



Politechnika Świętokrzyska

Wydział Elektrotechniki, Automatyki i Informatyki

Temat

Graficzne narzędzie do modelowania oraz tworzenia zapytań do relacyjnej bazy danych MySQL

Kierunek	Wykonali	Rok akademicki
Informatyka	Kamil Porada Adam Rychlik	2023/2024
Grupa		Data
1IZ21A		15.06.2024

SPIS TREŚCI

SPIS TREŚCI	2
WSTĘP	3
1. Relacyjna baza danych	4
1.1 Podstawowe pojęcia i zasady relacyjnych baz danych	4
1.2 Normalizacja baz danych	6
1.3 Język zapytań SQL (Structured Query Language)	7
1.4 Przegląd wybranych relacyjnych baz danych	8
2. Implementacja aplikacji	12
2.1 Technologia	12
2.2 Komunikacja	13
2.3 Architektura aplikacji	13
3. Interfejs użytkownika	15
3.1 Nawigacja	15
3.2 Formularz połączenia z bazą danych	16
3.3 Sekcja Table	17
3.4 Sekcja Insert Query	20
3.5 Sekcja Update Query	21
3.6 Sekcja Delete Query	22
3.7 Sekcja Select Query	24
4. Wybrane przykłady implementacji	27
4.1 Tworzenie formularza zapytania select	27
4.2 Odbieranie danych z frontendu na backendzie	28
4.3 Zamiana JSON na obiekt	29
4.4 Generowanie zapytania SQL	31
4.5 Wykonanie zapytania SQL	32
4.6 MySQL w kontenerze Docker	34
4.7 Odebranie danych na frontendzie	34
PODSUMOWANIE	36
SPIS RYSUNKÓW	38

WSTĘP

Celem niniejszej dokumentacji jest przedstawienie funkcjonalności oraz możliwości narzędzia MySQL Designer, które zostało opracowane w ramach projektu zaliczeniowego na przedmiot Technologie Obiektowe. MySQL Designer to zaawansowane, graficzne narzędzie do modelowania oraz tworzenia zapytań do baz danych MySQL, które ma na celu wspomaganie użytkowników w zrozumieniu koncepcji relacyjnych baz danych oraz umożliwienie efektywnej pracy z nimi.

Narzędzie MySQL Designer będzie posiadało szereg podstawowych funkcjonalności, które obejmują między innymi:

- Połączenie z bazą danych MySQL, co umożliwi użytkownikom zarządzanie ich danymi w czasie rzeczywistym.
- Tworzenie nowych tabel w bazie danych, pozwalające na organizację danych według potrzeb użytkowników.
- Wyświetlanie istniejących tabel oraz ich zawartości, co ułatwi przeglądanie i analizę danych.
- Edycja tabel, umożliwiającą wprowadzanie zmian w strukturze oraz danych zawartych w tabelach.
- Usuwanie tabel z bazy danych, co pozwoli na zarządzanie i utrzymywanie porządku w bazie danych.
- Tworzenie zapytań do bazy danych, w tym zapytań typu SELECT, INSERT, UPDATE oraz DELETE, co umożliwi pełne zarządzanie danymi zawartymi w bazie.

Frontend narzędzia zostanie wykonany w technologii React z użyciem frameworka NextJS, co pozwoli na tworzenie dynamicznych i interaktywnych interfejsów użytkownika. Stylizacja aplikacji będzie zrealizowana z wykorzystaniem Tailwind CSS, co umożliwi szybkie i efektywne tworzenie responsywnych oraz estetycznych interfejsów. Backend narzędzia zostanie zaprojektowany w języku Java z wykorzystaniem frameworka SpringBoot, co zapewni stabilność i skalowalność aplikacji. Baza danych MySQL będzie stanowiła serce systemu, przechowując wszystkie dane zarządzane przez użytkowników.

MySQL Designer będzie narzędziem, które nie tylko ułatwi pracę z bazami danych, ale także przyczyni się do głębszego zrozumienia relacyjnych baz danych przez użytkowników. Intuicyjny interfejs graficzny oraz rozbudowane funkcjonalności pozwolą na łatwe modelowanie, zarządzanie oraz manipulowanie danymi w bazach MySQL, co przyczyni się do zwiększenia efektywności pracy oraz poprawy jakości przechowywanych danych.

1. Relacyjna baza danych

1.1 Podstawowe pojęcia i zasady relacyjnych baz danych

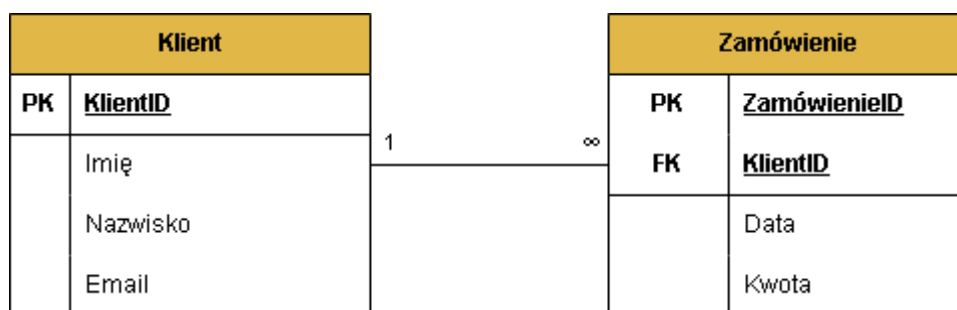
Relacyjne bazy danych stanowią fundament nowoczesnych systemów informatycznych, umożliwiając efektywne przechowywanie, zarządzanie i manipulowanie danymi. Koncepcja relacyjnych baz danych została wprowadzona przez E.F. Codda w 1970 roku, co zrewolucjonizowało podejście do organizacji danych [1]. W modelu relacyjnym dane są przechowywane w tabelach, które składają się z wierszy i kolumn. Każda tabela reprezentuje określony zbiór danych, a wiersze (rekordy) przechowują pojedyncze jednostki informacji, natomiast kolumny (atrybuty) definiują typy danych, które można w niej przechowywać.

Podstawową jednostką danych w relacyjnej bazie danych jest tabela, nazywana także relacją. Tabela składa się z określonej liczby kolumn, z których każda ma unikalną nazwę i typ danych, taki jak liczba całkowita, tekst, data itp. Każdy wiersz w tabeli musi mieć unikalną tożsamość, która jest zazwyczaj określana przez klucz główny (primary key). Klucz główny to jeden lub więcej atrybutów, które jednoznacznie identyfikują każdy rekord w tabeli. Na przykład, w tabeli przechowującej dane klientów, kluczem głównym może być unikalny identyfikator klienta.

Innym ważnym pojęciem w relacyjnych bazach danych jest klucz obcy (foreign key). Klucz obcy to atrybut lub zestaw atrybutów w jednej tabeli, który odnosi się do klucza głównego w innej tabeli. Klucze obce tworzą związki między tabelami, umożliwiając łączenie danych z różnych tabel w spójny sposób. Dzięki kluczom obcym można zapewnić integralność referencyjną, co oznacza, że każdy klucz obcy w tabeli musi odpowiadać istniejącemu kluczowi głównemu w powiązanej tabeli. Na przykład, w tabeli zamówień, klucz obcy może odnosić się do klucza głównego tabeli klientów, wskazując, który klient złożył dane zamówienie.

Typy danych w relacyjnych bazach danych są fundamentalnym elementem definiowania struktury tabel i przechowywanych w nich informacji. Każda kolumna w tabeli musi być przypisana do określonego typu danych, który określa rodzaj wartości, jakie mogą być w niej przechowywane. Do podstawowych typów danych należą typy numeryczne (takie jak INTEGER, FLOAT, DECIMAL), znakowe (CHAR, VARCHAR, TEXT), daty i czasu (DATE, TIME, DATETIME) oraz typy binarne (BLOB). Każdy typ danych ma swoje specyficzne właściwości i ograniczenia. Na przykład, typ INTEGER przechowuje liczby całkowite, natomiast VARCHAR umożliwia przechowywanie tekstu o zmiennej długości, z limitem określonym podczas definiowania kolumny. Poprawny wybór typów danych jest kluczowy dla optymalizacji wydajności bazy danych, zarządzania przestrzenią dyskową oraz zapewnienia integralności danych. Na przykład, użycie zbyt dużego typu danych może prowadzić do niepotrzebnego marnowania przestrzeni dyskowej, podczas gdy niewłaściwy typ danych może uniemożliwić przechowywanie określonych wartości.

Ustanawianie relacji między tabelami jest jednym z kluczowych aspektów projektowania relacyjnych baz danych. Relacje umożliwiają łączenie danych z różnych tabel w spójny i logiczny sposób, co jest podstawą modelu relacyjnego. Relacje między tabelami są realizowane za pomocą kluczy obcych (foreign keys). Klucz obcy to kolumna lub zestaw kolumn w jednej tabeli, która odnosi się do klucza głównego (primary key) w innej tabeli. Na przykład, w systemie zamówień, tabela "Zamówienia" może mieć kolumnę "KlientID", która jest kluczem obcym odnoszącym się do kolumny "KlientID" w tabeli "Klienci" (rysunek 1). Dzięki temu możliwe jest powiązanie każdego zamówienia z odpowiednim klientem. Relacje mogą być jedno-do-jednego, jedno-do-wielu lub wiele-do-wielu, w zależności od modelu danych. W relacji jedno-do-wielu, jeden rekord w tabeli może być powiązany z wieloma rekordami w innej tabeli, co jest typowe dla relacji między klientami a zamówieniami. Ustanawianie i zarządzanie relacjami między tabelami jest kluczowe dla zapewnienia integralności referencyjnej, co oznacza, że relacje między danymi są zgodne i spójne w całej bazie danych [2].



Rysunek 1 Schemat ilustrujący ustanawianie relacji między tabelami

1.2 Normalizacja baz danych

Normalizacja baz danych to proces organizowania danych w relacyjnej bazie danych w taki sposób, aby zminimalizować redundancję i zapewnić integralność danych. Jest to kluczowy element projektowania baz danych, który umożliwia efektywne przechowywanie danych i zarządzanie nimi. W tym rozdziale zostaną omówione zasady normalizacji baz danych.

Zasady normalizacji

Normalizacja baz danych opiera się na kilku podstawowych zasadach, które zostały sformalizowane w postaci tzw. postaci normalnych. Najważniejsze z nich to pierwsza, druga i trzecia postać normalna (1NF, 2NF, 3NF).

1. Pierwsza postać normalna (1NF):

- Każda kolumna w tabeli musi zawierać tylko pojedyncze wartości atomowe, czyli nie mogą one być podzielone na mniejsze części.
- Każdy rekord musi być unikalny.

2. Druga postać normalna (2NF):

- Spełnia wymagania pierwszej postaci normalnej.
- Wszystkie atrybuty nie będące kluczem głównym muszą być w pełni zależne od klucza głównego.
- Oznacza to, że tabela nie powinna zawierać żadnych częściowych zależności, czyli sytuacji, gdzie atrybut jest zależny tylko od części klucza głównego.

3. Trzecia postać normalna (3NF):

- Spełnia wymagania drugiej postaci normalnej.
- Żaden atrybut nie będący kluczem głównym nie może być przechodnio zależny od klucza głównego.
- Oznacza to, że atrybuty nie będące kluczem głównym powinny być zależne bezpośrednio tylko od klucza głównego, a nie od innych atrybutów [3].

Denormalizacja to proces odwrotny do normalizacji, polegający na celowym wprowadzeniu pewnej redundancji do bazy danych w celu poprawy wydajności zapytań i operacji na danych.

Denormalizację warto rozważyć w systemach, gdzie priorytetem jest szybkość odczytu danych, a nie ich spójność. Analiza może wykazać, że normalizacja powoduje zbyt dużą liczbę złożonych zapytań i połączeń między tabelami, co spowalnia wydajność systemu.

Korzyści denormalizacji obejmują zmniejszenie liczby operacji JOIN, co może znacząco przyspieszyć zapytania, oraz poprawę wydajności w systemach, gdzie dominuje odczyt danych nad operacjami zapisu.

Jednak denormalizacja ma również swoje zagrożenia. Zwiększa redundancję danych, co może prowadzić do większego ryzyka niespójności. Trudniejsze staje się zarządzanie danymi, ponieważ zmiany w jednym miejscu mogą wymagać modyfikacji w wielu miejscach. Ponadto, denormalizacja prowadzi do większego zużycia przestrzeni dyskowej z powodu powielania danych [5].

Normalizacja i denormalizacja są kluczowymi narzędziami w projektowaniu baz danych, które pozwalają na optymalne zorganizowanie danych w zależności od specyficznych potrzeb systemu. Normalizacja pomaga zapewnić integralność i spójność danych, podczas gdy denormalizacja może poprawić wydajność odczytu danych w określonych sytuacjach.

1.3 Język zapytań SQL (Structured Query Language)

Język zapytań SQL (Structured Query Language) jest podstawowym narzędziem do komunikacji z relacyjnymi bazami danych. Pozwala on na tworzenie, modyfikowanie, zarządzanie oraz manipulowanie danymi w bazach danych, co czyni go nieodzownym w pracy z systemami zarządzania bazami danych (DBMS). SQL składa się z kilku podstawowych poleceń, które umożliwiają wykonywanie różnych operacji na danych.

Do najważniejszych poleceń SQL należą: SELECT, INSERT, UPDATE oraz DELETE. Polecenie SELECT służy do pobierania danych z jednej lub więcej tabel. Dzięki niemu można wyświetlić określone rekordy spełniające zadane kryteria. Polecenie INSERT umożliwia dodawanie nowych rekordów do tabeli, co jest podstawowym sposobem wprowadzania danych do bazy. Polecenie UPDATE służy do modyfikowania istniejących rekordów, pozwalając na aktualizację danych w tabeli. Natomiast polecenie DELETE pozwala na usuwanie rekordów z tabeli, co jest niezbędne w przypadku zarządzania i utrzymywania aktualności danych. Przykładowy kod zapytania SELECT ma postać:

```
select LastName, FirstName, BirthDate,  
       YEAR(BirthDate) as BirthYear,  
       DATEDIFF(yy, BirthDate, getdate()) as Years  
from employees
```

Tworzenie i zarządzanie schematami bazy danych jest kolejnym ważnym aspektem SQL. Obejmuje to definiowanie struktury bazy danych, czyli tworzenie tabel, definiowanie relacji między nimi oraz zarządzanie indeksami. Tworzenie tabel polega na definiowaniu kolumn oraz typów danych, które mogą być w nich przechowywane. Definiowanie relacji między tabelami, realizowane za pomocą kluczy obcych, pozwala na ustanawianie powiązań między różnymi tabelami, co jest kluczowe dla zapewnienia spójności i integralności danych. Zarządzanie indeksami natomiast polega na tworzeniu struktur, które przyspieszają operacje wyszukiwania i filtrowania danych, co znacząco poprawia wydajność zapytań.

Zaawansowane zapytania SQL umożliwiają wykonywanie bardziej złożonych operacji na danych. Jednym z kluczowych elementów zaawansowanych zapytań jest łączenie tabel

(JOIN), które pozwala na łączenie danych z różnych tabel na podstawie powiązanych kolumn. Podzapytania (subqueries) to zapytania zagnieżdżone w innych zapytaniach, które umożliwiają wykonywanie bardziej złożonych operacji filtrujących i analitycznych. Funkcje agregujące, takie jak SUM, AVG, COUNT, MIN oraz MAX, pozwalają na wykonywanie operacji matematycznych na zestawach danych, co jest przydatne w raportowaniu i analizie danych. Przykładowy kod wykonujący łączenie tabel ma postać:

```
SELECT *  
FROM dbo.EMP as e INNER JOIN dbo.CAR as c ON e.IdPrac=c.IdPrac
```

Język SQL jest niezbędnym narzędziem w pracy z relacyjnymi bazami danych, oferując szeroki zakres poleceń do manipulacji danymi i zarządzania ich strukturą. Dzięki znajomości podstawowych i zaawansowanych zapytań SQL, możliwe jest efektywne zarządzanie i analizowanie danych, co jest kluczowe dla działania nowoczesnych systemów informatycznych [4].

1.4 Przegląd wybranych relacyjnych baz danych

MySQL jest jedną z najpopularniejszych relacyjnych baz danych na świecie, szeroko stosowaną w różnych aplikacjach webowych i korporacyjnych. Historia MySQL sięga 1995 roku, kiedy to została opracowana przez szwedzką firmę MySQL AB. W 2008 roku MySQL zostało przejęte przez firmę Sun Microsystems, a w 2010 roku Sun zostało przejęte przez Oracle Corporation. MySQL wyróżnia się swoją prostotą, wysoką wydajnością i skalowalnością, co sprawia, że jest często wybieranym rozwiązaniem do obsługi dużych i średnich projektów (rysunek 2).

Główne cechy MySQL obejmują wsparcie dla różnych systemów operacyjnych, łatwość instalacji i konfiguracji, a także wsparcie dla różnych typów danych i mechanizmów przechowywania, takich jak InnoDB i MyISAM. MySQL jest dostępny w dwóch wersjach: Community i Enterprise. Wersja Community jest open-source i dostępna bezpłatnie, co czyni ją atrakcyjną dla mniejszych firm i projektów. Wersja Enterprise oferuje dodatkowe funkcje, takie jak zaawansowane narzędzia monitorowania, wsparcie techniczne oraz bardziej zaawansowane mechanizmy bezpieczeństwa, co jest istotne dla dużych przedsiębiorstw i krytycznych aplikacji biznesowych.

Przykłady typowych zastosowań MySQL obejmują systemy zarządzania treścią (CMS), takie jak WordPress, Joomla i Drupal, platformy e-commerce, takie jak Magento, oraz aplikacje webowe, takie jak Facebook i Twitter, które wykorzystują MySQL do zarządzania ogromnymi ilościami danych.



Rysunek 2 Logo relacyjnej bazy danych MySQL
Źródło: <https://www.mysql.com/about/legal/logos.html>

PostgreSQL, znany również jako Postgres, jest zaawansowaną relacyjną bazą danych, która cieszy się uznaniem za swoją niezawodność, elastyczność i wsparcie dla standardów SQL. PostgreSQL zostało opracowane na Uniwersytecie Kalifornijskim w Berkeley, a jego korzenie sięgają 1986 roku. PostgreSQL jest open-source i rozwijane przez społeczność programistów na całym świecie (rysunek 3).

Jedną z kluczowych cech PostgreSQL jest jego zgodność ze standardami SQL i wsparcie dla zaawansowanych funkcji, takich jak transakcje ACID, klucze obce, widoki, wyzwalacze i procedury składowane. PostgreSQL wyróżnia się również możliwością rozszerzania funkcjonalności poprzez dodawanie własnych typów danych, funkcji, operatorów i metod indeksowania. Wsparcie dla replikacji i wysokiej dostępności sprawia, że PostgreSQL jest często wybierane do obsługi krytycznych aplikacji biznesowych [5].

Przykłady firm i projektów wykorzystujących PostgreSQL obejmują platformy takie jak Reddit, Skype, oraz licznych dostawców usług chmurowych, takich jak Heroku i Amazon Web Services (AWS), którzy oferują PostgreSQL jako część swoich usług bazodanowych.



Rysunek 3 Logo relacyjnej bazy danych PostgreSQL
Źródło: https://wiki.postgresql.org/wiki/File:PostgreSQL_logo.3colors.120x120.png

Oracle Database jest jednym z najpotężniejszych i najbardziej rozbudowanych systemów zarządzania bazami danych na rynku. Została stworzona przez firmę Oracle Corporation, założoną w 1977 roku przez Larry'ego Ellisona i jego współpracowników. Oracle Database jest znana ze swojej skalowalności, niezawodności i wsparcia dla zaawansowanych funkcji analitycznych i obliczeniowych (rysunek 4).

Kluczowe cechy Oracle Database obejmują wsparcie dla dużych ilości danych, zaawansowane mechanizmy zarządzania transakcjami, wysoka dostępność i bezpieczeństwo danych. Oracle Database oferuje również narzędzia do analizy danych, takie jak Oracle Analytics, oraz wsparcie dla Big Data i przetwarzania w chmurze. Te cechy sprawiają, że

Oracle Database jest często wybierana przez duże przedsiębiorstwa i instytucje finansowe, które potrzebują niezawodnego systemu do zarządzania ogromnymi ilościami danych.

Zastosowania Oracle Database obejmują systemy ERP (Enterprise Resource Planning), systemy CRM (Customer Relationship Management), oraz zaawansowane aplikacje analityczne i biznesowe w sektorach takich jak bankowość, telekomunikacja i administracja publiczna.



Rysunek 4 Logo relacyjnej bazy danych Oracle
Źródło: https://pl.m.wikipedia.org/wiki/Plik:Oracle_logo.svg

Microsoft SQL Server jest zaawansowanym systemem zarządzania relacyjnymi bazami danych, opracowanym przez firmę Microsoft. Został wprowadzony na rynek w 1989 roku i od tego czasu rozwijany jest jako integralna część ekosystemu Microsoft, co zapewnia jego głęboką integrację z innymi produktami tej firmy, takimi jak Windows Server, Azure, oraz narzędzia programistyczne z rodziny Visual Studio (rysunek 5).

Główne funkcje i możliwości Microsoft SQL Server obejmują zaawansowane mechanizmy zarządzania danymi, takie jak replikacja, mirroring, clustering, oraz wsparcie dla procedur składowanych i wyzwalaczy. Microsoft SQL Server oferuje również zaawansowane narzędzia analityczne, takie jak SQL Server Analysis Services (SSAS), oraz narzędzia do zarządzania i raportowania, takie jak SQL Server Reporting Services (SSRS) i SQL Server Integration Services (SSIS).

Typowe scenariusze użycia Microsoft SQL Server obejmują systemy BI (Business Intelligence), systemy zarządzania danymi w dużych przedsiębiorstwach, oraz aplikacje webowe i mobilne, które korzystają z integracji z innymi usługami Microsoft.



Rysunek 5 Logo relacyjnej bazy danych Microsoft SQL Server
Źródło: <https://www.svgrepo.com/svg/303229/microsoft-sql-server-logo>

SQLite to lekka, wbudowana baza danych, która jest szeroko stosowana w aplikacjach mobilnych, małych projektach oraz jako baza danych wbudowana w różne urządzenia i aplikacje. SQLite została opracowana przez D. Richarda Hipp'a w 2000 roku i jest dostępna jako oprogramowanie open-source (rysunek 6).

Specyficzne cechy SQLite obejmują brak potrzeby konfiguracji serwera, co oznacza, że cała baza danych jest przechowywana w jednym pliku na dysku, oraz minimalne wymagania dotyczące zasobów systemowych. SQLite jest bardzo prosty w użyciu i nie wymaga instalacji ani zarządzania skomplikowanymi ustawieniami serwera.

Zastosowanie SQLite obejmuje aplikacje mobilne, takie jak te na systemach Android i iOS, które korzystają z SQLite do przechowywania danych aplikacji lokalnie. SQLite jest również używany w małych projektach, prototypach, oraz jako baza danych wbudowana w różne urządzenia, takie jak routery, telewizory, a nawet samochody.



Rysunek 6 Logo relacyjnej bazy danych SQLite
Źródło: <https://www.sqlite.org>

Każda z omówionych baz danych ma swoje unikalne cechy i zastosowania, co sprawia, że wybór odpowiedniego systemu zarządzania bazą danych zależy od specyficznych potrzeb projektu i środowiska, w którym ma być używany [6].

2. Implementacja aplikacji

2.1 Technologia

Frontend aplikacji MySQL Designer został napisany w React, a konkretnie w frameworku Next.js. Next.js zapewnia możliwość tworzenia dynamicznych aplikacji z wykorzystaniem serwera, co pozwala na optymalizację i lepszą wydajność aplikacji. Stylizacja komponentów odbyła się za pomocą Tailwind CSS, co umożliwiło szybkie i efektywne tworzenie estetycznego i responsywnego interfejsu użytkownika. Tailwind CSS, jako framework narzędziowy do CSS, pozwala na tworzenie spójnych stylów poprzez zdefiniowane klasy utility, co znacznie przyspiesza proces projektowania i kodowania.

Do pisania kodu frontendowego wykorzystano zintegrowane środowisko programistyczne Visual Studio Code. Jest to popularne narzędzie wśród programistów frontendu ze względu na jego szerokie możliwości, takie jak wsparcie dla wielu rozszerzeń, debugowanie, integracja z systemami kontroli wersji oraz intuicyjny interfejs użytkownika.

Backend aplikacji został napisany w języku Java, a do jego tworzenia wykorzystano framework Spring Boot. Spring Boot ułatwia tworzenie niezawodnych i skalowalnych aplikacji serwerowych, zapewniając wbudowane mechanizmy zarządzania zależnościami, konfiguracji oraz bezpiecznego dostępu do danych. Kod backendu był tworzony przy użyciu zintegrowanego środowiska programistycznego IntelliJ IDEA, które jest cenione za swoją zaawansowaną funkcjonalność, wsparcie dla Javy oraz integrację z narzędziami do testowania i wdrażania aplikacji.

Backend został skonfigurowany tak, aby łączyć się z relacyjną bazą danych MySQL, co umożliwia rzeczywiste działanie aplikacji i prezentację pełnej funkcjonalności MySQL Designer. Dzięki temu użytkownicy mogą tworzyć, modyfikować i zarządzać bazami danych bezpośrednio z poziomu aplikacji, wykorzystując wszystkie możliwości oferowane przez MySQL.

Wszystkie zmiany w kodzie zarówno frontendowym, jak i backendowym, są wgrywane na zdalne repozytorium GitHub. Wykorzystanie GitHub, jako platformy do kontroli wersji zapewnia śledzenie wszystkich zmian w kodzie, możliwość współpracy zespołowej oraz łatwe zarządzanie wersjami aplikacji. Dzięki temu cały proces rozwoju oprogramowania jest dobrze zorganizowany i zabezpieczony, a wszelkie zmiany są dokładnie dokumentowane i kontrolowane.

2.2 Komunikacja

W aplikacji MySQL Designer komunikacja między frontendem a backendem odbywa się przy użyciu protokołu HTTP. Frontend, który jest napisany w React z użyciem biblioteki Next.js, wysyła żądania HTTP do serwera backendowego, który został zaimplementowany w języku Java przy użyciu frameworka Spring Boot.

W trakcie komunikacji, frontend wysyła zapytania HTTP do określonych punktów końcowych (endpoints) na serwerze backendowym. Na przykład, może to być żądanie pobrania listy nazw tabel z bazy danych lub żądanie dodania nowej kolumny do istniejącej tabeli. Serwer backendowy odbiera te żądania i przetwarza je zgodnie z logiką aplikacji.

Po przetworzeniu żądania, serwer generuje odpowiedź, która zawiera żądane dane lub informacje o wyniku operacji. Odpowiedź ta jest wysyłana z powrotem do frontendu, który ją odbiera i przetwarza. Na podstawie otrzymanych danych, frontend może aktualizować interfejs użytkownika, wyświetlając nowe informacje lub reagując na wykonane operacje.

Komunikacja przez protokół HTTP umożliwia dynamiczną interakcję użytkownika z aplikacją, pozwalając na dodawanie, edycję i usuwanie danych w bazie danych. Dodatkowo, pozwala na przysyłanie różnych rodzajów danych, takich jak tekst, obrazy czy pliki, co umożliwia bogatą funkcjonalność aplikacji.

2.3 Architektura aplikacji

Architektura aplikacji MySQL Designer oparta jest na architekturze klient-serwer, gdzie frontend, backend oraz baza danych MySQL współpracują ze sobą w celu obsługi żądań użytkownika oraz przechowywania danych (rysunek 7).

Frontend aplikacji, napisany w React z użyciem biblioteki Next.js, odpowiada za interakcję z użytkownikiem oraz prezentację danych. Interfejs użytkownika komunikuje się z serwerem backendowym, wysyłając żądania HTTP do odpowiednich punktów końcowych. Frontend jest odpowiedzialny za walidację danych wprowadzanych przez użytkownika oraz za prezentację wyników przetwarzania danych z serwera.

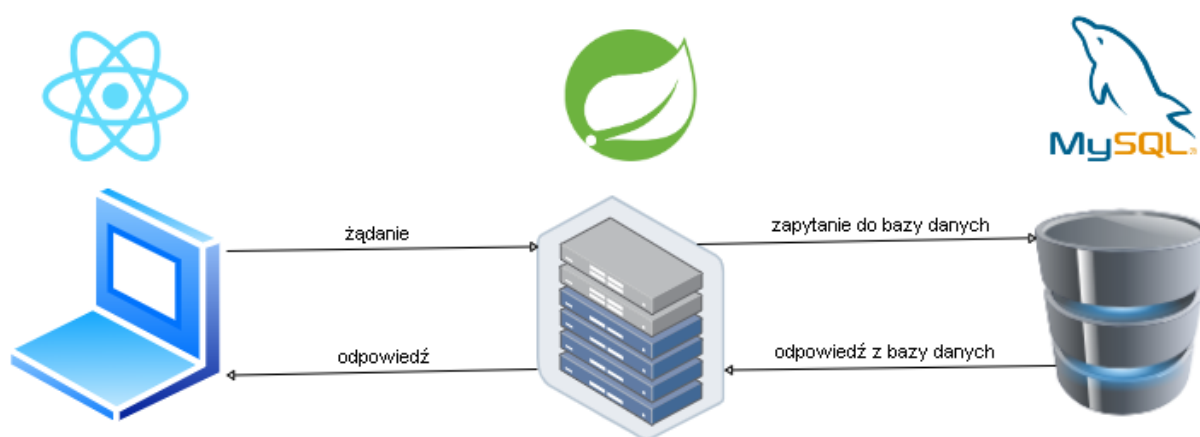
Serwer backendowy, zaimplementowany w języku Java przy użyciu frameworka Spring Boot, przyjmuje żądania HTTP od frontendu, przetwarza je zgodnie z logiką biznesową aplikacji oraz zarządza dostępem do bazy danych. Backend zajmuje się obsługą operacji CRUD (Create, Read, Update, Delete) na danych, które są przechowywane w bazie danych MySQL. Odpowiada również za logikę biznesową aplikacji, w tym walidację danych oraz autoryzację użytkowników.

Komunikacja między frontendem a backendem odbywa się poprzez protokół HTTP. Frontend wysyła żądania HTTP do serwera backendowego, a ten po przetworzeniu żądania generuje odpowiedź, która zawiera żądane dane lub informacje o wyniku operacji. Dane te mogą być następnie wyświetlane w interfejsie użytkownika frontendu.

Serwer backendowy łączy się z bazą danych MySQL, aby pobierać, modyfikować oraz przechowywać dane. Dane przesyłane między backendem a bazą danych są zazwyczaj w formacie JSON, co umożliwia łatwą komunikację między różnymi warstwami aplikacji.

Serwer backendowy odpowiada również za zapewnienie integralności danych oraz bezpieczeństwa aplikacji poprzez odpowiednie zabezpieczenia.

Cała architektura aplikacji jest zaprojektowana w sposób modułowy i skalowalny, co umożliwia rozwój i rozbudowę aplikacji w przyszłości. Dzięki zastosowaniu takiej architektury, aplikacja MySQL Designer może obsługiwać różnorodne operacje na danych oraz spełniać wymagania użytkowników w sposób efektywny i niezawodny.



Rysunek 7 Schemat architektury aplikacji MySQL Designer

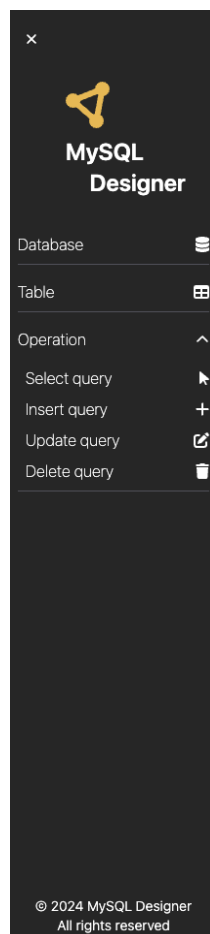
3. Interfejs użytkownika

3.1 Nawigacja

Nawigacja składa się z logo aplikacji oraz jej nazwy "MySQL Designer". Poniżej znajdują się przyciski:

- **Database:** Przenosi użytkownika do formularza połączenia z bazą danych.
- **Table:** Po kliknięciu użytkownik może dodawać, edytować, usuwać oraz wyświetlać listę tabel.
- **Operation:** Po rozwinięciu tego przycisku, użytkownik może wybierać spośród różnych rodzajów zapytań, takich jak Select Query, Insert Query, Update Query, Delete Query. Po wybraniu odpowiedniego zapytania, pojawi się formularz do tworzenia zapytania.

Nawigacja ma również możliwość zwinięcia poprzez kliknięcie ikonki "x" znajdującej się w lewym górnym rogu, co sprawia, że jest responsywna, co jest istotne zwłaszcza na małych ekranach, gdzie miejsce jest ograniczone (rysunek 8).



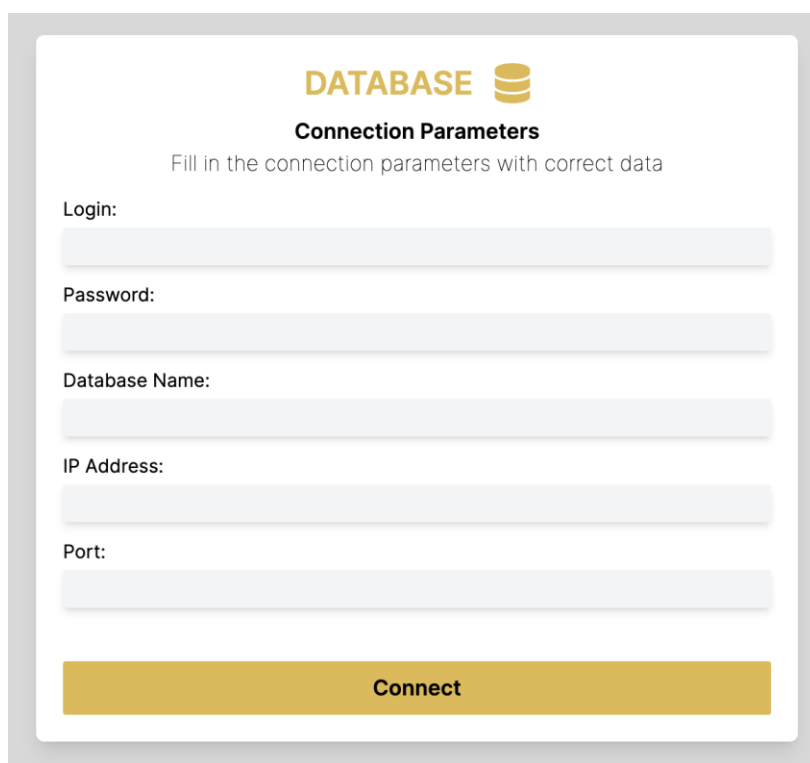
Rysunek 8 Pasek nawigacji w aplikacji MySQL Designer

3.2 Formularz połączenia z bazą danych

Po przejściu z paska nawigacyjnego w opcję "Database", użytkownikowi ukazuje się formularz zawierający pola:

- **Login:** Pole, w którym użytkownik wprowadza login do bazy danych.
- **Password:** Pole, gdzie użytkownik wpisuje hasło do bazy danych.
- **Database Name:** Pole, w którym użytkownik podaje nazwę bazy danych, do której chce się połączyć.
- **IP Address:** Pole, gdzie użytkownik wpisuje adres IP serwera bazy danych.
- **Port:** Pole, w którym użytkownik podaje numer portu, na którym działa serwer bazy danych.

Po wypełnieniu wszystkich pól użytkownik może nacisnąć przycisk "Connect", co spowoduje wysłanie danych do backendu. Backend przetwarza te dane i próbuje nawiązać połączenie z bazą danych, używając dostarczonych danych uwierzytelniających (login, hasło, nazwa bazy danych, adres IP, numer portu). Jeśli dane są poprawne i połączenie z bazą danych zostaje pomyślnie ustanowione, użytkownik może wykonywać operacje, przechodząc do kolejnych funkcjonalności aplikacji (rysunek 9).



The image shows a web form titled "DATABASE" with a database icon. Below the title is the subtitle "Connection Parameters" and a instruction "Fill in the connection parameters with correct data". The form contains five input fields, each with a label: "Login:", "Password:", "Database Name:", "IP Address:", and "Port:". At the bottom of the form is a yellow button labeled "Connect".

Rysunek 9 Formularz połączenia się z bazą danych

3.3 Sekcja Table

Po kliknięciu na przycisk "Table" w pasku nawigacyjnym, użytkownik uzyskuje dostęp do funkcjonalności zarządzania tabelami w bazie danych. W tej sekcji użytkownik może dodawać, edytować, usuwać oraz wyświetlać listę tabel, co pozwala na pełne zarządzanie strukturą bazy danych z poziomu aplikacji.

Podczas tworzenia nowej tabeli, użytkownik zaczyna od podania nazwy tabeli. Jest to pierwszy krok w procesie definiowania nowej tabeli. Następnie, użytkownik może dodać kolumny, które będą składały się na strukturę tej tabeli. W celu dodania kolumn, użytkownik klika przycisk plusa, który otwiera formularz do dodawania nowych kolumn. Dla każdej kolumny należy podać nazwę oraz wybrać typ danych z dostępnych opcji, takich jak integer, bool, float, double, decimal, date, datetime, timestamp, time, char, varchar.

Dla kolumn typu decimal użytkownik musi dodatkowo podać dwie wartości: liczbę cyfr całkowitych (number of integer digits) oraz liczbę miejsc po przecinku (number of decimal places). W przypadku kolumn typu varchar lub char, należy podać długość ciągu znaków (string length). Dla innych typów danych, takich jak integer czy bool, nie są wymagane dodatkowe informacje.

Poza podstawowymi właściwościami kolumn, użytkownik ma możliwość ustawienia dodatkowych opcji. Może oznaczyć kolumnę jako klucz główny (primary key), co identyfikuje każdą rekord w tabeli unikalnie. Jeśli kolumna ma być kluczem obcym (foreign key), użytkownik zaznacza odpowiednie pole, po czym pojawiają się dwa dodatkowe pola umożliwiające wybór tabeli oraz kolumny, do której będzie odnosił się klucz obcy. Istnieje także możliwość ustawienia kolumny jako automatycznie inkrementowanej (auto increment), unikalnej (unique) oraz takiej, która nie może zawierać wartości null (not null).

Po skonfigurowaniu wszystkich kolumn i ich właściwości, użytkownik zatwierdza utworzenie tabeli klikając przycisk "Create Table". W ten sposób nowa tabela jest tworzona w bazie danych i staje się dostępna do dalszego użytku (rysunek 10).




NEW TABLE

Create Table

Enter the table name and then use the plus sign to add columns to the table. Choose its name, data type, and other additional parameters!

Table Name:

Table Columns +

 id	INTEGER	PRIMARY KEY, AUTO INCREMENT, UNIQUE, NOT NULL	 
--	---------	---	---

Column username

Column Name:

Variable Type:

String Length:

Primary Key ☐

Foreign Key ☐

Auto Increment ☐

Unique ☐

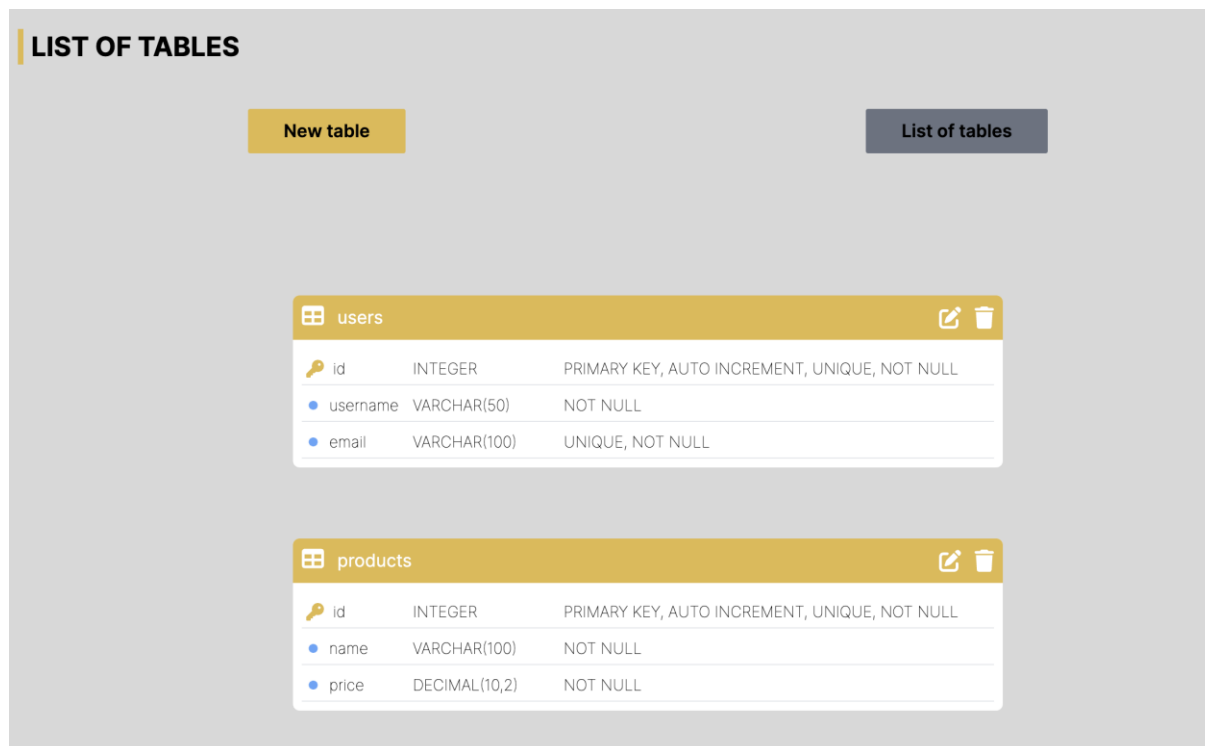
Not Null ☒

Add Column

Create New Table

Rysunek 10 Formularz tworzenia nowej tabeli

Po przejściu na listę tabel, użytkownik widzi wszystkie istniejące tabele, w tym również nowo utworzoną tabelę. Lista tabel pozwala na przeglądanie struktury bazy danych oraz wykonywanie dalszych operacji na tabelach. Aby usunąć tabelę, użytkownik klika na ikonę kosza znajdującą się obok wybranej tabeli. Usunięcie tabeli powoduje jej trwałe usunięcie z bazy danych (rysunek 11).



Rysunek 11 Lista tabel

Edycja tabeli jest równie prosta – po kliknięciu na ikonę edycji, wyświetlany jest formularz z aktualnymi danymi tabeli. Użytkownik może zmienić nazwę tabeli oraz edytować jej kolumny. Kolumny można usuwać, edytować lub dodawać nowe. Kolorystyka tła kolumn ułatwia zarządzanie zmianami: białe tło oznacza kolumnę, która nie została zmieniona; czerwone tło wskazuje na kolumnę usuniętą (możliwa jest jej przywrócenie poprzez kliknięcie strzałki powrotu zamiast ikony kosza); niebieskie tło oznacza kolumnę edytowaną, a zielone tło – nowo dodaną kolumnę (rysunek 12).

The screenshot shows the "EDIT TABLE" form. At the top, it says "EDIT TABLE" with a table icon. Below that is the title "Edit Table" and a instruction: "Enter the table name and then use the plus sign to add columns to the table. Choose its name, data type, and other additional parameters!". There is a "Table Name:" label and a text input field containing "users". Below that is the "Table Columns" label and a list of columns. Each column has a color-coded background: "id" is white, "username" is red, "email" is blue, and "address" is green. Each column also has edit and delete icons. At the bottom, there is a large yellow "Edit Table" button.

EDIT TABLE			
Edit Table			
Enter the table name and then use the plus sign to add columns to the table. Choose its name, data type, and other additional parameters!			
Table Name: <input type="text" value="users"/>			
Table Columns +			
id	INTEGER	PRIMARY KEY, AUTO INCREMENT, UNIQUE, NOT NULL	
username	VARCHAR(50)	NOT NULL	
email	VARCHAR(150)	UNIQUE, NOT NULL	
address	VARCHAR(200)	UNIQUE, NOT NULL	
Edit Table			

Rysunek 12 Formularz edycji tabeli

Dzięki tym funkcjom, użytkownik może w pełni zarządzać strukturą swoich tabel, dostosowując je do zmieniających się potrzeb aplikacji i bazy danych. Cały proces zarządzania tabelami jest intuicyjny i efektywny, co zapewnia sprawne działanie aplikacji i bazy danych.

3.4 Sekcja Insert Query

Po kliknięciu na przycisk "Insert Query" w pasku nawigacyjnym, uruchamia się formularz tworzenia zapytania INSERT. Formularz ten umożliwia użytkownikowi dodanie nowego rekordu do wybranej tabeli w bazie danych. Na wstępie użytkownik wybiera tabelę, do której chce dodać dane, korzystając z rozwijanego menu (select), które pobiera dostępne tabele z bazy danych.

Po wybraniu konkretnej tabeli, aplikacja dynamicznie ładuje pola tej tabeli bezpośrednio z bazy danych. Dla każdego pola tabeli tworzone są odpowiednie elementy formularza (inputy) wraz z etykietami, które umożliwiają użytkownikowi wprowadzenie danych do każdego pola.

Jeśli jednak dana kolumna w tabeli jest oznaczona jako klucz główny (primary key), jest typu integer i posiada atrybut auto increment, input dla tego pola nie jest tworzony. Wynika to z faktu, że wartość dla takiej kolumny jest automatycznie generowana przez bazę danych podczas wstawiania nowego rekordu, co eliminuje potrzebę ręcznego wprowadzania tej wartości przez użytkownika.

Formularz jest zaprojektowany tak, aby ułatwić użytkownikowi wprowadzanie danych do tabeli w sposób intuicyjny i efektywny. Użytkownik widzi tylko te pola, które są wymagane do ręcznego wprowadzenia, co minimalizuje możliwość popełnienia błędu i przyspiesza proces dodawania nowych rekordów do bazy danych (rysunek 13).

Po wprowadzeniu wszystkich niezbędnych danych użytkownik zatwierdza formularz, a aplikacja generuje odpowiednie zapytanie INSERT, które jest następnie wysyłane do backendu. Backend przetwarza zapytanie i wykonuje je na bazie danych, dodając nowy rekord do wybranej tabeli. W ten sposób użytkownik może łatwo i szybko wprowadzać nowe dane do swojej bazy danych, korzystając z intuicyjnego interfejsu użytkownika.

INSERT QUERY +

Add New Row

Fill in the values for the new row

Select Table:
City

CityName:
Radom

Area:
111,8

Voivodeship:
Mazowieckie

Population:
213715

Insert Row

Rysunek 13 Formularz tworzenia zapytania insert

3.5 Sekcja Update Query

Sekcja "Update Query" umożliwia użytkownikowi aktualizację danych w wybranej tabeli. Proces tworzenia zapytania UPDATE rozpoczyna się analogicznie do sekcji "Insert Query". Na początku użytkownik musi wybrać nazwę tabeli, z której chce zaktualizować dane, korzystając z rozwijanego menu (select).

Po wybraniu tabeli, użytkownik może dodać kolumny, które chce zaktualizować, klikając przycisk plusa. Po dodaniu kolumny, użytkownik wybiera jej etykietę oraz wpisuje nową wartość, którą chce przypisać do tej kolumny. Dzięki temu, użytkownik może precyzyjnie określić, które dane mają zostać zaktualizowane i na jakie wartości.

Podobnie jak w przypadku tworzenia klauzuli WHERE, użytkownik może określić warunki, na podstawie których dane będą aktualizowane. Kliknięcie przycisku plusa pozwala na dodanie trzech kontroltek, które tworzą pojedynczy warunek klauzuli WHERE: wybór nazwy kolumny, wybór operatora porównania (znaku), oraz wpisanie wartości.

Po dodaniu pierwszego warunku, kliknięcie przycisku plusa powoduje wyświetlenie opcji wyboru operatora logicznego (np. AND, OR), który pozwala na połączenie warunków. Następne kliknięcie przycisku plusa powoduje dodanie kolejnych trzech kontroltek do tworzenia następnego warunku. Proces ten może być powtarzany wielokrotnie, co pozwala na zbudowanie złożonych zapytań z wieloma warunkami.

Gdy użytkownik wprowadzi wszystkie niezbędne dane i warunki, zatwierdza formularz. Aplikacja generuje odpowiednie zapytanie UPDATE, które jest następnie wysyłane do backendu. Backend przetwarza zapytanie i wykonuje je na bazie danych, aktualizując wybrane rekordy zgodnie z wprowadzonymi przez użytkownika wartościami i warunkami (rysunek 14).

Taka struktura formularza zapewnia użytkownikowi elastyczność w definiowaniu, które dane mają być zaktualizowane oraz umożliwia precyzyjne określenie warunków, na podstawie których aktualizacja zostanie przeprowadzona. Dzięki temu proces aktualizacji danych w bazie jest intuicyjny i efektywny.

UPDATE QUERY

Update Row

Fill in the values for the row to be updated

Select Table:

City

Columns to update: +

Column name: New Value:

Population 190000

Columns to where statement: +

Column name: Sign: Where Value:

ID = 5

Update Row

Rysunek 14 Formularz tworzenia zapytania update

3.6 Sekcja Delete Query

Sekcja "Delete Query" w aplikacji umożliwia użytkownikowi usuwanie danych z wybranej tabeli. Proces tworzenia zapytania DELETE rozpoczyna się, podobnie jak w poprzednich sekcjach, od wyboru nazwy tabeli z rozwijanego menu (select). Użytkownik wybiera tabelę, z której chce usunąć dane, co jest pierwszym krokiem do skonfigurowania zapytania.


Po wybraniu tabeli, użytkownik może dodać warunki, które określają, które rekordy mają zostać usunięte. Aby to zrobić, użytkownik klika przycisk plusa, który dodaje trzy kontrolki do formularza, tworzące pojedynczy warunek klauzuli WHERE: wybór nazwy

kolumny, wybór operatora porównania (znaku) oraz wpisanie wartości. Te kontrolki pozwalają na precyzyjne określenie warunku, według którego rekordy zostaną usunięte.

Analogicznie do sekcji "Update Query", po dodaniu pierwszego warunku, użytkownik może kliknąć przycisk plusa ponownie, aby dodać operator logiczny (np. AND, OR), który umożliwia połączenie kilku warunków. Następne kliknięcie przycisku plusa pozwala na dodanie kolejnych trzech kontrolki do tworzenia następnego warunku. Dzięki temu użytkownik może stworzyć złożoną klauzulę WHERE z wieloma warunkami, które precyzyjnie określają, które rekordy mają być usunięte z tabeli.

Gdy wszystkie warunki zostaną wprowadzone, użytkownik zatwierdza formularz. Aplikacja generuje odpowiednie zapytanie DELETE, które następnie jest wysyłane do backendu. Backend przetwarza zapytanie i wykonuje je na bazie danych, usuwając wybrane rekordy zgodnie z określonymi przez użytkownika warunkami (rysunek 15).


Struktura formularza w sekcji "Delete Query" zapewnia użytkownikowi elastyczność w definiowaniu, które rekordy mają zostać usunięte, a także umożliwia precyzyjne określenie warunków, na podstawie których usunięcie zostanie przeprowadzone. Dzięki temu proces usuwania danych z bazy jest intuicyjny, bezpieczny i skuteczny.

DELETE QUERY 

Delete Row

Fill in the values for the row to be deleted

Select Table:
City

Columns to where statement: 

Column name: Sign: Where Value:
ID = 3

Delete Row

Rysunek 15 Formularz tworzenia zapytania delete

3.7 Sekcja Select Query

Sekcja "Select Query" w aplikacji jest kluczowym elementem, umożliwiającym użytkownikowi wykonywanie zapytań SELECT w celu pobierania danych z bazy danych. Proces tworzenia zapytania SELECT zaczyna się tradycyjnie od wyboru nazwy tabeli z rozwijanego menu (select). Użytkownik wybiera tabelę, z której chce pobrać dane, co stanowi pierwszy krok w konfigurowaniu zapytania.

Po wyborze tabeli użytkownik może za pomocą pól wyboru (checkboxów) określić, które kolumny chce pobrać. Użytkownik ma możliwość zaznaczenia opcji *(all), aby pobrać wszystkie kolumny z wybranej tabeli, lub zaznaczenia pojedynczych kolumn, które zostały zaczytane z bazy danych. To daje użytkownikowi elastyczność w określaniu zakresu danych, które mają zostać pobrane.

Następnie, podobnie jak w poprzednich przypadkach (Insert i Update Query), użytkownik może dodać warunki klauzuli WHERE. Klikając przycisk plusa, użytkownik dodaje trzy kontrolki do formularza, które umożliwiają tworzenie pojedynczego warunku: wybór nazwy kolumny, wybór operatora porównania (znaku) oraz wpisanie wartości. Dalsze kliknięcia przycisku plusa pozwalają na dodanie kolejnych warunków, które mogą być łączone za pomocą operatorów logicznych (np. AND, OR), tworząc złożoną klauzulę WHERE.

Użytkownik ma również możliwość wpisania wartości w polu limit, aby ograniczyć liczbę zwracanych rekordów. Jest to szczególnie przydatne w przypadku pracy z dużymi zbiorami danych, gdy użytkownik chce pobrać tylko określoną ilość danych.

Na koniec, użytkownik może skonfigurować sortowanie wyników za pomocą klauzuli ORDER BY. Wybiera z rozwijanego menu kolumnę, względem której dane mają być sortowane, a następnie wybiera jedną z opcji radiobuttonów: none (brak sortowania), ascending (rosnąco) lub descending (malejąco). Dzięki temu użytkownik może łatwo kontrolować sposób prezentacji zwracanych danych (rysunek 16).

SELECT QUERY

Select Rows

Fill in the values for the row to be selected

Select Table:

City
▼

Columns to select:

☒ ALL (*)
☐ CityName
☐ Area
☐ Voivodeship
☐ Population

Columns for where statement: +

Column name:

Sign:

Where Value:

Column
▼

Select Sign
▼

Limit:

Order By:
None
▼

☒ None
☐ Ascending
☐ Descending

Select Rows

Rysunek 16 Formularz tworzenia zapytania select

Po skonfigurowaniu wszystkich ustawień, użytkownik klika przycisk select rows, aby wysłać zapytanie do bazy danych. Backend przetwarza zapytanie, a baza danych zwraca dane, które następnie są ilustrowane na froncie w postaci tabeli. Tabela wyświetla zwrócone rekordy w przejrzysty sposób, umożliwiając użytkownikowi łatwe przeglądanie i analizowanie pobranych danych (rysunek 17).

Select query results

Area	Voivodeship	Population	ID	CityName
111.8000	Mazowieckie	213715.0	1	Radom
110.0000	Świętokrzyskie	195942.0	2	Kielce
326.8500	Małopolskie	766683.0	4	Kraków
91.1600	Śląskie	190000.0	5	Sosnowiec

Close

Rysunek 17 Wyniki zapytania select przedstawione w postaci tabelarycznej

Ta sekcja zapewnia użytkownikowi pełną kontrolę nad zapytaniami SELECT, umożliwiając precyzyjne określenie, które dane mają zostać pobrane i jak mają być sortowane oraz filtrowane. Dzięki temu proces pobierania danych z bazy jest intuicyjny, efektywny i dostosowany do potrzeb użytkownika.

4. Wybrane przykłady implementacji

W tym rozdziale zostaną przedstawione poszczególne etapy implementacji na frontendzie, backendzie oraz bazie danych. Zostanie omówiony sposób przesyłania danych i format tych danych na przykładzie tworzenia zapytania SELECT. Szczegółowo zostanie omówione, jak dane są przekazywane między frontendem a backendem, jak backend przetwarza te dane i jak są one ostatecznie pobierane z bazy danych i zwracane do użytkownika.

4.1 Tworzenie formularza zapytania select

Formularz zapytania SELECT został opisany w punkcie 3.7 (rysunek 16). Poniżej opisany jest kod odpowiedzialny za wysyłanie danych do backendu.

```
const handleSubmit = async () => {
  const cleanWhereStatementValues = whereStatementValues.filter(value => {
    return value.whereColumnName !== '' || value.whereColumnSign !== '' || value.whereColumnValue !== ''
  })

  const cleanWhereColumnOperatorValue = whereColumnOperatorValue.filter(value => {
    return value.whereOperator !== ''
  })

  const dataToSend = {
    tableName: tableName,
    selectedColumns: selectedColumns,
    whereStatementValues: cleanWhereStatementValues,
    whereColumnOperatorValue: cleanWhereColumnOperatorValue,
    limit: limit === '' ? 'none' : limit,
    orderBy: orderBy,
    orderByColumn: orderByColumn,
  }

  console.log('Data to send:', dataToSend)
  try {
    const response = await fetch('/api/select', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify(dataToSend),
    })

    if (response.ok) {
      const responseData = await response.json()
      setSqlCode(responseData.response)
      handleModalOpen()
    } else {
      console.error('Error sending data:', response.statusText)
    }
  } catch (error) {
    console.error('Error sending data:', error)
  }
}
```

Rysunek 18 Implementacja metody handleSubmit w komponencie SelectQueryForm

Funkcja `handleSubmit` realizuje następujące kroki:

1. Filtruje dane wprowadzone do klauzuli `WHERE`, aby usunąć puste wartości.
2. Przygotowuje obiekt `dataToSend`, który zawiera:
 - o nazwę tabeli (`tableName`),
 - o wybrane kolumny (`selectedColumns`),
 - o przefiltrowane wartości klauzuli `WHERE` (`cleanWhereStatementValues`),
 - o przefiltrowane operatory logiczne klauzuli `WHERE` (`cleanWhereColumnOperatorValue`),
 - o limit (`limit`),
 - o opcje sortowania (`orderBy` i `orderByColumn`).
3. Wysyła dane do backendu za pomocą żądania `HTTP POST` na endpoint `/api/select`.
4. Jeśli żądanie zakończy się sukcesem, odbiera dane odpowiedzi i ustawia je w stanie aplikacji, otwierając modal z wynikami. Jeśli żądanie się nie powiedzie, wyświetlany jest błąd w konsoli.

Taki sposób przesyłania danych pozwala na dynamiczne tworzenie i wysyłanie zapytań do backendu, co umożliwia elastyczne i efektywne zarządzanie danymi z bazy danych przez użytkownika (rysunek 18).

4.2 Odbieranie danych z frontendu na backendzie

Kod odpowiedzialny za odbieranie danych wysłanych z frontendu na backend działa, jako punkt końcowy obsługujący zapytania typu `POST` dla operacji `SELECT` (rysunek 19).



```
adam
@PostMapping("/select")
public ResponseEntity<Map<String,String>> select(@RequestBody SelectClass sel){

    Map<String, String> data = new HashMap<>();
    String Response=sel.toString();
    data.put("response",Response);
    return ResponseEntity.ok(data);
}
```

Rysunek 19 Implementacja punktu końcowego obsługujący zapytanie typu `POST`

Kod ten wykonuje następujące operacje:

1. **Definiowanie Endpointu:**
 - o Adnotacja `@PostMapping("/select")` wskazuje, że metoda `select` jest odpowiedzialna za obsługę żądań typu `POST` skierowanych na endpoint `/select`.

2. Odbieranie Danych:

- o Parametr `@RequestBody SelectClass sel` oznacza, że dane przesyłane w ciele żądania są deserializowane do obiektu klasy `SelectClass`. Dzięki temu możliwe jest bezpośrednie wykorzystanie danych przesłanych z frontendu.

3. Przetwarzanie Danych:

- o Tworzona jest mapa `data` typu `HashMap`, która będzie przechowywać odpowiedź.
- o Z obiektu `sel` pobierana jest jego tekstowa reprezentacja (`sel.toString()`), która następnie zostaje przypisana do zmiennej `response`.

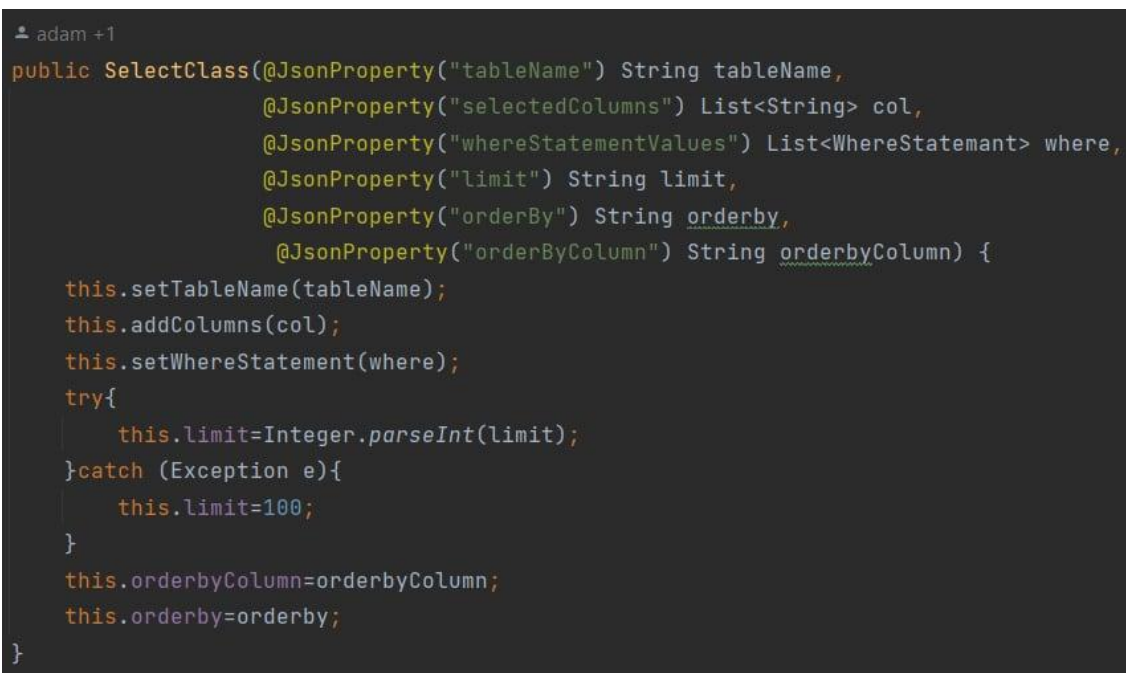
4. Przygotowanie Odpowiedzi:

- o W mapie `data` umieszczana jest para klucz-wartość, gdzie kluczem jest `"response"`, a wartością jest zawartość zmiennej `response`.
- o Ostatecznie metoda zwraca odpowiedź HTTP 200 (OK) wraz z mapą `data` jako zawartością odpowiedzi, wykorzystując `ResponseEntity.ok(data)`.

Ten proces zapewnia, że dane przesłane z frontendu są odbierane i przetwarzane na backendzie, a następnie odpowiednia odpowiedź jest zwracana do frontendu. Dzięki temu możliwe jest dalsze przetwarzanie danych lub wyświetlanie wyników zapytań użytkownikowi.

4.3 Zamiana JSON na obiekt

Kod odpowiedzialny za zamianę JSON na obiekt wykorzystuje adnotacje `@JsonProperty` z biblioteki Jackson. Poniżej przedstawiono konstruktor klasy `SelectClass`, który dokonuje deserializacji danych JSON (rysunek 20).

A screenshot of a code editor showing the constructor of the SelectClass class. The code is in Java and uses Jackson annotations for JSON deserialization. The constructor takes several parameters: tableName, col, where, limit, orderby, and orderbyColumn. It then sets these values on the object, with some error handling for the limit parameter.

```
adam +1
public SelectClass(@JsonProperty("tableName") String tableName,
                  @JsonProperty("selectedColumns") List<String> col,
                  @JsonProperty("whereStatementValues") List<WhereStatement> where,
                  @JsonProperty("limit") String limit,
                  @JsonProperty("orderBy") String orderby,
                  @JsonProperty("orderByColumn") String orderbyColumn) {
    this.setTableName(tableName);
    this.addColumns(col);
    this.setWhereStatement(where);
    try{
        this.limit=Integer.parseInt(limit);
    }catch (Exception e){
        this.limit=100;
    }
    this.orderbyColumn=orderbyColumn;
    this.orderby=orderby;
}
```

Rysunek 20 Zamiana JSON na obiekt

Konstruktor ten wykonuje następujące operacje:

1. Adnotacje @JsonProperty:

- o Adnotacje @JsonProperty wskazują na nazwę pola w danych JSON, które powinno zostać przypisane do odpowiedniego parametru konstruktora. Dzięki temu Jackson może automatycznie przetworzyć dane JSON na obiekt Javy.

2. Przypisanie wartości:

- o `this.setTableName(tableName);` - przypisanie nazwy tabeli do odpowiedniego pola obiektu.
- o `this.addColumns(col);` - dodanie listy wybranych kolumn do obiektu.
- o `this.setWhereStatement(where);` - ustawienie wartości klauzuli WHERE.

3. Obsługa limitu:

- o `try { this.limit = Integer.parseInt(limit); } catch (Exception e) { this.limit = 100; }` - próba konwersji wartości limitu z typu String na int. W przypadku niepowodzenia ustawiany jest domyślny limit wynoszący 100.

4. Przypisanie wartości dla sortowania:

- o `this.orderbyColumn = orderByColumn;` - przypisanie nazwy kolumny, względem której ma być wykonane sortowanie.
- o `this.orderby = orderBy;` - przypisanie kierunku sortowania (rosnąco lub malejąco).

Ten konstruktor zapewnia, że dane JSON są prawidłowo przetwarzane i przypisywane do odpowiednich pól obiektu `SelectClass`. Dzięki temu możliwe jest łatwe i bezbłędne przekształcenie struktury danych JSON na obiekt Javy, który może być dalej wykorzystywany w aplikacji.

4.4 Generowanie zapytania SQL

Kod generujący zapytanie SQL znajduje się w metodzie `toString()` klasy, która tworzy pełne zapytanie na podstawie danych przechowywanych w obiekcie (rysunek 21).

```
@Override
public String toString() {
    String returnetString="Select ";
    if(getColumnName().isEmpty())
        returnetString+="* ";
    else {
        returnetString+=getColumnsString()+" ";
    }

    returnetString+="from "+getTableName()+" ";
    for (JoinClass join: joins) {
        returnetString+= join.toString()+" ";
    }
    returnetString+=(!getWhereString().isEmpty())?getWhereString():"";
    returnetString+=(!orderby.equals("none")?" order by "+orderbyColumn+" "+orderby:"");
    returnetString+=" limit "+limit;
    return returnetString;
}
```

Rysunek 21 Generowanie zapytania SQL

Metoda `toString()` wykonuje następujące operacje:

1. Inicjalizacja zapytania:

- o `String returnetString = "Select ";` - inicjalizuje zmienną, która będzie przechowywać zapytanie SQL, zaczynając od słowa kluczowego "Select".

2. Dodanie kolumn:

- o `if (getColumnName().isEmpty()) returnetString += "* "; else returnetString += getColumnsString() + " ";` - sprawdza, czy lista kolumn jest pusta. Jeśli tak, dodaje "*", co oznacza wybór wszystkich kolumn. W przeciwnym razie dodaje nazwy wybranych kolumn.

3. Dodanie nazwy tabeli:

- o `returnetString += "from " + getTableName() + " ";` - dodaje słowo kluczowe "from" oraz nazwę tabeli, z której mają być pobrane dane.

4. Dodanie klauzul JOIN:

- o `for (JoinClass join : joins) returnetString += join.toString() + " ";` - iteruje przez listę obiektów `JoinClass` i dodaje odpowiednie klauzule JOIN do zapytania.

5. Dodanie klauzuli WHERE:

- o `returnetString += (!getWhereString().isEmpty()) ? getWhereString() : " ";` - sprawdza, czy istnieją warunki klauzuli WHERE. Jeśli tak, dodaje je do zapytania.

6. Dodanie klauzuli ORDER BY:

- o `returnetString += !orderby.equals("none") ? "order by " + orderbyColumn + " " + orderby : "";` - sprawdza, czy użytkownik wybrał opcję sortowania. Jeśli tak, dodaje odpowiednią klauzulę ORDER BY.

7. Dodanie limitu:

- o `returnetString += " limit " + limit;` - dodaje limit wyników do zapytania.

8. Zwrócenie zapytania:

- o `return returnetString;` - zwraca pełne zapytanie SQL jako ciąg znaków.

Metoda `toString()` zapewnia dynamiczne generowanie zapytania SQL na podstawie atrybutów obiektu. Dzięki temu możliwe jest elastyczne tworzenie zapytań SELECT, które mogą zawierać różne kolumny, klauzule JOIN, WHERE, ORDER BY oraz limit wyników, w zależności od potrzeb użytkownika.

4.5 Wykonanie zapytania SQL

Kod przedstawia metodę wykonującą zapytanie SQL na bazie danych. Metoda ta przyjmuje zapytanie SQL w postaci ciągu znaków, a następnie wykonuje je, zwracając wyniki w odpowiedzi (rysunek 22).

```
public ResponseEntity<ExequteResponse> execute(String sql) {
    try {
        Statement stmt = connection.createStatement();
        stmt.execute( sql: "use "+params.getName()+";");
        sql=sql.replace( target: "\\n", replacement: "");
        List<Map<String,String>>rows=new ArrayList<>();
        Map<String,String>row=new HashMap<>();
        if(sql.trim().toLowerCase().startsWith("select")){
            ResultSet s=stmt.executeQuery( sql: sql+";");
            ResultSetMetaData md = s.getMetaData();
            while (s.next()) {
                row=new HashMap<>();
                for(int i=1;i<=md.getColumnCount();i++) {
                    row.put(md.getColumnName(i), s.getString(i));
                }
                rows.add(row);
            }
        }
        else{
            stmt.execute(sql);
            row.put("not", "select");
            rows.add(row);
        }

        return ResponseEntity.ok(new ExequteResponse( status: true,rows));
    } catch (SQLException e) {
        System.out.println(e);
        Map<String,String>res = new HashMap<>();
        res.put("statusText",e.toString());
        List<Map<String,String>> rpp=new ArrayList<>();
        rpp.add(res);
        return ResponseEntity.ok(new ExequteResponse( status: false, rpp));
    }
}
```

Rysunek 22 Wykonanie zapytania na bazie danych

Metoda `select` wykonuje następujące operacje:

1. Tworzenie obiektu `statement`:

- o `Statement stmt = connection.createStatement();` - Tworzy obiekt `Statement` używany do wykonania zapytania SQL.

2. Wybór bazy danych:

- o `stmt.execute("use " + params.getName() + ";");` - Wykonuje zapytanie `USE`, aby wybrać odpowiednią bazę danych na podstawie parametrów połączenia.

3. Czyszczenie zapytania SQL:

- o `sql = sql.replace("\\n", "");` - Usuwa znaki nowej linii z zapytania SQL.

4. Inicjalizacja struktury wyników:

- o `List<Map<String, String>> rows = new ArrayList<>();` - Tworzy listę `map`, gdzie każda mapa reprezentuje wiersz wyników.
- o `Map<String, String> row = new HashMap<>();` - Inicjalizuje mapę do przechowywania danych pojedynczego wiersza.

5. Wykonanie zapytania `SELECT`:

- o `if (sql.trim().toLowerCase().startsWith("select")) { ... }` - Sprawdza, czy zapytanie jest zapytaniem `SELECT`.
- o `ResultSet s = stmt.executeQuery(sql + ";");` - Wykonuje zapytanie `SELECT` i otrzymuje wynik w postaci `ResultSet`.
- o `ResultSetMetaData md = s.getMetaData();` - Pobiera metadane wyników, aby uzyskać informacje o kolumnach.
- o `while (s.next()) { ... }` - Iteruje przez wyniki, wypełniając mapę `row` nazwami kolumn i ich wartościami, a następnie dodaje mapę do listy `rows`.

6. Wykonanie innych zapytań SQL:

- o `else { ... }` - Jeśli zapytanie nie jest zapytaniem `SELECT`, wykonuje je bez oczekiwania na wynik.
- o `stmt.execute(sql);` - Wykonuje zapytanie.
- o `row.put("not", "select");` - Dodaje informację, że zapytanie nie było `SELECT`.
- o `rows.add(row);` - Dodaje mapę do listy wyników.

7. Zwracanie odpowiedzi:

- o `return ResponseEntity.ok(new ExequiteResponse(true, rows));` - Zwraca odpowiedź z wynikami zapytania w obiekcie `ExequiteResponse`.

8. Obsługa wyjątków:

- o `catch (SQLException e) { ... }` - Obsługuje wyjątki SQL.
- o `System.out.println(e);` - Wypisuje wyjątek na konsolę.
- o `Map<String, String> res = new HashMap<>();` - Tworzy mapę do przechowywania informacji o błędzie.
- o `res.put("statusText", e.toString());` - Dodaje informację o błędzie do mapy.

- o `List<Map<String, String>> rpp = new ArrayList<>();` - Tworzy listę z mapą błędu.
- o `rpp.add(res);` - Dodaje mapę błędu do listy.
- o `return ResponseEntity.ok(new ExeecuteResponse(false, rpp));` - Zwraca odpowiedź z informacją o błędzie w obiekcie `ExeecuteResponse`.

Metoda `select` umożliwia wykonanie zapytań SQL i zwraca ich wyniki w odpowiednim formacie, obsługując zarówno zapytania `SELECT`, jak i inne zapytania SQL, takie jak `INSERT`, `UPDATE`, `DELETE`.

4.6 MySQL w kontenerze Docker

Dane są wysyłane do relacyjnej bazy danych MySQL, która jest uruchomiona w kontenerze Docker. Przedstawiony przykład wykonania zapytania `SELECT` w terminalu Dockera, ilustruje interakcję z bazą danych MySQL (rysunek 23).

```
mysql> select * from City;
+-----+-----+-----+-----+-----+
| ID | CityName | Area | Voivodeship | Population |
+-----+-----+-----+-----+-----+
| 1 | Radom | 111.8000 | Mazowieckie | 213715.0 |
| 2 | Kielce | 110.0000 | ?wi?tokrzyskie | 195942.0 |
| 4 | Krakw | 326.8500 | Ma?opolskie | 766683.0 |
| 5 | Sosnowiec | 91.1600 | ?l?skie | 190000.0 |
+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> 
```

Rysunek 23 Wykonanie zapytania `select` w terminalu Dockera

4.7 Odebranie danych na frontendzie

Dane odebrane z backendu są ustawiane w stanie komponentu przy użyciu hooka `useState`. Następnie są one wyświetlane w formie tabeli, opisanej w punkcie 3.7 (rysunek 24).

```
setSelectResults({
  status: responseData.status,
  rows: responseData.rows,
})

if (responseData.status === false) {
  setSqlError(responseData.rows.statusText)
} else {
  handleModalClose()
  handleSelectResultModalOpen()
}
```

Rysunek 24 Odebranie danych na frontendzie

Dane zwrócone przez backend są ustawiane w stanie komponentu `selectResult` przy użyciu funkcji `setSelectResult`. Funkcja `setSqlCode` służy do ustawienia kodu SQL, który został wygenerowany, a `handleModalOpen` otwiera modal z danymi.

Dane przechowywane w `selectResult` są następnie wykorzystywane do wyświetlenia tabeli, zgodnie z opisem w punkcie 3.7 (rysunek 17). Tabela wyświetla wyniki zapytania w czytelny sposób, umożliwiając użytkownikowi przeglądanie wyników bezpośrednio w aplikacji.

PODSUMOWANIE

Dokumentacja narzędzia MySQL Designer szczegółowo przedstawia proces jego tworzenia, architekturę oraz funkcjonalności, które zostały zrealizowane. Narzędzie to, stworzone z wykorzystaniem nowoczesnych technologii, takich jak React dla frontendu i Spring Boot dla backendu, umożliwia łatwe i intuicyjne zarządzanie bazami danych MySQL. Jego główne funkcjonalności obejmują możliwość nawiązywania połączenia z bazą danych, przeglądanie i zarządzanie tabelami, a także wykonywanie zapytań SQL typu SELECT, INSERT, UPDATE oraz DELETE.

Aplikacja została zaprojektowana z myślą o prostocie użytkowania, co przejawia się w przejrzystym interfejsie użytkownika. Użytkownik może nawigować pomiędzy różnymi sekcjami, takimi jak Database, Table oraz Operation, co umożliwia szybkie przechodzenie pomiędzy różnymi funkcjami narzędzia. W sekcji Database można nawiązać połączenie z bazą danych poprzez wprowadzenie danych logowania i adresu serwera. Sekcja Table pozwala na dodawanie, edytowanie, usuwanie oraz przeglądanie tabel w bazie danych.

Jednym z kluczowych elementów narzędzia jest dynamiczne generowanie formularzy do tworzenia zapytań SQL. W sekcjach Insert Query, Update Query, Delete Query oraz Select Query użytkownik może wprowadzać szczegółowe parametry zapytań, które następnie są wysyłane do backendu, przetwarzane i wykonywane na bazie danych. Wyniki zapytań SELECT są zwracane do frontendu i prezentowane w czytelnej formie tabeli, co pozwala na łatwe przeglądanie danych.

W procesie tworzenia narzędzia uwzględniono najlepsze praktyki związane z architekturą aplikacji webowych. Frontend komunikuje się z backendem za pomocą REST API, co zapewnia elastyczność i skalowalność. Backend, napisany w Javie z wykorzystaniem Spring Boot, odpowiada za przetwarzanie danych oraz interakcję z bazą danych MySQL, która jest hostowana w kontenerze Docker. Taka architektura zapewnia modularność i łatwość utrzymania aplikacji.

Cały proces implementacji został dokładnie opisany, począwszy od tworzenia interfejsu użytkownika, poprzez kodowanie logiki aplikacji, aż po obsługę bazy danych. Przedstawiono również przykłady kodu odpowiedzialnego za wysyłanie danych z frontendu do backendu, ich przetwarzanie oraz zwracanie wyników. Dzięki temu dokumentacja stanowi cenne źródło wiedzy dla przyszłych deweloperów, którzy mogą rozwijać i modyfikować narzędzie zgodnie z własnymi potrzebami.

Podsumowując, wszystkie założone cele projektu zostały osiągnięte. Stworzono funkcjonalne narzędzie do zarządzania bazami danych MySQL, które jest łatwe w użyciu i oferuje szeroki zakres możliwości. MySQL Designer to nowoczesne rozwiązanie, które z pewnością znajdzie zastosowanie w codziennej pracy z bazami danych, ułatwiając i automatyzując wiele czynności związanych z zarządzaniem danymi. Dzięki zastosowaniu

nowoczesnych technologii oraz starannemu podejściu do implementacji, narzędzie to spełnia wysokie standardy jakości i wydajności.

SPIS RYSUNKÓW

Rysunek 1 Schemat ilustrujący ustanawianie relacji między tabelami.....	5
Rysunek 2 Logo relacyjnej bazy danych MySQL	9
Rysunek 3 Logo relacyjnej bazy danych PostgreSQL	9
Rysunek 4 Logo relacyjnej bazy danych Oracle	10
Rysunek 5 Logo relacyjnej bazy danych Microsoft SQL Server	10
Rysunek 6 Logo relacyjnej bazy danych SQLite	11
Rysunek 7 Schemat architektury aplikacji MySQL Designer	14
Rysunek 8 Pasek nawigacji w aplikacji MySQL Designer.....	15
Rysunek 9 Formularz połączenia się z bazą danych	16
Rysunek 10 Formularz tworzenia nowej tabeli.....	18
Rysunek 11 Lista tabel	19
Rysunek 12 Formularz edycji tabeli.....	19
Rysunek 13 Formularz tworzenia zapytania insert	21
Rysunek 14 Formularz tworzenia zapytania update	22
Rysunek 15 Formularz tworzenia zapytania delete.....	23
Rysunek 16 Formularz tworzenia zapytania select	25
Rysunek 17 Wyniki zapytania select przedstawione w postaci tabelarycznej	25
Rysunek 18 Implementacja metody <code>handleSubmit</code> w komponencie <code>SelectQueryForm</code>	27
Rysunek 19 Implementacja punktu końcowego obsługujący zapytanie typu POST	28
Rysunek 20 Zamiana JSON na obiekt.....	29
Rysunek 21 Generowanie zapytania SQL.....	31
Rysunek 22 Wykonanie zapytania na bazie danych	32
Rysunek 23 Wykonanie zapytania <code>select</code> w terminalu Dockera	34
Rysunek 24 Odebranie danych na frontendzie	34

BIBLIOGRAFIA

- [1] https://pl.wikipedia.org/wiki/Edgar_Frank_Codd, [02.05.2024].
- [2] Steve Suehring, *“MySQL Bible”*. Wiley Publishing, Inc. 2002, New York.
- [3] Beata Pańczyk, Arkadiusz Sławiński, *“Technologie mapowania obiektowo-relacyjnego w aplikacjach PHP”*. Politechnika Lubelska, Wydział Elektrotechniki i Informatyki, 2015, Lublin.
- [4] Kevin Yank, *„PHP i MySQL Od nowicjusza do wojownika ninja”*. Helion S.A., 2012, Gliwice.
- [5] Maria Chałon, *„Ochrona i bezpieczeństwo danych oraz tendencje rozwojowe baz danych”*. Oficyna Wydawnicza Politechniki Wrocławskiej, 2007, Wrocław.
- [6] Katarzyna Kroc, Oleksandra Kizun, Maria Skublewska-Paszkowska, *„Analiza porównawcza wydajności relacyjnych baz danych MySQL, PostgreSQL, MariaDB oraz H2”*. Politechnika Lubelska, Katedra Informatyki, 2019, Lublin.