

Materiały do przedmiotu:

Bazy danych

Laboratorium nr 1:

**Wprowadzenie do podstawowych zasad działania
i organizacji baz danych relacyjnych.
Definiowanie encji i związków między nimi**

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	20
Pytania pomocnicze	22
Podsumowanie	22
Literatura	23



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Wprowadzenie, tematyka zajęć

Bazy danych relacyjne stanowią podstawę większości współczesnych systemów informatycznych. Pozwalają na uporządkowane przechowywanie informacji w tabelach, które pozostają ze sobą w określonych relacjach. Dzięki temu możliwe jest efektywne wyszukiwanie, aktualizowanie i kontrolowanie spójności danych. Wprowadzenie do zasad działania baz relacyjnych obejmuje zrozumienie pojęcia encji, atrybutów i związków, które stanowią fundament projektowania i organizacji danych.

Cel ćwiczenia/laboratorium

- Zrozumienie, czym jest baza danych i jej rola w systemach informatycznych.
- Poznanie pojęcia encji jako reprezentacji obiektów świata rzeczywistego.
- Rozpoznawanie atrybutów i ich znaczenia w opisie encji.
- Wyjaśnienie pojęcia relacji między encjami.
- Zrozumienie roli kluczy głównych i obcych.
- Umiejętność wskazania przykładu bazy relacyjnej z życia codziennego.
- Rozumienie różnicy między danymi a informacją.
- Poznanie podstawowych zasad organizacji danych w tabelach.
- Wyjaśnienie roli modelu relacyjnego w informatyce.
- Ćwiczenie identyfikacji encji i relacji w prostych scenariuszach.
- Zrozumienie pojęcia krotki i domeny w kontekście relacyjnych baz danych.
- Omówienie problemu redundancji danych i jej konsekwencji.
- Uświadomienie znaczenia spójności danych.
- Zrozumienie pojęcia schematu bazy danych.
- Wprowadzenie do terminologii używanej w relacyjnych bazach danych.
- Poznanie środowiska pracy phpMyAdmin.
- Poznanie zasady działania indeksów.

Instrukcja laboratoryjna/ćwiczeniowa

Charakter projektowy przedmiotu

Przedmiot Bazy danych realizowany jest w formie projektowej. Oznacza to, że w trakcie semestru studenci pracują nad własnym projektem, rozwijając go o kolejne elementy i funkcjonalności zgodnie z planem zajęć oraz wskazówkami



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



prowadzącego. Projekt pełni rolę praktycznego narzędzia dydaktycznego – każda nowo poznana koncepcja teoretyczna zostaje wdrożona w postaci rozwiązania bazodanowego, które następnie staje się częścią większej całości.

Podstawą rozpoczęcia pracy projektowej jest wybór przez grupę – składającą się od jednego do trzech studentów – określonego fragmentu rzeczywistości, który będzie odwzorowany w formie bazy danych. Wybrany obszar stanie się punktem wyjścia do stworzenia modelu oraz zaprezentowania możliwości systemu zaprojektowanego przez studentów.

Na tym etapie warto podkreślić wagę kamieni milowych, czyli elementów, które każdy projekt powinien zawierać. Ich brak, niestaranność wykonania, błędy lub nadmierne uproszczenia mogą stanowić podstawę do obniżenia oceny końcowej.

Pomocne dla studentów może być przyjęcie metafory, w której prowadzący pełni rolę klienta „zlecającego” realizację projektu. Grupa projektowa, poszerzając swoją wiedzę w trakcie semestru, prezentuje kolejne postępy, a tym samym potwierdza zdobyte umiejętności i weryfikuje poprawność zaproponowanego rozwiązania.

Celem grupy jest udowodnienie, że przygotowane rozwiązanie:

1. zostało wykonane starannie,
2. spełnia standardy projektowania baz danych,
3. umożliwia zastosowanie szerokiego wachlarza operacji w języku SQL,
4. znajduje praktyczne zastosowanie w wybranym obszarze.

Wymagania końcowego projektu zaliczeniowego

Każdy projekt powinien zawierać następujące elementy:

1. Motywację wykonania bazy danych – uzasadnienie, dlaczego warto zrealizować dany projekt i jakie problemy ma on rozwiązać.
2. Opis słowny wybranej rzeczywistości (np. uczelnia, sklep, szpital).
3. Wyodrębnienie niezbędnych tabel. Ważne jest, aby nazwy tabel zostały wyróżnione pogrubieniem w opisie słownym z punktu drugiego. Odzwierciedla to naturalny proces, w którym baza danych powstaje jako odpowiedź na oczekiwania klienta lub zleceniodawcy, po zebraniu informacji o potrzebach i specyfice organizacji.
4. Określenie atrybutów encji oraz relacji między encjami.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



5. Opracowanie pełnego diagramu ERD dla bazy danych.
6. Wypełnienie bazy danymi syntetycznymi. Można w tym celu wykorzystać specjalne generatory, np. bibliotekę faker w języku Python lub bibliotekę charlatan w języku R.
7. Opracowanie i prezentacja zróżnicowanych zapytań SELECT, które pozwolą pobierać dane z bazy, wykorzystując możliwości języka SQL omawiane na zajęciach.
8. Opracowanie i prezentacja zapytań modyfikujących dane i strukturę bazy, takich jak edycja, usuwanie rekordów czy zmiany w schemacie tabel.
9. Opracowanie i prezentacja widoków.
10. Opracowanie i prezentacja wyzwalaczy (triggerów).
11. Opracowanie i prezentacja procedur składowanych.
12. Prezentacja procesu zarządzania użytkownikami w systemie bazodanowym, obejmująca tworzenie kont z różnymi zestawami uprawnień oraz testowanie ich działania poprzez uwierzytelnianie i wykonywanie przykładowych operacji.
13. Prezentacja procesu tworzenia kopii zapasowej, importu i eksportu bazy danych.
14. Opracowanie i prezentacja uproszczonej wersji analogicznego systemu w dokumentowej bazie danych. Należy przedstawić różnice między systemem dokumentowym a relacyjnym oraz wskazać, jak baza dokumentowa może zostać wykorzystana – jako uzupełnienie istniejącego rozwiązania o nowe funkcjonalności lub jako alternatywa, w której projekt zrealizowany w bazie relacyjnej zostaje przełożony na język baz dokumentowych.

Czym są i do czego służą systemy bazodanowe

Baza danych to zorganizowany zbiór informacji, który umożliwia ich łatwe przechowywanie, wyszukiwanie i modyfikację. Współcześnie korzystamy z systemów zarządzania bazami danych (DBMS – Database Management Systems), które pełnią rolę „pośrednika” między użytkownikiem a danymi. Dzięki DBMS można tworzyć tabele, wprowadzać rekordy, wyszukiwać informacje przy pomocy języka SQL, a także kontrolować uprawnienia i bezpieczeństwo. Już od pierwszego kontaktu ważne jest, aby zrozumieć, że baza danych nie jest tylko prostym magazynem informacji, ale narzędziem wspierającym logikę działania aplikacji i organizacji. W trakcie pracy na zajęciach będziemy korzystać zarówno z interfejsu przeglądarkowego (phpMyAdmin), który pozwala na szybkie i intuicyjne tworzenie tabel i relacji, jak i z



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



bardziej zaawansowanego środowiska MySQL Workbench, które umożliwia pisanie zapytań w SQL oraz projektowanie diagramów baz danych.

Encja to pojęcie teoretyczne opisujące obiekt, który występuje w rzeczywistości i który chcemy odzwierciedlić w bazie danych. Encją może być na przykład student, książka, zamówienie czy pracownik. Każda encja posiada zestaw cech, które ją opisują – i to właśnie one stają się atrybutami w bazie danych.

Tabela jest praktycznym odpowiednikiem encji w bazie danych relacyjnej. Dane przechowywane są w wierszach (rekordach), a kolumny odpowiadają atrybutom encji. Na przykład tabela Students może zawierać kolumny takie jak id_student, imię, nazwisko, kierunek. Tabela jest więc miejscem, w którym encja znajduje swoje odzwierciedlenie w systemie komputerowym.

Atrybut to pojedyncza cecha encji, przechowywana jako kolumna w tabeli. Dla encji „Student” mogą to być imię, nazwisko czy numer albumu. Każdy atrybut ma swój typ danych (np. liczba całkowita, tekst, data), który określa, jakie wartości można w nim zapisać.

Relacje między encjami

Relacja określa sposób powiązania encji między sobą. W świecie rzeczywistym obiekty często występują w powiązaniach – pracownik należy do działu, student zapisuje się na kurs, klient składa zamówienie. W modelu relacyjnym takie związki odzwierciedlamy przez klucze główne i obce. Klucz główny jednoznacznie identyfikuje każdy rekord w tabeli, a klucz obcy wskazuje na rekord w innej tabeli, tworząc w ten sposób powiązanie.

Wyróżniamy trzy podstawowe rodzaje relacji:

1. Relacja jeden do jednego (1:1) – każdemu rekordowi w jednej tabeli odpowiada najwyżej jeden rekord w drugiej tabeli. Na przykład każdemu pracownikowi przypisany jest dokładnie jeden identyfikator służbowy.
2. Relacja jeden do wielu (1:N) – jednemu rekordowi w tabeli odpowiada wiele rekordów w innej tabeli. Przykład: jeden autor może napisać wiele książek, ale każda książka ma tylko jednego głównego autora.
3. Relacja wiele do wielu (N:M) – rekordy z jednej tabeli mogą odpowiadać wielu rekordom w drugiej i odwrotnie. Przykład: studenci mogą zapisywać się na



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



wiele kursów, a każdy kurs może być realizowany przez wielu studentów. W relacyjnej bazie danych relacje tego typu wymagają utworzenia dodatkowej tabeli pośredniczącej, w której przechowywane są pary powiązanych identyfikatorów.

Rodzaje kluczy w bazach danych

Klucze w relacyjnych bazach danych to mechanizm pozwalający jednoznacznie identyfikować dane i kontrolować powiązania pomiędzy tabelami. Dzięki nim możemy mieć pewność, że w bazie nie pojawią się sprzeczne informacje oraz że odwołania pomiędzy tabelami zawsze prowadzą do istniejących rekordów.

Klucz główny (PRIMARY KEY) to najważniejszy klucz w każdej tabeli. Określa on atrybut (lub zestaw atrybutów), którego wartości muszą być unikalne i nie mogą być puste. Dzięki temu każdy rekord w tabeli można jednoznacznie zidentyfikować. Typowym przykładem klucza głównego jest numer PESEL w tabeli Pracownicy albo identyfikator studenta w tabeli Studenci. Jeżeli pojedyncza kolumna nie wystarcza do jednoznacznej identyfikacji, można zdefiniować klucz złożony – np. w tabeli zapisów na kursy kluczem może być kombinacja (id_studenta, id_kursu). Oprócz klucza głównego w tabeli mogą istnieć także inne unikalne zestawy atrybutów. Nazywamy je kluczami alternatywnymi. Przykładowo, w tabeli studenci oprócz id_studenta unikalny może być również adres e-mail – wówczas e-mail można potraktować jako klucz alternatywny. W praktyce często spotykamy się też z kluczami sztucznymi (surrogate keys). Są to specjalnie wygenerowane identyfikatory, takie jak licznik automatycznie zwiększający się przy każdym nowym wpisie (AUTO_INCREMENT w MySQL), czy identyfikatory UUID. Klucz sztuczny sam w sobie nie niesie treści biznesowej, ale ułatwia obsługę bazy i zapewnia jednoznaczność.

Drugim ważnym pojęciem jest **klucz obcy** (FOREIGN KEY). To atrybut w jednej tabeli, który wskazuje na klucz główny w innej tabeli. Dzięki kluczom obcym budujemy relacje między encjami. Na przykład w tabeli Zamówienia możemy umieścić kolumnę id_klienta, która będzie kluczem obcym odwołującym się do tabeli Klienci. Dzięki temu każde zamówienie jest przypisane do konkretnego klienta, a baza danych nie pozwoli wprowadzić zamówienia z nieistniejącym identyfikatorem klienta.

Czym jest język SQL oraz plan budowy zapytania



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Język SQL (Structured Query Language) stanowi podstawowe narzędzie pracy z relacyjnymi bazami danych. Służy zarówno do definiowania struktur (DDL – Data Definition Language), jak i do operowania na danych (DML – Data Manipulation Language). Poniżej przedstawiono zestaw najważniejszych konstrukcji, które stanowią fundament praktycznej pracy z SQL.

Pisząc zapytanie w języku SQL, konieczne jest zachowanie określonej kolejności klauzul i operatorów. Nawet jeśli w praktyce pewne fragmenty są pomijane (bo nie zawsze wszystkie są potrzebne), logika wykonania zapytania zawsze przebiega w tej samej sekwencji:

- **WITH (CTE)** – definiowanie wspólnych wyrażeń tablicowych, czyli tymczasowych „mini-tabel”, które można wykorzystać w dalszej części zapytania.
- **SELECT** – wybór kolumn, obliczeń, aliasów, funkcji agregujących i innych elementów, które mają być wyświetlone w wyniku.
- **FROM** – wskazanie tabel lub podzapytań, z których pobierane są dane.
- **JOIN** – łączenie danych z wielu tabel na podstawie kluczy i warunków powiązań.
- **WHERE** – filtrowanie danych na poziomie pojedynczych rekordów.
- **GROUP BY** – grupowanie rekordów według wskazanych kolumn.
- **HAVING** – filtrowanie grup po dokonaniu agregacji.
- **WINDOW / OVER** – funkcje okna, pozwalające na obliczenia w ramach określonego „okna” danych (np. sumy czy średnie w działach).
- **UNION / UNION ALL / EXCEPT / INTERSECT** – operatory zbiorowe, umożliwiające łączenie wyników kilku zapytań lub wyciąganie części wspólnych czy różnic.
- **ORDER BY** – sortowanie wyników według wybranych kolumn.
- **OFFSET** – pominięcie określonej liczby pierwszych wierszy (np. przy paginacji).
- **LIMIT** – ograniczenie liczby zwróconych rekordów.

Elementy języka SQL

SQL pozwala pobierać, filtrować i prezentować dane w elastyczny sposób. Tworzenie zapytań SQL opiera się na pewnej ustalonej strukturze, w której poszczególne



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



klauzule wykonywane są w określonej kolejności. Całość zaczyna się od SELECT, czyli wyboru kolumn, które mają zostać zwrócone. To tutaj decydujemy, jakie dane będą widoczne w wyniku – mogą to być pojedyncze kolumny, wszystkie dane z tabeli (*), albo bardziej złożone elementy, takie jak aliasy, funkcje agregujące czy wyrażenia na ciągach znaków. Przykładowo możemy pobrać imię, nazwisko oraz pensję pracownika, a jednocześnie zbudować nową kolumnę „Pełne Imię i Nazwisko”, łącząc dwie wartości funkcją CONCAT.

Kolejnym krokiem jest FROM, czyli określenie tabeli lub tabel, z których dane mają być pobrane. Tutaj wskazujemy główne źródło danych. W prostych przypadkach będzie to pojedyncza tabela, np. employees, ale nic nie stoi na przeszkodzie, by w tym miejscu pojawiło się podzapytanie zwracające dane pośrednie.

W praktyce bardzo często korzystamy również z JOIN-ów, czyli mechanizmów łączenia danych z różnych tabel. Najczęściej stosowany jest INNER JOIN, który zwraca tylko te rekordy, które mają swoje odpowiedniki w obu tabelach. LEFT JOIN pozwala pobrać wszystkie dane z tabeli lewej, nawet jeśli nie mają dopasowania po stronie prawej. Analogicznie działa RIGHT JOIN. Jeżeli interesują nas wszystkie rekordy niezależnie od dopasowania, możemy użyć FULL JOIN. Istnieje też CROSS JOIN, który generuje iloczyn kartezjański wszystkich wierszy, choć używany jest rzadziej. Dzięki JOIN-om możemy w jednym zapytaniu odwołać się np. do tabeli pracownicy, pensje i działy, łącząc dane według kluczy obcych.

Aby ograniczyć liczbę zwracanych rekordów, stosujemy klauzulę WHERE, która filtruje dane przed wykonaniem grupowania i agregacji. To tutaj umieszczamy warunki logiczne (np. pensja większa niż 50 000, data zatrudnienia po roku 2000) oraz operatory porównania (=, >, <, BETWEEN, IN, LIKE, IS NULL). Klauzula WHERE pozwala nam skoncentrować się tylko na tych danych, które rzeczywiście spełniają interesujące nas kryteria.

Następnym etapem jest GROUP BY, które grupuje dane według wskazanych kolumn. Jest to niezbędne, gdy chcemy wykonywać operacje agregujące, takie jak liczenie średniej, sumy czy liczby rekordów w każdej grupie. Ważna zasada mówi, że kolumny pojawiające się w SELECT muszą być częścią grupowania lub zostać użyte wewnątrz funkcji agregujących.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Gdy już pogrupujemy dane, możemy zastosować dodatkowe filtrowanie za pomocą klauzuli **HAVING**. W przeciwieństwie do **WHERE**, które działa na poziomie pojedynczych rekordów, **HAVING** służy do filtrowania całych grup wyników. Możemy tu np. odrzucić działy, w których średnia pensja jest niższa niż określona wartość.

Otrzymane wyniki można uporządkować przy użyciu **ORDER BY**. Ta klauzula określa, według których kolumn i w jakiej kolejności mają zostać posortowane dane. Domyślnie sortowanie odbywa się rosnąco (**ASC**), ale można wymusić malejącą kolejność (**DESC**). Nic nie stoi na przeszkodzie, aby sortować po kilku kolumnach jednocześnie, np. najpierw po wysokości pensji, a potem po dacie zatrudnienia.

Jeżeli danych jest bardzo dużo, w wielu sytuacjach przydaje się **LIMIT**, które ogranicza liczbę zwróconych rekordów. Dzięki temu możemy np. wyświetlić tylko 10 najlepiej zarabiających pracowników, zamiast całej tabeli.

Czasem pojawia się potrzeba łączenia wyników wielu zapytań. W tym celu stosujemy **UNION** oraz **UNION ALL**. Pierwsze z nich zwraca unikalne rekordy, eliminując duplikaty, drugie natomiast wyświetla wszystkie dane, w tym powtarzające się wiersze. Istnieje przy tym ważna zasada: liczba i typy kolumn w zapytaniach łączonych przez **UNION** muszą być zgodne.

Wreszcie, w **SQL** często korzysta się z podzapytań. Mogą one być nieskorelowane, czyli niezależne od zapytania głównego (np. lista wartości w klauzuli **IN**), albo skorelowane, czyli odwołujące się do danych aktualnie przetwarzanych przez zapytanie nadrzędne. W tym drugim przypadku podzapytanie wykonywane jest wielokrotnie, np. aby sprawdzić za pomocą **EXISTS**, czy dany rekord ma powiązane dane w innej tabeli.

Wszystkie te elementy można ze sobą łączyć, tworząc bardziej złożone zapytania. Przykładowo, zapytanie może jednocześnie wybierać kolumny z kilku tabel połączonych **JOIN**-ami, filtrować dane przy użyciu **WHERE**, grupować je według działu, stosować agregację, a następnie sortować i ograniczać liczbę wyników.

Indeksy w bazach danych

Indeks w bazie danych to specjalna struktura, której zadaniem jest przyspieszenie wyszukiwania rekordów w tabeli. Można go porównać do indeksu w książce –



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



zamiast przeglądać wszystkie strony po kolei, sięgamy od razu do odpowiedniego miejsca. W praktyce oznacza to, że baza danych nie musi analizować każdego wiersza w tabeli przy wykonywaniu zapytania, lecz korzysta z uporządkowanej struktury, najczęściej w postaci drzewa B+. Dzięki temu procesy filtrowania, wyszukiwania i sortowania danych przebiegają znacznie szybciej.

Należy jednak wyraźnie rozróżnić indeksy od kluczy. Klucze (PRIMARY KEY, FOREIGN KEY, UNIQUE) pełnią funkcję reguł integralności. Określają one dopuszczalne wartości w kolumnach i wymuszają poprawność powiązań między tabelami. Na przykład klucz główny gwarantuje unikalność i niepustość identyfikatora, a klucz obcy zapewnia, że dane w jednej tabeli odwołują się wyłącznie do istniejących rekordów w tabeli nadrzędnej. Indeksy natomiast nie stanowią reguły biznesowej – są narzędziem czysto technicznym, które wspiera wydajność działania bazy. Możemy stworzyć indeks na dowolnej kolumnie (np. na kolumnie „nazwisko”), aby przyspieszyć zapytania zawierające filtr WHERE nazwisko = 'Kowalski'.

Warto pamiętać, że wiele kluczy automatycznie powoduje utworzenie indeksu. Zdefiniowanie klucza głównego wiąże się z utworzeniem indeksu unikalnego, podobnie działa klucz UNIQUE. W niektórych systemach również dla kluczy obcych automatycznie sugerowane jest utworzenie indeksu, aby ułatwić sprawdzanie poprawności powiązań. Należy jednak podkreślić, że nie każdy indeks oznacza klucz. Indeks można utworzyć wyłącznie w celu przyspieszenia raportów czy zestawień i w żaden sposób nie wpływa on na poprawność danych.

W praktyce wyróżnia się kilka podstawowych rodzajów indeksów. Indeks jednokolumnowy dotyczy jednej kolumny i przyspiesza zapytania bazujące na jej wartościach. Indeks wielokolumnowy obejmuje kilka kolumn, co jest przydatne przy filtrach i sortowaniach uwzględniających wiele kryteriów jednocześnie. Indeks unikalny zapewnia brak duplikatów w kolumnie, np. w przypadku adresu e-mail użytkownika. Indeksy pełnotekstowe służą do wyszukiwania słów i fraz w dużych tekstach, a indeksy przestrzenne (GIS) wspierają operacje na danych geograficznych, takich jak współrzędne czy kształty geometryczne.

Indeksy są szczególnie przydatne, gdy tabela zawiera dużą liczbę rekordów, gdy często korzystamy z zapytań filtrujących (WHERE), sortujących (ORDER BY) lub łączących (JOIN), a także wtedy, gdy chcemy wymusić unikalność wartości w



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



kolumnie. Nie są jednak pozbawione ograniczeń. Każdy indeks przyspiesza odczyt, ale spowalnia operacje zapisu – każde dodanie, usunięcie lub modyfikacja rekordu wymaga również aktualizacji indeksów. Nadmiar indeksów zwiększa również zużycie pamięci i przestrzeni dyskowej. Źle dobrany indeks, np. na kolumnie, w której większość wartości jest identyczna, może w praktyce nie dawać żadnych korzyści wydajnościowych.

Polecenia SQL związane z indeksami

- Tworzenie indeksu na jednej kolumnie:

```
CREATE INDEX idx_nazwisko ON Pracownicy(nazwisko);
```

- Tworzenie indeksu złożonego (na kilku kolumnach):

```
CREATE INDEX idx_nazwisko_imie ON Pracownicy(nazwisko, imie);
```

- Tworzenie indeksu unikalnego:

```
CREATE UNIQUE INDEX idx_email_unique ON Uzytkownicy(email);
```

- Usuwanie indeksu:

```
DROP INDEX idx_nazwisko ON Pracownicy;
```

- Wyświetlenie indeksów w tabeli (w MySQL)

```
SHOW INDEXES FROM Pracownicy;
```

Rys historyczny najważniejsze formaty przechowywania danych

Zanim zaczniemy pracować z bazami danych, warto najpierw zrozumieć ich pochodzenie, potrzebę powstania oraz to, jak przez lata się rozwijały. Bazy danych nie pojawiły się z dnia na dzień — ich historia sięga dawnych czasów, kiedy ludzie zaczęli prowadzić interesy i chcieli przechowywać informacje o kontrahentach, zbierać podatki, spisy ludności czy inwentarza. Przez całe wieki dane były zapisywane na papierze, w księgach, na kamiennych tabliczkach, czasem na liściach



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



czy pergaminach. Współcześnie wielu z nas pamięta jeszcze czasy sprzed masowej cyfryzacji, gdy dokumentacja, faktury, ewidencje – odbiory, dostawy czy zamówienia – były prowadzone ręcznie i papierowo.

Rozwój technologii komputerowych – początkowo stosowanych głównie w zastosowaniach wojskowych i naukowych – sprawił, że zaczęto dostrzegać, iż komputery mogą znacznie ułatwić, przyspieszyć i uodpornić na błędy procesy związane ze zbieraniem, przechowywaniem i przetwarzaniem danych. W latach sześćdziesiątych XX wieku zaczęły się pojawiać pierwsze systemy komputerowych baz danych, jak systemy hierarchiczne i sieciowe.

Format CSV (Comma-Separated Values) jest jednym z najstarszych i najprostszych sposobów przechowywania danych tabelarycznych w plikach tekstowych. Jego historia sięga lat siedemdziesiątych, kiedy wczesne języki programowania, takie jak Fortran, korzystały z prostych konwencji rozdzielania pól przecinkami. Plik CSV to nic innego jak sekwencja wierszy, w których poszczególne wartości oddzielone są separatorem – najczęściej przecinkiem, choć bywa nim również średnik, tabulacja czy inny znak. Pierwszy wiersz zwykle pełni funkcję nagłówka zawierającego nazwy kolumn. Format ten, dzięki swojej prostocie, stał się powszechny i uniwersalny. Można go łatwo odczytać w prostym edytorze tekstu, w arkuszu kalkulacyjnym czy w bazie danych. CSV sprawdza się doskonale do prostych zestawień, raportów czy eksportu danych między różnymi programami. Jego wadą jest brak możliwości reprezentowania bardziej złożonych struktur, takich jak dane hierarchiczne, brak jednoznacznej specyfikacji typów danych oraz problemy z kodowaniem czy cytowaniem pól zawierających przecinki lub znaki nowej linii. Jest to więc format niezwykle praktyczny, lecz ograniczony do prostych tabel.

XML, czyli eXtensible Markup Language, został opracowany w latach dziewięćdziesiątych przez konsorcjum W3C jako uproszczona i bardziej elastyczna wersja wcześniejszego standardu SGML. Został zaprojektowany do przechowywania i wymiany danych w sposób czytelny zarówno dla ludzi, jak i maszyn. Struktura XML opiera się na elementach oznaczonych tagami otwierającymi i zamykającymi, wewnątrz których mogą znajdować się dane, atrybuty oraz kolejne zagnieżdżone elementy. XML świetnie nadaje się do reprezentowania danych hierarchicznych i złożonych, a dodatkowo pozwala na walidację poprawności dokumentu dzięki



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



wykorzystaniu schematów DTD lub XSD. Format ten znalazł szerokie zastosowanie w integracji systemów, przesyłaniu dokumentów biznesowych, tworzeniu kanałów RSS czy w standardach wymiany danych w administracji publicznej. Jego główną wadą jest duża „ciężkość” – pliki XML bywają rozbudowane i zawierają wiele nadmiarowych znaczników, co zwiększa rozmiar danych i wydłuża czas przetwarzania. Pomimo tego XML pozostaje bardzo istotnym formatem, szczególnie tam, gdzie kluczowa jest precyzja opisu, walidacja i standaryzacja danych.

JSON, czyli JavaScript Object Notation, pojawił się na początku lat dwutysięcznych jako prostsza alternatywa dla XML. Wywodzi się bezpośrednio z notacji obiektowej języka JavaScript, ale szybko stał się niezależnym i uniwersalnym formatem wymiany danych. JSON opiera się na zapisie par klucz–wartość oraz tablic wartości, przy czym dopuszcza typy takie jak liczby, teksty, wartości logiczne czy null. Jego składnia jest niezwykle przejrzysta, zbliżona do sposobu, w jaki programista definiuje struktury w kodzie. Format ten jest lekki, szybki w przetwarzaniu i wyjątkowo popularny w aplikacjach webowych, szczególnie w komunikacji serwer–klient poprzez API. JSON bardzo dobrze wspiera wymianę danych w usługach internetowych, a także stał się podstawowym formatem przechowywania danych w wielu bazach dokumentowych, takich jak MongoDB. Wadą JSON jest brak możliwości bezpośredniego dodawania komentarzy oraz brak wbudowanego mechanizmu walidacji tak precyzyjnego, jak w przypadku XML. Istnieją wprowadzone rozszerzenia takie jak JSON Schema, jednak ich stosowanie nie jest jednolicie ustandaryzowane. Mimo to JSON dzięki prostocie i efektywności stał się najpopularniejszym formatem wymiany danych w nowoczesnych systemach informatycznych.

XLSX to format plików arkuszy kalkulacyjnych wprowadzony przez Microsoft wraz z pakietem Office 2007 jako część standardu Office Open XML. Zastąpił on wcześniejsze pliki w formacie binarnym XLS, wprowadzając strukturę opartą na XML i przechowywaną w skompresowanej paczce ZIP. Każdy plik XLSX składa się z zestawu plików XML, które opisują dane w komórkach, formatowanie, style, formuły, metadane, wykresy czy grafiki. Dzięki temu format XLSX obsługuje wiele arkuszy w jednym pliku, zaawansowane funkcje obliczeniowe, formatowanie warunkowe, wykresy i tabele przestawne. Jego ogromną zaletą jest bogactwo funkcji, które pozwalają nie tylko przechowywać dane, lecz także je analizować i prezentować w



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



atrakcyjnej formie. XLSX jest także lepiej zabezpieczony przed uszkodzeniami niż wcześniejsze formaty binarne, ponieważ uszkodzenie jednego pliku wewnętrznego nie musi oznaczać utraty całej zawartości. Jako format otwarty (ISO/IEC 29500) zapewnia też kompatybilność z innymi pakietami biurowymi, takimi jak LibreOffice czy Google Sheets. Wadą XLSX jest większa złożoność oraz większe zapotrzebowanie na zasoby przy odczycie i zapisie w porównaniu z prostymi formatami tekstowymi. Pomimo tego XLSX pozostaje standardem de facto w świecie arkuszy kalkulacyjnych i jest niezastąpiony wszędzie tam, gdzie oprócz danych istotna jest również ich analiza i prezentacja.

Wykorzystanie pakietu XAMPP oraz phpMyAdmin

Aby uzyskać praktyczny obraz tego, do czego może służyć baza danych i jak rozpocząć z nią pracę, warto skorzystać z pakietu XAMPP. Jest to darmowe środowisko, które zawiera w sobie wszystkie niezbędne narzędzia do uruchomienia serwera lokalnego:

- Apache – serwer WWW obsługujący zapytania HTTP,
- MySQL/MariaDB – system zarządzania relacyjną bazą danych,
- PHP – interpreter języka PHP pozwalający uruchamiać aplikacje internetowe,
- phpMyAdmin – przeglądarkowe narzędzie graficzne do zarządzania bazami danych.

Dzięki XAMPP możemy uruchomić lokalny serwer i od razu rozpocząć naukę pracy z bazami danych, bez konieczności skomplikowanej konfiguracji. Po uruchomieniu panelu sterowania XAMPP należy włączyć dwa podstawowe moduły: Apache oraz MySQL i wybrać przycisk Admin. Zostało to przedstawione na rysunku 1. To wystarczy, aby w przeglądarce otworzyć phpMyAdmin i rozpocząć tworzenie pierwszej bazy.



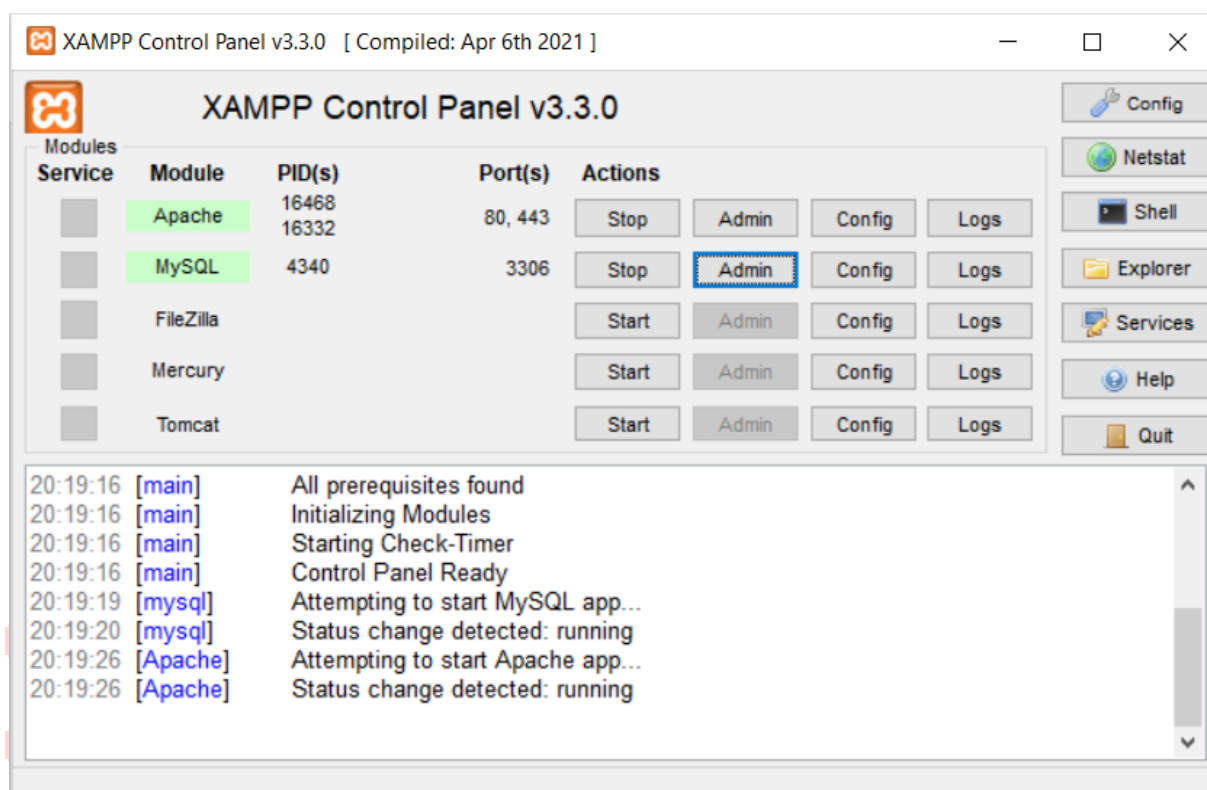
Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską





Rysunek 1. Widok programu XAMPP

phpMyAdmin to narzędzie dostępne z poziomu przeglądarki internetowej pod adresem <http://localhost/phpmyadmin>. Domyślnie logowanie odbywa się przy pomocy użytkownika root, bez hasła. Jest to rozwiązanie wygodne do pracy lokalnej, ale w przypadku faktycznego wdrożenia w przedsiębiorstwie czy pracy w zespole należy utworzyć własnych użytkowników oraz zmodyfikować konfigurację, tak aby uniemożliwić korzystanie z konta root w celach produkcyjnych. Zmiany tego typu można wprowadzać w plikach konfiguracyjnych serwera MySQL.

W tym celu należy edytować plik konfiguracyjny phpMyAdmin znajdujący się w katalogu:

C:\xampp\phpMyAdmin\config.inc.php

W pliku tym należy odnaleźć linijkę:



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską

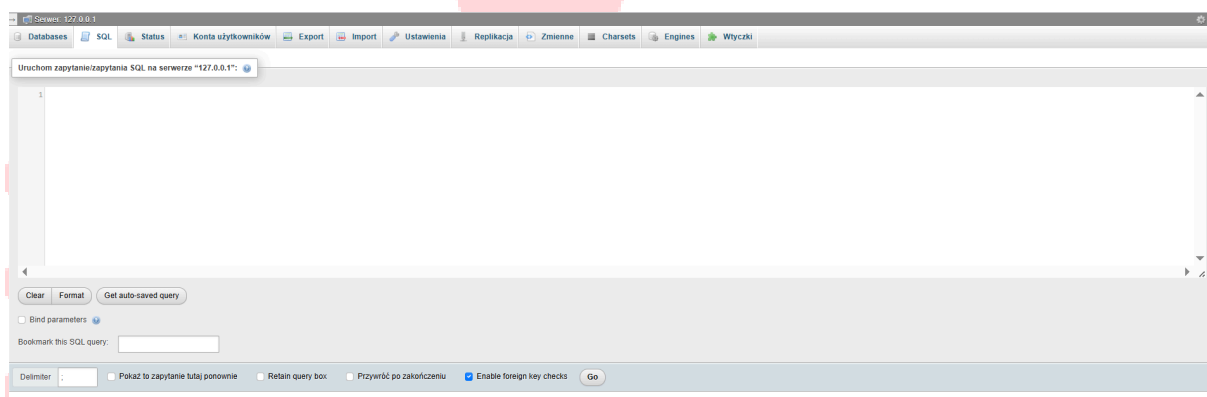


```
$cfg['Servers'][$i]['auth_type'] = 'config';
```

i zmienić ją na:

```
$cfg['Servers'][$i]['auth_type'] = 'cookie';
```

Ustawienie *auth_type* na wartość *cookie* powoduje, że przy każdym logowaniu phpMyAdmin poprosi o podanie nazwy użytkownika i hasła. Dzięki temu możemy zalogować się zarówno na konto root, jak i na innych użytkowników utworzonych w systemie MySQL. Po dokonaniu tej zmiany należy zresetować serwer w panelu XAMPP (zatrzymać i ponownie uruchomić moduły Apache oraz MySQL), aby nowe ustawienia zaczęły obowiązywać.



Rysunek 2 phpMyAdmin z otwartą zakładką służącą wywoływaniu kwerend SQL.

Zapoznanie interfejsu phpMyAdmin

Po zalogowaniu do phpMyAdmin widzimy panel główny, w którym znajdują się różne zakładki ułatwiające zarządzanie bazą zostały one przedstawione na rysunku 2:

- Databases (Bazy danych) – umożliwia tworzenie nowych baz i przeglądanie istniejących. To od tej zakładki najczęściej zaczynamy pracę.
- SQL – pozwala ręcznie wpisywać i wykonywać zapytania w języku SQL, co jest kluczowe przy nauce praktycznej składni.
- Status – wyświetla informacje o obciążeniu serwera i statystykach pracy.
- Konta użytkowników – umożliwia dodawanie i zarządzanie użytkownikami baz danych, przypisywanie im haseł i uprawnień.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



- Import – pozwala wczytywać dane z plików w różnych formatach (np. CSV, SQL).
- Export – umożliwia zapisanie bazy lub tabeli do pliku, co jest szczególnie ważne w kontekście tworzenia kopii zapasowych.
- Ustawienia (Variables/Settings) – pozwalają dostosować parametry pracy serwera bazy danych.
- Replikacja – moduł służący do konfiguracji replikacji baz danych, wykorzystywany w zaawansowanych wdrożeniach.
- Wtyczki (Plugins) – sekcja umożliwiająca zarządzanie dodatkowymi funkcjonalnościami phpMyAdmin.
- Charsets – ta zakładka dotyczy obsługi zestawów znaków i porównań (collation) w bazie danych. Można tutaj podejrzeć i skonfigurować, jakie kodowanie znaków obsługuje serwer (np. UTF-8, latin1). Jest to szczególnie istotne w przypadku baz zawierających dane w różnych językach – niewłaściwy zestaw znaków może prowadzić do błędnego wyświetlania liter, np. polskich znaków diakrytycznych.
- Engines – zakładka pozwala zobaczyć, jakie silniki bazodanowe są obsługiwane przez serwer MySQL/MariaDB (np. InnoDB, MyISAM, MEMORY). Każdy silnik ma swoje właściwości – np. InnoDB obsługuje transakcje i klucze obce, a MyISAM jest szybszy w niektórych zastosowaniach, ale nie zapewnia integralności referencyjnej. W zakładce Engines można również podejrzeć, jaki silnik jest domyślny dla nowo tworzonych tabel.

Warto zwrócić uwagę, że w prawym górnym rogu znajduje się informacja o tym w jakiej bazie danych oraz w jakiej tabeli obecnie się znajdujemy. Wybranie przycisku Serwer 127.0.0.1 zawsze spowoduje przejście na stronę główną serwera i wyjście z aktywnej bazy. Przycisk strzałki znajdujący się w lewym górnym boku spowoduje zamknięcie lub otwarcie listy bocznej z dostępnymi bazami danych. Elementy te zostały pokazane na rysunku 3.

Mając aktywną bazę danych lub tabele z bazy dostajemy dostęp do dodatkowych zakładek opisanych poniżej:



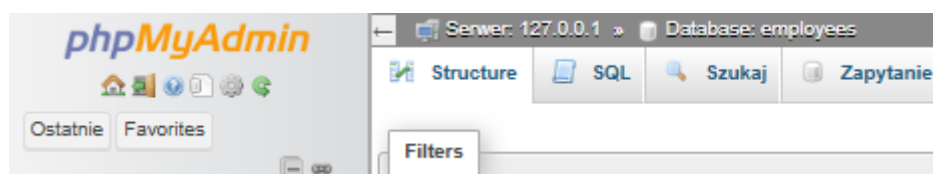
Fundusze Europejskie
dla Rozwoju Społecznego



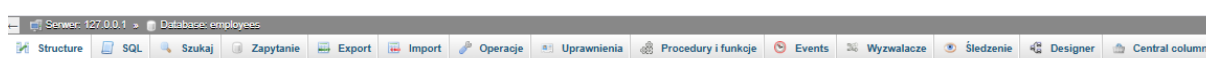
Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską





Rysunek 3. Informacja o aktywnej bazie danych



Rysunek 4. Zakładki dostępne dla bazy danych

Po wejściu do konkretnej bazy danych w phpMyAdmin oprócz zakładek serwerowych pojawiają się dodatkowe, przedstawione na rysunku 4:

- Structure – pokazuje strukturę bazy, czyli listę tabel wraz z ich podstawowymi informacjami (liczba rekordów, typ silnika, zestaw znaków).
- Szukaj – pozwala wyszukiwać dane w całej bazie.
- Zapytanie (Query) – umożliwia skonstruowanie zapytania w graficznym kreatorze, bez konieczności ręcznego pisania SQL.
- Operacje – umożliwia zarządzanie całą bazą, np. zmianę nazwy, zestawu znaków, a także skopiowanie lub usunięcie bazy.
- Uprawnienia – pozwala ustawić prawa dostępu użytkowników do danej bazy danych.
- Procedury i funkcje – umożliwia zarządzanie obiektami z logiką programistyczną zapisanymi w bazie.
- Events – pozwala definiować i zarządzać zdarzeniami cyklicznymi w bazie (scheduler).
- Wyzwalacze – umożliwia przeglądanie i tworzenie triggerów dla tabel w tej bazie.
- Śledzenie (Tracking) – pozwala monitorować zmiany w strukturze i zapytaniach w bazie.
- Designer – narzędzie graficzne do wizualnego projektowania schematu bazy.
- Central columns - służy do centralnego zarządzania kolumnami, które pojawiają się w wielu tabelach. Dzięki niej można definiować wzorce kolumn, a



Fundusze Europejskie
dla Rozwoju Społecznego

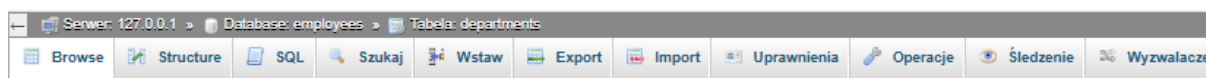


Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



następnie łatwo dodawać je do różnych tabel bez konieczności każdorazowego ręcznego tworzenia od nowa.



Rysunek 5. Zakładki dostępne po wybraniu tabeli.

Jeżeli wybierzemy konkretną tabelę w bazie, otrzymujemy dodatkowy zestaw zakładek przedstawionych na rysunku 5:

- Browse (Przeglądaj) – umożliwia przeglądanie zawartości tabeli w formie rekordów.
- Structure (Struktura) – pokazuje listę kolumn tabeli wraz z ich typami, kluczami i atrybutami.
- Wstaw – pozwala dodać nowy rekord do tabeli poprzez formularz.
- Szukaj – umożliwia wyszukiwanie rekordów w tabeli z warunkami.
- Export – pozwala wyeksportować zawartość tabeli do różnych formatów (np. SQL, CSV, JSON).
- Import – służy do wczytania danych do tabeli z pliku.
- Uprawnienia – daje możliwość zarządzania dostępem użytkowników do danej tabeli.
- Operacje – pozwala wykonać akcje administracyjne na tabeli, np. zmianę nazwy, silnika, usunięcie czy kopiowanie.
- Śledzenie – podobnie jak w bazie, monitoruje zmiany w tabeli.
- Wyzwalacze – pokazuje, jakie triggery są powiązane z tabelą i pozwala dodawać nowe.

Dane do wykorzystania w trakcie zajęć

Na stronie: <https://dev.mysql.com/doc/index-other.html> znajdują się przykładowe bazy danych do pobrania, sugerowane do celów edukacyjnych (rysunek 6). W dalszej pracy na zajęciach będzie wykorzystywana baza sakilla, natomiast na potrzeby wprowadzenia w podstawowe koncepty baz danych, zostanie wykorzystana baza



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



world która cechuje się prostotą, ponieważ zawiera tylko 3 tabele: kontynenty, kraje, miasta.

Example Databases

Title	DB Download	HTML Setup Guide	PDF Setup Guide
employee data (large dataset, includes data and test/verification suite)	GitHub	View	US Ltr A4
world database	TGZ Zip	View	US Ltr A4
sakila database	TGZ Zip	View	US Ltr A4
airportdb database (large dataset, intended for MySQL on OCI and HeatWave)	TGZ Zip	View	US Ltr A4
menagerie database	TGZ Zip		

Rysunek 6. Źródła do pobrania baz danych używanych na zajęciach dostępne na stronie: <https://dev.mysql.com/doc/index-other.html>

Ćwiczenia samodzielne

1. Zainstaluj pakiet XAMPP i uruchom podstawowe moduły: Apache oraz MySQL.
2. Otwórz phpMyAdmin z poziomu przeglądarki (<http://localhost/phpmyadmin>) i zapoznaj się z jego interfejsem.
3. Pobierz przykładową bazę danych world ze strony podanej w instrukcji.
4. Załaduj bazę world do phpMyAdmin. Przetestuj różne metody importu:
 - wgranie pliku SQL w zakładce Import,
 - ręczne wklejenie kodu SQL do zakładki SQL,
 - użycie terminala (np. `mysql -u root -p < world.sql`).
5. Przeglądaj strukturę bazy world i wypisz nazwy wszystkich dostępnych tabel.
6. Zidentyfikuj klucze główne i obce w każdej z tabel (podpowiedź: zwróć uwagę na oznaczenia w zakładce Structure).
7. Dla tabeli City wskaż wszystkie atrybuty wraz z ich typami danych i krótko opisz, do czego mogą służyć.
8. Napisz proste zapytanie SELECT, które pobierze nazwy wszystkich państw z tabeli Country.
9. Utwórz kilka przykładowych zapytań SELECT:
 - Wyświetl wszystkie rekordy z tabeli City – użyj `SELECT * FROM City;`
 - pobierz miasta o populacji większej niż 5 milionów,
 - Pobierz tylko nazwy miast z tabeli City (kolumna Name).
 - Wyświetl nazwy miast oraz liczbę ich mieszkańców (Name, Population).



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



- wypisz wszystkie kraje z kontynentu „Europe”,
 - Pobierz wszystkie kraje z kontynentu „Europe” – `SELECT * FROM Country WHERE Continent = 'Europe';`.
 - Wyświetl wszystkie miasta, które mają populację większą niż 1 000 000 mieszkańców.
 - Pokaż państwa, które mają powierzchnię mniejszą niż 50 000 km².
 - Pobierz wszystkie miasta z kraju „Poland” (kolumna `CountryCode` = 'POL').
 - Wyświetl nazwy miast, które zaczynają się na literę „A” (użyj `LIKE 'A%'`).
 - Pokaż miasta, których nazwy kończą się na literę „n” (`LIKE '%n'`).
 - Wyświetl miasta, których nazwy zawierają literę „burg”.
 - Pobierz kraje, które należą do kontynentów: Europa lub Azja (użyj `IN`).
 - Wyświetl wszystkie miasta o populacji pomiędzy 500 000 a 1 000 000 mieszkańców (operator `BETWEEN`).
 - Wyświetl miasta, dla których nie podano populacji (`Population IS NULL`).
 - Pokaż listę wszystkich kontynentów bez powtórzeń (użyj `DISTINCT Continent`).
 - Wyświetl 10 pierwszych miast z tabeli `City` (`LIMIT 10`).
 - Wyświetl 10 największych miast na świecie według populacji (`ORDER BY Population DESC LIMIT 10`).
 - Wyświetl 10 najmniejszych miast na świecie według populacji (`ORDER BY Population ASC LIMIT 10`).
10. Napisz zapytanie `UPDATE`, które zmieni nazwę wybranego miasta na inną (np. zaktualizuj nazwę miasta testowego).
11. Napisz zapytanie `DELETE`, które usunie wybrany rekord z tabeli `City`. Upewnij się, że rozumiesz konsekwencje usuwania danych.
12. Dodaj nowy rekord do tabeli `City` za pomocą zapytania `INSERT` – wybierz dowolne miasto, określ nazwę, identyfikator kraju i liczbę mieszkańców.
13. Usuń całą bazę `world` z poziomu `phpMyAdmin`.
14. Zadanie projektowe (praca w grupach)
- Wybierzcie w grupie projektowej fragment rzeczywistości (np. uczelnia, biblioteka, sklep internetowy, przychodnia lekarska), który znacie z własnych doświadczeń.
 - Napiszcie wstęp teoretyczny zawierający motywację podjęcia tematu – jakie problemy ma rozwiązać baza danych i dlaczego warto ją stworzyć.
 - Opracujcie wstępny opis słowny bazy, w którym wyodrębnione zostaną najważniejsze encje (tabele) i ich atrybuty.

Pytania pomocnicze



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



1. Czym jest baza danych?
2. Jakie są główne zalety stosowania baz danych?
3. Czym jest encja?
4. Co oznacza atrybut encji?
5. Na czym polega relacja między encjami?
6. Czym różni się klucz główny od klucza obcego?
7. Podaj przykład relacji jeden-do-wielu.
8. Podaj przykład relacji wiele-do-wielu.
9. Co to jest krotka w tabeli?
10. Jak rozumiesz pojęcie schematu bazy danych?
11. Dlaczego redundancja danych jest problematyczna?
12. Czym różni się dane od informacji?
13. Podaj przykład encji z Życia codziennego.
14. Jakie są główne elementy modelu relacyjnego?
15. Jakie znaczenie ma spójność danych?

Podsumowanie

W trakcie pierwszego laboratorium studenci poznali podstawowe pojęcia związane z relacyjnymi bazami danych, takie jak encja, atrybut, tabela, relacja oraz klucze główne i obce, co pozwoliło im zrozumieć fundamenty modelowania danych. Ćwiczenie umożliwiło zapoznanie się ze środowiskiem XAMPP i narzędziem phpMyAdmin, które stanowią praktyczną bazę do dalszej pracy nad projektami i nauki języka SQL. Zrealizowane zadania pozwoliły osiągnąć cele dydaktyczne, takie jak umiejętność tworzenia i przeglądania baz danych, rozpoznawania struktury tabel oraz interpretowania relacji między encjami. Ważnym wnioskiem jest uświadomienie sobie, że baza danych nie jest jedynie zbiorem informacji, lecz systemem, który dzięki spójności i odpowiedniemu modelowaniu wspiera realne procesy techniczne i organizacyjne.

Literatura

1. Elmasri, R., & Navathe, S. B. (2016). Fundamentals of Database Systems (7th ed.). Pearson.
2. Date, C. J. (2019). Database Design and Relational Theory: Normal Forms and All That Jazz (2nd ed.). O'Reilly Media.



Fundusze Europejskie
dla Rozwoju Społecznego

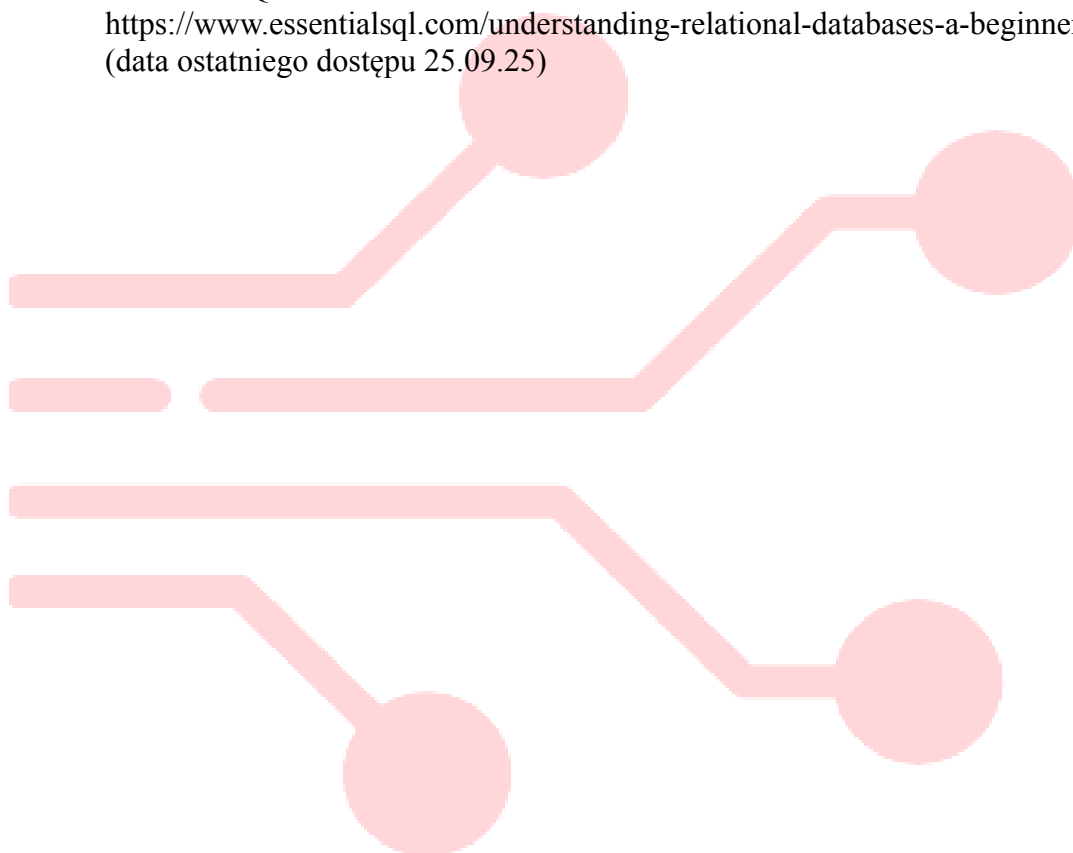


Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



3. Hernandez, M. J. (2013). Database Design for Mere Mortals: A Hands-On Guide to Relational Database Design (3rd ed.). Addison-Wesley.
4. Coronel, C., & Morris, S. (2018). Database Systems: Design, Implementation, & Management (13th ed.). Cengage Learning.
5. Watt, A., & Eng, A. (2020). Database Design (2nd Edition). BCcampus Open Textbook. Dostępne online:
<https://open.umn.edu/opentextbooks/textbooks/database-design-2nd-edition>
6. Relacje w bazie danych: Understanding Relational Databases: A Beginner's Guide - Essential SQL:
<https://www.essentialsql.com/understanding-relational-databases-a-beginners-guide/>
(data ostatniego dostępu 25.09.25)



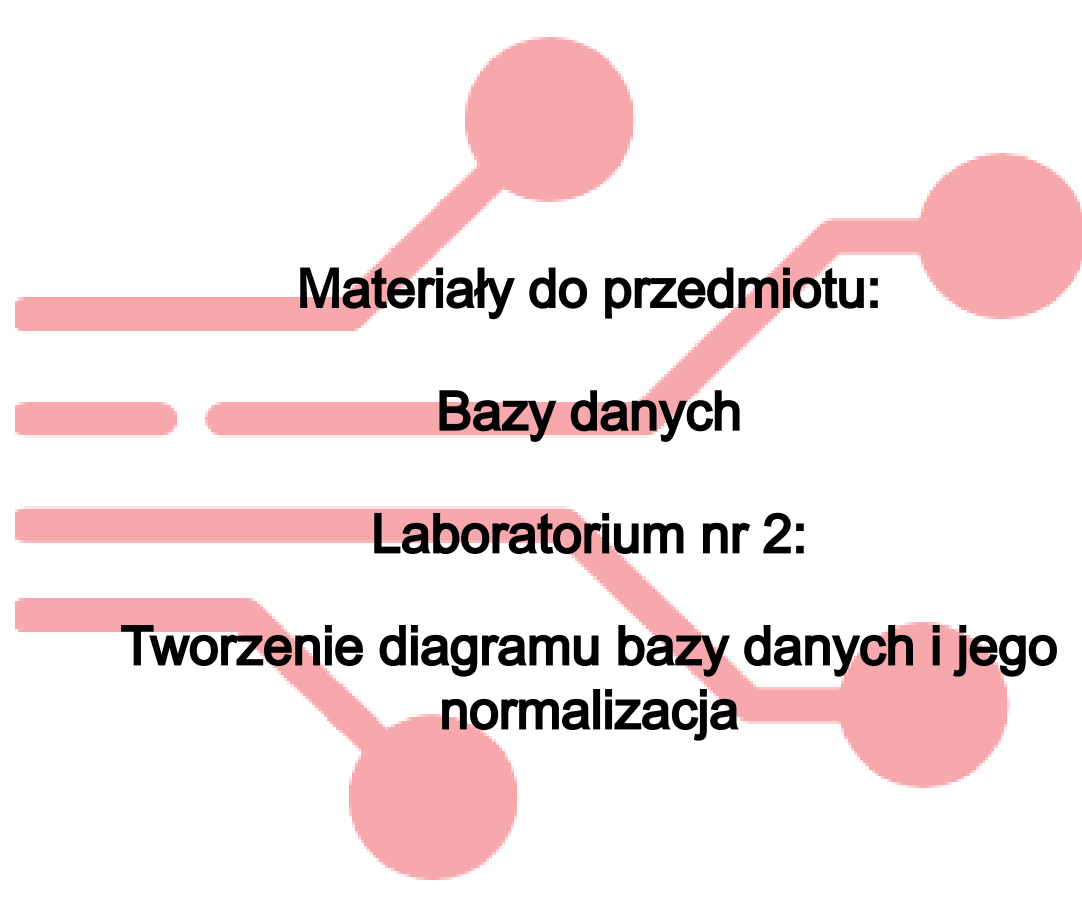
Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską





Materiały do przedmiotu:

Bazy danych

Laboratorium nr 2:

**Tworzenie diagramu bazy danych i jego
normalizacja**

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	26
Pytania pomocnicze	27
Podsumowanie	28
Literatura	28



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Wprowadzenie, tematyka zajęć

Projektowanie bazy danych wymaga nie tylko zdefiniowania encji i relacji, ale także ich graficznej reprezentacji w postaci diagramów. Diagramy ułatwiają zrozumienie struktury systemu i stanowią podstawę do implementacji bazy. Ważnym krokiem jest normalizacja, czyli proces porządkowania danych w taki sposób, aby wyeliminować redundancję i zapewnić spójność. Dzięki temu baza danych staje się bardziej przejrzysta, wydajna i łatwa w utrzymaniu.

Cel ćwiczenia/laboratorium

- Zapoznanie się z diagramem ERD (Entity-Relationship Diagram).
- Umiejętność wskazania encji i atrybutów na diagramie.
- Odróżnianie różnych typów relacji (1:1, 1:N, N:M).
- Zrozumienie pojęcia kardynalności i jej oznaczeń.
- Nauczenie się, jak przenieść diagram ERD do tabel w relacyjnej bazie danych.
- Zrozumienie, czym jest normalizacja.
- Poznanie pojęcia postaci normalnych (1NF, 2NF, 3NF).
- Umiejętność rozpoznawania nieznormalizowanej struktury danych.
- Eliminowanie redundancji w bazie danych.
- Zrozumienie pojęcia zależności funkcyjnych.
- Praktyczne ćwiczenie przejścia od nienormalizowanego modelu do 3NF.
- Poznanie znaczenia kluczy głównych i kandydujących w normalizacji.
- Analiza przykładowych błędów projektowych.
- Wskazanie zalet i wad nadmiernej normalizacji.
- Ćwiczenie samodzielnego tworzenia diagramów baz danych.

Instrukcja laboratoryjna/ćwiczeniowa

Modelowanie bazy danych

Jednym z filarów pracy z bazą danych jest jej poprawne zaprojektowanie, czyli modelowanie. Modelowanie polega na odwzorowaniu fragmentu rzeczywistości w taki sposób, aby baza mogła wiernie reprezentować obiekty i powiązania między nimi. Projektant bazy danych zaczyna od analizy – zastanawia się, jakie informacje są istotne, jakimi cechami te informacje się charakteryzują i w jaki sposób obiekty



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



pozostają ze sobą w relacjach. Modelowanie ma na celu stworzenie struktury, która będzie spójna, pozbawiona zbędnej redundancji oraz łatwa do późniejszej obsługi.

Proces modelowania bazy danych najlepiej rozpocząć od wybrania wycinka rzeczywistości, który chcemy odwzorować. Dla przykładu możemy posłużyć się środowiskiem uczelni. W pierwszym kroku trzeba wyraźnie określić, jakie elementy tego świata będą nas interesować – czy chcemy skupić się na studentach i ich wynikach w nauce, czy może na pracownikach i prowadzonych kursach, a może na całym procesie rejestracji i zaliczania semestrów. Zbyt szeroki zakres utrudni pracę, zbyt wąski może sprawić, że model będzie zbyt prosty i mało praktyczny.

Kiedy wybierzemy już odpowiedni fragment rzeczywistości, kolejnym etapem jest wyodrębnienie obiektów, które w nim istnieją. W przypadku uczelni mogą to być na przykład: Student, Wykładowca, Kurs, Sala, Wydział, Egzamin. Każdy z tych obiektów w świecie realnym może zostać odzwierciedlony w bazie danych w postaci encji. Tutaj ważne jest zastanowienie się, czy encja reprezentuje ogólną abstrakcję obiektu (na przykład Kurs jako pojęcie ogólne) czy też pojedynczą instancję (konkretny kurs prowadzony w semestrze letnim 2024/2025).

Następnie należy opisać atrybuty tych encji, czyli cechy, które są dla nich charakterystyczne. Student będzie miał imię, nazwisko, numer albumu i kierunek studiów; Kurs może być opisany nazwą, kodem i liczbą punktów ECTS; Egzamin będzie miał datę, godzinę i przypisaną salę. Już na tym etapie warto zwrócić uwagę, jakie typy danych są odpowiednie dla poszczególnych atrybutów, aby odzwierciedlały one właściwie charakter przechowywanych informacji.

Kolejnym ważnym krokiem jest ustalenie relacji między encjami. Na uczelni student zapisuje się na kurs, wykładowca prowadzi kurs, a egzamin jest przypisany do konkretnego kursu i konkretnej sali. Takie powiązania trzeba przeanalizować pod kątem tego, czy są to relacje jeden do jednego, jeden do wielu czy wiele do wielu. Relacje wiele do wielu zazwyczaj wymagają wprowadzenia dodatkowej tabeli pośredniczącej, która przechowuje pary powiązanych kluczy.

Na etapie projektowania należy również zastanowić się nad normalizacją danych. Chodzi o to, aby atrybuty były przechowywane w taki sposób, by nie powielać zbędnie informacji i by uniknąć niejednoznaczności. Przykładowo, zamiast wpisywać za każdym razem pełną nazwę wydziału w tabeli Student, lepiej utworzyć tabelę



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Wydział i odwoływać się do niej poprzez klucz obcy. Dzięki temu modyfikacja nazwy wydziału w jednym miejscu zostanie automatycznie odzwierciedlona we wszystkich powiązanych rekordach.

Warto też pamiętać, że proces modelowania nie polega wyłącznie na mechanicznym przekształceniu opisu rzeczywistości w tabele i kolumny. To również refleksja nad tym, jaką rolę pełnią poszczególne encje: czy mają one charakter trwały i autonomiczny (np. Student), czy też istnieją tylko w kontekście innych obiektów (np. Pozycja w planie zajęć, która nie ma sensu bez kursu i sali).

W praktyce dobrze jest przyjąć podejście iteracyjne – najpierw stworzyć prosty model, a następnie go rozwijać i poprawiać wraz z kolejnymi obserwacjami i testami. Już na wstępnym etapie warto też zastanowić się nad wydajnością: jakie atrybuty będą najczęściej używane do wyszukiwania, jakie relacje będą intensywnie wykorzystywane w zapytaniach. To pozwala z wyprzedzeniem przewidzieć potrzebę stosowania indeksów czy odpowiedniego doboru kluczy.

Normalizacja bazy danych

Normalizacja to proces organizowania struktury tabel w relacyjnej bazie danych w taki sposób, aby dane były przechowywane w uporządkowanej, spójnej i pozbawionej nadmiarowości formie. Jej głównym celem jest eliminacja redundancji (powielania informacji), poprawa integralności danych oraz ułatwienie ich utrzymania i rozwijania.

Idea normalizacji została zaproponowana przez Edgara F. Codd'a, twórcę modelu relacyjnego. Polega ona na dzieleniu danych na logiczne tabele i definiowaniu między nimi odpowiednich zależności (kluczy). Proces normalizacji opisuje się za pomocą tzw. postaci normalnych (ang. normal forms), które stanowią kolejne stopnie uporządkowania bazy danych.

1. Pierwsza postać normalna (1NF)

Tabela spełnia 1NF, jeśli:

- wszystkie kolumny zawierają wartości atomowe (niepodzielne),
- w każdej kolumnie przechowywany jest tylko jeden typ danych,



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



- nie ma powtarzających się grup kolumn.

Przykład: zamiast kolumny telefony z wpisem „123456, 987654”, należy utworzyć osobną tabelę z numerami telefonów, powiązaną z tabelą klientów.

2. Druga postać normalna (2NF)

Tabela spełnia 2NF, jeśli:

- jest w 1NF,
- każdy atrybut niekluczowy zależy od całego klucza głównego, a nie tylko od jego części (dotyczy kluczy złożonych).

Przykład: w tabeli zapisów studentów na kursy, jeśli kluczem jest para (id_studenta, id_kursu), to kolumna nazwisko studenta zależy tylko od id_studenta, a nie od całego klucza. Powinna więc zostać przeniesiona do tabeli Studenci.

3. Trzecia postać normalna (3NF)

Tabela spełnia 3NF, jeśli:

- jest w 2NF,
- żaden atrybut niekluczowy nie zależy od innego atrybutu niekluczowego (brak tzw. zależności przechodnich).

Przykład: w tabeli Pracownicy mamy kolumny: id_pracownika, id_działu, nazwa_działu. Kolumna nazwa_działu zależy od id_działu, a nie bezpośrednio od id_pracownika. Dlatego należy utworzyć osobną tabelę Działy i tam przechowywać nazwę działu.

4. Postać normalna Boyce’a–Codda (BCNF)

BCNF jest wzmocnieniem 3NF.

Tabela spełnia BCNF, jeśli:

- dla każdej nietrywialnej zależności funkcyjnej $X \rightarrow Y$, zbiór atrybutów X jest nadkluczem.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



W praktyce oznacza to, że każda kolumna jest w pełni zależna od klucza głównego i nie występują bardziej subtelne zależności funkcyjne, które mogą powodować anomalie.

5. Czwarta postać normalna (4NF)

Tabela spełnia 4NF, jeśli:

- jest w BCNF,
- nie zawiera wielowartościowych zależności, czyli sytuacji, w której jeden rekord wiąże się niezależnie z wieloma wartościami dwóch różnych atrybutów.

Przykład: jeśli pracownik może mieć wiele umiejętności i wiele projektów, nie powinno się tego przechowywać w jednej tabeli Pracownik–Umiejętność–Projekt. Należy rozdzielić na dwie tabele: Pracownik–Umiejętność oraz Pracownik–Projekt.

6. Piąta postać normalna (5NF) – postać projekcji–złączenia

Tabela spełnia 5NF, jeśli:

- jest w 4NF,
- nie zawiera anomalii związanych z dekompozycją i rekonstrukcją danych.

Oznacza to, że dane nie powinny być dzielone na mniejsze fragmenty w sposób, który prowadziłby do błędnych wyników przy ich ponownym łączeniu.

7. Szósta postać normalna (6NF)

6NF jest stosowana bardzo rzadko, głównie w hurtowniach danych i systemach temporalnych (przechowujących historię zmian). Polega na dalszym rozbijaniu tabel tak, aby każdy atrybut był w pełni niezależny czasowo. W praktyce oznacza to, że dane są przechowywane w maksymalnie szczegółowej, rozbitej formie, a łączenie odbywa się dopiero przy zapytaniach.

W większości systemów komercyjnych i edukacyjnych normalizacja kończy się na 3NF lub BCNF. To poziom, który zapewnia równowagę między spójnością danych a



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



wydajnością działania bazy. 4NF i 5NF stosuje się w bardzo złożonych systemach, gdzie występują nietypowe zależności wielowartościowe. 6NF spotykana jest głównie w hurtowniach danych i specjalistycznych rozwiązaniach analitycznych.

Więzy integralności

Pojęcie więzów integralności (integrity constraints) odnosi się do zbioru reguł, które system bazodanowy egzekwuje po to, by dane były poprawne, spójne i wiernie odzwierciedlały rzeczywistość. Dzięki nim baza nie przechowa rekordu niekompletnego, sprzecznego z logiką biznesową albo odwołującego się do nieistniejącego obiektu. W praktyce więzy integralności działają jak wbudowane „testy” jakości danych: pilnują kompletności, unikalności, dozwolonych zakresów wartości oraz poprawnych powiązań między tabelami. Można je definiować już w chwili tworzenia tabel lub dodać później; warto nadawać im czytelne nazwy, bo ułatwia to przegląd, modyfikację i diagnostykę.

Najbardziej rozpoznawalnym więzem jest PRIMARY KEY. To atrybut (lub zestaw atrybutów) jednoznacznie identyfikujący każdy wiersz. Integralność encji oznacza tu: brak duplikatów i brak wartości pustych dla klucza. W wielu modelach używa się kluczy sztucznych (surrogate keys), np. liczników lub identyfikatorów UUID, ale klucz naturalny (wynikający z domeny problemu) też bywa właściwy – ważne, by rzeczywiście gwarantował niepowtarzalność. Gdy pojedyncza kolumna nie wystarcza, definiuje się klucz złożony, np. (student_id, course_id, year) w tabeli zapisów na kurs.

NOT NULL odpowiada za kompletność danych w kolumnach, w których brak wartości byłby bez sensu (np. hasło użytkownika czy data wystawienia faktury). UNIQUE wymusza niepowtarzalność wartości w danej kolumnie lub kombinacji kolumn, ale – w odróżnieniu od PRIMARY KEY – tabela może mieć wiele różnych ograniczeń UNIQUE i nie muszą one identyfikować wiersza jako „tego jednego” w sensie klucza głównego. Często łączy się NOT NULL z UNIQUE, aby zachować unikalność bez dopuszczania pustych wartości.

DEFAULT to domyślna wartość nadawana wtedy, gdy użytkownik nie poda żadnej. Dzięki temu dane pozostają kompletne logicznie (np. status „Nowy”, flaga „aktywny = TRUE”, kierunek „Undeclared” u studenta bez deklaracji). DEFAULT nie „naprawia”



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



błędów merytorycznych, ale ogranicza liczbę braków w danych i ujednolica zachowanie aplikacji.

CHECK to wszechstronny mechanizm kontroli integralności dziedziny (domain integrity). Pozwala zapisać warunki, które musi spełniać zawartość kolumny lub relacja między kilkoma kolumnami: zakresy liczbowe (0–100), listy dozwolonych wartości (np. stany zamówienia), zależności logiczne (gdy grade $\geq$ 60, wtedy is_passing = TRUE, a gdy grade jest nieznana, is_passing też jest nieznane). CHECK działa blisko danych i jest niezależny od aplikacji – dzięki temu reguły obowiązują wszędzie, niezależnie od tego, z jakiego programu ktoś zapisuje rekordy.

FOREIGN KEY utrzymuje integralność referencyjną między tabelami: kolumna w tabeli podrzędnej musi wskazywać na istniejący klucz główny (lub unikalny) w tabeli nadrzędnej. Ten więc zapobiega „wiszącym” odwołaniom (np. zapis na kurs do nieistniejącego studenta). Warto rozumieć akcje referencyjne: ON DELETE/UPDATE RESTRICT (zabroń operacji, jeśli są zależne wiersze), CASCADE (przenieś zmianę albo usuń powiązane wiersze), SET NULL / SET DEFAULT (zneutralizuj powiązanie). Dobór akcji to decyzja projektowa – w systemie finansowym częściej ograniczamy kasowanie (RESTRICT), a w słownikach pomocniczych czasem dopuszczamy SET NULL.

Z punktu widzenia projektanta można zgrupować te więzy według obszarów: integralność encji (PRIMARY KEY), integralność dziedziny (NOT NULL, DEFAULT, CHECK, UNIQUE) i integralność referencyjna (FOREIGN KEY). Razem tworzą one siatkę zabezpieczeń, która sprawia, że błędy są zatrzymywane „u źródła”, jeszcze zanim trafią do raportów, modeli analitycznych lub zewnętrznych systemów. To ważne, bo późniejsze korygowanie danych jest kosztowne i ryzykowne.

W projektowaniu baz danych należy stosować zestaw ugruntowanych dobrych praktyk, które zapewniają wysoką jakość i spójność danych. Przede wszystkim wszelkie więzy integralności powinny być definiowane bezpośrednio w bazie danych, a nie wyłącznie na poziomie aplikacji, ponieważ gwarantuje to ich egzekwowanie niezależnie od źródła wprowadzania danych. Wszystkie więzy powinny być ponadto jednoznacznie i opisowo nazwane, aby ułatwiać ich późniejsze utrzymanie i zarządzanie. Projektant powinien w sposób świadomy rozważyć dopuszczalność wartości NULL, pamiętając, że w niektórych przypadkach brak wartości sam w sobie



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



stanowi istotną informację. Weryfikację listy dopuszczalnych wartości zaleca się realizować przy pomocy ograniczeń CHECK albo poprzez wykorzystanie tabel słownikowych. W przypadku więzów referencyjnych (kluczy obcych) należy starannie zaplanować akcje towarzyszące operacjom ON DELETE i ON UPDATE, tak aby odpowiadały one logice biznesowej systemu. Wreszcie, należy pamiętać, że więzy integralności działają w ścisłej współpracy z mechanizmem transakcji, co umożliwia zapewnienie atomowości zmian – całość operacji może zostać zatwierdzona albo w całości wycofana, co istotnie zwiększa spójność i bezpieczeństwo danych.

Zależność funkcjonalna (functional dependency) to formalne pojęcie teorii baz danych, które opisuje powiązania pomiędzy atrybutami w relacyjnej bazie danych. Mówimy, że w tabeli zachodzi zależność funkcjonalna $X \rightarrow Y$, jeżeli każdej wartości (lub kombinacji wartości) atrybutów ze zbioru X odpowiada dokładnie jedna wartość (lub kombinacja wartości) atrybutów ze zbioru Y. Innymi słowy, wartości kolumny Y są jednoznacznie wyznaczone przez wartości kolumny X.

Przykład: w tabeli Pracownicy atrybut PESEL jednoznacznie określa Imię, Nazwisko i DataUrodzenia. Oznacza to, że zachodzi zależność $PESEL \rightarrow (Imię, Nazwisko, DataUrodzenia)$. Natomiast odwrotna zależność, np. $Imię \rightarrow PESEL$, nie zachodzi, ponieważ wielu pracowników może mieć to samo imię.

Zależności funkcjonalne są podstawą do analizy poprawności projektu bazy danych i procesu normalizacji. To właśnie one decydują o tym, jak należy podzielić atrybuty na tabele, aby uniknąć redundancji i anomalii przy aktualizacji danych.

Relacje identyfikujące a nieidentyfikujące

W modelowaniu danych wyróżnia się dwa podstawowe typy relacji między encjami: identyfikujące i nieidentyfikujące. Różnica między nimi dotyczy sposobu, w jaki encja podrzędna powiązana jest z encją nadrzędną oraz tego, czy klucz obcy staje się częścią jej klucza głównego.

Relacja identyfikująca oznacza, że encja podrzędna nie może istnieć samodzielnie i jej tożsamość wynika bezpośrednio z powiązania z encją nadrzędną. W praktyce oznacza to, że klucz obcy w tabeli podrzędnej wchodzi w skład jej klucza głównego. Przykładem może być relacja pomiędzy zamówieniem a jego pozycjami: pojedyncza pozycja zamówienia nie ma sensu bez samego zamówienia, a jej identyfikator



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



tworzony jest zwykle jako kombinacja numeru zamówienia oraz identyfikatora produktu. Na diagramach ERD takie relacje przedstawiane są zazwyczaj linią ciągłą, aby podkreślić ich „mocniejsze” powiązanie.

Relacja nieidentyfikująca jest luźniejsza. W tym przypadku encja podrzędna posiada własny niezależny identyfikator, a powiązanie z encją nadrzędną jest realizowane wyłącznie poprzez klucz obcy, który nie wchodzi do klucza głównego. Przykładem jest relacja pomiędzy klientem a zamówieniami: każde zamówienie ma swój własny numer, a klucz obcy do klienta jedynie wskazuje, kto je złożył. Zamówienie wciąż pozostaje odrębną jednostką i jego identyfikator nie zależy od tego, do jakiego klienta jest przypisane. W diagramach ERD relacje nieidentyfikujące przedstawiane są zwykle linią przerywaną.

W praktycznych projektach częściej stosuje się relacje nieidentyfikujące. Dzieje się tak dlatego, że pozwalają one utrzymać proste, jednoczęściowe klucze główne, najczęściej realizowane w postaci automatycznie inkrementowanego identyfikatora. Upraszcza to zarówno budowę zapytań SQL, jak i implementację w systemach korzystających z warstw pośrednich, na przykład w aplikacjach wykorzystujących frameworki ORM. Relacje identyfikujące pojawiają się raczej w sytuacjach, gdy logiczny sens encji podrzędnej istnieje wyłącznie w powiązaniu z encją nadrzędną i chcemy tę zależność wyraźnie zaznaczyć w strukturze klucza.

Przechowywanie kluczy obcych

Jednym z kluczowych zagadnień w modelowaniu relacyjnych baz danych jest prawidłowe przypisanie miejsca, w którym powinien znajdować się klucz obcy. To właśnie klucze obce odpowiadają za powiązania między tabelami i decydują, w jaki sposób dane ze sobą współpracują. Błędne ułożenie klucza prowadzi do trudności w utrzymaniu spójności, a także do niejasności w interpretacji modelu.

W przypadku relacji jeden do jednego (1:1) ogólna zasada mówi, że klucz obcy przechowuje ta tabela, która jest rozszerzeniem lub uzupełnieniem drugiej. Innymi słowy, jeśli jedna encja nie może istnieć bez drugiej, to właśnie ona powinna wskazywać klucz. Dobrym przykładem jest relacja Student – Legitymacja studencka. Legitymacja istnieje wyłącznie wtedy, gdy istnieje student, dlatego klucz obcy umieszczamy w tabeli Legitymacja. Podobnie w relacji Użytkownik – Ustawienia konta, to tabela Ustawienia przechowuje klucz obcy do Użytkownika, ponieważ sama



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



w sobie nie ma sensu istnienia bez niego. Z kolei, gdy dane są bardziej symetryczne, decyzję podejmuje się na podstawie logiki biznesowej i wygody późniejszego użytkownika.

Jeszcze wyraźniejsza jest sytuacja w przypadku relacji jeden do wielu (1:N). Tutaj zasada jest niezmienna: klucz obcy znajduje się po stronie „wielu”. To rekordy wtórne muszą wskazywać rekord nadrzędny, do którego należą. Przykładowo, w relacji Autor – Książka to tabela Książka przechowuje klucz obcy do tabeli Autor, ponieważ każda książka ma swojego autora, ale jeden autor może napisać wiele książek. Podobnie, w relacji Produkt – Opinia, to tabela Opinia trzyma klucz obcy do tabeli Produkt, ponieważ opinia nie istnieje w oderwaniu od produktu, którego dotyczy. Warto podkreślić, że podczas modelowania należy bardzo uważnie określić kierunek tej relacji. Odwrócenie jej prowadzi do błędnego odwzorowania logiki rzeczywistości, a w konsekwencji do niepoprawnego działania całego systemu.

Wreszcie, w przypadku relacji wiele do wielu (N:M), konieczne jest wprowadzenie tabeli pośredniczącej. To ona przechowuje pary kluczy obcych wskazujące jednocześnie na obie tabele główne. Tego typu konstrukcja jest powszechnie stosowana w sytuacjach, gdy jeden obiekt może być powiązany z wieloma innymi i odwrotnie – na przykład Studenci – Kursy. Student może uczęszczać na wiele kursów, a kurs może być realizowany przez wielu studentów. Dlatego relacja ta wymaga osobnej tabeli, np. Zapisy, w której przechowywane będą kombinacje kluczy: id_studenta oraz id_kursu. Warto dodać, że w narzędziach takich jak MySQL Workbench tabela pośrednicząca tworzona jest automatycznie, gdy użytkownik definiuje relację wiele do wielu. Jednak przy ręcznym pisaniu skryptów SQL lub korzystaniu z frameworków i bibliotek ORM (np. Hibernate, SQLAlchemy) trzeba samodzielnie pamiętać o jej utworzeniu i poprawnym zdefiniowaniu kluczy obcych.

Zapoznanie środowiska Workbench

Po uruchomieniu MySQL Workbench (rysunek 1) u góry aplikacji widzimy główny pasek menu, który pełni rolę centrum sterowania wszystkimi funkcjami programu. Warto wiedzieć, do czego służą poszczególne zakładki, bo to właśnie z ich poziomu możemy rozpocząć modelowanie bazy, otwierać zapytania SQL czy zarządzać połączeniami.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



1. File (Plik)

To tutaj tworzymy i otwieramy nowe elementy. Możemy rozpocząć pracę nad nowym modelem bazy danych (New Model), otworzyć nowe zapytanie SQL (New Query Tab), załadować istniejący model, skrypt lub plik projektu. Z tego menu zamykamy też poszczególne zakładki albo cały program.

Najczęściej używane w laboratorium: New Model (do projektowania ERD) i New Query Tab (do pisania zapytań SQL).

2. Edit (Edycja)

Zawiera standardowe operacje, takie jak kopiowanie, wklejanie, cofanie czy powtarzanie akcji. Dodatkowo znajdziemy tu ustawienia związane z preferencjami pracy w Workbenchu, np. konfigurację edytora SQL.

3. View (Widok)

Odpowiada za to, co jest widoczne w interfejsie Workbenchu. Możemy włączać i wyłączać panele (np. panel obiektów, panel logów SQL), a także przełączać się między trybami widoku. To przydatne, gdy chcemy dostosować środowisko do własnych potrzeb.

4. Database (Baza danych)

To jedna z najważniejszych zakładek. Umożliwia tworzenie nowych połączeń z serwerem MySQL, synchronizowanie modeli z istniejącą bazą, wykonywanie forward engineering (czyli generowanie schematu bazy na podstawie modelu ERD) i reverse engineering (czyli odtwarzanie modelu z istniejącej bazy).

5. Tools (Narzędzia)

W tej sekcji znajdziemy dodatkowe funkcje – np. konfigurację serwera, monitorowanie jego stanu, narzędzia do migracji danych czy do zarządzania użytkownikami.

6. Scripting (Skrypty)



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

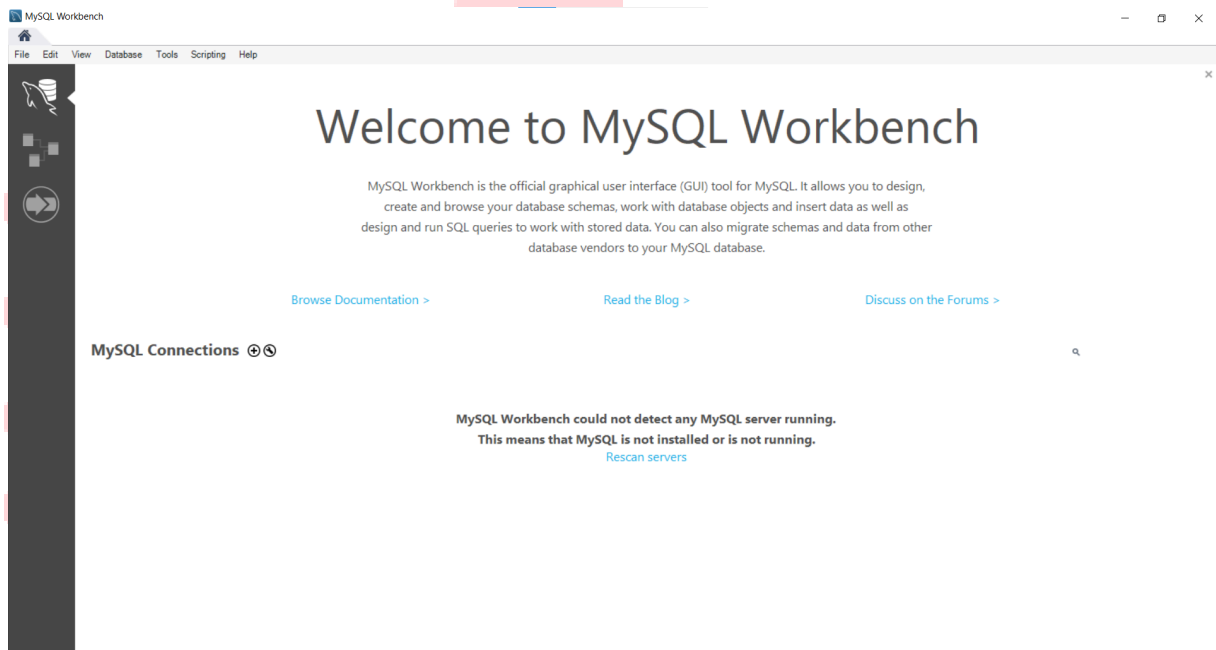
Dofinansowane przez
Unię Europejską



Pozwala uruchamiać własne skrypty w Pythonie lub Lua bezpośrednio z Workbench. Jest to bardziej zaawansowana funkcjonalność, zwykle nieużywana na podstawowych laboratoriach, ale warto wspomnieć, że Workbench daje taką elastyczność.

7. Help (Pomoc)

Oferuje dostęp do dokumentacji MySQL Workbench, samouczków, a także informacji o wersji programu. W razie problemów jest to pierwsze miejsce, do którego student może zajrzeć.



Rys. 1. Widok startowy Workbench



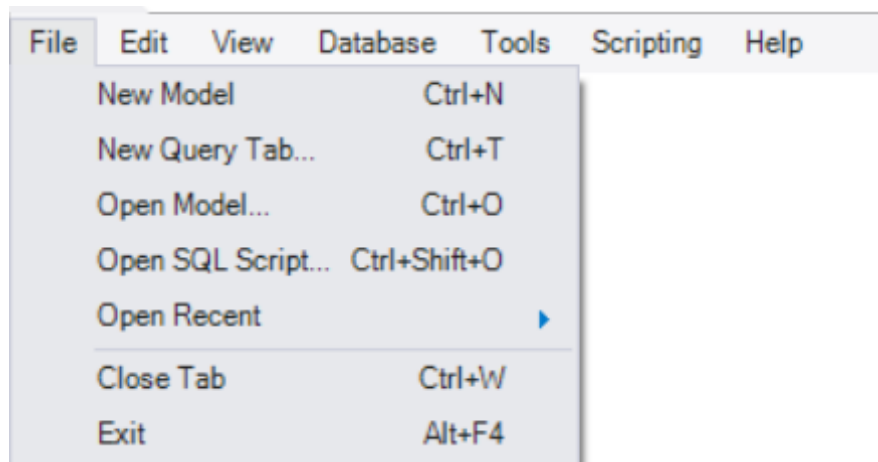
Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

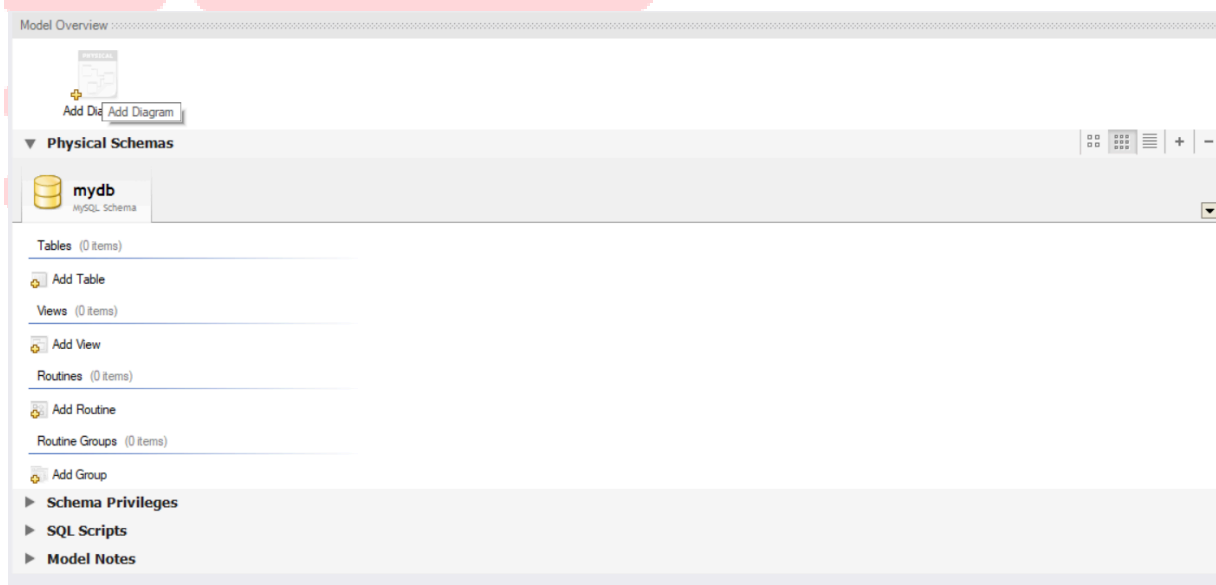
Dofinansowane przez
Unię Europejską





Rys. 2. Pasek narzędzi Workbench, zakładka Plik posiada przyciski służące np. tworzeniu nowego diagramu lub otwarciu istniejącego.

Aby utworzyć pusty model bazy danych należy wybrać opcję New Model z zakładki File, co zostało przedstawione na rysunku 2. Następnie trzeba wybrać opcję dodania nowego diagramu przedstawioną na rysunku 3.



Rys. 3. Tworzenie nowego diagramu.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Rysunek 4 przedstawia obszar roboczy przeznaczony do tworzenia diagramu. Widać po jego lewej stronie przyciski służące do zarządzania diagramem ERD. Są to kolejno patrząc od góry:

1. Select (strzałka) – pozwala zaznaczać elementy w diagramie (tabele, relacje, notatki) i je przesuwać.
2. Place a New Table (ikona tabeli) – służy do dodania nowej tabeli do diagramu. Po kliknięciu możesz zdefiniować jej nazwę, kolumny, klucze i atrybuty.
3. Place a View (ikona prostokąta z „V”) – dodaje widok (VIEW) do modelu, czyli zapisaną kwerendę SQL, którą można traktować jak tabelę logiczną.
4. Place a Routine Group (ikona kółka) – umożliwia grupowanie procedur składowanych i funkcji w modelu, dla porządku logicznego.
5. Place a Layer (ikona prostokąta z warstwami) – pozwala dodawać warstwy, czyli „obszary logiczne” w diagramie, które grupują tabele np. według modułów systemu (np. moduł studenci, moduł finanse).
6. Place a Note (ikona karteczki) – dodaje notatkę do diagramu. To bardzo praktyczne do opisywania części modelu czy wyjaśnień dla innych osób.
7. Place a Text Object (ikona „T”) – umożliwia dodanie bloku tekstu w dowolnym miejscu diagramu. Można tu wprowadzać komentarze czy dodatkowe objaśnienia.
8. Place a Relationship (1:1) – rysuje relację jeden do jednego (1:1) między tabelami.
9. Place a Relationship (1:N) – rysuje relację jeden do wielu (1:N) między tabelami. Najczęściej używany typ powiązania.
10. Place a Relationship (N:M) – rysuje relację wiele do wielu (N:M), która automatycznie tworzy tabelę pośrednią.
11. Place a Relationship (identifying 1:1) – tworzy relację identyfikującą 1:1, czyli taką, gdzie encja podrzędna nie może istnieć bez encji nadrzędnej.
12. Place a Relationship (identifying 1:N) – tworzy relację identyfikującą 1:N. W tabeli podrzędnej klucz obcy staje się częścią klucza głównego.
13. Place a Relationship (identifying N:M) – tworzy relację identyfikującą wiele do wielu (N:M), czyli związek, gdzie dodatkowa tabela asocjacyjna jest integralną częścią powiązania mapując klucze obu powiązanych tabel.



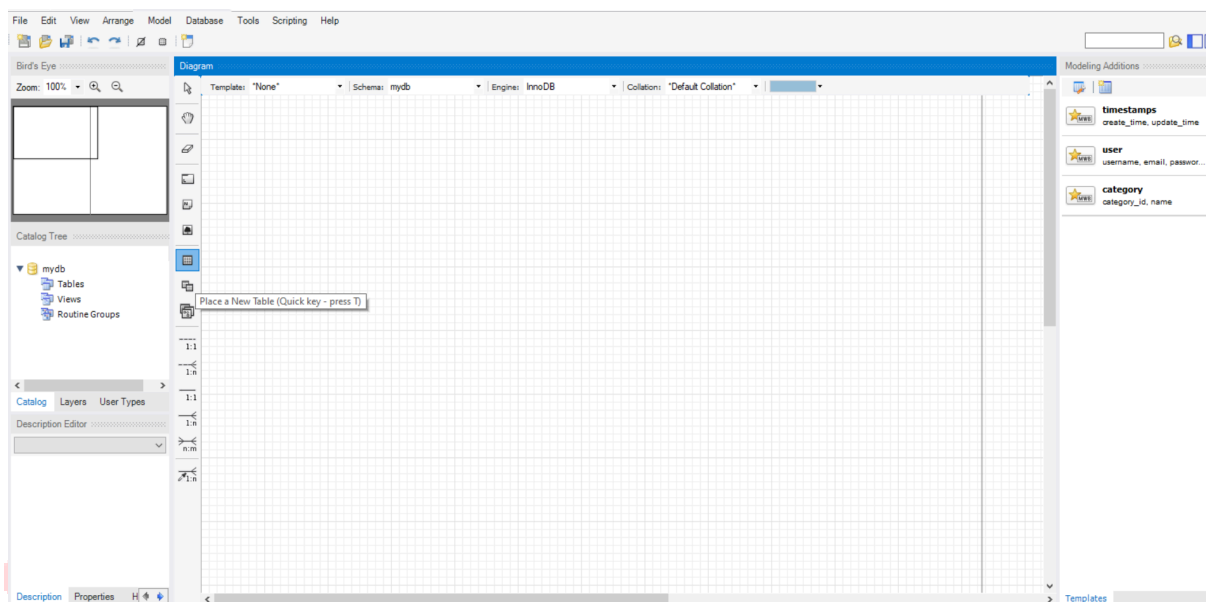
Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską





Rys. 4. Obszar roboczy Workbench.

W praktyce rozpoczęcie pracy w Workbenchu zaczyna się od wstawienia kilku tabel, dodania im atrybutów oraz połączenia relacjami, zostało to przedstawione na rysunku 5. Aby utworzyć nową tabelę, wystarczy skorzystać z ikony Place a New Table w pasku narzędzi (skrót klawiszowy – T), a następnie kliknąć w diagramie w miejscu gdzie chcemy aby tabela się pojawiła. Następnie tabela pojawia się jako obiekt graficzny, a po kliknięciu w nią dwukrotnie na dolnym panelu otwiera się edytor, w którym możemy nadać jej nazwę oraz dodać kolumny. W tabeli która nie posiada jeszcze atrybutów dwukrotne kliknięcie w nazwę kolumny automatycznie oznacza ją jako klucz główny (Primary Key) – co jest podstawowym krokiem w modelowaniu relacyjnej bazy danych.

W każdej kolumnie należy określić nazwę, typ danych (np. INT, VARCHAR, DATE), a także dodatkowe właściwości:

- PK (Primary Key) – oznaczenie kolumny jako klucza głównego.
- NN (Not Null) – kolumna nie może zawierać wartości pustych, co gwarantuje pełność danych.



Fundusze Europejskie
dla Rozwoju Społecznego



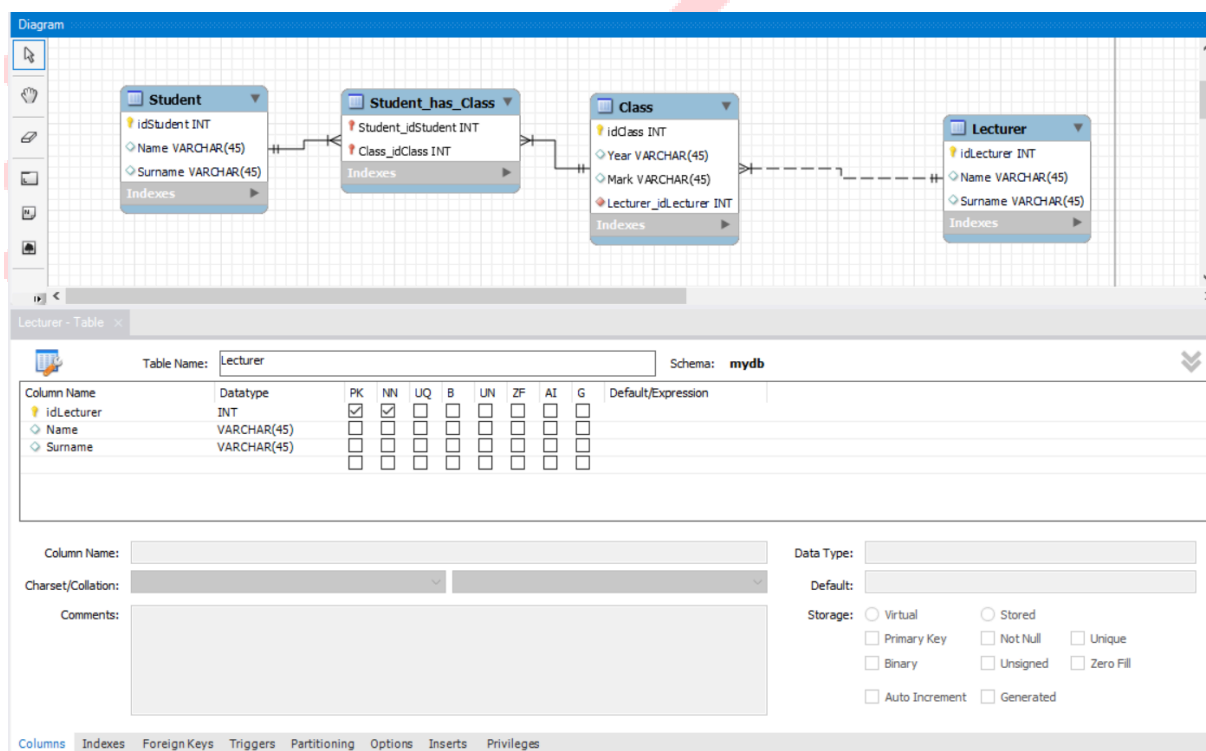
Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



- UQ (Unique) – wymusza unikalność wartości w kolumnie (np. adres e-mail użytkownika).
- B (Binary) – kolumna przechowuje dane w formacie binarnym.
- UN (Unsigned) – liczby nieujemne (np. populacja, ID).
- ZF (Zero Fill) – liczby uzupełniane wiodącymi zerami (np. 000123).
- AI (Auto Increment) – automatyczne zwiększanie wartości przy każdym nowym rekordzie (najczęściej stosowane w kluczach głównych).
- G (Generated) – oznacza kolumnę obliczaną na podstawie innych danych (np. suma, różnica, concatenacja).

Po zdefiniowaniu kolumn i ich właściwości, kolejnym krokiem jest łączenie tabel relacjami. Workbench pozwala tworzyć wszystkie typy relacji. Dzięki tym mechanizmom MySQL Workbench umożliwia stworzenie kompletnego diagramu ERD, który następnie można wyeksportować do skryptu SQL i załadować bezpośrednio do serwera bazy danych.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Rys. 5. Przykład prostego schematu ERD

MySQL Workbench jest narzędziem służącym do projektowania, wizualizacji oraz zarządzania bazami danych, ale sam w sobie nie jest silnikiem bazodanowym. Oznacza to, że nie przechowuje danych ani nie wykonuje zapytań niezależnie – pełni raczej funkcję graficznego frontendu. Aby faktycznie działał z danymi, musi być połączony z systemem zarządzania bazą danych (DBMS), takim jak MySQL lub MariaDB. Dzięki temu możliwe jest projektowanie schematów w trybie offline, a następnie synchronizowanie ich z rzeczywistą bazą. Jedną z kluczowych funkcji Workbench jest możliwość eksportu modelu do kodu SQL (tzw. Forward Engineer), przedstawiony na rysunku 6, co pozwala wygenerować gotowy skrypt CREATE TABLE, ALTER TABLE i definicji kluczy obcych. Wygenerowany kod można następnie uruchomić w innym środowisku – np. w phpMyAdmin, który działa jako aplikacja webowa do zarządzania serwerem MySQL/MariaDB. W ten sposób Workbench staje się narzędziem wspierającym cały proces: od projektowania koncepcyjnego (diagramy ERD), przez dokumentację (eksport schematu w formacie PDF, PNG), aż po implementację (eksport do SQL). W praktyce oznacza to, że student czy projektant może najpierw zaprojektować bazę w Workbenchu, a następnie łatwo przenieść ją na serwer działający w XAMPP, WAMP czy LAMP i kontynuować pracę w phpMyAdmin, mając pewność, że struktura tabel i relacji została odwzorowana zgodnie z modelem.



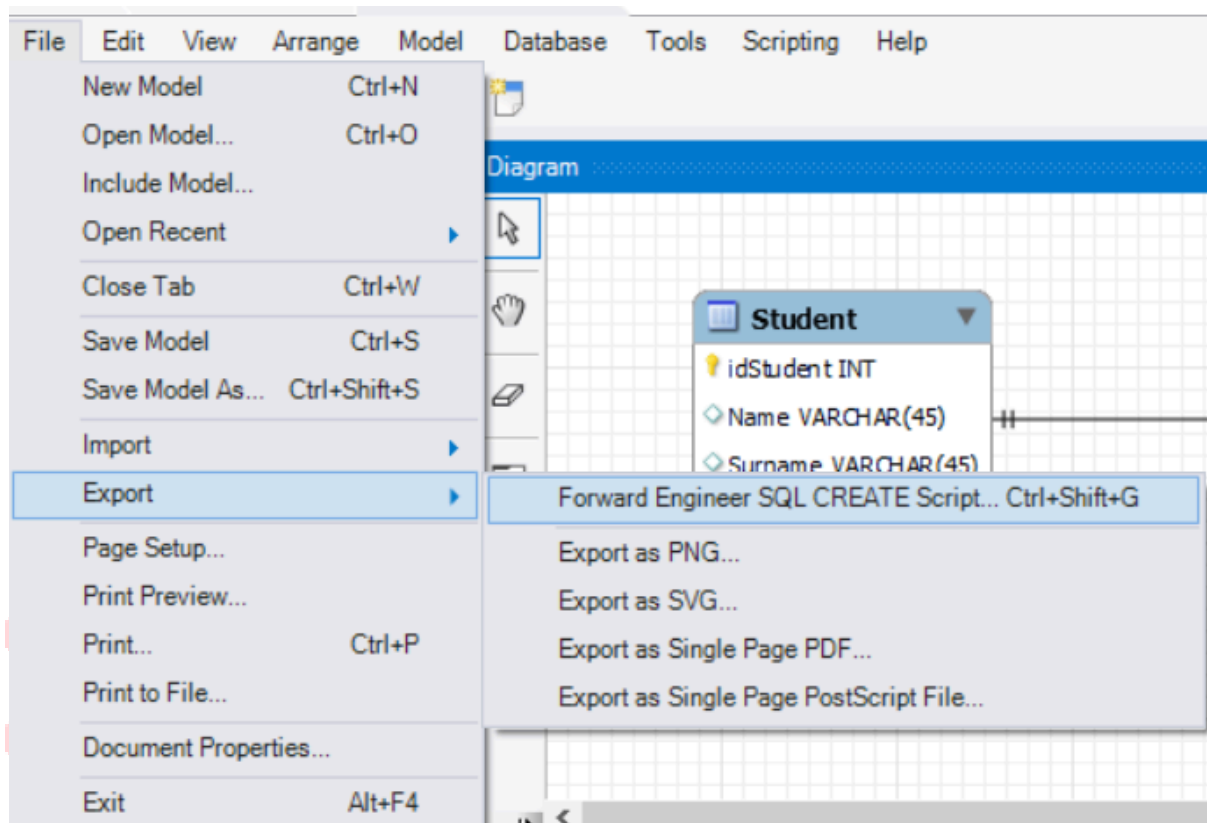
Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską





Rys. 6. Narzędzie Forward Engineer

Połączenie bazy danych z zewnętrznymi aplikacjami i interfejsami

Połączenie z bazą danych w Pythonie realizuje się poprzez tzw. uchwyt do bazy danych (database connection handle), który stanowi podstawowy interfejs komunikacyjny między aplikacją a systemem zarządzania bazą danych (DBMS).

Uchwyt pozwala wykonywać zapytania SQL, wstawiać dane, modyfikować rekordy oraz pobierać wyniki zapytań w sposób bezpieczny i kontrolowany.

W praktyce uchwyt tworzony jest za pomocą dedykowanego modułu (np. mysql.connector, sqlite3, psycopg2), który udostępnia zestaw funkcji zgodny z tzw. DB-API 2.0 (ang. Database Application Programming Interface).



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Standard ten określa spójny sposób pracy z różnymi bazami danych, dzięki czemu kod programu może być łatwo przenoszony między systemami MySQL, PostgreSQL czy SQLite.

Tworzenie uchwytu powinno przebiegać według kilku dobrych praktyk:

1. Parametry połączenia powinny być konfigurowalne, a nie zapisane bezpośrednio w kodzie źródłowym. Zaleca się przechowywanie danych dostępowych (login, hasło, host, port) w pliku `.env` lub w zmiennych środowiskowych, aby nie narażać ich na przypadkowe ujawnienie.
2. Połączenie powinno być zamykane po zakończeniu operacji. Niezamknięte uchwytów prowadzą do wycieków zasobów (ang. resource leaks) i mogą blokować dostęp innych użytkowników. W Pythonie można użyć konstrukcji `try/finally` lub menedżera kontekstu (`with`), który automatycznie zwalnia połączenie.
3. Każde połączenie powinno tworzyć własny kursor. Kursor reprezentuje aktualny kontekst zapytania SQL i pozwala na bezpieczne wykonywanie wielu operacji w ramach jednego połączenia.
4. Zaleca się używanie zapytań parametryzowanych. Zamiast łączyć ciągi tekstowe w stylu `"... WHERE name=" + user_input + ""`, należy korzystać z symboli zastępczych (`%s` lub `?`). Chroni to przed atakami typu SQL Injection i poprawia bezpieczeństwo aplikacji.
5. Transakcje powinny być jawnie zatwierdzone (`commit`) lub wycofywane (`rollback`). Daje to pełną kontrolę nad spójnością danych i pozwala przywrócić stan bazy po wystąpieniu błędu.
6. Obsługa błędów powinna być realizowana przez wyjątki. Moduły bazodanowe Pythona zgłaszają wyjątki (np. `mysql.connector.Error`), które należy przechwytywać, aby program reagował w kontrolowany sposób — np. zapisem błędu do logu lub wycofaniem transakcji.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Poniżej znajduje się przykład jak połączyć się z bazą MySQL za pomocą języka Python i wykonać na niej zestaw poleceń:

```
import mysql.connector
```

```
# Nawiązanie połączenia z bazą
```

```
conn = mysql.connector.connect(  
    host="localhost",  
    user="root",  
    password="haslo_do_bazy",  
    database="testowa_baza"  
)
```

```
# Utworzenie kursora
```

```
cursor = conn.cursor()
```

```
# Utworzenie przykładowej tabeli
```

```
cursor.execute("""  
CREATE TABLE IF NOT EXISTS studenci (  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    imie VARCHAR(50),  
    nazwisko VARCHAR(50),  
    email VARCHAR(100)  
)  
""")
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Wstawienie przykładowego rekordu

```
cursor.execute("""
```

```
INSERT INTO studenci (imie, nazwisko, email)
```

```
VALUES (%s, %s, %s)
```

```
""", ("Anna", "Kowalska", "anna.kowalska@example.com"))
```

Zatwierdzenie zmian i zamknięcie połączenia

```
conn.commit()
```

```
cursor.close()
```

```
conn.close()
```

Generowanie danych syntetycznych

Biblioteka **Faker**, dostępna w języku Python umożliwia automatyczne tworzenie tysięcy wiarygodnych wpisów (nazwiska, adresy, numery PESEL, e-maile, daty, firmy itp.) w różnych językach i formatach. Generatory pozwalają tworzyć realistyczne, ale sztuczne dane, które można bezpiecznie wykorzystywać w procesach walidacji, szkolenia modeli, projektowania baz danych czy testów wydajnościowych bez ryzyka ujawnienia danych rzeczywistych. Eliminuje to konieczność ręcznego wprowadzania rekordów i przyspiesza proces prototypowania. Dzięki temu programista lub administrator może w krótkim czasie przygotować kompletny zestaw danych testowych o strukturze odpowiadającej rzeczywistemu systemowi, zachowując kontrolę nad ich objętością, losowością i spójnością logiczną.

Aby połączyć się z bazą MySQL z poziomu Pythona, najczęściej wykorzystuje się bibliotekę **mysql-connector-python**, rozwijaną przez firmę Oracle. Zapewnia ona pełną zgodność z SQL i obsługuje transakcje, kursory oraz przygotowane zapytania.

```
pip install mysql-connector-python
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Instalacja odbywa się za pomocą standardowego menedżera pakietów Pythona:

```
pip install faker
```

Po zainstalowaniu biblioteki należy ją zaimportować i utworzyć obiekt klasy Faker. Generator może działać w różnych językach i formatach regionalnych — określa się to parametrem locale. Dla języka polskiego stosuje się ustawienie pl_PL, co pozwala generować imiona, nazwiska i adresy w konwencji polskiej. Przykład inicjalizacji wraz z przykładem generowania danych w języku polskim:

```
from faker import Faker
```

```
fake = Faker('pl_PL')
```

```
print(fake.name())
```

```
print(fake.address())
```

```
print(fake.email())
```

W zastosowaniach praktycznych biblioteka Faker pozwala generować duże zbiory danych w sposób wydajny i kontrolowany. W przypadku wypełniania tabel bazodanowych najlepszym rozwiązaniem jest tworzenie rekordów hurtowo, zamiast wykonywania wielu pojedynczych operacji wstawiania. Dane można zbudować lokalnie w pamięci programu w postaci listy krotek, a następnie zapisać do bazy danych przy użyciu metody executemany(), która umożliwia wykonanie jednego polecenia SQL dla całej serii wierszy. Przykładowo, zamiast wywoływać INSERT w pętli, można przygotować listę zawierającą tysiąc rekordów wygenerowanych przez Fakera i przekazać ją jednorazowo do kursora bazy danych. Takie podejście znacząco przyspiesza operacje wejścia/wyjścia i redukuje obciążenie serwera.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



```
from faker import Faker
```

```
fake = Faker('pl_PL')
```

```
records = [  
    (fake.first_name(), fake.last_name(), fake.email())  
    for _ in range(1000)  
]
```

```
cursor.executemany(  
    "INSERT INTO studenci (imie, nazwisko, email) VALUES (%s, %s, %s)",  
    records  
)
```

W niektórych przypadkach konieczne jest zapewnienie unikalności generowanych danych, na przykład adresów e-mail lub identyfikatorów użytkowników. Biblioteka Faker udostępnia w tym celu atrybut `unique`, który wymusza, aby każda wygenerowana wartość była niepowtarzalna w danym kontekście. Należy jednak pamiętać, że jeśli zakres możliwych wartości jest ograniczony, użycie `unique` może prowadzić do wyjątku w sytuacji, gdy pula unikalnych danych zostanie wyczerpana.

```
fake = Faker()
```

```
unique_email = fake.unique.email()
```

Dla zapewnienia powtarzalności wyników, co jest szczególnie ważne w testach automatycznych i eksperymentach naukowych można ustawić ziarno generatora pseudolosowego. Funkcja `seed_instance()` pozwala określić stały punkt startowy dla procesu losowania, dzięki czemu kolejne uruchomienia programu będą generować identyczny zestaw danych.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



```
fake = Faker()
```

```
fake.seed_instance(12345)
```

Biblioteka Faker obsługuje również wiele lokalizacji językowych. Ustawienie parametru locale pozwala generować dane zgodne z konwencją danego kraju, na przykład `Faker('pl_PL')` dla języka polskiego lub `Faker('en_US')` dla języka angielskiego. Możliwe jest także jednocześnie wykorzystanie kilku lokalizacji — w takim przypadku Faker losowo wybiera język dla każdego generowanego rekordu, co pozwala tworzyć zbiory o charakterze międzynarodowym.

```
fake_pl = Faker('pl_PL')
```

```
fake_en = Faker('en_US')
```

```
multi = Faker(['pl_PL', 'en_US'])
```

Zastosowanie tych wzorców w praktyce umożliwia tworzenie elastycznych i realistycznych zbiorów danych, które mogą być używane zarówno w testach systemowych, jak i w ćwiczeniach z zakresu baz danych czy analizy danych.

Najważniejsze metody dostępne z poziomu faker:

`name()`, `first_name()`, `last_name()`, `middle_name()`

- `address()`, `street_address()`, `city()`, `postcode()`, `country()`
- `email()`, `safe_email()`, `company_email()`, `phone_number()`
- `text()`, `sentence()`, `paragraph()`, `words()`
- `date()`, `date_time()`, `date_between()`
- `random_int()`, `random_element()`, `boolean()`, `uuid4()`
- `credit_card_number()`, `iban()`, `bank_country()`
- `file_name()`, `file_path()`, `mime_type()`
- `company()`, `job()`
- `bothify()`, `numerify()`, `lexify()` (formaty)
- `unique`, `seed_instance`, `add_provider`, `simple_profile`, `profile`



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Ćwiczenia samodzielne

1. Wybór tematu projektu

Wybierzcie w grupie fragment rzeczywistości, który chcecie odwzorować w bazie danych. Pamiętajcie, aby temat był na tyle bogaty, by dało się w nim wskazać wymaganą liczbę encji:

1 osoba → 10–15 encji

2 osoby → 15–20 encji

3 osoby → 20–25 encji lub więcej

2. Identyfikacja encji i atrybutów

Sporządźcie listę encji.

Do każdej encji przypiszcie atrybuty wraz z typami danych.

Wskażcie klucze główne i zaproponujcie potencjalne klucze obce.

3. Relacje i diagram ERD

Określcie, jakie relacje występują między encjami (1:1, 1:N, N:M).

Rozważcie, które relacje są identyfikujące, a które nieidentyfikujące.

Utwórzcie czytelny diagram ERD przedstawiający całą strukturę.

4. Normalizacja

Przeanalizujcie encje pod kątem normalizacji do 3NF.

Wskażcie, gdzie pojawiała się redundancja i jak ją wyeliminowaliście.

5. Dokumentacja projektu

Przygotujcie opis wybranego obszaru rzeczywistości (1–2 strony).



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Opiszcie encje i ich powiązania w języku naturalnym.

6. Dane przykładowe

Przygotujcie przynajmniej 2–3 przykładowe rekordy dla każdej tabeli, aby sprawdzić, czy model jest logiczny.

7. Hierarchie encji

Rozważcie, czy w Waszym modelu występuje dziedziczenie encji (np. Pracownik → Wykładowca i Administracja). Jeśli tak – pokażcie to na diagramie.

8. Indeksy

Zastanówcie się, które atrybuty będą najczęściej wyszukiwane (np. nazwisko, numer rejestracyjny) i wskażcie kolumny, dla których warto utworzyć indeksy.

9. Bezpieczeństwo i dostęp

Zaproponujcie trzech przykładowych użytkowników systemu z różnymi uprawnieniami (np. administrator, pracownik, klient) i wskażcie, do jakich encji/tabel powinni mieć dostęp.

10. Scenariusz zapytań

Każdy członek zespołu przygotowuje 5 przykładowych pytań biznesowych, na które Wasza baza powinna mieć odpowiedzieć (np. „Którzy studenci zapisali się na dany kurs?”, „Jakie produkty zostały sprzedane w ostatnim miesiącu?” itp.).

Pytania pomocnicze

1. Czym jest diagram ERD?
2. Jakie elementy zawiera diagram ERD?
3. Czym różnią się relacje 1:1, 1:N i N:M?
4. Co oznacza kardynalność relacji?
5. Jak przenieść diagram ERD do tabeli relacyjnej?
6. Czym jest normalizacja?
7. Na czym polega pierwsza postać normalna (1NF)?



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



8. Na czym polega druga postać normalna (2NF)?
9. Na czym polega trzecia postać normalna (3NF)?
10. Podaj przykład redundancji danych.
11. Czym są zależności funkcyjne?
12. Co to jest klucz kandydujący?
13. Jakie są zalety normalizacji?
14. Jakie są potencjalne wady nadmiernej normalizacji?

Podsumowanie

W trakcie drugiego laboratorium studenci rozpoczęli pracę projektową, której celem było stworzenie wstępnego modelu bazy danych odwzorowującego wybrany fragment rzeczywistości. Zrealizowane zadania pozwoliły na praktyczne przełożenie pojęć teoretycznych – encji, atrybutów, kluczy oraz relacji – na schemat struktury tabel, co ułatwia dalsze etapy projektowania. Szczególną uwagę zwrócono na proces identyfikacji encji i relacji oraz na pierwsze próby normalizacji, które pozwoliły zrozumieć, dlaczego eliminowanie redundancji i zapewnienie spójności danych są kluczowe dla jakości modelu. Ćwiczenie unaocniło także, że modelowanie baz danych wymaga nie tylko znajomości języka SQL, ale również umiejętności analizy rzeczywistych procesów i dostrzegania w nich zależności, które można przełożyć na relacje w bazie. Studenci powinni zastanowić się, czy ich model odzwierciedla rzeczywiste potrzeby użytkowników oraz jakie dodatkowe elementy mogłyby go wzbogacić w przyszłości, co stanowi punkt wyjścia do dalszej pracy projektowej w kolejnych laboratoriach.

Literatura

1. How to Draw Entity Relationship Diagrams (ERDs),
<https://www.geeksforgeeks.org/sql/how-to-draw-entity-relationship-diagrams/>,
(data ostatniego dostępu 25.09.25)
2. Introduction to Database Normalization,
<https://www.geeksforgeeks.org/dbms/introduction-of-database-normalization/>,
(data ostatniego dostępu 25.09.25)



Fundusze Europejskie
dla Rozwoju Społecznego

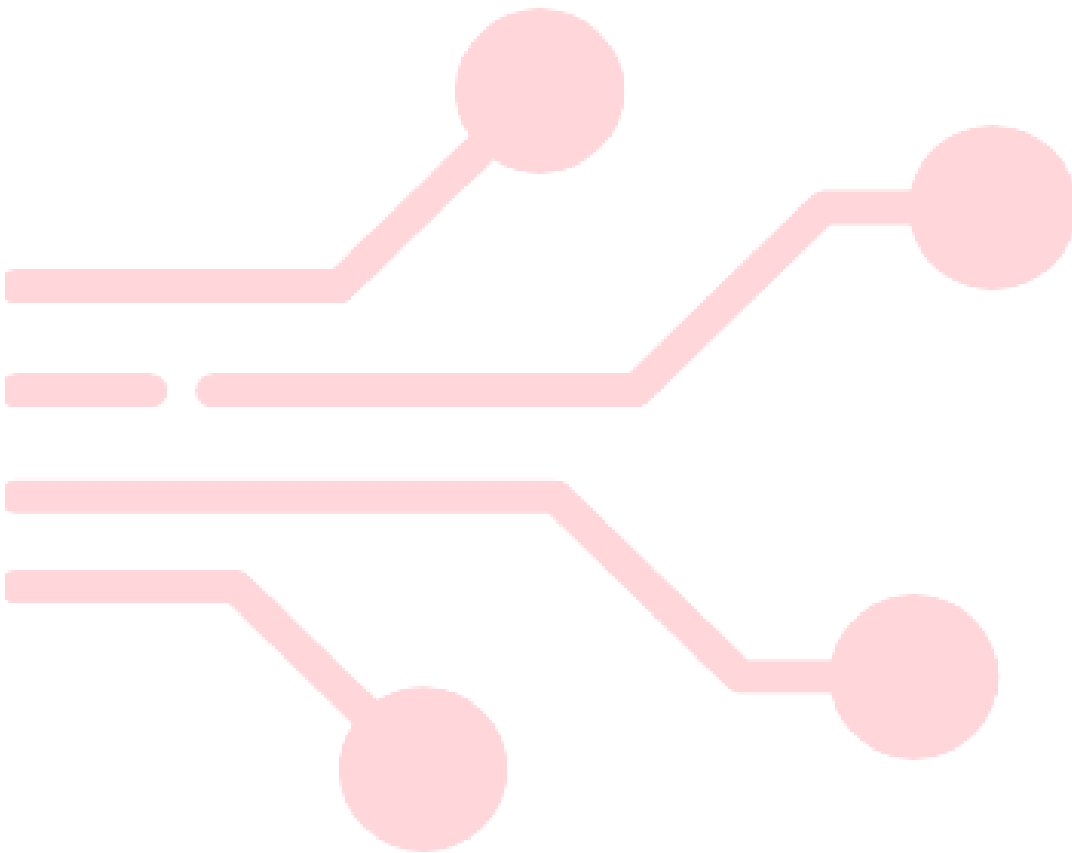


Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



3. Database Normalization – Normal Forms 1nf 2nf 3nf Table Examples, <https://www.freecodecamp.org/news/database-normalization-1nf-2nf-3nf-table-examples/>, (data ostatniego dostępu 25.09.25)



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



a

Materiały do przedmiotu:

Bazy danych

Laboratorium nr 3:

Podstawy języka SQL: polecenia DDL i DML

Autor:

Alicja Żmudzińska



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Spis treści

Wprowadzenie, tematyka zajęć	3
Cel ćwiczenia/laboratorium	3
Instrukcja laboratoryjna/ćwiczeniowa	3
Ćwiczenia samodzielne	30
Pytania pomocnicze	32
Podsumowanie	32
Literatura	32



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU
SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Wprowadzenie, tematyka zajęć

Podczas zajęć uczestnicy poznają podstawy pracy z relacyjnymi bazami danych i językiem SQL, który jest standardowym narzędziem do tworzenia, modyfikowania i pobierania danych. Punktem wyjścia będzie przykładowa baza Sakila, symulująca wypożyczalnię filmów. Na jej przykładzie krok po kroku omówiony zostanie cały proces: od pobrania i instalacji bazy, poprzez jej wypełnienie danymi, aż po tworzenie własnych zapytań i modyfikowanie struktury. W ramach zajęć wyjaśnione zostanie, czym jest baza danych, jakie są jej rodzaje i w jakich sytuacjach jest wykorzystywana. Następnie uczestnicy wprowadzeni zostaną w świat języka SQL, poznając zarówno polecenia służące do manipulowania danymi (DML), jak i te, które pozwalają definiować strukturę bazy (DDL). Kolejnym krokiem będzie praca z bazą Sakila w środowisku XAMPP i phpMyAdmin – tworzenie nowej bazy, import struktury i danych oraz weryfikacja poprawności importu. Istotną część zajęć stanowią ćwiczenia praktyczne, obejmujące wykonywanie zapytań SELECT z różnymi klauzulami, a także wstawianie, modyfikowanie i usuwanie rekordów. Uczestnicy nauczą się tworzyć i modyfikować tabele, dodawać klucze główne i obce oraz dbać o integralność danych. Na zakończenie przewidziane jest samodzielne rozszerzanie bazy, na przykład poprzez dodanie nowych tabel i tworzenie powiązań między nimi. Celem spotkania jest zdobycie umiejętności praktycznego posługiwania się SQL oraz swobodnego zarządzania relacyjną bazą danych w graficznym środowisku phpMyAdmin, co stanowi doskonałe przygotowanie do pracy z rzeczywistymi systemami bazodanowymi.

Cel ćwiczenia/laboratorium

Celem ćwiczeń jest zapoznanie się z podstawowymi zagadnieniami dotyczącymi baz danych oraz języka SQL. Uczestnicy zajęć nauczą się rozróżniać rodzaje baz danych i zrozumieją, dlaczego SQL jest powszechnie stosowanym językiem do zarządzania nimi. W trakcie pracy praktycznej zostanie wykorzystana przykładowa baza danych Sakila, dzięki której możliwe będzie przećwiczenie zarówno tworzenia i importowania bazy w środowisku phpMyAdmin, jak i wykonywania podstawowych operacji na danych. Studenci będą mieli okazję poznać zapytania SQL związane z manipulowaniem danymi oraz definiowaniem struktury tabel, co pozwoli im lepiej zrozumieć relacje między tabelami oraz praktyczne zastosowanie kluczy obcych w relacyjnych bazach danych.

Instrukcja laboratoryjna/ćwiczeniowa

Co to jest baza danych?

Baza danych (ang. Database) to uporządkowany zbiór danych, który umożliwia ich przechowywanie, organizację i szybki dostęp. Dzięki bazom danych można efektywnie



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



zarządzać dużymi ilościami informacji w różnych systemach, np. w sklepach internetowych, bankach, wypożyczalniach filmów czy portalach społecznościowych.

Rodzaje baz danych:

- Relacyjne (RDBMS) – dane przechowywane w tabelach powiązanych ze sobą (np. MySQL, PostgreSQL, Oracle).
- NoSQL – dane przechowywane w formie dokumentów, klucz-wartość, grafów (np. MongoDB, Redis).

Dlaczego SQL?

SQL (Structured Query Language) to standardowy język do zarządzania relacyjnymi bazami danych.

Dzięki SQL możemy:

- Tworzyć i modyfikować struktury baz danych,
- Wstawiać, modyfikować i usuwać dane,
- Pobierać dane zgodnie z określonymi kryteriami,
- Zarządzać uprawnieniami użytkowników.

SQL jest używany praktycznie w każdej aplikacji, która potrzebuje przechowywać dane w uporządkowany sposób.

Jak uczymy się SQL?

Do nauki SQL warto używać gotowych baz danych testowych, które pozwalają ćwiczyć zapytania DML i DDL na realistycznych przykładach. Jedną z popularnych takich baz jest Sakila, która symuluje działanie wypożyczalni filmów.

Baza sakila:

Baza danych Sakila to przykładowa baza danych, która została stworzona przez MySQL, aby pomóc w nauce i demonstracji możliwości systemu zarządzania bazą danych (DBMS) MySQL. Sakila symuluje środowisko wypożyczalni filmów i zawiera dane dotyczące filmów, aktorów, klientów, pracowników, wypożyczeń, sklepów, oraz inne powiązane informacje. Kluczowe elementy bazy danych Sakila:

Struktura Tabel:

- Film (film): Zawiera informacje o filmach, takie jak tytuł, opis, długość, data wydania, itp.
- Aktor (actor): Przechowuje dane aktorów, w tym imię, nazwisko i inne szczegóły.
- Kategoria (category): Zawiera kategorie filmów, takie jak komedia, dramat, akcja itp.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

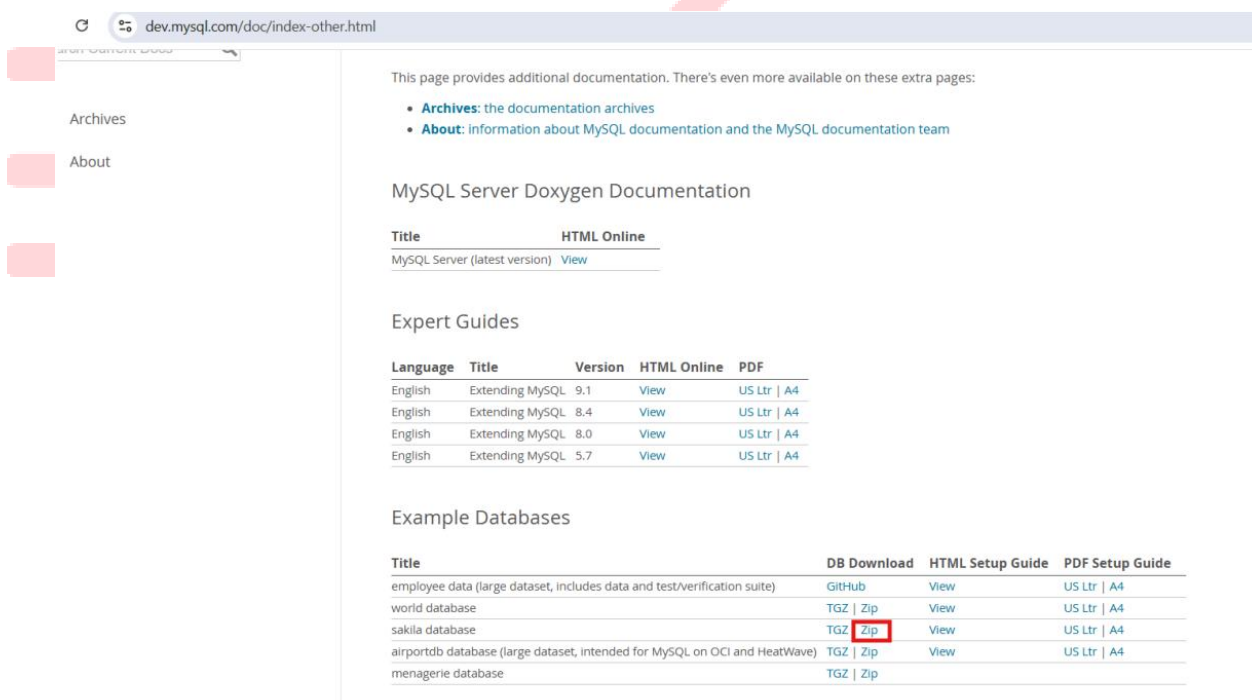
Dofinansowane przez
Unię Europejską



- Klient (customer): Zawiera dane klientów, w tym imię, nazwisko, adres itp.
- Wypożyczenia (rental): Przechowuje informacje o wypożyczeniach filmów przez klientów, w tym daty wypożyczenia i zwrotu.
- Pracownik (staff): Zawiera dane pracowników, którzy zarządzają wypożyczalnią.
- Relacje: Baza danych Sakila zawiera relacje między tabelami, co pozwala na przeprowadzanie skomplikowanych zapytań, takich jak pobieranie informacji o filmach wypożyczanych przez konkretnego klienta lub aktorach występujących w danym filmie.

Zastosowanie: Sakila jest często używana do nauki SQL, testowania funkcji MySQL, a także do prezentacji możliwości systemu w kontekście relacyjnych baz danych. Umożliwia programistom i analitykom przetestowanie i zrozumienie zagadnień związanych z normalizacją danych, tworzeniem zapytań, używaniem kluczy obcych i innych koncepcji baz danych.

Należy ją pobrać ze strony: <https://dev.mysql.com/doc/index-other.html>, tak jak przedstawiono na rysunku 1.



The screenshot shows the MySQL documentation page with a sidebar on the left containing 'Archives' and 'About'. The main content area includes a list of links for 'Archives' and 'About', followed by 'MySQL Server Doxygen Documentation' with a table for 'Title' and 'HTML Online'. Below this is 'Expert Guides' with a table listing various MySQL guides. At the bottom is 'Example Databases' with a table listing databases like 'employee data', 'world database', 'sakila database', 'airportdb database', and 'menagerie database' with links for 'DB Download', 'HTML Setup Guide', and 'PDF Setup Guide'.

Language	Title	Version	HTML Online	PDF
English	Extending MySQL	9.1	View	US Ltr A4
English	Extending MySQL	8.4	View	US Ltr A4
English	Extending MySQL	8.0	View	US Ltr A4
English	Extending MySQL	5.7	View	US Ltr A4

Title	DB Download	HTML Setup Guide	PDF Setup Guide
employee data (large dataset, includes data and test/verification suite)	GitHub	View	US Ltr A4
world database	TGZ Zip	View	US Ltr A4
sakila database	TGZ Zip	View	US Ltr A4
airportdb database (large dataset, intended for MySQL on OCI and HeatWave)	TGZ Zip	View	US Ltr A4
menagerie database	TGZ Zip		

Rysunek 1. Strona pobierania przykładowych baz danych MySQL – dokumentacja i linki do plików



Fundusze Europejskie
dla Rozwoju Społecznego



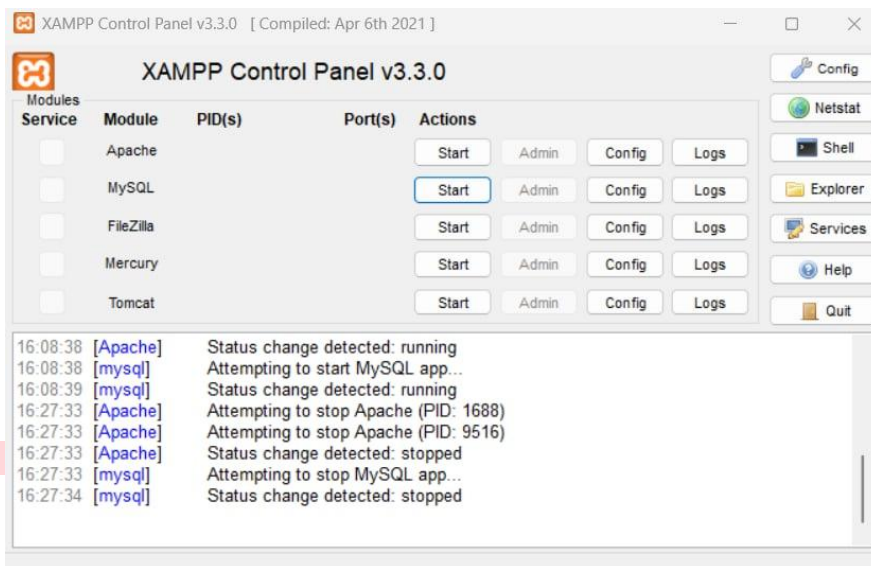
Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



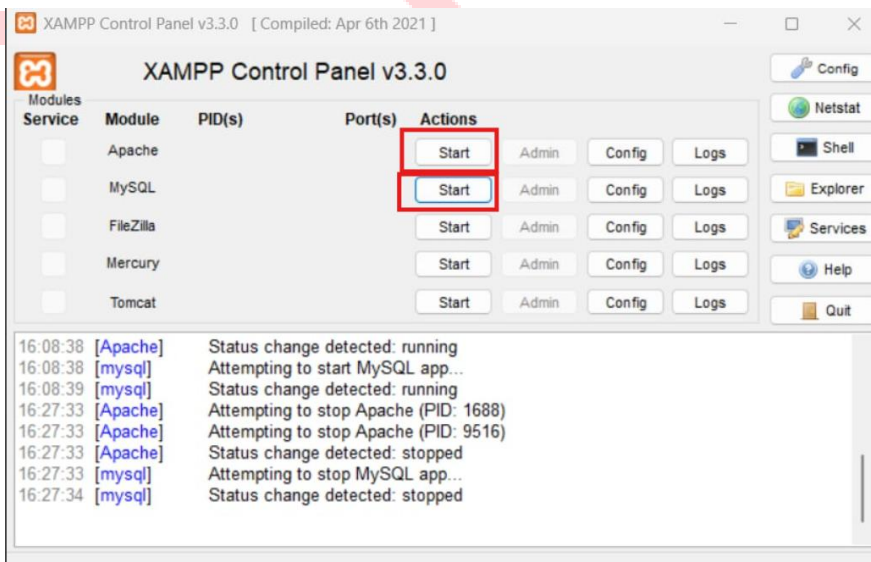
Dodanie bazy sakilka w phpMyAdmin:

W celu dodania bazy danych w phpMyAdmin należy uruchomić XAMPP Control Panel, co zilustrowano na rysunku 2.



Rysunek 2. Panel sterowania XAMPP (XAMPP Control Panel)

Następnie należy wcisnąć przycisk Start w wierszach Apache i MySQL, tak jak przedstawiono na rysunku 3.



Rysunek 3. Uruchamianie usług Apache i MySQL w XAMPP Control Panel



Fundusze Europejskie
dla Rozwoju Społecznego

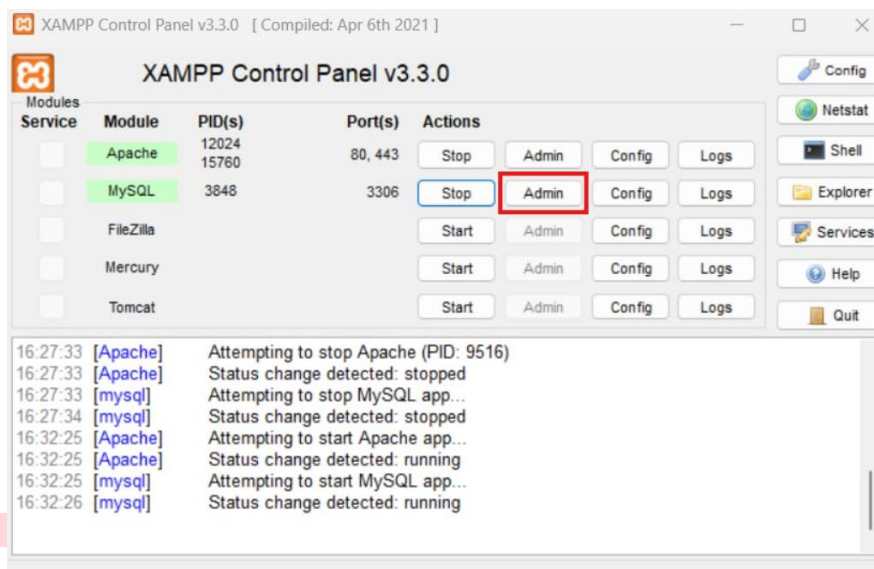


Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską

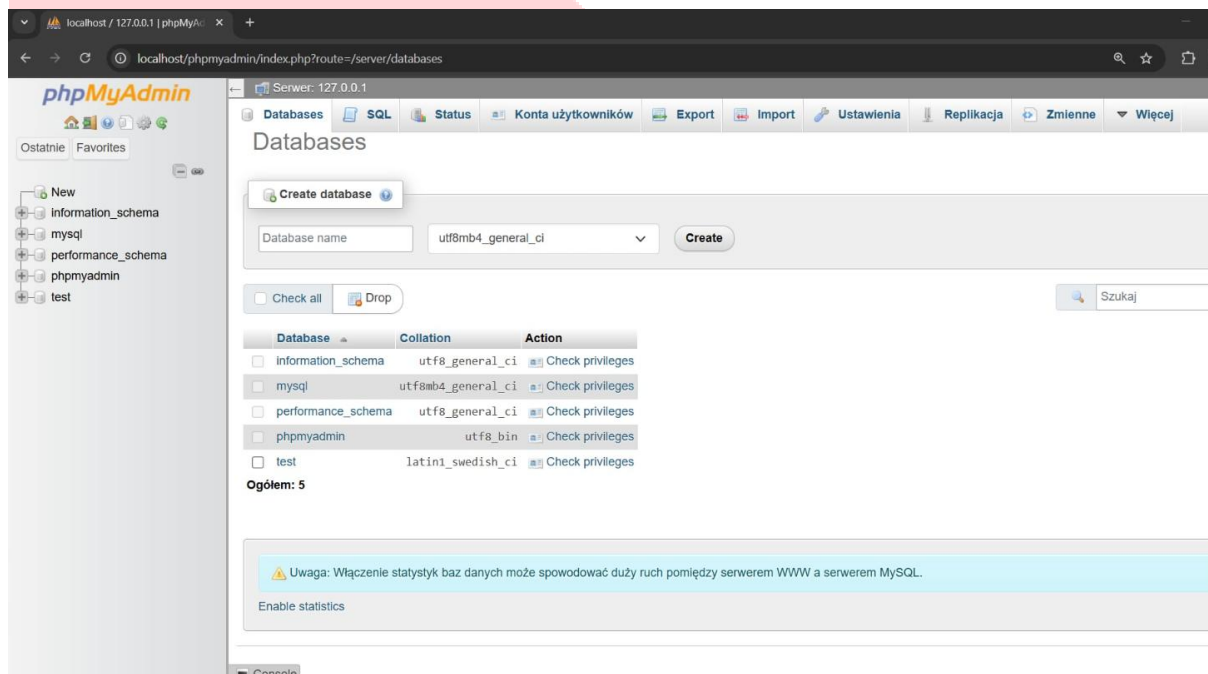


Po uruchomieniu usług przyciski Admin powinny się podświetlić. W celu uruchomienia phpMyAdmin należy kliknąć przycisk Admin w wierszu MySQL, tak jak przedstawiono na rysunku 4.



Rysunek 4. Uruchamianie phpMyAdmin z poziomu XAMPP Control Panel

Po kliknięciu przycisku Admin w panelu XAMPP w przeglądarce otworzy się okno phpMyAdmin, tak jak przedstawiono na rysunku 5.



Fundusze Europejskie
dla Rozwoju Społecznego



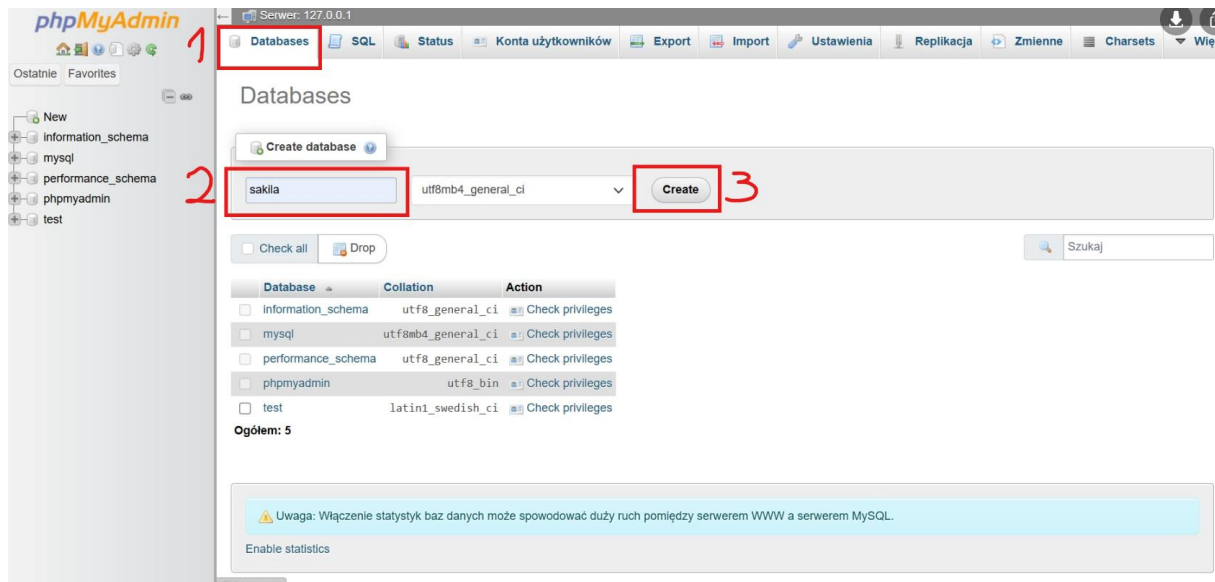
Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



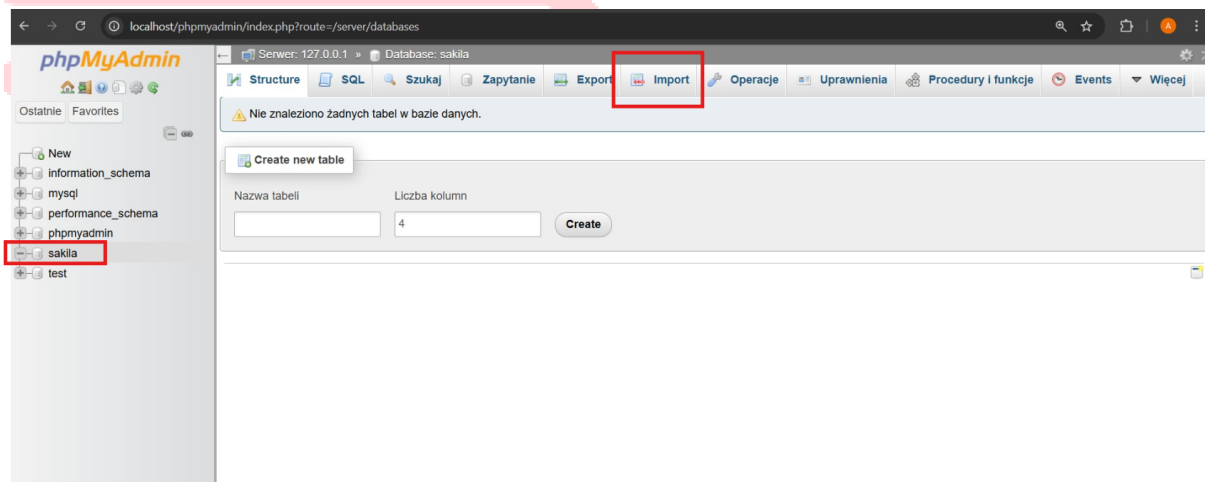
Rysunek 5. Ekran główny phpMyAdmin w przeglądarce

Teraz należy dodać bazę danych sakila. W tym celu w zakładce Databases (1) należy wpisać nazwę bazy (2) i kliknąć przycisk Create (3), tak jak przedstawiono na rysunku 6.



Rysunek 6. Dodanie bazy danych „sakila” w phpMyAdmin

Utworzy się pusta baza danych sakila. Aby dodać do niej dane pobrane wcześniej, należy w menu górnym przejść do zakładki Import, jak pokazano na rysunku 7.



Rysunek 7. Import danych do bazy „sakila” w phpMyAdmin

W zakładce Import kliknij przycisk Wybierz plik, a następnie wskaż wcześniej pobrany plik sakila-schema.sql. staw pozostałe opcje tak, jak przedstawiono na rysunku 8, a następnie naciśnij przycisk Import w celu wczytania struktury bazy danych.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Importowanie do bazy danych "sakila"



File to import:

File may be compressed (gzip, bzip2, zip) or uncompressed.
A compressed file's name must end in `.[format].[compression]`. Example: `.sql.zip`

Browse your computer: (Maksymalny rozmiar: 40MB)

Wybierz plik sakila-schema.sql

Możesz także przeciągnąć i upuścić plik na dowolnej stronie.

Character set of the file:

utf-8

Częściowy import:

☐ Allow the interruption of an import in case the script detects it is close to the PHP timeout limit.
This might be a good way to import large files, however it can break transactions.

Pomiń tę liczbę zapytań (dla SQL), zaczynając od pierwszej:

0

Other options

☐ Enable foreign key checks

Format

SQL

Format-specific options:

Tryb zgodności SQL:

NONE

☐ Do not use AUTO_INCREMENT for zero values

Import

Rysunek 8. Import pliku sakila-schema.sql do bazy danych w phpMyAdmin

Po zakończeniu importu powinno pojawić się okno z komunikatem Import zakończony sukcesem oraz listą wykonanych zapytań, jak pokazano na rysunku 9.



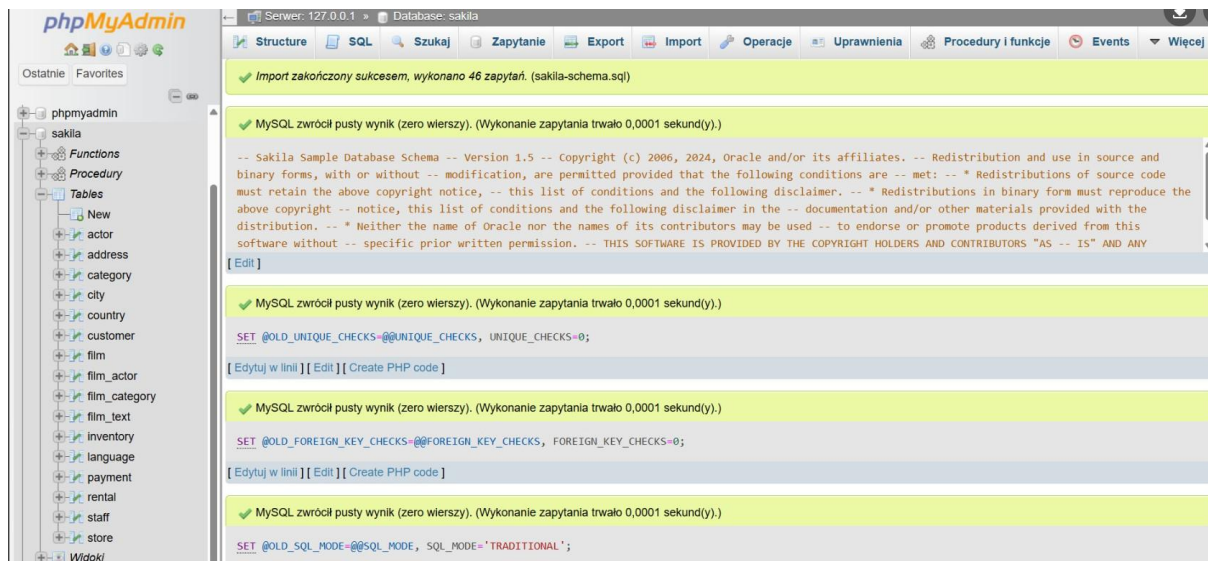
Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską

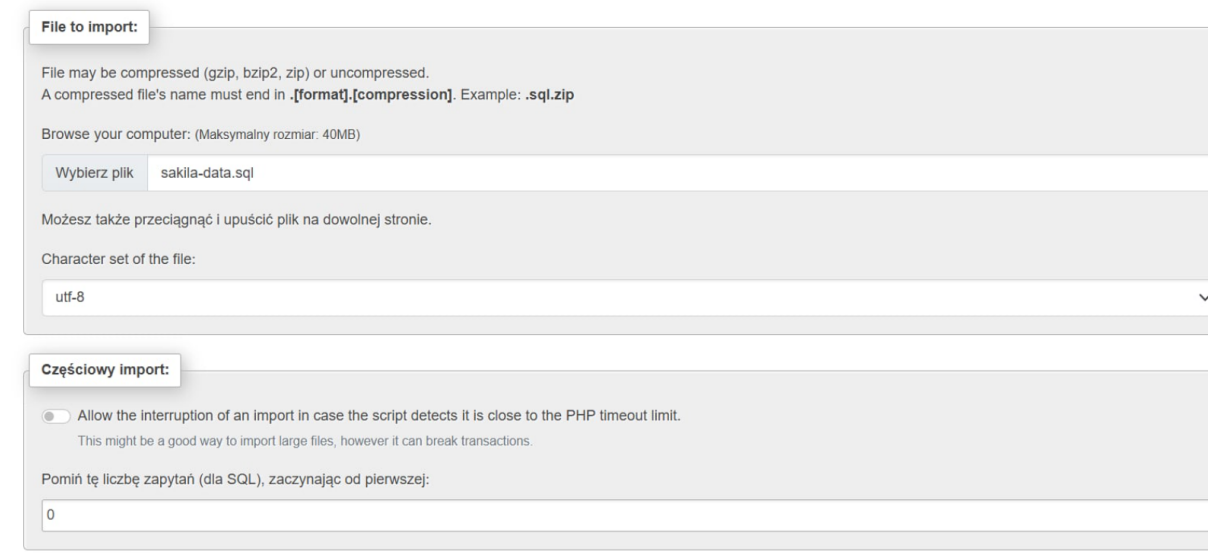




Rysunek 9. Wynik pomyślnego importu struktury bazy danych „sakila” w phpMyAdmin

Po utworzeniu struktury tabel należy wypełnić je danymi. W tym celu wybierz z listy po lewej stronie bazę sakila, przejdź do zakładki Import i tym razem wskaż plik sakila-data.sql, jak przedstawiono na rysunku 10. Pozostałe pola wypełnij tak jak przy imporcie pliku sakila-schema.sql. Następnie kliknij Import.

Importowanie do bazy danych "sakila"



Rysunek 10. Import pliku sakila-data.sql w celu wypełnienia bazy danymi w phpMyAdmin



Fundusze Europejskie
dla Rozwoju Społecznego

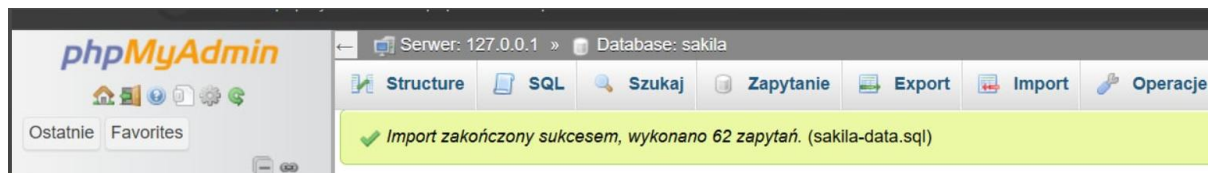


Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską

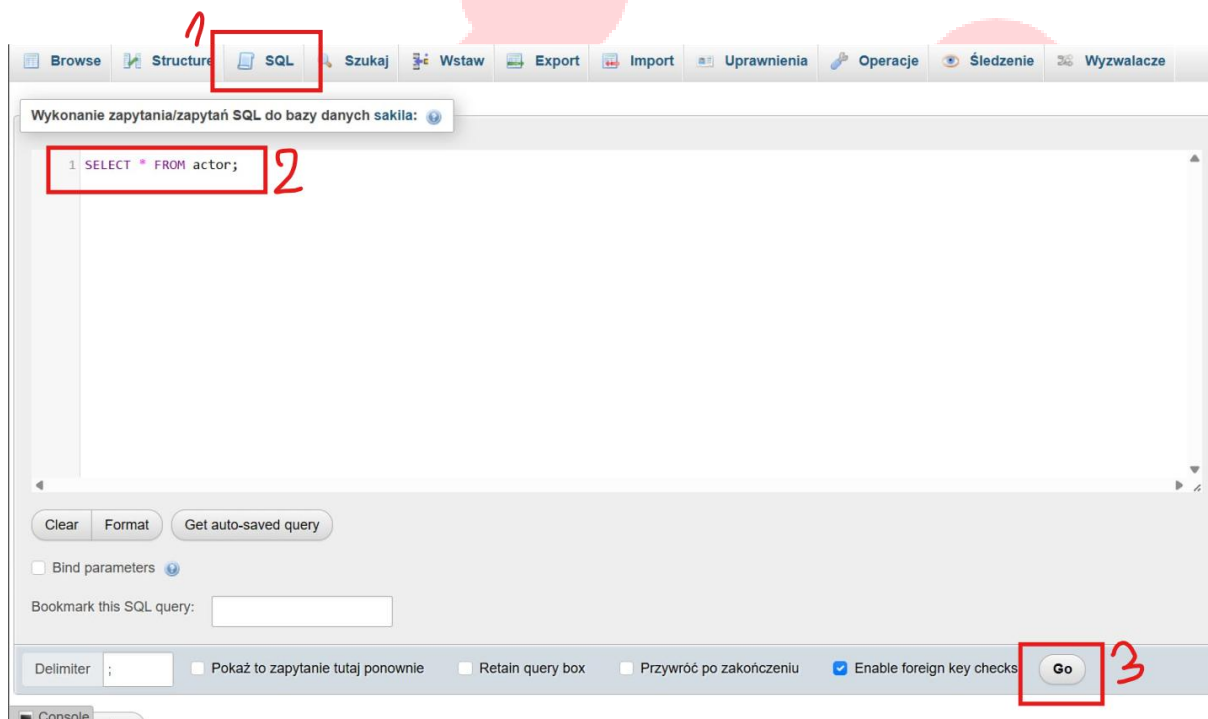


Po zakończeniu importu danych powinien pojawić się komunikat Import zakończony sukcesem, wykonano 62 zapytań (sakila-data.sql), jak pokazano na rysunku 11.



Rysunek 11. Potwierdzenie pomyślnego importu danych do bazy „sakila” w phpMyAdmin

Aby sprawdzić, czy baza danych sakila została poprawnie zaimportowana, przejdź do zakładki SQL (1), wpisz zapytanie `SELECT * FROM aktor;` (2), a następnie kliknij przycisk Go (3), jak przedstawiono na rysunku 12.



Rysunek 12. Wykonanie zapytania SQL w phpMyAdmin w celu sprawdzenia bazy „sakila”

Jeśli baza została poprawnie zaimportowana powinna wyświetlić nam się lista rekordów z tabeli aktor odpowiadająca naszemu zapytaniu:



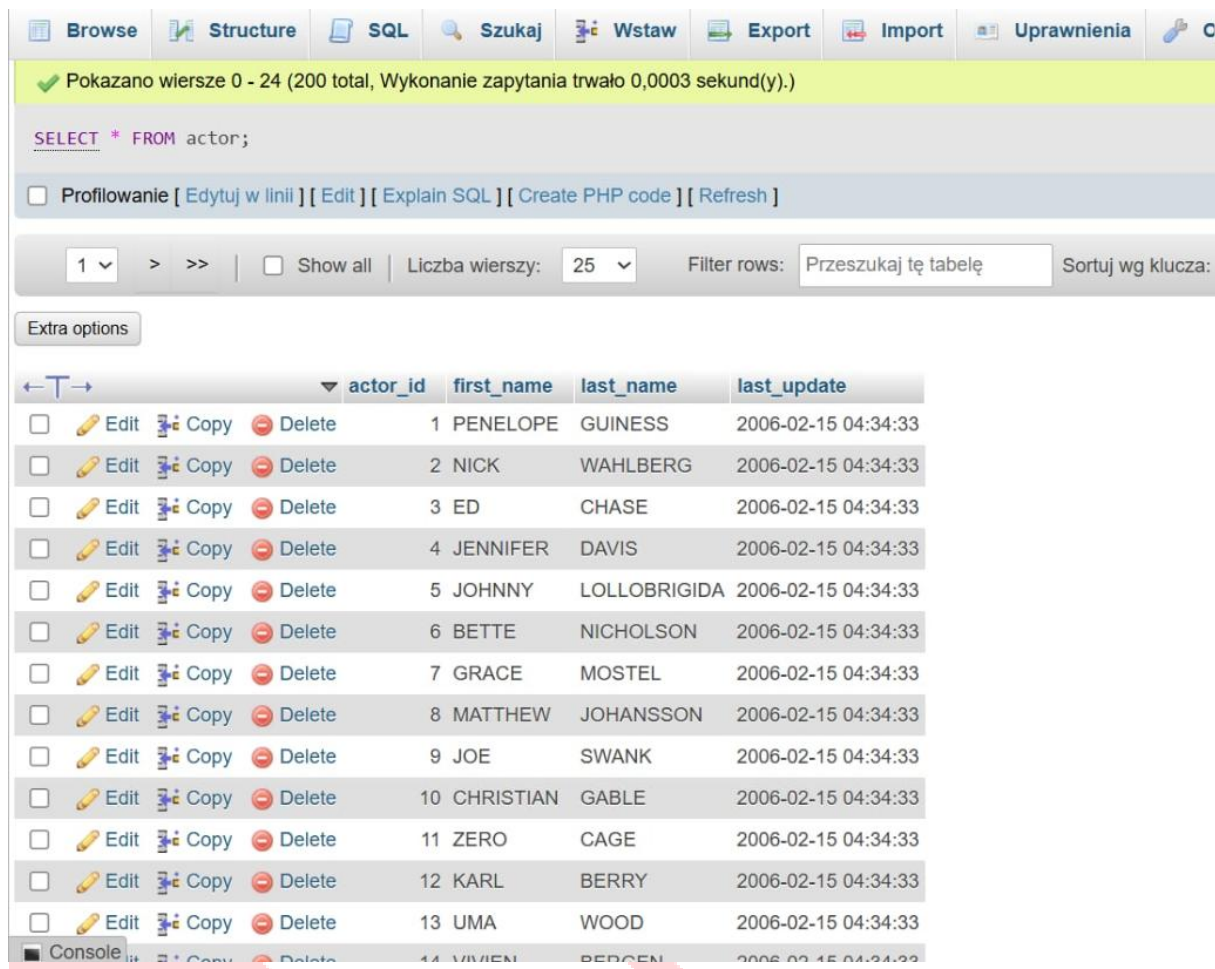
Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską





	actor_id	first_name	last_name	last_update
☐	1	PENELOPE	GUINNESS	2006-02-15 04:34:33
☐	2	NICK	WAHLBERG	2006-02-15 04:34:33
☐	3	ED	CHASE	2006-02-15 04:34:33
☐	4	JENNIFER	DAVIS	2006-02-15 04:34:33
☐	5	JOHNNY	LOLLOBRIGIDA	2006-02-15 04:34:33
☐	6	BETTE	NICHOLSON	2006-02-15 04:34:33
☐	7	GRACE	MOSTEL	2006-02-15 04:34:33
☐	8	MATTHEW	JOHANSSON	2006-02-15 04:34:33
☐	9	JOE	SWANK	2006-02-15 04:34:33
☐	10	CHRISTIAN	GABLE	2006-02-15 04:34:33
☐	11	ZERO	CAGE	2006-02-15 04:34:33
☐	12	KARL	BERRY	2006-02-15 04:34:33
☐	13	UMA	WOOD	2006-02-15 04:34:33
☐	14	VIVIEN	BERGEN	2006-02-15 04:34:33

Rysunek 13. Wynik zapytania SQL SELECT * FROM actor w phpMyAdmin

Wprowadzenie do SQL:

SQL (Structured Query Language) to standardowy język zapytań używany do komunikacji z systemami zarządzania bazami danych (DBMS). Dzięki SQL można:

- Tworzyć bazy danych i tabele,
- Wstawiać, aktualizować i usuwać dane,
- Wykonywać zapytania do pobierania danych,
- Zarządzać uprawnieniami użytkowników.

Dla porządku wyróżnia się dwie podstawowe grupy poleceń:

- **DML (Data Manipulation Language)** – manipulacja danymi,



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



- **DDL (Data Definition Language)** – definiowanie struktury bazy.

DML – język manipulowania danymi:

DML skupia się na manipulowaniu danymi przechowywanymi w tabelach, a nie na samej strukturze bazy. Instrukcje DML są integralną częścią języka zapytań SQL, używanego do zarządzania relacyjnymi bazami danych. Jest to kluczowy mechanizm dla tworzenia aplikacji, ponieważ pozwala na precyzyjne zarządzanie danymi i zapewnienie ich jakości.

Kluczowe operacje DML:

SELECT – pobieranie danych

Klauzula SELECT, musi znaleźć się w każdym zapytaniu. Bez niej, nie wykonamy żadnej kwerendy, bo to właśnie tutaj, określamy w jaki sposób chcemy oglądać wynik zapytania. Pomimo że występuje zawsze jako pierwsza, jej treść jest przetwarzana jako jedna z ostatnich (jeśli nie korzystamy z sortowania zbioru wynikowego ORDER BY, będzie to faktycznie ostatni krok).

- (1) SELECT
- (2) FROM
- (3) WHERE
- (4) GROUP BY
- (5) HAVING
- (6) ORDER BY

SELECT to komenda używana do pobierania danych z bazy danych. Może zwracać jedną lub więcej kolumn oraz wszystkie lub tylko wybrane rekordy.

Składnia:

```
SELECT [kolumna1, kolumna2, ...]  
FROM [nazwa_tabeli]  
WHERE [warunki]  
ORDER BY [kolumna] [ASC|DESC];
```

SELECT: Określa kolumny, które chcesz pobrać z tabeli. kolumna1, kolumna2, ... – lista kolumn, które chcemy wybrać. Użycie * oznacza, że chcemy wyświetlić wszystkie kolumny.

FROM: Określa tabelę (lub tabele), z których chcesz pobrać dane. nazwa_tabeli – tabela, z której pobieramy dane. **WHERE:** Filtruje wiersze według podanego warunku.

ORDER BY: Sortuje wyniki według określonych kolumn, rosnąco (ASC) lub malejąco (DESC).



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Przykład:

```
SELECT * FROM actor;
```

To zapytanie pobiera wszystkie kolumny i wszystkie wiersze z tabeli actor.

```
SELECT first_name, last_name FROM actor;
```

Pobiera tylko kolumny first_name i last_name z tabeli actor.

INSERT – wstawianie danych:

Klauzula INSERT służy do dodawania nowych wierszy (rekordów) do istniejącej tabeli. Pozwala ona na wprowadzenie danych do tabeli, wypełniając je konkretnymi wartościami lub danymi pobranymi z innej tabeli, tym samym rozszerzając zbiory danych w bazie.

Jak działa INSERT:

1. **Wskazanie tabeli i kolumn:** Określasz, do której tabeli ma zostać dodany nowy rekord, oraz listę kolumn, do których chcesz wstawić dane.
2. **Podanie wartości:** Wskazujesz konkretne wartości, które mają być zapisane w tych kolumnach, lub używasz wartości z innej instrukcji SELECT

Przykład: użycia:

```
INSERT INTO actor (first_name, last_name)  
VALUES ('Jan', 'Kowalski');
```

UPDATE – aktualizacja danych

Instrukcja UPDATE w SQL służy do modyfikowania istniejących danych w tabeli. Pozwala ona na zmianę wartości w jednej lub wielu kolumnach dla wybranych rekordów (wierszy) tabeli, na podstawie określonych kryteriów.

Jak działa UPDATE?

1. Identyfikacja tabeli: Musisz wskazać, w której tabeli chcesz dokonać zmiany, używając nazwy tabeli.
2. Określenie zmian: Za pomocą klauzuli SET wskazujesz, które kolumny chcesz zaktualizować i jakie nowe wartości mają przyjąć.
3. Filtracja wierszy: Klauzula WHERE służy do określenia, które konkretne wiersze w tabeli mają zostać objęte zmianą. Jeśli pominiesz klauzulę WHERE, zmiana zostanie zastosowana do wszystkich rekordów w tabeli.

```
UPDATE actor  
SET last_name = 'Nowak'  
WHERE first_name = 'Jan';
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



DELETE – usuwanie danych

Instrukcja DELETE służy do usuwania jednego lub więcej rekordów (wierszy) z tabeli bazy danych, zachowując jednocześnie strukturę tabeli, jej schemat i ograniczenia. Można usunąć wszystkie rekordy, jeśli klauzula WHERE nie zostanie użyta, lub usunąć tylko te rekordy, które spełniają określony warunek.

```
DELETE FROM actor  
WHERE last_name = 'Nowak';
```

DDL – język definiowania danych:

DDL (Data Definition Language) skupia się na **strukturze bazy danych**, a nie na samych danych. Instrukcje DDL pozwalają tworzyć, modyfikować i usuwać obiekty bazy danych, takie jak tabele, widoki, indeksy czy schematy. Są to podstawowe polecenia używane przy projektowaniu i zarządzaniu architekturą bazy. Polecenia DDL wykonują się natychmiast i najczęściej nie można ich cofnąć (automatycznie wywołują COMMIT).

Kluczowe operacje DDL:

CREATE – tworzenie obiektów

Instrukcja CREATE służy do zakładania nowych obiektów w bazie danych, np. tabel, widoków, indeksów. Najczęściej używana jest do tworzenia tabel, które będą przechowywać dane.

```
CREATE TABLE actor (  
    actor_id INT PRIMARY KEY,  
    first_name VARCHAR(50),  
    last_name VARCHAR(50)  
);
```

ALTER – modyfikacja obiektów

Polecenie ALTER służy do zmiany już istniejących obiektów w bazie, np. dodania nowej kolumny do tabeli, zmiany typu danych kolumny czy dodania klucza obcego

```
ALTER TABLE actor  
ADD birth_date DATE;
```

DROP – usuwanie obiektów

Instrukcja DROP całkowicie usuwa obiekt z bazy danych, łącznie z jego strukturą i wszystkimi danymi.

```
DROP TABLE actor;
```

TRUNCATE – czyszczenie tabeli



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Polecenie TRUNCATE usuwa wszystkie dane z tabeli, ale **pozostawia jej strukturę** (czyli kolumny, typy danych i klucze). Jest szybsze od DELETE, bo nie zapisuje każdego usunięcia w logu transakcji.

TRUNCATE TABLE actor;

Przykład 1: Wyświetl strukturę tabeli actor.

DESCRIBE sakila.actor;

Wyjaśnienie:

1. DESCRIBE (albo krócej DESC) – to polecenie w MySQL, które służy do wyświetlania struktury tabeli.
2. sakila – to nazwa przykładowej bazy danych dostarczanej z MySQL. Zawiera ona dane o wypożyczalni filmów.
3. actor – to nazwa tabeli, która w tej bazie przechowuje informacje o aktorach.

Co zwróci polecenie:

Zapytanie SQL zostało wykonane pomyślnie.

```
DESCRIBE sakila.actor;
```

[Edytuj w linii] [Edit] [Create PHP code]

Extra options

Field	Type	Null	Key	Default	Extra
actor_id	smallint(5) unsigned	NO	PRI	NULL	auto_increment
first_name	varchar(45)	NO		NULL	
last_name	varchar(45)	NO	MUL	NULL	
last_update	timestamp	NO		current_timestamp()	on update current_timestamp()

Przykład 2: Wyświetl wszystkie kolumny i wiersze z actor.

SELECT * FROM sakila.actor;

Wyjaśnienie krok po kroku:

1. SELECT – służy do wybierania danych z tabeli.
2. * – oznacza „wszystkie kolumny”. W tym przypadku pobierzesz wszystkie kolumny tabeli actor.
3. FROM sakila.actor – wskazuje, że dane mają pochodzić z tabeli actor w bazie danych sakila.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Co zwróci zapytanie:

✓ Pokazano wiersze 0 - 24 (200 total, Wykonanie zapytania trwało 0,0002 sekund(y).)

```
SELECT * FROM sakila.actor;
```

☐ Profilowanie [[Edytuj w linii](#)] [[Edit](#)] [[Explain SQL](#)] [[Create PHP code](#)] [[Refresh](#)]

1 > >> ☐ Show all Liczba wierszy: 25 Filter rows:

Extra options

	actor_id	first_name	last_name	last_update
☐ Edit Copy Delete	1	PENELOPE	GUINNESS	2006-02-15 04:34:33
☐ Edit Copy Delete	2	NICK	WAHLBERG	2006-02-15 04:34:33
☐ Edit Copy Delete	3	ED	CHASE	2006-02-15 04:34:33
☐ Edit Copy Delete	4	JENNIFER	DAVIS	2006-02-15 04:34:33
☐ Edit Copy Delete	5	JOHNNY	LOLLOBRIGIDA	2006-02-15 04:34:33
☐ Edit Copy Delete	6	BETTE	NICHOLSON	2006-02-15 04:34:33
☐ Edit Copy Delete	7	GRACE	MOSTEL	2006-02-15 04:34:33
☐ Edit Copy Delete	8	MATTHEW	JOHANSSON	2006-02-15 04:34:33
☐ Edit Copy Delete	9	JOE	SWANK	2006-02-15 04:34:33
☐ Edit Copy Delete	10	CHRISTIAN	GABLE	2006-02-15 04:34:33

Przykład 3: Wyświetl tylko first_name i last_name aktorów.

```
SELECT first_name, last_name FROM sakila.actor;
```

Wyjaśnienie krok po kroku:

1. SELECT first_name, last_name – wybiera tylko kolumny first_name i last_name z tabeli.
 - o W przeciwieństwie do SELECT *, tutaj nie pobieramy wszystkich kolumn, tylko konkretne, które nas interesują.
2. FROM sakila.actor – wskazuje tabelę actor w bazie danych sakila.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Co zwróci zapytanie:

✓ Pokazano wiersze 0 - 24 (200 total, Wykonanie zapytania trwało 0,0002 sekund(y).)

```
SELECT first_name, last_name FROM sakila.actor;
```

☐ Profilowanie [[Edytuj w linii](#)] [[Edit](#)] [[Explain SQL](#)] [[Create PHP code](#)] [[Refresh](#)]

1 > >> | ☐ Show all | Liczba wierszy: 25 Filter rows:

Extra options

	first_name	last_name
☐ Edit Copy Delete	PENELOPE	GUINNESS
☐ Edit Copy Delete	NICK	WAHLBERG
☐ Edit Copy Delete	ED	CHASE
☐ Edit Copy Delete	JENNIFER	DAVIS
☐ Edit Copy Delete	JOHNNY	LOLLOBRIGIDA
☐ Edit Copy Delete	BETTE	NICHOLSON
☐ Edit Copy Delete	GRACE	MOSTEL
☐ Edit Copy Delete	MATTHEW	JOHANSSON
☐ Edit Copy Delete	JOE	SWANK
☐ Edit Copy Delete	CHRISTIAN	GABLE

W zapytaniach SQL często skracamy nazwy tabel i porządkujemy nagłówki wyników, nadając im tzw. aliasy. To po prostu krótsze, tymczasowe etykiety używane wyłącznie w trakcie wykonywania zapytania. Dzięki nim zapis staje się czytelniejszy, a przy złączeniach łatwiej uniknąć niejednoznaczności. Alias tabeli nadajemy po jej nazwie, np. FROM sakila.customer c, a potem odwołujemy się do kolumn przez skrót (c.first_name, c.last_name). Podobnie możemy nadać alias kolumnie, aby nadać sensowny nagłówek w raporcie: SUM(amount) AS suma albo f.title AS tytuł. Warto pamiętać, że w UNION nagłówki wyników biorą się z pierwszego SELECT-a, więc to tam opłaca się dodać aliasy (np. SELECT first_name AS imie FROM ... UNION SELECT first_name FROM ...). W złożonych zapytaniach — zwłaszcza z wieloma JOIN-ami lub podzapytaniem EXISTS — aliasy są praktycznie niezbędne: pozwalają jednoznacznie wskazać, z której tabeli pochodzi dana kolumna i znacząco skracają kod. Same aliasy nie wpływają na wydajność, ale poprawiają czytelność i ułatwiają utrzymanie zapytań.

Przykład 4: Wyświetl imię i nazwisko aktorów z aliasami imię, nazwisko.

```
SELECT first_name AS imię, last_name AS nazwisko
FROM sakila.aktor;
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Wyjaśnienie krok po kroku:

1. `SELECT first_name AS imie` – pobiera kolumnę `first_name`, ale w wyniku nadaje jej nową nazwę: `imie`.
2. `last_name AS nazwisko` – pobiera kolumnę `last_name` i zmienia jej nagłówek w wynikach na `nazwisko`.
3. `FROM sakila.actor` – określa tabelę `actor` w bazie `sakila`.

Co zwróci zapytanie:

✓ Pokazano wiersze 0 - 24 (200 total, Wykonanie zapytania trwało 0,0002 sekund(y).)

```
SELECT first_name AS imie, last_name AS nazwisko FROM sakila.actor;
```

☐ Profilowanie [[Edytuj w linii](#)] [[Edit](#)] [[Explain SQL](#)] [[Create PHP code](#)] [[Refresh](#)]

1 > >> | ☐ Show all | Liczba wierszy: 25 | Filter rows:

Extra options

	imie	nazwisko
☐ Edit Copy Delete	PENELOPE	GUINNESS
☐ Edit Copy Delete	NICK	WAHLBERG
☐ Edit Copy Delete	ED	CHASE
☐ Edit Copy Delete	JENNIFER	DAVIS
☐ Edit Copy Delete	JOHNNY	LOLLOBRIGIDA
☐ Edit Copy Delete	BETTE	NICHOLSON
☐ Edit Copy Delete	GRACE	MOSTEL
☐ Edit Copy Delete	MATTHEW	JOHANSSON
☐ Edit Copy Delete	JOE	SWANK
☐ Edit Copy Delete	CHRISTIAN	GABLE

Przykład 5: Wyświetl pełne imię i nazwisko w jednej kolumnie `pelne_nazwisko`.

```
SELECT CONCAT(first_name, ' ', last_name) AS pelne_nazwisko
FROM sakila.actor;
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

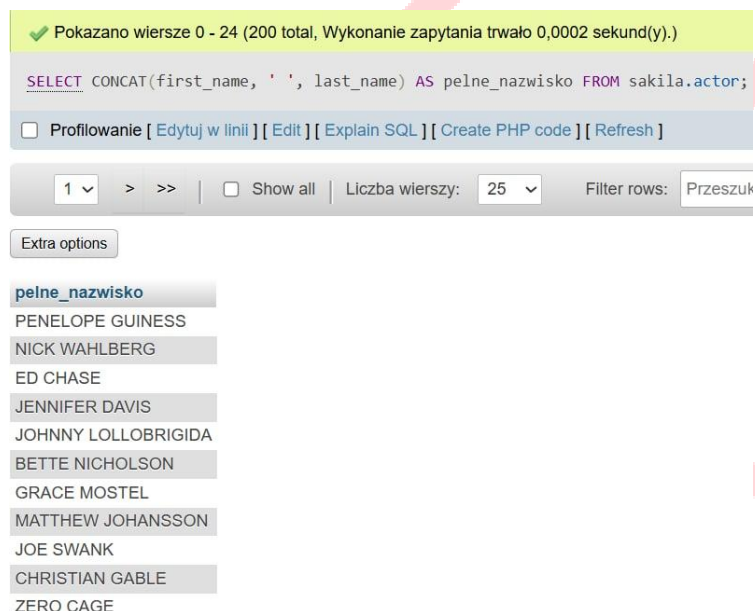
Dofinansowane przez
Unię Europejską



Wyjaśnienie krok po kroku:

1. `CONCAT(first_name, ' ', last_name)`
 - Funkcja `CONCAT` w SQL łączy kilka wartości w jeden ciąg znaków.
 - Tutaj łączymy imię (`first_name`), spację ' ' i nazwisko (`last_name`) w jedno pole tekstowe.
2. `AS pelne_nazwisko`
 - Nadaje wynikowej kolumnie nazwę `pelne_nazwisko`, dzięki czemu nagłówek tabeli jest czytelny.
3. `FROM sakila.actor`
 - Określa tabelę, z której pobieramy dane.

Co zwróci zapytanie:



✓ Pokazano wiersze 0 - 24 (200 total, Wykonanie zapytania trwało 0,0002 sekund(y).)

```
SELECT CONCAT(first_name, ' ', last_name) AS pelne_nazwisko FROM sakila.actor;
```

☐ Profilowanie [Edytuj w linii] [Edit] [Explain SQL] [Create PHP code] [Refresh]

1 > >> | ☐ Show all | Liczba wierszy: 25 | Filter rows: Przeszuk

Extra options

pelne_nazwisko
PENELOPE GUINNESS
NICK WAHLBERG
ED CHASE
JENNIFER DAVIS
JOHNNY LOLLOBRIGIDA
BETTE NICHOLSON
GRACE MOSTEL
MATTHEW JOHANSSON
JOE SWANK
CHRISTIAN GABLE
ZERO CAGE

Przykład 6: Wyświetl tytuły i długości filmów dłuższych niż 120 min.

```
SELECT title, length
FROM sakila.film
WHERE length > 120;
```

Wyjaśnienie krok po kroku:

1. `SELECT title, length`
 - Wybiera kolumny `title` (tytuł filmu) i `length` (długość filmu w minutach).



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



2. FROM sakila.film

- Określa tabelę film w bazie sakila.

3. WHERE length > 120

- Filtruje dane – w wyniku pojawiają się tylko filmy dłuższe niż 120 minut.
- > oznacza „większe niż”. Można też używać <, >=, <=, <> itd.

Co zwróci zapytanie:

✓ Pokazano wiersze 0 - 24 (457 total, Wykonanie zapytania trwało 0,0010 sekund(y).)

```
SELECT title, length FROM sakila.film WHERE length > 120;
```

☐ Profilowanie [Edytuj w linii] [Edit] [Explain SQL] [Create PHP code] [Refresh]

1 > >> | ☐ Show all | Liczba wierszy: 25 | Filter rows:

Extra options

	title	length
☐ Edit Copy Delete	AFRICAN EGG	130
☐ Edit Copy Delete	AGENT TRUMAN	169
☐ Edit Copy Delete	ALAMO VIDEOTAPE	126
☐ Edit Copy Delete	ALASKA PHANTOM	136
☐ Edit Copy Delete	ALI FOREVER	150
☐ Edit Copy Delete	ALLEY EVOLUTION	180
☐ Edit Copy Delete	AMERICAN CIRCUS	129
☐ Edit Copy Delete	ANALYZE HOOSIERS	181
☐ Edit Copy Delete	ANONYMOUS HUMAN	179
☐ Edit Copy Delete	ANTITRUST TOMATOES	168

Przykład 7: Wyświetl first_name, last_name, actor_id i posortuj rosnąco po actor_id, a w ramach tego malejąco po last_name.

```
SELECT first_name, last_name, actor_id
FROM sakila.actor
ORDER BY actor_id ASC, last_name DESC;
```

Wyjaśnienie krok po kroku:



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



1. `SELECT first_name, last_name, actor_id`
 - Wybiera kolumny: imię, nazwisko i identyfikator aktora.
2. `FROM sakila.actor`
 - Określa tabelę actor w bazie sakila.
3. `ORDER BY actor_id ASC, last_name DESC`
 - Sortuje wyniki według kilku kryteriów:
 - `actor_id ASC` – rosnąco po identyfikatorze aktora.
 - `last_name DESC` – malejąco po nazwisku (dla rekordów o tym samym `actor_id`).
 - ASC oznacza ascending (rosnąco), a DESC descending (malejąco).
 - Można sortować po jednej lub wielu kolumnach w różnym porządku.

Co zwróci zapytanie:

✓ Pokazano wiersze 0 - 24 (200 total, Wykonanie zapytania trwało 0,0003 sekund(y).) [actor_id: 1... - 25...] [last_name: GUINNESS... - BLOOM...]

```
SELECT first_name, last_name, actor_id FROM sakila.actor ORDER BY actor_id ASC, last_name DESC;
```

☐ Profilowanie [[Edytuj w linii](#)] [[Edit](#)] [[Explain SQL](#)] [[Create PHP code](#)] [[Refresh](#)]

1 > >> ☐ Show all | Liczba wierszy: 25 | Filter rows: Przeszukaj tę tabelę | Sortuj wg klucza: Żaden

Extra options

	first_name	last_name	actor_id
☐ Edit Copy Delete	PENELOPE	GUINNESS	1
☐ Edit Copy Delete	NICK	WAHLBERG	2
☐ Edit Copy Delete	ED	CHASE	3
☐ Edit Copy Delete	JENNIFER	DAVIS	4
☐ Edit Copy Delete	JOHNNY	LOLLOBRIGIDA	5
☐ Edit Copy Delete	BETTE	NICHOLSON	6
☐ Edit Copy Delete	GRACE	MOSTEL	7
☐ Edit Copy Delete	MATTHEW	JOHANSSON	8
☐ Edit Copy Delete	JOE	SWANK	9
☐ Edit Copy Delete	CHRISTIAN	GABLE	10

Przykład 8: Wyświetl kraje rosnąco alfabetycznie.

```
SELECT country
FROM sakila.country
ORDER BY country ASC;
```

Wyjaśnienie krok po kroku:



Fundusze Europejskie
dla Rozwoju Społecznego



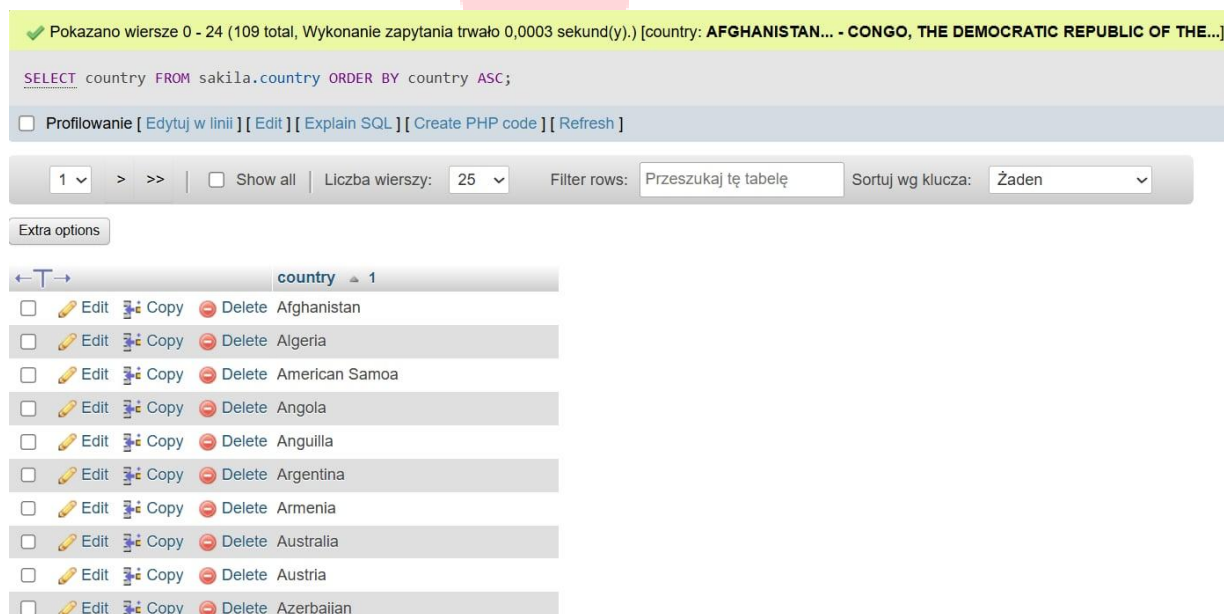
Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



1. SELECT country
 - Pobiera tylko kolumnę country z tabeli country.
2. FROM sakila.country
 - Określa tabelę country w bazie sakila.
3. ORDER BY country ASC
 - Sortuje wyniki rosnąco (ASC), czyli alfabetycznie od A do Z.
 - ASC jest domyślny, więc można go pominąć: ORDER BY country; da taki sam efekt.

Co zwróci zapytanie:



✓ Pokazano wiersze 0 - 24 (109 total, Wykonanie zapytania trwało 0,0003 sekund(y).) [country: AFGHANISTAN... - CONGO, THE DEMOCRATIC REPUBLIC OF THE...]

SELECT country FROM sakila.country ORDER BY country ASC;

☐ Profilowanie [Edytuj w linii] [Edit] [Explain SQL] [Create PHP code] [Refresh]

1 > >> | ☐ Show all | Liczba wierszy: 25 | Filter rows: Przeszukaj tę tabelę | Sortuj wg klucza: Żaden

Extra options

country 1

☐	Edit	Copy	Delete	Afghanistan
☐	Edit	Copy	Delete	Algeria
☐	Edit	Copy	Delete	American Samoa
☐	Edit	Copy	Delete	Angola
☐	Edit	Copy	Delete	Anguilla
☐	Edit	Copy	Delete	Argentina
☐	Edit	Copy	Delete	Armenia
☐	Edit	Copy	Delete	Australia
☐	Edit	Copy	Delete	Austria
☐	Edit	Copy	Delete	Azerbaijan

Przykład 9: Wyświetl last_name, last_update, actor_id i actor_id + 100 jako id_po_zmianie.

```
SELECT last_name AS nazwisko,
       last_update AS data_aktualizacji,
       actor_id AS id,
       actor_id + 100 AS id_po_zmianie
FROM sakila.aktor;
```

Wyjaśnienie krok po kroku:

1. last_name AS nazwisko
 - Pobiera kolumnę last_name i nadaje jej nagłówek nazwisko.
2. last_update AS data_aktualizacji



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



- Pobiera kolumnę last_update i nadaje jej nagłówek data_aktualizacji.
- 3. actor_id AS id
 - Pobiera kolumnę actor_id i nadaje jej nagłówek id.
- 4. actor_id + 100 AS id_po_zmianie
 - Wykonuje prostą operację matematyczną – dodaje 100 do wartości actor_id i zapisuje wynik w kolumnie o nagłówku id_po_zmianie.
- 5. FROM sakila.actor
 - Określa tabelę actor w bazie sakila.

Co zwróci zapytanie:

✓ Pokazano wiersze 0 - 24 (200 total, Wykonanie zapytania trwało 0,0002 sekund(y).)

```
SELECT last_name AS nazwisko, last_update AS data_aktualizacji, actor_id AS id, actor_id + 100 AS id_po_zmianie FROM sakila.actor;
```

☐ Profilowanie [Edytuj w linii] [Edit] [Explain SQL] [Create PHP code] [Refresh]

1 > >> ☐ Show all Liczba wierszy: 25 Filter rows: Przeszukaj tę tabelę Sortuj wg klucza: Żaden

Extra options

		nazwisko	data_aktualizacji	id	id_po_zmianie
☐	Edit Copy Delete	GUINNESS	2006-02-15 04:34:33	1	101
☐	Edit Copy Delete	WAHLBERG	2006-02-15 04:34:33	2	102
☐	Edit Copy Delete	CHASE	2006-02-15 04:34:33	3	103
☐	Edit Copy Delete	DAVIS	2006-02-15 04:34:33	4	104
☐	Edit Copy Delete	LOLLOBRIGIDA	2006-02-15 04:34:33	5	105
☐	Edit Copy Delete	NICHOLSON	2006-02-15 04:34:33	6	106
☐	Edit Copy Delete	MOSTEL	2006-02-15 04:34:33	7	107
☐	Edit Copy Delete	JOHANSSON	2006-02-15 04:34:33	8	108
☐	Edit Copy Delete	SWANK	2006-02-15 04:34:33	9	109
☐	Edit Copy Delete	GABLE	2006-02-15 04:34:33	10	110

Przykład 10: Pokaż 10 pierwszych aktorów wg actor_id.

```
SELECT *
FROM sakila.actor
ORDER BY actor_id
LIMIT 10;
```

Wyjaśnienie krok po kroku:

1. SELECT *
 - Pobiera wszystkie kolumny z tabeli actor.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



2. FROM sakila.actor
 - Określa tabelę actor w bazie sakila.
3. ORDER BY actor_id
 - Sortuje wyniki rosnąco według kolumny actor_id.
 - ASC jest domyślne, więc nie trzeba go pisać.
4. LIMIT 10
 - Ogranicza wynik do pierwszych 10 rekordów po posortowaniu.
 - Przydatne przy dużych tabelach lub do podglądu danych.

Co zwróci zapytanie:

✓ Pokazano wiersze 0 - 9 (10 total, Wykonanie zapytania trwało 0,0003 sekund(y).) [actor_id: 1... - 10...]

```
SELECT * FROM sakila.actor ORDER BY actor_id LIMIT 10;
```

☐ Profilowanie [[Edytuj w linii](#)] [[Edit](#)] [[Explain SQL](#)] [[Create PHP code](#)] [[Refresh](#)]

Extra options

	actor_id	first_name	last_name	last_update
☐ Edit Copy Delete	1	PENELOPE	GUINNESS	2006-02-15 04:34:33
☐ Edit Copy Delete	2	NICK	WAHLBERG	2006-02-15 04:34:33
☐ Edit Copy Delete	3	ED	CHASE	2006-02-15 04:34:33
☐ Edit Copy Delete	4	JENNIFER	DAVIS	2006-02-15 04:34:33
☐ Edit Copy Delete	5	JOHNNY	LOLLOBRIGIDA	2006-02-15 04:34:33
☐ Edit Copy Delete	6	BETTE	NICHOLSON	2006-02-15 04:34:33
☐ Edit Copy Delete	7	GRACE	MOSTEL	2006-02-15 04:34:33
☐ Edit Copy Delete	8	MATTHEW	JOHANSSON	2006-02-15 04:34:33
☐ Edit Copy Delete	9	JOE	SWANK	2006-02-15 04:34:33
☐ Edit Copy Delete	10	CHRISTIAN	GABLE	2006-02-15 04:34:33

Przykład 11: Pomiń 20 pierwszych aktorów i pokaż kolejnych 10.

```
SELECT *
FROM sakila.actor
ORDER BY actor_id
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



LIMIT 10 OFFSET 20;

Wyjaśnienie krok po kroku:

1. SELECT *
 - Pobiera wszystkie kolumny z tabeli actor.
2. FROM sakila.actor
 - Określa tabelę actor w bazie sakila.
3. ORDER BY actor_id
 - Sortuje wyniki rosnąco według kolumny actor_id.
4. LIMIT 10 OFFSET 20
 - LIMIT 10 – pobiera maksymalnie 10 rekordów.
 - OFFSET 20 – pomija pierwsze 20 rekordów w wynikach.
 - Innymi słowy, pobieramy rekordy od 21 do 30 w kolejności rosnącej według actor_id.

Co zwróci zapytanie:

✓ Pokazano wiersze 20 - 29 (10 total, Wykonanie zapytania trwało 0,0003 sekund(y).) [actor_id: 21... - 30...]

```
SELECT * FROM sakila.actor ORDER BY actor_id LIMIT 10 OFFSET 20;
```

☐ Profilowanie [[Edytuj w linii](#)] [[Edit](#)] [[Explain SQL](#)] [[Create PHP code](#)] [[Refresh](#)]

Extra options

	actor_id	first_name	last_name	last_update
☐ Edit Copy Delete	21	KIRSTEN	PALTROW	2006-02-15 04:34:33
☐ Edit Copy Delete	22	ELVIS	MARX	2006-02-15 04:34:33
☐ Edit Copy Delete	23	SANDRA	KILMER	2006-02-15 04:34:33
☐ Edit Copy Delete	24	CAMERON	STREEP	2006-02-15 04:34:33
☐ Edit Copy Delete	25	KEVIN	BLOOM	2006-02-15 04:34:33
☐ Edit Copy Delete	26	RIP	CRAWFORD	2006-02-15 04:34:33
☐ Edit Copy Delete	27	JULIA	MCQUEEN	2006-02-15 04:34:33
☐ Edit Copy Delete	28	WOODY	HOFFMAN	2006-02-15 04:34:33
☐ Edit Copy Delete	29	ALEC	WAYNE	2006-02-15 04:34:33
☐ Edit Copy Delete	30	SANDRA	PECK	2006-02-15 04:34:33



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Przykład 12: Dodaj aktora Jan Kowalski.

```
INSERT INTO sakila.actor (first_name, last_name)
VALUES ('Jan', 'Kowalski');
```

Wyjaśnienie krok po kroku:

1. INSERT INTO sakila.actor (first_name, last_name)

- Określa tabelę actor, do której chcemy dodać rekord.
- W nawiasach podajemy **kolumny**, które chcemy wypełnić (tutaj first_name i last_name).

2. VALUES ('Jan', 'Kowalski')

- Podaje wartości dla wybranych kolumn:
 - first_name = 'Jan'
 - last_name = 'Kowalski'

Co zrobi zapytanie:

- Dodaje nowy rekord do tabeli actor.
- Kolumna actor_id zostanie automatycznie wypełniona (bo jest auto_increment).
- Kolumna last_update zostanie automatycznie ustawiona na aktualną datę i czas.

Przykład 13: Zmień nazwisko Jan Kowalski na Kowalski-Nowy.

```
UPDATE sakila.actor
SET last_name = 'Kowalski-Nowy'
WHERE first_name = 'Jan' AND last_name = 'Kowalski';
```

Wyjaśnienie krok po kroku:

1. UPDATE sakila.actor

- Określa tabelę actor, w której chcemy zmienić dane.

2. SET last_name = 'Kowalski-Nowy'

- Określa nową wartość dla kolumny last_name.
- W tym przypadku zmieniamy nazwisko z Kowalski na Kowalski-Nowy.

3. WHERE first_name = 'Jan' AND last_name = 'Kowalski'



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



- Filtruje rekordy, które mają być zmienione.
- Aktualizacja dotyczy tylko aktora o imieniu Jan i nazwisku Kowalski.
- Bardzo ważne: bez WHERE wszystkie rekordy w tabeli zostałyby zmienione!

Co robi zapytanie:

Znajdzie wszystkie rekordy spełniające warunek (`first_name = 'Jan'` i `last_name = 'Kowalski'`) i zmieni ich nazwisko na Kowalski-Nowy.

Przykład 14: Usuń aktora Jan Kowalski-Nowy.

```
DELETE FROM sakila.actor  
WHERE first_name = 'Jan' AND last_name = 'Kowalski-Nowy';
```

Wyjaśnienie krok po kroku:

1. `DELETE FROM sakila.actor`
 - Określa tabelę `actor`, z której chcemy usunąć rekordy.
2. `WHERE first_name = 'Jan' AND last_name = 'Kowalski-Nowy'`
 - Filtruje rekordy do usunięcia.
 - Usunięty zostanie tylko aktor o imieniu Jan i nazwisku Kowalski-Nowy.
 - Bardzo ważne: bez WHERE wszystkie rekordy w tabeli zostałyby usunięte!

Co robi zapytanie:

- Znajdzie rekordy spełniające warunek i je usunie.

Przykład 15: Utwórz w bazie danych `sakila` nową tabelę o nazwie `favorite_actor`, która będzie przechowywać informacje o ulubionych aktorach użytkowników. Tabela powinna zawierać:

- kolumnę `favorite_id` typu `INT` z automatycznym zwiększaniem wartości (`AUTO_INCREMENT`), pełniącą rolę klucza głównego;
- kolumnę `actor_id` typu `INT`, która nie może być pusta i powinna stanowić klucz obcy wskazujący na kolumnę `actor_id` w tabeli `actor`;
- kolumnę `comment` typu `VARCHAR(100)`, w której można umieścić krótki komentarz.

```
CREATE TABLE sakila.favorite_actor (
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



```
favorite_id INT AUTO_INCREMENT PRIMARY KEY,  
actor_id INT NOT NULL,  
comment VARCHAR(100),  
FOREIGN KEY (actor_id) REFERENCES sakila.actor(actor_id)  
);
```

Wyjaśnienie krok po kroku:

1. CREATE TABLE sakila.favorite_actor
 - Tworzy nową tabelę favorite_actor w bazie sakila.
2. favorite_id INT AUTO_INCREMENT PRIMARY KEY
 - favorite_id – kolumna typu liczbowego (INT).
 - AUTO_INCREMENT – numer rośnie automatycznie przy dodawaniu kolejnych rekordów.
 - PRIMARY KEY – oznacza, że jest to klucz główny tabeli (unikalny identyfikator rekordu).
3. actor_id INT NOT NULL
 - Kolumna do przechowywania identyfikatora aktora.
 - NOT NULL – nie może być pusta.
4. comment VARCHAR(100)
 - Kolumna tekstowa, w której można dodać komentarz (maks. 100 znaków).
5. FOREIGN KEY (actor_id) REFERENCES sakila.actor(actor_id)
 - Tworzy klucz obcy, który wskazuje na kolumnę actor_id w tabeli actor.
 - Zapewnia integralność danych – nie można dodać do favorite_actor aktora, którego nie ma w tabeli actor.

Co robi zapytanie:

- Tworzy tabelę do przechowywania ulubionych aktorów z komentarzami.
- Każdy rekord ma swój unikalny identyfikator (favorite_id) i odnosi się do istniejącego aktora (actor_id).

Przykład 16: Dodaj kolumnę date_added z domyślną bieżącą datą/czasem.

```
ALTER TABLE sakila.favorite_actor  
ADD date_added DATETIME DEFAULT NOW();
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Wyjaśnienie krok po kroku:

1. ALTER TABLE sakila.favorite_actor
 - Określa tabelę favorite_actor, którą chcemy zmodyfikować.
2. ADD date_added DATETIME
 - Dodaje nową kolumnę date_added typu DATETIME do tabeli.
3. DEFAULT NOW()
 - Ustawia domyślną wartość kolumny na aktualną datę i czas w momencie wstawiania rekordu.
 - Dzięki temu, jeśli nie podamy wartości dla date_added przy wstawianiu rekordu, system automatycznie zapisze datę i godzinę dodania.

Co zrobi zapytanie:

- Tabela favorite_actor zyska nową kolumnę date_added.
- Każdy nowo dodany rekord automatycznie zapisze datę i godzinę utworzenia, jeśli nie zostanie podana wartość ręcznie.

Przykład 17: Usuń tabelę favorite_actor.

```
DROP TABLE sakila.favorite_actor;
```

Wyjaśnienie krok po kroku:

1. DROP TABLE sakila.favorite_actor
 - Usuwa całkowicie tabelę favorite_actor z bazy sakila.
 - Tabela wraz ze wszystkimi rekordami, strukturą kolumn, kluczami i indeksami **zostaje trwale usunięta**.

Ważne informacje:

- Operacja jest **nieodwracalna** – po wykonaniu zapytania dane zostają utracone.
- Należy upewnić się, że naprawdę chcemy usunąć tabelę, szczególnie jeśli zawiera ważne dane.
- Jeśli tabela była powiązana kluczem obcym z innymi tabelami, MySQL może zgłosić błąd lub automatycznie usunąć zależności w zależności od ustawień (CASCADE).

Ćwiczenia samodzielne



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



1. Wyświetl wszystkie kolumny z sakila.category.
2. Wyświetl first_name, last_name z sakila.actor.
3. Wyświetl first_name jako imię i last_name jako nazwisko (aliasy).
4. Wyświetl title i release_year z sakila.film.
5. Posortuj aktorów rosnąco po last_name.
6. Posortuj aktorów malejąco po last_name.
7. Posortuj klientów rosnąco po last_name, a następnie po first_name.
8. Wyświetl klientów ze sklepu o store_id = 2, posortowanych malejąco po create_date.
9. Wyświetl tytuły i długości filmów, które mają length > 120.
10. Wyświetl klientów z create_date >= '2006-01-01', posortowanych po create_date.
11. Wyświetl kraje (country) rosnąco alfabetycznie.
12. Wyświetl filmy (title, rental_rate) oraz dodatkową kolumnę rental_rate * 1.2 jako „podwyżka_20”.
13. Wyświetl aktorów o actor_id < 50 (pokaż first_name, last_name, actor_id).
14. Wyświetl title, release_year filmów, których rok produkcji jest < 2000 lub > 2002.
15. Wyświetl 10 pierwszych wierszy z sakila.actor wg actor_id.
16. Pomiń 20 pierwszych aktorów i pokaż kolejnych 10 (LIMIT/OFFSET).
17. Wyświetl first_name, last_name, actor_id, sortując rosnąco po actor_id, a w ramach tego malejąco po last_name.
18. Wyświetl last_name, last_update, actor_id oraz actor_id + 100 jako id_po_zmianie.
19. Wyświetl wszystkie kolumny z sakila.category dla wierszy z category_id > 5.
20. Wyświetl wszystkie kolumny z sakila.country dla wierszy z country_id > 50.
21. Wstaw nowego aktora ('Anna', 'Nowak') do sakila.actor.
22. Wstaw dwóch aktorów jednocześnie (dowolne poprawne dane).
23. Zmień nazwisko wybranego aktora (podaj warunek po actor_id).
24. Dodaj sufiks '*' do last_name aktorów o actor_id < 10.
25. Usuń aktora o podanym actor_id (upewnij się, że istnieje).
26. twórz tabelę p3_notes z kolumnami: note_id INT AUTO_INCREMENT (PK), note_text VARCHAR(200) NOT NULL.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



27. Dodaj do p3_notes kolumnę created_at typu DATETIME z domyślną wartością bieżącą.
28. Zmień typ note_text na VARCHAR(255).
29. Zmień nazwę tabeli p3_notes na p3_notes_archive.
30. Usuń tabelę p3_notes_archive.

Pytania pomocnicze

1. Czym jest baza danych i do czego służy?
2. Jakie są podstawowe rodzaje baz danych i czym się różnią?
3. Dlaczego SQL jest ważny w pracy z relacyjnymi bazami danych?
4. Do czego służy przykładowa baza Sakila i co zawiera?
5. Jakie kroki należy wykonać, aby zaimportować bazę Sakila w phpMyAdmin?
6. Na czym polegają operacje DML i DDL w SQL?
7. Jakie są przykłady poleceń DML i do czego się ich używa?
8. Jakie są przykłady poleceń DDL i jakie mają zastosowanie?
9. Czym różnią się instrukcje DELETE i TRUNCATE?
10. Dlaczego relacje między tabelami są istotne w relacyjnych bazach danych?

Podsumowanie

Bazy danych to uporządkowane zbiory informacji, które umożliwiają ich przechowywanie, organizację i szybki dostęp. Kluczową rolę w pracy z nimi odgrywa język SQL, pozwalający tworzyć struktury baz, wstawiać, modyfikować i usuwać dane oraz pobierać je zgodnie z określonymi kryteriami. Przykładowa baza Sakila, symulująca działanie wypożyczalni filmów, stanowi narzędzie do nauki i ćwiczeń, dzięki któremu można praktycznie poznać podstawowe operacje DML i DDL. Zrozumienie tych mechanizmów pozwala lepiej opanować relacyjny model baz danych i stanowi fundament dalszej pracy z systemami informatycznymi.

Literatura

Karwin, Bill, i in. Antywzorce języka SQL : jak uniknąć pułapek podczas programowania baz danych. Wyd. Helion, 2012.

<https://dev.mysql.com/doc/sakila/en/>

<https://www.w3schools.com/sql/>



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Materiały do przedmiotu:

Bazy danych

Laboratorium nr 4:

Wybrane polecenia języka SQL: Filtry, unie, funkcje agregujące, łączenie tabel, operacje na danych

Autor:

Alicja Żmudzińska



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Spis treści

Wprowadzenie, tematyka zajęć	3
Cel ćwiczenia/laboratorium	3
Instrukcja laboratoryjna/ćwiczeniowa	3
Ćwiczenia samodzielne	14
Pytania pomocnicze	16
Podsumowanie	16
Literatura	16



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Wprowadzenie, tematyka zajęć

SQL to znacznie więcej niż `SELECT * FROM ...`; — to język, który pozwala wydobyć z bazy dokładnie te informacje, które są potrzebne, połączyć je z różnych tabel i przygotować tak, by nadawały się do raportu lub analizy. Na tych zajęciach przejdziemy od prostych filtrów (klauszula `WHERE`, operatory porównania i logiczne) przez zaawansowane techniki filtrowania (`LIKE`, `IN`, `BETWEEN`, `IS NULL`, `EXISTS`, `REGEXP`) aż po łączenia tabel (`JOIN`), operacje na datach i czasie oraz agregacje z `GROUP BY` i `HAVING`. Będziemy korzystać z bazy Sakila, aby ćwiczyć rzeczywiste scenariusze: wyszukiwanie rekordów, łączenie relacji, obsługę braków danych oraz tworzenie podsumowań i rankingów.

Cel ćwiczenia/laboratorium

Celem zajęć jest nauczenie pisania czytelnego, poprawnego i efektywnego zapytania, takiego, które wybiera tylko potrzebne kolumny, filtrowanych przy użyciu właściwych warunków, łączących właściwe tabele i zwracających wynik gotowy do dalszej analizy.

Instrukcja laboratoryjna/ćwiczeniowa

Pisząc zapytania SQL, zwykle nie ograniczamy się tylko do prostego `SELECT * FROM ...`. Najczęściej chodzi o coś więcej: o wyciągnięcie tylko potrzebnych danych, połączenie ich z różnych tabel i odpowiednie przekształcenie, żeby dało się z tego zrobić raport albo analizę. Do tego właśnie służą różne konstrukcje języka SQL.

Filtry pozwalają ograniczyć liczbę zwracanych wierszy poprzez stosowanie warunków. Najczęściej używa się do tego klauzuli `WHERE`, w której można korzystać zarówno z operatorów porównania (`=`, `<`, `>`, `<>`), jak i logicznych (`AND`, `OR`, `NOT`). Dzięki nim wybieramy tylko te rekordy, które naprawdę nas interesują.

Przykład 1: Wyświetl wszystkich aktorów o nazwisku „Davis”

```
SELECT *  
FROM sakila.actor  
WHERE last_name = 'Davis';
```

Wyjaśnienie krok po kroku:

1. `SELECT first_name AS imie` – pobiera kolumnę `first_name` i nadaje jej alias `imie`.
2. `last_name AS nazwisko` – pobiera `last_name` i zmienia nagłówek na `nazwisko`.
3. `FROM sakila.actor` – wskazuje tabelę `actor` w bazie `sakila`.
4. `WHERE last_name = Davis` – filtruje wiersze do tych z nazwiskiem `Davis`.

Co zwróci polecenie: listę aktorów o nazwisku `Davis` (kolumny: `imie`, `nazwisko`).



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



✓ Pokazano wiersze 0 - 2 (3 total, Wykonanie zapytania trwało 0,0008 sekund(y).)

```
SELECT * FROM sakila.actor WHERE last_name = 'Davis';
```

☐ Profilowanie [[Edytuj w linii](#)] [[Edit](#)] [[Explain SQL](#)] [[Create PHP code](#)] [[Refresh](#)]

☐ Show all | Liczba wierszy: 25 | Filter rows: Przeszukaj tę tabelę | Sortuj w

Extra options

	actor_id	first_name	last_name	last_update
☐ Edit Copy Delete	4	JENNIFER	DAVIS	2006-02-15 04:34:33
☐ Edit Copy Delete	101	SUSAN	DAVIS	2006-02-15 04:34:33
☐ Edit Copy Delete	110	SUSAN	DAVIS	2006-02-15 04:34:33

Dodatkowe operatory filtrowania:

LIKE / NOT LIKE – dopasowanie wzorca:

% – dopasowuje **dowolny ciąg znaków (także pusty)**

- 'A%' → wszystko, co zaczyna się na „A” (np. AL, ANNA, A)
- '%son' → wszystko, co kończy się na „son” (np. Jackson, Harrison)
- '%ann%' → wszystko, co **zawiera** „ann” gdziekolwiek).

_ (podkreślnik) – dopasowuje dokładnie jeden dowolny znak

- '_a%' → drugi znak to „a” (np. Jack, Max)
- 'J_n' → Jan, Jon, Jin itd.

Przykład: Wyświetl aktorów z nazwiskiem zaczynającym się na „A”

```
SELECT first_name, last_name
FROM sakila.actor
WHERE last_name LIKE 'A%';
```

Wyjaśnienie krok po kroku:

1. SELECT first_name, last_name – wybiera imię i nazwisko aktora.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



2. FROM sakila.actor – z tabeli aktorów.
3. WHERE last_name LIKE 'A%' – filtr: nazwisko zaczyna się od „A”.

Co zwróci polecenie: listę aktorów, których nazwisko zaczyna się od litery „A”.

✓ Pokazano wiersze 0 - 6 (7 total, Wykonanie zapytania trwało 0,0008 sekund(y).)

```
SELECT first_name, last_name FROM sakila.actor WHERE last_name LIKE 'A%';
```

☐ Profilowanie [[Edytuj w linii](#)] [[Edit](#)] [[Explain SQL](#)] [[Create PHP code](#)] [[Refresh](#)]

☐ Show all | Liczba wierszy: 25 | Filter rows:

Extra options

	first_name	last_name
☐ Edit Copy Delete	CHRISTIAN	AKROYD
☐ Edit Copy Delete	KIRSTEN	AKROYD
☐ Edit Copy Delete	DEBBIE	AKROYD
☐ Edit Copy Delete	CUBA	ALLEN
☐ Edit Copy Delete	KIM	ALLEN
☐ Edit Copy Delete	MERYL	ALLEN
☐ Edit Copy Delete	ANGELINA	ASTAIRE

IN/NOT IN – filtr do listy wartości lub podzapytania.

Przykład: klienci z wybranych miast.

```
SELECT c.first_name, c.last_name
FROM sakila.customer c
JOIN sakila.address a USING(address_id)
JOIN sakila.city ci USING(city_id)
WHERE ci.city IN ('London', 'Paris', 'Rome');
```

Wyjaśnienie krok po kroku:

1. JOIN łączy klientów z ich adresem i miastem.
2. WHERE ci.city IN (...) – przepuszcza tylko wiersze z listy miast.
3. SELECT zwraca imię i nazwisko klientów spełniających warunek.

Co zwróci polecenie: klientów mieszkających w Londynie, Paryżu lub Rzymie



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



✓ Pokazano wiersze 0 - 1 (2 total, Wykonanie zapytania trwało 0,0048 sekund(y).)

```
SELECT c.first_name, c.last_name FROM sakila.customer c JOIN sakila.address a USING(address_id) JOIN sakila.city ci USING(city_id) WHERE ci.city IN ('London', 'Paris', 'Rome');
```

☐ Profilowanie [Edytuj w linii] [Edit] [Explain SQL] [Create PHP code] [Refresh]

☐ Show all | Liczba wierszy: 25 | Filter rows: Przeszukaj tę tabelę | Sortuj wg klucza: Żaden

Extra options

first_name	last_name
MATTIE	HOFFMAN
CECIL	VINES

BETWEEN / NOT BETWEEN – zakres (włącznie z granicami).

Przykład: filmy o stawce wypożyczenia w przedziale 0.99–2.99.

```
SELECT title, rental_rate
FROM sakila.film
WHERE rental_rate BETWEEN 0.99 AND 2.99;
```

Wyjaśnienie krok po kroku:

1. SELECT title, rental_rate – pobiera tytuł i stawkę.
2. WHERE rental_rate BETWEEN 0.99 AND 2.99 – wybiera wiersze w podanym zakresie (łącznie z 0.99 i 2.99).

Co zwróci polecenie: filmy, których rental_rate mieści się w tym przedziale.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



✓ Pokazano wiersze 0 - 24 (664 total, Wykonanie zapytania trwało 0,0004 sekund(y).)

```
SELECT title, rental_rate FROM sakila.film WHERE rental_rate BETWEEN 0.99 AND 2.99;
```

☐ Profilowanie [Edytuj w linii] [Edit] [Explain SQL] [Create PHP code] [Refresh]

1 > >> | Liczba wierszy: 25 | Filter rows: Przeszukaj tę tabelę S

Extra options

	title	rental_rate
☐ Edit Copy Delete	ACADEMY DINOSAUR	0.99
☐ Edit Copy Delete	ADAPTATION HOLES	2.99
☐ Edit Copy Delete	AFFAIR PREJUDICE	2.99
☐ Edit Copy Delete	AFRICAN EGG	2.99
☐ Edit Copy Delete	AGENT TRUMAN	2.99
☐ Edit Copy Delete	ALABAMA DEVIL	2.99
☐ Edit Copy Delete	ALAMO VIDEOTAPE	0.99
☐ Edit Copy Delete	ALASKA PHANTOM	0.99
☐ Edit Copy Delete	ALICE FANTASIA	0.99
☐ Edit Copy Delete	ALIEN CENTER	2.99

IS NULL / IS NOT NULL – sprawdzanie braków danych.

Przykład: wypożyczenia bez daty zwrotu.

```
SELECT rental_id, customer_id
FROM sakila.rental
WHERE return_date IS NULL;
```

Wyjaśnienie krok po kroku:

1. SELECT rental_id, customer_id – wybiera identyfikator wypożyczenia i klienta.
2. FROM sakila.rental – z tabeli wypożyczeń.
3. WHERE return_date IS NULL – filtr: brak daty zwrotu.

Co zwróci polecenie: wypożyczenia, które nie zostały jeszcze zwrócone.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



✓ Pokazano wiersze 0 - 24 (183 total, Wykonanie zapytania trwało 0,0090 sekund(y).)

```
SELECT rental_id, customer_id FROM sakila.rental WHERE return_date IS NULL;
```

☐ Profilowanie [[Edytuj w linii](#)] [[Edit](#)] [[Explain SQL](#)] [[Create PHP code](#)] [[Refresh](#)]

1 > >> ☐ Show all | Liczba wierszy: 25 Filter rows: Przeł

Extra options

	rental_id	customer_id
☐ Edit Copy Delete	11496	155
☐ Edit Copy Delete	11541	335
☐ Edit Copy Delete	11563	83
☐ Edit Copy Delete	11577	219
☐ Edit Copy Delete	11593	99
☐ Edit Copy Delete	11611	192
☐ Edit Copy Delete	11646	11
☐ Edit Copy Delete	11652	597
☐ Edit Copy Delete	11657	53
☐ Edit Copy Delete	11672	521

EXISTS / NOT EXISTS – sprawdzenie istnienia powiązanych wierszy (wydajne przy podzapytaniach).

Przykład: klienci, którzy mają jakiegokolwiek płatności.

```
SELECT c.customer_id, c.first_name, c.last_name
FROM sakila.customer c
WHERE EXISTS (
  SELECT 1 FROM sakila.payment p
  WHERE p.customer_id = c.customer_id);
```

Wyjaśnienie krok po kroku:

1. Zewnętrzne SELECT – kandydaci: wszyscy klienci.
2. EXISTS (podzapytanie) – sprawdza, czy istnieje **co najmniej jeden** wiersz w payment dla danego klienta.
3. Jeśli tak – klient przechodzi filtr; jeśli nie – jest odrzucony.

Co zwróci polecenie: klientów, którzy mają przynajmniej jedną płatność.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



✓ Pokazano wiersze 0 - 24 (599 total, Wykonanie zapytania trwało 0,0065 sekund(y).)

```
SELECT c.customer_id, c.first_name, c.last_name FROM sakila.customer c WHERE EXISTS ( SELECT 1 FROM sakila.payment p WHERE p.customer_id = c.customer_id );
```

☐ Profilowanie [Edytuj w linii] [Edit] [Explain SQL] [Create PHP code] [Refresh]

1 > >> | Liczba wierszy: 25 | Filter rows: Przeszukaj tę tabelę | Sortuj wg klucza: Żaden

Extra options

	customer_id	first_name	last_name
☐ Edit Copy Delete	1	MARY	SMITH
☐ Edit Copy Delete	2	PATRICIA	JOHNSON
☐ Edit Copy Delete	3	LINDA	WILLIAMS
☐ Edit Copy Delete	4	BARBARA	JONES
☐ Edit Copy Delete	5	ELIZABETH	BROWN
☐ Edit Copy Delete	6	JENNIFER	DAVIS
☐ Edit Copy Delete	7	MARIA	MILLER
☐ Edit Copy Delete	8	SUSAN	WILSON
☐ Edit Copy Delete	9	MARGARET	MOORE

REGEXP – dopasowanie wyrażeń regularnych (precyzyjniejsze niż LIKE).

Przykład: nazwiska zaczynające się od „Mc” lub „Mac”.

```
SELECT first_name, last_name
FROM sakila.actor
WHERE last_name REGEXP '^(Mc|Mac)';
```

Wyjaśnienie krok po kroku:

1. FROM sakila.actor – wskazuje tabelę z danymi aktorów, z której będziemy pobierać wiersze.
2. WHERE last_name REGEXP '^(Mc|Mac)' – filtruje wiersze za pomocą wyrażenia regularnego:
 - REGEXP – operator dopasowania wzorca (regex) w MySQL.
 - ^ – kotwica początku tekstu (wzorec musi pasować od pierwszego znaku nazwiska).
 - (Mc|Mac) – alternatywa: pasuje „Mc” lub „Mac”.

Co zwróci polecenie: Listę aktorów, których nazwisko zaczyna się na „Mc” lub „Mac”



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



✓ Pokazano wiersze 0 - 6 (7 total, Wykonanie zapytania trwało 0,0020 sekund(y).)

```
SELECT first_name, last_name FROM sakila.actor WHERE last_name REGEXP '^(MC|Mac)';
```

☐ Profilowanie [Edytuj w linii] [Edit] [Explain SQL] [Create PHP code] [Refresh]

☐ Show all | Liczba wierszy: 25 | Filter rows: Przeszukaj tę tabelę | Sortuj w

Extra options

	first_name	last_name
☐ Edit Copy Delete	JULIA	MCQUEEN
☐ Edit Copy Delete	TOM	MCKELLEN
☐ Edit Copy Delete	MICHELLE	MCCONAUGHEY
☐ Edit Copy Delete	CARY	MCCONAUGHEY
☐ Edit Copy Delete	MORGAN	MCDORMAND
☐ Edit Copy Delete	CATE	MCQUEEN
☐ Edit Copy Delete	GENE	MCKELLEN

W relacyjnych bazach danych informacje są podzielone na różne, powiązane ze sobą tabele. Aby uzyskać pełny obraz, musimy je połączyć, a służy do tego polecenie **JOIN**. Podstawą takich połączeń są klucze: **PRIMARY KEY**, czyli unikalny identyfikator w tabeli, oraz **FOREIGN KEY**, wskazujący na rekord w innej tabeli. W bazie Sakila przykładem jest tabela inventory, w której kolumna film_id jest kluczem obcym odnoszącym się do tabeli film. Najczęściej spotykane rodzaje złączeń to: **INNER JOIN**, zwracający tylko pasujące rekordy z obu tabel; **LEFT JOIN**, który zwraca wszystkie rekordy z tabeli po lewej oraz pasujące z prawej, a brakujące uzupełnia wartością NULL; **RIGHT JOIN**, czyli odwrotność **LEFT JOIN**; **FULL OUTER JOIN**, obejmujący wszystkie rekordy z obu tabel (w MySQL trzeba go symulować za pomocą **UNION**); **CROSS JOIN**, który tworzy iloczyn kartezjański obu tabel; oraz **SELF JOIN**, w którym tabela łączy się sama ze sobą, np. w celu porównania rekordów.

Przykład

```
SELECT f.title AS tytuł, a.last_name AS nazwisko
FROM sakila.film AS f
INNER JOIN sakila.film_actor AS fa ON f.film_id = fa.film_id
INNER JOIN sakila.actor AS a ON fa.actor_id = a.actor_id;
```

Wyjaśnienie krok po kroku:

1. sakila.film AS f – tabela filmów (alias f).
2. sakila.film_actor AS fa – tabela wiążąca filmy z aktorami.
3. sakila.actor AS a – dane aktorów.
4. INNER JOIN ... ON f.film_id = fa.film_id – związanie filmów z relacjami.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



5. INNER JOIN ... ON fa.actor_id = a.actor_id – dociągnięcie aktorów.

6. SELECT f.title AS tytuł, a.last_name AS nazwisko – wybór i aliasy.

Co zwróci polecenie: listę par „tytuł filmu – nazwisko aktora”.

✓ Pokazano wiersze 0 - 24 (5462 total, Wykonanie zapytania trwało 0,0014 sekund(y).)

```
SELECT f.title AS tytuł, a.last_name AS nazwisko FROM sakila.film AS f INNER JOIN sakila.film_actor AS fa ON f.film_id = fa.film_id INNER JOIN sakila.actor AS a ON fa.actor_id = a.actor_id;
```

☐ Profilowanie [Edytuj w linii] [Edit] [Explain SQL] [Create PHP code] [Refresh]

1 > >> Liczba wierszy: 25 Filter rows: Przeszukaj tę tabelę Sortuj wg klucza: Żaden

Extra options

tytuł	nazwisko
BACKLASH UNDEFEATED	AKROYD
BETRAYED REAR	AKROYD
CAPER MOTIONS	AKROYD
CATCH AMISTAD	AKROYD
CHANCE RESURRECTION	AKROYD
CONFUSED CANDLES	AKROYD
CUPBOARD SINNERS	AKROYD
DIVIDE MONSTER	AKROYD
DOOM DANCING	AKROYD

Na koniec warto wspomnieć o pracy z datami i czasem, bo to bardzo częsty element analiz. W bazie Sakila występują na przykład kolumny rental_date, return_date czy payment_date. SQL obsługuje różne typy czasowe: DATE, zawierający wyłącznie datę (rok, miesiąc, dzień), DATETIME i TIMESTAMP, które przechowują zarówno datę, jak i godzinę, oraz TIME, przechowujący sam czas. Przykładowo w tabeli rental kolumna rental_date zapisuje dokładny moment wypożyczenia filmu. Do pracy z datami przydają się również funkcje systemowe, takie jak NOW(), zwracająca bieżącą datę i godzinę, CURRENT_DATE, która podaje dzisiejszą datę, czy CURRENT_TIMESTAMP, będąca odpowiednikiem funkcji NOW().

Przykład

```
SELECT rental_id AS id, rental_date AS data_wypozyczenia
FROM sakila.rental
WHERE rental_date <= CURRENT_DATE;
```

Wyjaśnienie krok po kroku:

1. sakila.rental – historia wypożyczeń.
2. rental_id AS id – identyfikator z aliasem id.
3. rental_date AS data_wypozyczenia – data/godzina wypożyczenia.
4. WHERE rental_date <= CURRENT_DATE – zwraca rekordy do początku dzisiejszego dnia (00:00:00) włącznie.

Co zwróci polecenie:



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



✓ Pokazano wiersze 0 - 24 (16044 total, Wykonanie zapytania trwało 0,0003 sekund(y).)

```
SELECT rental_id AS id, rental_date AS data_wypozyczenia FROM sakila.rental WHERE rental_date <= CURRENT_DATE;
```

☐ Profilowanie [Edytuj w linii] [Edit] [Explain SQL] [Create PHP code] [Refresh]

1 > >> | Liczba wierszy: 25 | Filter rows: Przeszukaj tę tabelę | Sortuj wg klucza: Żaden

Extra options

		id	data_wypozyczenia
☐	Edit	1	2005-05-24 22:53:30
☐	Edit	2	2005-05-24 22:54:33
☐	Edit	3	2005-05-24 23:03:39
☐	Edit	4	2005-05-24 23:04:41
☐	Edit	5	2005-05-24 23:05:21
☐	Edit	6	2005-05-24 23:08:07
☐	Edit	7	2005-05-24 23:11:53
☐	Edit	8	2005-05-24 23:31:46
☐	Edit	9	2005-05-25 00:00:40
☐	Edit	10	2005-05-25 00:02:21

Kiedy dane pochodzą z kilku zapytań i chcemy je połączyć w jedno zestawienie, korzystamy z unii. Operator UNION łączy dane i usuwa duplikaty, dzięki czemu każda kombinacja wartości pojawia się tylko raz. Z kolei UNION ALL działa szybciej, ponieważ łączy wszystkie wyniki, pozostawiając duplikaty i pozwalając zobaczyć faktyczną liczbę danych. Trzeba jednak pamiętać, że każde z łączonych zapytań musi zwracać tę samą liczbę kolumn o zgodnych typach danych.

Przykład 2: Wyświetl listę imion aktorów oraz imion klientów

```
SELECT first_name FROM sakila.actor
UNION
SELECT first_name FROM sakila.customer;
```

Wyjaśnienie krok po kroku:

1. Pierwsze zapytanie wybiera first_name z tabeli actor i podaje je jako imię.
2. Drugie zapytanie wybiera first_name z customer, również jako imię.



Fundusze Europejskie
dla Rozwoju Społecznego



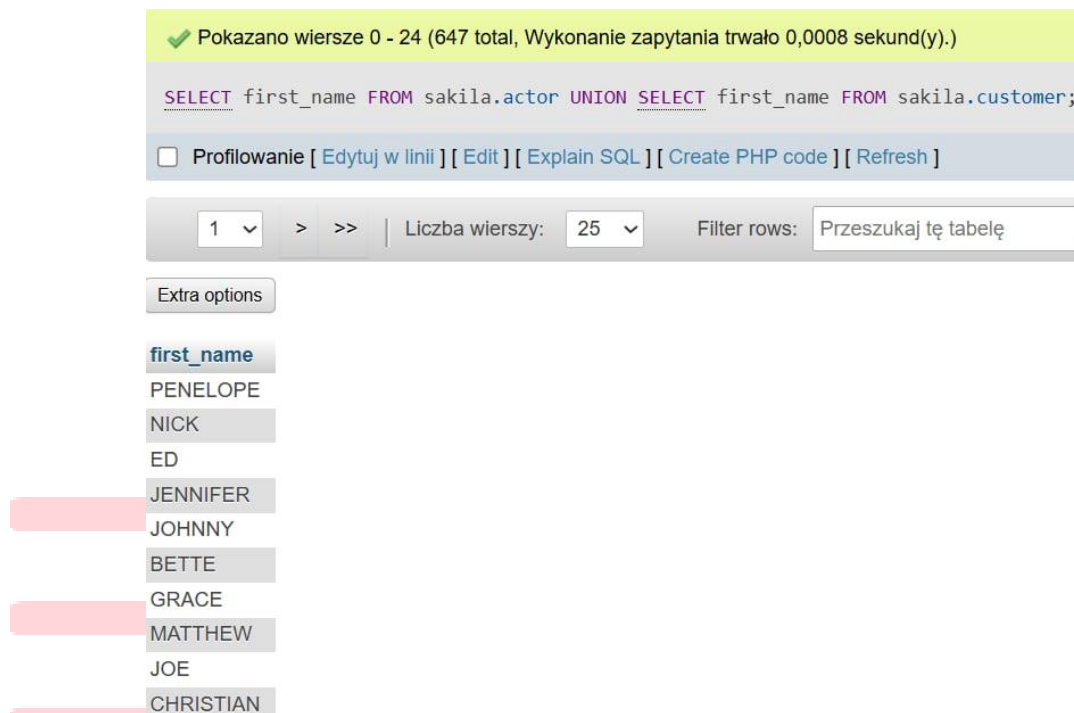
Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



3. UNION – łączy oba wyniki i usuwa duplikaty (wymaga takiej samej liczby i typów kolumn po obu stronach).

Co zwróci polecenie: unikalną listę imion z obu tabel.



✓ Pokazano wiersze 0 - 24 (647 total, Wykonanie zapytania trwało 0,0008 sekund(y).)

```
SELECT first_name FROM sakila.actor UNION SELECT first_name FROM sakila.customer;
```

☐ Profilowanie [Edytuj w linii] [Edit] [Explain SQL] [Create PHP code] [Refresh]

1 > >> | Liczba wierszy: 25 | Filter rows: Przeszukaj tę tabelę

Extra options

first_name

PENELOPE

NICK

ED

JENNIFER

JOHNNY

BETTE

GRACE

MATTHEW

JOE

CHRISTIAN

SQL umożliwia także wykonywanie obliczeń na całych zbiorach rekordów, a nie tylko na pojedynczych wierszach. Służą do tego funkcje agregujące: COUNT() do zliczania wierszy, SUM() do sumowania wartości, AVG() do obliczania średniej, a także MIN() i MAX() do znajdowania najmniejszej lub największej wartości. Żeby miały sens, często stosuje się je razem z klauzulą GROUP BY, która grupuje dane według wybranej kolumny. Jeśli dodatkowo chcemy odfiltrować tylko niektóre grupy, używamy klauzuli HAVING, która działa już po agregacji, w przeciwieństwie do WHERE.

```
SELECT city_id, COUNT(*) AS ile
FROM sakila.address
GROUP BY city_id
ORDER BY ile DESC
LIMIT 10;
```

Wyjaśnienie krok po kroku:

1. city_id – kolumna zwracana i kolumna grupująca (identyfikator miasta).
2. COUNT(*) AS ile – zlicza wiersze (adresy) w każdej grupie miasta; alias wyniku to ile.
3. FROM sakila.address – źródło danych: tabela adresów.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



4. GROUP BY city_id – grupuje rekordy według miasta, aby COUNT(*) liczył per miasto.
5. ORDER BY ile DESC – sortuje miasta malejąco według liczby adresów.
6. LIMIT 10 – zwraca Top 10 miast o największej liczbie adresów.

Co zwróci polecenie: dziesięć city_id z największą liczbą wierszy w sakila.address wraz z tą liczbą

✓ Pokazano wiersze 0 - 9 (10 total, Wykonanie zapytania trwało 0,0004 sekund(y).)

```
SELECT city_id, COUNT(*) AS ile FROM sakila.address GROUP BY city_id ORDER BY ile DESC LIMIT 10;
```

☐ Profilowanie [[Edytuj w linii](#)] [[Edit](#)] [[Explain SQL](#)] [[Create PHP code](#)] [[Refresh](#)]

Extra options

city_id	ile
576	2
312	2
42	2
300	2
13	1
29	1
45	1
61	1
77	1
93	1

Ćwiczenia samodzielne

1. Wyświetl wszystkich aktorów o nazwisku „SMITH”.
2. Wyświetl klientów z miasta „London”.
3. Wyświetl tytuły filmów dłuższych niż 120 minut.
4. Wyświetl filmy, których koszt wypożyczenia mieści się w przedziale od 0.99 do 2.99.
5. Wyświetl aktorów, których nazwisko zaczyna się na literę „A”.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



6. Wyświetl płatności większe niż 10.
7. Wyświetl wszystkie wypożyczenia dokonane przez klienta o nazwisku „Brown”.
8. Wyświetl listę imion aktorów i klientów (bez duplikatów).
9. Wyświetl listę imion aktorów i klientów (z duplikatami).
10. Wyświetl wspólną listę nazwisk aktorów i pracowników.
11. Wyświetl liczbę aktorów dla każdej litery alfabetu (pierwsza litera nazwiska).
12. Wyświetl średnią długość filmów w kategoriach „Action” i „Comedy”.
13. Wyświetl liczbę filmów w każdej kategorii.
14. Wyświetl 10 miast z największą liczbą klientów.
15. Wyświetl klientów, którzy wypożyczyli więcej niż 20 filmów.
16. Wyświetl sumę płatności dla każdego klienta.
17. Wyświetl minimalną i maksymalną opłatę za wypożyczenie filmu.
18. Wyświetl tytuły filmów wraz z językiem, w którym zostały nagrane.
19. Wyświetl listę klientów wraz z nazwą miasta, w którym mieszkają.
20. Wyświetl wszystkich aktorów grających w filmie „ACADEMY DINOSAUR”.
21. Wyświetl tytuły filmów należących do kategorii „Action”.
22. Wyświetl wszystkich klientów i sumę ich płatności (uwzględnij także tych, którzy nie mają żadnych płatności).
23. Wyświetl pary aktorów mających to samo nazwisko.
24. Wyświetl listę klientów i ich płatności używając symulacji pełnego złączenia (FULL OUTER JOIN).
25. Wyświetl listę filmów z liczbą aktorów występujących w każdym z nich.
26. Wyświetl wszystkie wypożyczenia z ostatnich 7 dni.
27. Wyświetl łączną kwotę płatności dokonanych w bieżącym miesiącu.
28. Wyświetl średnią kwotę płatności w podziale na dni.
29. Wyświetl klientów, którzy nie wypożyczali filmów od 60 dni.
30. Wyświetl listę wypożyczeń wraz z czasem (w dniach) pomiędzy wypożyczeniem a zwrotem filmu.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Pytania pomocnicze

1. Po co w ogóle stosujemy klauzulę WHERE w zapytaniach SQL?
2. Dlaczego LIKE 'A%' i LIKE '%A' zwrócą różne wyniki?
3. Czym różni się INNER JOIN od LEFT JOIN?
4. Jaka jest różnica między UNION a UNION ALL?
5. Dlaczego oba zapytania łączone przez UNION muszą zwracać tę samą liczbę kolumn?
6. Kiedy stosujemy klauzulę GROUP BY?
7. Do czego służy znak %, a do czego _ w operatorze LIKE?

Podsumowanie

SQL to narzędzie, które pozwala nie tylko pobierać dane, ale też je filtrować, łączyć i analizować. Dzięki klauzuli WHERE możemy ograniczyć liczbę wyników do tych, które nas interesują. Operatory takie jak LIKE, IN, BETWEEN, IS NULL czy EXISTS pozwalają formułować bardziej precyzyjne warunki. Łączenie tabel (JOIN) umożliwia uzyskanie pełnego obrazu danych z relacyjnej bazy, a praca z datami i czasem jest kluczowa w analizach biznesowych. Z kolei UNION pozwala łączyć wyniki z wielu zapytań, a funkcje agregujące (COUNT, SUM, AVG, MIN, MAX) wraz z GROUP BY i HAVING dają możliwość tworzenia zestawień i raportów.

Literatura

Karwin, Bill, i in. Antywzorce języka SQL : jak uniknąć pułapek podczas programowania baz danych. Wyd. Helion, 2012.

<https://www.w3schools.com/sql/>



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



a

Materiały do przedmiotu:

Bazy danych

Laboratorium nr 5:

**Pisanie kwerend wykorzystujących podzapytania
nieskorelowane oraz skorelowane w celu uzyskania
specyficznych informacji z bazy danych**

Autor:

mgr Alicja Żmudzińska



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Spis treści

Wprowadzenie, tematyka zajęć	3
Cel ćwiczenia/laboratorium	3
Instrukcja laboratoryjna/ćwiczeniowa	3
Ćwiczenia samodzielne	8
Pytania pomocnicze	10
Podsumowanie	10
Literatura	11



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Wprowadzenie, tematyka zajęć

W języku SQL często zachodzi potrzeba uzyskania informacji, które nie wynikają bezpośrednio z jednej tabeli czy prostego zapytania. Zdarza się, że zanim pobierzemy ostateczne dane, musimy najpierw coś policzyć, przefiltrować lub porównać z wynikiem innego zapytania. Właśnie do tego służą podzapytania, nazywane także subqueries. Podzapytanie to po prostu zapytanie umieszczone wewnątrz innego zapytania SQL. Można je potraktować jako „zapytanie w zapytaniu” najpierw wykonywane jest zapytanie wewnętrzne, a jego wynik jest następnie używany przez zapytanie główne. Dzięki temu SQL pozwala uzyskiwać bardzo precyzyjne i złożone informacje, bez konieczności tworzenia dodatkowych tabel czy widoków.

Cel ćwiczenia/laboratorium

Celem ćwiczeń jest poznanie zasad tworzenia oraz stosowania podzapytań w języku SQL, a także zrozumienie różnicy pomiędzy podzapytaniami nieskorelowanymi a skorelowanymi. Podczas pracy z bazą danych Sakila uczestnik zajęć nauczy się w praktyce, jak wykorzystać podzapytania do uzyskania bardziej szczegółowych informacji z relacyjnych zbiorów danych. W trakcie ćwiczeń zostaną utrwalone podstawy składni SQL w zakresie zapytań zagnieżdżonych, a także omówione zostaną sytuacje, w których warto zastosować podzapytanie zamiast tradycyjnego złączenia tabel. W ramach realizacji ćwiczeń uczestnik pozna sposób działania zapytań wewnętrznych oraz mechanizm przekazywania wyników pomiędzy zapytaniem podrzędnym a zapytaniem głównym. Uświadomi sobie, że podzapytania mogą być wykonywane niezależnie lub w ściślejszej korelacji z danymi zapytania nadrzędnego, co pozwala uzyskać dynamiczne i precyzyjne wyniki. Ćwiczenia obejmują również praktyczne zastosowanie operatorów logicznych i porównawczych takich jak IN, EXISTS, NOT EXISTS, ANY czy ALL, które stanowią nieodłączny element pracy z podzapytaniami.

Instrukcja laboratoryjna/ćwiczeniowa

Podzapytanie (ang. *subquery*) to zapytanie SQL umieszczone wewnątrz innego zapytania, które można traktować jako „zapytanie w zapytaniu”. Umożliwia ono uzyskanie dodatkowych informacji na podstawie wyników pośrednich i wykorzystanie tych danych w zapytaniu głównym, co sprawia, że SQL staje się niezwykle elastycznym narzędziem do analizy i przetwarzania danych. Podzapytania mogą pojawiać się w różnych częściach instrukcji SQL. Najczęściej w klauzuli WHERE, gdy chcemy porównać dane z innymi wartościami; w klauzuli HAVING, gdy filtrujemy grupy po wykonaniu agregacji; w klauzuli FROM, gdy traktujemy wynik innego zapytania jako tymczasową tabelę; oraz w klauzuli SELECT, gdy chcemy, aby wynik podzapytania został wykorzystany jako dodatkowa kolumna.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Podzapytania mogą przyjmować różne formy w zależności od sposobu, w jaki są powiązane z zapytaniem głównym. W praktyce oznacza to, że niektóre z nich działają niezależnie, a inne muszą odwoływać się do danych z zapytania nadrzędnego. Na tej podstawie można wyróżnić dwa główne typy podzapytań: nieskorelowane i skorelowane.

Podzapytania nieskorelowane to takie, które może być wykonane samodzielnie, niezależnie od zapytania głównego. Najpierw wykonywane jest podzapytanie, jego wynik jest zapamiętywany, a następnie używany przez zapytanie nadrzędne. Tego typu zapytania są zazwyczaj bardziej wydajne, ponieważ SQL musi wykonać je tylko raz.

Przykład 5.1: Wyświetlenie filmów dłuższy niż średnia długość wszystkich filmów

```
SELECT title, length  
FROM sakila.film  
WHERE length > (SELECT AVG(length) FROM sakila.film);
```

Opis działania:

1. Podzapytanie (SELECT AVG(length) FROM sakila.film) oblicza średnią długość wszystkich filmów w bazie.
2. Zapytanie główne pobiera tytuły i długości tylko tych filmów, które są dłuższe niż ta średnia.

To bardzo często spotykany wzorzec – porównanie każdej wartości z wartością globalną (np. średnią, maksymalną lub minimalną).

Przykład 5.2: Wyświetlenie klientów, którzy wypożyczyli jakikolwiek film

```
SELECT first_name, last_name  
FROM sakila.customer  
WHERE customer_id IN (SELECT DISTINCT customer_id FROM sakila.rental);
```

Opis działania:

Podzapytanie wewnętrzne zwraca identyfikatory klientów, którzy dokonali co najmniej jednego wypożyczenia. Zapytanie główne wybiera imiona i nazwiska tych klientów z tabeli customer. To przykład użycia podzapytania z operatorem IN, który sprawdza, czy dana wartość znajduje się w zbiorze wyników podzapytania.

Podzapytania skorelowane nie może zostać wykonane samodzielnie, ponieważ jego wynik zależy od bieżącego wiersza zapytania głównego. W praktyce oznacza to, że dla każdego wiersza z zapytania nadrzędnego SQL ponownie uruchamia podzapytanie, dlatego takie konstrukcje mogą być wolniejsze, ale dają ogromne możliwości analizy danych.

Przykład 5.3: Wyświetlenie filmów droższych niż średnia cena w ich kategorii

```
SELECT f.title, f.rental_rate, c.name AS category  
FROM sakila.film f
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



```
JOIN sakila.film_category fc ON f.film_id = fc.film_id
JOIN sakila.category c ON fc.category_id = c.category_id
WHERE f.rental_rate > (
    SELECT AVG(f2.rental_rate)
    FROM sakila.film f2
    JOIN sakila.film_category fc2 ON f2.film_id = fc2.film_id
    WHERE fc2.category_id = fc.category_id
);
```

Opis działania:

- Dla każdego filmu z zapytania głównego SQL oblicza średnią stawkę wypożyczenia w danej kategorii.
- Następnie zwraca tylko te filmy, których cena wypożyczenia jest wyższa od tej średniej.

To klasyczny przykład podzapytania skorelowanego, ponieważ w podzapytaniu odwołujemy się do `fc.category_id` – kolumny z zapytania nadrzędnego.

Przykład 5.4: Wyświetlenie klientów, którzy wypożyczyli więcej filmów niż wynosi średnia

```
SELECT c.first_name, c.last_name
FROM sakila.customer c
WHERE (
    SELECT COUNT(*)
    FROM sakila.rental r
    WHERE r.customer_id = c.customer_id
) > (
    SELECT AVG(rental_count)
    FROM (
        SELECT COUNT(*) AS rental_count
        FROM sakila.rental
        GROUP BY customer_id
    ) AS sub
);
```

Opis działania:

1. Wewnętrzne podzapytanie oblicza średnią liczbę wypożyczeń przypadającą na jednego klienta.
2. Drugie podzapytanie (skorelowane) liczy, ile razy dany klient wypożyczył film.
3. Do wyniku trafiają tylko ci klienci, którzy wypożyczyli więcej filmów niż średnia.

Podzapytania mogą być wykorzystywane w różnych częściach zapytania SQL:



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Miejsce użycia	Przykład	Zastosowanie
WHERE	WHERE salary > (SELECT AVG(salary) FROM employees)	porównanie wartości z wynikiem podzapytania
HAVING	HAVING COUNT(*) > (SELECT AVG(order_count) FROM ...)	filtrowanie grup po agregacji
FROM	SELECT * FROM (SELECT ...) AS sub	użycie podzapytania jako tymczasowej tabeli
SELECT	SELECT name, (SELECT COUNT(*) FROM orders WHERE ...) AS liczba_zamowien	wstawienie wyniku podzapytania jako kolumny

W podzapytaniach można wykorzystywać różne operatory, które pozwalają na elastyczne porównywanie i filtrowanie danych. Operator IN (oraz jego przeciwieństwo NOT IN) sprawdza, czy dana wartość znajduje się w zbiorze wyników zwróconym przez podzapytanie. Z kolei EXISTS i NOT EXISTS zwracają wartość logiczną TRUE lub FALSE w zależności od tego, czy podzapytanie zwróciło jakiegokolwiek wiersze. Operatory ANY, SOME oraz ALL umożliwiają porównywanie wartości z wieloma elementami jednocześnie, co daje większą kontrolę nad warunkami logicznymi. Natomiast standardowe operatory porównania, takie jak =, < czy >, są stosowane wtedy, gdy podzapytanie zwraca pojedynczą wartość, czyli ma charakter skalarny.

Przykład 5.5. Wyświetlenie filmów dłuższych niż wszystkie filmy z kategorii „Action”.

```
SELECT title, length
FROM sakila.film
WHERE length > ALL (
    SELECT length
    FROM sakila.film f2
    JOIN sakila.film_category fc2 ON f2.film_id = fc2.film_id
    JOIN sakila.category c2 ON fc2.category_id = c2.category_id
    WHERE c2.name = 'Action'
);
```

Opis działania:

1. Podzapytanie wewnętrzne wybiera długości wszystkich filmów należących do kategorii „Action”.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



2. Główne zapytanie pobiera tytuły i długości filmów z tabeli film.
3. Warunek WHERE length > ALL (...) sprawia, że do wyniku trafiają tylko te filmy, które są dłuższe niż **wszystkie** filmy z kategorii „Action”, czyli mają większą długość niż najdłuższy film akcji.

Różnice między podzapytaniami skorelowanymi a nieskorelowanymi:

Cecha	Podzapytanie nieskorelowane	Podzapytanie skorelowane
Zależność od zapytania głównego	Nie	Tak
Liczba wykonań	Jednorazowo	Dla każdego wiersza
Wydajność	Zazwyczaj wyższa	Może być niższa
Można uruchomić samodzielnie	Tak	Nie
Typowe zastosowanie	Porównanie do wartości globalnej	Analiza w kontekście danego wiersza

Podczas tworzenia podzapytań warto stosować kilka dobrych praktyk, które poprawiają wydajność i czytelność kodu SQL. Przede wszystkim zaleca się używanie operatora EXISTS zamiast IN przy porównywaniu dużych zbiorów danych, ponieważ działa on szybciej i jest odporny na błędy wynikające z wartości NULL. Należy również unikać nadmiernego zagnieżdżania wielu poziomów podzapytań – w takich sytuacjach lepszym rozwiązaniem jest skorzystanie z tymczasowych tabel lub instrukcji WITH, które ułatwiają zarządzanie kodem. Dobrą praktyką jest także nadawanie aliasów podzapytaniom umieszczonym w klauzuli FROM, co znacznie poprawia czytelność zapytań. Warto zwracać uwagę na indeksy, szczególnie w kolumnach wykorzystywanych w warunkach korelacji, ponieważ mają one duży wpływ na czas wykonania zapytania. Dodatkowo, korzystanie z polecenia EXPLAIN pozwala przeanalizować plan wykonania zapytania i lepiej zrozumieć sposób, w jaki optymalizator bazy danych przetwarza poszczególne operacje.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Ćwiczenia samodzielne

1. Filmy dłuższe niż średnia długość wszystkich filmów.
Wskazówka: użyj AVG(length) w podzapytaniu w klauzuli WHERE.
2. Filmy droższe niż średnia cena wypożyczenia (rental_rate) wszystkich filmów.
3. Filmy, które kosztują tyle samo co najdroższy film.
Wskazówka: użyj MAX(rental_rate).
4. Filmy, które mają długość równą minimalnej długości filmu w bazie.
Wskazówka: MIN(length).
5. Wyświetl tytuły filmów, które zostały wypożyczone przez klientów z Polski.
Wskazówka: użyj IN i połącz tabele country, city, address, customer.
6. Wyświetl nazwiska aktorów, którzy grali w filmach o długości większej niż średnia długość wszystkich filmów.
7. Znajdź klientów, którzy mają taką samą nazwę miasta jak któryś z pracowników (staff).
8. Wyświetl nazwy kategorii, które mają więcej filmów niż średnia liczba filmów przypadająca na kategorię.
Wskazówka: użyj GROUP BY + HAVING + podzapytania.
9. Wyświetl filmy, których długość jest większa niż średnia długość filmów o tej samej ocenie (rating).
10. Znajdź wszystkich klientów, którzy zapłacili kwotę większą niż średnia wszystkich płatności w tabeli payment.
11. Znajdź klientów, którzy wypożyczyli jakikolwiek film.
Wskazówka: WHERE customer_id IN (SELECT customer_id FROM rental).
11. Znajdź klientów, którzy nigdy nic nie wypożyczyli.
Wskazówka: użyj NOT EXISTS.
12. Wyświetl filmy, które nigdy nie zostały wypożyczone.
Tabele: film, inventory, rental.
13. Znajdź aktorów, którzy grali w przynajmniej jednym filmie z kategorii „Comedy”.
14. Znajdź aktorów, którzy nie wystąpili w żadnym filmie z kategorii „Horror”.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



15. Wyświetl kraje, w których istnieje przynajmniej jeden klient (EXISTS).
16. Znajdź klientów, którzy dokonali przynajmniej jednej płatności większej niż 8.00 USD.
Użyj EXISTS i warunku w podzapytaniu.
17. Znajdź klientów, którzy wypożyczyli wszystkie filmy, które są w kategorii „Action”.
Wskazówka: użyj ALL (lub zagnieżdżonych NOT EXISTS).
18. Wyświetl filmy, które są dłuższe niż jakikolwiek film z kategorii „Family”.
Operator: > ANY.
19. Wyświetl filmy, które są dłuższe niż wszystkie filmy z kategorii „Music”.
Operator: > ALL.
21. Dla każdego klienta wyświetl jego imię, nazwisko i liczbę wypożyczeń.
*Wskazówka: użyj podzapytania w klauzuli SELECT z korelacją r.customer_id = c.customer_id.**
22. Wyświetl klientów, którzy mają więcej wypożyczeń niż średnia liczba wypożyczeń na klienta.
Zagnieżdżenie dwóch poziomów: jedno podzapytanie do liczenia średniej, drugie skorelowane do liczenia wypożyczeń.
23. Znajdź klientów, którzy zapłacili łącznie więcej niż średnia suma płatności wszystkich klientów.
24. Wyświetl tytuły filmów, które zostały wypożyczone więcej razy niż średnia liczba wypożyczeń na film.
25. Znajdź klientów, którzy dokonali swojej ostatniej płatności po dacie ostatniej płatności pracownika o ID = 1.
26. Wyświetl filmy, których stawka wypożyczenia (rental_rate) jest wyższa niż średnia w ich kategorii.
Korelacja przez category_id.
27. Wyświetl klientów, których średnia wartość płatności jest większa niż średnia globalna.
28. Dla każdego aktora wyświetl jego imię, nazwisko i liczbę filmów, w których zagrał.
Podzapytanie w kolumnie SELECT.
29. Wyświetl klientów, którzy mają aktywne wypożyczenia (czyli nie oddali jeszcze filmu).
Użyj EXISTS z warunkiem return_date IS NULL.
30. Znajdź klientów, którzy wypożyczali tylko filmy dłuższe niż 100 minut.
Skorelowane podzapytanie z warunkiem „nie istnieje film krótszy niż 100 minut wypożyczony przez tego klienta”.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Pytania pomocnicze

1. Czym jest podzapytanie i w jakim celu stosuje się je w języku SQL?
2. Na czym polega różnica pomiędzy podzapytaniem nieskorelowanym a skorelowanym?
3. Jak przebiega wykonanie zapytania skorelowanego – czy podzapytanie uruchamiane jest raz, czy dla każdego wiersza osobno?
4. W jakich sytuacjach bardziej odpowiednie będzie użycie podzapytania, a w jakich złączenia tabel (JOIN)?
5. Jakie znaczenie mają operatory IN, EXISTS, ANY i ALL w kontekście podzapytań?
6. Co się stanie, jeśli podzapytanie zwróci więcej niż jeden wiersz, a zapytanie główne oczekuje tylko jednej wartości?
7. Dlaczego operator NOT EXISTS jest w niektórych przypadkach bezpieczniejszy niż NOT IN?
8. W jakich częściach zapytania SQL można stosować podzapytania (np. WHERE, FROM, SELECT, HAVING)?
9. W jaki sposób podzapytania mogą współpracować z funkcjami agregującymi, takimi jak AVG, MAX czy COUNT?
10. Jakie korzyści daje stosowanie podzapytań w analizie danych w porównaniu do prostych zapytań jednowarstwowych?

Podsumowanie

Podzapytania są jednym z najważniejszych elementów języka SQL. Pozwalają wykonywać bardziej zaawansowane analizy, łączyć dane z wielu tabel w sposób pośredni, porównywać wartości w obrębie całych grup lub kategorii, a także budować warunki zależne od danych. Podzapytania nieskorelowane działają jak niezależne mini-zapytania, natomiast skorelowane tworzą dynamiczne zależności między danymi – dzięki czemu możemy uzyskać bardzo precyzyjne wyniki, np. dla każdego klienta, filmu czy pracownika osobno. W praktyce, dobry analityk SQL potrafi elastycznie przełączać się między różnymi technikami: używać podzapytań tam, gdzie są naturalne, a w innych przypadkach zastępować je złączeniami lub funkcjami okienkowymi, jeśli poprawi to wydajność i czytelność kodu.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską

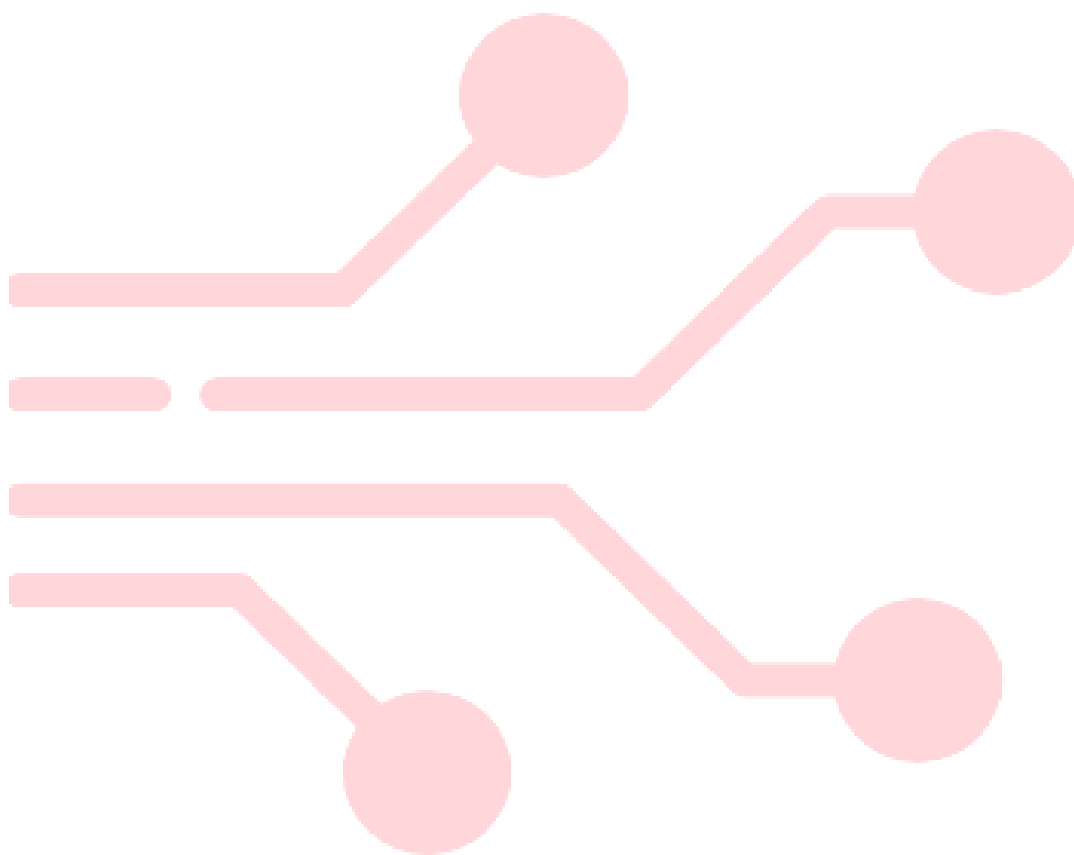


Literatura

Karwin, Bill, i in. Antywzorce języka SQL : jak uniknąć pułapek podczas programowania baz danych. Wyd. Helion, 2012.

<https://dev.mysql.com/doc/sakila/en/>

<https://www.w3schools.com/sql/>



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



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



Materiały do przedmiotu:

Bazy danych

Laboratorium nr 7:

Zarządzanie użytkownikami, przydzielanie uprawnień i zarządzanie dostępem do bazy danych. Kopia zapasowa, import i eksport danych

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	3
Ćwiczenia samodzielne	9
Pytania pomocnicze	10
Podsumowanie	10
Literatura	11



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Wprowadzenie, tematyka zajęć

Bezpieczeństwo i stabilność bazy danych zależy nie tylko od jej struktury, ale także od odpowiedniego zarządzania użytkownikami i ich uprawnieniami. Administrator musi kontrolować, kto i w jaki sposób uzyskuje dostęp do danych. Dodatkowo niezwykle istotne jest tworzenie kopii zapasowych oraz umiejętność przywracania danych w razie awarii. Dopelnieniem tych działań są mechanizmy importu i eksportu, które pozwalają przenosić dane między systemami.

Cel ćwiczenia/laboratorium

- Poznanie pojęcia użytkownika bazy danych.
- Tworzenie nowych użytkowników w bazie.
- Nadawanie uprawnień użytkownikom.
- Odbieranie uprawnień użytkownikom.
- Tworzenie ról i grup użytkowników.
- Zrozumienie znaczenia ról w zarządzaniu dostępem.
- Ćwiczenie z nadawania uprawnień SELECT, INSERT, UPDATE, DELETE.
- Poznanie pojęcia kopii zapasowej bazy danych.
- Tworzenie kopii zapasowych.
- Odtwarzanie danych z kopii zapasowej.
- Poznanie metod importu danych.
- Poznanie metod eksportu danych.
- Zrozumienie różnicy między importem a eksportem.
- Ćwiczenie z importu danych CSV do bazy.
- Omówienie dobrych praktyk w zarządzaniu użytkownikami.

Instrukcja laboratoryjna/ćwiczeniowa

Użytkownicy bazy danych – pojęcie, cel i zasady

Użytkownik bazy danych to tożsamość, pod którą serwer DBMS dopuszcza wykonywanie operacji. W MySQL/MariaDB tożsamość tę określa się parą nazwa–host, dzięki czemu „anna@localhost” i „anna@10.0.0.%” to dwa różne konta o potencjalnie zupełnie innych uprawnieniach i sposobach logowania. Sens istnienia użytkowników jest podwójny: po pierwsze rozdzielają oni dostęp do danych i funkcji



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



systemu między konkretne osoby, aplikacje i usługi; po drugie wymuszają kontrolę nad tym, kto co może zrobić oraz pozwalają później odtworzyć „kto, kiedy i jak” działał na bazie. Konto użytkownika obejmuje mechanizm uwierzytelniania (hasło, czasem certyfikat X.509), reguły bezpieczeństwa (wymóg SSL/TLS, polityka haseł, długość, rotacja, blokada po błędnych logowaniach), ograniczenia kontekstowe (z jakich hostów wolno się łączyć) oraz – co najważniejsze – zestaw przydzielonych uprawnień. W dojrzałych instalacjach rozróżnia się konta osobiste administratorów i analityków, które podlegają pełnej polityce haseł i audytowi, od kont usługowych, pod którymi łączy się kod aplikacji. Te drugie mają zwykle bardzo wąski zakres możliwości: tylko to, czego naprawdę potrzebuje dana usługa w swoim podstawowym scenariuszu, i nic więcej. Całość spina zasada najmniejszych uprawnień: tworząc konto, nadajemy minimalny niezbędny zestaw praw, ograniczamy miejsca logowania do konkretnych adresów lub podsięci, wymagamy połączeń szyfrowanych i cyklicznej zmiany haseł, a w razie potrzeby konto czasowo blokujemy zamiast usuwać, by zachować historię przypisań. Uprawnienia w MySQL/MariaDB można nadawać na wielu poziomach szczegółowości – globalnie dla całego serwera, dla wybranego schematu, dla pojedynczej tabeli, a nawet dla wskazanych kolumn, a także dla obiektów proceduralnych takich jak procedury i funkcje. Dzięki temu można precyzyjnie dopasować widoczność i zakres modyfikacji do roli danej osoby lub aplikacji: analityk dostaje wyłącznie prawo SELECT do widoków raportowych, operator importu tylko INSERT do wybranych tabel stagingowych, a administracja kadrowa UPDATE wyłącznie do kolumn nietajnych. Cykl życia konta powinien być formalny i przewidywalny: utworzenie z przeglądem ryzyk, bieżące przeglądy i redukcje nadanych praw, okresowa zmiana hasła lub jego unieważnienie, blokada przy odejściu użytkownika z organizacji oraz całkowite usunięcie dopiero po upewnieniu się, że nie ma już zależności technicznych i nie będzie konieczne odtwarzanie działań z przeszłości.

Role i delegacja uprawnień – jak porządkować dostęp

Role to abstrakcja grupująca uprawnienia w logiczne paczki, które następnie przypisuje się użytkownikom. Zamiast nadawać kilkanaście identycznych GRANT-ów każdej nowej osobie w zespole, definiuje się rolę „analityk_raportowy”, do której



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



trafiają prawa odczytu do właściwych schematów i widoków; przypisanie roli użytkownikowi natychmiast daje mu komplet przygotowanych uprawnień, a odebranie roli – natychmiast je cofa. Role wspierają porządek, skalowalność i zgodność z politykami bezpieczeństwa, a przy tym ułatwiają audyt, bo widać nie tylko „kto co ma”, ale też „z jakiego powodu” to ma. W systemach z rozbudowanym RBAC często buduje się hierarchię ról od ogólnych do wyspecjalizowanych, wyraźnie rozdzielając obowiązki administracyjne (tworzenie struktur, zmiany schematu), operacyjne (importy, wsadowe aktualizacje), analityczne (odczyt hurtowni, widoki) i serwisowe (zadania utrzymaniowe). Delegacja uprawnień musi być świadoma: w MySQL/MariaDB istnieje rozróżnienie pomiędzy samym posiadaniem uprawnienia a prawem do jego dalszego przekazywania innym. To drugie zapewnia atrybut GRANT OPTION przy klasycznych uprawnieniach i ADMIN OPTION przy rolach; stosuje się je oszczędnie, wyłącznie tam, gdzie faktycznie przewidziana jest decentralizacja administracji, ponieważ nieostrożne użycie prowadzi do cichej eskalacji dostępu i trudnych do uchwycenia łańcuchów dziedziczenia. Mechanizm ról dobrze łączy się z technikami „warstwy dostępu”, takimi jak widoki i procedury o bezpieczeństwie „definer”: użytkownik końcowy często nie musi mieć praw do tabel źródłowych, wystarczy mu rola dająca EXECUTE na procedurach realizujących zatwierdzone ścieżki modyfikacji albo SELECT na widokach maskujących dane wrażliwe. W praktyce wdrożeniowej warto ustalić katalog ról referencyjnych, ich domyślność po zalogowaniu oraz cykl regularnego przeglądu – rola powinna odzwierciedlać stanowisko i zakres obowiązków, a nie „historię projektu”. Jeśli organizacja korzysta z wielu środowisk (dev/test/prod), te same role powinny istnieć w każdym z nich, lecz różnić się zakresem i siłą uprawnień zgodnie z polityką separacji. Dzięki rolom i kontrolowanej delegacji przenosimy zarządzanie dostępem z poziomu pojedynczych, rozproszonych GRANT-ów na poziom zrozumiałych, biznesowo nazwanych pakietów, co czyni system bezpieczniejszym, łatwiejszym w utrzymaniu i bardziej odpornym na błędy ludzkie.

Audyt i przegląd użytkowników oraz uprawnień

Audyt użytkowników i okresowe przeglądy uprawnień są fundamentem bezpiecznego zarządzania bazą danych. Chodzi w nich o regularną weryfikację, kto ma dostęp do systemu, jakie prawa posiada i czy odpowiadają one jego aktualnym obowiązkom. W praktyce oznacza to nie tylko sprawdzenie listy aktywnych kont, ale także analizę



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



logów logowania, historii wykonywanych poleceń oraz zestawienie nadanych ról i przywilejów. Audyt powinien ujawniać sytuacje niepożądane: konta osierocone po odejściu pracownika, uprawnienia nadane „tymczasowo” i nigdy nieodebrane, czy też nadmiarowe prawa, które wykraczają poza faktyczne potrzeby użytkownika. Przegląd uprawnień musi być procesem cyklicznym, wpisany w politykę bezpieczeństwa organizacji – wykonywanym np. raz na kwartał lub przy każdej istotnej zmianie personalnej. Warto, aby w ramach audytu stosować zasadę czterech oczu: przeglądu dokonuje inna osoba niż ta, która nadała uprawnienia, co minimalizuje ryzyko błędów i nadużyć. Rezultatem audytu powinny być nie tylko raporty, ale także konkretne działania – odbieranie zbędnych praw, blokowanie nieaktywnych kont, a w skrajnych przypadkach eskalacja incydentów bezpieczeństwa. Systemy DBMS, takie jak MySQL czy MariaDB, umożliwiają prowadzenie logów binarnych i general logs, które w połączeniu z narzędziami monitorującymi mogą służyć jako źródło danych do audytu. Ostatecznym celem audytu jest zapewnienie zgodności z polityką bezpieczeństwa i regulacjami prawnymi, a także zbudowanie przejrzystości w tym, kto ma realny wpływ na dane.

Kopia zapasowa, oraz import i eksport danych.

Kopia zapasowa w MySQL to podstawowy mechanizm zabezpieczenia danych przed ich utratą, uszkodzeniem lub nieautoryzowaną modyfikacją. Jest to proces tworzenia pełnej lub częściowej kopii bazy danych w formie, która pozwala później odtworzyć jej stan w razie awarii sprzętu, błędu administratora czy ataku złośliwego oprogramowania. W świecie MySQL kopie zapasowe można wykonywać na kilka sposobów. Najczęściej stosuje się narzędzie mysqldump, które generuje plik zawierający skrypt SQL odtwarzający strukturę i zawartość bazy. Bardziej zaawansowane mechanizmy, takie jak mysqlpump czy narzędzia zewnętrzne (np. Percona XtraBackup), umożliwiają tworzenie kopii przy minimalnym zatrzymaniu pracy serwera oraz wykonywanie kopii przyrostowych lub różnicowych. Istnieją również fizyczne kopie zapasowe, polegające na skopiowaniu plików danych i logów transakcyjnych przechowywanych w katalogach serwera, co pozwala na wierne odtworzenie całego środowiska. Kopia zapasowa ma charakter systemowy – obejmuje nie tylko tabele i ich zawartość, ale również indeksy, widoki, procedury,



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



wyzwalacze, użytkowników i ich uprawnienia. W praktyce tworzenie backupów odbywa się cyklicznie, w określonych interwałach czasowych, często z wykorzystaniem harmonogramów i automatycznych zadań, a pliki kopii przechowywane są w bezpiecznej lokalizacji, nierzadko poza serwerem produkcyjnym.

Eksport i import danych to procesy, które w MySQL pełnią odmienną funkcję niż backup. Eksport polega na wyciągnięciu danych lub struktury z bazy do pliku w formacie nadającym się do przenoszenia – może to być klasyczny skrypt SQL, ale także formaty takie jak CSV czy JSON, szczególnie gdy dane mają być wykorzystane w innych programach lub systemach. W narzędziach graficznych, takich jak phpMyAdmin czy MySQL Workbench, eksport można wykonać wybierając odpowiednie opcje w menu, a w konsoli poleceniem `mysqldump`, np. `mysqldump -u root -p baza_danych > kopia.sql`. Import jest operacją odwrotną – polega na wczytaniu danych z przygotowanego pliku z powrotem do bazy. W przypadku plików SQL najczęściej odbywa się to komendą `mysql -u root -p baza_danych < kopia.sql` albo poprzez zakładkę „Import” w phpMyAdmin. Import może również dotyczyć plików CSV – wówczas dane ładowane są do konkretnej tabeli, przy czym często wymaga to wcześniejszego przygotowania struktury kolumn.

Różnica pomiędzy eksportem a kopią zapasową jest istotna. Eksport to narzędzie do migracji i przenoszenia danych – pozwala łatwo wyciągnąć tylko wybrane elementy, np. jedną tabelę czy część rekordów, i przenieść je do innego środowiska. Backup natomiast jest kompletnym zabezpieczeniem systemu, tworzonym regularnie i obejmującym całość struktury i danych w taki sposób, by baza mogła zostać w pełni odtworzona po awarii. Eksport zwykle wykonuje się ad hoc, w zależności od potrzeby, natomiast backup powinien być elementem stałej polityki bezpieczeństwa organizacji. Oba mechanizmy się uzupełniają: eksport jest wygodny przy pracy programistycznej i integracji, natomiast backup daje gwarancję bezpieczeństwa i ciągłości działania systemu.

Tworzenie użytkowników w phpMyAdmin

1. Zaloguj się do phpMyAdmin (<http://localhost/phpmyadmin>).



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



2. Wybierz zakładkę Konta użytkowników w menu górnym.
3. Kliknij przycisk Dodaj użytkownika.
4. W formularzu podaj:
 - Nazwę użytkownika (np. student1),
 - Host (najczęściej localhost),
 - Hasło (dwukrotnie).
5. Możesz zaznaczyć opcję Utwórz bazę danych o takiej samej nazwie i przyznaj wszystkie uprawnienia, jeśli chcesz, aby użytkownik od razu miał swoją bazę.
6. W sekcji Globalne uprawnienia wybierz, jakie prawa przydzielić (np. SELECT, INSERT, UPDATE).
7. Zatwierdź przyciskiem Wykonaj – użytkownik zostanie utworzony.

Tworzenie kopii zapasowej (backup)

Metoda graficzna (phpMyAdmin):

1. Zaloguj się do phpMyAdmin i wybierz z menu po lewej bazę danych do skopiowania.
2. Kliknij zakładkę Eksport.
3. Wybierz tryb Szybki i format SQL (najczęściej stosowany).
4. Kliknij Wykonaj – plik .sql zostanie pobrany na komputer i stanowi kopię bazy.

Metoda konsolowa (mysqldump):

```
mysqldump -u root -p nazwa_bazy > kopia.sql
```

Import danych

Metoda graficzna (phpMyAdmin):

1. Wybierz bazę danych, do której chcesz zaimportować dane.
2. Kliknij zakładkę Import.
3. Wskaż plik .sql (lub np. .csv) z danymi do załadowania.
4. Kliknij Wykonaj – dane zostaną wczytane do bazy.

Metoda konsolowa:

```
mysql -u root -p nazwa_bazy < kopia.sql
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Eksport danych

Metoda graficzna (phpMyAdmin):

1. Wybierz bazę lub tabelę, którą chcesz wyeksportować.
2. Kliknij zakładkę Eksport.
3. Wybierz tryb Szybki lub Dostosowany (jeśli chcesz wybrać konkretne tabele).
4. Kliknij Wykonaj – dane zostaną zapisane do pliku.

Metoda konsolowa:

```
mysqldump -u root -p nazwa_bazy > eksport.sql
```

Ćwiczenia samodzielne

1. Utwórz nowego użytkownika w bazie danych o nazwie student_lab7, który będzie mógł logować się wyłącznie z localhosta. Ustaw mu hasło i spróbuj zalogować się na jego konto przez phpMyAdmin lub terminal.
2. Nadaj użytkownikowi student_lab7 uprawnienia tylko do odczytu (SELECT) w jednej wybranej tabeli z bazy danych. Przetestuj, czy potrafi wykonywać zapytania SELECT, a następnie sprawdź, że nie może wstawić nowego wiersza ani usunąć istniejącego.
3. Utwórz użytkownika editor_lab7, który będzie miał prawo do wykonywania zapytań SELECT, INSERT i UPDATE w wybranej bazie danych. Przetestuj jego możliwości.
4. Zabierz wybrane uprawnienia użytkownikowi editor_lab7 (np. UPDATE) i ponownie spróbuj wykonać operację modyfikacji danych. Sprawdź, jak reaguje baza.
5. Utwórz rolę raportowanie, przypisz do niej prawo SELECT w całej bazie i nadaj tę rolę użytkownikowi student_lab7. Zaloguj się ponownie i sprawdź, czy dostęp do danych zmienił się zgodnie z oczekiwaniami.
6. Zablokuj konto jednego z użytkowników i spróbuj się na nie zalogować. Następnie odblokuj je i powtórz test.
7. Wykonaj kopię zapasową swojej bazy danych:
 - najpierw za pomocą narzędzia graficznego (phpMyAdmin – zakładka Eksport),
 - następnie w terminalu przy użyciu polecenia mysqldump.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



8. Usuń jedną tabelę z bazy danych, a następnie spróbuj ją odtworzyć z przygotowanej kopii zapasowej.
9. Zaimportuj plik CSV do nowej tabeli w bazie danych, wykorzystując narzędzie graficzne (Import w phpMyAdmin).
10. Przeprowadź import i eksport z terminala:
 - eksport: `mysqldump -u root -p moja_baza > backup.sql`
 - import: `mysql -u root -p moja_baza < backup.sql`
11. Sprawdź, czy dane odtworzyły się poprawnie.

Pytania pomocnicze

1. Czym jest użytkownik bazy danych?
2. Jak utworzyć nowego użytkownika?
3. Jakie są podstawowe uprawnienia w bazie danych?
4. Jak odebrać uprawnienia użytkownikowi?
5. Czym jest rola w bazie danych?
6. Jakie są zalety stosowania ról?
7. Podaj przykład nadania uprawnienia SELECT.
8. Podaj przykład nadania uprawnienia INSERT.
9. Podaj przykład nadania uprawnienia DELETE.
10. Czym jest kopia zapasowa bazy danych?
11. Jak utworzyć kopię zapasową?
12. Jak odtworzyć bazę z kopii?
13. Jakie formaty danych są wykorzystywane do importu?
14. Jakie są metody eksportu danych?
15. Jakie są najlepsze praktyki przy zarządzaniu dostępem?

Podsumowanie

Podczas laboratorium studenci zapoznali się z zagadnieniami zarządzania użytkownikami i kontrolą dostępu w systemach bazodanowych. Ćwiczenia pozwoliły zrozumieć, że poprawne definiowanie kont, ról i uprawnień jest fundamentem bezpieczeństwa danych oraz umożliwia wdrożenie zasady najmniejszych uprawnień. Studenci mieli okazję praktycznie utworzyć konta, przypisać im odpowiednie zestawy



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



praw oraz przetestować różnice w możliwościach poszczególnych użytkowników, co pokazało, jak w praktyce realizowana jest polityka bezpieczeństwa i audyt działań w bazie.

Drugim ważnym elementem zajęć była praca z kopiami zapasowymi oraz mechanizmami importu i eksportu danych. Studenci poznali różnicę między pełnym backupem bazy a eksportem danych do pliku oraz przeciwiczyli obie metody zarówno w narzędziach graficznych (phpMyAdmin), jak i w pracy z terminalem. Wnioskiem płynącym z tych zadań jest świadomość, że backup jest podstawowym narzędziem ochrony danych przed utratą, natomiast import i eksport pełnią głównie rolę w przenoszeniu i wymianie danych między systemami.

Laboratorium miało charakter nie tylko praktyczny, ale i refleksyjny – pokazało, że bezpieczeństwo i ciągłość działania bazy danych zależą w równym stopniu od właściwego zarządzania użytkownikami, jak i od skutecznych procedur tworzenia kopii zapasowych. Zdobyte doświadczenie ma bezpośrednie zastosowanie w praktyce administracyjnej i stanowi fundament pracy w realnych środowiskach, gdzie dane są strategicznym zasobem organizacji.

Literatura

1. Access Control and Account Management, <https://dev.mysql.com/doc/refman/8.4/en/access-control.html>, (data ostatniego dostępu 25.09.25)
2. Backup and Recovery, <https://dev.mysql.com/doc/refman/8.4/en/backup-and-recovery.html>, (data ostatniego dostępu 25.09.25)
3. Database Backup from MySQL, <https://www.geeksforgeeks.org/sql/database-backup-from-mysql/>, (data ostatniego dostępu 25.09.25)



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Materiały do przedmiotu:

Bazy danych

Laboratorium nr 8:

Dokumentowe systemy bazodanowe NoSQL

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	18
Pytania pomocnicze	20
Podsumowanie	21
Literatura	22



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Wprowadzenie, tematyka zajęć

Bazy dokumentowe stanowią jedną z kluczowych gałęzi systemów NoSQL, zaprojektowanych z myślą o elastycznym i skalowalnym przechowywaniu danych w formacie półstrukturalnym. W odróżnieniu od tradycyjnych relacyjnych baz danych, które wymagają sztywnego schematu tabel i relacji, systemy dokumentowe przechowują dane w postaci obiektów JSON (lub BSON w przypadku MongoDB), umożliwiając dynamiczne dopasowanie struktury danych do potrzeb aplikacji. Ich popularność wynika z rosnących wymagań współczesnych systemów informatycznych – zwłaszcza aplikacji webowych i mobilnych – które generują zróżnicowane dane o zmiennej strukturze i wymagają szybkiego dostępu w czasie rzeczywistym.

W ramach laboratorium studenci poznają koncepcję i architekturę baz dokumentowych, ze szczególnym uwzględnieniem systemu MongoDB jako jednego z najczęściej wykorzystywanych silników tego typu. Omówione zostaną podstawy modelowania danych w formie dokumentów, różnice między zagnieżdżaniem a odwołaniami oraz dobre praktyki projektowania kolekcji. W części praktycznej uczestnicy nauczą się wykonywać operacje CRUD (Create, Read, Update, Delete), tworzyć zapytania filtrujące i agregujące z wykorzystaniem potoków, a także analizować i przekształcać dane przy użyciu operatorów logicznych i obliczeniowych.

Dodatkowo zostaną przedstawione przykłady zastosowań baz dokumentowych w kontekście nowoczesnych rozwiązań, takich jak RethinkDB czy Firebase Firestore, które integrują przechowywanie danych z funkcjami przetwarzania w czasie rzeczywistym.

Celem zajęć jest zrozumienie zasad organizacji danych w modelu dokumentowym oraz wyrobienie umiejętności projektowania i optymalizacji struktur kolekcji w sposób zgodny z logiką aplikacji, a nie ograniczeniami relacyjnego schematu.

Cel ćwiczenia/laboratorium

1. Zapoznanie z koncepcją baz dokumentowych NoSQL oraz ich różnicami względem relacyjnych systemów baz danych.
2. Poznanie zasad przechowywania danych w formacie JSON/BSON oraz organizacji danych w kolekcjach.
3. Zrozumienie architektury i sposobu działania systemu MongoDB jako przykładowej bazy dokumentowej.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



4. Nabycie umiejętności wykonywania operacji CRUD (Create, Read, Update, Delete) na kolekcjach MongoDB.
5. Opanowanie technik filtrowania danych z wykorzystaniem operatorów porównania, logicznych i tablicowych.
6. Poznanie potoków agregacji (Aggregation Pipeline) i umiejętność ich stosowania w analizie danych.
7. Ćwiczenie praktycznego tworzenia zapytań z wykorzystaniem operatorów \$match, \$group, \$project, \$lookup, \$sort, \$limit, \$unwind.
8. Zrozumienie zasad projektowania kolekcji – zagnieżdżania (embedding) i odwołań (referencing) – oraz ich wpływu na wydajność i spójność danych.
9. Praktyczne zapoznanie z różnicami między denormalizacją a normalizacją danych w środowisku dokumentowym.
10. Poznanie dobrych praktyk w zakresie indeksowania, projektowania struktur danych i optymalizacji zapytań w MongoDB.
11. Zrozumienie zastosowań innych baz dokumentowych, takich jak RethinkDB i Firebase Firestore, oraz ich roli w aplikacjach czasu rzeczywistego i środowiskach chmurowych.
12. Ukształtowanie umiejętności wyboru odpowiedniego modelu danych w zależności od wymagań systemu i charakteru danych.

Instrukcja laboratoryjna/ćwiczeniowa

Dokumentowe bazy NoSQL

Bazy dokumentowe stanowią jedną z najważniejszych gałęzi systemów NoSQL i są projektowane w odpowiedzi na ograniczenia klasycznych relacyjnych baz danych w środowiskach o dużej zmienności struktury danych, skalowalności horyzontalnej oraz wymogach pracy w czasie rzeczywistym. Ich podstawową cechą jest przechowywanie informacji w postaci dokumentów, zazwyczaj w formacie JSON lub jego binarnej reprezentacji BSON, co pozwala na naturalne odwzorowanie danych aplikacyjnych w strukturze odpowiadającej obiektom znanym z programowania. Każdy dokument jest niezależną jednostką zawierającą komplet informacji, nie wymaga odwoływania się do innych tabel czy relacji, co eliminuje konieczność wykonywania złożonych operacji łączenia (JOIN) charakterystycznych dla systemów relacyjnych.

Model dokumentowy stanowi kompromis między elastycznością a spójnością. W odróżnieniu od relacyjnego podejścia, w którym dane są ściśle znormalizowane i powiązane, bazy dokumentowe kładą nacisk na swobodę kształtowania struktury



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



danych i szybki dostęp. Dzięki temu są fundamentem nowoczesnych systemów informatycznych, w których adaptacyjność, skalowalność i czas reakcji są ważniejsze niż bezwzględna kontrola nad relacjami między rekordami. Model dokumentowy zakłada elastyczność schematu (tzw. schema-less lub schema-flexible), dzięki czemu kolejne dokumenty w tej samej kolekcji mogą różnić się zestawem pól, typami wartości, a nawet głębokością zagnieżdżeń. Taka konstrukcja szczególnie dobrze sprawdza się w środowiskach, gdzie dane pochodzą z dynamicznych źródeł – np. z aplikacji webowych, czujników IoT, platform e-commerce czy systemów analitycznych, w których struktura informacji może ewoluować w czasie. Elastyczność modelu dokumentowego sprzyja szybkiemu prototypowaniu, integracji z nowymi źródłami danych i iteracyjnemu rozwojowi aplikacji bez konieczności migracji schematów.

W systemach dokumentowych, takich jak MongoDB, CouchDB czy RethinkDB, dane grupowane są w kolekcje będące odpowiednikami tabel, a zapytania wykonywane są przy użyciu języków inspirowanych JSON-em. W przypadku MongoDB operacje realizowane są w postaci zagnieżdżonych struktur opisujących warunki, projekcje i agregacje. Kluczową rolę odgrywa tu potok agregacji (aggregation pipeline), który pozwala przetwarzać dokumenty w wielu etapach: filtrować, grupować, sortować, przekształcać i analizować, w sposób przypominający przepływ danych w systemach przetwarzania strumieniowego. Taki model operowania na danych umożliwia wykonywanie złożonych analiz bez konieczności przenoszenia danych do zewnętrznych narzędzi, co znacząco redukuje opóźnienia i obciążenie sieci.

Bazy dokumentowe charakteryzują się także wysoką skalowalnością horyzontalną, co oznacza, że można je rozbudowywać przez dodawanie kolejnych węzłów (ang. sharding), a nie tylko poprzez zwiększanie mocy pojedynczej maszyny. Dzięki temu możliwe jest przetwarzanie dużych wolumenów danych w rozproszonych środowiskach, takich jak klastry serwerów czy chmury obliczeniowe. Wbudowane mechanizmy replikacji i rozproszonego zapisu zapewniają odporność na awarie oraz równoległy dostęp wielu użytkowników. W przeciwieństwie do tradycyjnych systemów transakcyjnych (ACID), wiele implementacji baz dokumentowych realizuje model spójności opartej na zasadzie eventual consistency, co oznacza, że dane w różnych węzłach systemu mogą być chwilowo niespójne, lecz z czasem osiągają zgodność. Takie podejście pozwala uzyskać większą dostępność i odporność na opóźnienia w środowiskach sieciowych o dużej skali.

Zaletą baz dokumentowych jest ich naturalne dopasowanie do współczesnych modeli aplikacyjnych, zwłaszcza w architekturze mikroserwisowej, gdzie każdy komponent może korzystać z własnego zestawu danych w odpowiednim dla siebie



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



formacie. Umożliwia to redukcję kosztów integracji oraz łatwiejsze odwzorowanie danych biznesowych na poziomie kodu aplikacji. Ponadto struktura dokumentowa pozwala przechowywać dane heterogeniczne – tekstowe, liczbowo-analityczne, a nawet binarne (np. obrazy w postaci osadzonych obiektów).

Jednocześnie bazy dokumentowe nie są pozbawione ograniczeń. Brak sztywnego schematu utrudnia egzekwowanie spójności danych oraz może prowadzić do rozbieżności strukturalnych przy niekontrolowanym rozwoju aplikacji. Operacje wymagające korelacji między różnymi kolekcjami (odpowiednikiem relacji w SQL) bywają bardziej kosztowne i trudniejsze do optymalizacji, ponieważ model dokumentowy preferuje denormalizację – czyli powielanie danych w różnych dokumentach – w celu poprawy szybkości odczytu. W efekcie aktualizacja powtarzających się informacji wymaga mechanizmów aplikacyjnych, a nie wbudowanych relacji.

Bazy dokumentowe są zatem wybierane tam, gdzie elastyczność i wydajność odczytu mają wyższy priorytet niż ścisła integralność danych. Sprawdzają się szczególnie w aplikacjach o dużym wolumenie nieustrukturyzowanych informacji, takich jak logi systemowe, dane telemetryczne, profile użytkowników, systemy rekomendacji, analityka w czasie rzeczywistym czy aplikacje mobilne i webowe, które muszą szybko reagować na zmieniające się dane. W porównaniu z klasycznymi bazami SQL, systemy dokumentowe wygrywają w środowiskach o dużej dynamice danych, rozproszonych architekturach i konieczności pracy w trybie online.

System MongoDB

MongoDB stanowi najbardziej rozpowszechnioną implementację modelu dokumentowego, łączącą prostotę pracy z danymi w formacie JSON z wysoką skalowalnością i elastycznością architektury. Jej projekt zakładał odejście od sztywnych schematów typowych dla SQL i umożliwienie dynamicznego modelowania danych w sposób zbliżony do ich rzeczywistej struktury w aplikacjach. Dokumenty w MongoDB są zapisywane w formacie BSON (Binary JSON), który stanowi rozszerzenie klasycznego JSON-a o typy binarne, daty, liczby całkowite o różnych rozmiarach, a nawet wskaźniki do innych dokumentów. Dzięki temu możliwe jest efektywne przechowywanie i przetwarzanie dużych wolumenów danych o zróżnicowanej naturze, od metadanych po wyniki analityczne i dane telemetryczne.

MongoDB należy rozpatrywać nie tylko jako bazę danych, ale jako kompletną platformę przetwarzania informacji, zdolną obsługiwać dane transakcyjne, analityczne i strumieniowe w jednym środowisku. Jej architektura łączy cechy



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



systemu OLTP z elementami analizy danych w stylu OLAP, dzięki czemu stanowi fundament wielu współczesnych systemów e-commerce, portali społecznościowych, aplikacji mobilnych czy systemów IoT. W praktyce jej przewaga nie wynika jedynie z wydajności, ale z adaptacyjności, z umiejętności dostosowania się do dynamicznego charakteru danych i aplikacji, które te dane generują.

Jednym z filarów MongoDB jest indeksowanie. System oferuje szeroki wachlarz typów indeksów, od klasycznych jednopolowych, przez złożone (compound), po indeksy tekstowe, geolokalizacyjne i wielokluczowe (multikey), które umożliwiają szybkie wyszukiwanie w tablicach. Dobrze zaprojektowany zestaw indeksów ma kluczowe znaczenie dla wydajności, zwłaszcza przy dużych kolekcjach. MongoDB posiada też mechanizm tzw. partial i sparse indexes, pozwalający na indeksowanie tylko wybranych dokumentów lub pól, co jest istotne przy danych o nieregularnej strukturze. Dzięki temu można ograniczyć rozmiar indeksów i poprawić efektywność operacji odczytu.

Kolejnym aspektem wyróżniającym MongoDB jest replikacja i wysoka dostępność. Baza operuje w modelu tzw. replica set, czyli zestawu serwerów, w którym jeden pełni rolę głównego węzła (primary), a pozostałe stanowią jego kopie zapasowe (secondaries). W przypadku awarii system automatycznie przeprowadza wybór nowego węzła głównego (automatic failover), co gwarantuje ciągłość pracy aplikacji. Replikacja może być również wykorzystywana do odciążania serwera głównego – zapytania tylko do odczytu mogą być kierowane do węzłów podrzędnych.

MongoDB wspiera również sharding, czyli partycjonowanie danych na wiele węzłów. Mechanizm ten rozdziela dokumenty między serwery na podstawie wybranego klucza shardującego. W praktyce pozwala to na niemal liniową skalowalność w poziomie – przy rosnącej liczbie danych wystarczy dodawać kolejne maszyny zamiast zwiększać moc jednej. Jednak wybór odpowiedniego klucza shardującego jest decyzją architektoniczną o dużym znaczeniu: zbyt nierównomierny rozkład wartości może prowadzić do tzw. hot spots, czyli przeciążenia wybranych shardów.

MongoDB zapewnia również mechanizmy transakcyjne, które wprowadzono od wersji 4.0. Choć pierwotnie operacje w MongoDB były atomowe tylko na poziomie pojedynczego dokumentu, obecnie można wykonywać transakcje obejmujące wiele dokumentów i kolekcji. Pozwala to na realizację złożonych scenariuszy biznesowych przy zachowaniu pełnej spójności danych. Warto jednak pamiętać, że transakcje te zwiększają koszty operacyjne i mogą ograniczyć przepustowość przy dużych wolumenach danych – dlatego stosuje się je głównie w sytuacjach wymagających ścisłej integralności logicznej.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Z punktu widzenia analizy danych MongoDB dysponuje rozbudowanym silnikiem agregacji opartym na koncepcji potoków (aggregation pipelines). Pozwala on łączyć wiele etapów przetwarzania: filtrowania, grupowania, sortowania, projekcji, mapowania i transformacji w jeden spójny proces. W praktyce pełni on rolę zbliżoną do podzapytań i funkcji agregujących w SQL, jednak jego elastyczność jest znacznie większa. Dodatkowo MongoDB oferuje integrację z językami analitycznymi i platformami przetwarzania danych, w tym z Pythonem (biblioteka pymongo), środowiskiem Spark czy frameworkami uczenia maszynowego.

Istotnym elementem architektury MongoDB jest mechanizm walidacji schematów. Choć baza pozostaje systemem schemaless, umożliwia definiowanie reguł kontrolujących poprawność dokumentów na poziomie kolekcji. Administrator może wymusić, aby określone pola istniały, miały konkretny typ lub spełniały warunki logiczne. Takie rozwiązanie łączy elastyczność NoSQL z częściową formalizacją znaną z relacyjnych baz danych, pozwalając ograniczyć ryzyko wprowadzenia niepoprawnych danych.

MongoDB wyróżnia się również bogatym ekosystemem narzędzi administracyjnych. Narzędzie mongosh stanowi interaktywną powłokę umożliwiającą pracę z dokumentami, natomiast MongoDB Compass oferuje graficzny interfejs do eksploracji danych, wizualizacji zapytań i analizy indeksów. W środowiskach produkcyjnych wykorzystuje się także platformę Atlas, zapewniającą zarządzanie klastrami w chmurze, automatyczne kopie zapasowe, monitorowanie i mechanizmy bezpieczeństwa oparte na szyfrowaniu w spoczynku i w transmisji.

Na szczególną uwagę zasługuje podejście MongoDB do modelowania danych. W przeciwieństwie do relacyjnych baz, gdzie nacisk kładzie się na normalizację, w MongoDB często preferuje się modelowanie z użyciem zagnieżdżonych dokumentów i tablic, co eliminuje konieczność łączenia wielu kolekcji. Wybór między zagnieżdżeniem (embedding) a referencją (referencing) stanowi kluczową decyzję projektową. Zagnieżdżenie poprawia wydajność odczytu, natomiast referencje zwiększają spójność danych przy dużej redundancji. W praktyce stosuje się podejścia hybrydowe, łączące obie strategie w zależności od charakteru aplikacji.

Z perspektywy bezpieczeństwa MongoDB zapewnia autoryzację opartą na rolach (RBAC) oraz mechanizmy uwierzytelniania oparte o SCRAM, LDAP lub X.509. System pozwala także na rejestrowanie operacji administracyjnych, co ułatwia audyt i detekcję nieautoryzowanych działań. W nowoczesnych wdrożeniach szczególną rolę odgrywa szyfrowanie komunikacji (TLS) oraz izolacja danych użytkowników w środowiskach wielodomenowych.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



System RethinkDB

RethinkDB jest dokumentową bazą danych o odmiennej filozofii działania niż MongoDB, została zaprojektowana z myślą o natychmiastowej reakcji na zmiany danych, oraz o integracji z aplikacjami czasu rzeczywistego. Jej kluczowym wyróżnikiem jest mechanizm realtime push, dzięki któremu aplikacja kliencka nie musi cyklicznie odpytywać bazy o nowe dane. Zamiast tego RethinkDB utrzymuje otwarte połączenie i przesyła zmiany do klienta w momencie ich wystąpienia. To podejście czyni ją szczególnie atrakcyjną w zastosowaniach takich jak systemy monitorowania, tablice wyników, aplikacje kolaboracyjne czy dashboardy biznesowe, gdzie kluczowa jest aktualność informacji. RethinkDB należy traktować jako bazę reaktywną, a nie wyłącznie dokumentową. Wygrywa w zastosowaniach, gdzie najważniejsza jest aktualność informacji, prostota tworzenia strumieni danych i bezpośrednia komunikacja z warstwą aplikacji. W systemach, które stawiają na niskie opóźnienia i ciągłą synchronizację między użytkownikami, RethinkDB zapewnia funkcjonalność, której w MongoDB i innych bazach NoSQL trzeba dopiero zbudować za pomocą dodatkowych komponentów.

W przeciwieństwie do MongoDB, które koncentruje się na prostocie zapisu i skalowalności, RethinkDB stawia nacisk na reaktywność i spójność logiczną danych. Oferuje natywny język zapytań ReQL, przypominający składnią języki funkcyjne, co pozwala tworzyć złożone operacje mapowania, filtrowania i agregacji bezpośrednio w logice bazy. Operacje te są deklaratywne i strumieniowe, dzięki czemu mogą być przetwarzane sekwencyjnie bez konieczności ładowania całego zbioru danych do pamięci. W odróżnieniu od MongoDB, które operuje na dokumentach w formacie BSON, RethinkDB przechowuje dane w czystym formacie JSON, co ułatwia ich integrację z aplikacjami webowymi i językami skryptowymi. Pod względem architektury RethinkDB jest bazą rozproszoną z natywnym wsparciem dla replikacji i sharding-u, jednak jej celem nie jest obsługa setek terabajtów danych, lecz zapewnienie niskich opóźnień w komunikacji między warstwą danych a aplikacją kliencką. Silnik bazy implementuje transakcje oparte na zasadzie write-durability i zapewnia spójność odczytu w ramach pojedynczej repliki, co pozwala uniknąć typowych problemów eventual consistency spotykanych w innych systemach NoSQL.

Z punktu widzenia projektanta systemów, RethinkDB zyskuje przewagę tam, gdzie dane muszą być natychmiast propagowane do użytkownika, np. w czatach, systemach powiadomień, giełdach danych, grach online, analityce w czasie rzeczywistym czy panelach administracyjnych, które dynamicznie odświeżają wyniki. MongoDB, choć szybsze przy klasycznych operacjach odczytu i zapisu, wymaga dodatkowych narzędzi (np. Change Streams) do osiągnięcia podobnej



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



funkcjonalności, podczas gdy RethinkDB oferuje ją natywnie. Zaletą RethinkDB jest również przejrzysty model replikacji i kontroli klastrów – każdy węzeł może pełnić różne role (primary, secondary), a system automatycznie równoważy obciążenie i synchronizuje dane między replikami. Interfejs administracyjny bazy, dostępny w przeglądarce, pozwala wizualizować przepływ danych w czasie rzeczywistym, analizować zapytania i monitorować zużycie zasobów, co znacząco ułatwia utrzymanie środowisk developerskich i produkcyjnych. W kontekście ograniczeń należy zauważyć, że RethinkDB nie dorównuje MongoDB pod względem popularności, ekosystemu i liczby integracji. Projekt, choć otwartoźródłowy, przez pewien czas był w stagnacji po zakończeniu komercyjnego wsparcia, a jego rozwój kontynuowany jest obecnie przez społeczność. Nie stanowi jednak alternatywy uniwersalnej, jest to baza specjalizowana, przeznaczona do systemów wymagających natychmiastowej propagacji danych i prostoty programistycznej, a nie do hurtowni danych czy długoterminowego przechowywania miliardów rekordów.

Usługa Cloud Firestore

Cloud Firestore, będąca częścią ekosystemu Firebase rozwijanego przez Google, reprezentuje nowoczesne podejście do baz dokumentowych zaprojektowanych z myślą o środowiskach rozproszonych, mobilnych i webowych. W odróżnieniu od klasycznych baz NoSQL, które koncentrują się na lokalnej wydajności i elastycznym modelu danych, Firestore jest bazą natywnie chmurową, zarządzaną, skalowaną i replikowaną automatycznie w infrastrukturze Google Cloud, co eliminuje potrzebę administracji serwerami i konfiguracji klastrów. Pod względem modelu danych Firestore wykorzystuje hierarchiczną strukturę kolekcji i dokumentów, w której każda kolekcja może zawierać wiele dokumentów, a każdy dokument może mieć swoje podkolekcje. To rozwiązanie łączy prostotę modelu JSON znaną z MongoDB z logiczną strukturą przypominającą system plików. Dzięki temu Firestore umożliwia intuicyjne odwzorowanie hierarchii danych aplikacji, np. użytkownicy → zamówienia → szczegóły pozycji, bez potrzeby stosowania relacji lub dodatkowych warstw mapujących.

Jednym z kluczowych wyróżników Firestore jest jego asynchroniczny model synchronizacji w czasie rzeczywistym. Dane zapisane w bazie są natychmiast propagowane do wszystkich połączonych klientów, zarówno aplikacji webowych, jak i mobilnych. Klient SDK utrzymuje lokalny bufor (cache) danych i automatycznie synchronizuje go z chmurą przy każdej zmianie po stronie serwera lub klienta. Ten mechanizm sprawia, że aplikacje oparte na Firestore działają nawet w trybie offline, a po przywróceniu połączenia zmiany są automatycznie scalane (tzw. offline persistence with conflict resolution). W praktyce pozwala to budować aplikacje o



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



zachowaniu „real-time” bez konieczności implementacji własnych serwerów push czy kolejek komunikacyjnych.

Firestore działa w oparciu o rozproszony model spójności o wysokiej dostępności (multi-region replication). Dane przechowywane są równolegle w kilku centrach danych, co gwarantuje odporność na awarie i minimalne opóźnienia globalne. System zapewnia silną spójność odczytów (strong consistency) w obrębie dokumentu, co odróżnia go od wielu innych baz NoSQL, które oferują jedynie spójność ostateczną (eventual consistency). Dla aplikacji użytkowych oznacza to, że dane widoczne po zapisie są natychmiast aktualne, bez ryzyka odczytu niesynchronizowanej wersji. Z punktu widzenia programisty Firestore integruje się bezpośrednio z aplikacjami poprzez API SDK dostępne w popularnych językach (JavaScript, Python, Kotlin, Swift, Go). Zapytania mają charakter deklaratywny i opierają się na filtrach, zakresach i indeksach, wszystkie operacje są indeksowane automatycznie, a dla zapytań złożonych użytkownik może definiować indeksy złożone (composite indexes). W przeciwieństwie do MongoDB, Firestore nie pozwala na dowolne zapytania agregacyjne czy pełne przetwarzanie danych po stronie serwera, jednak jego siłą jest prostota, deterministyczne działanie i minimalne opóźnienia w komunikacji. W kontekście bezpieczeństwa i autoryzacji Firestore wykorzystuje model Firebase Security Rules, który łączy warstwę danych z logiką uwierzytelniania. Reguły bezpieczeństwa definiowane są w języku deklaratywnym i opisują, kto może odczytywać, zapisywać lub modyfikować dane na poziomie kolekcji i dokumentów. Można w nich odwoływać się do kontekstu użytkownika (np. `request.auth.uid`) oraz wartości samych dokumentów, co pozwala budować złożone, dynamiczne polityki bezpieczeństwa bez pośrednictwa backendu. To rozwiązanie czyni Firestore szczególnie atrakcyjnym dla projektów typu serverless, gdzie logika biznesowa znajduje się po stronie klienta, a kontrola dostępu w samej bazie.

Nie jest to jednak baza uniwersalna. Firestore ma pewne ograniczenia, takie jak limit głębokości hierarchii, brak klasycznych transakcji wieloetapowych w stylu SQL oraz koszty zależne od liczby operacji, a nie od rozmiaru danych. W projektach wymagających rozbudowanej analityki, przetwarzania masowych zapytań lub integracji z danymi relacyjnymi, lepiej sprawdzają się narzędzia takie jak MongoDB.

Projektowanie kolekcji w MongoDB



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Projektowanie kolekcji w MongoDB jest jednym z kluczowych etapów budowy efektywnego systemu opartego na bazie dokumentowej. W przeciwieństwie do relacyjnych baz danych, w których struktura jest sztywno określona przez schemat tabel, w MongoDB projektowanie polega na doborze odpowiedniego modelu dokumentów i relacji między nimi — czyli świadomym wyborze pomiędzy zagnieżdżaniem danych (embedding) a odwołaniami (referencing). Oba podejścia mogą współistnieć, a właściwe ich zastosowanie decyduje o wydajności, skalowalności i prostocie utrzymania systemu.

Model zagnieżdżania (Embedding)

Zagnieżdżanie polega na umieszczaniu powiązanych danych w jednym dokumencie JSON. Przykładowo, zamiast tworzyć osobną kolekcję dla adresów użytkowników, można przechowywać je wewnątrz dokumentu użytkownika:

```
{  
  _id: 1,  
  name: "Anna Kowalska",  
  email: "anna.kowalska@example.com",  
  address: {  
    city: "Kraków",  
    street: "Długa 5",  
    postalCode: "31-123"  
  }  
}
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Taki model zapewnia lokalność danych – wszystkie informacje potrzebne do obsługi użytkownika znajdują się w jednym miejscu, dzięki czemu odczyt jest szybki, a zapytania nie wymagają łączenia wielu kolekcji. To rozwiązanie idealne w przypadkach, gdy dane są ściśle powiązane semantycznie i często pobierane razem. Zagnieżdżanie minimalizuje liczbę zapytań do bazy, co znacząco poprawia wydajność w aplikacjach odczytowo-intensywnych. Jednak zbyt głębokie zagnieżdżenie może prowadzić do nadmiernego rozrostu dokumentów, co utrudnia ich aktualizację. MongoDB ma ograniczenie rozmiaru pojedynczego dokumentu (16 MB), dlatego zagnieżdżanie dużych struktur, takich jak historia transakcji lub setki komentarzy, nie jest zalecane. Aktualizacje w strukturze zagnieżdżonej są kosztowne – każda zmiana wymaga modyfikacji całego dokumentu, co może generować blokady i spadek wydajności przy dużym współczynniku zapisu. W takich przypadkach bardziej elastyczne okazuje się podejście oparte na odwołaniach.

Model odwołań (Referencing)

Odwołania (referencje) polegają na łączeniu dokumentów z różnych kolekcji za pomocą identyfikatorów, zamiast umieszczania danych bezpośrednio w sobie. Na przykład użytkownik może mieć pole `address_id`, które wskazuje na dokument w kolekcji `addresses`:

```
{
  _id: 1,
  name: "Anna Kowalska",
  email: "anna.kowalska@example.com",
  address_id: ObjectId("64bce2c9f1a3a0d9c2a3d2e1")
}
```

a odpowiadający adres może znajdować się w osobnej kolekcji:

```
{
  _id: ObjectId("64bce2c9f1a3a0d9c2a3d2e1"),
  city: "Kraków",
}
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



street: "Długa 5",

postalCode: "31-123"

}

To podejście przypomina relacje w klasycznym modelu SQL, ale bez sztywnego wymuszania integralności referencyjnej. Odwołania są korzystne, gdy dane są współdzielone między wieloma dokumentami lub gdy ich rozmiar i liczba rosną niezależnie. Ułatwiają aktualizacje – zmiana jednego adresu automatycznie odnosi się do wszystkich użytkowników, którzy się do niego odwołują.

Z drugiej strony, model referencyjny wymaga łączenia danych na poziomie aplikacji lub potoków agregacji, np. za pomocą operatora \$lookup. Wpływa to na wydajność, ponieważ każde łączenie wymaga dodatkowego przetwarzania po stronie serwera i może generować narzut przy dużej liczbie dokumentów. Dlatego nie powinno się nadużywać tego podejścia w przypadkach, gdy dane są zawsze pobierane razem.

Zagnieżdżanie kontra odwołania — wybór strategii

Decyzja, czy użyć zagnieżdżania, czy odwołań, zależy od charakteru powiązań między danymi.

Zagnieżdżanie preferuje się, gdy:

- dane są ściśle powiązane i współwystępują w zapytaniach,
- rozmiar dokumentu pozostaje niewielki,
- dane są niemodyfikowalne lub rzadko aktualizowane,
- priorytetem jest szybkość odczytu.

Odwołania stosuje się, gdy:

- dane są współdzielone między wieloma dokumentami,
- ich liczba lub rozmiar jest dynamiczny,
- występuje potrzeba aktualizacji niezależnych fragmentów,
- zależy nam na modularności i skalowalności danych.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Często w praktyce wykorzystuje się podejście hybrydowe, w którym dane główne są zagnieżdżone, a bardziej szczegółowe, często zmieniające się informacje – odwoływane. Przykładowo, w systemie e-commerce dokument zamówienia może zawierać wbudowaną listę produktów (z nazwą i ceną), ale odwoływać się do osobnej kolekcji customers, która przechowuje pełne dane klienta. Pozwala to zachować balans między wydajnością odczytu a elastycznością modyfikacji.

Dobre praktyki projektowania kolekcji

Efektywne projektowanie w MongoDB wymaga myślenia od strony zapytań, a nie struktury danych. Oznacza to, że najpierw należy przeanalizować, jakie operacje będą wykonywane najczęściej, a następnie dostosować model dokumentów do tych przypadków użycia. MongoDB premiuje modelowanie pod kątem odczytu, nawet kosztem pewnej redundancji danych — ponieważ odczyty są znacznie częstsze niż zapisy.

Kluczowe zalecenia obejmują:

- Projektuj pod kątem zapytań, nie odwrotnie. Dane, które są często pobierane razem, powinny znajdować się w jednym dokumencie.
- Unikaj nadmiernej głębokości zagnieżdżenia. MongoDB ma limit 100 poziomów zagnieżdżenia, ale w praktyce zaleca się nie więcej niż kilka.
- Kontroluj rozmiar dokumentów. Duże dokumenty mogą powodować fragmentację i spowolnienia operacji I/O.
- Stosuj indeksy z rozwagą. Tworzenie indeksów na polach często używanych w zapytaniach znacznie przyspiesza ich działanie, ale zbyt wiele indeksów obniża wydajność zapisów.
- Denormalizuj dane tam, gdzie to uzasadnione. MongoDB dobrze znosi powielanie informacji, o ile przyspiesza to odczyt.
- Używaj agregacji zamiast nadmiarowych zapytań. Potoki agregacyjne mogą zastąpić złożone operacje łączenia i przetwarzania w warstwie aplikacji.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



- Zachowuj spójność logiczną. W przypadku danych krytycznych warto rozważyć użycie transakcji ACID, które MongoDB obsługuje w ramach dokumentów i wielu kolekcji w tym samym klastrze.

W ujęciu inżynierskim projektowanie kolekcji w MongoDB to proces równoważenia trzech aspektów: wydajności, spójności i prostoty zapytań. Zagnieżdżanie zapewnia szybkość i lokalność, ale ogranicza elastyczność; odwołania oferują modularność i skalowalność, kosztem większego narzutu zapytań. W praktyce najlepsze rozwiązania powstają wtedy, gdy architekt traktuje model danych nie jako strukturę statyczną, lecz jako element dostosowany do logiki systemu i jego dynamiki – elastyczny, czytelny i skalowalny w czasie. MongoDB umożliwia pełną swobodę w tym zakresie, dlatego umiejętność właściwego wyboru strategii modelowania stanowi o jakości i długowieczności całego projektu.

Zapytania w MongoDB

W MongoDB zapytania stanowią jeden z najważniejszych elementów pracy z danymi i jednocześnie klucz do pełnego wykorzystania potencjału tego systemu. W przeciwieństwie do języka SQL, MongoDB operuje w sposób dokumentowy, co oznacza, że zapytania są formułowane w postaci obiektów JSON opisujących warunki, operatory i projekcje danych. Dzięki temu praca z danymi staje się bardziej naturalna z punktu widzenia programisty, zwłaszcza w aplikacjach opartych na strukturach zagnieżdżonych i dynamicznych.

Podstawą każdego zapytania jest polecenie `find()`, które pełni rolę odpowiednika `SELECT` z języka SQL. Jego składnia pozwala jednocześnie określić filtr, projekcję i sposób sortowania. Na przykład zapytanie:

```
db.students.find({ age: { $gt: 20 } }, { name: 1, age: 1, _id: 0 }).sort({ age: -1 })
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



zwróci wszystkie dokumenty, w których pole age jest większe niż 20, wyświetlając jedynie imię i wiek, posortowane malejąco po wieku. W tym miejscu warto zauważyć, że MongoDB operuje na operatorach porównania (\$eq, \$ne, \$gt, \$lt, \$gte, \$lte), operatorach logicznych (\$and, \$or, \$not, \$nor) oraz operatorach tablicowych (\$in, \$nin, \$all, \$elemMatch), co daje bardzo dużą swobodę w formułowaniu złożonych warunków filtrujących. Z kolei projekcja określa, które pola mają zostać zwrócone, co w praktyce ogranicza rozmiar przesyłanych danych i przyspiesza odczyt.

MongoDB udostępnia również zaawansowany mechanizm przetwarzania danych w potokach, zwany Aggregation Pipeline. To potężne narzędzie pozwala wykonywać wieloetapowe operacje na danych, w których wynik jednego etapu staje się wejściem dla kolejnego, w sposób analogiczny do potoków w systemach uniksowych. Potok agregacji składa się z szeregu etapów zdefiniowanych za pomocą operatorów takich jak \$match, \$group, \$project, \$sort, \$limit, \$skip, \$unwind czy \$lookup. Każdy etap realizuje określone zadanie — filtrowanie, grupowanie, transformację, łączenie danych lub tworzenie nowych pól.

Etap \$match odpowiada za filtrowanie danych na wzór klauzuli WHERE w SQL. Przykładowo:

```
{ $match: { status: "active" } }
```

wybiera tylko dokumenty o określonym stanie. Kolejnym etapem może być \$group, który pełni funkcję zbliżoną do GROUP BY. Umożliwia grupowanie dokumentów według określonego pola i obliczanie agregatów, takich jak suma, średnia czy liczność. Przykładowa definicja:

```
{ $group: { _id: "$category", total: { $sum: "$price" }, avgPrice: { $avg: "$price" } } }
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



pozwała uzyskać łączną i średnią wartość produktów w każdej kategorii. Etap \$project z kolei służy do przekształcania struktury danych, wybierania konkretnych pól lub tworzenia nowych, np. poprzez wyrażenia arytmetyczne lub logiczne. W połączeniu z \$sort, \$limit i \$skip można kontrolować kolejność i liczbę zwracanych dokumentów, co jest szczególnie przydatne w paginacji wyników.

Jednym z najbardziej interesujących elementów agregacji jest operator \$unwind, który rozбивa tablice na pojedyncze dokumenty, co pozwala analizować dane złożone. Z kolei \$lookup pełni rolę łączenia kolekcji (odpowiednik JOIN), umożliwiając pobieranie powiązanych informacji bez konieczności ręcznego wykonywania wielu zapytań. Przykładowe użycie:

```
{  
  $lookup: {  
    from: "departments",  
    localField: "dept_id",  
    foreignField: "_id",  
    as: "department_info"  
  }  
}
```

łącząc kolekcję employees z departments na podstawie identyfikatora działu. MongoDB wspiera także tzw. pipeline operators w ramach \$lookup, które umożliwiają wprowadzenie dodatkowych etapów filtrujących lub projekcyjnych już podczas łączenia danych, co znacząco zwiększa elastyczność zapytań.

W potokach agregacji można wykorzystywać również wyrażenia obliczeniowe i warunki logiczne, takie jak \$cond, \$ifNull, \$add, \$multiply, \$concat, \$toUpper czy



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



\$dateToString. Dzięki temu możliwe jest dynamiczne tworzenie pól, konwersja typów i przekształcanie wartości bezpośrednio po stronie serwera, bez potrzeby ich obróbki w aplikacji. MongoDB traktuje potoki agregacyjne jako mechanizm przetwarzania danych bliski modelowi map-reduce, lecz znacznie bardziej czytelny i wydajny.

Istotnym uzupełnieniem potoków są operatory wyszukiwania tekstowego (\$text, \$search), umożliwiające tworzenie indeksów pełnotekstowych i zapytań opartych na dopasowaniu słów kluczowych. W połączeniu z operatorami wyrażeń regularnych (\$regex) pozwalają one budować wyszukiwarki i systemy rekomendacji. Dodatkowo MongoDB umożliwia stosowanie filtrów geolokalizacyjnych (\$geoWithin, \$near, \$geoIntersects) do zapytań przestrzennych, wykorzystywanych w aplikacjach mapowych i IoT.

W praktyce zapytania w MongoDB można traktować jako kombinację trzech głównych komponentów: filtrowania danych (\$match, warunki w find()), projekcji i transformacji (\$project, find(..., projection)), oraz agregacji i obliczeń (\$group, \$sum, \$avg, \$count). Ich składanie w potoki tworzy logiczne, deterministyczne i w pełni równoległe przepływy danych, które są wykonywane w obrębie serwera, minimalizując transfer i obciążenie po stronie klienta.

Ćwiczenia samodzielne

Do wykonania zadań użyj kolekcji z datasetu sample_training dostępnych pod linkiem:

https://github.com/neelabalan/mongodb-sample-dataset/tree/main/sample_training

Analiza kolekcji stories

1. Wyświetl tytuły dokumentów, w których liczba komentarzy przekracza 100, a status to „popular”.
2. Policz średnią wartość pola diggs wśród wpisów, w których media to „news”.
3. Dla każdej wartości pola container.name oblicz sumę komentarzy i posortuj wynik malejąco.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



4. Wskaż użytkownika (user.name), który ma najwyższą sumaryczną liczbę profileviews.
5. Zapisz wynik zapytania z pkt. 7 do nowej kolekcji top_users przy pomocy operatora \$out.
6. Użyj wyrażenia regularnego, aby znaleźć dokumenty, w których user.name zaczyna się literą „M” i posiada więcej niż 50 komentarzy.

Analiza kolekcji sales

7. Znajdź wszystkie transakcje, w których przynajmniej jeden produkt zawiera w polu tags wartość „school”.
8. Wyświetl adresy e-mail klientów oraz datę zakupu w formacie DD-MM-YYYY.
9. Oblicz średnią wartość zakupu (total), grupując po polu storeLocation.
10. Znajdź klienta, który dokonał największej liczby transakcji metodą „Online”.
11. Dla każdego sklepu (storeLocation) oblicz sumaryczną sprzedaż tylko dla klientów z oceną customer.satisfaction > 4.
12. Policz, ile transakcji zawierało dokładnie cztery elementy w tablicy items.

Analiza kolekcji trips

13. Oblicz średni czas trwania wycieczek (tripduration) dla każdej stacji startowej i posortuj wynik malejąco.
14. Znajdź dni tygodnia z największą liczbą rozpoczętych podróży.
15. Policz liczbę podróży w zależności od typu użytkownika (usertype).
16. Oblicz średnią długość podróży tylko dla subskrybentów (usertype: "Subscriber").
17. Znajdź wszystkie wycieczki, które rozpoczęły się w stacjach zawierających w nazwie słowo „Centre”.
18. Oblicz średnią długość podróży w zależności od godziny rozpoczęcia (\$hour: "\$start time").
19. Określ, która stacja startowa jest najczęściej używana wśród użytkowników płci „2”.

Agregacja między kolekcjami

20. Wykorzystując \$lookup, połącz dane z kolekcji sales i customers (jeśli występują) na podstawie adresu e-mail klienta.
21. Utwórz agregację, która dla każdego klienta pokaże liczbę transakcji oraz łączną wartość zakupów.
22. Zapisz wynik do kolekcji customer_stats i wyświetl pięć pierwszych rekordów.

Zadania rozszerzone



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



23. Wykorzystaj operator \$facet, aby jednocześnie obliczyć:
24. średnią liczbę komentarzy w stories,
25. średnią długość wycieczki w trips,
26. średnią wartość sprzedaży w sales.
27. Za pomocą \$merge zapisz wynik z pkt. 26 do nowej kolekcji summary_metrics.
28. Zastosuj \$bucket w kolekcji sales, aby pogrupować transakcje według wartości total w przedziałach: <50, 50–200, >200.
29. Utwórz zapytanie wykorzystujące \$regex i \$expr, które wyświetli wszystkie produkty (items.name), których nazwa zawiera słowo „Book” i których ilość (items.quantity) przekracza 5.
30. Przygotuj raport łączący dane z sales i stores, w którym pokażesz lokalizacje z najwyższym średnim wynikiem customer.satisfaction.

Pytania pomocnicze

1. Czym różni się model danych w bazach dokumentowych od modelu relacyjnego?
2. W jakim formacie przechowywane są dane w MongoDB i czym jest BSON?
3. Na czym polega różnica między kolekcją a tabelą w klasycznej bazie SQL?
4. Jakie operacje składają się na cykl CRUD i jak są realizowane w MongoDB?
5. Jak działa polecenie find() i w jaki sposób można stosować filtry z operatorami \$gt, \$lt, \$and, \$or?
6. Do czego służy potok agregacji i jakie etapy może obejmować (\$match, \$group, \$project, \$lookup, \$sort)?
7. Jakie są zalety stosowania operatora \$lookup w porównaniu z tradycyjnym zagnieżdżaniem danych?
8. Czym różni się zagnieżdżanie danych (embedding) od odwołań (referencing)?
9. Kiedy warto stosować zagnieżdżanie, a kiedy lepiej użyć odwołań?
10. Jakie są dobre praktyki projektowania kolekcji w MongoDB pod kątem wydajności i skalowalności?
11. Jakie ograniczenia dotyczą rozmiaru dokumentów w MongoDB i jakie ma to znaczenie przy projektowaniu schematów danych?



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



12. Czym jest denormalizacja danych i dlaczego jest często stosowana w bazach NoSQL?
13. Jak działają indeksy w MongoDB i w jaki sposób wpływają na szybkość zapytań?
14. W jakich przypadkach warto sięgnąć po inne bazy dokumentowe, takie jak RethinkDB lub Firebase Firestore?

Podsumowanie

Zajęcia poświęcone bazom dokumentowym pozwalają zrozumieć nowoczesne podejście do przechowywania i przetwarzania danych, w którym elastyczność i skalowalność stają się równie istotne jak spójność i wydajność. Praca z MongoDB i pokrewnymi systemami pokazała, że dokumentowy model danych umożliwia projektowanie struktur bliższych rzeczywistej logice aplikacji, bez konieczności dopasowywania się do sztywnych tabel i relacji. Studenci zapoznali się z praktycznym wykorzystaniem zapytań, filtrów, operatorów agregujących oraz potoków przetwarzania danych, które w MongoDB pełnią rolę równoważną złożonym zapytaniom SQL, lecz w sposób znacznie bardziej elastyczny i naturalny dla danych półstrukturalnych.

Zrozumienie zasad zagnieżdżania, odwołań i denormalizacji danych pozwala świadomie projektować kolekcje, równoważąc wydajność, prostotę zapytań i skalowalność systemu. Dzięki temu uczestnicy potrafią dobrać właściwe strategie modelowania danych do różnych scenariuszy – od prostych aplikacji użytkowych po złożone systemy rozproszone.

W efekcie zdobyta wiedza stanowi solidną podstawę do dalszej pracy z systemami NoSQL, projektowania nowoczesnych architektur danych oraz integracji baz dokumentowych z aplikacjami w środowiskach chmurowych i mikroservisowych.

Literatura

1. How to Create Database and Collection in MongoDB, <https://www.geeksforgeeks.org/mongodb/how-to-create-database-collection-in-mongodb/>, (data ostatniego dostępu 25.09.25)
2. Getting Started with MongoDB, <https://www.mongodb.com/docs/manual/tutorial/getting-started/>, (data ostatniego dostępu 25.09.25)



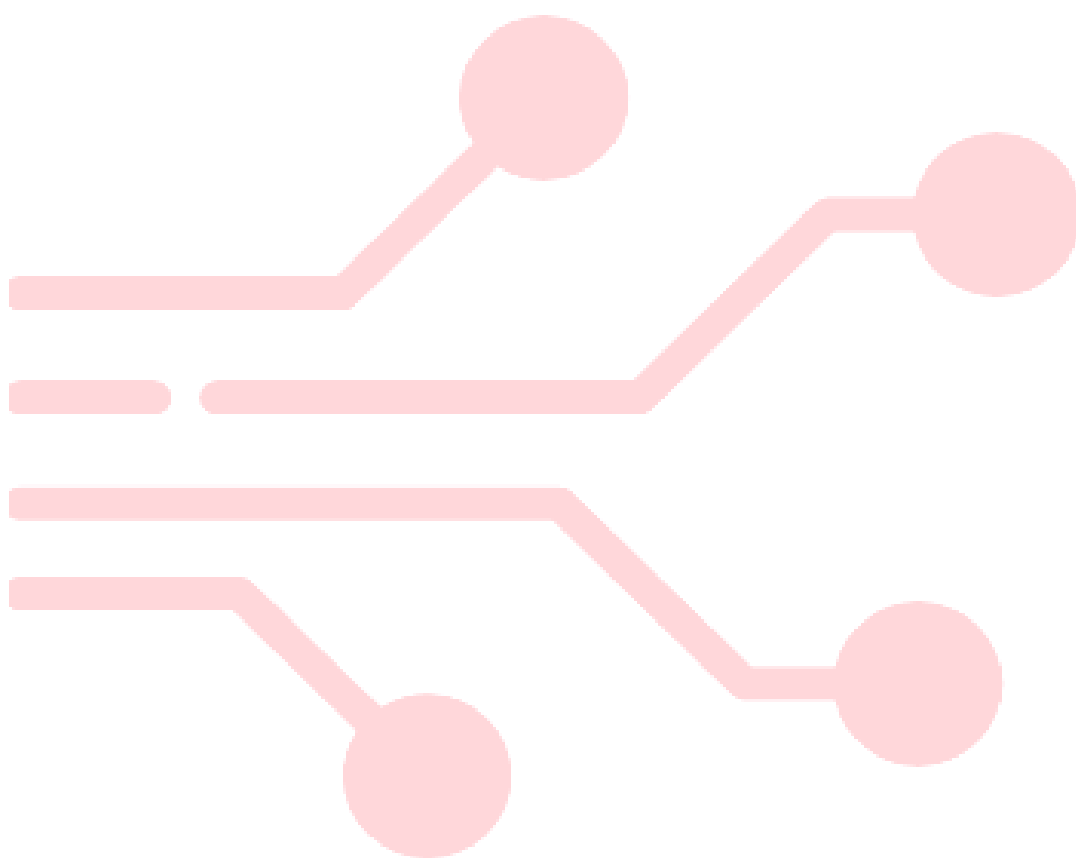
Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską





Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską

