje poprawności.

SQL

DAO, repozytorium i warstwa usług

JAVA + MYSQL W PRAKTYCE

Oddziel regułę biznesową od sposobu zapisu

01 DAO zna JDBC

DAO przygotowuje SQL, ustawia parametry i mapuje wiersze. Ukrywa techniczne szczegóły odczytu oraz zapisu; nie decyduje samodzielnie o przebiegu całego zakupu.

02 Repozytorium zna model

Repozytorium udostępnia operacje nazwane językiem domeny, na przykład findProduct albo saveOrder. W małej aplikacji może pokrywać się z DAO; nie mnożymy warstw bez korzyści.

03 Serwis prowadzi przypadek

OrderService sprawdza dodatnią ilość, rezerwuje zapas i zapisuje zamówienie z pozycjami. Określa granicę transakcji obejmującą wszystkie zależne operacje.

04 Przekaż wspólną sesję

Fragment przedstawia kontrakt interfejsu JDBC. Wywołujący zarządza połączeniem i transakcją; implementacja metody zamyka jedynie własne instrukcje i wyniki.

```
interface ProductDao {
    boolean reserve(Connection c, long productId,
            int quantity) throws SQLException;
}
```

05 Zwracaj dane domenowe

Oddawaj Product, Optional<Product> albo listę obiektów. Nie wypuszczaj ResultSet poza jego zakres życia. Brak produktu odróżniaj od technicznego błędu bazy.

06 Testuj dwa poziomy

Reguły serwisu sprawdzaj z kontrolowanym zastępstwem repozytorium. SQL, mapowanie, ograniczenia i rollback testuj integracyjnie z MySQL; sam mock nie potwierdzi poprawności zapytania.

07 Sprawdź się

Która warstwa powinna zatwierdzić zapis orders, order_items i zmianę stock?

Odpowiedź

Serwis lub mechanizm transakcyjny prowadzący cały przypadek użycia, po powodzeniu wszystkich kroków.

PUŁAPKA

Osobny commit w każdym DAO może pozostawić zamówienie bez pozycji po awarii kolejnej operacji.

ZAPAMIĘTAJ

- Granica przypadku użycia powinna wyznaczać granicę transakcji.

Każda warstwa otrzymuje jasno określony obowiązek.

SQL

ACID na przykładzie zamówienia

Jedna decyzja biznesowa, cztery gwarancje



JAVA + MYSQL W PRAKTYCE

01 A: atomowość

Zapis **orders**, zapis **order_items** i zmniejszenie **products.stock** tworzą całość. W transakcji InnoDB zatwierdzasz komplet zmian albo wycofujesz je po błędzie.

02 C: spójność

Klucze obce pilnują powiązań, a ograniczenia i reguły aplikacji chronią dane. ACID nie wymyśli za programistę warunku „quantity > 0” ani zasad rezerwowania zapasu.

03 I: izolacja

Równoległe zakupy wpływają na siebie według poziomu izolacji i blokad. Zwykły SELECT zapasu, po którym następuje zapis obliczonej wartości, może być niewystarczający.

04 Praktyczna rezerwacja

Fragment SQL wymaga dodatniej ilości i tego samego q w obu miejscach. Ustaw parametry, a potem wymagaj jednego dopasowanego wiersza; zero oznacza odrzucenie rezerwacji.

```
UPDATE products
SET stock = stock - ?
WHERE id = ? AND stock >= ?;
```

05 D: trwałość

Po potwierdzonym commit dane mają przetrwać awarię zgodnie z gwarancjami konfiguracji serwera i sprzętu. Trwałość nie zastępuje backupu ani planu odtwarzania po utracie danych.

06 Granice gwarancji

Transakcja MySQL nie cofnie wysłanego emaila ani płatności zewnętrznej. Takie skutki wymagają osobnej koordynacji. Nie mieszaj operacji DDL z biznesową transakcją DML.

07 Sprawdź się

Po rezerwacji zapasu zapis pozycji zamówienia się nie udał. Co należy zrobić?

Odpowiedź

Wycofać całą transakcję, aby przywrócić zapas i usunąć niezakończony zapis zamówienia.

PUŁAPKA

Brak wyjątku przy UPDATE z licznikiem 0 nie oznacza udanej rezerwacji produktu.

ZAPAMIĘTAJ

Gwarancje silnika działają razem z poprawnym modelem i logiką aplikacji.

Zdefiniuj, co dokładnie znaczy „wszystko”.

SQL

Transakcje JDBC commit i rollback

Kilka zmian jako jedna całość



01 Auto-commit

- Domyślnie w nowym połączeniu JDBC.
- Każde zakończone polecenie: osobna transakcja.
- Dwa INSERT nie tworzą wspólnej całości.

```
Connection con =
  dataSource.getConnection();

boolean ac = con.getAutoCommit();
System.out.println(ac);
// domyślnie: true
```

02 Jedna transakcja

- Wyłącz auto-commit przed zmianami.
- Używaj jednego połączenia.
- Tabele muszą obsługiwać transakcje: InnoDB.

```
Connection con =
  dataSource.getConnection();
con.setAutoCommit(false);
// wszystkie kolejne zmiany
// w jednej transakcji
```

03 Commit czy rollback?

- commit: zatwierdza zmiany.
- rollback: wycofuje niezatwierdzone zmiany.
- Rollback nie cofnie wcześniejszego commit.



04 Domknięcie

- Najpierw commit lub rollback.
- Potem przywróć tryb połączenia.
- Na końcu zamknij zasoby.
- DDL może niejawnie zatwierdzić zmiany.

1. commit lub rollback
2. przywróć poprzedni tryb
3. zamknij połączenie

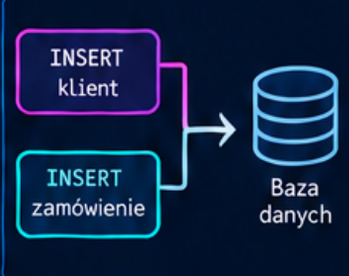
05 Fragment Java

```
con.setAutoCommit(false);
try {
    utworzKlienta(con);
    utworzZamowienie(con);
    con.commit();
} catch (SQLException e) {
    con.rollback(); throw e;
}
```

Metody wykonują INSERT przez con.

06 Efekt

- Klient i zamówienie: zapis razem albo wycofanie zmian.
- Skrót pomija zarządzanie zasobami i obsługę błędu rollback.



07 Sprawdź się

Drugi INSERT zawiódł. Czy rollback cofnie pierwszy?

Tak — w tej samej niezatwierdzonej transakcji innoDB.

PUŁAPKA

false → true w auto-commit zatwierdza otwartą transakcję.

ZAPAMIĘTAJ

Jedna operacja biznesowa, jedno połączenie i jawna decyzja.



Testuj również awarię pomiędzy zapisami.

SQL

JEDNA TRANSAKCJA, JEDNO POŁĄCZENIE

JAVA + MYSQL W PRAKTYCE

Wszystkie kroki muszą trafić do tej samej sesji



01 Wskaż właściciela

Serwis pobiera Connection na czas całego zamówienia. DAO otrzymują to samo c; każde tworzy własne instrukcje, ale nie otwiera drugiej sesji do zależnego zapisu.



02 Wyłącz autocommit

Przed pierwszą operacją ustaw setAutoCommit(false).
Zwykła lokalna transakcja JDBC obejmuje pracę jednego połączenia; wspólny DataSource nie łączy transakcji różnych wypożyczeń.



03 Zatwierdź albo wycofaj

Fragment wewnątrz try-with-resources dla c. saveOrder(c) oznacza wszystkie operacje zamówienia na tym połączeniu. Obsługa zachowuje także ewentualny błąd rollback().

```
c.setAutoCommit(false);
try { saveOrder(c); c.commit(); }
catch (SQLException | RuntimeException e) {
  try { c.rollback(); } catch (SQLException rb) {
    e.addSuppressed(rb); }
  throw e;  }
```

04 Nie zatwierdzaj sprzątaniem

Zmiana autocommit z false na true może zatwierdzić aktywną transakcję. Nie wykonuj jej bezwarunkowo w finally po nieudanym commit lub rollback.



05 Rozpoznaj niepewny wynik

Utrata łączności podczas commit może oznaczać, że serwer zatwierdził zmiany, lecz odpowiedź nie dotarła. Nie ponawiaj ślepo zakupu; najpierw ustal wynik operacji.



06 Zwolnij bezpiecznie

Po zakończeniu pracy zamknij uchwyt. Stan połączenia resetuje właściwie skonfigurowana pula; po uszkodzeniu lub niepewnym stanie zastosuj jej mechanizm unieważniania połączeń.



07 Sprawdź się

Dwa DAO pobrały osobne połączenia z tej samej puli. Czy rollback pierwszego cofnie drugpił?

Odpowiedź

Nie. Są to odrębne sesje i odrębne lokalne transakcje.



PUŁAPKA

Rollback po błędzie commit nie jest dowodem, że wcześniejszy commit nie nastąpił.

ZAPAMIĘTAJ

- Wspólna transakcja wymaga wspólnej sesji oraz jednego właściciela decyzji.



Niepewność wyniku wymaga sprawdzenia, nie zgadywania.

SQL

POZIOMY IZOLACJI i ANOMALIE

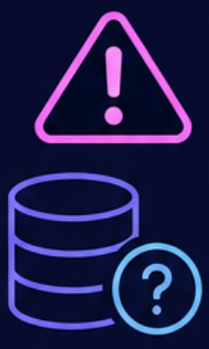
JAVA + MYSQL W PRAKTYCE

Wybieraj gwarancję dla konkretnego rodzaju odczytu



01 Nazwij ryzyko

- **Brudny odczyt** widzi niezatwierdzone zmiany.
- **Niepowtarzalny odczyt** daje inną wartość tego samego wiersza.
- **Fantom** zmienia zestaw wierszy pasujących do tego samego warunku.



02 READ UNCOMMITTED

Może ujawniać niezatwierdzone dane innych transakcji. Odczytany stock może później zostać wycofany, dlatego nie traktuj takiego odczytu jako pewnej podstawy decyzji.



03 READ COMMITTED

Zwykły odczyt spójny dostaje nowy obraz danych dla każdego zapytania. Dwa SELECT w jednej transakcji mogą zobaczyć różne zatwierdzone wartości i nowe wiersze.



04 REPEATABLE READ

To domyślny poziom InnoDB. Zwykłe odczyty spójne korzystają z obrazu ustalonego przez pierwszy taki odczyt; transakcja widzi też własne zmiany. Blokujące odczyty działają inaczej.



05 SERIALIZABLE

Zapewnia silniejsze uporządkowanie kosztem ograniczenia współbieżności. W InnoDB przy wyłączonym autocommit zwykłe SELECT otrzymują semantykę FOR SHARE, co zwiększa blokowanie.



06 Sprawdź zachowanie

Fragment przed rozpoczęciem transakcji. InnoDB ma mechanizmy ponad minimalne gwarancje JDBC, w tym blokady next-key. Oceniaj równoległy scenariusz, a nie samą nazwę poziomu.

```
c.setTransactionIsolation(
    Connection.TRANSACTION_READ_COMMITTED);
c.setAutoCommit(false);
```

07 Sprawdź się

Przy READ COMMITTED inna sesja zatwierdziła zmianę ceny między dwoma SELECT. Czy wynik może się zmienić?

Odpowiedź

Tak. Kolejny zwykły odczyt spójny korzysta z nowego obrazu danych.

PUŁAPKA

Stabilny snapshot nie jest blokadą zapasu; regułę rezerwacji zabezpiecz odpowiednim zapisem lub blokowaniem.

ZAPAMIĘTAJ

- Testuj poziom izolacji razem z używanymi zapytaniami i transakcjami.

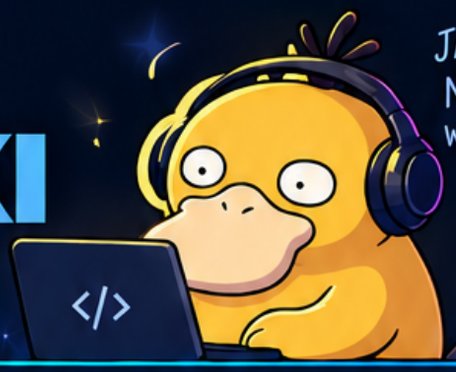


Silniejsza izolacja ma koszt współbieżności.

SQL

BLOKADY i DEADLOCKI

Krótko blokuj, rozpoznawaj konflikt, ponawiaj świadomie



01 Odczyt przed zmianą

`SELECT ... FOR UPDATE` blokuje wybrane rekordy indeksu dla późniejszej zmiany. Wykonuj go w jawnej transakcji. Zwykły `SELECT` oparty na MVCC nie daje takiej ochrony.



02 Zablokuj konkretny produkt

Fragment SQL wewnątrz transakcji z wyłączonym autocommit. Parametr wskazuje istniejący klucz główny products.id; zakończenie transakcji zwalnia blokadę.

```
SELECT stock
FROM products
WHERE id = ?
FOR UPDATE;
```

03 Cykl oczekiwania

Sesja A blokuje produkt 1 i czeka na 2. Sesja B blokuje 2 i czeka na 1. Przy wykrywaniu deadlocków InnoDB wybiera ofiarę i wycofuje jej transakcję.



04 Ogranicz konflikty

Modyfikuj produkty w stałej kolejności identyfikatorów. Dodaj właściwe indeksy, skróć transakcje i nie czekaj w nich na użytkownika ani wywołanie zewnętrznej usługi.



05 Ponów całą jednostkę

Po potwierdzonym deadlocku wykonaj ograniczoną liczbę prób całej transakcji z przerwą i losowym rozrzutem. Nie ponawiaj jedynie ostatniego UPDATE; wcześniejsze zmiany zostały cofnięte.



06 Odróżnij timeout

Timeout oczekiwania na blokadę domyślnie cofa tylko instrukcję, nie całą transakcję. Aplikacja powinna jawnie wycofać jednostkę przed ponowieniem. Diagnozuj przez `SHOW ENGINE INNODB STATUS`.



07 Sprawdź się

Czy zwiększenie timeoutu usuwa cykl A → B → A?

Odpowiedź

Nie. Cykl wymaga przerwania jednej transakcji; dłuższe czekanie nie zmienia zależności.

PUŁAPKA

Próba ponowienia musi być bezpieczna dla skutków zewnętrznych; nie wysyłaj dwukrotnie tego samego zamówienia.

ZAPAMIĘTAJ

- Deadlock jest przewidywalnym konfliktem współbieżności, który aplikacja musi umieć obsłużyć.



Zapobiegaj, diagnozuj i ponawiaj z limitem.

SQL INJECTION i DYNAMICZNE ZAPYTANIA

JAVA + MYSQL W PRAKTYCE

Oddziel dane od składni, także gdy użytkownik wybiera filtry i sortowanie.



01 Zagrożenie

Tekst użytkownika doklejony do SQL może zmienić warunek albo strukturę polecenia. Walidacja długości i formatu pomaga, lecz nie zastępuje wiązania parametrów.



02 Wartości przez parametry

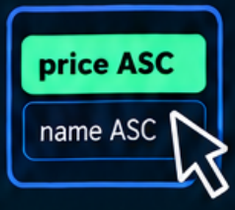
Założenie: używasz PreparedStatement. W poniższym SQL związ email przez setString(1, email). Apostrof w adresie pozostaje częścią wartości, zamiast zamykać literal SQL.



```
SELECT id, name FROM customers
WHERE email = ?
```

03 Struktura z listy

Parametr ? nie reprezentuje nazwy kolumny ani kierunku sortowania. Wybór użytkownika mapuj na zamknięty zestaw własnych fragmentów, na przykład price ASC lub name ASC.



04 Filtry opcjonalne

Buduj WHERE wyłącznie ze stałych fragmentów. Razem z każdym dodanym warunkiem zapamiętaj wartość i jej kolejność, aby indeksy parametrów odpowiadały znakom zapytania.



05 Lista identyfikatorów

Dla trzech produktów utwórz IN (?, ?, ?) i związ trzy wartości. Ogranicz długość listy. Dla pustej listy zwróć pusty wynik bez generowania IN ().



06 Sprawdź efekt

Testuj apostrof, komentarz SQL i tekst OR 1=1 jako zwykłe dane. Wynik powinien zależeć wyłącznie od dopasowania wartości, nigdy od dostarczonej składni.



07 Sprawdź się

Czy ORDER BY ? z parametrem „price” wybiera kolumnę price?

Odpowiedź **Nie.** Identyfikator wybierz z listy dozwolonych fragmentów kodu.

PUŁAPKA

Ręczne zamienianie apostrofów jest kruche i nie zabezpiecza dynamicznych identyfikatorów.

ZAPAMIĘTAJ

- Wartości wiąż; strukturę wybieraj spośród znanych wariantów.



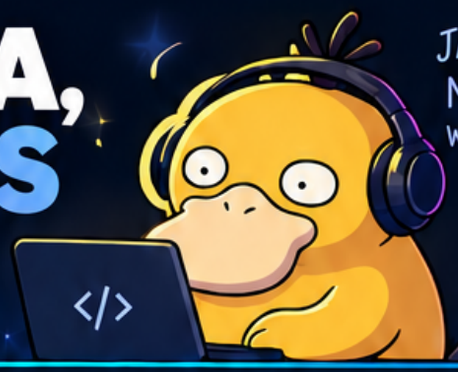
Bezpieczna dynamika ma ograniczoną gramatykę.

SQL

UPRAWNIENIA, SEKRETY i TLS

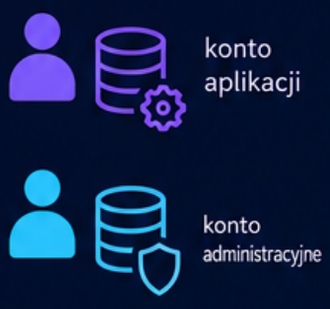
JAVA + MYSQL W PRAKTYCE

Zabezpiecz konto aplikacji, transport oraz sposób przekazywania konfiguracji.



01 Oddziel tożsamości

Aplikacja działa na dedykowanym koncie, a migracje na osobnym koncie administracyjnym. Ogranicz również hosty, z których konto aplikacji może się łączyć.



02 Najmniejsze uprawnienia

Przyznaj tylko operacje potrzebne konkretnym tabelom: odczyt katalogu, zapis zamówień i aktualizację zapasu. Konto obsługujące zakupy nie potrzebuje DROP, GRANT ani administracji serwerem.



03 Sekret poza kodem

Hasło pobieraj z kontrolowanej konfiguracji wdrożenia lub magazynu sekretów.
Nie umieszczaj go w repozytorium, obrazie kontenera, adresie JDBC ani komunikacie błędu.



04 Zweryfikuj serwer

Założenie: serwer obsługuje TLS, certyfikat obejmuje używaną nazwę hosta, a klient ufa jego CA. Ustaw właściwość Connector/J wymuszającą także weryfikację tożsamości.



```
sslMode=VERIFY_IDENTITY
```

05 Łańcuch zaufania

Skonfiguruj zaufany certyfikat CA w magazynie zaufania używanym przez sterownik. Samo szyfrowanie nie potwierdza tożsamości serwera; REQUIRED nie zastępuje VERIFY_IDENTITY.



06 Próba negatywna

W środowisku testowym sprawdź odrzucenie błędnej nazwy hosta i niezaufanego CA. Zaplanuj rotację hasła oraz certyfikatu i sprawdź ponowne zestawianie połączeń.



07 Sprawdź się

Połączenie działa dopiero po wyłączeniu weryfikacji. Czy to naprawa?

Odpowiedź: Nie. Sprawdź nazwę hosta, ważność certyfikatu i zaufany łańcuch CA.

PUŁAPKA

Nie rozwiązuj błędów TLS przez `sslMode=DISABLED` ani bezrefleksyjne zaufanie dowolnemu certyfikatowi.

ZAPAMIĘTAJ

- Minimalne prawa + kontrolowany sekret + zweryfikowany TLS.



Sprawdź również scenariusz odmowy połączenia.

SQL

PULA POŁĄCZEŃ i TIMEOUTY

JAVA + MYSQL W PRAKTYCE

Ogranicz koszt sesji i czas oczekiwania na każdym etapie operacji.

01 Co daje pula

Pula utrzymuje fizyczne połączenia i udostępnia je kolejnym operacjom. DataSource jest interfejsem pozyskiwania połączeń; sama jego obecność nie oznacza, że skonfigurowano pulę.



02 Pożycz na krótko

Pobierz Connection tuż przed pracą z bazą. Zamknij je przez try-with-resources po commit albo rollback. W puli zamknięcie logicznego uchwytu zwykle oddaje sesję.



03 Trzy różne oczekiwania

Rozróżniaj oczekiwanie na wolne połączenie w puli, zestawienie połączenia sieciowego i wykonanie zapytania. Każdy limit dotyczy innego etapu i wymaga oddzielnego ustawienia.



04 Jednostki mają znaczenie

Connector/J podaje connectTimeout i socketTimeout w milisekundach. Statement.setQueryTimeout przyjmuje sekundy. Fragment zakłada otwarty PreparedStatement ps; to limit zapytania, nie całego żądania.



```
ps.setQueryTimeout(3);
```

05 Dobierz pojemność

Mierz aktywne połączenia, kolejkę oczekujących i czas trzymania sesji. Uwzględnij sumę pul wszystkich instancji aplikacji oraz rezerwę na administrację bazy danych.



06 Obsłuż przerwanie

Po timeoutach zakończ transakcję zgodnie z polityką błędów. Zerwane połączenie może pozostawić nieznany wynik commit; samo przekroczenie czasu nie dowodzi braku zapisu.



07 Sprawdź się

Pula jest pełna. Czy większy socketTimeout udostępni wolną sesję?

Odpowiedź

Nie. To limit operacji sieciowych, a nie oczekiwania na pulę.

PUŁAPKA

Zbyt duża pula może zwiększyć przeciążenie zamiast je usunąć.

ZAPAMIĘTAJ

- Mierz kolejkę, czas zapytania
- i czas zajęcia połączenia osobno.

Krótka transakcja szybciej oddaje wspólny zasób.

SQL

STRONICOWANIE I DUŻE ZBIORY DANYCH

JAVA + MYSQL W PRAKTYCE

Czytaj porcjami, zachowując jednoznaczny porządek oraz świadomy model spójności.



01 Porządek jest obowiązkowy

Każda strona potrzebuje ORDER BY. Przy sortowaniu po cenie dodaj id jako rozstrzygnięcie remisów; sama cena nie określa kolejności produktów o identycznej wartości.

02 Prosty OFFSET

LIMIT z OFFSET jest wygodny dla małych list. Duży offset oznacza jednak pomijanie wielu wierszy; równoczesne zmiany danych mogą przesuwać granice stron.

03 Kursor po kluczu

Założenie: dodatnie id, sortowanie rosnące i PreparedStatement. Związ ostatnie id oraz ograniczony rozmiar strony; dla pierwszej strony użyj granicy 0.

```
SELECT id, name, price FROM products
WHERE id > ?
ORDER BY id LIMIT ?
```

04 Odczyt partiami

Dla eksportu Connector/J umożliwia useCursorFetch=true i dodatnie setFetchSize. Samo ustawienie wielkości pobrania bez odpowiedniej konfiguracji nie gwarantuje ograniczenia buforowania po stronie klienta.

05 Pamięć aplikacji

Przetwarzaj wiersze na bieżąco i zamykaj ResultSet. Jeśli każdy rekord dodasz do jednej ogromnej listy, odzyskasz problem pamięci mimo pobierania partiami.

06 Spójność wyniku

Kursor po id ogranicza przesuwanie granic, ale nie tworzy migawki danych. Dla eksportu wymagającego jednego obrazu uzgodnij izolację i koszt długiej transakcji.

07 Sprawdź się

Ostatnie id wynosi 42. Czy kolejna strona zaczyna się od id >= 42?

Odpowiedź Nie. Użyj id > 42, aby nie powtórzyć ostatniego produktu.

PUŁAPKA

LIMIT bez jednoznacznego ORDER BY nie gwarantuje powtarzalnego podziału.

ZAPAMIĘTAJ

- Strona ogranicza wynik; pobieranie partiami ogranicza buforowanie.



Dobierz spójność do celu odczytu.

SQL

PROBLEM N+1 i POMIAR WYDAJNOŚCI

JAVA + MYSQL W PRAKTYCE

Najpierw policz komunikację z bazą, potem optymalizuj rzeczywiste wąskie gardło.



01 Rozpoznaj wzorzec

Pobranie 20 zamówień, a następnie osobne zapytanie o pozycje każdego zamówienia daje 21 zapytań. Czas przejść sieciowych może przeważyć nad samym wykonaniem SQL.



02 Pobierz relacje zbiorczo

Dla małego wyniku rozważ JOIN. Pamiętaj, że jedno zamówienie wystąpi w wielu wierszach, po jednym dla każdej pozycji; mapper musi je pogrupować.



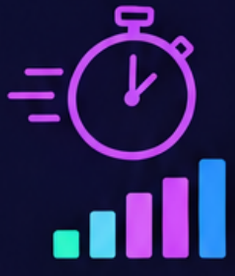
03 Stronicuj rodziców

Dla listy zamówień pobierz najpierw stronę nagłówków, potem wszystkie ich pozycje jednym zapytaniem IN. LIMIT na złączeniu może uciąć pozycje jednego zamówienia.



04 Mierz całą operację

Zapisz liczbę SQL, czas pobrania połączenia, wykonania oraz mapowania. Porównuj te same dane i obciążenie; uwzględnij rozgrzewkę oraz percentyle, nie tylko średnią.



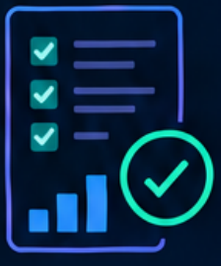
05 Czytaj plan

EXPLAIN pokazuje plan. **EXPLAIN ANALYZE** naprawdę wykonuje polecenie. Przykład uruchamiaj na kontrolowanych danych; porównaj szacunki z rzeczywistymi wierszami i liczbą iteracji.

```
EXPLAIN ANALYZE
SELECT id, name FROM products
WHERE price < 50.00;
```

06 Potwierdź poprawę

Test regresji powinien ograniczać liczbę zapytań przy rosnącej liczbie zamówień. Sprawdź także kompletność pozycji oraz zużycie pamięci po zmianie sposobu pobierania.



07 Sprawdź się

Ile zapytań wymaga strona nagłówków i jeden zbiorczy odczyt pozycji?

Odpowiedź Dwa, niezależnie od liczby nagłówków w tej ograniczonej stronie.

✕ PUŁAPKA

Jedno ogromne złączenie może być droższe niż dwa rozsądnie ograniczone zapytania.

✓ ZAPAMIĘTAJ

- Mniej przejść sieciowych, sprawdzony plan, poprawny wynik.



Optymalizacja zaczyna się od pomiaru.

SQL

SQLException i SQLState

JAVA + MYSQL W PRAKTYCE

Rozpoznaj kategorię awarii i świadomie zdecyduj o wycofaniu oraz ponowieniu.

01 Odczytaj metadane

SQLException udostępnia komunikat, SQLState oraz kod producenta. Fragment zakłada przechwycony wyjątek ex; SQLState może być null, więc sprawdź go przed analizą prefiksu.

```
String state = ex.getSQLState();
int vendorCode = ex.getErrorCode();
```

02 Klasy błędów

Pierwsze dwa znaki SQLState oznaczają klasę, 08 dotyczy połączeń, 23 integralności, 40 wycofania transakcji. To wskazówki do klasyfikacji, a nie uniwersalna zgoda na retry.

08
23
40

03 Szczegóły MySQL

Deadlock InnoDB: kod 1213, SQLState 40001. Serwer wycofuje transakcję ofiary. Przekroczenie oczekiwania na blokadę domyślnie wycofuje tylko bieżące polecenie, więc obsłuż resztę transakcji.

04 Zachowaj przyczynę

Przejrzyj `getNextException()` oraz wyjątki tłumione przy zamykaniu zasobów. Jeśli rollback także zawiedzie, zachowaj pierwotny błąd i dołącz dodatkowy, zamiast go zastępować.

05 Ponów całą operację

Po potwierdzonym wycofaniu przejściowego konfliktu można ponowić całą transakcję, z limitem prób i opóźnieniem. Nie ponawiaj samej ostatniej pozycji zamówienia.

06 Nieznany commit

Utrata połączenia podczas commit może pozostawić niepewny wynik. Bez mechanizmu idempotencji lub uzgodnienia stanu automatyczne powtórzenie zamówienia grozi podwójnym zapisem.

07 Sprawdź się

Czy naruszenie unikalności email naprawi pięć identycznych ponowień?

Odpowiedź

Nie. Trzeba rozwiązać konflikt danych lub zmienić decyzję biznesową.

PUŁAPKA

Komunikat błędu może zawierać dane klienta; nie pokazuj go bezpośrednio użytkownikowi.

ZAPAMIĘTAJ

- Klasyfikuj → zakończ transakcję → zdecyduj o bezpiecznym retry.

Nie każdy SQLException jest przejściowy.

SQL

LOGOWANIE BEZ UJAWNIANIA DANYCH

JAVA + MYSQL W PRAKTYCE

Zbieraj informacje potrzebne do diagnozy bez kopiowania danych klientów i sekretów.



01 Nazwij operację

- Loguj stały identyfikator, na przykład orders.create, oraz losowy identyfikator korelacji żądania. Pozwala to połączyć zdarzenia bez używania adresu email jako klucza diagnostycznego.

02 Dobierz pola

- Przydatne pola to czas trwania, wynik, liczba wierszy, SQLState i kod błędu.
- Nie zapisuj haseł, tokenów, pełnego adresu JDBC ani wartości parametrów klienta.

03 Bezpieczny przykład

- Poniższy zapis to wymyślony przykład formatu, nie wynik uruchomionego testu. Nazwy pól pozostają stałe, a tekst wejściowy użytkownika nie jest doklejany do komunikatu.

```
operation=orders.create outcome=rollback
sqlState=40001 vendorCode=1213
durationMs=34 correlationId=test-17
```

04 Kontroluj wyjątki

- Komunikat SQLException może zawierać wartość naruszającą ograniczenie, na przykład email. Sam stos wywołań nie jest gwarancją bezpieczeństwa: rejestrowany wyjątek zwykle zawiera również komunikat.

05 Sprawdź serwer

- Logi SQL oraz próbki Performance Schema też mogą ujawniać literały. Znormalizowany digest nie oznacza, że sąsiednie SQL_TEXT lub QUERY_SAMPLE_TEXT zostały pozbawione danych.

06 Weryfikuj politykę

- Ogranicz dostęp i czas przechowywania logów. Testem wywołaj kontrolowany błąd z charakterystycznym sekretem i sprawdź, czy nie pojawia się w żadnym wyjściu diagnostycznym.

07 Sprawdź się

Czy getMessage() można bez sprawdzania dopisać do odpowiedzi HTTP?

Odpowiedź Nie. Użytkownik otrzymuje bezpieczny komunikat i identyfikator korelacji.

PUŁAPKA

Maskowanie kilku cyfr lub znaków nie zawsze usuwa możliwość identyfikacji osoby.

ZAPAMIĘTAJ

- Loguj zachowanie systemu, a dane wejściowe ogranicz do minimum.

Diagnoza i prywatność mogą działać razem.

SQL

TESTY INTEGRACYJNE Z MySQL I TESTCONTAINERS

JAVA + MYSQL W PRAKTYCE

Sprawdź prawdziwy silnik, sterownik, migrację oraz zachowanie repozytorium.



01 Co testujesz

Test integracyjny uruchamia kod JDBC przeciw rzeczywistemu MySQL. Dzięki temu sprawdza dialekt, typy, ograniczenia i transakcje, których atrapa repozytorium nie potwierdzi.

02 Przygotuj środowisko

Potrzebujesz działającego środowiska kontenerowego zgodnego z Testcontainers, modułu MySQL i Connector/J. Ustal zgodne wersje zależności testowych; dokumentacja modułu opisuje wymagane artefakty.

03 Uruchom bazę testową

Wariant JDBC wymaga modułów Testcontainers JDBC/MySQL i sterownika na classpath. Ten specjalny URL tworzy kontener; nie wskazuje bazy produkcyjnej.

```
jdbc:tc:mysql:8.4:///shop
```

04 Odtwórz schemat

Uruchom te same migracje co w aplikacji, a potem minimalny zestaw danych. Tag 8.4 wybiera linię wersji; dla powtarzalnego CI przypnij konkretny obraz lub digest.

05 Sprawdź kontrakt

Zapisz produkt, odczytaj go przez repozytorium i porównaj cenę jako BigDecimal. Dodaj przypadek braku wyniku oraz naruszenia ograniczenia; samo pomyślne połączenie nie wystarcza.

06 Posprzątaj stan

Wybierz izolację na poziomie kontenera, schematu albo danych testu. Współdzielony kontener nie czyści tabel pomiędzy metodami; sprzątanie musi uwzględniać klucze obce.

07 Sprawdź się

Czy test przechodzący na H2 dowodzi zgodności z MySQL 8.4?

Odpowiedź Nie. Krytyczne zachowania należy sprawdzić na docelowym silniku.

✕ PUŁAPKA

Nie mieszaj automatycznego wariantu JDBC z ręcznym uruchamianiem drugiego kontenera dla tej samej bazy.

✓ ZAPAMIĘTAJ

- Prawdziwy MySQL + rzeczywiste migracje + izolowany zestaw danych.

To scenariusz do uruchomienia, nie deklaracja zaliczonych testów.

SQL

TESTOWANIE OGRANICZEŃ I ROLLBACKU

Udowodnij, że błędne dane nie zostają w bazie nawet po częściowym zapisie.



JAVA + MYSQL W PRAKTYCE

01 Zdefiniuj warunki

Założenia testu: tabele InnoDB, aktywne klucze obce, UNIQUE(email), wymagane NOT NULL oraz CHECK dla dodatniej quantity i nieujemnego stock. Schemat utwórz przed transakcją.

02 Jedna wada naraz

Przygotuj poprawny rekord bazowy, a następnie zmieniaj pojedynczy warunek: nieistniejący customer_id, powtórzony email albo quantity = 0. Dzięki temu znasz przyczynę odrzucenia.

03 Błąd po sukcesie

W jednej transakcji zapisz poprawny nagłówek orders i pierwszą pozycję. Druga pozycja powinna celowo naruszyć ograniczenie; ścieżka błędu wykonuje rollback całej operacji.

04 Sprawdź stan końcowy

Po zakończeniu transakcji odczytaj dane nowym połączeniem. Oczekuj braku nagłówka i obu pozycji oraz pierwotnego zapasu produktów, jeśli transakcja go zmniejszała.

05 Nie zakładaj automatyzmu

Odrzucone INSERT nie oznacza automatycznie wycofania wcześniejszych poleceń. Test ma sprawdzić obsługę transakcji w aplikacji, a nie tylko pojawienie się SQLException.

06 Odróżnij DDL

CREATE, ALTER i TRUNCATE mogą powodować niejawne commit. Nie używaj ich wewnątrz testowanej transakcji biznesowej ani jako dowodu, że rollback obejmie zmianę schematu.

DDL

07 Sprawdź się

Druga pozycja zawiodła, ale nagłówek pozostał. Czy test może przejść?

Odpowiedź: Nie. Naruszona została atomowość całego zamówienia.

✕ PUŁAPKA

CHECK przepuszcza wynik UNKNOWN, dlatego wymagane wartości zabezpiecz także NOT NULL.

✓ ZAPAMIĘTAJ

Wyjątek jest sygnałem; dowodem rollbacku jest stan bazy.

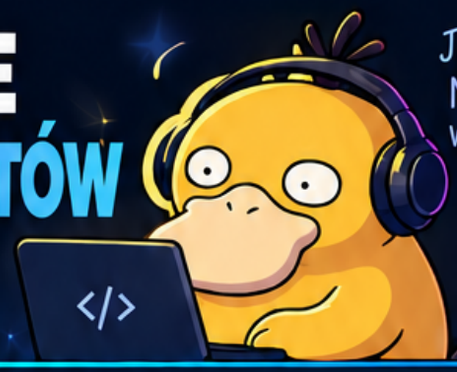
Sprawdzaj brak wszystkich skutków częściowego zapisu.

SQL

DANE BRZEGOWE i POWTARZALNOŚĆ TESTÓW

JAVA + MYSQL W PRAKTYCE

Dobry zestaw testów obejmuje granice domeny i nie zależy od przypadkowego środowiska.



01 Ilość i zapas

Dla stanu 5 sprawdź zakup 1, 5 oraz 6 sztuk. Osobno odrzuć quantity = 0 i wartości ujemne. Pusty koszyk także wymaga jednoznacznej decyzji.



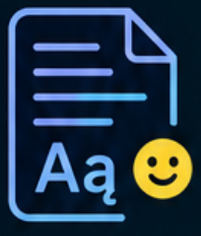
02 Pieniądze

Dla DECIMAL(10,2) testuj 0.00, 0.01 oraz granicę precyzji. Ustal politykę większej skali przed zapisem; nie pozwalaj, aby przypadkowe zaokrąglenie definiowało reguły biznesowe.



03 Tekst i NULL

Uwzględnij polskie znaki, emoji przy utf8mb4, apostrof, pusty tekst i graniczną długość. NULL jest innym przypadkiem niż pusty String; ograniczenia powinny odzwierciedlać tę różnicę.



04 Stały kontekst

Ustal SQL mode, kolację oraz strefę czasową sesji. Dla czasu tworzenia zamówienia użyj kontrolowanego zegara aplikacji albo asercji zakresu, zamiast dokładnego porównania do teraz.



05 Izolowane dane

Każdy test tworzy własne rekordy i zna ich identyfikatory. Nie zakładaj, że AUTO_INCREMENT zacznie od 1 ani że po rollbacku numer zostanie ponownie użyty.



06 Powtarzalna współbieżność

Dla rywalizacji o zapas użyj dwóch połączeń i jawnej synchronizacji startu. Oceniaj końcowe niezmienniki, a nie kolejność wątków lub szczęśliwie dobrany Thread.sleep.



07 Sprawdź się

Czy new BigDecimal("10.0").equals(new BigDecimal("10.00")) potwierdza równość?

Odpowiedź

Nie. equals uwzględnia skalę; do równości liczbowej użyj compareTo(...) == 0.

PUŁAPKA

Losowość bez zapisanego ziarna utrudnia odtworzenie awarii.

ZAPAMIĘTAJ

- Test granicy powinien wskazywać dokładnie jedną regułę domeny.



Powtarzalność zaczyna się od kontrolowanych założeń.

SQL

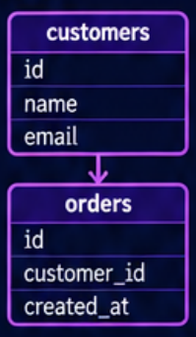
SCHEMAT BAZY i REGUŁY BIZNESOWE

JAVA + MYSQL W PRAKTYCE

Zaprojektuj konkretny zakup: klient wybiera produkty, a system utrwala cenę i zużywa zapas.

01 Klient i zamówienie

customers przechowuje id, name i email. **orders** zawiera id, customer_id oraz created_at. Klucz obcy łączy zamówienie z istniejącym klientem; wymagane pola mają **NOT NULL**.



02 Katalog produktów

products zawiera id, name, price i stock. Przyjmij **DECIMAL(10,2)** dla ceny oraz całkowity zapas. **CHECK** pilnuje **price >= 0** i **stock >= 0**.



03 Pozycja zakupu

order_items łączy order_id z product_id. Przyjmij klucz główny (**order_id**, **product_id**), dodatnią **quantity** i nieujemne **unit_price**. Powtórzenia produktu w koszyku połącz przed zapisem.



04 Cena historyczna

unit_price jest kopią ceny zaakceptowanej podczas zakupu. Późniejsza zmiana **products.price** nie zmienia wartości historycznego zamówienia. Sumę licz z **quantity × unit_price**, używając **BigDecimal**.



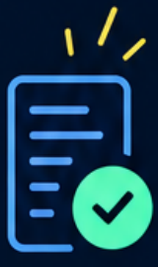
05 Ochrona relacji

Klucze obce zabezpieczają powiązania pozycji z zamówieniami i produktami. Dla tego projektu przyjmij **RESTRICT** przy usuwaniu używanych rekordów, aby przypadkiem nie skasować historii sprzedaży.



06 Granica operacji

Jedna transakcja **InnoDB** obejmuje nagłówek, wszystkie pozycje i zmniejszenie zapasu. Pusty koszyk odrzuć przed rozpoczęciem; brak dowolnego produktu lub zapasu przerywa całość.



07 Sprawdź się

Produkt kosztował 20.00, a dziś kosztuje 25.00. Jak wycenić starą pozycję?

Odpowiedź

Zapisanym **order_items.unit_price**, czyli 20.00 za sztukę.

PUŁAPKA

Sam klucz obcy nie wymusza, aby zamówienie zawierało przynajmniej jedną pozycję.

ZAPAMIĘTAJ

- Relacje chroni schemat; kompletność zakupu chroni transakcja.



Masz kontrakt dla pięciu końcowych map.

SQL

WYSZUKIWANIE PRODUKTÓW

JAVA + MYSQL W PRAKTYCE

Zbuduj ograniczone wyszukiwanie prefiksowe, które prowadzi do poprawnie zweryfikowanego koszyka.



01 Kontrakt wyszukiwania

Użytkownik podaje początek nazwy i maksymalną cenę. Ustal limit długości tekstu, zakres ceny oraz rozmiar strony od 1 do 100 produktów.

02 Dosłowny prefiks

Założenie: term to zweryfikowany, niepusty String. Dla ESCAPE ! najpierw zabezpiecz wykrzykniki, potem symbole wzorca.

```
String pattern = term.replace("!", "!!")
    .replace("%", "!%").replace("_", "!_") + "%";
```

03 Zapytanie strony

SQL dla PreparedStatement: zwiąż pattern, maksymalną cenę, ostatnie id oraz limit. Pierwsza strona używa lastId = 0; zakładamy dodatnie identyfikatory.

```
SELECT id, name, price, stock FROM products
WHERE name LIKE ? ESCAPE '!'
  AND price <= ? AND stock > 0 AND id > ?
ORDER BY id LIMIT ?
```

04 Znaczenie dopasowania

Kolacja wpływa na rozróżnianie wielkości liter i akcentów. Sprawdź oczekiwania dla polskich znaków; różne kolacje mogą dawać różne dopasowania.

05 Przekaż tylko wybór

Do koszyka trafiają product_id i quantity. Wyświetlona cena oraz zapas są informacyjne; serwer ponownie odczytuje cenę i sprawdza dostępność wewnątrz transakcji zakupu.

06 Sprawdź wydajność

Porównaj plan dla typowych prefiksów i pustego wyniku. Indeks pomagający filtrować nazwę nie musi równocześnie obsługiwać sortowania po id; decyzję oprzyj na pomiarze.

07 Sprawdź się

Klient wpisuje 50%. Czy procent ma pasować do dowolnego ciągu?

Odpowiedź

Nie w tym kontrakcie. Pattern zawiera 50!%%: dosłowny procent i końcowy wildcard.

PUŁAPKA

Parametryzacja SQL sama nie wyłącza specjalnego znaczenia % i _ w LIKE.

ZAPAMIĘTAJ

- Ogranicz wynik i ponownie sprawdź warunki przy zakupie.

Wynik wyszukiwania nie rezerwuje towaru.

SQL

ZAPIS ZAMÓWIENIA i JEGO POZYCJI

JAVA + MYSQL W PRAKTYCE

Przeprowadź zakup na jednym połączeniu i zatwierdź dopiero kompletny wynik.



01 Przygotuj koszyk

Przed transakcją odrzuć pusty koszyk, niepoprawne ilości i przepełnienie sumy. Połącz powtarzające się produkty, następnie uporządkuj je po product_id dla spójnej kolejności blokad.

02 Otwórz transakcję

- Założenia: tabele InnoDB, jedno sprawne Connection, wyłączony autoCommit i istniejący klient.
- Wszystkie repozytoria dostają to samo połączenie; żadne nie wykonuje własnego commit.

03 Zapisz nagłówek

- PreparedStatement dla INSERT przygotuj z RETURN_GENERATED_KEYS.
- Zwiąż customer_id oraz czas.
- Po udanym zapisie odczytaj wygenerowane id przez getGeneratedKeys() i sprawdź next().

```
INSERT INTO orders (customer_id, created_at)
VALUES (?, ?)
```

04 Ustal cenę zakupu

- Dla każdego produktu odczytaj price przez SELECT ... WHERE id = ? FOR UPDATE.
- Brak wiersza kończy operację błędem.
- Blokada chroni odczytaną cenę do końca transakcji.

05 Zapisz pozycję

- Na tym samym połączeniu zmniejsz zapas warunkowym UPDATE z następnej mapy.
- Wymagaj jednego wiersza, potem zapisz pozycję z otrzymanym order_id i odczytaną ceną.

```
INSERT INTO order_items
(order_id, product_id, quantity, unit_price)
VALUES (?, ?, ?, ?)
```

06 Zamknij całość

- Commit wykonaj po wszystkich pozycjach.
- Przy błędzie wycofaj całą transakcję, także zmiany zapasu.
- Zasoby zamknij, a połączenie oddaj z poprawnie zakończonym stanem transakcji.

07 Sprawdź się

Czy repozytorium pozycji może otworzyć własne Connection?

Odpowedź

Nie w tym przepływie. Oddzielna sesja nie należy do tej samej lokalnej transakcji.

✕ PUŁAPKA

Nie wyliczaj nowego order_id przez SELECT MAX(id); równoległy zakup może go zmienić.

✓ ZAPAMIĘTAJ

- Nagłówek → cena i zapas → pozycje → jeden commit.



Sukces oznacza kompletne zamówienie.

SQL

AKTUALIZACJA STANU MAGAZYNOWEGO

JAVA + MYSQL W PRAKTYCE

Podejmij decyzję o sprzedaży atomowo, również gdy dwa zamówienia konkurują o ostatnie sztuki.



01 Warunek wejściowy

Fragment zakłada quantity > 0, istniejące ograniczenia schematu i aktywną transakcję zamówienia na jednym Connection. Nie wykonuj go w oddzielnym autoCommit.

02 Jedno polecenie

PreparedStatement wiąże kolejno quantity, productId, quantity. Warunek zapasu i jego zmiana należą do jednego UPDATE, więc decyzja nie opiera się na starym odczycie.

```
UPDATE products
SET stock = stock - ?
WHERE id = ? AND stock >= ?
```

03 Sprawdź liczbę

executeUpdate() musi zwrócić 1. Dodatnia ilość rzeczywiście zmienia wiersz. Wynik 0 oznacza niedostateczny zapas albo brak produktu; aplikacją przerywa wtedy całe zamówienie.

04 Dwa zakupy

Dla stock = 5 dwa równoległe zamówienia po 4 sztuki nie mogą oba zakończyć się sukcesem. Aktualizacje konkurują o blokadę wiersza; po udanym zakupie zostaje 1.

05 Kilka produktów

Przetwarzaj produkty w stałej kolejności id i krótko trzymaj blokady. Jeśli zabraknie ostatniej pozycji, rollback przywraca również zapasy wcześniej obsłużonych produktów i usuwa nagłówek.

06 Konflikt przejściowy

Deadlock pozostaje możliwy mimo starannej kolejności. Po wycofaniu ofiary można ponowić całą transakcję z ograniczeniem prób; nie powtarzaj bezmyślnie pojedynczego zmniejszenia stanu.

07 Sprawdź się

Stan wynosi 3, a zamówienie żąda 4. Jaki wynik UPDATE jest oczekiwany?

Odpowiedź

0 wierszy; stan pozostaje 3, a zamówienie należy wycofać.

PUŁAPKA

Odczyt stock, obliczenie w Javie i bezwarunkowe UPDATE mogą zgubić równoległą zmianę.

ZAPAMIĘTAJ

- Warunek + zmiana + liczba wierszy + wspólna transakcja.

Nie obiecuj towaru przed potwierdzonym commit.

SQL

Test całego procesu i checklista wdrożenia

JAVA + MYSQL W PRAKTYCE

Zamknij projekt dowodami poprawności, a potem sprawdź gotowość do bezpiecznego uruchomienia.



01 Scenariusz sukcesu

Dane testowe: produkt A kosztuje 12.50 i ma 5 sztuk; B kosztuje 7.00 i ma 4. Zakup 2 × A oraz 1 × B daje 32.00.



02 Sprawdź zapis

Po commit nowe połączenie powinno znaleźć jeden nagłówek, dwie pozycje, ceny 12.50 i 7.00 oraz zapasy 3 i 3. Zmiana ceny katalogowej nie zmienia zapisanych pozycji.



03 Scenariusz odmowy

Powtórz test, żądając niedostępnej ilości B po przetworzeniu A. Oczekuj braku nowego zamówienia i pozycji oraz niezmienionych zapasów obu produktów po rollback.



04 Wyścig o towar

Dwa połączenia kupują po 4 sztuki produktu ze stanem 5. Zsynchronizuj start, poczekaj na zakończenie obu prób i sprawdź: jeden sukces, jeden brak zapasu, stan 1.



05 Konfiguracja wdrożenia

Zweryfikuj migracje, kopię zapasową i odtworzenie, minimalne prawa konta, zewnętrzny sekret oraz VERIFY_IDENTITY. Sprawdź limity puli, timeouty i brak danych klientów w logach.



06 Granice projektu

Obecny schemat nie ma klucza idempotencji. Przed automatycznym ponawianiem niepewnego commit dodaj trwałą identyfikację żądania i unikalność; samo order_id tego problemu nie rozwiązuje.



07 Sprawdź się

Czy poprawna odpowiedź aplikacji wystarcza do zaliczenia testu zakupu?

Odpowiedź Nie. Sprawdź trwałe wiersze, historyczne ceny i końcowy zapas osobnym odczytem.

✕ PUŁAPKA

To plan weryfikacji. Oznacz test jako zaliczony dopiero po rzeczywistym uruchomieniu i sprawdzeniu asercji.

✓ ZAPAMIĘTAJ

- Poprawny zakup jest atomowy,
- odporny na wyścig i możliwy do zdiagnozowania.



Od mapy myśli przejdź do sprawdzonej implementacji.