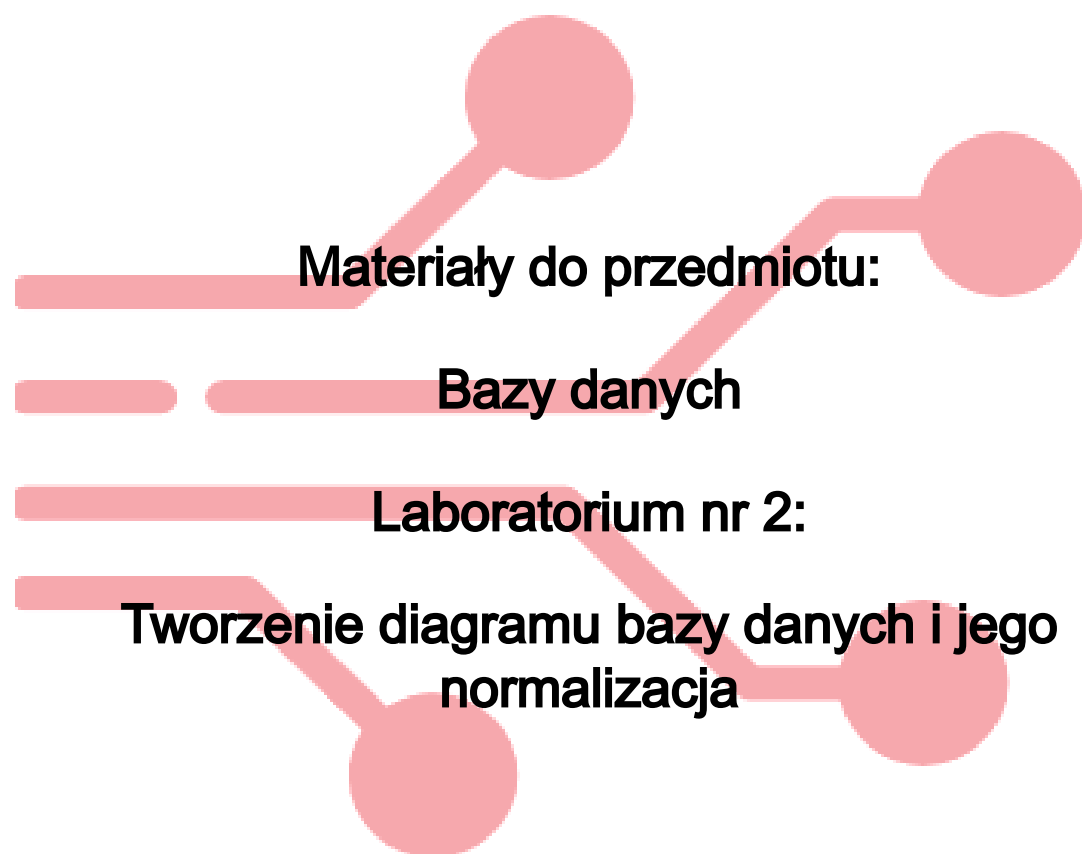




Materiały do przedmiotu:

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 węzłem 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 $\text{grade} \geq 60$, wtedy $\text{is_passing} = \text{TRUE}$, 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 ulokowanie 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

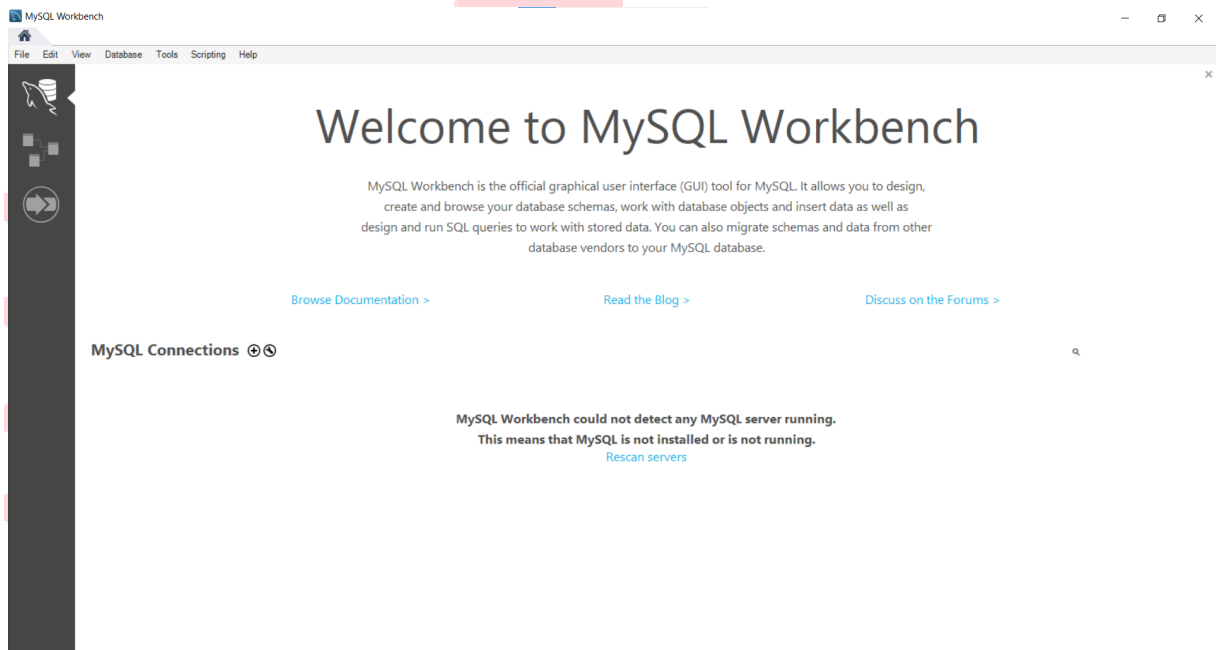
Dofinansowane przez
Unię Europejską



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



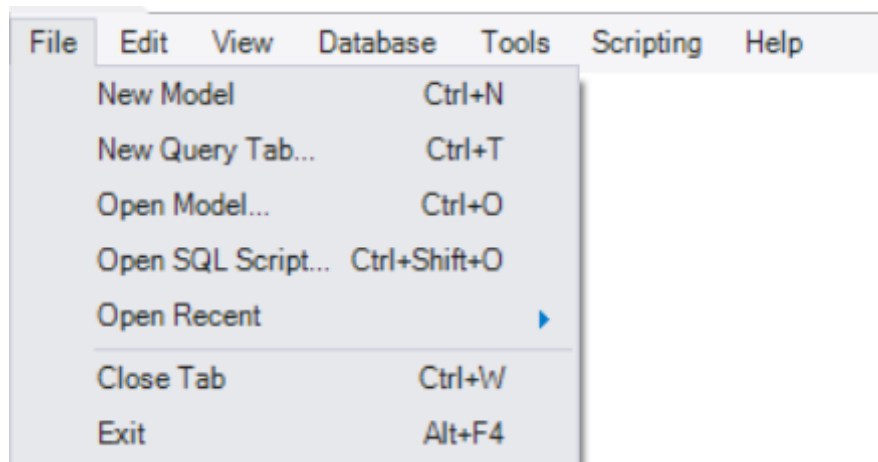
Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

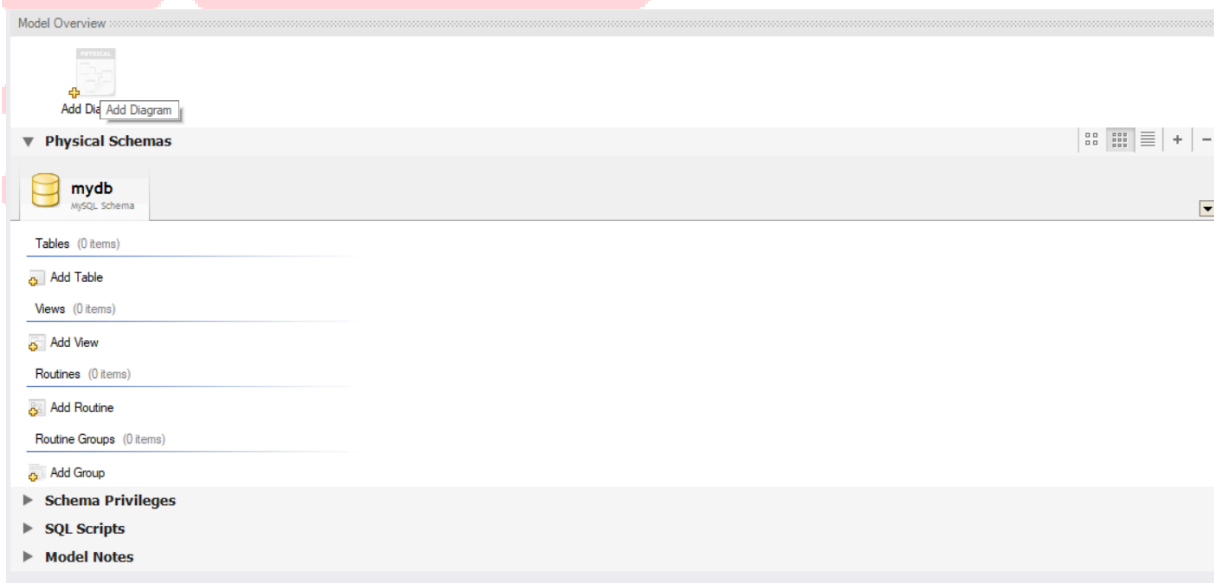
Dofinansowane przez
Unię Europejską





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.



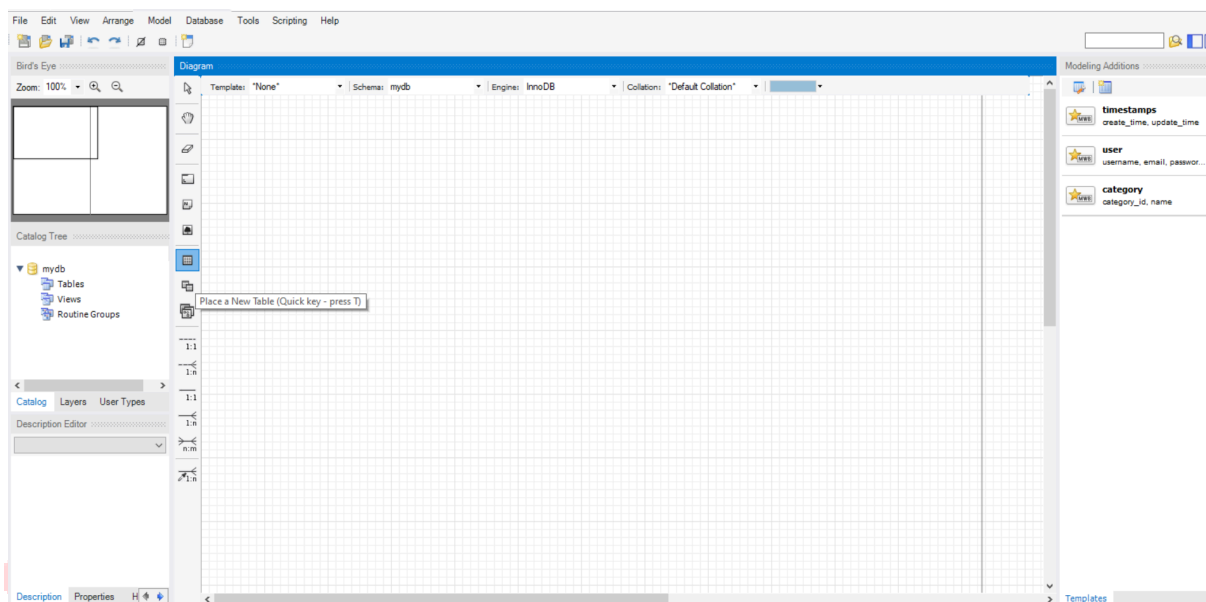
Fundusze Europejskie
dla Rozwoju Społecznego



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



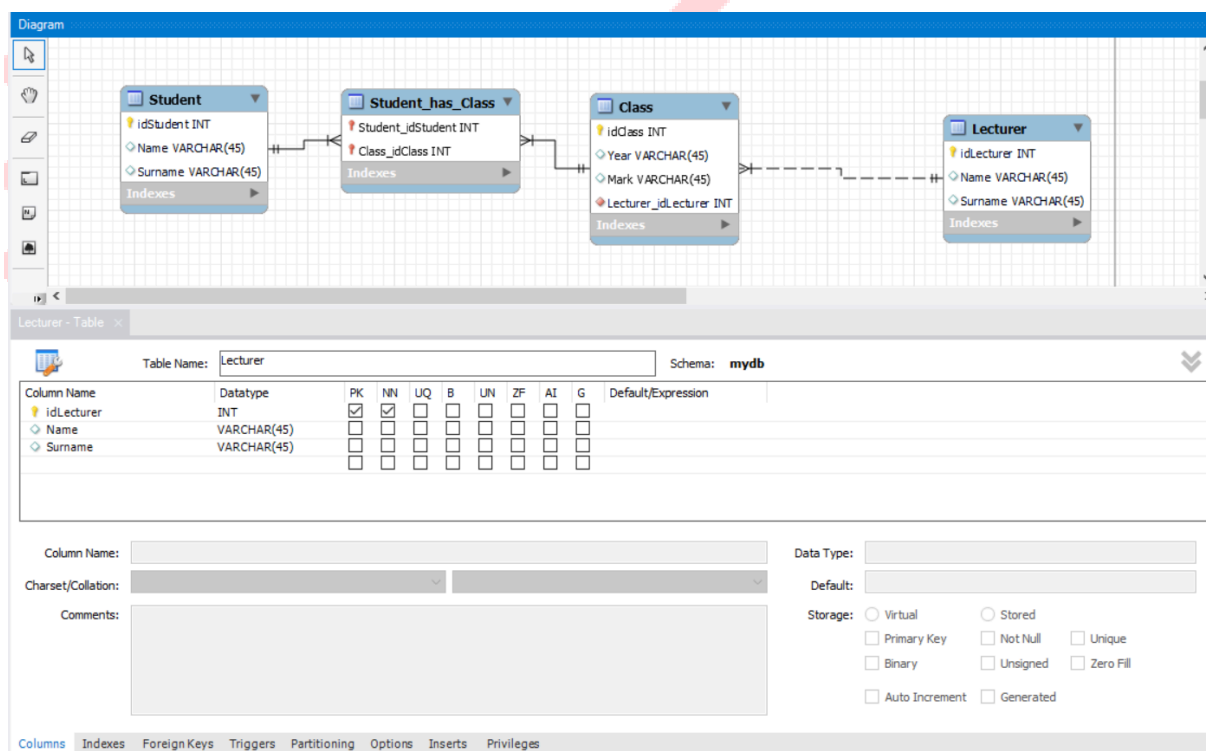
Rzeczpospolita
Polska

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.



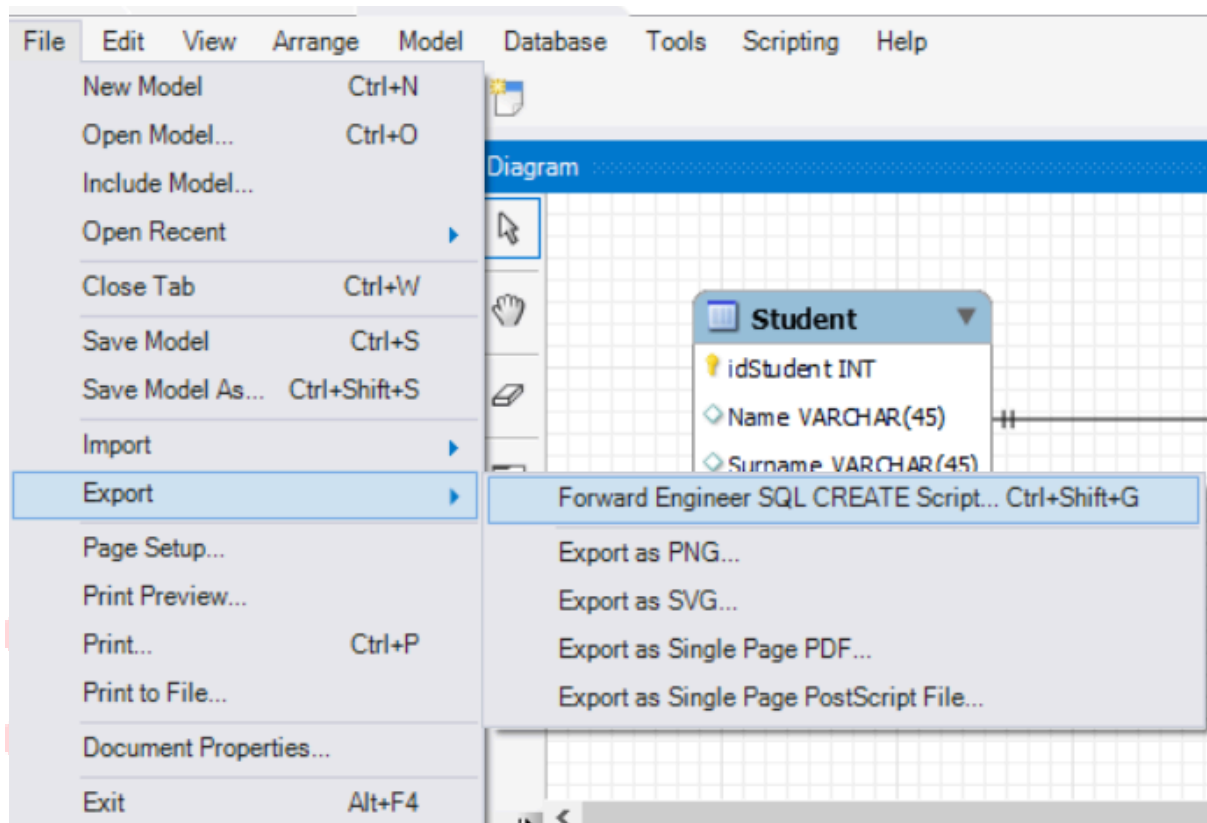
Fundusze Europejskie
dla Rozwoju Społecznego



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 zatwierdzane (`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 unaoczniało 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

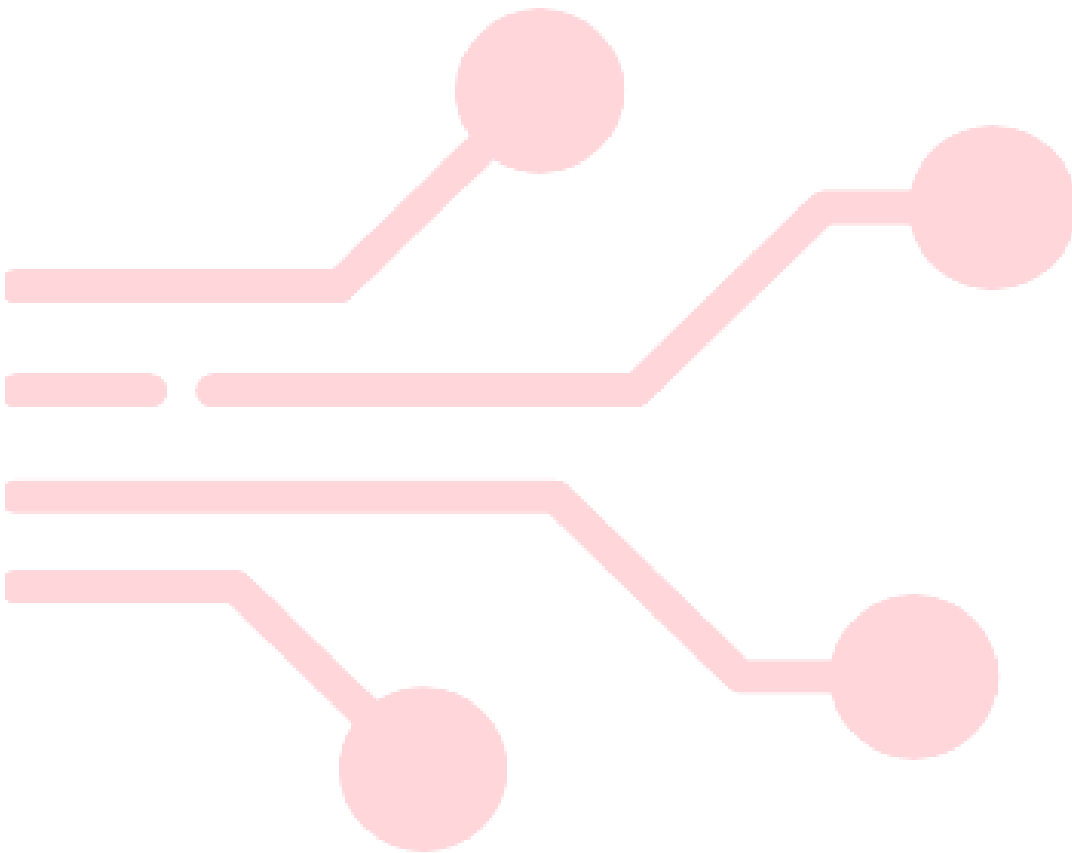


Rzeczpospolita
Polska

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

