POLSKO-JAPOŃSKA WYŻSZA SZKOŁA TECHNIK KOMPUTEROWYCH

Wydział Informatyki

Katedra Inżynierii Oprogramowania

Inżynieria Oprogramowania i Baz Danych

Maciej Dziubak
Nr albumu 4236

Elastyczny interfejs użytkownika do współpracy z bazą danych

Praca magisterska napisana
pod kierunkiem:

dr inż. Mariusza Trzaski

Warszawa, czerwiec 2011

Streszczenie

W niniejszej pracy skupiono się na analizie aplikacji zapewniających dostęp i obsługę bazy danych, niezależnie od układu i struktury danych. Szczególną uwagę zwrócono na komponenty do edycji rekordów bazy danych. Wyszczególniono cechy tych programów i w po przeprowadzeniu dyskusji rozwiązań zaproponowano model, który rozszerzałby w elastyczny sposób aplikacje o nową funkcjonalność oraz pozwoliłby na stworzenie edytora baz danych dostosowanego do indywidualnych potrzeb użytkownika. Szczególnie zwrócono uwagę na zapewnienie obsługi wielu typów danych multimedialnych oraz rozszerzonej walidacji danych, zgodnie z przeznaczeniem danego rekordu.

W pracy czytelnik zostanie wprowadzony w zagadnienia pisania programów do obsługi bazy danych, opartych na meta-danych (danych o danych). Następnie zaprezentowano prototyp rozwiązania edytora zawartości bazy danych i jej struktury opartego na powstałym modelu. Na jego podstawie aplikacja dynamicznie tworzy zapytania do bazy danych. W rozwiązaniu zwrócono szczególną uwagę na obsługę zawartości multimedialnych, takich jak pliki mp3, obrazy JPG, pliki RTF.

Spis treści

1. Wstęp

Tradycyjne podejście projektowania aplikacji opartych o bazę danych zakłada pisanie aplikacji bezpośrednio pod określoną strukturę baz danych i znane użytkownikowi typy danych zawartych w bazie. Zapytania do bazy danych są konstruowane pod konkretny model danych, zdefiniowany wcześniej przez programistę.

Zdecydowanie trudniejsze jest zagadnienie, które dotyczy nieznanego modelu danych oraz ich struktur. Wymaga ono stworzenia systemu odpytującego bazę o potrzebne dane do dynamicznego kreowania zapytań oraz dynamicznego mapowania typów bazy danych na typy języka programowania. Aplikacje te muszą być uniwersalne. Mają jednak zazwyczaj ograniczoną formę prezentacji danych.

Rozwój sieci Internet w ostatnich latach, zapoczątkował potrzebę rozwoju baz danych w stronę przechowywania coraz większej ilości typów multimedialnych. Wymaga to dostarczenia odpowiednich narzędzi do obsługi takich danych.

W niniejszej pracy zaprezentowano aplikację niezależną od struktury danych i modelu zawartego w bazie. Przedstawiono rozwiązanie do edycji zawartości bazy danych z możliwością zmiany jej struktury oraz dodatkowo zaproponowano rozszerzalny model prezentacji danych.

1.1. Cel Pracy

Celem pracy jest określenie cech oraz dyskusja rozwiązań dotyczącej aplikacji do zarządzania bazami danych. Praca szczególnie koncentruje się na analizie modułów do modyfikacji danych zawartych w bazie. Zaproponuje model intuicyjnej reprezentacji danych konkretnego typu z możliwością rozszerzenia funkcjonalności za pomocą dodatków.

Przedstawi prototyp takiego rozwiązania oraz zwróci uwagę na praktyczne problemy powstałe podczas projektowania oraz implementowania aplikacji.

1.2. Rozwiązania przyjęte w pracy

Przedstawione rozwiązania wprowadzają nowe podejście do aplikacji zarządzającymi zawartościom bazy danych. Prezentuję model, który elastycznie rozszerzałby funkcjonalność za pomocą dodatków (ang. plugin). Rozwiązanie zapewnia możliwości prezentacji danych w intuicyjny sposób oraz edycję w sposób odpowiedni dla danego typu danych.

W celu zaprezentowania rozwiązań użyto środowiska programistycznego Java SE w wersji 6. Baza danych użyta w projekcie to MySQL w wersji 5.1.

1.3. Rezultaty pracy

Głównym rezultatem pracy będzie przedstawienie wymagań funkcjonalnych, założeń analitycznych i propozycja architektury aplikacji, która w sposób elastyczny dostarczy narzędzi do edycji zawartości bazy danych w tym obsługi różnorodnych typów danych.

Pośrednim wynikiem pracy będzie prototyp aplikacji, która wdroży główne założenia powyższej analizy, jak również opis doświadczeń i dyskusja problemów implementacyjnych pojawiających się w trakcie programowania prototypu.

1.4. Organizacja Pracy

W pracy na początku opisano zagadnienia tworzenia i historii graficznego interfejsu użytkownika oraz charakterystykę baz danych, szczególnie relacyjnych baz danych. Wyszczególniono dobre praktyki tworzenia graficznego interfejsu użytkownika oraz konieczną funkcjonalność do obsługi baz danych.

Następnie przedstawiono aplikacje, zarządzające bazami danych. Wyszczególniono ich funkcjonalność oraz zalety i wady. Zwrócono uwagę na rozwiązania prezentacji danych. Informacje te posłużą za podstawę do dyskusji rozwiązań oraz przeprowadzenia analizy aplikacji, która zapewniałaby elastyczną obsługę bazy danych, szczególnie nastawianą na zarządzanie różnorodnymi typami danych w tym multimedialnych.

W dalszej części praca pokazano autorskie rozwiązanie w formie aplikacji, opisujące technologie wykorzystane przy tworzeniu prototypu, oraz wprowadzono użytkownika w

architekturę rozwiązania. Potem zaprezentowano interfejs graficzny i przedstawiono poszczególne funkcje programu.

Na koniec opisano problemy powstałe podczas implementowania rozwiązania i przedyskutowano możliwości rozbudowy funkcjonalności.

2. Wprowadzenie do projektowania graficznego interfejsu użytkownika oraz opis charakterystyki baz danych.

Graficzny interfejs użytkownika (graphical user interface) w skrócie (GUI), to nazwa sposobu interakcji za pomocą komponentów graficznych pomiędzy użytkownikiem a maszyną, czyli najczęściej komputerem. Zazwyczaj są to ikony, pola tekstowe, listy rozwijalne lub etykiety. Działania realizowane są zwykle poprzez bezpośrednie manipulowanie elementami graficznymi. Interfejs ten wyparł wcześniej stosowany CLI (ang. Command Line Interface) – czyli wpisywanie komend w linii poleceń.

Pierwowzór GUI narodził się w instytucie badawczym uniwersytetu Stanford, dzięki badaniom prowadzonym przez Douglasa Engelbarta. Stworzył On i jego grupa koncepcje używania hiperłączy tekstowych, obsługiwanych za pomocą myszy do wykonywania operacji w systemie on-line.

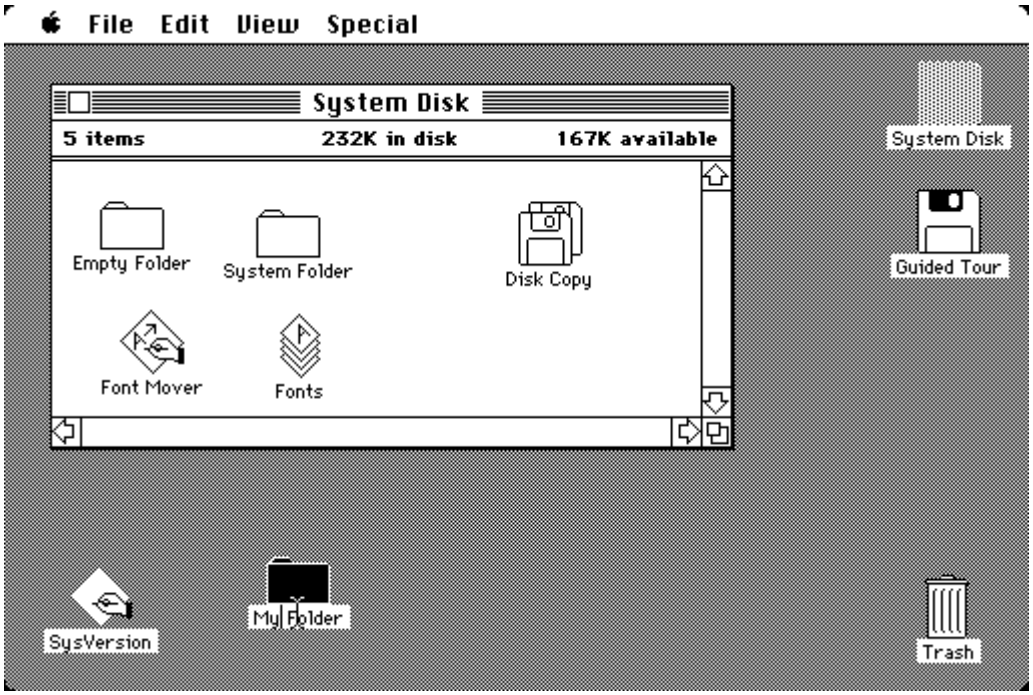
Koncepcja hiperłączy została udoskonalona przez naukowców z Xerox PARC (Palo Alto Research Center) . Zamiast hiperłączy wprowadzili oni elementy graficzne. Stworzyli oni pierwszy interfejs graficzny dla komputera osobistego Xerox Alto o nazwie Xerox Star. Większość nowoczesnych interfejsów użytkownika pochodzi właśnie z tego systemu.

Firma Xerox nie zdawała sobie sprawy z potencjału leżącego w pracy inżynierów z firmy PARC. W 1979 umożliwiła dostęp do ich pracy Stevowi Jobsowy z firmy Apple. Apple, jako akcjonariusz Xerox uzyskał bezpłatny dostęp do powyższych prac. Steve Jobs prezes Apple był pod wielkim wrażeniem nowego wynalazku. Stworzyli oni pierwszy ogólnodostępny komputer z graficznym systemem operacyjnym Apple Lisa.

W roku 1980 koncepcje zawarte w systemie Xerox Star zostały nazwane przez Merzouga Wilberts terminem WIMP (Window , Icon , Menu and Pointers). W tłumaczeniu (okna, ikony, menu i wskaźniki). Koncepcja ta określa sposób prezentacji zawartości graficznej na ekranie komputera.

W górnej części ekranu na wąskim pasku znajdowało się menu rozwijalne i opisywało one możliwe operacje. W pozostałej części można było swobodnie operować oknami lub ikonami za pomocą wskaźnika, jakim najczęściej była mysz komputerowa - również

wynalazek firmy Xerox. Przykładem takiego interfejsu może być zaprezentowany na Rys 2.1.1 interfejs komputera Macintosh.



Rys 2.1.1 Interfejs Użytkownika w paradygmacie WIMP (Macintosh z roku 1984) źródło: Wikipedia

Następnie bardzo wiele firm zaczęło wprowadzać koncepcje WIMP do swoich produktów. W latach osiemdziesiątych głównym liderem była firma Apple, jednak zaczęły pojawiać się inne interfejsy. W 1985 firma Comodore w komputerze Amiga zaprezentowała środowisko o nazwie Workbench.

W roku 1982 firma Microsoft, późniejszy największy konkurent firmy Apple zaczęła prace nad nakładkami graficznymi na system operacyjny MS-DOS. Prace nad nakładką trwały aż do roku 1985 i niestety nie stała się ona popularna.

Większą popularność zdobyła natomiast nakładka Graphical Environment Manager (GEM) firmy Digital Research (DRI). Była to nakładka na istniejące systemy operacyjne-CP/M i MS-DOS kompatybilna z komputerami firmy IBM. Powstała ona na bazie wcześniejszego oprogramowania firmy DRI o nazwie GSX zaprojektowanego przez byłego pracownika zespołu XEROX PARC. Wyglądała o wiele lepiej niż Windows przede wszystkim dlatego, że powielała wiele rozwiązań z interfejsu Macintoshy, na przykład kosz (Microsoft wprowadził tą koncepcję dopiero w Windows 95) i ogólną interakcję z pulpitem

m.in. drag and drop. Nakładka GEM dała później początek systemom operacyjnym firmy ATARI.

Firma Microsoft długo ulegała konkurencji. Sytuacja zaczęła się zmieniać, gdy w styczniu 1987 na rynku pojawił się program Aldus PageMaker w wersji dla Windows. Był on pierwszym programem do składu publikacji (DTP) zrealizowanym w technologii WYSIWYG dla platformy PC. Niektórzy historycy świata komputerów uważają, że pojawienie się tego programu to początek sukcesu Microsoft Windows.

Znaczące dla firmy Microsoft było wydanie 32 bitowej nakładki na MS- DOS o nazwie Windows 3.11. Zyskał on popularność dzięki pojawieniu się oprogramowania DTP Corel i arkusza kalkulacyjnego Excel.

Krokiem milowym dla firmy Microsoft było wydanie w roku 1995 systemu w pełni okienkowego Windows 95. Windows 95 okazał się przełomem, w pełni wykorzystywał możliwości interfejsu i całkowicie pokrywał funkcjonalność sytemu MS-DOS. Był przy tym przyjazny dla użytkownika nieznającego się na komputerach. Kolejne wersje i systemy dodawały funkcjonalności oraz poprawiały wygląd samego systemu.

Koncepcja WIMP obecna jest w systemach praktycznie do dziś. Nawet koncepcja systemu „Windows 7" jest rozszerzeniem interfejsu Windows XP poprzez nałożenie na niego komponentu Windows Aero używającego zalety technologii „pixel shader" zastosowanej powszechnie w przemyśle trójwymiarowych gier komputerowych. Windows Aero zapewnia zaawansowane efekty wizualne dla sytemu operacyjnego między innymi przezroczystość ramek.

2.1. Metody budowania graficznego interfejsu użytkownika

2.1.1 Ręczne pisanie kodów

Programowanie ręczne oparte jest na użyciu API (ang. Application Programming Interface) interfejsu programistycznego danej biblioteki np. Swing lub SWT dla języka Java.

Pomimo, iż programista ma pełną kontrolę nad wyglądem interfejsu, to zaprojektowanie dobrze wyglądającej aplikacji wymaga dużo czasu. Powodem istniejącego stanu rzeczy jest specyfika niektórych komponentów, które często zachowują się inaczej w interakcji z różnymi rodzajami kontrolek.

2.1.2 Projektowanie wizualne

Projektowanie za pomocą aplikacji (nakładek na środowiska programistyczne), umożliwiających „narysowanie" graficznego interfejsu za pomocą edytora, użyciem technologii przeciągania komponentów (ang. Drag and Drop), w myśl idei „uzyskasz to, co widzisz" (ang. What You See Is What You Get). W rezultacie otrzymujemy wygenerowany kod reprezentujący dokładne odwzorowanie widoku, który użytkownik widzi na ekranie swojego komputera.

Taka metoda posiada swoje zalety i wady. Przede wszystkim użytkownik otrzymuje rezultaty bardzo szybko. Niestety często generatory generują bardzo dużą ilość tzw. „brudnego kodu" czyli kodu, który uniemożliwia dalszą jego edycję poprzez programistę. Jest on projektowany według reguł, które generują bardzo dużo zbędnych ograniczeń lub elementów. Mnogość tych elementów powoduje, że kod jest nieprzydatny.

Warto zwrócić uwagę, że niektóre programy generują kod i całkowicie zakazują jego edycji, gdyż wszelkie zmiany i tak będą nadpisane przez generator. Przykładem takiego rozwiązania jest edytor graficzny zawarty w środowisku programistycznym Netbeans. Istnieje oczywiście też inna grupa edytorów, która odzwierciedla również ręczne modyfikacje kodu, pracująca według tak zwanej inżynierii wahadłowej (ang. round- trip)

2.1.3 Podejście deklaratywne

Ogólnie podejście deklaratywne do problemu programistycznego oznacza, że nie określa się schematu postępowania przy wdrażaniu rozwiązania. Opisuje się jedynie rezultaty, które chce się otrzymać. Programista opisuje komponenty, jakie chce uzyskać w specjalnym języku programowania - języku opisu.

2.1.4 Podsumowanie

Różnorodność i sposoby tworzenia graficznego języka programowania stwarzają konieczność dostosowania się do użytkowników i zapewnienia im jak największej swobodności przy wyborze metody tworzenia graficznego interfejsu użytkownika. Należy więc programować aplikacje w sposób, który umożliwiałby edycję samego wyglądu aplikacji, a nie zmieniał jej logiki. Sposób jest dobrą praktyką programistyczną i jest realizowany w ramach wzorca projektowego MVC (Model-View-Controler).

2.2. Wzorzec projektowy MVC

Wzorzec projektowy Model-View-Controller (pol. Model-Widok-Kontroler) to wzorzec do organizowania struktury aplikacji posiadających graficzne interfejsy użytkownika, polegający na separacji fragmentów kodu pomiędzy części:

- model - określa dane związane z komponentem lub stany komponentu,
- widok (view) - określa wizualną reprezentację danych lub stanów,
- sterownik (controller) - zapewnia interakcję użytkownika z widokiem i wynikające stąd modyfikacje modelu.

Separacja kodu na powyższe części zapewnia możliwość zmiany poszczególnych elementów nie wpływając na kompatybilność z pozostałymi komponentami.

Wzorzec MVC może występować w kilku wersjach. Pierwsze implementacje zapewniały komunikacje miedzy trzema częściami, każdy z każdym. W nowocześniejszych implementacjach odchodzi się od takiego podejścia. Rolę komunikowania między widokiem a modelem w pełni przejmuje kontroler.

2.3. Historia baz danych i ich charakterystyka

Początek historii baz danych sięga dziewiętnastego wieku i jest związany jest z wynalezieniem kart perforowanych, oraz urządzeń mechaniczno-elektrycznych, które z tych kart korzystały.

„Pierwszymi wynalazkami tego typu było np. krosno Josepha Jacquarda, sterowane za pomocą zapisu na karcie perforowanej, pianole, które odtwarzały utwory zakodowane na takich kartach, czy maszyna analityczna Charlesa Babbage'a[1], która miała być sterowana za pomocą instrukcji odczytywanych z kart perforowanych" -[3]

Prawdziwym przełomem było jednak użycie wynalazku Hermana Holleritha podczas spisu ludności w 1890r.. Wynalazł on „Tabulator" maszynę umożliwiająca odczytywania danych demograficznych z kart perforowanych, sumowanie ich zawartości i drukowanie raportów. Wynalazek ten skrócił czas spisu ludności z 8 lat do jednego roku.

[1] Maszyna ta nigdy nie powstała była tylko projektem

W roku 1896 Hollerith założył firmę Tabulating Machine Company, która zajmowała się maszynami przetwarzającymi karty perforowane. Następnie w roku 1911 firma ta połączyła się z trzema innymi korporacjami zmieniając nazwę na Computing Tabulating Recording Corporation. W roku 1924 pod zarządem Thomasa J. Watsona przyjęła nazwę International Bussiness Machines (IBM).

W latach 40-tych dwudziestego wieku powstały pierwsze komputery. Początkowo zajmowały duże rozmiary i były wykorzystywane tyko do obliczeń naukowych. W dziedzinie przetwarzania danych wprowadzono natomiast wynalazek taśm magnetycznych. Taśma magnetyczna mogła pomieścić tyle danych, co dziesięć tysięcy kart perforowanych. Pionierem tej technologii była firma Univac. W roku 1951 ta sama firma stworzyła pierwszy komercyjny komputer ogólnego przeznaczenia UNIVAC1. Nowe komputery były mniejszych rozmiarów od konkurencyjnych maszyn (tabulatorów).

Były programowalne (za pomocą taśm) w odróżnieniu od sterownych obwodami maszynami IBM. Zaczęły pojawiać się języki programowania do operacji na danych między innymi FORTRAM, LISP, COBOL, RPG. W oprogramowaniu wprowadzono wtedy koncepcje przetwarzania danych zorientowanych na pliki.

Ówczesne systemy plików umożliwiały tylko sekwencyjny dostęp do danych. Przez co interakcja z komputerem odbywała się w systemie wsadowym i trwała długo. Takie rozwiązania nie pasowały do potrzeb dziedzin takich jak np. rynek papierów wartościowych, czy obsługa biur podróży.

Następnym krokiem rozwoju było opracowanie monitorów CRT i rozwój dysków magnetycznych. Dyski magnetyczne umożliwiły swobodny dostęp do danych. Rozwinęły się systemy plików, które przenosiły adres logiczny plików na adres (fizyczny) sektorów dysków.

Kolejnym osiągnięciem były komputery z wieloma terminalami dostępu. Zapoczątkowały one pierwsze systemy baz danych. Pierwszy system baz danych o nazwie IMS został opracowany dla środowiska MVS (Multiply Virtual Storage). Charakteryzował się budową hierarchiczną. Jego pierwsza wersja (IMS 360 Version 1) została wydana w 1968 roku. Budowa hierarchiczna była w tamtych czasach intuicyjnym rozwiązaniem, ponieważ pozwalała na wykorzystywanie urządzeń z sekwencyjnym dostępem do danych. Wiele

urządzeń stosowało to rozwiązanie. IMS wprowadziło także zasadę, która oddzielała dane od aplikacji, tworząc tzw. centralny mechanizm składowania danych.

Były również wady systemu hierarchicznego przechowywania danych, takie jak:

- redundancje (powtórzenia) tego samego rekordu w wielu hierarchiach,
- utrudniona edycja danych, usuwania oraz dodawanie rekordów do wielu hierarchii.

Wady te przyczyniły się do powstania nowego modelu przechowania danych – modelu sieciowego, który został zaimplementowany w systemie Integerated Data Storage (IDS) w Firmie General Electric. Jego Autorem był Charles Bachman. Model ten wyróżniała zamiana hierarchii w graf (każda informacja mogła zawierać połączenia z wieloma „rodzicami" i wieloma „potomkami").

W latach sześćdziesiątych grupa programistów z firmy CODALSYS zaczęła rozbudować język programowania Cobol o sieciowy model danych. Grupa ta opracowała dwa języki:

- język definicji danych (ang. data definition language – DDL), ·definiował struktury danych.
- język manipulacji danymi (ang. data manipulation language – DML), ·do zarządzania zawartością danych.

Ponadto zmieniono sposób koncepcji opisu danych zawartych w bazie wprowadzając następujące schematy:

- schemat logiczny- opisuje logiczną organizację całej bazy.
- podschemat – określa fragmenty bazy widoczne przez użytkownika
- schemat fizyczny – opisuje rozmieszczenie danych na nośnikach.

Systemy baz danych musiały rozwiązać wiele trudności związanych z dostępem zdalnym, szczególnie związanych z współbieżnym dostępem do danych. Zaproponowano koncepcję wprowadzania zmian za pomocą transakcji, czyli zapisanie wielu zmian w bazie jako atomowej jednostki (zatwierdzenie wszystkich zmian lub żadnej). Transakcje blokowały dostęp do bazy dla innych użytkowników. Dodatkowo wprowadzono, też system zabezpieczający zwany dziennikiem zmian. Umożliwiało to wycofywanie zmian transakcji, które się nie powiodły tzw. Rollback.

System sieciowy wymagał jednak pisania skomplikowanych programów nawigujących po bazie (zapisanej w formie grafu).

Rewolucje uwalniającą programistów od pisania skomplikowanych programów zapoczątkował artykuł „A Relational Model of Data for Large Shared Data Banks" E. F. Codda, opisujący nowy model baz danych zwany relacyjnym. Pomysły zawarte w artykule okazały się rewolucyjne. Dane przechowywane są, w tabelach dwuwymiarowych zwanych relacjami. Związki między tabelami zapisane są przez atrybuty w sposób niejawny. Zawierają kolumny, w których zazwyczaj jedna (czasem wiele kolumn) są kluczem głównym tzn. jednoznacznie identyfikuje dany rekord. Klucz ten jest używany do identyfikacji danego rekordu w innych tabelach tworząc związek.

Wynalazek Edwarda Codda zainspirował świat naukowy do rozwoju zagadnień związanych z bazami danych. Powstał między innymi język zapytań wysokiego poziomu SQL(Structured Query Language) znany i używany do dziś. Powstała też teoria normalizująca struktury baz danych pozwalająca uniknięcia błędów i redundancji w bazie.

System relacyjny oferował wiele udogodnień, przez co jego popularność wyparła z rynku rozwiązania sieciowe i hierarchiczne. Szerokie grono odbiorców na całym świecie, zdemaskowały wady podejścia relacyjnego.

„Jednym z podstawowym zarzutów była ograniczona liczba typów danych, które mogły być przechowywane w relacjach.(...)Próba rozszerzenia języka SQL o nowe typy danych takie jak: czas, interwał czasowy, znacznik czasowy, data, waluta oraz wiele różnych wariantów liczb i łańcuchów okazała się niewystarczająca. Zaistniała konieczność, ze względu na potrzeby nowych aplikacji, zapewnienia możliwości przechowywania danych o złożonych typach, takich jak np. obrazy, dźwięki albo mapy.” [3]

W 1985 roku społeczność akademicka zaczęła pracę nad nowymi modelami danych. W wyniku prac powstał nowy model obiektowy. Zakładał on:

- połączenie danych i operacji na nich wykonywanych (atrybuty i akcje),
- identyfikacje obiektów ,
- dziedziczenie, (przejęcie atrybutów i akcji innych zbiorów obiektów)
- typy abstrakcyjne i zagnieżdżone.

Niestety model obiektowy nie przyjął się na rynku. Przedsiębiorcy nie zaufali nowym rozwiązaniom. Przyczyniło się to do pomieszania relacyjnego modelu danych z obiektowym. Producenci baz danych zaczęli rozbudowywać swoje bazy o możliwości obiektowe. Ta mieszanka ma wiele wad, ale zalety mechanizmów takich jak transakcje, mechanizmy ochrony danych (dziennik powtórzeń), systemy zabezpieczeń przed utratą danych, powodują ciągle przywiązanie do relacyjnych baz danych.

2.4. Relacyjne bazy danych

W relacyjnych bazach danych dane zapisane są w tabelach dwuwymiarowych nazwanych relacjami. Każda relacja opisana jest przez zestaw atrybutów. Każdy atrybut zapisany jest w kolumnie. Atrybuty mają określoną dziedzinę wartości, zwaną typem danych. Każdy wiersz tabeli nazywany jest krotką tj. uporządkowany ciąg wartości.

Wymagane jest, aby wartość zapisana w pojedynczej komórce tabeli była atomowa (nie była zbiorem wartości). Wymagane jest również, aby dla każdej relacji istniał atrybut bądź zbiór atrybutów, których wartość (bądź kombinacja wartości) jednoznacznie identyfikuje każdą krotkę. Taki zbiór atrybutów nazywany jest nadkluczem.

Minimalny nadklucz, dla którego nie można usunąć ani jednego atrybutu bez utraty własności identyfikacji, nazywany jest kluczem. Jeżeli relacja posiada więcej niż jeden klucz, wyróżnia się jeden z nich, jako tak zwany klucz główny. Wyróżniamy w relacji również tak zwane klucze obce. Kluczem obcym nazywamy atrybut (bądź zbiór atrybutów), który przyjmuje wartości klucza innej relacji. Klucze obce reprezentują powiązania pomiędzy krotkami w relacjach, ale możliwe jest definiowanie ad-hoc innych warunków łączących krotki, w ramach wykonywania określonego zapytania skierowanego do bazy danych.

2.5. Identyfikacja podstawowej funkcjonalności interfejsów użytkownika do edycji systemów baz danych.

Systemy do edycji baz danych to zazwyczaj aplikacje przeznaczone dla administratorów baz danych. Umożliwiają one edycje struktury bazy lub szybką edycję istniejących danych zawartych w bazie. Głównym celem takich aplikacji jest dostrojenie bazy, pod aplikacje zbudowane na tych bazach lub dostosowanie istniejących danych. Są

bardzo przydatne w trakcie samego procesu projektowania aplikacji opartych na bazie. Przeglądanie zawartości bazy, ich edycja oraz wstępna inicjacja znacznie ułatwia pisanie aplikacji i umożliwia przeprowadzenie wstępnych testów aplikacji.

Aplikacje te zazwyczaj dostarczone są z istniejącymi systemami baz konkretnych producentów. Dzieje się to zazwyczaj, ponieważ wielu producentów rozszerza model relacyjny baz danych o własne funkcje i przyczynia się to do rezygnacji ze standaryzacji. Tworzy to systemy Relacyjno –obiektowe, które często są niezgodne z algebrą relacji, ponieważ tracą paradygmat atomowości danych w rekordzie. Producenci baz danych powinni wdrażać dokumenty związane ze standaryzacją swoich systemów. W praktyce wiele specyfikacji nie wdrożyło nawet standardu SQL:99. Prawdopodobnie producentom nie zależy na standaryzacji, ponieważ mając specyficzny system uzależniają swoich klientów od swoich usług (migracja będzie bardzo uciążliwa lub wręcz niemożliwa).

Aplikacje do obsługi baz danych wymagają dostarczenia jak największej ilości metadanych o samej bazie. Dane te będą wykorzystywane do pobierania danych o strukturze bazy oraz do edycji poszczególnych danych trzymanych w krotkach.

Dla modelu relacyjnego aplikacje powinny dostarczać narzędzi do obsługi:

- Zarządzania bazami danych.
- Zarządzania relacji (w postaci tabel dwuwymiarowych)
- Zarządzaniem kolumnami w relacjach (W tym ich parametrami min. typem danych)
- Obsługi kluczy głównych i obcych
- Zarządzanie połączeniami miedzy tabelami.

Ze względu na funkcjonalność konieczną do edycji danych w tabelach interfejs użytkownika powinien dostarczać operacje wejścia- wyjścia dla danych.

W poz. [2] możemy przeczytać:

„Przeciętny użytkownik aplikacji komputerowej wykorzystuje graficzny interfejs użytkownika, jako:

-- Wejście dla danych w celu wypełnienia ich zawartością.(…)

-- Wyjście, celem zaprezentowania informacji przechowywanych w modelu.”

W kontekście baz danych powyższy model jest rozszerzony o edytowanie i usuwanie istniejących danych nazywany jest paradygmatem CRUD (ang. create , read, update, delete). W języku polskim oznacza to cztery funkcjonalności

- twórz,
- wczytaj
- edytuj,
- skasuj.

Poszczególne człony skrótu CRUD odpowiadają analogicznie podstawowym operacjom zawartych w instrukcjach standardu SQL.

Tabela 3.3.1 Analogia funkcjonalności zawartej w architekturze CRUD do wyrażeń SQL

Działanie	**Instrukcja SQL**
Create	INSERT
Read (Retrieve)	SELECT
Update	UPDATE
Delete (Destroy)	DELETE

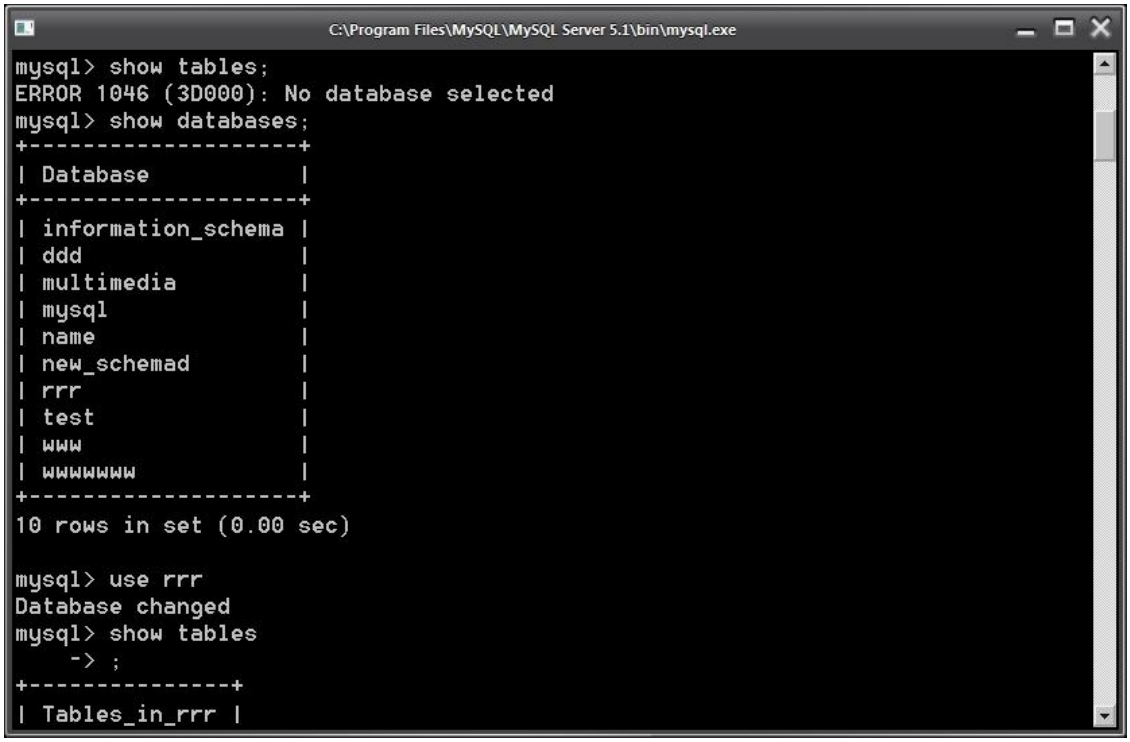
Bez tych czterech działań oprogramowanie zazwyczaj nie może być uznane za kompletne. Ponieważ te operacje są działaniami podstawowymi, są one często opisywane pod wspólnym tytułem takim, jak "zarządzanie informacją" lub "zarządzanie dokumentami".

3. Przegląd istniejących rozwiązań.

Poniższe podrozdziały omawiają istniejące rozwiązania interfejsów zapewniających dostęp do bazy danych. Poczynając od najprostszego, poprzez narzędzia internetowe do samodzielnych aplikacji. Przedstawione programy posłużą do dyskusji rozwiązań, analizy funkcjonalności i propozycji architektury aplikacji z elastycznym interfejsem dostępu do bazy danych.

3.1. Interfejs wiersza poleceń (MySQL Console)

Wiesz poleceń to standardowy interfejs dostępu do bazy danych w tym przypadku do bazy MySQL (Rys. 3.1.1). Interakcja przebiega za pomocą poleceń (komend), ze ściśle określonego zestawu i określonej składni. Komendy najczęściej wpisywane są z klawiatury lub wczytywane z zapisanych skryptów. Polecenie jest przetwarzane przez interpretatora wiersza poleceń, który przetwarza polecenia na operacje. Składnia poleceń to zazwyczaj język zapytań SQL.



Rys 3.1.1 Interfejs tekstowy typu CLI (wyświetlone nazwy baz danych w systemie w formie tabeli w zapisie tekstowym)

Komendy dostarczają m.in. następującej funkcjonalności:

- Tworzenie/ kasowanie bazy danych i tabel
- Wyświetlanie baz zawartych w bazie
- Wyświetlanie wszystkich tabel w bazie i opisu formatów pól zawartych w bazie
- Tworzenie/edytowanie użytkowników bazy danych
- Nadawanie/odbieranie przywilejów użytkownikom
- Tworzenie i zarządzanie procedurami i funkcjami baz danych.
- Zarządzanie wyzwalaczami (ang. trigers).
- Wyświetlanie poszczególnych danych z bazy poprzez rozbudowane zapytania SELECT.
- Polecenia SQL zliczające ilości rekordów lub sumujące wartości danych;
- Zarządzanie indeksami (funkcjami, które przyśpieszają wyszukiwanie danych w poszczególnych kolumnach)
- Dodawanie nowych wierszy do bazy za pomocą poleceń SQL. (Insert)
- Zmiana zawartości poszczególnych danych. (Update)
- Możemy uzyskać za ich pomocą dostęp do meta-danych (danych opisujących dane czyli informacji o typach zawartych w bazie strukturze tabel)

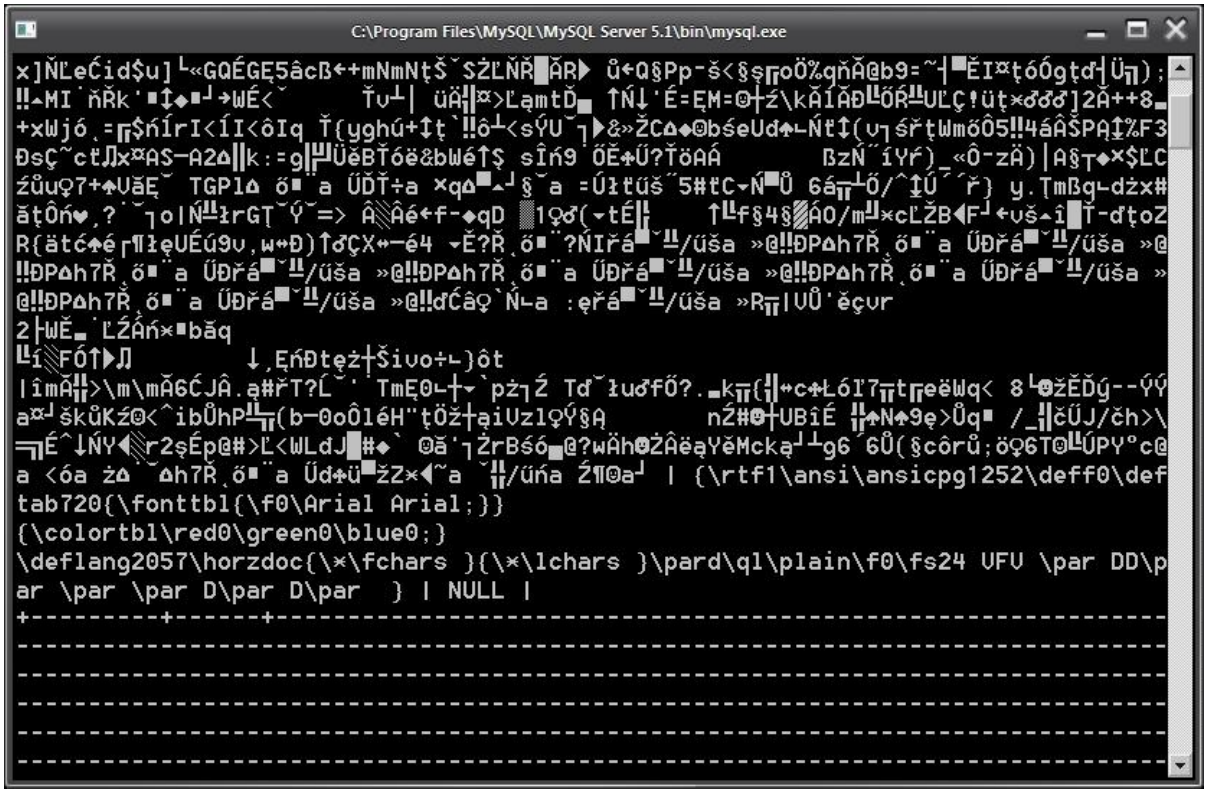
Zalety wiersza poleceń:

- Programista może w pełni zarządzać bazą z tego poziomu.
- Można uruchamiać skrypty zewnętrzne zawierające skomplikowane procedury i ciągi komend.
- Można tworzyć perspektywy, dodawać procedury i wyzwalacze do bazy danych.

Wady wiersza poleceń:

- Wymagana jest znajomości komend języka zapytań SQL.
- Tworzenie bazy danych zajmuje wiele czasu i wymaga użycia skomplikowanego układu komend.
- Wymagana jest znajomość konkretnego dialektu języka SQL zaimplementowanego przez producenta bazy danych

- Trudny w obsłudze dla nowych użytkowników - wymaga dużej wiedzy w porównaniu do interfejsów graficznych.
- Dane binarne prezentowane są w formie tekstowej (w kodowaniu binarnym rys. 3.1.2) co jest bardzo niewygodne i całkowicie uniemożliwia odczytanie danych w pozostałych niebinarnych kolumnach.



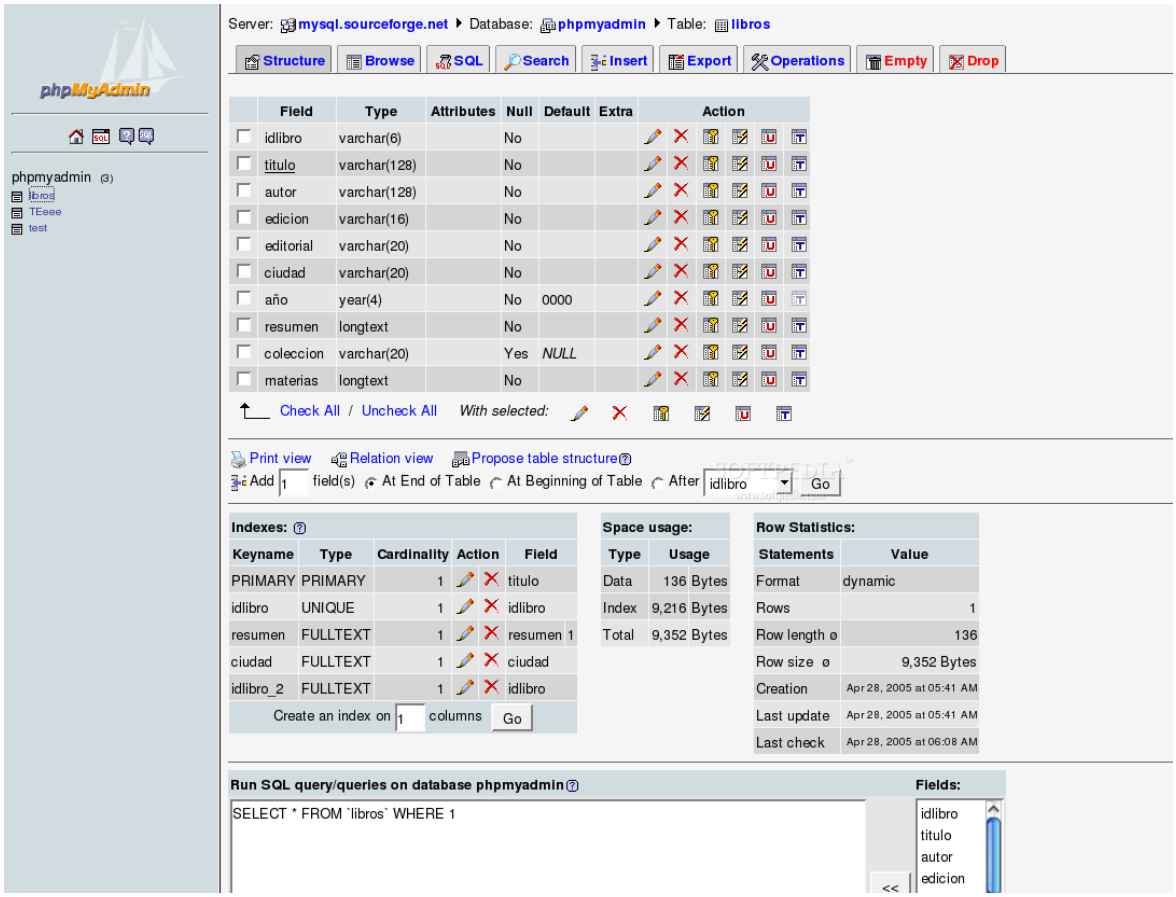
Rys. 3.1.2 Dane binarne prezentowane w formie tekstowej

Pomimo oczywistych wad użytkowania interfejs CLI jest bardzo popularnym narzędziem szczególnie pośród doświadczonych programistów lub administratorów baz danych, często z powodu przyzwyczajenia.

3.2. Dostęp przez stronę web (PhpMyAdmin)

Interfejsy CLI zostały wyparte przez bardziej przejrzyste i łatwe w obsłudze interfejsy graficzne widoczne na rysunku 3.2.1. Rozwój stron internetowych, szczególnie tych współpracujących w językami programowania (dynamiczne strony www) m.in. php umożliwił administrowanie bazą danych za pomocą stron internetowych. Do dziś aplikacje te

są bardzo pomocne, ponieważ umożliwiają dostęp zdalny do bazy tylko za pomocą przeglądarki komputerowej. Aby uruchomić aplikację PhpMyAdmin użytkownik musi zainstalować jeszcze serwer www (zazwyczaj Apache) i zainstalować interpretator języka php na komputerze, na którym znajduje się baza danych. Użytkownicy jednak często korzystają z tak zwanych pakietów zawierających odpowiednie skonfigurowane zestawy programów, dzięki czemu znacznie ułatwiają proces instalacji i dostęp do administratora bazy danych. Niektóre serwisy hostingowe dostarczają środowisko PhpMyAdmin od razu z zakupionym planem, jako podstawowe środowisko zarządzania zdalnego bazą MySQL.

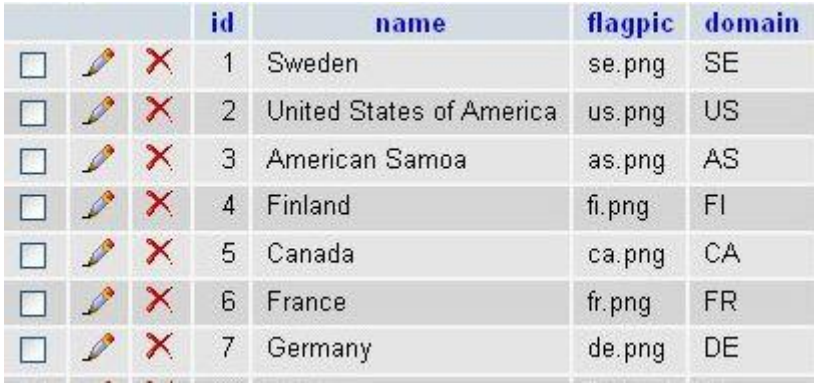


Rys.3.2.1 Interfejs użytkownika aplikacji PhpPMyAdmin

3.2.1 Edytor struktury bazy danych

PhpMyAdmin jest bardzo zaawansowaną aplikacją. Obsługuje w sposób graficzny praktycznie każdą funkcjonalność zawartą w interfejsie CLI. Bardzo ułatwia to obsługę bazy danych. Dostarcza on bardzo dużej funkcjonalności pomocnej w zarządzaniu bazą:

- Przeglądanie (Rys.3.2.2) i edycja struktur danych



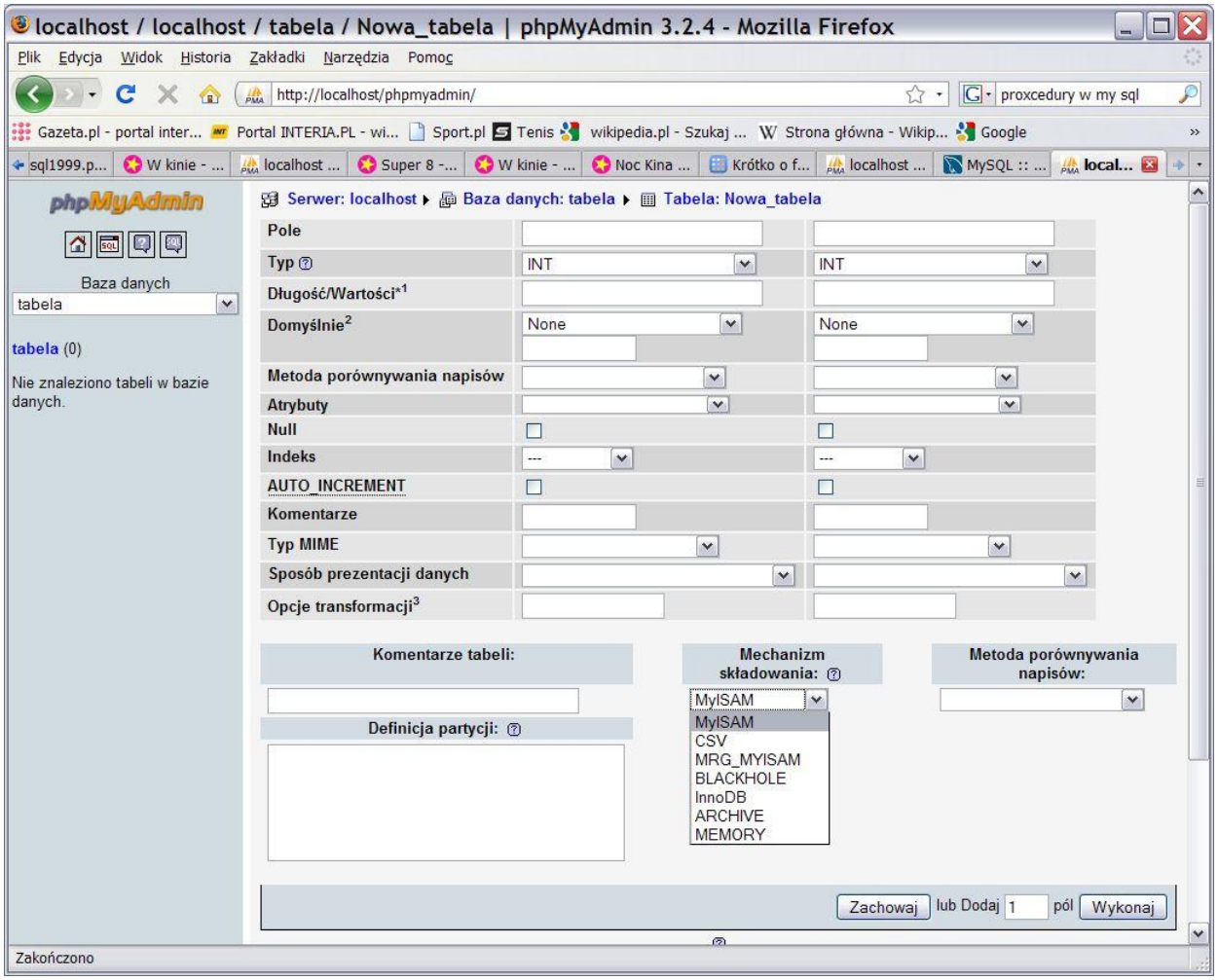
	id	name	flagpic	domain
	1	Sweden	se.png	SE
	2	United States of America	us.png	US
	3	American Samoa	as.png	AS
	4	Finland	fi.png	FI
	5	Canada	ca.png	CA
	6	France	fr.png	FR
	7	Germany	de.png	DE

Rys. 3.2.2 Prezentacja danych w PhpMyAdmin

- Zarządzanie bazami danych (dodawanie, usuwanie, edycja).
- Zarządzanie tabelami, kolumnami, typami danych (dodawanie, usuwanie, edycja) . Warto dodać, że definiowanie nowej tabeli wymaga od użytkownika zadeklarowania na początku ilości kolumn, które będą się znajdować w tabeli. Spowodowane jest to ograniczeniami protokołu http.
- Tworzenie nowej tabeli – polega na wyświetleniu formularza tekstowego o określonych polach widocznych na rysunku 3.2.3. Pola te oznaczają:
    - Nazwa kolumny w tabeli (Pole)
    - Typ danych w bazie danych (Typ)
    - Wielkość danych(Długość/Wartość);
    - Wartość domyślna podczas inicjacji wiersza w tabeli.
    - Metoda porównania napisów
    - Atrybuty Kolumny (Atrybuty). Są to dodatkowe funkcjonalności takie jak znacznik, czy dane są binarne(BINARY), czy wartości liczbowe są tylko dodatnie (unsigned), dodatnie wartości liczbowe wypełniane wartościami zerowymi tzw. (unsigned zerofill), jeśli tak to umożliwia to użycie zapytań agregujących np. sumujących na takich rekordach.
    - Atrybut automatycznego przyrostu wartości liczbowej Auto_Increment
    - Następne dwa pola określają sposób prezentacji danej kolumny w aplikacji phpMyAdmin tzw. transformacji tzn. dane zapisane w bazie mogą być

prezentowane w aplikacji jako rysunek, tekst, hiperłącze lub przekazane do innej aplikacji (w przypadku systemu Linux). Możemy tu wybrać sposób prezentacji kolumny po zastosowaniu zapytania „Select" w części przeglądania danych. Dokładne opisanie transformacji w dalszej części rozdziału.

- Wybór silnika baz danych – specyficzna funkcjonalność dla serwera mySQL Serwer ten zawiera kilka sposobów w jaki przechowuje dane. Wszystkie Silniki mają specyficzne funkcje opisane bardziej dokładniej w rozdziale 5. W podrozdziale dotyczącym technologii MySQL



Rys. 3.2.3 Widok tworzenia nowej tabeli.

- Obsługa powiązań między relacjami (klucze obce, ograniczenia integralności)

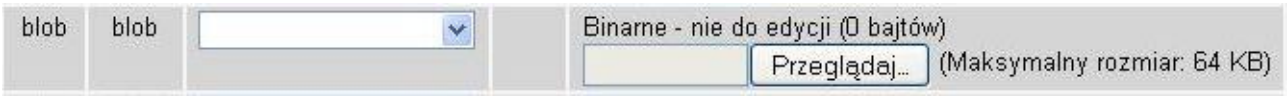
- Zarządzanie indeksami w tabelach. (w bazach danych używamy indeksów do przyspieszenia wyszukiwania poszczególnych wartości z bazy lub do nałożenia specjalnych ograniczeń na kolumnę np. unikalność wartości zawartych w rekordzie).

3.2.2 Edytor Danych phpMyAdmin

PhpMyAdmin daje możliwość prostej edycji baz danych za pomocą interfejsu graficznego. Edycja odbywa się za pomocą pól tekstowych (Rys. 3.2.4).

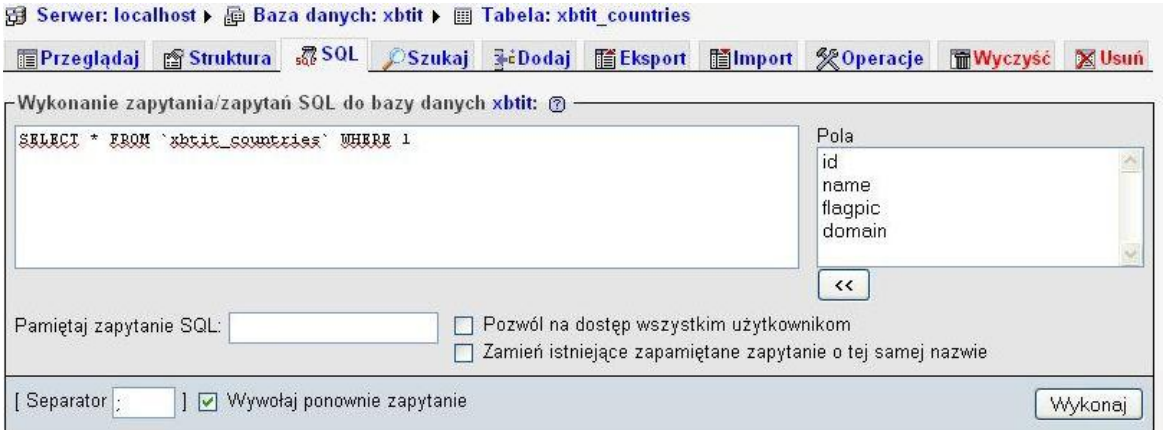
Rys. 3.2.4 przykład edycji bazy za pomocą interfejsu webowego

Możliwe jest wczytanie również danych binarnych z plików. Program informuje czy dany rekord był poprzednio zainicjowany. Niestety PhpMyAdmin nie umożliwia edycji żadnego typu binarnego (Rys. 3.2.5).



Rys. 3.2.5 informacja o zawartości danych binarnych w kolumnie.

Poza tym możliwe jest wykonywanie zapytań i skryptów SQL. Edytor do projektowania zapytań przedstawiono na rysunku 3.2.6. Ciekawostką jest to, że interfejs podpowiada nazwy pól wybranej tabeli jako pomoc przy konstruowaniu poleceń SQL.



Rys. 3.2.6 Edytor zapytań SQL

Aplikacja umożliwia poza tym:

- Import/ Export zawartości bazy do plików(szeroka gama formatów)
- Zarządzanie połączeniami między relacjami
- Zarządzanie więzami integralności(Constrains)

3.2.3 Sposób prezentacji danych

Zdecydowanie najciekawszą funkcjonalnością PhpMyAdmin są tzw. transformacje, które definiujemy na etapie tworzenia lub edycji kolumn. Transformacje to sposób prezentacji danych podczas wykonywania zapytania select. Przekształcają one wyjście za pomocą kodu html i php, aby zaprezentować je na ekranie. Istniejące transformacje dotyczą zazwyczaj prezentawania tekstu, rysunków, oraz hiperłączy. Poniżej w tabeli 3.2.1 znajduje się opis możliwych transformacji w phpMyAdmin.

Tab. 3.2.1 Opis transformacji phpMyAdmin. [10]

Sposób prezentacji danych	Opis
application/octetstream: download	Wyświetla link do ściągnięcia binarnych danych z tego pola. Pierwsza opcja to nazwa pliku binarnego. Drugą opcją jest możliwa nazwa pola zawierającego nazwę pliku. Jeżeli dana jest druga opcja, pierwsza musi być pustym napisem
application/octetstream: hex	Wyświetla szesnastkową reprezentację danych. Opcjonalny pierwszy parametr określa jak często dodawane będą spacje (domyślnie: co 2 półbajty).
image/jpeg: inline	Wyświetla klikalną miniaturkę; opcje: szerokość,wysokość w pikselach (oryginalne proporcje zostaną zachowane)

image/jpeg: link	Wyświetla link do tego obrazu (bezpośrednie ściągnięcie bloba).
image/png: inline	Zobacz image/jpeg: inline
text/plain: dateformat	Wyświetla pola typu TIME, TIMESTAMP, DATETIME lub numeryczne uniksowe znaczniki czasu jako sformatowane daty. Pierwsza opcja to przesunięcie (w godzinach) które zostanie dodane do znacznika czasu (domyślnie: 0). Drugą opcją można określić inny napis formatujący datę/czas. Trzecia opcja określa czy daty mają być lokalne ("local") czy w UTC ("utc"). Od tego wyboru zależy format daty: w przypadku "local" jest taki jak dla funkcji PHP strftime(), w przypadku "utc" — gmdate().
text/plain: external	TYLKO LINUX: Uruchamia zewnętrzną aplikację i przekazuje dane pól na standardowe wejście. Zwraca standardowe wyjście tej aplikacji. Domyślnie jest to Tidy, który porządkuje kod HTML. Ze względu na bezpieczeństwo, należy ręcznie zmodyfikować plik libraries/transformations/text_plain__external.inc.php i dodać narzędzie, na którego uruchamianie pozwalasz. Pierwszą opcją jest liczba programów, których chcesz użyć, a drugą są parametry programu. Jeżeli trzeci parametr jest ustawiony na 1 (jest to domyślna wartość), zostanie dokonana konwersja wyjścia poprzez użycie htmlspecialchars(). Jeżeli czwarty parametr został ustawiony na 1 (jest to domyślna wartość), zawartość komórki nie będzie zawijana, tak że całe wyjście zostanie pokazane bez zmian formatu.
text/plain: formatted	Zachowuje oryginalne formatowanie pola. Neutralizowanie znaków niespecjalnych nie jest dokonywane.
text/plain: imagelink	Wyświetla obrazek i link, pole zawiera nazwę pliku; pierwszą opcję jest prefiks, taki jak "http://domena.com/", drugą opcją jest szerokość w pikselach, trzecią opcją jest wysokość.
text/plain: link	Wyświetla link, pole zawiera nazwę pliku; pierwsza opcja to prefiks, taki jak "http://domena.com/", druga opcja to tytuł linku.
text/plain: longToIpv4	*Transformacja ta nie ma opisu.*
text/plain: SQL	Formatuj tekst traktując jako zapytanie SQL z podświetlaniem składni.
text/plain: substr	Pokazuje jedynie część napisu. Pierwsza opcja to offset, od którego ma zacząć się wyświetlanie tekstu (domyślnie 0). Druga opcja to ilość zwracanego tekstu. Jeżeli jest pusta, zwracany jest cały pozostały tekst. Trzecia opcja określa jakie znaki zostaną dodane do wyjścia, jeżeli zwracany jest część napisu (domyślnie: …) .

Dane o transformacjach zawarte są w bazie danych, która tworzy się podczas instalacji aplikacji o nazwie phpMyAdmin. Dane te zawarte są w tabeli pma_column_info. Zawiera następujące kolumny.

- id - identyfikator

- db_name - nazwa bazy danych mysql którego dotyczy transformacja
- table_name - nazwa tabeli transformacji.
- column_name – analogicznie nazwa kolumny.
- comment – komentarz do transformacji wedle uznania użytkownika.
- mimetype – możliwość wybrania tranformacji w standardzie MINE okreslenie typu tego standardu (dostępne typy to: application/octetstream, image/jpeg, image/png, text/plain).
- transformation – nazwa pliku z rozszerzeniem PHP który będzie parsował wyjście .
- transformation_options - parametry przekształcenia specyficzne dla danego dodatku.

Transformacje to bardzo interesująca funkcjonalność. Ogranicza się jednak tylko do prezentacji danych. Nie można zastosować ich podczas edycji danych, prawdopodobnie ze względu na ograniczenia protokołu http. Z praktycznego punktu widzenia programiści rzadko z nich korzystają. Zazwyczaj spowodowane jest to koniecznością zagłębiania się w dokumentacje. Dodawanie transformacji do istniejącej kolumny powinno znajdować się w części aplikacji odpowiadającej za przeglądanie danych, a nie w edycji tabeli/kolumny. Użytkownik może mieć też wrażenie, że edytując strukturę tabeli/kolumny, a w niej sposób prezentacji, zmienia zawartość struktury bazy. Sposób prezentacji związany jest jednak tylko z aplikacją PhpMyAdmin a nie z strukturą bazy danych.

3.3. Aplikacja graficzna - Workbench (wcześniej MySQL Gui Tools)

MySQL Workbench 5.2 to zbiór narzędzi do projektowania i edycji oraz zarządzania bazą danych MySQL. Zawiera trzy główne podsystemy:

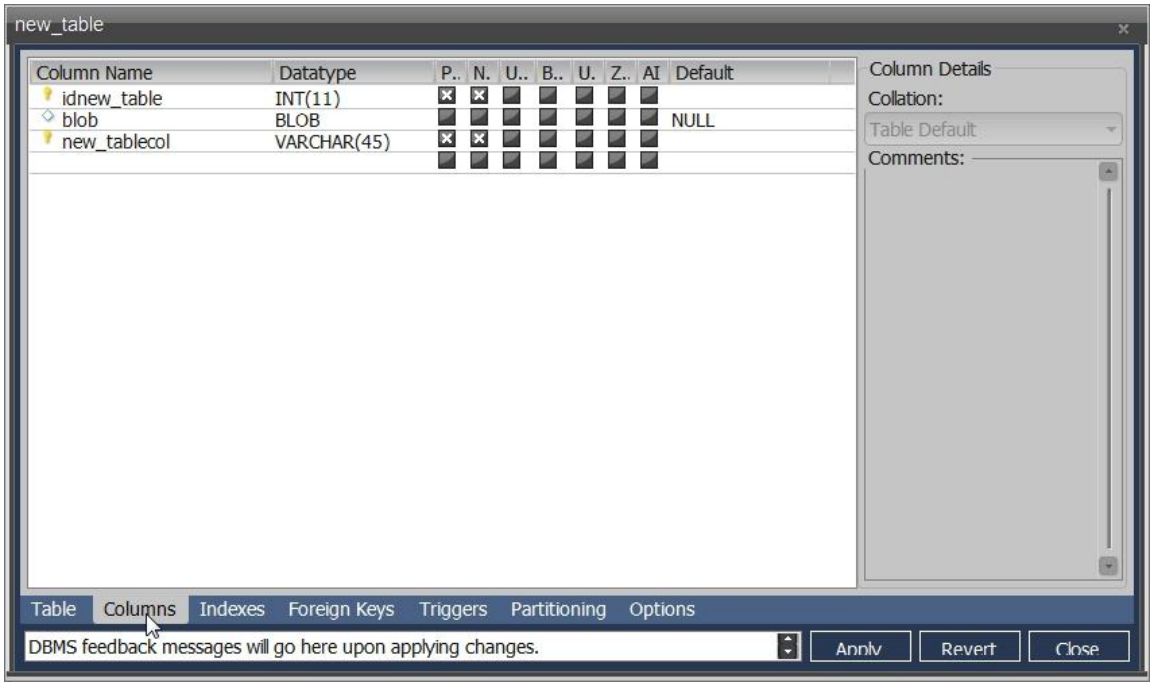
- SQL Development – interfejs do podłączenia do istniejącej bazy, wykonywania zapytań i skryptów SQL, zawierających możliwość edycji rekordów zawartych w bazie.
- DataModeling – narzędzie do graficznego projektowania bazy danych.
- Server Administration – aplikacja do zarządzania ustawienia serwera bazy danych.

3.3.1 Edytor struktury bazy danych

Część SQL Development jest bardzo funkcjonalnym edytorem struktury bazy danych z możliwością edycji baz danych wykonana w bardzo przyjaznym graficznym interfejsie, oto jego funkcjonalności:

- Zarządzanie bazami danych
- Obsługa tabel, kolumn z możliwością ich projektowania i edytowania.
- Dodawanie indeksów.
- Zarządzanie kluczami i połączeniami między relacjami.
- Edycja perspektyw (ang. Views).
- Zarządzanie tzw. Routines (jęz. pol. Rutyna) tworem przechowującym funkcje i procedury w bazie danych.
- Pisanie wyzwalaczy(ang. Trigers).
- Przypisywanie opcji partycjonowania do typów

Główny widok edytora struktury ma postać tabeli widocznej na rysunku 3.3.1.



Rys. 3.3.1 Widok edytora struktury

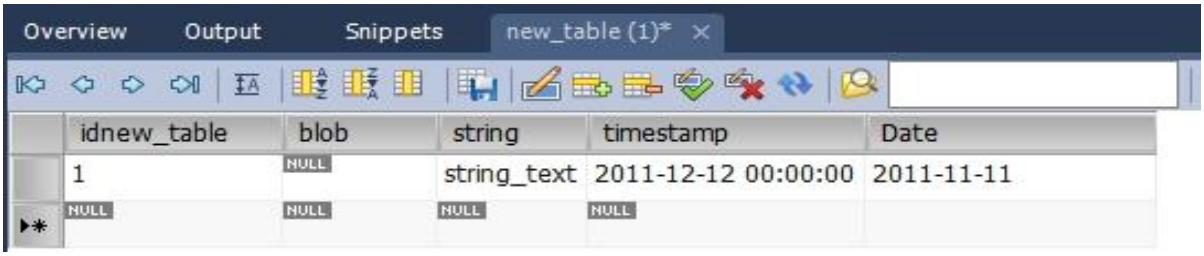
Podobnie jak w phpMyAdmin możemy wybrać typ danych dla poszczególnych kolumn. Określić, czy będą identyfikować krotkę. Ustawić następujące atrybuty:

- Unique- unikalność danych w kolumnie;
- Not Null- określający czy rekord może przyjąć wartość null
- BINARY- określający czy zainicjujemy typ danych jako ciąg binarny (niektóre typy danych mogą być trzymane w bazie jako ciągi binarne lub jako ciągi znaków).
- Unsigned- definiujący czy wartość typów liczbowych będzie przyjmowała wartości tylko dodatnie czy również minusowe.
- Zerofill- opcjonalne wypełnienie pustych wartości liczbowych zerami.
- Auto_Increament- atrybut ustawiający automatyczny przyrost dla wartości liczbowych.
- Default- określenie domyślnej wartości kolumny podczas inicjacji.

Na uwagę zasługuje również dobrze zaprojektowany edytor zarządzający połączeniami między relacjami, czyli kluczami obcymi. Każde połączenie tworzone jest na podstawie nazwy i tabeli, z którą chcemy zawrzeć połączenie. Po wyborze tabeli, z którą chcemy się połączyć, wyświetlane są pola bieżącej relacji. Każdemu takiemu polu możemy przyporządkować klucz główny lub jego część z wybranej wcześniej tabeli. Taka kolumna tworzy jedno połączenie. Jeśli klucz główny wybranej tabeli składa się z więcej niż jednej kolumny to należy wykonać powyższą czynność dwukrotnie.

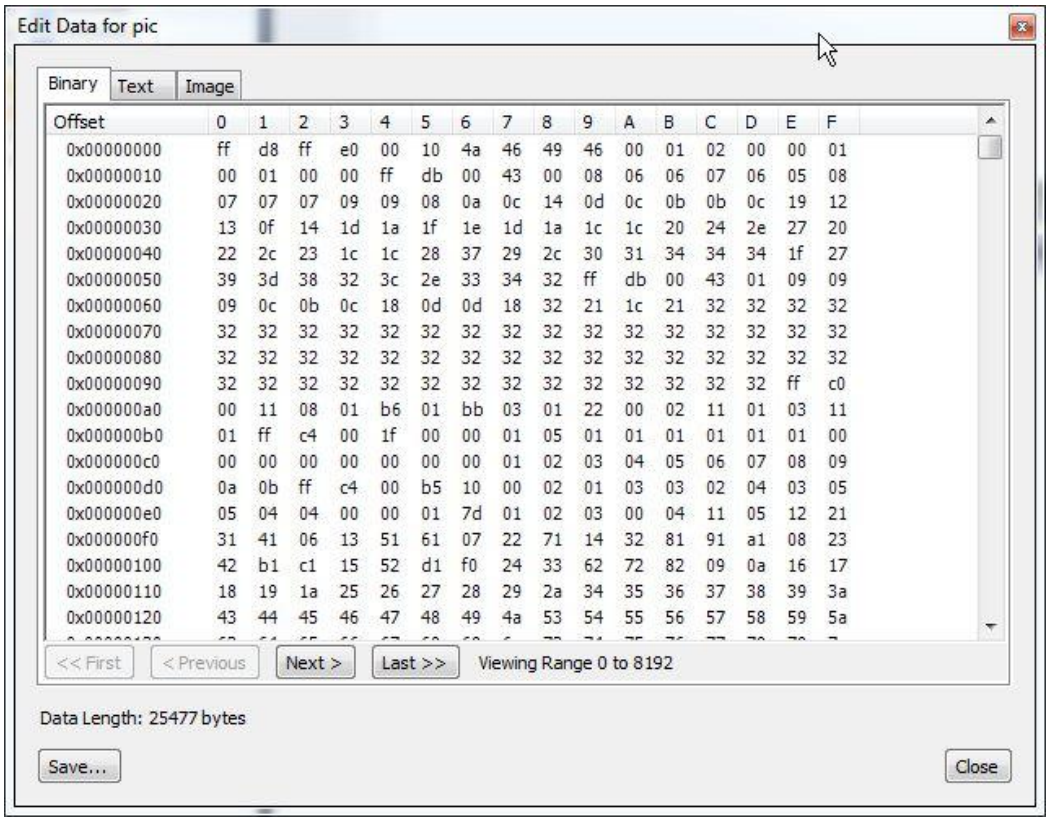
3.3.2 Prezentacja i edytowania danych

Jedną z funkcji narzędzia SQL Development jest edytor komórek zawartych w bazie. Umożliwia on modyfikację komórek zapisanych w sposób tekstowy, m.in. liczby, wartości logiczne (boolean), daty, ciągi znaków. Wyglądem przypomina arkusz kalkulacyjny (Rys 3.3.2).



Rys 3.3.2 Widok edytora danych.

Aplikacja pozwala na prostą edycję danych binarnych, poprzez operacje dyskowe (wejścia-wyjścia) oraz przeglądarkę zawartości w zapisie Hex(Rys 3.2.3), tekstowym (Rys. 3.2.5), lub jako przeglądarka obrazków (Rys. 3.2.4).



Rys 3.2.3 Przeglądanie danych w formie binarnej zapisie Heksagonalnym

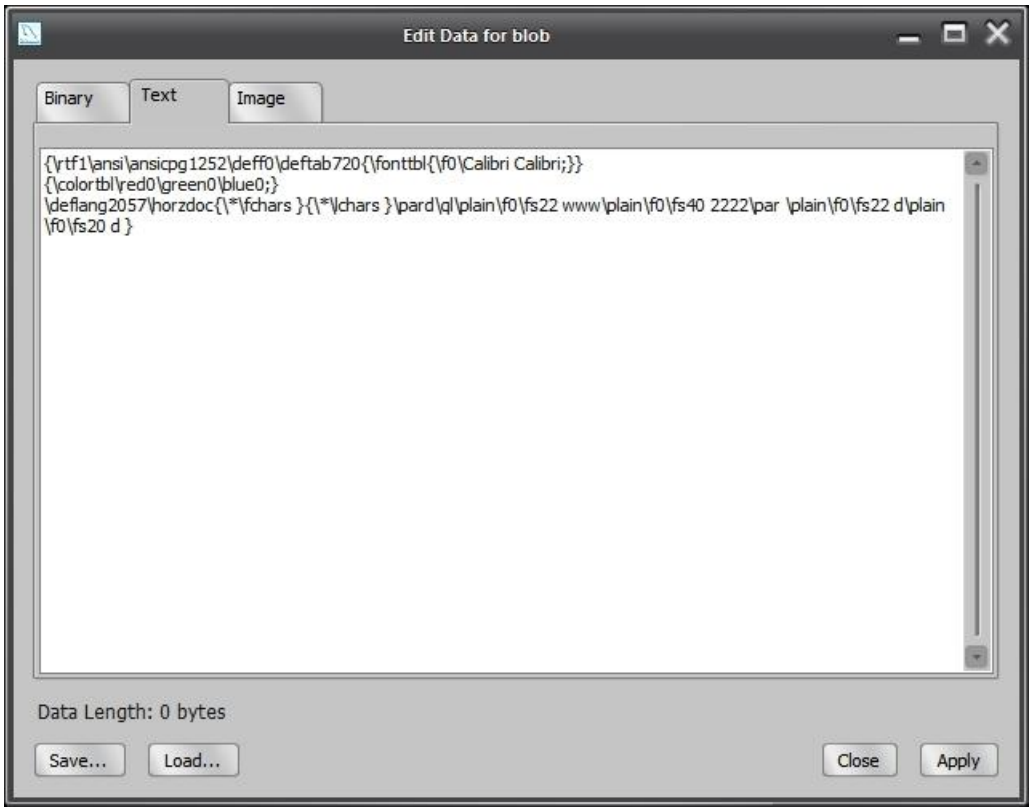
Część odpowiedzialna za prezentacje danych w postaci kodowania bitów w systemie szesnastkowym przedstawia dane w postaci tabeli z możliwością stronicowania. Możemy zobaczyć również wielkość tych danych.

Przeglądarka rysunków zaprezentowana na rys 3.2.4 wyświetla zawartość plików o różnych formatach m.in. JPEG, BMP, PNG

Rys. 3.2.4 Widok prezentacji rysunku

Przeglądanie danych, jako tekst polega na odkodowaniu bitów do formatu tekstowego. Poniżej na rys 3.2.5 przedstawiono tekst zakodowany w formacie RTF. Program nie obsługuje żadnych zaawansowanych formatowań tekstu(w tym formatu RTF), ale czysty tekst (ang. plain text) znany użytkownikom z aplikacji notatnik.

Przeglądanie danych binarnych zakłada więc tylko trzy możliwe z góry określone typy danych. Nie ma możliwości rozbudowy modelu prezentacji ani edycji danych.



Rys. 3.2.5 Widok zapisu tekstowego.

Architektura tego narzędzia opiera się na stworzeniu zapytań na podstawie informacji o układzie bazy danych. Następnie stworzenie graficznej reprezentacji pobranych danych. Potem użytkownik ma możliwość edycji wyświetlonych komórek w tabeli. Wszelkie zmiany na komórkach zostaną przepisane na język zapytań bazy danych.

Analizując tego typu narzędzie warto wyróżnić zalety i wady:

Zalety:

- Uniwersalność- dostęp do każdego schematu w bazie danych, dzięki meta-danym opisującym zawartość bazy.
- Dostęp do bazy ad-hoc - charakteryzuje się szczególnie dużą użytecznością programistów, ponieważ zapewnia szybki dostęp do bazy danych i pozwala na jego szybką edycję.

Wady:

- Dosłowne mapowanie danych z bazy w formie tekstowej.
- Brak możliwości przedstawienia komórek w intuicyjny sposób dla użytkownika.

3.4. SqlRazor

RazorSQL to rozbudowany program komercyjny obsługujący około 29 baz danych widocznych w tabeli 3.4.1. Pozwala na połączenia zarówno przez sterowniki ODBC jak i JDBC.

Tab. 3.4.1 Bazy danych obsługiwane przez program RazorSQL

DB2	Ingres	Pervasive	Cache	Mimer SQL
Derby	InterBase	PostgreSQL	Daffodil	Netezza
Firebird	JavaDB	SimpleDB	DBASE	Paradox
FrontBase	MS SQL Server	SQLite	FileMaker	PointBase
H2	MySQL	SQL Anywhere	Mckoi	Solid
HSQLDB	OpenBase	Sybase (ASE)	Microsoft Access	Teradata
Informix	Oracle	Sybase (IQ)		

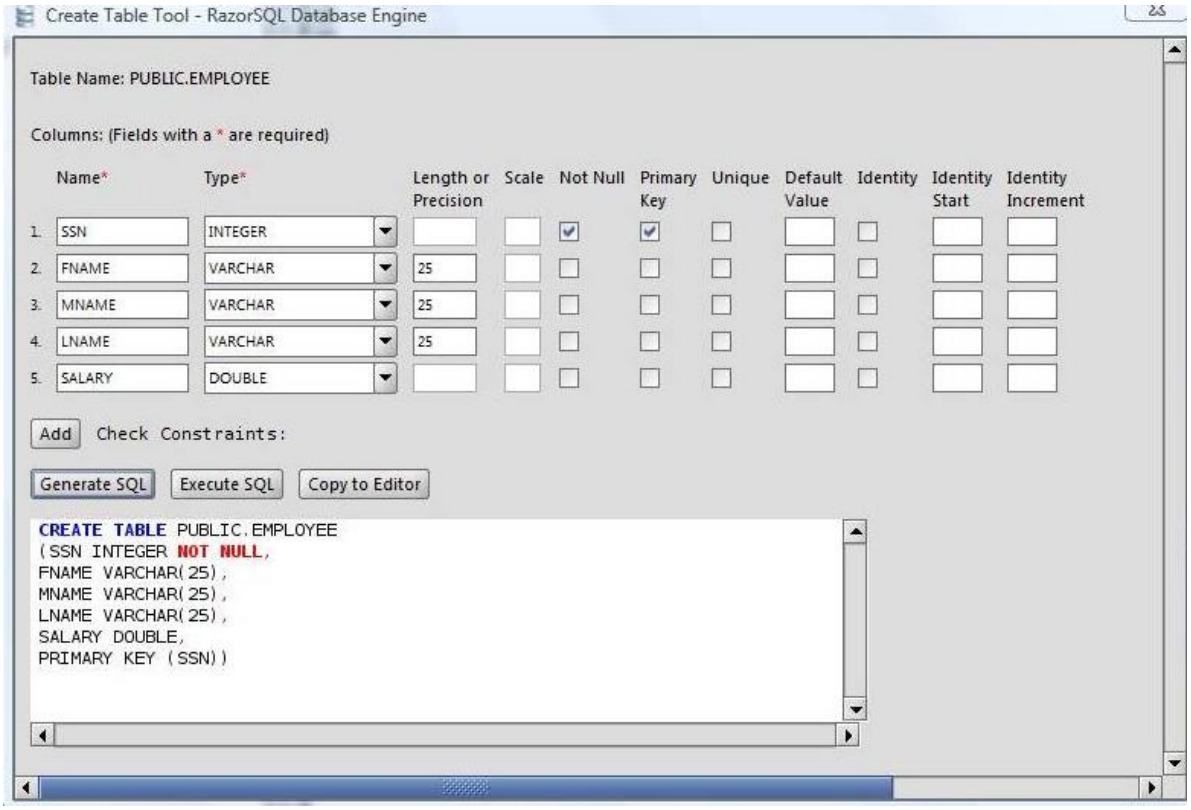
Aplikacja ta składa się z wielu narzędzi, między innymi:

- **Database Browser** – jest narzędziem do przeglądania bazy schematów(katalogów) tabel, kolumn, perspektyw (ang. Views), zarządzania kluczami głównymi i obcymi, indeksami i procedurami.
- **SQL Editor** – pozwala tworzyć, edytować oraz uruchamiać skrypty języka zapytań SQl. Używa rozbudowanego edytora o nazwie EditRocket. Zapewnia on obsługę ponad 20 językom programowania w tym SQL, PL/SQL, TransactSQL, SQL PL, HTML, Java, XML.
- **Database Tools -** dostarcza zestaw wizualnych narzędzi do tworzenia, opisywania i zarządzania elementami baz danych, takimi jak tabele, perspektywy, indeksy, procedury, funkcje, wyzwalacze.
- **Edit Table Tool –** obsługuje edycje bazy danych tworząc instrukcje imperatywne do manipulacji zawartością bazy danych(insert, update, and delete). Edycja danych przebiega podobnie do arkusza kalkulacyjnego w formie tabeli. Przed zatwierdzeniem instrukcji użytkownik może podejrzeć formę instrukcji, jaka ma być wysłana do bazy.
- **Database Query Tool –** specjalne narzędzie do tworzenia zapytań do bazy. Pozwala na późniejsze filtrowanie, sortowanie oraz przeszukiwanie wyników.
- **Import Data** - importuje dane z różnych formatów plików m.in. z arkuszy kalkulacyjnych.

- **Export Data** – pozwala wyeksportować dane do różnych formatów m.in. XML, HTML, arkusz Excel, instrukcje języka SQL w formie skryptu.
- **SQL Query Builder –** narzędzie ułatwiające tworzenie skomplikowanych zapytań do baz danych w tym skomplikowanych z wielokrotnym użyciem łącznika tabel (ang. Join)
- **Built-in Database –** zawiera wbudowane środowisko relacyjnych baz danych (HSQLDB), które działa bezpośrednio po dokonanej instalacji, bez konieczności konfiguracji.
- **Data Compare –** to narzędzie do porównywania wyników zapytań, nawet z różnych systemów baz danych.

3.4.1 Edytor struktury bazy danych

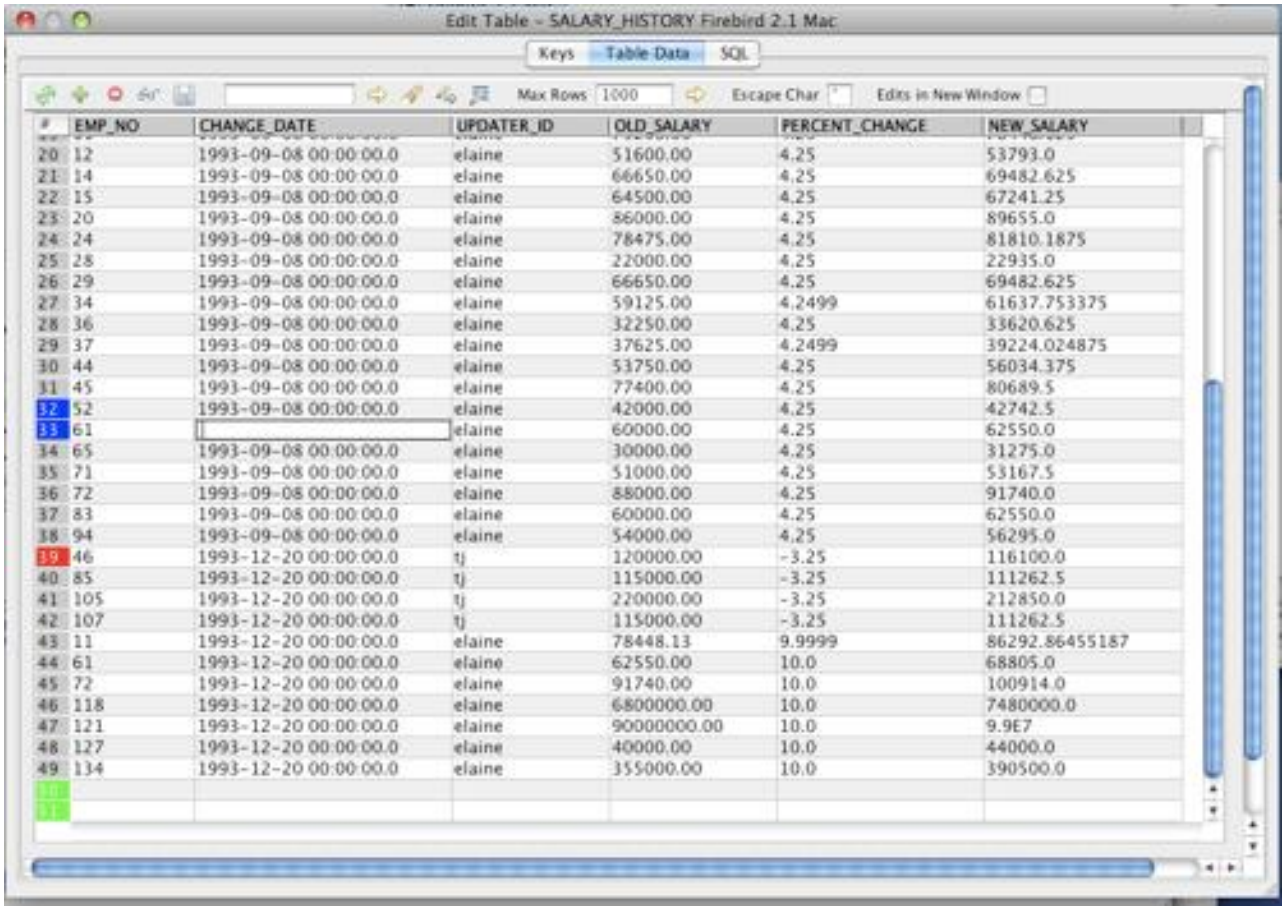
Edytor struktur, a w szczególności tabeli (najważniejszej części edycji struktury bazy) przypomina edytor z Workbench, ale ze względu na uniwersalność (obsługę więcej niż jedną bazę danych) posiada inne atrybuty i opcje. Różnica pomiędzy polami najbardziej widoczna jest w ostatnich trzech kolumnach na rysunku 3.4.1.



Rys. 3.4.1 Interfejs użytkownika programu edytora tabeli Razor SQL

3.4.2 Opis edytora baz danych

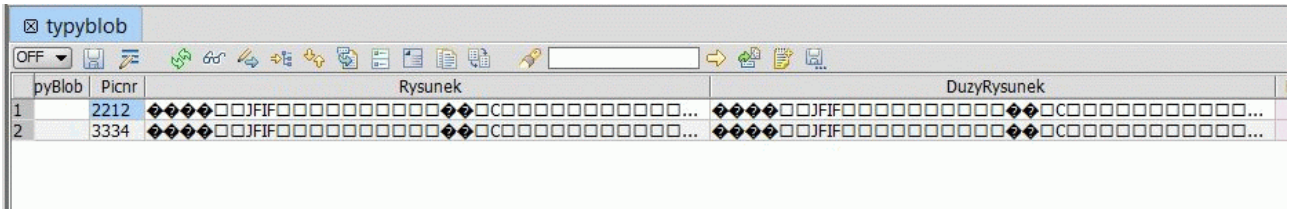
Edytor zawartości bazy danych przypomina arkusz kalkulacyjny. Typy danych reprezentowane są za pomocą tekstu (Rys.3.4.2).



#	EMP_NO	CHANGE_DATE	UPDATER_ID	OLD_SALARY	PERCENT_CHANGE	NEW_SALARY
20	12	1993-09-08 00:00:00.0	elaine	51600.00	4.25	53793.0
21	14	1993-09-08 00:00:00.0	elaine	66650.00	4.25	69482.625
22	15	1993-09-08 00:00:00.0	elaine	64500.00	4.25	67241.25
23	20	1993-09-08 00:00:00.0	elaine	86000.00	4.25	89655.0
24	24	1993-09-08 00:00:00.0	elaine	78475.00	4.25	81810.1875
25	28	1993-09-08 00:00:00.0	elaine	22000.00	4.25	22935.0
26	29	1993-09-08 00:00:00.0	elaine	66650.00	4.25	69482.625
27	34	1993-09-08 00:00:00.0	elaine	59125.00	4.2499	61637.753375
28	36	1993-09-08 00:00:00.0	elaine	32250.00	4.25	33620.625
29	37	1993-09-08 00:00:00.0	elaine	37625.00	4.2499	39224.024875
30	44	1993-09-08 00:00:00.0	elaine	53750.00	4.25	56034.375
31	45	1993-09-08 00:00:00.0	elaine	77400.00	4.25	80689.5
32	52	1993-09-08 00:00:00.0	elaine	42000.00	4.25	42742.5
33	61		elaine	60000.00	4.25	62550.0
34	65	1993-09-08 00:00:00.0	elaine	30000.00	4.25	31275.0
35	71	1993-09-08 00:00:00.0	elaine	51000.00	4.25	53167.5
36	72	1993-09-08 00:00:00.0	elaine	88000.00	4.25	91740.0
37	83	1993-09-08 00:00:00.0	elaine	60000.00	4.25	62550.0
38	94	1993-09-08 00:00:00.0	elaine	54000.00	4.25	56295.0
39	46	1993-12-20 00:00:00.0	tj	120000.00	-3.25	116100.0
40	85	1993-12-20 00:00:00.0	tj	115000.00	-3.25	111262.5
41	105	1993-12-20 00:00:00.0	tj	220000.00	-3.25	212850.0
42	107	1993-12-20 00:00:00.0	tj	115000.00	-3.25	111262.5
43	11	1993-12-20 00:00:00.0	elaine	78448.13	9.9999	86292.86455187
44	61	1993-12-20 00:00:00.0	elaine	62550.00	10.0	68805.0
45	72	1993-12-20 00:00:00.0	elaine	91740.00	10.0	100914.0
46	118	1993-12-20 00:00:00.0	elaine	6800000.00	10.0	7480000.0
47	121	1993-12-20 00:00:00.0	elaine	90000000.00	10.0	9.9E7
48	127	1993-12-20 00:00:00.0	elaine	40000.00	10.0	44000.0
49	134	1993-12-20 00:00:00.0	elaine	355000.00	10.0	390500.0

Rys. 3.4.2 Wygląd edytora danych w programie Razor SQL (reprezentacja różnych typów danych w edytorze).

Nawet w przypadku danych binarnych zapisane są one w kodzie tekstowym (Rys. 3.4.3), który nie jest zrozumiały dla użytkowania i praktycznie niemożliwy do edycji.

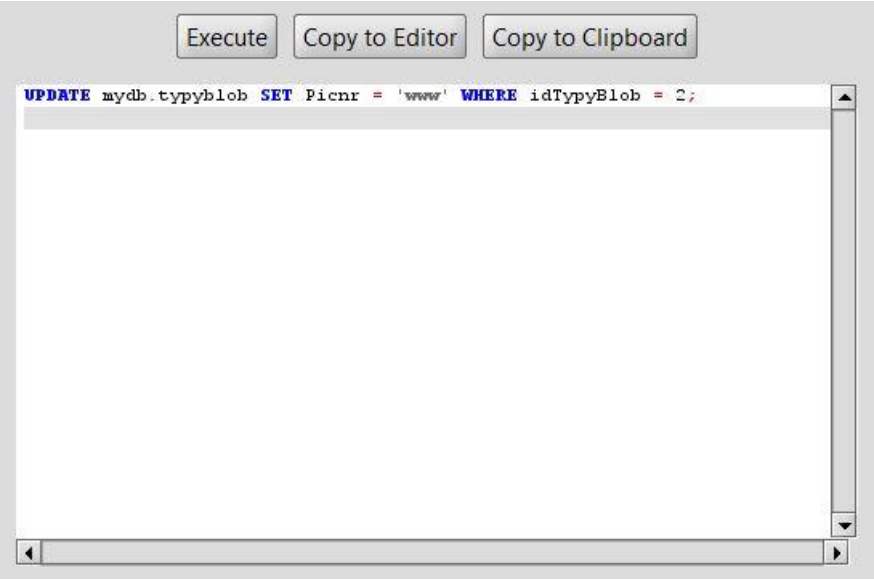


	pyBlob	Picnr	Rysunek	DuzyRysunek
1		2212	����□□JFIF□□□□□□□□□□□□□□□□C□□□□□□□□□□□□□□□...	����□□JFIF□□□□□□□□□□□□□□□□C□□□□□□□□□□□□□□□...
2		3334	����□□JFIF□□□□□□□□□□□□□□□□C□□□□□□□□□□□□□□□...	����□□JFIF□□□□□□□□□□□□□□□□C□□□□□□□□□□□□□□□...

Rys. 3.4.3 Reprezentacja typów blob w edytorze.

Każde pole binarne zapisane w edytorze możemy podejrzeć w osobnej ramce w bardzo podobny sposób jak w programie Workbench z rozdziału 3.3. Właściwie rozwiązania te są bardzo podobne. Również do dyspozycji mamy widok w systemie szestnastkowym, podejrzenie tekstu jako wartości tekstowej oraz przegladanie rysunków. Formą edycji danych są operacje wejścia-wyjścia (wczytywanie i zapisywanie z plików).

Edycja danych przewiduje wygenerowanie instrukcji imperatywnej (Insert, Update,Delete) i wyświetlenie jej na ekranie (Rys. 3.4.4) . Następnie użytkownik może potwierdzić wykonanie polecenia. Alternatywnie może wkleić inne polecenia lub zapisać bieżące polecenie do schowka.



Rys. 3.4.4. Edytor zapytań imperatywnych

Rozwiązanie to jest również stosowane do uaktualniania danych binarnych, co niestety powoduje wyświetlanie bardzo długich zapytań z danymi binarnymi i przy edycji wielu pól bazy, użytkownik musi przeglądać wielostronicowe zapytania.

4. Propozycja nowego rozwiązania

W niniejszym rozdziale wyszczególniono cechy zawarte w rozdziałach 2 i 3. Zidentyfikowano podstawowe wymagania funkcjonalne dotyczące aplikacji do zarządzania zawartością baz danych. Przeanalizowano wady i zalety poszczególnych rozwiązań. Niniejszy rozdział przeprowadzi podsumowanie oraz wyszczególnienie cech, jakie powinien spełniać system do edycji baz danych.

Zaprezentuje propozycje rozszerzenia istniejącego modelu o nową funkcjonalność dodającą elastyczność do obsługi różnych typów danych.

4.1. Podsumowanie zaprezentowanych programów

W rozdziale 2. niniejszej pracy przedstawiono różne podejścia, metody i praktyki tworzenia graficznego interfejsu użytkownika. Techniki te charakteryzują się dużą różnorodnością i z tego powodu zasugerowano użycie wzorca projektowego MVC, aby zapewnić rozdzielność części funkcjonalnej interfejsu od części wizualnej. W dalszej części rozdziału 2. opisano charakterystykę relacyjnych baz danych i zidentyfikowano podstawową funkcjonalność, jaką powinien dostarczać interfejs dla bazy danych

W rozdziale 3. przeprowadzono przegląd interfejsów dostępnych na rynku. Wyszczególniono cechy – zalety i wady poszczególnych rozwiązań.

Z zaprezentowanych interfejsów do obsługi bazy danych możemy wyróżnić dwie części logiczne:

- Przeglądarkę i edytor rekordów o funkcjonalności CRUD.
- Edytor struktury bazy danych.

Funkcjonalności te idą zazwyczaj w parze ze względu na cel użytkowania takich aplikacji, czyli możliwość podglądu oraz edycji danych zawartych w dowolnej bazie, oraz możliwość dostosowania modelu bazy danych.

Poszczególne programy wyróżnia odmienna prezentacją danych. Szczególnie, że mamy do czynienia zarówno z aplikacją sieciową (webową) jak i aplikacjami samodzielnymi, zwanymi potocznie okienkowymi. PhpMyAdmin i Workbench to narzędzia skierowane dla

użytkowników bazy danych MySql. SqlRazor to komercyjne i bardziej uniwersalne narzędzie obsługujące wiele baz danych.

Aplikacja phpMyAdmin jest aplikacją sieciową (Webową) i sama oparta jest o bazę, którą administruje (MySQL). Zapisuje ona w niej wszelkie działania historyczne, wykonane zapytania i m.in. metadane dotyczące prezentacji danych w jezyku html, wspomnianych wcześniej tzw. transformacji. Narzędzia typu standalone (samodzielne), takie jak aplikacje okienkowe (Workbench, SQLRazor) nie powinny integrować się w bazę, ponieważ często nie mają takich uprawnień. Ze względów bezpieczeństwa uniemożliwiałoby to obsługę bazy.

Zapis metadanych m.in. dotyczących prezentacji w takich aplikacjach powinien odbywać się w plikach zewnętrznych np. w plikach XML lub YAML. Pojawia się tu problem wiązania plików z różnymi bazami danych i serwerami. Załóżmy, że na dwóch innych serwerach znajdują się bazy o takich samych nazwach i tabelach, ale innej strukturze kolumn. Identyfikacja danej tabeli wymagałaby, więc sprawdzenia nazw serwera, bazy, tabeli i wszystkich kolumn w tabeli i ich typów. Jeśli pojedynczą tabelę, przypisalibyśmy do pojedynczego pliku to nazwa pliku byłaby bardzo długa.

4.2. Propozycja nowego rozwiązania

Praca proponuje rozszerzenie modelu edytora bazy danych, o możliwość rozbudowy jego funkcjonalności za pomocą dodatków (ang. plugin) do obsługi różnych typów danych. Dotyczyłoby to zarówno prezentacji, jak i edycji danych. Architektura umożliwiałaby szybką implementację nowych dodatków dla dowolnych widgetów (kontrolek lub JavaBean) obsługujących przetwarzanie danych.

Aplikacja dostarczałaby funkcjonalność CRUD dla danych w bazie oraz umożliwiałaby edycje struktury bazy. Koncepcja ta zaczerpnięta jest z programów sieciowych do administracji bazą. Celem pracy jest przeniesienie funkcjonalności prezentowania danych na różny sposób na stronach www, do aplikacji okienkowych, które zazwyczaj obsługują tylko binarne dane tekstowe lub graficzne.

4.2.1 Motywacja przyjętego rozwiązania

W dzisiejszych czasach coraz więcej informacji trzyma się w bazach danych. Coraz większe znaczenie mają różne dane multimedialne i w niedalekiej przyszłości bazy danych

będą przechowywały duże zasoby plików multimedialnych. Dane multimedialne niezależnie od zawartości trzymane są w bazie, jako strumień danych binarnych (najczęściej jako obiekt typu BLOB –Binary Large Object).

Motywacją do stworzenia tej aplikacji była potrzeba stworzenia uniwersalnej przeglądarki/edytora zawartości bazy danych dla różnej zawartości binarnej. Często podczas procesu programowania aplikacji na podstawie bazy danych, konieczne jest szybkie zainicjowanie bazy konkretnymi typami danych, aby później móc przeprowadzić testy poprawności aplikacji (wprowadzenia jakiś zmian i konieczność szybkiego podejrzenia zawartości bazy).

Poza tym, rozszerzalna konstrukcja programu wyeliminowałaby konieczność używania zewnętrznych aplikacji tzn. proces edycji danych binarnych nie wymagałby żmudnego pobierania danych z bazy, zapisywania do pliku, edytowania w zewnętrznej aplikacji, a następnie ponownego wstawiania do bazy. Wszystko odbywałoby się ad-hoc w jednym systemie, co znacznie skraca czas i jest bardzo wygodne.

Pomysł zakłada również zaprojektowanie interfejsów oraz reguł prostego programowania nowych dodatków, obsługujących nowe typu danych, korzystając z bibliotek zewnętrznych.

4.2.2 Wymagania funkcjonalne

Na podstawie analizy potrzeb z prezentacji oprogramowania w rozdziale 3 oraz wstępnych założeń określonych w rozdziale 2. można nakreślić wymagania funkcjonalne systemu m.in.:

- Zapewnienie interfejsu dostępu do bazy danych - tworzenie połączeń
- Zapisywanie konfiguracji połączeń w plikach
- Niezależność działania od struktury bazy danych
- Stworzenie meta-modelu obiektowego opisującego relacyjną bazę danych
- Manipulacje danymi i zapewnienie funkcjonalności CRUD
- Obsługa różnych typów danych w tym wielu typów multimedialnych
- Zapewnienie walidacji wprowadzanych danych
- Możliwość programowania obsługi nowych typów

- Możliwość przystosowania istniejących komponentów do obsługi danych
- Stworzenia zasad projektowania dodatków do obsługi danych multimedialnych
- Zapewnienie kompatybilności rozszerzeń z istniejącymi komponentami biblioteki graficznej i ich modelami
- Przypisywanie automatyczne dodatków do typów baz danych
- Edycja struktury bazy danych.
- Zarządzanie bazami danych oraz tabelami, edycja kolumn oraz zarządzanie kluczami obcymi.

4.3. Analiza

Na podstawie powyższych założeń ogólnych warto pogrupować poszczególne funkcję i zidentyfikować poszczególne podsystemy. Aplikacja powinna być podzielona na moduły:

- Moduł pobierający metadane z bazy danych i przenoszenie tej informacji na model obiektowy

  Moduł ten powinien umożliwiać, pobranie informacji o strukturze bazy, typach danych zawartych w konkretniej bazie, związkach pomiędzy tabelami, (informacje o kluczach głównych i obcych). Następnie dane te powinny być zapisane w modelu obiektowym. Model obiektowy powinien być jak najbardziej użyteczny. Warunkiem tego w tym wypadku jest:

  1. swobodna nawigacja po modelu w obie strony,
  2. zachowanie elementów charakterystycznych dla modelu relacyjnego,
  3. możliwość korzystania z modelu danych w innych komponentach.
- Moduł automatycznego tworzenia zapytań do bazy danych (Select, Insert, Delete, Update) na podstawie meta danych.
- Moduł dynamicznego mapowania typów bazy danych na typy języka programowania.
- Moduł prezentacji bazy danych
- Reprezentacje poszczególnych kolumn w dodatkach
- Moduł pobierania i rejestracji dodatków dla poszczególnych typów
- Moduł rozdzielający dodatki.

- Moduł rozpoznawania dodatków multimedialnych
- Moduł pobierania metadanych
- Moduł projektowania bazy danych.
- Moduł dodawania tabeli do bazy
- Moduł edycji istniejącej tabeli
- Moduł zarządzania kluczami obcymi.

4.3.1 Moduł zapytań

Moduł ten ma być możliwie jak najbardziej odseparowany od rodzaju bazy danych, jaką obsługuje. Architektura powinna pozwalać na zmianę dostawcy bazy danych. System powinien tworzyć zapytania dla różnych typów danych zarówno pobieranych z bazy, jak i danych języka programowania.

Program powinien dostarczać następujące zapytania dla systemu CRUD takie jak:

- Dostarczanie całej zawartości tabeli (dla dużych baz z ograniczeniem ilości rekordów)
- Ewentualne stronicowanie dużej ilości danych
- Dostarczanie wierszy po wartości klucza głównego
- Wkładanie nowych rekordów do bazy,
- Modyfikacja poszczególnego wiersza
- Zliczenie zawartości rekordów w tabeli

Dodatkowo potrzebne będzie zapewnienie możliwości znajdowania klucza głównego dla dowolnego wiersza bazy danych.

4.3.2 Prezentacja danych edytora

System prezentacji danych powinien być podzielony na części logiczne przypominające strukturę relacyjnej bazy danych, takie jak między innymi:

- Reprezentacje tabeli
- Reprezentacje poszczególnych kolumn
- Formularz reprezentujący wiersz
- Kontener prezentujący całość.

Poszczególne komponenty kolumn powinny przechowywać „widgety” do manipulacji danymi. Jeśli w jednym wierszu do jednej kolumny pasuje kilka „widgetów”, użytkownik powinien mieć możliwość wyboru aktualnego. „Widgety” powinny zawierać funkcje, które sprawdzałyby, czy można przetwarzać dane zawarte w danej kolumnie.

Reprezentacja tabeli powinna dostarczać możliwości przeglądania i edycji, kasowania danych oraz dodawania nowych wierszy.

4.3.3 Obsługa różnych typów danych

W modelu relacyjnym jednym z podstawowych wymagań, jest paradygmat o atomowości danych przechowywanych w rekordach. Ograniczona liczba typów danych, w praktyce wykazała słabość modelu relacyjnego[3]. W roku 1992 Standard SQL charakteryzował następujące typy danych zapisane w tab. 4.3.1.

Tab. 4.3.1 Typy danych 1992 Standard SQL

Typy napisowe (String):	CHAR(N), VARCHAR(N). CHAR(N)
Typy liczbowe (Numeric):	INT, BIGINT, FLOAT, DECIMAL FLOAT, DOUBLE, PRECISION, DECIMAL(M,D)
Typy daty i godziny (Datetime):	DATE, TIME, TIMESTAMP.
Interval (typ opisujący przedział czasu)	W MySQL nie występuje.
Logiczna	BIT

Powyższy standard jest niestety bardzo ograniczony i często nie pasuje do nowoczesnych oczekiwań. Dobrym przykładem jest typ VARCHAR, który w niektórych implementacjach baz danych (m.in. MySQL) ograniczony jest do maksymalnej długości 255 znaków. Wiele aplikacji współpracujących z bazami danych (np. Internetowe blogi) potrzebują często zapisywania znacznie dłuższych ciągów znaków.

Dopiero w standardzie SQL:1999 [5]wprowadzono ustandaryzowane typy do obsługi danych binarnych:

- CLOB – (CHARACTER LARGE OBJECT) – duży obiekt znakowy
- BLOB – (BINARY LARGE OBJECT) – duży obiekt binarny

Typ BLOB może być użyty do przechowywania obiektów multimedialnych.

Dodatkowo wiele producentów relacyjnych baz danych wprowadza wiele własnych typów. Przykładem MySQL i PortrageSQL są m.in. typy geometryczne tabela 4.3.2

Tab. 4.3.2 Geometryczne typy danych [14]

Name	Storage Size	Representation	Description
Point	16 bytes	Point on the plane	(x,y)
Line	32 bytes	Infinite line (not fully implemented)	((x1,y1),(x2,y2))
Lseg	32 bytes	Finite line segment	((x1,y1),(x2,y2))
Box	32 bytes	Rectangular box	((x1,y1),(x2,y2))
Path	16+16n bytes	Closed path (similar to polygon)	((x1,y1),...)
Path	16+16n bytes	Open path	[(x1,y1),...]
polygon	40+16n bytes	Polygon (similar to closed path)	((x1,y1),...)

Niestety wiele technologii m.in. JDCB nie obsługuje tych typów. Przez co mają one zastosowanie głównie w wyspecjalizowanych programach.

Problem składowania danych multimedialnych wiąże się często z koniecznością przechowywania dodatkowych informacji np.:

- informacji na temat pochodzenia
- formatu multimediów
- charakterystyki multimediów (rozdzielczość obrazu, liczba kanałów audio itp.)
- opisu treści zawartej w przekazie multimedialnym

Ponieważ multimedia i opisujące je metadane stanowią złożone obiekty, to do ich przechowywania bardziej niż „czysty” model relacyjny odpowiednie są modele obiektowy i obiektowo-relacyjny.

Dodatkowo sposób prezentacji wyników w relacyjnych bazach, jako wynik zapytania, nie jest przystosowany do przechowywania dużych plików takich jak filmy czy utwory muzyczne:

„Prezentacja danych rozumiana, jako udostępnienie wyników zapytań, (…) nie stanowi problemu w tradycyjnych systemach baz danych, ale staje się kluczowym zagadnieniem, gdy udostępnianą zawartością są dane audio lub wideo. Utwory muzyczne i filmy wideo ze względu na duży rozmiar często dostarczane są klientowi strumieniowo. W takim wypadku konieczne jest zapewnianie klientom odpowiedniej, jakości usług i wydajne zarządzanie współbieżnie transmitowanymi strumieniami.” [3]

W chwili obecnej bazy danych nie są przystosowane do intuicyjnej obsługi danych multimedialnych. Dane trzymane są w plikach binarnych w oderwaniu od meta danych. Jedyną informację o wartości binarnej dostarcza nam nazwa kolumny w relacji.

4.3.4 Edytor struktury bazy danych

Minimalna funkcjonalność edytora struktury wynika z relacyjnego modelu danych, opisanego w rozdziale 2. Wynika z niego konieczność:

- Zarządzania bazami danych.(dodawanie, edycja, usuwanie)
- Zarządzania tabelami (dodawanie, edycja, usuwanie)
- Zarządzania kolumnami w relacjach (W tym ich parametrami m.in. typem danych, długością danych, kluczem głównym)
- Zarządzania połączeniami między tabelami (obsługą kluczy obcych)

4.4. Podsumowanie

Ze względu na niezgodność ze standardami w różnych systemach bazodanowych, system musi zapewnić, elastyczny sposób obsługi nowych/niestandardowych typów danych. Wszystkie dane powinny być prezentowane i edytowane w intuicyjny sposób z pomocą kontrolek graficznych.

Dodatki powinny być wielofunkcyjne to znaczy współpracować z różnymi elementami graficznego interfejsu użytkownika i różnymi modelami danych (np. edycji istniejących baz danych, tworzenia nowych wierszy w osobnej tabeli). Główny element prezentacji powinien być dostosowany do obsługi przez interfejs programistyczny. Ze względu na ograniczenia modelu relacyjnego należy zaimplementować dodatki multimedialne obsługujące małe pliki binarne.

Większość danych zapisywanych w bazie wymaga określenia pewnych norm, które określałyby poprawność danych wejściowych. W systemach baz danych posiadamy ograniczenia naturalne związane z typami danych. Jednak często programista wymaga od systemu większych ograniczeń składni np. (prawidłowy format adresu e-mail).

System, więc powinien uwzględniać taką możliwość i zaproponować standard, który umożliwiłby w łatwy sposób weryfikacje danych. Weryfikacja danych winna być określona przez użytkownika aplikacji. Program miałby umożliwiać modyfikację pojedynczych dodatków proponując użytkownikowi dostosowywanie formy walidacji.

5. Zastosowane technologie

W poniższym rozdziale opisano technologie, które zostały użyte do wdrożenia nowego rozwiązania opisanego w rozdziale 4.2. w formie aplikacji. Wybrano język programowania Java, ze względu na ogólnodostępność i nowoczesność. Język ten zawiera komponenty do projektowania graficznego interfejsu (Biblioteki Swing). Następnie opisano technologie zapewniającą dostęp do bazy danych - standard JDCB. Jako bazę danych wybrano bardzo popularną darmową bazę MySQL.

5.1. Język Java

Java to obiektowy język programowania wydany przez firmę Sun Microsystems w roku 1995. Jego składnia wywodzi się z języka C++, a niektóre elementy zostały zaczerpnięte z języka Smalltalk.

Java jest językiem przenośnym, którego kod źródłowy kompiluje się do kodu pośredniego, następnie jest przetwarzany przez abstrakcyjny element pośredni, zwany „Maszyną Wirtualną". Następnie „Maszyna Wirtualna" przetwarza pre-kompilowany kod na język zrozumiały dla danego systemu operacyjnego i procesora. Oznacza to, że dowolny kod jest niezależny od środowiska, w którym się wykonuje, czyli jest wieloplatformowy. W tej chwili wirtualna maszyna Javy jest już dostępna dla większości systemów operacyjnych i procesorów.

Charakterystyczne cechy języka Java to :

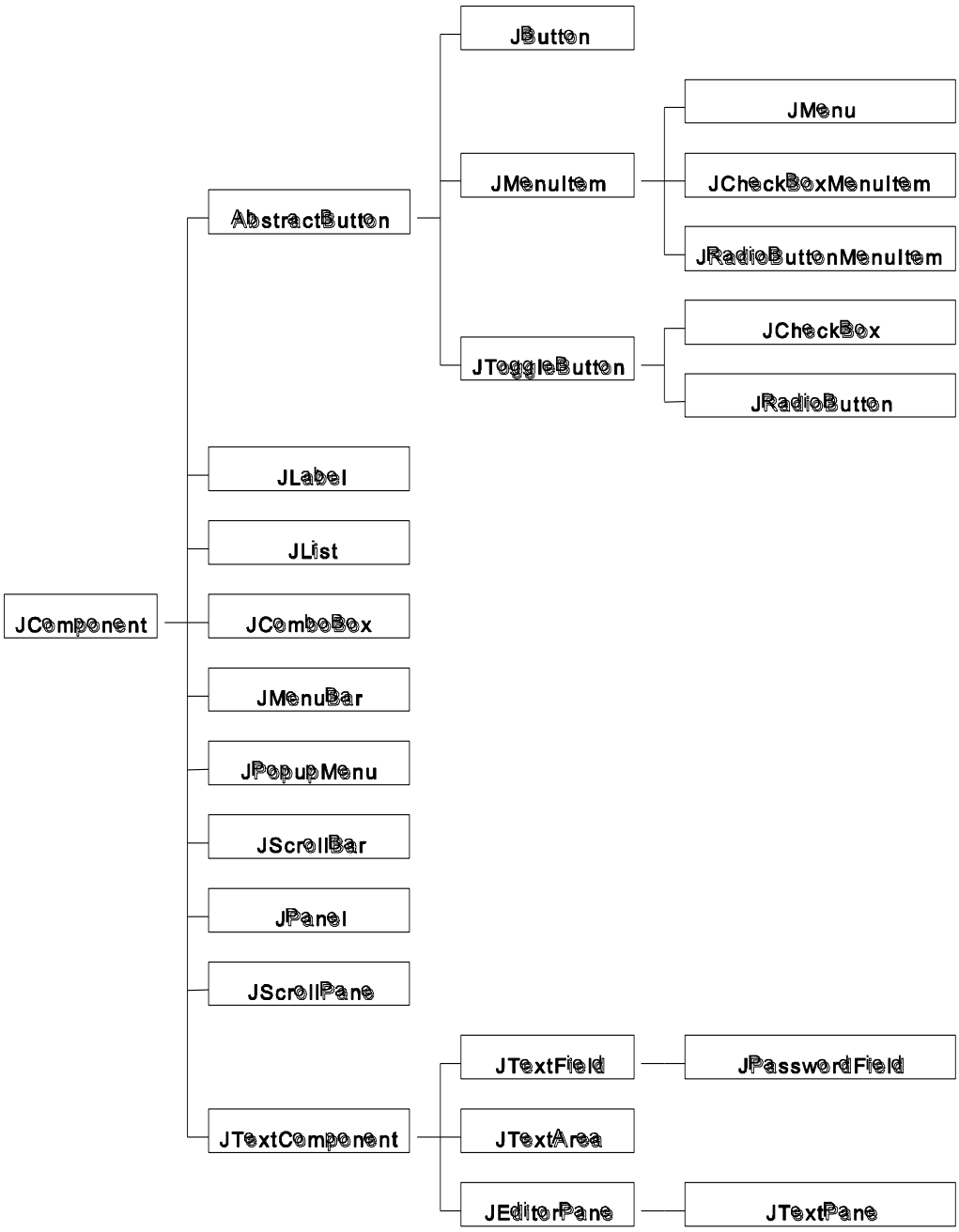
- Pełna Obiektowość- wszystkie dane zapisane są w obiektach, które tworzone są za pomocą opisu zwanego „Klasą". Klasa ma za zadanie w sposób abstrakcyjny modelować cechy rzeczywistych bytów. Cechy te zapisane są w postaci atrybutów i metod. Atrybuty to informacja opisowa wyrażona za pomocą innej klasy lub typu prostego. Metoda natomiast to reprezentacja czynności wykonanych na obiekcie lub ich zbiorze.
- Dziedziczenie – to rozszerzanie istniejącej klasy z możliwością dodania własnej funkcjonalności lub modyfikacji istniejących metod z klasy dziedziczonej. W języku Java zrezygnowano z wielodziedziczenia znanego z C++. Jedna klasa może

dziedziczyć tylko po jednej klasie. W zastępstwie wprowadzono tzw. interfejsy programistyczne. Określają one zbiór abstrakcyjnych cech, które mają być zaimplementowane w danej klasie, tzn. że dana klasa kontraktuje napisanie kodu spełniającego dane cechy.

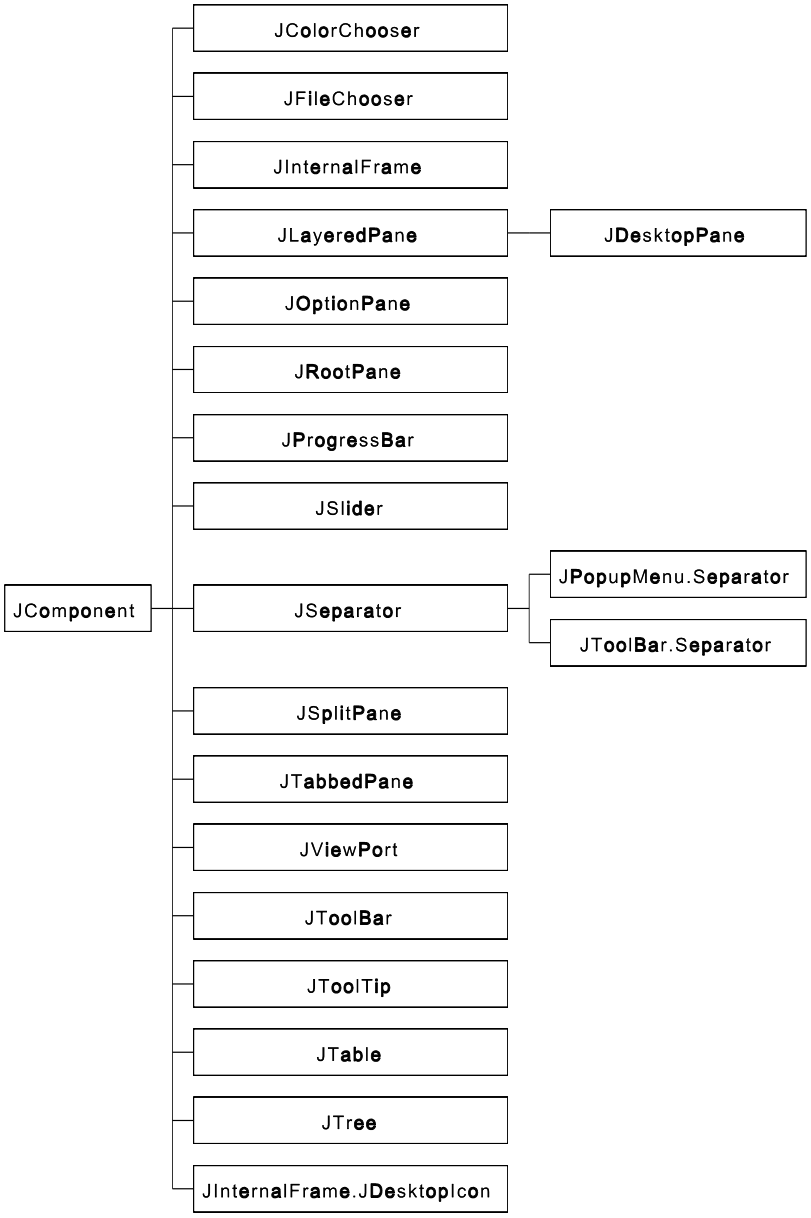
- Polimorfizm – to mechanizm powiązany z dziedziczeniem. Umożliwia on łącznie funkcjonalności klasy dziedziczonej i klasy dziedziczącej. „Idea polimorfizmu bazuje na tym, że użytkownik obiektu nie wie i nie musi wiedzieć, czy konkretne zachowanie wykorzystywanego obiektu zostało zrealizowane bezpośrednio w tym obiekcie czy też w tym, po którym dziedziczy on swoje właściwości.”[15]
- Hermetyzacja – to charakterystyczna własność umożliwiająca ograniczenie dostępu do poszczególnych pól zapisanych w klasach. Umożliwia ustawianie określonej widoczności pól oraz metod na różnym poziomie widzialności.
- Obsługa zdarzeń – za pomocą specjalnych klas nasłuchujących dostarcza mechanizmu obsługi programowania zdarzeniowego to znaczy, że można zaprogramować wywoływanie odpowiednich klas po nastąpieniu określonej czynności.
- Obsługa wyjątków – dostarcza specjalnej składni oraz klas do obsługi możliwych błędów wykonania programów. Dzięki czemu programista może sam obsłużyć zachowanie programu podczas występowania

5.2. Biblioteki SWING

Biblioteka Swing zawiera komponenty graficzne zapewniające możliwość budowania interfejsów graficznych wykorzystywać graficzne komponenty, jako elementy wejścia-wyjścia dla danych. Biblioteka ta wywodzi się z podstawowej implementacji graficznego interfejsu Javy o nazwie AWT (Abstract Widget Toolkit). Cześć jej elementów odpowiada elementom AWT (Rys. 5.2.1), inne są nowymi komponentami(Rys. 5.2.2).



Rys.5.2.1 Komponenty Swingu o rozbudowanych możliwościach w stosunku do AWT
źródło: Magellan Institute Swing Short Course.



Rys. 5.2.2 Hierarchia klas komponentów Swingu, nie mających odpowiedników w AWT źródło: Magellan Institute Swing Short Course.

Biblioteka Swing w znaczący sposób korzysta z architektury wzorca Model / View / Controller [9], koncepcja ta wspomniana w rozdziale 2, koncepcyjnie oddziela płaszczyznę danych od kontrolek interfejsu, czyli wyglądy oraz logiki aplikacji. Z tego powodu większość komponentów Swing zawiera klasy odpowiedzialne tzw. modele danych, (które są określone w postaci interfejsów programistycznych języka Java), a programista może korzystać z różnych domyślnych implementacji lub stworzyć własne.

Tab. 5.2.1 Przedstawienie interfejsów reprezentujących model dla danych komponentów Swing

Component	Model Interface	Model Type
JButton	ButtonModel	GUI
JToggleButton	ButtonModel	GUI/data
JCheckBox	ButtonModel	GUI/data
JRadioButton	ButtonModel	GUI/data
JMenu	ButtonModel	GUI
JMenuItem	ButtonModel	GUI
JCheckBoxMenuItem	ButtonModel	GUI/data
JRadioButtonMenuItem	ButtonModel	GUI/data
JComboBox	ComboBoxModel	Data
JProgressBar	BoundedRangeModel	GUI/data
JScrollBar	BoundedRangeModel	GUI/data
JSlider	BoundedRangeModel	GUI/data
JTabbedPane	SingleSelectionModel	GUI
JList	ListModel	Data
JList	ListSelectionModel	GUI
JTable	TableModel	Data
JTable	TableColumnModel	GUI
JTree	TreeModel	Data
JTree	TreeSelectionModel	GUI
JEditorPane	Document	Data
JTextPane	Document	Data
JTextArea	Document	Data
JTextField	Document	Data
JPasswordField	Document	Data

Architektura modeli danych opiera się na trzech założeniach:

1. Programista może skorzystać z domyślnie zaimplementowanych modeli danych, reprezentowanych za pomocą klas z przedrostkiem Default. Np. DefaultListModel
2. Programista może skorzystać z klasy abstrakcyjnej, która zawiera implementacje niektórych mechanizmów odsługujących GUI.
3. Całkowicie stworzyć własną klasę opartą na interfejsie programistycznym z tab.5.2.1

Warto zauważyć, że zmiany modeli wymagają użycia specjalnych metod, które odświeżałyby elementy widoku. Zazwyczaj są one już zaimplementowane w klasach abstrakcyjnych ad. pkt 2. Metody te mają zazwyczaj określone nazwy zaczynające się od przedrostka fire. Korzystanie z czystych interfejsów ad. 3 wymaga całościowego zaimplementowania metod odświeżających widok. W modelu należy także obsłużyć przechowywanie tzw. klas nasłuchujących zdarzenia tzw. listenerów.

Biblioteka pod względem zarządzania i rozmieszczenia elementów graficznych posiada mechanizmy zwane zarządcami rozkładu (ang. Layout Manager). Zarządcy rozkładu to komponenty, które w sposób dynamiczny określają reguły zarządzania odległościami pomiędzy elementami oraz sposoby, w jaki element reaguje na zmianę rozmiarów okna. Często zarządcy rozkładów wpływają na elementy, które zawierają, zmieniając rozmiar, sposób wyświetlania niezależnie od ustalonych wcześniej rozmiarów samego elementu. Prawidłowe użycie zarządców rozkładu jest zagadnieniem dość trudnym. Wymaga dużej wiedzy i doświadczenia empirycznego .

Wyróżniamy zastępujące rozkłady:

- FlowLayout – aranżuje komponenty w sposób podobny do tekstu w akapicie
- BoxLayout – zarządca pozwala na rozłożenie komponentów w osiach pionowych lub poziomych.
- BorderLayout – layout pozwalający na prezentacje
- GridLayout – pozwala na równomierne rozmieszczenie elementów.
- Spring Layout rozkład opierający się na zależnościach pomiędzy elementami poprzez ustanawianie ograniczeń.
- GroupLayout – jest rozkładem, zaprojektowanym do użytku przez generatory kodu Gui typu WYSIWYG m.in. jest wykorzystywany przez środowisko NetBeans.

5.3. Standard JDBC

W 1996 roku firma Sun udostępniła pierwszą wersję pakietu Java Database Connectivity (JDBC). Pakiet ten umożliwia programistom łączenie się z bazami danych oraz wykonywanie zapytań i aktualizację danych w języku SQL(Structured Query Language), który jest standardowym językiem dostępu do baz danych.

Java wraz z JDBC zapewnia dużą mobilność i elastyczność w stosunku do istniejących wcześniej środowisk tworzenia aplikacji baz danych, ponieważ powstałe aplikacje są niezależne od platformy, na której działają, jak i od bazy danych, której używają.

Dzięki jednemu standardowi API programista nie musi poznawać specyfikacji każdej bazy danych z osobna, a migracja z jednego silnika baz danych do drugiego jest znacznie łatwiejsza.

Dane, z których korzysta program mogą znajdować się w zaawansowanym systemie zarządzania bazami danych, takimi jak Microsoft SQL Server czy Oracle, ale i w prostej bazie wbudowanej w urządzenie.

Jest to istotna zaleta w porównaniu z tradycyjnymi sposobami tworzenia aplikacji baz danych, które prowadzą do powstania aplikacji korzystających jedynie z systemu zarządzania baz danych konkretnego producenta i działających jedynie na jednej lub na dwu platformach.

Interfejs programowania JDBC API implementowany jest przez producentów baz danych. Dostarczają oni sterowniki zazwyczaj w postaci plików z rozszerzeniem jar. Sterownik należy dodać do ścieżki klas (classpath) danego projektu. Dzięki czemu uzyskujemy dostęp do różnorodnych interfejsów programistycznych. Umożliwiają one:

- Uzyskanie połączenia z bazą danych, (za pomocą interfejsu Connection)
- Tworzenie i wykonywanie zapytań SQL, (za pomocą interfejsu Statement, i PreparedStatement).
- Przetwarzanie otrzymanych wyników (za pomocą interfejsu ResultSet)

Zaimplementowany interfejs Connection reprezentuje połączenie do bazy danych. Jedną z wad JDBC jest właśnie, ciągłe utrzymywanie połączenia z bazą.

Obiekty Statement oraz PreparedStatement są narzędziami do konstruowania poleceń SQL. Statement umożliwia wykonywanie prostych zapytań, natomiast PreparedStatement pozwala na parametryzowanie, czyli uzupełnianie gotowych zapytań w sposób dynamiczny podczas pracy programu.

Obiekt ResultSet, jak nazwa wskazuje obsługuje i dostarcza metod dla zbioru wyników zapytań. Pozwala na iterowanie po tym zbiorze oraz dostarcza metod do mapowania typów bazy danych na typy języka programowania.

5.3.1 System transakcji JDBC

W JDBC domyślnie nie jest ustawiony na transakcyjny model wykonywania operacji. Każda operacja jest zatwierdzana automatycznie (ang. autocommit) po wywołaniu metod execute/executeUpdate i pomyślnym zatwierdzeniu przez serwer bazy danych.

Aby wykonywać transakcje kilku operacji, jako wartości atomowej (zatwierdzenie wszystkich zmian lub żadnej). należy wywołać metodę setAutoCommit obiektu klasy Connection z parametrem false. Następnie po wykonaniu ciągu poleceń funkcją execute zatwierdzamy transakcję metodą commit, lub w całości wycofujemy metodą rollback.

5.3.2 Poziomy izolacji transakcji

System transakcyjny zaimplementowany w JDBC daje możliwość ustalenia poziomu izolacji. Ustalamy go za pomocą metody `Connection.setTransactionIsolation (int value)`, gdzie parametr `value` wyznaczany jest za pomocą stałych:

- TRANSACTION_NONE
- TRANSACTION_READ_COMMITTED
- TRANSACTION_READ_UNCOMMITTED
- TRANSACTION_REPEATABLE_READ
- TRANSACTION_SERIALIZABLE

Zastosowanie wybranego poziomu izolacji musi być wspierane przez dany system baz danych.

5.3.3 Wycofywanie części transakcji

Specyfikacja JDCB od wersji 3.0 dostarcza nowej metody przetwarzania transakcji. Metoda savepoint umożliwia wycofanie (ang Rollback) części transakcji, a nie całej. Savepoint wskazuje wewnętrzny punkt transakcji. Ustawia się go za pomocą unikalnej nazwy i podczas wykonania pozwala na zatwierdzanie części zapytań, a cofnięcia reszty.

5.4. Język SQL i problem wielu dialektów

Język SQL (ang. Structured Query Language) to deklaratywny język zapytań dla relacyjnych baz danych. Definiuje on czynności, które należy wykonać, zamiast sposobu, w jaki należy to zrobić. Język SQL składa się z trzech części:

1. DML (ang. Data Mainpulation Language),
2. DDL (ang. Data Definition Language) ,

3. DCL (ang. Data Control Language).

Ad. 1 DML to najbardziej charakterystyczne zapytania SQL.

Zapytanie te służą do:

- Pobierania danych z bazy (zapytania SELECT),
- Dodawania nowych rekordów (INSERT),
- Modyfikowania istniejących danych (UPDATE) ,
- Usuwania (DELETE) danych z bazy.

Ad.2 Zapytania DDL służą do manipulacji strukturami bazodanowymi (np. tabelami, indeksami). Są to m.in. zapytania CREATE, DROP oraz ALTER.

Ad.3 DCL to zbiór zapytań służących do zarządzania uprawnieniami. Użytkownicy bazy danych mogą bowiem mieć dostęp tylko do wybranych obiektów baz danych. Najważniejsze zapytania DCL to: GRANT, REVOKE oraz DENY.

5.5. Baza danych MySQL

MySQL to darmowa wieloplatformowa baza danych o udostępnionym kodzie wspierana przez firmę Oracle. Ta baza danych wspiera model relacyjny oraz wiele typów danych. Charakteryzuje się ona wieloplatformowością oraz nastawieniem na szybkie wykonywanie zapytań. Dostępna jest na następujące platformy:

- Microsoft Windows
- SuSE Linux Enterprise Server [12]
- Red Hat
- Linux
- Mac OS X
- FreeBSD

Baza ta wyróżnia się różnymi silnikami do obsługi przeznaczonymi do przechowywania danych. Każdy z nich ma inną charakterystykę (Tab. 5.5.1).

Tab. 5.5.1 Silniki MySQL

InnoDB	• InnoDB- Jest domyślnym silnikiem zaprojektowanym na obsługę transakcji. • Wspomaga odzyskiwanie danych po awarii • Automatycznie wymusza integralność danych • Obsluguje współbiezny dostep do bazy • Wspomaga połączenia między relacjami. (klucze obce)
Cluster	• Wspomaga transakcje • Nie obsługuje kluczy obcych • Pozwala na replikacje • Potrafi zapewnić dostępność danych w 99.999%
MyISAM	• Nie wspiera transakcji • Bardzo szybki dostęp • Wspiera rozbudowane indeksy B-drzewa, pełno-tektowy , indeks GIS (obsługi typów geometryczny) • Nieograniczana przestrzeń do przechowywania danych
Archive	• przeznaczony do archiwizacji danych oszczędza 20% przestrzeni dyskowej • nie wspiera instrukcji UPDATE/DELETE/REPLACE • nie wspiera odzyskiwania danych po wypadku.
Blackhole	• System do trwałego usuwania danych • Wszystkie dane wysłane go tego silnika zostaną zniszczone
CSV	• Silnik zapisujący dane do pliku CSV(Dane oddzielone przecinkami) • Nie wspiera transakcji

Federated	• System do łączenia wielu odległych baz danych
Memory	• To baza w pełni działająca w pamięci RAM • Nietrwała (po wyłączeniu serwera traci dane) • Brak obsługi transakcji • Nie może przechowywać danych binarny BLOB I typu tekstowe TEXT
Merge	• Silnik do połączenia dwóch lub więcej baz typu MyISAM w tabele logiczną • Brak transakcji • Nie wspiera odzyskania danych po wypadku

Producent dostarcza następujących sterowników do obsługi bazy danych przez developerów:

- Connector/ODBC – sterownik w standardzie (Open DataBase Connectivity)
- Connector/J – sterownik obsługujący API JDBC
- Connector/Net – sterownik dla języka .NET
- Connector/C++ - sterownik dla języka C++.
- Connector/C (libmysql) - sterownik przeznaczony dla języka C.

Typy danych MySQL

MySQL posiada rozszerzony model typów w stosunku do standardu SQL:92 (standard ten posiada najbardziej bazowe typy danych). Zestaw ten posiada

- Dane tekstowe CHAR, VARCHAR, Cztery typy TEXT, ENUM and SET
- Dane Binarne BINARY i cztery rodzaje Binary Large Object (BLOB).
- Numeric: BIT, TINYINT, SMALLINT, MEDIUMINT, INT, BIGINT, trzy typy FLOAT, DOUBLE, DECIMAL(Ułamek).
- • Datetime: wartość zapisana w milisekundach określająca datę. MySQL ma pięć typów obsługujących datę: DATE, TIME, DATETIME, TIMESTAMP, i YEAR.
- • OpenGIS – typy danych opisujące geograficzne położenie oraz typy do tworzenia map.

6. Prototyp nowego rozwiązania.

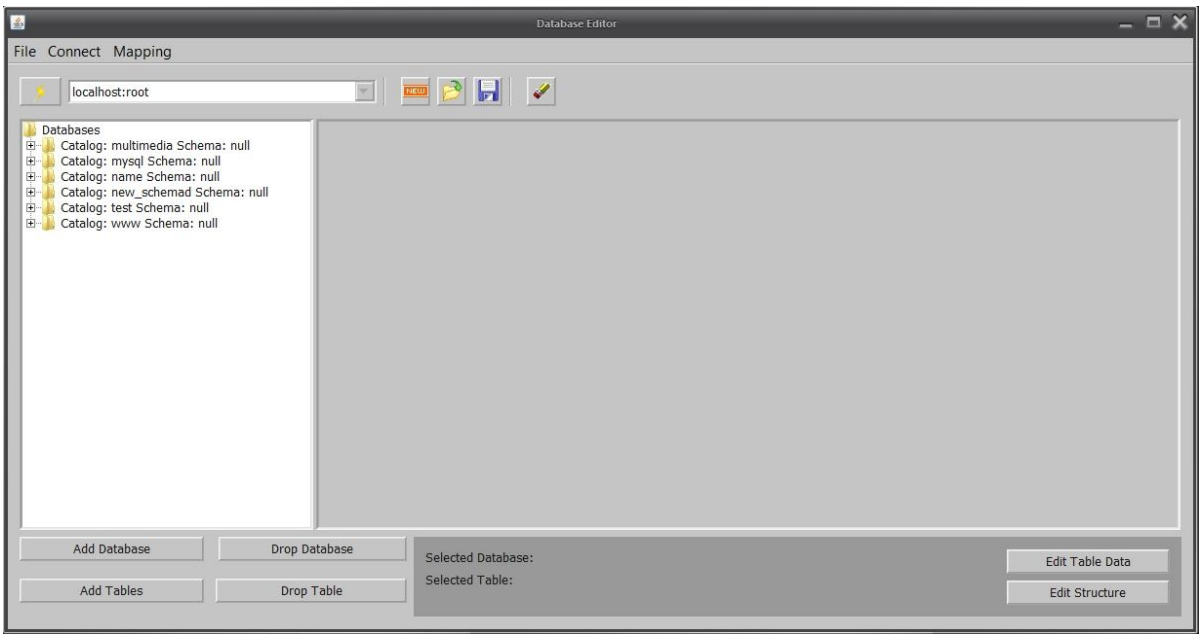
W poniższym rozdziale przedstawiono propozycję rozwiązania, aplikacje do edycji danych oraz struktury bazy danych rozszerzającej model prezentacji i edycji bazy danych. Zaprezentowano rozwiązania techniczne przyjęte w prototypie oraz trudności napotkane podczas implementowania rozwiązań. Na końcu rozdziału zaproponowano możliwości rozwoju aplikacji.

6.1. Prezentacja interfejsu użytkownika

W widoku głównym użytkownik widzi trzyczęściowy interfejs (Rys. 6.1.1).

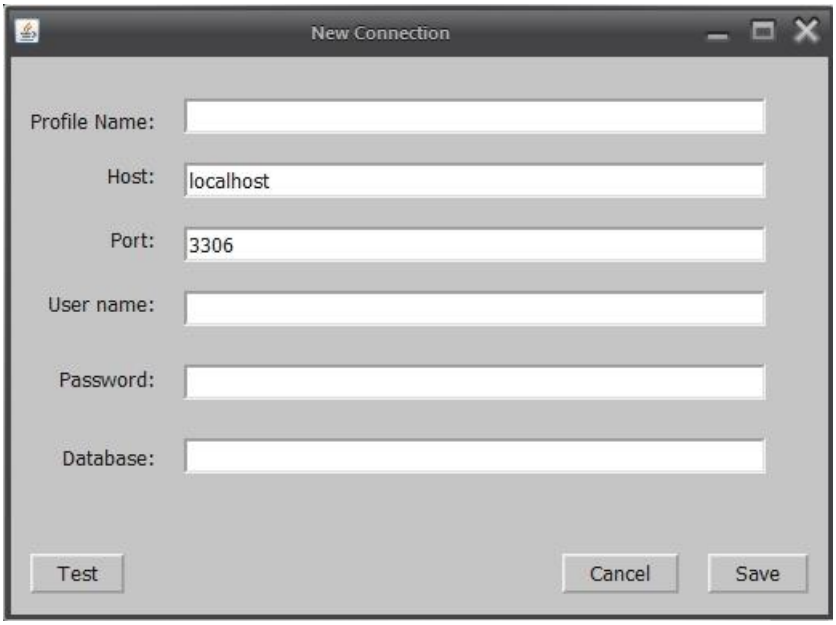
- Panel górny obsługujący połączenia z bazą danych
- Panel główny, podzielony na dwie części
- Panel dolny do obsługi operacjami na strukturze bazy.

Panel górny zapewnia podstawową funkcjonalność w zakresie zarządzania połączeniami do baz danych. Pozwala na stworzenie nowego połączenia z bazą danych za pomocą formularza w osobnym oknie.



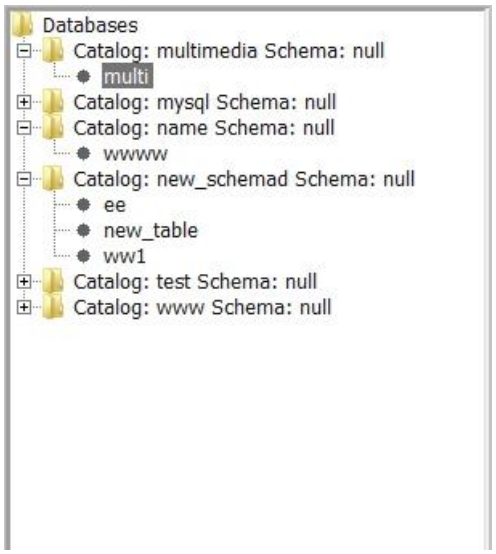
Rys 6.1.1 Ekran główny interfejsu użytkownika

W formularzu widocznym na rysunku 6.1.2 użytkownik wpisuję najpierw nazwę, pod jaką ma się wyświetlać nasze połączenie na ekranie głównym. Następnie nazwę serwera, z którym się chce się połączyć. W dalszym ciągu wybiera port bazy danych, pod którym nasłuchuje serwer. Dalej uzupełniamy dane użytkownika login i hasło naszego konta w bazie danych. Po wypełnieniu tych danych możemy przeprowadzić test wpisanych danych wybierając przycisk „Test”. Jeśli połączenie się powiodło możemy wybrać opcje „Save”- zapisz, aby dodać połączenie do listy rozwijalnej `JCombobox` w panelu górnym.



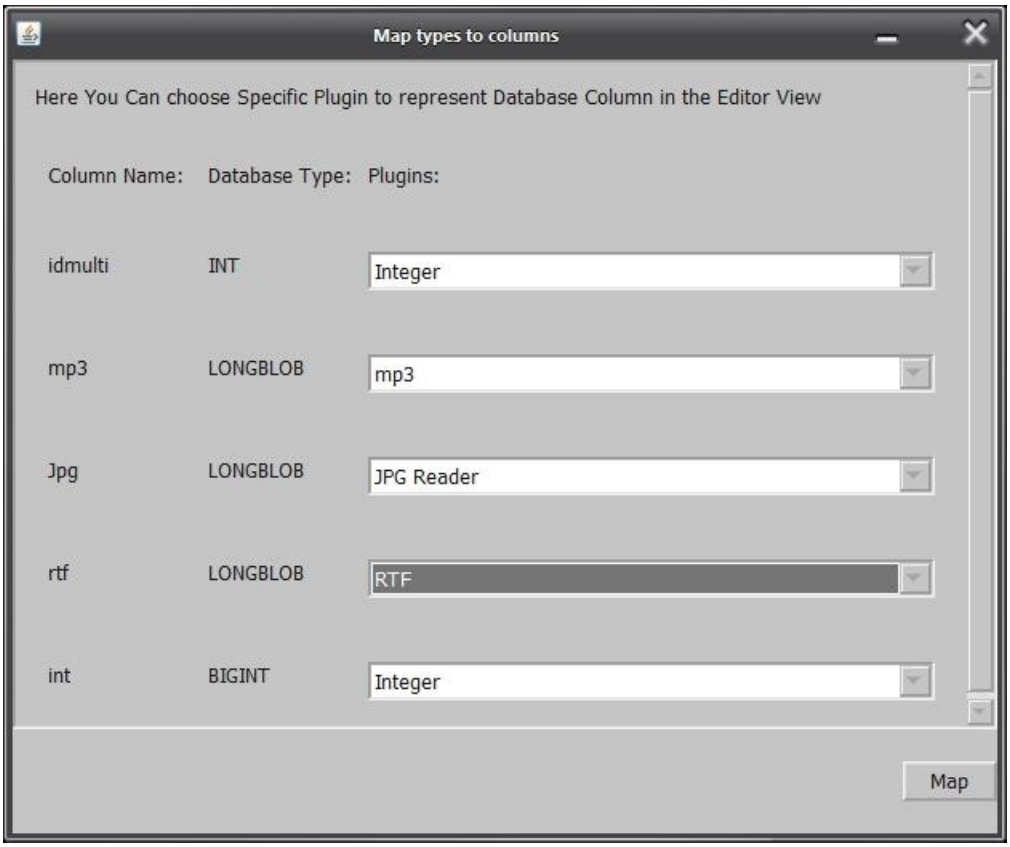
Rys. 6.1.2 Widok kreatora nowego połączenia

Wybranie istniejącego połączenia z listy i wybranie przycisku z ikoną pioruna po lewej stronie spowoduje inicjacje programu. Po lewej stronie panelu głównego pojawi się struktura bazy danych w formie komponentu zwanego drzewem (Tree). Komponent ten (Rys. 6.1.3) zawiera hierarchie struktur zawartych w bazie danych. A dokładnie reprezentacje baz danych i tabel (relacji) w nich zawartych.



Rys. 6.1.3 Komponentu nawigacji po bazach danych zwany drzewem

Po wyborze określonej bazy danych i tabeli wybór ten będzie wyświetlony na dolnym panelu informacyjnym. Automatycznie zostanie wywołana opcja edytowania zawartości bazy danych i pojawi się ramka pozwalająca na wybór dodatków do reprezentacji poszczególnych dodatków (Rys. 6.1.4). Dodatki są przeselekcjonowane, dlatego możemy wybrać tylko dodatki przypisane do poszczególnego typu bazy danych. Ponieważ użytkownik zazwyczaj zna zawartość bazy i specyfiki typów danych, które musi obsłużyć założono, że to właśnie on najlepiej wybierze odpowiedni dodatek (często nazwa typów kolumn jest informacją o typie danych, jakie się w nim znajdują).



Rys. 6.1.4 Komponent do przypisywania dodatków do obsługi bazy danych.

Po wybraniu określonych typów w list rozwijalnych `(JCombobox)` i wybraniu przycisku „Map”, w panelu głównym po prawej stronie pojawia się edytor danych. Przypomina on arkusz kalkulacyjny. Po wybraniu określonej komórki w tabeli edycji w zależności od typu danych i dodatku, pojawi się wyskakujące okienko do prezentacji i edycji danych lub w przypadku typów prostych edycja zacznie się w komórkach tabeli(wartości liczbowe, znakowe). Szczegółowy opis dodatków zamieszczono w dalszej części w podrozdziale 6.1.2.

Rys. 6.1.5 Panel dolny, informacja o wybranej tabeli

Jeśli użytkownik wybierze opcje „Edit Structure” w prawym dolnym rogu panelu dolnego (Rys. 6.1.5) to w prawej części panelu głównego pojawi się edytor struktury bieżącej

tabeli (o nazwie zapisanej w dolnym panelu części środkowej). Edytor struktury opisany jest dokładniej w dalszej części podrozdziału 6.1.3.

W panelu dolnym znajdują się też opcje do zarządzania strukturą bazy:

- Add Database- uruchamia osobną ramkę do projektowania nowej bazy danych.
- Drop Database- pozwala na usunięcie wybranej bazy danych. (widocznej w panelu po prawej stronie)
- Add Table – pozwala na projektowanie nowej tabeli w bazie danych.
- Drop Table – usuwanie tabeli (poprzednio wybranej).

6.1.1 Widok edycji danych

Ma formę dwóch tabel jedną przeznaczoną do edycji istniejących wierszy, a drugą - pojedyńczy wiersz do wprowadzania nowych danych. Każdej kolumnie w obu tabelach przypisane są tzw. edytory komórek, które odpowiadają za edycje danego rekordu. Edytory komórek muszą być zaprojektowane tak, aby były niezależne od modelu danych. Po wybraniu określonej komórki z tabeli edycji lub z tabeli przeznaczonej do dodawania nowych wierszy, w zależności od struktury dodatku, pojawi się edytor w możliwej jednej z trzech form:

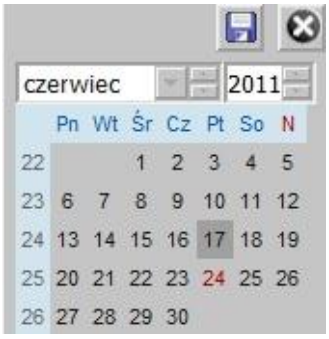
- Edycja bezpośrednio w komórce tabeli (typy liczbowe lub znakowe)
- Osobna ramka- poza wyskakującym okienkiem, dane mogą być prezentowane w zewnętrznej ramce.
- „Wyskakujące okienko” - jest to specjalnie dostosowane okienko typu JPopupMenu, które niezależnie od ramki aplikacji pokazuje zaawansowany komponent do edycji danych. Wyskakujące okienko w połączeniu z przyciskiem jest bardzo intuicyjną formą prezentacji danych, wzorowanych na stronach WWW np. przez kontrolki obsługujące datę.

Edytor po zatwierdzeniu edycji (np. za pomocą klawisza enter) od razu próbuje wykonać automatycznie instrukcje update na bazie danych. Jeśli dane są nieprawidłowe na ekranie pojawi się odpowiedni komunikat.

6.1.2 Opis dodatków dołączonych do bazy

Na obecną chwilę aplikacja dostarcza następujące dodatki:

- Dodatek obsługi String to najprostszy standardowy odczyt, zapisany w formie tekstowej lub liczbowej.
- Dodatek Integer– to dodatek obsługujący wartości liczbowe całkowite.
- Dodatek BigInteger – dodatek obsługujący duże liczby całkowite.
- Dodatek Double – obsługa liczb zapisanych w formie ułamkowej w zapisie dziesiętnym.
- Dodatek Checkbox – jest to dodatek do wszelkiego rodzaju komórek logicznych. (reprezentuje wartości prawda/ fałsz)
- Dodatek Date – jest to dodatek do obsługi dat (Rys. 6.1.6). Oparty jest na komponencie zewnętrznym, kontrolce opensourcowej `JCalendar`. `JCalendar` pozwala w intuicyjny sposób wybrać datę, rok za pomocą kontrolki `JSpiner`, miesiąc za pomocą listy rozwijalnej `JCombobox` oraz dzień w bezpośrednio z kalendarza. Oczywiście od razu widać, która data reprezentuje jaki dzień tygodnia. Dodatek ten zaprezentowany jest w formie „wyskakującego okienka". Dzięki czemu jego użytkowanie jest bardzo wygodne.



Rys. 6.1.6 Dodatek do zmiany daty

- Dodatek TimeStamp – to rozszerzenie dodatku `JCalendar` o wartość czasu (Rys. 6.1.7). Dodatkowo na górze kontrolki pojawia się komponent `JSpiner` przeznaczony do

wyboru czasu. Prezentacja również odbywa się za pomocą komponentu wyskakującego okienka.

Rys. 6.1.7 Komponent obsługujący typ TIMESTAMP

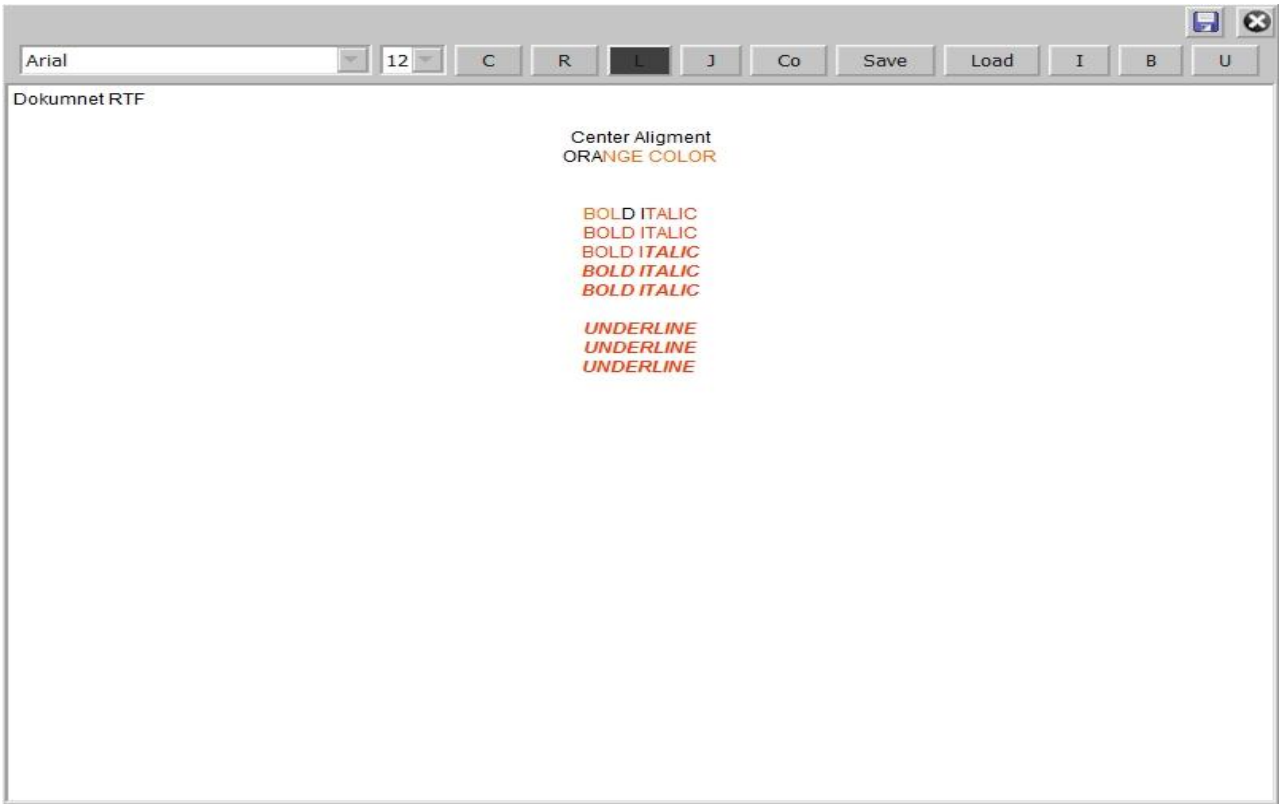
- Dodatek JPG - to dodatek do obsługi plików graficznych o kodowaniu Jpeg (Rys. 6.1.8). Posiada możliwość edycji danych takich jak operacje wejścia wyjścia (wczytywanie nowych rysunków, zapisywanie ich na dysku), jak również prostą edycję skalowania rysunku oraz obracania w dowolną stronę o kąt 90 stopni.

Rys. 6.1.8 Dodatek edytujący rysunki JPG

- Dodatek MP3 - to dodatek, który umożliwia odtwarzanie ze strumienia bazy plików muzycznych o kodowaniu mp3 (Rys. 6.1.9). Pozwala na szybkie, rozpoczęcie odtwarzania, pauzowanie, oraz zastopowanie (odtwarzanie i możliwość rozpoczęcia odtwarzania od nowa). Poza tym pozwala na szybkie operacje wejścia wyjścia (wczytywanie plików, zapisywanie ich na dysku).

Rys. 6.1.9 Dodatek MP3

- Dodatek RTF - to dodatek w formie edytora tekstów oparty na zewnętrznej bibliotece autorstwa Rod Higginsa opublikowanego o opublikowanych źródłach w 2002r (Rys. 6.1.10). Został dostosowany do potrzeb aplikacji. Charakteryzuje się on następującą funkcjonalnością:
  - Dodatek ten może wczytywać, zapisywać dane w bazie danych
  - Wczytywać i zapisywać tekst RTF do plików
  - W pełni edytować tekst
  - Wyrównywać tekst (L-lewa, R - prawa, C - środek, J - Justowanie)
  - Kolorować tekst(Przycisk Co)
  - Obsługiwać różne czcionki (Lista rozwijalna)
  - Modyfikować czcionki o pogrubienie(Bold), podkreślenie (Underline) oraz kursywę (Italic) .



Rys. 6.1.10 Dodatek prezentowany jest w formie wyskakującego okienka.

Wszystkie dodatki prezentowane w formie „Wyskakującego okienka" umieszczone są w kontenerze, w którym w prawym górnym rogu umieszczone są dwa przyciski (Rys. 6.1.11). Przyciski te odpowiadają za zakończenie edycji danej komórki. Jeśli wybierzemy przycisk z dyskietką, zmiany naszej edycji zostaną wysłane do modelu, jeśli wybierzemy krzyżyk to zmiany nie zostaną zapisane.

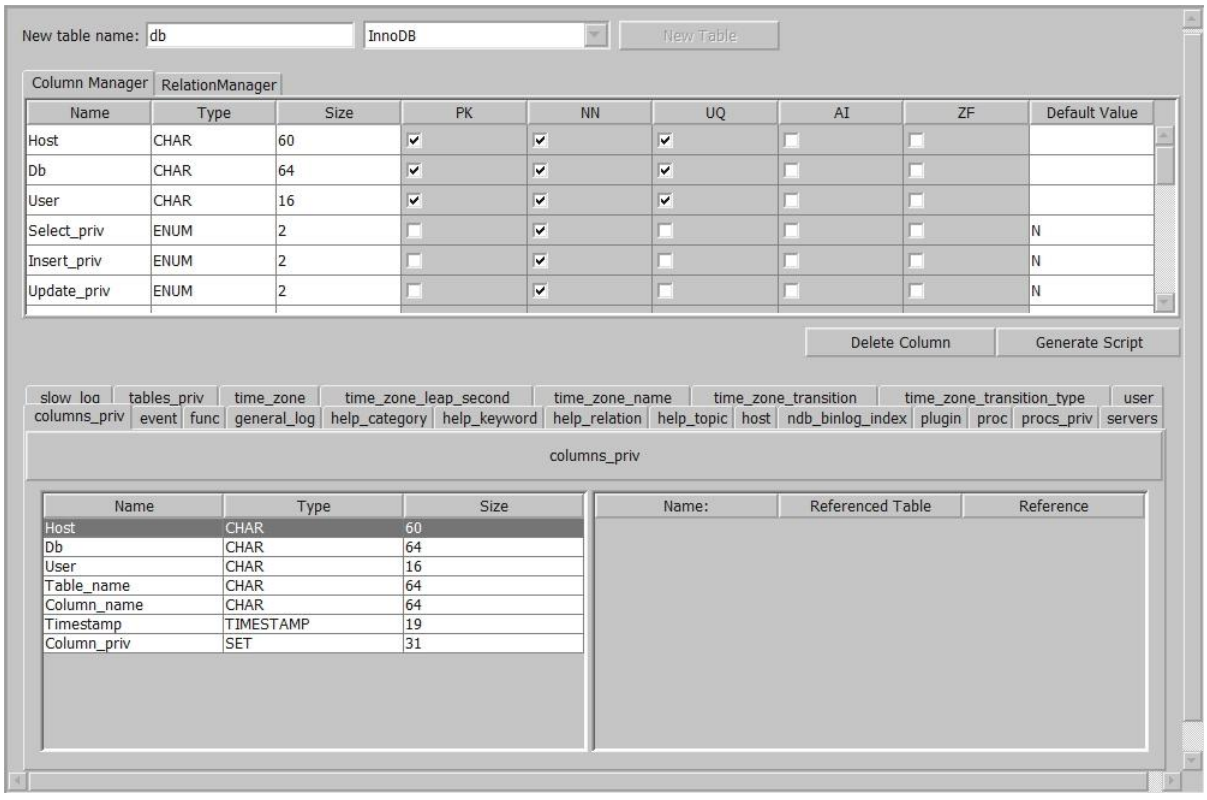
Rys. 6.1.11 Przyciski odpowiadające za potwierdzenie lub odrzucenie zmian w kontrolce.

6.1.3 Edytor Struktury danych

Po wcześniejszym wybraniu tabeli oraz przycisku „Edit Structure" (Rys. 6.1.5) przechodzimy do widoku edycji struktury tabeli. W górnej części mamy możliwość zmiany nazwy tabeli oraz wybranie silnika baz danych (dostępne w wersji MySQL).

Poniżej w komponencie `JTabbedPane` umieszczono tabele do edycji kolumn bazy danych. Edytor ten (Rys. 6.1.12) posiada atrybuty:

- Name – nazwę kolumny.
- Type – umożliwiający wybranie typu danych.
- Size- określający rozmiar np. długość napisu. (Posiada dodatkową funkcje opisaną poniżej)
- PK – (Primary Key) - definiujący czy dana kolumna jest kluczem głównym.
- NN- (Not Null) – zapisujący czy kolumna może przyjmować wartości null.
- UQ – (Unique) – nakładający indeks, który powoduje, że wartości w tej kolumnie nie będą mogły się powtarzać.
- AI – (Auto Increment) – wyznaczający automatyczny przyrost wartości liczbowych o 1 (jeden).
- ZF- (Zero Fill) – wypełniający puste wartości liczbowe zerami.
- Default Value- zawierający dane o wartości inicjacyjnej dla nowych wierszy.

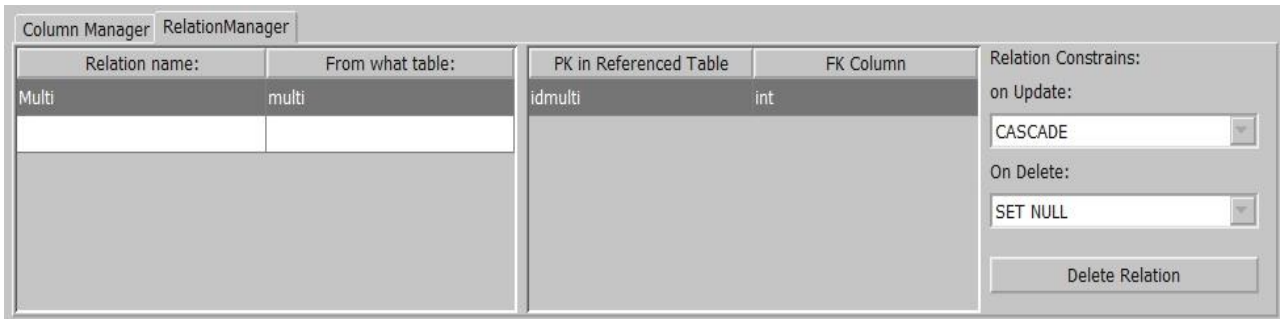


Rys. 6.1.12 Edytor struktury bazy danych

Podczas tworzenia niektórych kolumn używamy typów złożonych, parametryzowanych kilkoma wartościami (więcej niż jednym parametrem „Size"). Aby poprawnie tworzyć takie kolumny należy wpisać je w nawiasie „`()`" w kolumnie „Size". Zapis parametrów jest taki sam, jak w instrukcjach języka SQL. Zgodny dokumentacji serwera baz danych. Do typów złożonych należą m.in.

- ENUM- enumeracja tworzymy ją według przykładu `('wartosc1','wartosc2')`
- SET – zbiór wartości znakowej `("wartosc1","wartosc2")`
- DECIMAL, FLOAT, DOUBLE PRECISION- typy liczbowe z określonym przecinkiem tworzymy określając parametr ilości cyfr w liczbie (M), oraz ilość cyfr po przecinku (D). Parametr M musi być większy od parametru D np. `(12,3)`. Wyrażenie to oznacza, że tworzymy cyfrę o liczbie 9 cyfr przed przecinkiem oraz 3 cyfr po przecinku. W zależności od typu występują zróżnicowane wartości dopuszczalne parametrów M i D.
  - DECIMAL – M(od 1 do 65), D(od 0 do 30)
  - FLOAT – M(od 1 do 255), D(0 to 23)
  - DOUBLE PRECISION – M(od 1 do 255), D(0 do 53);

W komponencie `JTabbedPane` posiadamy też drugą zakładkę o nazwie „Relation Manager" (Rys. 6.1.13). Zakładka zarządza połączeniami między relacjami, czyli tabelami. Stworzenie nowego połączenia polega na wypełnieniu dowolną unikalną nazwą „Relation Name".
W następnej kolumnie wybieramy tabelę, z którą chcemy zawrzeć połączenie. Natychmiast w tabelce obok pojawią się klucze główne wybranej tabeli, którym możemy przyporządkować kolumny w nowego klucza obcego- czyli jednej z kolumn bieżącej tabeli.



Rys. 6.1.13 Edytor połączeń między tabelami

Z prawej strony możemy określić tzw. więzy integralności zawartego połączenia i w razie potrzeby skasować zaznaczone połączenie.

Aby dobrze zaprojektować połączenia w dolnej części całego edytora struktury wprowadzono przeglądarkę wszystkich tabel poza tą, którą obecnie edytujemy. Dzięki czemu łatwiej zaplanować nowe połączenia, ponieważ możemy swobodnie uzyskać informację o typie i długości danych.

Aby dodać zmiany do bazy danych należy wybrać opcje „generate script", wtedy odpowiednia instrukcja typu Alter przeprowadzi edycje danych tabeli.

6.1.4 Dodawanie nowej tabeli

Dodawanie nowej tabeli do bazy danych odbywa za pomocą tego samego komponentu, co edycja. Na początku musimy wpisać nazwę tabeli (unikalną), a następnie wypełnić pola kolumn i połączeń między tabelami, podobnie jak podczas edytowania tabeli.

6.1.5 Projektowanie nowej bazy danych.

Projektowanie nowej bazy danych zaczynamy od wyboru nazwy bazy. Następnie wybieramy przycisk „Dalej" i pojawia nam się nowy rozbudowany komponent (Rys. 6.1.14), znany z edycji struktury tabeli (podrozdział 6.1.3). Komponent ten jest zmodyfikowany i pozwala na tworzenie wielu tabeli. Następnie po wybraniu przycisku „Add Table to Model" stworzona tabela z górnej części zostaje przeniesiona do dolnej części. Użytkownik może dodawać także wiele tabel, jak również ustalać połączenia między nimi.

Rys. 6.1.14 Widok projektowania bazy danych

Wybraną tabelę z dolnej części możemy ponownie z edytować w górnej części po wybraniu opcji „Edit Table”. Możemy także ją wykasować z modelu przyciskiem „Delete from Model”

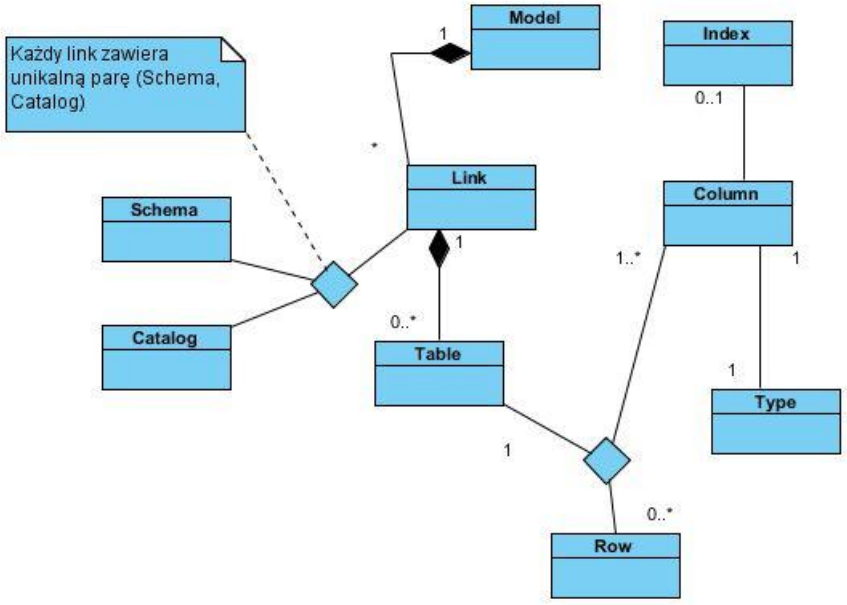
Po zaprojektowaniu wszystkich tabel wybieramy przycisk „Generate Script” aby wykonać polecenia tworzące bazę, jeśli zostały dobrze zaprojektowane.

6.2. Rozwiązania Techniczne

Podczas implementacji aplikacji spotkano się z wieloma zagadnieniami programistycznymi a także, z następującymi problemami:

- Pobieranie Metadanych z bazy i tworzenie modelu obiektowego reprezentującego go.
- Mapowanie typów bazy danych na typy języka programowania.
- Dynamiczne mapowanie typów danych języka na zapytania.
- Stworzenie silnika dodatków
- Mechanizm tworzenia nowych dodatków

Pierwszym modułem systemu jest moduł pobierający z bazy danych informacji o jej schemacie i strukturze. System za pomocą narzędzi dostępnych w sterowniku JDBC pobiera meta-dane dotyczące konkretnej bazy danych i zapisuję je w obiektach (diagram klas na Rys. 6.2.1). Dzięki temu programista może swobodnie korzystać z modelu i uzyskiwać potrzebne informacje do tworzenia zapytań.



Rys. 6.2.1 Schemat modelu meta-danych

Bazę danych reprezentuje obiekt Link, który jest identyfikowany za pomocą unikalnej pary obiektów `Schema` i `Catalog`. Dzieje się tak, dlatego, że niektóre bazy zawierają swoje bazy w jednostkach zwanych `Schema` a inne w `Catalog`. Różnica ta np. występuje w innych wersjach MySQL. Kiedyś baza przechowywana była w `Schema`, a teraz występuje w `Catolog`.

Model całej bazy tworzony jest w klasie o tej samej nazwie. Zwiera on wiele Linków w kolekcji, która powiązana jest z kolekcją tabel. Tabela może zawierać zero lub więcej wierszy, zero lub wiele kolumn. Jedna kolumna może zawierać wiele indeksów. Każda kolumna zawiera jeden typ.

Model ten jest źródłem meta-danych w programie. Pozwala na dynamiczne tworzenie zapytań.

6.2.1 Moduł dynamicznego mapowania typów bazy danych na typy języka oraz typów języka programowania na zapytania SQL.

Zgodnie z podrozdziałem producenci baz danych dostarczają wiele niestandardowych typów danych. Aby system był elastyczny i miał możliwości adaptacji do nowych typów danych lub zmian standardów sterowników JDBC, zapis mapowań typów danych na typy języka programowania powinien być zapisany w oddzielnym pliku konfiguracyjnym. Najbardziej intuicyjnym formatem i najlepiej obsługiwanym wydaje się zapis w formacie XML. Sterowniki JDCB dostarczają następujących metod do mapowania typów danych(tab. 6.2.1).

Tab. 6.2.1 Metody sterownika JDBC do mapowania typów danych[13]

getBlob(int columnIndex): Blob getBoolean(int columnIndex) : boolean getByte(int columnIndex) : byte getBytes(int columnIndex) : byte[] getCharacterStream(int columnIndex): Reader getClob(int columnIndex): Clob getDate(int columnIndex) : Date getDouble(int columnIndex): double getFloat(int columnIndex) : float getInt(int columnIndex) : int getLong(int columnIndex) : long getNCharacterStream(int columnIndex): Reader	getNClob(int columnIndex) : NClob getNString(int columnIndex) : String getObject(int columnIndex) : Object getRef(int columnIndex): Ref getShort(int columnIndex) : short getSQLXML(int columnIndex) : SQLXML getString(int columnIndex) : String getTime(int columnIndex) : Time getTimestamp(int columnIndex) : Timestamp getURL(String columnLabel) : URL

W tradycyjnym podejściu programistycznym zapytania do bazy danych mapowane są od razu na bazę danych.

W podejściu dynamicznym zastosowano następujące rozwiązanie. Za pomocą pliku XML opracowano formę mapowania typów bazy danych na klasy pośrednie (Listing 6.2.1). Klasy pośrednie muszą zawierać pusty konstruktor, aby mieć możliwość instancjonowania, czyli stworzenia obiektu za pomocą mechanizmu refleksji, w tym wypadku wywołania metody `Class.getInstance()`. Typy pośrednie są konieczne, ponieważ niektóre typy plików Java nie mają możliwości instancjonowania. Dzięki utworzonym klasą pośrednim z pomocą mechanizmu przeciążenia metod, wybierałby odpowiednią metodę do mapowania danych z bazy.

```
<?xml version="1.0" encoding="UTF-8"?>
<mapping>
<map><name>BIT</name><class>TypyDanych.BooleanType</class></map>
```

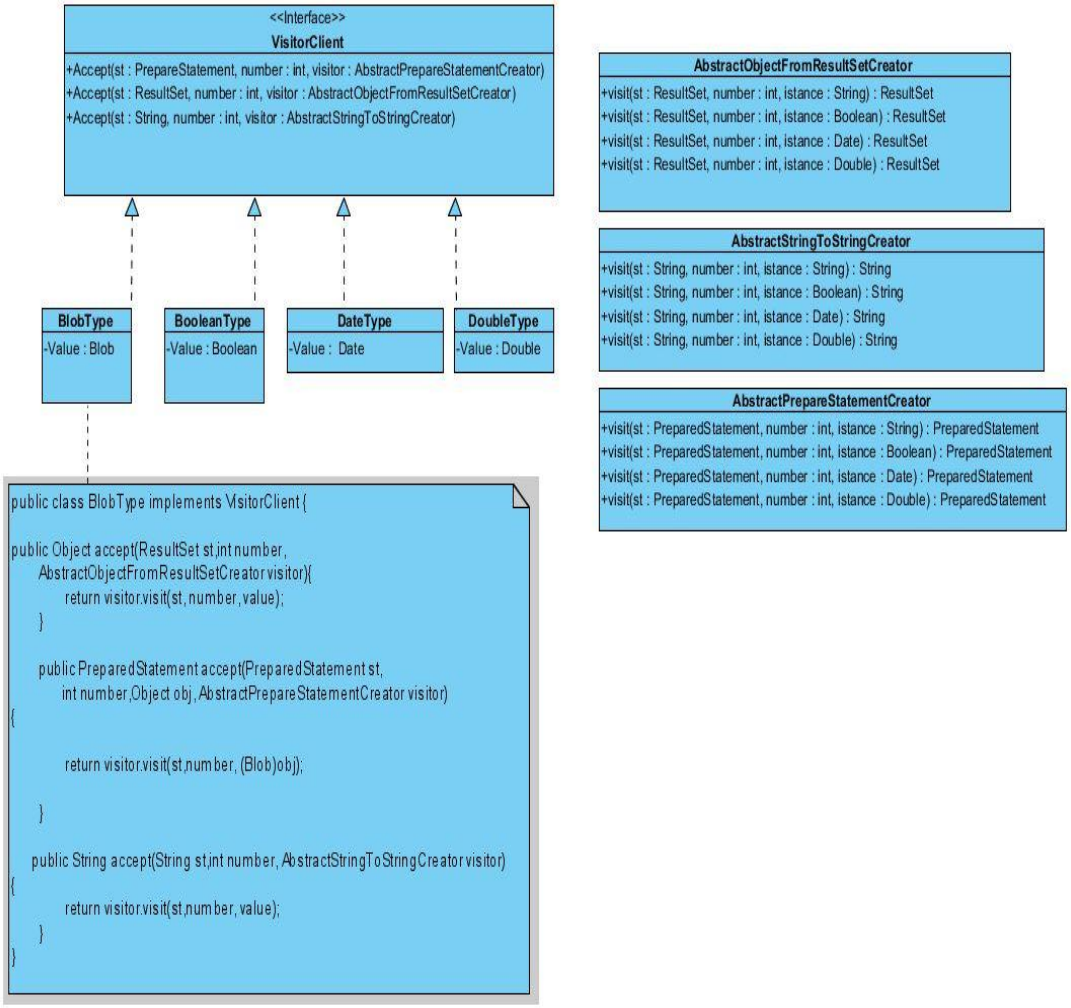
```
<map><name>BOOL</name><class>TypyDanych.BooleanType</class></map>
<map><name>TINYINT</name><class> TypyDanych.IntegerType</class></map>
<map><name>TINYINT UNSIGNED</name><class>
TypyDanych.IntegerType</class></map>
<map><name>BIGINT</name><class>TypyDanych.IntegerType </class></map>
<map><name>BIGINT UNSIGNED</name><class>TypyDanych.IntegerType
</class></map>
<map><name>LONG VARBINARY</name><class>TypyDanych.IntegerType</class></map>
<map><name>MEDIUMBLOB</name><class>TypyDanych.BlobType</class></map>
<map><name>LONGBLOB</name><class>TypyDanych.BlobType</class></map>
<map><name>BLOB</name><class> TypyDanych.BlobType</class></map>
<map><name>TINYBLOB</name><class>TypyDanych.BlobType</class></map>
<map><name>VARBINARY</name><class>TypyDanych.BlobType</class></map>
<map><name>BINARY</name><class>TypyDanych.BlobType</class></map>
<map><name>LONG VARCHAR</name><class>TypyDanych.StringType</class></map>
<map><name>MEDIUMTEXT</name><class>TypyDanych.StringType </class></map>
<map><name>LONGTEXT</name><class>TypyDanych.StringType</class></map>
<map><name>TEXT</name><class>TypyDanych.StringType</class></map>
<map><name>TINYTEXT</name><class>TypyDanych.StringType</class></map>
<map><name>CHAR</name><class>TypyDanych.StringType</class></map>
<map><name>NUMERIC</name><class>TypyDanych.IntegerType</class></map>
<map><name>DECIMAL</name><class> TypyDanych.BigDecimalType</class></map>
<map><name>INTEGER</name><class> TypyDanych.IntegerType</class></map>
<map><name>INTEGER UNSIGNED</name><class>TypyDanych.IntegerType
</class></map>
<map><name>INT</name><class> TypyDanych.IntegerType</class></map>
<map><name>INT UNSIGNED</name><class>TypyDanych.IntegerType </class></map>
<map><name>MEDIUMINT</name><class> TypyDanych.IntegerType</class></map>
<map><name>MEDIUMINT UNSIGNED</name><class>TypyDanych.IntegerType
</class></map>
<map><name>SMALLINT</name><class> TypyDanych.IntegerType</class></map>
<map><name>SMALLINT UNSIGNED</name><class>
TypyDanych.IntegerType</class></map>
<map><name>FLOAT</name><class> TypyDanych.FloatType</class></map>
<map><name>DOUBLE</name><class> TypyDanych.DoubleType</class></map>
<map><name>DOUBLE PRECISION</name><class>TypyDanych.DoubleType
</class></map>
<map><name>REAL</name><class> TypyDanych.DoubleType</class></map>
<map><name>VARCHAR</name><class>TypyDanych.StringType</class></map>
<map><name>ENUM</name><class>TypyDanych.StringType</class></map>
<map><name>SET</name><class>TypyDanych.StringType</class></map>
<map><name>DATE</name><class> TypyDanych.DateType</class></map>
<map><name>TIME</name><class> TypyDanych.TimeType</class></map>
<map><name>DATETIME</name><class> TypyDanych.TimestampType</class></map>
<map><name>TIMESTAMP</name><class>TypyDanych.TimestampType </class></map>
</mapping>
```

Listing 6.2.1 Przykładowe mapowania typów bazy danych na typy pośrednie.

Opisany schemat zaimplementowany został w formie wzorca projektowego Wizytator (diagram dla dołączony na rysunku 6.2.2) oraz zastosowania interfejsów programistycznych. Podstawowym interfejsem jest interfejs `VisitorClient`, który realizują klasy pośredniczące.

Każda taka klasa musi posiadać instancjonowany przykładowy obiekt języka Java (na rys. Value), na który będzie mapował typ bazy danych. Poza tym w systemie są trzy interfejsy, które reprezentują określone działania podczas mapowania plików z bazy do aplikacji i na odwrót.

Działania te są zróżnicowane w zależności od typu danych języka programowanie tzw. przeciążenie metod. W trakcje wykonywania programu instancjonowany jest wiec typ pośredni danych, następnie wykonywana jest jego metoda Accept z odpowiednim działaniem jako parametr. A wewnątrz tej metody, na dołączonej klasie działania wywoływana jest jej wewnętrzna metoda z odpowiednim obiektem docelowym, na który chcemy mapować nasze dane.



Rys. 6.2.2 Schemat wzorca projektowego Wizytator

6.2.2 Silnik dodatków

W aplikacji zaimplementowano system do obsługi pluginów. System przeszukuje wybrany katalog (w projekcie `PluginManager/PLUGINS`) szukając bibliotek z rozszerzeniem .jar, następnie przeszukuje klasy zawarte w plikach .jar w poszukiwaniu zaimplementowanego interfejsu `IPluginWigdet` (reprezentacja dodatku w projekcie), Jeśli odnajdzie odpowiednią klasę, rejestruje ją w systemie. System sprawdza dodatki przy każdym uruchomieniu. W systemie musi zawierać się zestaw standardowych dodatków, aby móc poprawnie działać. System nie wymaga rejestrowania dodatków w dodatkowych plikach np. XML.

6.2.3 Tworzenie nowych dodatków

Tworzenie nowych dodatków odbywa się za pomocą interfejsu `IPluginWidget` (Listing 6.2.1). Ewentualnie możemy dziedziczyć klasę abstrakcyjną `PluginAdapter` z projektu `PluginManager.Interface`.

Listing 6.2.1 Kod interfejsu programistycznego `IPluginWidget`

```
Public interface IPluginWidget {

     //Zwraca typ pośredni, któremu przypisany jest ten dodatek
      public Class getMidleType();

      //Zwraca typ Jezyka Java, na który będzie mapowana wartość.
      public Class getFinalType();

      //określa nazwę Dodatku.
      public String getName();

      //określa sposób wylidacji typu danych
      public boolean validateType(Object val);
      //inicjuje dodatek
      public void init(Table table,Column col);
      //pozwala na stworzenie nowego obiektu za pomoca metody.
      public IPluginWidget getNewInstance();
      //pozwala na zaiplementowanie odświerzenia dodatku w miarę potrzeby
      public void update();

      //obsługa komend wejścia-wyjścia dla dodatku
      public void setValue(Object val);
      public Object getValue();

      //obsługa źródła danych
      public void  setDataSource(IDataSource source);
      public IDataSource getDataSource();
      public void setTableRepresentation(ITableRepresentation
representation);
```

```
        //Edytory i Renderery do prezentacji danych
        public IWidgetEditor getEditor();
        public IWidgetRenderer getRenderer();

        //obsługa Walidatorów do spradzanie poprawności danych wejściowych w
dodatkach
        public void setValidator(IValidator validator);
        public  IValidator getValidator();


}
```

Pierwszą metodą do zrealizowania w interfejsie `IPluginWidget` jest przypisanie w metodzie `getMidleType()` odnośnika do klasy pośredniczącej w mapowaniu typów (klasy te znajdują się w katalogu). Przypisanie to spowoduje, że otrzymane dane będę pobierane i wysyłane z udziałem klasy pośredniczącej. Informacja o typie, na jaki będzie mapowany typ pośredniczący zapisana jest w metodzie `getfinalType()`. Następnie określamy nazwę dodatku (ze względu na przejrzystość nazwy nie mogą się powtarzać). Należy wybrać taką wartość, aby nazwa mogła rozróżnić konkretny dodatek na ekranie.

Walidacja typu `validateType()` to metoda odpowiadająca, za zweryfikowanie czy dane przyjęte do dodatku mogą być edytowane czy też nie.

Dalej w interfejsie należy wykonać konieczne zainicjowanie dodatku odpowiednią wartością klasy `Table` i `Column` z meta-modelu. Każdy dodatek (jego instancja) powinien wiedzieć, jaką tabelę i kolumnę obsługuję. Należy zapamiętać referencje do tych obiektów.

Konieczne jest też zaimplementowanie ciała metody `getNewInstance()` tworzącej nową instancje klasy (jest to wymagane przez silnik dodatków).

Dalej musimy zaimplementować obsługę wejścia i wyjścia dla dodatku zewnętrznego - metody zapisane w listingu 6.2.2.

Listing 6.2.2 metody obsługujące wejścia-wyjścia dodatku.

```
        public void setValue(Object val);
        public Object getValue();
```

Obsługa źródła danych i walidatorów zaimplementowana została w klasie abstrakcyjnej `PluginAdapter`, a ich działanie polegają na przechowaniu referencji do źródeł danych oraz do modelu tabeli `JTable`- reprezentującego dane oraz przechowanie referencji do walidatorów.

Najważniejsze metody do obsługi danych zawarte są w listingu 6.2.3

Listing 6.2.3 metody odpowiedzialne za edytowanie i wyświetlanie komórek w tabeli.

```
      //Edytory i Renderery do prezentacji danych
      public IWidgetEditor getEditor();
      public IWidgetRenderer getRenderer();
```

Metody mają na celu rozszerzenie tzw. edytorów i wykreślaczy klasycznej kontrolki Javy `JTable` o funkcjonalności potrzebnej do dobrej prezentacji i edycji danych. Metody te powinny tworzyć nowe obiekty edytorów, ponieważ dodatki mogą być wykorzystywane przez wiele kolumn. Każdy edytor powinien dziedziczyć po klasie abstrakcyjnej `WidgetEditor` i analogicznie każdy wykreślacz powinien dziedziczyć po klasie `WidgetRenderer`.

Dzięki zastosowanemu dziedziczeniu pozostało tylko obsłużenie metod wymaganych przez edytora (Listing 6.2.3)i wykreślacza (Listing 6.2.4)`JTabel`.

Listing 6.2.3 metoda zwracająca `Component` do wyświetlania komórek w tabeli.

```
    @Override
    public abstract Component getTableCellRendererComponent(JTable table,
Object value, boolean isSelected, boolean hasFocus, int row, int column);
```

Listing 6.2.4 metody wymagane przez edytora komórek w komponencie `JTable`

```
    @Override
    public Object getCellEditorValue();

    @Override
    public Component getTableCellEditorComponent(JTable table, Object
value,  boolean isSelected, int row, int column) ;
```

Jeśli programista będzie chciał użyć komponentu do wyskakujących okienek, znajduje się on w projekcie `PluginManager` i występuje pod nazwą `PopOutBean`. W pełni współpracuje on z komponentem `WidgetRenderer`. Aby użyć komponentu z wyskakującym okienkiem należy stworzyć przykładowy guzik z prezentacją wartości, którą chcemy edytować, następnie stworzyć nowy obiekt z parametrami konstruktora

Listing 6.2.5 Konstruktor klasy `PopOutBean`

```
PopOutBean(Container container, JButton button, int mode,WidgetEditor
editor)
```

Poszczególne atrybuty w konstruktorze `PopOutBean` (Listing 6.2.5) oznaczają:

- container - to nasz dodatek który obsłuży dane,

- button - to `JButton`, który przed chwilą stworzyliśmy i jest on komponentem, do którego przypisane jest wyskakujące okienko.
- mode - to wybór trybu działania odpowiadający stałym statycznym klasy `PopOutBean` (`ViemMode, EditMode`).
- editor – bieżący edytor (wpisujemy this)

W metodzie `getTableCellEditorComponent` zwracamy wartość metody `getbutton()` z bieżącego obiektu `PopOutBean`, aby poprawnie zwrócić komponent obsługujący edycję.

Stworzenie nowego sposobu prezentacji dodatku wymaga tylko podstawowej wiedzy o klasach do edycji i prezentacji danych w komponencie `JTable`, co ogranicza się do wypełnienia trzech wyżej wypisanych metod (Listing 6.2.3 i Listing 6.2.4).

7. Podsumowanie

Poniższy rozdział jest próbą podsumowania zaprezentowanego podejścia do prezentacji danych multimedialnych w bazach danych.

7.1. Wady i zalety rozwiązania

Zaprezentowane rozwiązanie cechują następujące wady i zalety:

Wady

- Rozwiązanie jest jeszcze na etapie prototypu i nie jest to finalny produkt nadający się do wdrożenia na rynek. Prototyp wymaga dużej ilości testów, mimo iż zaimplementował podstawową funkcjonalność potrzebną do prezentacji powyższego modelu.
- Nie zaimplementowano chwilowego zapamiętywania (na okres uruchomienia aplikacji) wyboru dodatków do prezentowanej tabeli.
- Nie zaimplementowano obsługi perspektyw, procedur oraz wyzwalaczy.

Zalety

- Rozwiązanie jest proste w obsłudze.
- Rozwiązanie znacznie skraca czas procesu edycji danych binarnych
- Aplikacja może być rozbudowywana za pomocą dodatków.
- Potencjał aplikacji jest bardzo duży.
- Rozwiązanie zapewnia łatwy i elastyczny dostęp do szerokiego wachlarza typów danych

7.2. Kierunki rozwoju aplikacji

Aplikacja do obsługi bazy implementuje w bieżącej wersji podstawową funkcjonalność do obsługi bazy danych modelu relacyjnego. Zawiera bardzo funkcjonalny i elastyczny edytor danych. Jednak złożoność baz danych i zagadnień z nią związanych uniemożliwiła zaimplementowania funkcjonalności takiej jak:

- Obsługa perspektyw
- Zarządzanie funkcjami i procedurami
- Dodawanie wyzwalaczy (trigers)
- Funkcjonalności do administracji serwerem bazy danych
- Obsługa użytkowników
- Przydzielanie przywilejów

Aplikacja może rozwijać się także w kierunku stworzenia wersji wielojęzycznej, nowych wersji językowych.

Przyszłość aplikacji leży jednak w projektowaniu nowych dodatków do obsługi różnych typów multimedialnych m.in. obsługę formatów tekstowych np. edycja html, obsługę typów geometrycznych oraz nowych złożonych typów danych np. (SET).

Ciekawym pomysłem na nowy dodatek byłaby obsługa połączeń między tabelami z poziomu edytora bazy danych, Użytkownik mógłby wybierać od razu spośród wartości kluczy głównych połączonej tabeli z ewentualnym podglądem.

Możliwości rozwoju aplikacji są bardzo interesujące, ponieważ każda rozbudowa o nowe typy danych znacznie podwyższa użytkowość aplikacji, oraz grono potencjalnych odbiorców.

7.3. Wnioski końcowe

Praca miała na celu przeprowadzenie analizy sposobu prezentacji danych w aplikacjach do obsługi bazy danych. Przeprowadziła przegląd powyższych aplikacji, wyszczególniła ich rozwiązania. Praca zaproponowała rozszerzenie modelu prezentacji o możliwość rozszerzania funkcjonalności do obsługi edycji poprzez dodatki do edytora zawartości baz danych. Po przeprowadzonej analizie praca wyszczególniła funkcjonalność potrzebną do stworzenia prototypu rozwiązania zawierającą wyżej opisany model.

Po zaimplementowaniu aplikacji można dostrzec duży potencjał przedstawionego rozwiązania. Rozszerzanie prezentacji i edycji danych w dynamicznych aplikacjach obsługujących dowolną strukturę bazy danych sprawia, że narzędzie znacznie zmniejsza czas

edycji danych w porównaniu do zaprezentowanych aplikacji. Dzięki powyższemu faktowi, znacznie wzrasta zadowolenia użytkownika i użyteczność aplikacji.

Elastyczny Interfejs do współpracy z bazą danych oparty na przedstawionym modelu jest zdecydowanie dobrym kierunkiem rozwoju w tej klasie aplikacji.

Prace cytowane

[1]. Jeremy Reimer, artykuł zatytułowany „A History of the GUI" adres strony www: http://arstechnica.com/old/content/2005/05/gui.ars

[2]. Mariusz Trzaska, Automatyczne generowanie graficznych interfejsów użytkownika, (KKIO08) http://www.users.pjwstk.edu.pl/~mtrzaska/Files/Papers/Trzaska_KKIO08.pdf

[3]. Witold Andrzejewski, Zbyszko Królikowski, Tadeusz Morzy „BAZY DANYCH I SYSTEMY INFORMATYCZNE ORAZ ICH WPŁYW NA ROZWÓJ INFORMATYKI W POLSCE" - Instytut Informatyki Politechnika Poznańska

[4]. Robert Budzyński Wykład akademicki: Wprowadzenie do technologii baz danych http://bobo.fuw.edu.pl/DB/OLD/wyklad6.html

[5]. J. Melton, A. Eisenberg, SQL:1999, formerly known as SQL3, SIGMOD Record 28(1), 1999. http://www.cl.cam.ac.uk/teaching/2003/Databases/sql1999.pdf

[6]. Andrew Eisenberg, Krishna Kulkarni, Jim Melton, Jan-Eike Michels, Fred Zemke "SQL:2003 Has Been Published" SIGMOD Record, Vol. 33, No. 1, March 2004 http://web.archive.org/web/20071011220454/www.sigmod.org/sigmod/record/issues/0403/E.JimAndrew-standard.pdf

[7]. Cay Horstmann, Gary Cornell "Java 2. Techniki zaawansowane." Wydanie 2005r.

[8]. Swing A Swing Architecture Overview http://java.sun.com/products/jfc/tsc/articles/architecture/

[9]. Tutorial JGroupLayout, z serwisu Oracle dotyczacego Javy http://download.oracle.com/javase/6/docs/api/javax/swing/GroupLayout.html

[10]. Dokumentacja PHPMyAdmin http://www.phpmyadmin.net/localized_docs/pl/Documentation.html

[11]. Dokumentacja MySQL WorkBench http://dev.mysql.com/doc/workbench/en

[12]. Dokumentacja MySQL - http://dev.mysql.com/doc/

[13]. JAVA 6SE API –Overview http://download.oracle.com/javase/6/docs/api/

[14]. Dokumentacja PostgreSQL http://www.postgresql.org/docs/7.4/interactive/datatype-geometric.html

[15]. Marek Wierzbicki, Java.Programowanie obiektowe wydawnictwo Helion 2006r.