

Materiały do przedmiotu:

Bazy danych

Laboratorium nr 6:

Tworzenie wyzwalaczy automatyzujących operacje oraz widoków ułatwiających dostęp do danych. Tworzenie i zarządzanie procedurami w bazie danych uwzględniając wymagania bezpieczeństwa danych, filtrowania i zapewnienia kontroli dostępu do nich

Autor:

Albert Rachwał



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Spis treści

Wprowadzenie, tematyka zajęć	3
Cel ćwiczenia/laboratorium	3
Instrukcja laboratoryjna/ćwiczeniowa	4
Ćwiczenia samodzielne	24
Pytania pomocnicze	26
Podsumowanie	26
Literatura	27



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Wprowadzenie, tematyka zajęć

Zaawansowane mechanizmy baz danych pozwalają na automatyzację działań oraz zwiększenie bezpieczeństwa i wygody w korzystaniu z systemu. Wyzwalacze umożliwiają wykonywanie określonych akcji w reakcji na zdarzenia w bazie. Widoki ułatwiają dostęp do danych, jednocześnie chroniąc ich integralność. Procedury i funkcje przechowywane w bazie pozwalają zarządzać logiką biznesową z poziomu bazy danych, zapewniając większą kontrolę nad procesami oraz bezpieczeństwem.

Cel ćwiczenia/laboratorium

- Poznanie pojęcia wyzwalacza (triggera).
- Rozróżnianie rodzajów wyzwalaczy (BEFORE, AFTER, INSTEAD OF).
- Zrozumienie zastosowań wyzwalaczy w praktyce.
- Tworzenie przykładowych wyzwalaczy.
- Poznanie pojęcia widoku.
- Tworzenie widoków prostych i złożonych.
- Zrozumienie roli widoków w bezpieczeństwie danych.
- Poznanie pojęcia procedury składowanej.
- Tworzenie prostych procedur w SQL.
- Poznanie pojęcia funkcji składowanej.
- Wykorzystywanie procedur i funkcji do automatyzacji operacji.
- Zrozumienie, jak kontrolować dostęp do danych przez widoki i procedury.
- Ćwiczenie tworzenia filtrów w widokach.
- Omówienie znaczenia logiki biznesowej w bazie danych.
- Analiza zagrożeń i ryzyk związanych z wyzwalaczami i procedurami.

Instrukcja laboratoryjna/ćwiczeniowa

Widoki w bazach danych

Widok (ang. view) to obiekt bazy danych, który można traktować jako zapisane zapytanie SQL. W praktyce nie przechowuje on sam w sobie danych, lecz definiuje, w jaki sposób mają one zostać pobrane z tabel lub innych widoków. Dzięki temu użytkownik pracujący z widokiem ma wrażenie, jakby korzystał z „wirtualnej tabeli”, która zawiera interesujący go fragment danych.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Widoki zostały wprowadzone, aby uprościć dostęp do danych, poprawić bezpieczeństwo oraz zwiększyć czytelność i spójność pracy z bazą. Dla wielu osób pracujących na co dzień z bazą, widok jest jak filtr albo gotowy raport – ktoś raz opracował zapytanie, a później inni użytkownicy mogą z niego korzystać, nie wnikając w szczegóły implementacji.

Główne zastosowania widoków

1. Uproszczenie złożonych zapytań

W praktyce biznesowej często mamy do czynienia z bardzo rozbudowanymi zapytaniami SQL, które łączą wiele tabel i stosują liczne filtry. Jeśli takie zapytanie jest często powtarzane, warto stworzyć widok – dzięki temu użytkownicy mogą odwoływać się do niego jak do tabeli, a logika zapytania jest ukryta w definicji widoku.

Przykład: zamiast pisać za każdym razem złożony JOIN pomiędzy tabelami Orders, Customers i Products, można utworzyć widok OrdersWithCustomers, który zwraca najważniejsze informacje o zamówieniach wraz z danymi klientów.

2. Zwiększenie bezpieczeństwa danych

Widok pozwala ograniczyć dostęp do wrażliwych informacji. Administrator może utworzyć widok, który ukrywa niektóre kolumny (np. numery PESEL czy dane finansowe) i nadać uprawnienia użytkownikom wyłącznie do tego widoku, a nie do całej tabeli. Dzięki temu studenci, analitycy czy pracownicy określonego działu widzą tylko to, co powinni.

3. Standaryzacja i spójność

Widoki są przydatne, gdy różne osoby muszą korzystać z tych samych definicji danych. Zamiast aby każdy pisał zapytanie na własną rękę, widok zapewnia jedną, wspólną wersję – np. widok ActiveEmployees zawsze zwróci tylko pracowników aktualnie zatrudnionych, niezależnie od tego, kto go wywoła.

4. Ułatwienie migracji i abstrakcji



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Widok może pełnić rolę warstwy pośredniej między logiką aplikacji a strukturą bazy. Jeśli struktura tabel zmieni się w przyszłości, wystarczy zmodyfikować definicję widoku, a programy korzystające z niego nadal będą działać tak samo.

Rodzaje widoków:

- Widoki proste – oparte na pojedynczej tabeli, często służące do ukrycia niektórych kolumn.
- Widoki złożone – bazujące na wielu tabelach i zawierające np. agregacje lub złączenia.
- Widoki aktualizowalne – w niektórych systemach baza pozwala modyfikować dane za pomocą widoku (np. INSERT/UPDATE/DELETE), ale tylko jeśli zapytanie definiujące widok spełnia pewne warunki (brak złożonych funkcji, agregacji czy DISTINCT).
- Widoki materializowane – występują w niektórych DBMS (np. Oracle, PostgreSQL). W odróżnieniu od zwykłych widoków przechowują fizycznie wynik zapytania, co przyspiesza dostęp, ale wymaga mechanizmu odświeżania danych.

Widoki a bezpieczeństwo

Jednym z najważniejszych atutów widoków jest to, że pozwalają na kontrolowanie, co użytkownik może zobaczyć. Przykładowo, zamiast dawać pracownikom działu HR pełny dostęp do tabeli Employees, można przygotować widok:

```
CREATE VIEW PublicEmployees AS  
SELECT id, name, department, hire_date  
FROM Employees;
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Dzięki temu pracownicy widzą podstawowe informacje o zatrudnionych, ale nie mają dostępu do kolumn salary czy pesel. W połączeniu z mechanizmami uprawnień (GRANT/REVOKE) widoki stają się potężnym narzędziem zarządzania bezpieczeństwem danych.

Dobre praktyki przy tworzeniu widoków:

- Widok sam w sobie nie przyspiesza działania – za każdym razem wynik jest obliczany na nowo (chyba że korzystamy z widoków materializowanych).
- Zbyt skomplikowane widoki (np. wielokrotne zagnieżdżenia JOIN i agregacji) mogą utrudniać optymalizację i prowadzić do spadku wydajności.
- Widoki nie zawsze są aktualizowalne – trzeba pamiętać, że czasem służą tylko do odczytu.
- Warto nadawać widokom nazwy opisowe (np. AktywneZamowienia zamiast v1) i dokumentować ich przeznaczenie.

Procedury składowane w bazach danych

Procedura składowana (ang. stored procedure) to blok kodu SQL zapisany i przechowywany w bazie danych, który można wielokrotnie wywoływać tak, jak wywołuje się funkcję w programowaniu. Różni się jednak od zwykłych zapytań tym, że jest trwale zapisana w serwerze DBMS, dzięki czemu wszyscy użytkownicy mający odpowiednie uprawnienia mogą korzystać z tej samej logiki biznesowej, bez konieczności powielania kodu w aplikacjach.

Celem procedur składowanych jest centralizacja i standaryzacja operacji wykonywanych na danych. Zamiast udostępniać aplikacjom bezpośredni dostęp do tabel i wymagać, by każda z nich samodzielnie pisała własne zapytania, można przygotować procedury realizujące określone zadania. Dzięki temu zmiany logiki biznesowej wprowadza się w jednym miejscu – w bazie – a nie w dziesiątkach fragmentów kodu aplikacyjnego.

Procedura może przyjmować parametry wejściowe (np. numer zamówienia), zwracać wartości wyjściowe (np. status realizacji), a także modyfikować dane w tabelach poprzez polecenia DML (INSERT, UPDATE, DELETE). Daje to dużą elastyczność –



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



można w niej zrealizować zarówno proste operacje techniczne (np. dodanie nowego klienta), jak i bardziej złożone procesy biznesowe (np. weryfikację limitu kredytowego i rejestrację zamówienia).

W praktyce procedury często używane są do:

- automatyzacji powtarzalnych operacji administracyjnych (np. archiwizacji danych),
- weryfikacji reguł biznesowych przed zatwierdzeniem danych,
- przygotowania raportów i przekształceń danych,
- obsługi złożonych transakcji wymagających kilku kroków, które muszą być wykonane razem.

Z punktu widzenia bezpieczeństwa procedury są również cennym narzędziem. Można nadać użytkownikowi uprawnienia do uruchamiania konkretnej procedury (EXECUTE), bez konieczności udzielania mu bezpośrednich praw do tabel, które ta procedura modyfikuje. Dzięki temu użytkownik realizuje swój cel, ale nie ma dostępu do wrażliwych danych poza tym zakresem.

Podstawowa składnia tworzenia procedury w MySQL/MariaDB wygląda następująco:

```
DELIMITER //
```

```
CREATE PROCEDURE DodajKlienta(IN imie VARCHAR(50), IN nazwisko VARCHAR(50))
```

```
BEGIN
```

```
    INSERT INTO Klienci (Imie, Nazwisko, DataUtworzenia)
```

```
    VALUES (imie, nazwisko, NOW());
```

```
END //
```

```
DELIMITER ;
```

Wywołanie procedury odbywa się za pomocą polecenia:

```
CALL DodajKlienta('Jan', 'Kowalski');
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Warto zauważyć, że procedura składowana nie zwraca wartości wprost tak jak funkcja. Może jednak zwracać dane poprzez parametry wyjściowe (OUT), parametry wejściowo-wyjściowe (INOUT) lub wynik zapytania SELECT wewnątrz procedury.

Dobre praktyki pracy z procedurami obejmują:

- nadawanie im jednoznacznych nazw opisujących cel (np. DodajZamowienie, RaportSprzedazyMiesiac),
- unikanie zbyt dużej logiki w jednej procedurze – lepiej dzielić ją na mniejsze, łatwiejsze do utrzymania fragmenty,
- dodawanie komentarzy i dokumentacji, aby inni użytkownicy wiedzieli, do czego procedura służy,
- świadome korzystanie z transakcji w obrębie procedury, tak aby całość była atomowa i zgodna z zasadami ACID,
- kontrolę uprawnień EXECUTE, by ograniczyć dostęp tylko do osób, które rzeczywiście powinny korzystać z danej procedury.

Procedury składowane stanowią fundament logiki biznesowej w wielu systemach – są elastyczne, bezpieczne i wydajne, a ich rola rośnie zwłaszcza tam, gdzie baza danych jest centralnym elementem dużego systemu informatycznego.

Definiowanie funkcji użytkownika

Oprócz procedur składowanych istotną rolę w logice bazodanowej odgrywają także funkcje systemowe oraz własne funkcje definiowane przez użytkownika (User-Defined Functions, UDF). Funkcje systemowe, takie jak NOW(), CONCAT() czy ROUND(), stanowią zestaw gotowych narzędzi, które można stosować bezpośrednio w zapytaniach, ułatwiając obliczenia i operacje na danych. Z kolei funkcje własne pozwalają programistom rozszerzyć możliwości DBMS, tworząc operacje dopasowane do specyficznych potrzeb organizacji – w odróżnieniu od procedur muszą one zawsze zwracać jedną wartość i mogą być wykorzystywane wprost w instrukcjach SELECT, WHERE czy ORDER BY. Ważnym aspektem bezpieczeństwa w pracy z procedurami, funkcjami i widokami jest określenie, w jakim kontekście uprawnień będą one wykonywane. W MySQL i MariaDB dostępne są dwa



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



tryby: DEFINER, w którym obiekt działa z uprawnieniami jego twórcy, oraz INVOKER, w którym obowiązują uprawnienia użytkownika wywołującego. Mechanizm ten pozwala ograniczyć dostęp do tabel i operacji – przykładowo, użytkownik może uruchomić procedurę, która modyfikuje dane, mimo że sam nie posiada bezpośrednich praw do tych tabel.

Kilka zasad tworzenia własnych funkcji:

- RETURNS musi określać typ zwracanej wartości (np. INT, DOUBLE, VARCHAR).
- DETERMINISTIC oznacza, że funkcja zawsze zwróci ten sam wynik dla tych samych argumentów (ważne dla optymalizacji).
- Funkcje użytkownika można wykorzystywać w zapytaniach SELECT, WHERE, HAVING, ORDER BY, ale nie mogą modyfikować danych (czyli brak INSERT/UPDATE/DELETE).

Przykład 1: Prosta funkcja matematyczna

Funkcja obliczająca pole prostokąta:

DELIMITER \$\$

```
CREATE FUNCTION pole_prostokata(a DOUBLE, b DOUBLE)
```

```
RETURNS DOUBLE
```

```
DETERMINISTIC
```

```
BEGIN
```

```
    RETURN a * b;
```

```
END$$
```

DELIMITER ;



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Użycie:

```
SELECT pole_prostokata(5, 10) AS wynik;
```

Przykład 2: Funkcja tekstowa

Funkcja, która zwraca inicjały użytkownika:

```
DELIMITER $$
```

```
CREATE FUNCTION inicjaly(imie VARCHAR(50), nazwisko VARCHAR(50))
```

```
RETURNS VARCHAR(10)
```

```
DETERMINISTIC
```

```
BEGIN
```

```
    RETURN CONCAT(LEFT(imie,1), '.', LEFT(nazwisko,1), '.');
```

```
END$$
```

```
DELIMITER ;
```

Użycie:

```
SELECT inicjaly('Jan', 'Kowalski') AS skrót;
```

Przykład 3: Funkcja warunkowa

Funkcja oceniająca, czy dana osoba jest pełnoletnia:

```
DELIMITER $$
```

```
CREATE FUNCTION czy_pelnoletni(wiek INT)
```

```
RETURNS VARCHAR(20)
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



DETERMINISTIC

BEGIN

IF wiek >= 18 THEN

RETURN 'TAK';

ELSE

RETURN 'NIE';

END IF;

END\$\$

DELIMITER ;

Użycie:

SELECT nazwisko, wiek, czy_pelnoletni(wiek) AS status

FROM osoby;

Wyzwalacze (triggery) w bazach danych

Wyzwalacz (ang. trigger) to specjalny obiekt bazy danych, który automatycznie uruchamia się w odpowiedzi na określone zdarzenie związane z tabelą lub widokiem. Takim zdarzeniem może być dodanie, usunięcie albo modyfikacja rekordu. Trigger działa „w tle” – użytkownik wykonuje swoje polecenie (np. INSERT), a baza sama wywołuje dodatkowe operacje, które zostały zdefiniowane w wyzwalaczu.

Można powiedzieć, że wyzwalacze są narzędziem do automatyzacji i kontroli: pilnują, aby pewne zasady biznesowe były zawsze respektowane, albo aby przy każdej zmianie danych uruchamiała się dodatkowa logika. Dzięki temu część logiki biznesowej przenosi się bezpośrednio do bazy, co zwiększa spójność i bezpieczeństwo.

Główne zastosowania wyzwalaczy



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



1. Zapewnienie spójności danych

Wyzwalacze mogą np. automatycznie sprawdzać poprawność wprowadzanych wartości. Jeśli ktoś spróbuje wstawić pensję ujemną albo datę zatrudnienia późniejszą niż dzisiaj – trigger może przerwać operację (ROLLBACK) i zwrócić błąd.

2. Automatyczne uzupełnianie danych

Przy dodawaniu nowego rekordu można automatycznie wypełnić pewne pola – np. ustawić bieżącą datę, wygenerować numer zamówienia czy zaktualizować licznik.

3. Prowadzenie logów (audyt)

Bardzo popularne jest stosowanie wyzwalaczy do tworzenia kopii zmian w tabeli. Każdy INSERT, UPDATE czy DELETE zapisuje się w dodatkowej tabeli, np. Orders_log, co pozwala później śledzić historię zmian.

4. Synchronizacja danych

W bardziej złożonych systemach trigger może powodować, że zmiana w jednej tabeli automatycznie aktualizuje dane w innej – np. usunięcie klienta skutkuje skasowaniem jego zamówień albo ustawieniem im statusu „nieaktywny”.

5. Realizacja reguł biznesowych

Triggery pozwalają wymusić zasady, które nie zawsze da się łatwo opisać constraintami (np. CHECK czy FOREIGN KEY). Przykład: „pracownik może mieć tylko jedno aktywne stanowisko w danym momencie”.

Rodzaje wyzwalaczy:

- BEFORE – uruchamiają się przed wykonaniem operacji (INSERT, UPDATE, DELETE). Można tu np. zmienić wartości wstawiane do tabeli albo przerwać operację.
- AFTER – wywoływane po zakończeniu operacji. Typowe dla logów i działań wtórnych (np. zapis historii, aktualizacja innej tabeli).



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



- **INSTEAD OF** – używane głównie na widokach; zamiast standardowej operacji można zdefiniować własną logikę, co pozwala np. modyfikować dane widoku poprzez odpowiednie zmiany w tabelach bazowych.

Triggery a wydajność i bezpieczeństwo

- Plusy: automatyzacja, spójność, przejrzystość reguł biznesowych.
- Minusy: trigger działa „w tle”, więc użytkownik nie zawsze wie, że coś jeszcze się uruchamia. Nadmiar wyzwalaczy może spowalniać operacje, a w skrajnych przypadkach powodować zapętlenia (trigger wywołuje trigger).
- Bezpieczeństwo: wyzwalacze pozwalają ograniczyć możliwość wprowadzania niepoprawnych danych i stanowią ważną warstwę ochronną, ale wymagają bardzo świadomego projektowania.

Wyzwalacze w praktyce

Przykład logowania zmian. Ten wyzwalacz zapisuje każdą zmianę pensji do osobnej tabeli logów:

```
CREATE TRIGGER log_update_salary  
AFTER UPDATE ON Employees  
FOR EACH ROW  
INSERT INTO Salary_Log(emp_id, old_salary, new_salary, changed_at)  
VALUES (OLD.id, OLD.salary, NEW.salary, NOW());
```

Wymuszanie wartości domyślnej. Tutaj trigger automatycznie uzupełnia datę zamówienia, jeśli użytkownik jej nie podał.

```
CREATE TRIGGER set_default_date  
BEFORE INSERT ON Orders
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



FOR EACH ROW

```
SET NEW.order_date = IFNULL(NEW.order_date, NOW());
```

Wyzwalacze mogą być łączone z innymi obiektami bazodanowymi:

- Z procedurami: trigger może wywoływać procedurę składowaną, dzięki czemu logika biznesowa jest centralnie zdefiniowana i łatwa do ponownego wykorzystania.
- Z widokami: w przypadku widoków używa się triggerów typu INSTEAD OF, aby umożliwić aktualizację danych widoku.
- Z transakcjami: wszystkie działania triggera dzieją się w ramach tej samej transakcji co operacja główna – jeśli trigger zgłosi błąd, cała transakcja jest cofana.

Dobre praktyki w pracy z wyzwalaczami

- Projektuj minimalistycznie – każdy trigger powinien realizować jedną jasno określoną funkcję (np. logowanie zmian pensji). Dzięki temu jest łatwiejszy do zrozumienia i utrzymania.
- Dokumentuj działanie – nazwa i komentarz przy wyzwalaczu powinny od razu mówić, co on robi (np. log_update_salary zamiast trigger1).
- Stosuj BEFORE i AFTER świadomie –
 - BEFORE nadaje się do walidacji i uzupełniania danych,
 - AFTER do logowania i działań wtórnych.
- Dbaj o wydajność – kod w triggerze powinien być jak najprostszy. Operacje złożone (np. raporty) lepiej przenieść do procedur.
- Kontroluj transakcje – pamiętaj, że trigger działa w ramach tej samej transakcji co polecenie wyzwalające; jeśli trigger zgłosi błąd, cała operacja zostanie cofnięta.
- Testuj scenariusze brzegowe – sprawdź, co się stanie, gdy trigger uruchomi się przy masowej operacji (np. UPDATE 1000 rekordów).
- Używaj NEW i OLD konsekwentnie – w INSERT dostępne jest tylko NEW, w DELETE tylko OLD, a w UPDATE oba – warto jasno pokazywać, jakie wartości są używane.
- Zachowuj spójność z logiką biznesową – trigger powinien odzwierciedlać regułę, która jest stabilna i nie zmienia się co tydzień; w przeciwnym razie baza stanie się trudna w utrzymaniu.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Najczęstsze błędy i czego unikać w pracy z wyzwalaczami

- Nadmiar wyzwalaczy – zbyt duża liczba triggerów na jednej tabeli sprawia, że każda operacja jest wolniejsza i trudniejsza do debugowania.
- Ukryta logika – jeśli ważna część biznesu dzieje się „w tle” w triggerze, programiści mogą nie zdawać sobie sprawy, że baza modyfikuje dane dodatkowo. Zawsze informuj zespół, że trigger istnieje.
- Zapętlenia – trigger wywołuje zmianę w tej samej tabeli, która ponownie uruchamia trigger (np. AFTER UPDATE aktualizujący ten sam rekord). Może to prowadzić do nieskończonych pętli.
- Brak kontroli błędów – jeśli trigger nie obsłuży sytuacji wyjątkowej, cała transakcja zostanie wycofana. Lepiej przewidywać i obsługiwać takie sytuacje w kodzie triggera.
- Zbyt rozbudowana logika – wyzwalacze nie powinny być miejscem na złożone algorytmy; jeśli logika jest większa niż kilka linijek, lepiej wywołać procedurę.
- Nadużywanie triggerów zamiast constraintów – jeżeli regułę można zapisać prostym CHECK, UNIQUE albo FOREIGN KEY, to właśnie w ten sposób należy to zrobić – będzie szybciej i przejrzysiej.
- Brak indeksów – jeśli trigger często odwołuje się do innych tabel, a nie ma tam indeksów, każda operacja w tabeli bazowej może być bardzo wolna.
- Założenie, że kolejność triggerów jest przewidywalna – w wielu systemach (np. MySQL) przy wielu triggerach tego samego typu kolejność ich wykonywania nie jest gwarantowana.

Zdarzenia w bazach danych.

Uzupełnieniem wyzwalaczy i procedur jest także harmonogram zadań (Event Scheduler), który umożliwia automatyczne wykonywanie określonych operacji w czasie – takich jak okresowe czyszczenie logów, generowanie raportów czy kopiowanie danych do tabel archiwalnych. Dzięki schedulerowi baza danych zyskuje możliwość samodzielnej realizacji zaplanowanych zadań administracyjnych, co odciąża administratorów i zwiększa niezawodność systemu.

Domyślnie scheduler może być wyłączony, dlatego najpierw trzeba go uruchomić:

-- włączenie harmonogramu zdarzeń

SET GLOBAL event_scheduler = ON;



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



-- sprawdzenie statusu

SHOW VARIABLES LIKE 'event_scheduler';

Tworzenie prostego zdarzenia, które co minutę dopisuje wpis do tabeli logów:

```
CREATE TABLE logi (  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    opis VARCHAR(255),  
    czas TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
);  
  
DELIMITER $$  
  
CREATE EVENT log_co_minute  
ON SCHEDULE EVERY 1 MINUTE  
DO  
BEGIN  
    INSERT INTO logi (opis) VALUES ('Automatyczny wpis z event scheduler');  
END$$  
  
DELIMITER ;
```

Zdarzenie można zatrzymać lub usunąć:

ALTER EVENT log_co_minute DISABLE;

DROP EVENT log_co_minute;



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



A tu przykład bardziej praktyczny – czyszczenie starych rekordów z tabeli sesji:

```
CREATE EVENT czyszczenie_sesji  
ON SCHEDULE EVERY 1 DAY  
DO  
DELETE FROM sesje WHERE data_wygasniecia < NOW();
```

Transakcje i zasada ACID w bazach danych

Transakcje stanowią jeden z najważniejszych mechanizmów w relacyjnych bazach danych, zapewniając spójność i niezawodność operacji nawet w trudnych warunkach wielodostępności i awarii. Ich moc wynika z przestrzegania zasady ACID, która od dziesięcioleci stanowi standard projektowania systemów DBMS. Co istotne, transakcje nie działają w izolacji – można je łączyć z procedurami, widokami, wyzwalaczami, tabelami tymczasowymi czy CTE. Dzięki temu stają się fundamentem logiki biznesowej po stronie bazy danych i umożliwiają tworzenie aplikacji, które nie tylko działają szybko, ale przede wszystkim gwarantują poprawność i bezpieczeństwo danych.

Czym są transakcje i po co zostały stworzone

Transakcja w bazie danych to logiczny zbiór operacji, które powinny zostać wykonane jako nierozdzielna całość. Innymi słowy – albo wszystkie operacje wchodzące w skład transakcji zostaną zatwierdzone (ang. commit), albo żadna z nich nie zostanie trwale zapisana, a baza danych wróci do stanu sprzed rozpoczęcia transakcji (ang. rollback). Dzięki temu możemy zagwarantować, że dane pozostaną spójne nawet w obliczu awarii systemu, błędów w programie czy nieprzewidzianych przerw.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Koncepcja transakcji powstała jako odpowiedź na problemy pojawiające się w systemach wielodostępnych, gdzie wielu użytkowników i procesów jednocześnie wykonuje operacje na wspólnych danych. Bez transakcji mogłoby dochodzić do sytuacji, w których dane są niespójne – np. jeśli jeden użytkownik w połowie wprowadzi zmiany, a w tym samym czasie drugi zacznie je odczytywać. Klasycznym przykładem jest przelew bankowy: obciążenie jednego konta i uznanie drugiego muszą być wykonane razem, ponieważ zapisanie tylko jednej z tych operacji prowadziłoby do błędów finansowych.

Zasada ACID - kwas

Fundamentem transakcji są właściwości określane akronimem ACID:

- Atomicity (Atomowość) – transakcja jest niepodzielna. Wszystkie jej operacje muszą być wykonane w całości lub wcale. Jeśli nastąpi błąd, system przywraca stan sprzed jej rozpoczęcia.
- Consistency (Spójność) – baza po zakończeniu transakcji musi pozostawać w stanie zgodnym z regułami integralności. Dane nie mogą łamać żadnych zdefiniowanych ograniczeń (np. kluczy obcych, unikalności, reguł biznesowych).
- Isolation (Izolacja) – transakcje wykonywane równolegle nie powinny sobie przeszkadzać. Z punktu widzenia użytkownika każda transakcja działa tak, jakby była wykonywana samodzielnie, nawet jeśli w tle dzieją się inne operacje.
- Durability (Trwałość) – raz zatwierdzone zmiany w bazie danych są trwałe i przetrwają awarie systemu czy restart serwera. System DBMS zapewnia ich zapis na stabilnym nośniku.

Te cztery zasady zapewniają, że transakcje spełniają wymagania realnych systemów, takich jak bankowość, systemy rezerwacji lotniczych czy zarządzanie magazynem.

Zasady działania transakcji

W typowym systemie zarządzania bazą danych (DBMS) transakcja przebiega według cyklu:



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



- **START TRANSACTION** – rozpoczęcie transakcji (może być jawne lub automatyczne, w zależności od ustawienia autocommit).
- **Operacje DML** – wykonywanie poleceń modyfikujących dane, takich jak **INSERT**, **UPDATE**, **DELETE**.
- **COMMIT** – zatwierdzenie wszystkich operacji w ramach transakcji, zapisanie ich trwale w bazie.
- **ROLLBACK** – wycofanie wszystkich operacji, przywrócenie bazy do stanu sprzed rozpoczęcia transakcji.

Opcjonalnie można korzystać z **SAVEPOINT**, czyli punktów kontrolnych wewnątrz transakcji, które umożliwiają wycofanie się tylko do określonego miejsca, bez cofania całości.

Transakcje a inne elementy SQL

Transakcje nie są izolowanym mechanizmem – mogą być łączone z innymi obiektami i narzędziami bazy danych:

Procedury składowane – transakcje bardzo często występują w procedurach. Procedura może zawierać instrukcje **BEGIN TRANSACTION**, **COMMIT**, **ROLLBACK**, co pozwala traktować całą logikę biznesową jako jeden spójny blok. Typowe jest np. umieszczanie w procedurze obsługi przelewu bankowego – dopiero na końcu, jeśli wszystkie etapy się powiedą, następuje **COMMIT**.

Widoki – same w sobie nie zawierają logiki transakcyjnej (są jedynie zapisanymi zapytaniami), ale mogą być używane wewnątrz transakcji, np. jako źródło danych dla **UPDATE** czy **INSERT**. Wówczas transakcja kontroluje spójność zmian na tabelach bazowych, które stoją za widokiem.

Wyzwalacze (triggery) – działają w ramach tej samej transakcji, co wywołujące je polecenie. Oznacza to, że jeśli trigger wprowadza dodatkowe zmiany (np. aktualizuje log w tabeli audytowej), to w razie **ROLLBACK** cofnięte zostaną zarówno zmiany główne, jak i te wywołane przez wyzwalacz. Dzięki temu całość zachowuje spójność.

CTE (Common Table Expressions) – mogą być używane wewnątrz transakcji jako część zapytań DML. Na przykład, można utworzyć CTE wyciągające dane do



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



modyfikacji, a następnie użyć ich w UPDATE. Wszystko to zostanie zatwierdzone lub cofnięte razem z całą transakcją.

Tabele tymczasowe – również mogą być elementem transakcji. Dane w tabeli tymczasowej mogą być modyfikowane w ramach transakcji, a następnie zatwierdzone lub cofnięte. To daje dużą elastyczność przy skomplikowanych procesach ETL.

Transakcje a poziomy izolacji

Wielodostępność baz danych wymusza kontrolę nad tym, jak transakcje widzą wzajemnie swoje dane. Poziomy izolacji w SQL (wg standardu ANSI) to:

- READ UNCOMMITTED – najniższy poziom; transakcja może odczytywać dane jeszcze niezatwierdzone przez inne transakcje (dirty read).
- READ COMMITTED – transakcja widzi tylko dane zatwierdzone, ale te mogą się zmieniać w trakcie działania (non-repeatable read).
- REPEATABLE READ – dane odczytane raz w danej transakcji nie zmieniają się podczas jej trwania, ale mogą pojawić się nowe wiersze spełniające warunek (phantom read).
- SERIALIZABLE – najwyższy poziom, transakcje wykonywane są tak, jakby odbywały się jedna po drugiej; pełna izolacja, ale kosztem wydajności.

Dobór poziomu izolacji zależy od aplikacji: system bankowy najczęściej wymaga najwyższego poziomu, natomiast w systemach raportowych często wystarczy niższy, aby nie blokować innych użytkowników.

Dobre praktyki pracy z transakcjami

- Obejmuj transakcją tylko te operacje, które muszą być wykonane razem – im dłużej trwa transakcja, tym większe ryzyko blokad i konfliktów.
- Unikaj wykonywania operacji interaktywnych (np. oczekiwania na decyzję użytkownika) w środku transakcji.
- Pamiętaj o jawnym COMMIT lub ROLLBACK – pozostawienie otwartej transakcji może blokować innym dostęp do danych.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



- Testuj logikę w procedurach i triggerach, aby upewnić się, że w razie błędów system zawsze wykona poprawny ROLLBACK.
- Monitoruj blokady (ang. locks) i konflikty – źle zaprojektowane transakcje mogą prowadzić do zakleszczeń (deadlocków).

Czym jest CTE i po co się tego używa?

CTE (Common Table Expressions) to tymczasowy, nazwany wynik zapytania, którego można użyć w obrębie jednego polecenia SQL jakby był tabelą. Traktuj CTE jak „mini-tabelę roboczą” zbudowaną tuż przed właściwym SELECT-em. Ułatwia to:

- dzielenie skomplikowanych zapytań na czytelne etapy,
- ponowne użycie tego samego pośredniego wyniku kilka razy w jednym zapytaniu,
- zastąpienie zagnieżdżonych podzapytań czy „derived tables” czymś bardziej przejrzystym,
- pisanie zapytań rekurencyjnych (np. generowanie sekwencji liczb, przechodzenie drzew/hierarchii).

Współczesne MySQL/MariaDB obsługują CTE nierekurencyjne (WITH) i rekurencyjne (WITH RECURSIVE).

CTE a podzapytania i tabele tymczasowe – kiedy co wybrać?

CTE i podzapytania rozwiązują ten sam problem: chcemy policzyć coś „po drodze”, zanim zwrócimy wynik końcowy. Różnica jest głównie w czytelności i możliwości ponownego użycia. Podzapytanie wstawiasz w miejscu, w którym jest potrzebne, ale łatwo robi się z tego „spaghetti”. CTE, definiowane w sekcji WITH na początku polecenia, pozwala nazwać etap pośredni (np. duze_miasta) i odwołać się do niego kilka razy w tym samym poleceniu – zapytanie jest krótsze, warstwy logiki są widoczne, łatwiej je testować i utrzymywać.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Z kolei tabele tymczasowe sprawdzają się wtedy, gdy pośredni wynik chcesz wykorzystać w wielu poleceniach w ramach jednej sesji, chcesz dołożyć indeksy albo „zmaterializować” bardzo kosztowny wynik, żeby nie był liczone po raz kolejny. CTE „żyje” tylko w jednym poleceniu i nie wymaga sprząkania; tabela tymczasowa ma dłuższy żywot i daje więcej kontroli nad wydajnością, ale wymaga jawnej organizacji (tworzenie, ew. indeksowanie, czyszczenie).

Wydajnościowo nie ma złotej reguły: optymalizator w wielu silnikach potrafi „wciągnąć” CTE w plan wykonania tak, jakby to było podzapytanie. Czasem CTE bywa materializowane, czasem nie – dlatego nie zakładaj z góry, że CTE przyspieszy lub spowolni zapytanie. Używaj CTE dla czytelności i logiki krokowej, a o wydajności decyduj na podstawie planu (EXPLAIN) i pomiarów.

Dobre praktyki pracy z CTE

Warto nadawać CTE rzeczowe i jednoznaczne nazwy (np. europe, kalendarz, duze_miasta), bo po tygodniu nikt nie pamięta, co znaczy cte1. Zwróć uwagę, by w CTE wybierać tylko potrzebne kolumny – każdy nieużywany bajt to zbędny koszt. Jeśli wynik CTE zawiera wyrażenia lub funkcje, jawnie nadaj listę nazw kolumn lub aliasy, aby uniknąć nieczytelnych nazw automatycznych. Buduj zapytania etapami: najpierw filtr, potem wzbogacenie danych, na końcu wynik – jedno CTE = jeden zgrabny krok. W CTE rekurencyjnym zawsze domykaj warunek stopu, a jeśli nie potrzebujesz deduplikacji, stosuj UNION ALL zamiast UNION. Pamiętaj również, że ORDER BY wewnątrz CTE nie gwarantuje kolejności – ostateczną kolejność ustala ORDER BY w zapytaniu końcowym. Jeżeli wiesz, że ten sam pośredni wynik będzie wykorzystywany w wielu poleceniach lub wymaga indeksów, rozważ tabelę tymczasową.

Najczęstsze błędy CTE i jak ich uniknąć

Błąd „Unknown table ... in field list” w CTE rekurencyjnym zwykle oznacza, że część rekurencyjna nie odwołuje się do nazwy CTE (albo jest literówka). Brak warunku



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



stopu kończy się przekroczeniem limitu rekurencji – zawsze dopisz predykat zatrzymujący. Niejasne aliasy lub brak listy kolumn skutkują mylącymi nazwami w wyniku – nazywaj kolumny świadomie. Nie ufaj sortowaniu w CTE – sortuj na końcu. Unikaj monolitycznych CTE z dziesiątkami kolumn – rozbij etapy, aby zachować czytelność i kontrolę nad kosztami.

Kiedy CTE nie jest dobrym wyborem?

Jeśli ten sam pośredni zbiór danych ma być użyty w kilku poleceniach lub wymaga indeksów – lepsza będzie tabela tymczasowa. Gdy w jednym poleceniu wielokrotnie odwołujesz się do bardzo ciężkiego CTE, czasem korzystniej jest jawnie je zmaterializować (tymczasowa tabela), zamiast liczyć wszystko na nowo w ramach jednego planu wykonania. Jeśli CTE miałoby tylko „opakować” prosty SELECT, z którego i tak korzystasz raz – po prostu użyj SELECT bez dodatkowej warstwy.

Składnia ogólna (nierrekurencyjna) CTE

```
WITH nazwa_cte [(lista_kolumn)] AS (  
    SELECT ... -- dowolne zapytanie zwracające dane  
)  
SELECT ...  
FROM nazwa_cte  
WHERE ...;
```

Gdzie:

- nazwa_cte: unikalna w obrębie zapytania.
- (lista_kolumn): opcjonalna, ale przydatna, gdy chcesz nadać jasne nazwy kolumnom wyniku CTE.
- Zasięg: CTE „żyje” tylko w jednym poleceniu SQL — nie istnieje poza nim.
- Kolejność: możesz zdefiniować wiele CTE oddzielonych przecinkami; CTE zdefiniowane niżej może korzystać z tych wyżej.

Przykłady użycia CTE na bazie world:



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



W pierwszym kroku zawężamy dane do Polski, w drugim zostawiamy tylko duże miasta, a na końcu sortujemy i limitujemy wynik.

WITH polskie_miasta AS (

SELECT Name, Population

FROM City

WHERE CountryCode = 'POL'

),

duze_polskie_miasta AS (

SELECT Name, Population

FROM polskie_miasta

WHERE Population >= 300000

)

SELECT Name, Population

FROM duze_polskie_miasta

ORDER BY Population DESC

LIMIT 10;

Przykład drugi:

WITH

europe (Code, Name, Continent) AS (

SELECT Code, Name, Continent

FROM Country

WHERE Continent = 'Europe'

),

big_cities (CityName, CountryCode, Pop) AS (

SELECT Name, CountryCode, Population



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



FROM City

WHERE Population > 1000000

)

SELECT CityName, Pop

FROM big_cities

WHERE CountryCode IN (SELECT Code FROM europe)

ORDER BY Pop DESC;

Ćwiczenia samodzielne

1. Utwórz widok, który prezentuje wybrane, najczęściej używane dane z jednej z tabel w projekcie, ale ukrywa dane wrażliwe lub nadmiarowe.
2. Zbuduj widok raportowy, który łączy dane z co najmniej dwóch tabel projektu i zwraca podsumowanie istotnych informacji (np. zestawienie, raport zbiorczy, lista powiązanych encji).
3. Dodaj widok filtrujący tylko te rekordy, które spełniają określone warunki aktywności lub ważności w ramach projektu.
4. Napisz procedurę dodającą nowy rekord do jednej z głównych tabel projektu, uzupełniając przy tym automatycznie dodatkowe atrybuty (np. datę rejestracji, status początkowy).
5. Stwórz procedurę realizującą proces biznesowy składający się z kilku kroków w Twoim projekcie, np. dodanie rekordu do tabeli głównej, powiązanie go z tabelą szczegółową i aktualizacja danych powiązanych.
6. Zaimplementuj procedurę z parametrami wyjściowymi, np. zwracającą identyfikator nowo utworzonego rekordu albo komunikat o statusie operacji.
7. Utwórz własną funkcję użytkownika (UDF), która zwraca przetworzoną wartość na podstawie atrybutów istniejących w tabeli Twojego projektu (np. skrót tekstowy, wynik prostego obliczenia).
8. Zbuduj funkcję, która zwraca obliczoną wartość na podstawie jednej z kolumn liczbowych lub dat w projekcie (np. wiek, czas trwania, różnicę między wartościami).
9. Stwórz funkcję warunkową, która na podstawie wartości kolumny zwraca komunikat logiczny (np. TAK/NIE, aktywny/nieaktywny).
10. Napisz wyzwalacz BEFORE INSERT, który automatycznie uzupełnia pewne pola wstawianego rekordu (np. datę utworzenia, wartość domyślną statusu).



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



11. Stwórz wyzwalacz AFTER UPDATE, który zapisuje informacje o zmianach w osobnej tabeli logów.
12. Zaimplementuj wyzwalacz walidujący dane wprowadzane do jednej z tabel projektu – operacja powinna zostać przerwana, jeśli dane naruszają reguły biznesowe (np. liczba elementów nie może być mniejsza od 1).
13. Napisz procedurę wykorzystującą transakcję do realizacji procesu składającego się z kilku powiązanych kroków w Twoim projekcie. Wszystkie operacje muszą zostać zatwierdzone razem lub cofnięte w całości.
14. Dodaj do procedury punkt kontrolny (SAVEPOINT) i pokaż, jak można wycofać część operacji bez anulowania całej transakcji.
15. Utwórz CTE (Common Table Expression), które wyodrębni tylko wybraną grupę rekordów (np. aktywnych, spełniających określony warunek), a następnie na tej podstawie przygotuj raport.
16. Zbuduj rekurencyjne CTE, które przechodzi po strukturze hierarchicznej w Twoim projekcie (np. relacje nadrzędny–podrzędny, powiązania w drzewie danych).
17. Zdefiniuj zdarzenie w harmonogramie (Event Scheduler), które automatycznie usuwa lub archiwizuje przestarzałe dane w Twoim projekcie.
18. Utwórz zdarzenie, które w określonych odstępach czasu dopisuje rekord do wybranej tabeli (np. logów lub historii), aby zasymulować automatyczny monitoring danych.
19. Zaprojektuj scenariusz w ramach swojego projektu, w którym wykorzystasz łącznie widok, procedurę, wyzwalacz i transakcję do obsługi jednego procesu biznesowego.
20. Porównaj wydajność działania zapytań w projekcie wykonywanych bezpośrednio na tabelach i zapytań korzystających z widoków – zanotuj różnice i wnioski.
21. Przygotuj test „odporności” w projekcie – sprawdź, co się stanie, gdy trigger zgłosi błąd w ramach większej transakcji. Przeanalizuj, czy całość operacji zostanie wycofana zgodnie z zasadą atomowości.

Pytania pomocnicze

1. Czym jest wyzwalacz?
2. Jakie są rodzaje wyzwalaczy?
3. W jakich sytuacjach stosuje się wyzwalacze?
4. Podaj przykład wyzwalacza BEFORE.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



5. Podaj przykład wyzwalacza AFTER.
6. Czym jest widok?
7. Jakie są zalety stosowania widoków?
8. Czym różni się widok prosty od złożonego?
9. Czym jest procedura składowana?
10. Czym różni się procedura od funkcji składowanej?
11. Jak procedury zwiększają bezpieczeństwo danych?
12. Jak widoki mogą chronić dane?
13. Jak działa filtrowanie danych w widokach?
14. Co to jest logika biznesowa w bazie danych?
15. Jakie są potencjalne zagrożenia związane z wyzwalaczami?

Podsumowanie

W ramach laboratorium studenci poznali trzy kluczowe mechanizmy wspierające logikę biznesową w relacyjnych bazach danych: widoki, procedury składowane oraz wyzwalacze. Ćwiczenia pozwoliły zrozumieć, że widoki pełnią rolę wirtualnych tabel ułatwiających dostęp do danych, zwiększających bezpieczeństwo oraz umożliwiających standaryzację raportów. Procedury składowane pokazano jako narzędzie centralizujące logikę operacyjną po stronie bazy, co zwiększa kontrolę, redukuje powielanie kodu w aplikacjach i pozwala na wygodne zarządzanie transakcjami. Wyzwalacze z kolei zaprezentowano jako mechanizm automatyzacji, który reaguje na zdarzenia w bazie i zapewnia egzekwowanie reguł biznesowych oraz prowadzenie audytu zmian.

Dzięki realizacji zadań studenci zyskali praktyczne umiejętności projektowania obiektów bazodanowych, które nie tylko przechowują dane, ale aktywnie wspierają ich spójność, integralność i bezpieczeństwo. Kluczowym wnioskiem jest to, że odpowiednie zastosowanie widoków, procedur i wyzwalaczy pozwala znacząco uprościć pracę z danymi i podnieść jakość systemu informatycznego, jednak nadmierne lub nieprzemyślane ich użycie może prowadzić do spadku wydajności i trudności w utrzymaniu systemu.

Zajęcia te miały także wymiar refleksyjny – studenci mogli zastanowić się nad tym, w jaki sposób logika biznesowa powinna być rozdzielana pomiędzy warstwę aplikacyjną a bazodanową oraz jakie kompromisy wiążą się z wyborem konkretnego



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



rozwiązania. Wiedza ta ma bezpośrednie przełożenie na praktykę zawodową, gdzie poprawne wykorzystanie widoków, procedur i wyzwalaczy staje się fundamentem bezpiecznego, wydajnego i skalowalnego systemu zarządzania danymi.

Literatura

1. SQL Views, <https://www.geeksforgeeks.org/sql/sql-views/>, (data ostatniego dostępu 25.09.25)
2. SQL Triggers, <https://www.geeksforgeeks.org/dbms/sql-triggers/>, (data ostatniego dostępu 25.09.25)
CTE in SQL, <https://www.geeksforgeeks.org/sql/cte-in-sql/>, (data ostatniego dostępu 25.09.25)
3. SQL Stored Procedures, <https://www.geeksforgeeks.org/sql/what-is-stored-procedures-in-sql/>, (data ostatniego dostępu 25.09.25)



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską

