Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki

Krzysztof Kostałkowicz
Nr albumu: 219459

Panel administracyjny WWW dla bazy danych LoXiM

Praca magisterska
na kierunku INFORMATYKA

Praca wykonana pod kierunkiem
dra hab. Krzysztofa Stencla

Maj 2009

Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data Podpis kierującego pracą

Oświadczenie autora (autorów) pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data Podpis autora (autorów) pracy

Streszczenie

W niniejszej pracy opisano implementację panelu administracyjnego WWW służącego do zarządzania bazą danych LoXiM, oraz projekt interfejsów programistycznych umożliwiających powstanie aplikacji.

LoXiM jest to system zarządzania bazą danych implementujący język zapytań SBQL stworzony przez Kazimierza Subietę. Projekt rozwijany jest na Wydziale Matematyki Informatyki i Mechaniki Uniwersytetu Warszawskiego pod nadzorem promotora pracy dra hab. Krzysztofa Stencla.

Panel administracyjny umożliwia wykonywanie typowych zadań administratora, programisty i użytkownika bazy danych w prosty i szybki sposób. Dzięki wizualizacji danych korzystanie z bazy danych jest znacznie łatwiejsze niż za pomocą interfejsu tekstowego.

Słowa kluczowe

bazy danych, LoXiM, SBA, SBQL, AOQL, PHP, liblx, lxpanel, panel administracyjny

Dziedzina pracy (kody wg programu Socrates-Erasmus)

11 Matematyka, Informatyka:
11.3 Informatyka

Klasyfikacja tematyczna

D. Software
D.2 Software Engineering

H. - Information Systems
H.2 - Database Management
H.2.7 - Database Administration

Tytuł pracy w języku angielskim

Web administration panel for LoXiM database

Spis treści

Wprowadzenie

W niniejszej pracy opisano implementację panelu administracyjnego WWW służącego do zarządzania bazą danych LoXiM, oraz projekt interfejsów programistycznych umożliwiających powstanie aplikacji.

Jako język użyty do implementacji panelu wybrano PHP, ze względu na jego prostotę, ogromne możliwości oraz bardzo dużą popularność. System zarządzania bazą danych LoXiM jest zaimplementowany w języku C++ i w chwili pisania pracy jedynym innym językiem programowania, za pomocą którego można było korzystać z bazy danych był język Java (opis interfejsu można znaleźć w [Ros07]), którego użycie było jednak niezgodne z podstawowymi założeniami projektu. Stąd należało zaprojektować interfejsy programistyczne, za pomocą których programy klienckie, będące skryptami PHP, skomunikują się z serwerem LoXiM.

Powstanie odpowiednich interfejsów umożliwiło zaimplementowanie panelu administracyjnego, będącego aplikacją w języku PHP korzystającą z bazy danych. Panel administracyjny powinien umożliwiać wykonywanie typowych zadań administratora, programisty i użytkownika bazy danych. Oto przykłady niektórych zastosowań:

- definiowanie zawartości bazy danych
- przeglądanie utrwalonych obiektów
- konstruowanie zapytań
- zarządzanie użytkownikami
- zarządzanie transakcjami
- oglądanie statystyk

Rozdział 1 zawiera wyjaśnienia podstawowych pojęć pojawiających się w dalszej części pracy. W rozdziale 2 przedstawiona jest architektura komponentów umożliwiających korzystanie z bazy danych PostgreSQL za pomocą języka PHP, oraz zostanie krótko omówiony jeden z paneli administracyjnych. Rozdział 3 prezentuje architekturę powstałego kodu, której kolejne warstwy są pokazane w rozdziałach 4 i 5, gdzie opisane są interfejsy programistyczne dostarczane przez biblioteki wraz z przykładami ich użycia. Rozdział 6 skupia się na właściwej aplikacji panelu administracyjnego i wyjaśnia, w jaki sposób zastosowane technologie pomagają w zaspokojeniu potrzeb użytkownika. Modyfikacje serwera bazy danych niezbędne do realizacji projektu opisano w rozdziale 7. Ostatni rozdział 8 zawiera przegląd wykonanych komponentów oraz wskazanie potencjalnych kierunków dalszego ich rozwoju.

Rozdział 1

Podstawowe pojęcia

SBA (Stack Based Approach) – podejście stosowe w przetwarzaniu danych. Charakteryzuje się wprowadzeniem pojęcia stosu do opisu semantyki języka zapytań, co ma umożliwić jego dokładniejszą specyfikację.

SBQL (Stack Based Query Language) – nowoczesny język zapytań dla danych półstrukturalnych oparty na koncepcji SBA. Jedną z jego zalet jest kompozycjonalność, która umożliwia łatwe rozszerzanie zapytań oraz łączenie mniejszych zapytań w większe.

AOQL (Advanced Object Query Language) – inna nazwa języka SBQL, wprowadzona przez komitet standaryzacyjny OMG. Szczegóły można znaleźć w dokumencie [OMG07].

LoXiM – jeden z systemów zarządzania bazą danych implementujących język SBQL.

API (Application Programming Interface) – specyfikacja funkcji, struktur danych, klas obiektów i/lub protokołów dostarczona w celu umożliwienia komunikacji z biblioteką, która tę specyfikację implementuje.

Panel administracyjny – miejsce w którym administrator usługi (w szczególności aplikacji internetowej) może dokonać zmian w jej konfiguracji. Często zakres możliwych operacji rozszerza się o działania typowe dla zwykłych użytkowników, przez co panel administracyjny staje się graficznym interfejsem użytkownika udostępniającym daną usługę.

Rozdział 2

Rozwiązanie w PostgreSQL

2.1. Wstęp

PostgreSQL to (obiektowo-)relacyjny system zarządzania bazą danych implementujący język zapytań SQL. Jak twierdzą autorzy, projekt ten jest najbardziej zaawansowaną bazą danych Open Source. Praktyka pokazuje, że aplikacje pisane z wykorzystaniem tej bazy danych cechują się dość dużą niezawodnością, a także sprawiają względnie mało problemów przy typowych zadaniach administracyjnych, jak np. zmiana wersji serwera bazy danych. Ponadto korzystanie z poszczególnych komponentów jest dość wygodne.

Między innymi z wyżej wymienionych względów projekt ten wybrano jako punkt odniesienia dla powstających interfejsów programistycznych oraz do projektu interfejsu użytkownika panelu administracyjnego. Konsekwencją tego było wzorowanie się na ogólnej idei opisanego tu rozwiązania, przystosowanego do danych semistrukturalnych, z zachowaniem jego dobrych cech i jednocześnie pominięciem złych, jeżeli było to możliwe.

2.2. Organizacja

Całość projektu przechowywana jest w repozytorium CVS. Oprócz serwera i standardowego klienta trybu tekstowego, znajduje się tam również zestaw bibliotek dostępowych dla różnych języków programowania oraz kolekcja prostych programów przydatnych administratorowi, jak np. program do tworzenia i usuwania bazy danych, czy użytkownika, a także do sporządzania kopii zapasowej bazy danych. Wszystkie standardowe komponenty napisano w języku C.

Wymienione wyżej programy korzystają ze wspólnej biblioteki dostępowej, nazwanej libpq, która na większości systemów skompilowana jest jako biblioteka dzielona. Biblioteka ta jest także tworzona przy budowaniu całości dystrybucji. Wszystkie moduły z których korzysta kompilowane są wówczas niezależnie od ich budowy dla potrzeb serwera.

2.3. Biblioteka dla języka C

Dzielona biblioteka libpq w wersji 5.1 zajmuje na dysku nieco ponad 100Kb, co wydaje się małym rozmiarem patrząc na oferowane funkcjonalności oraz liczbę możliwych sposobów uwierzytelnienia.

libpq udostępnia programiście języka C proste API umożliwiające komunikację z serwerem bazy danych. Oczywiście korzystają z niej wszystkie programy wchodzące w skład dystrybucji. Poniżej przedstawiono prototypy najważniejszych funkcji:

```
// utworzenie połączenia z bazą danych
PGconn *PQconnectdb(const char *conninfo);
PGconn *PQsetdbLogin(const char *pghost, const char *pgport,
        const char *pgoptions, const char *pgtty,
        const char *dbName,
        const char *login, const char *pwd);

// zamknięcie połączenia
void PQfinish(PGconn *conn);

// wykonanie synchronicznego zapytania
PGresult *PQexec(PGconn *conn, const char *query);
PGresult *PQexecParams(PGconn *conn,
        const char *command,
        int nParams,
        const Oid *paramTypes,
        const char *const * paramValues,
        const int *paramLengths,
        const int *paramFormats,
        int resultFormat);

// zwolnienie wyniku zapytania
void PQclear(PGresult *res);

// konwersja danych binarnych w celu wstawienia do zapytania
size_t PQescapeStringConn(PGconn *conn,
        char *to, const char *from, size_t length,
        int *error);

// pytanie o liczbę pól i wierszy
int PQntuples(const PGresult *res);
int PQnfields(const PGresult *res);

// pobranie typu pola
Oid PQftype(const PGresult *res, int field_num);

// sprawdzenie czy pole na wartość NULL
int PQgetisnull(const PGresult *res, int tup_num, int field_num);

// pobranie długości wartości
int PQgetlength(const PGresult *res, int tup_num, int field_num);

// pobranie wartości
char *PQgetvalue(const PGresult *res, int tup_num, int field_num);
```

Wnioski jakie można wyciągnąć z obserwacji tego interfejsu:

- Specyficzne dla języka C jest ręczne zwalnianie zasobów przez programistę, za co od-

powiadają funkcje PQfinish i PQclear.

- Funkcja do ustanawiania połączenia ma dwie wersje: PQsetdbLogin z informacjami przekazywanymi jako parametry funkcji, oraz PQconnectdb z parametrami zakodowanymi w napisie. Druga wersja jest bardziej rozszerzalna, ponieważ dodanie nowych parametrów nie pociąga za sobą zmiany interfejsu.
- Treść zapytania nie może zawierać danych binarnych. Postgres umożliwia przekazanie danych binarnych przez escape-owanie, czyli zmianę niedozwolonych znaków na sekwencję z \ i kodem znaku. Powinno się do tego używać funkcji PQescapeStringConn.
- Zapytania z numerowanymi kolejno parametrami są obsługiwane. Wartości podaje się jako napisy. Można podać typy parametrów, ale nie trzeba ponieważ mogą zostać wydedukowane ze schematu bazy. Opcjonalne podawanie długości zapewnia zgodność z danymi binarnymi.
- Funkcje dostępu do danych są specyficzne dla danych relacyjnych i nie mają żadnego zastosowania dla danych półstrukturalnych.
- Wartości pobiera się jako napisy. Stała NULL jest reprezentowana przez pusty napis, przez co musi istnieć funkcja PQgetisnull, aby było możliwe rozróżnienie tych dwóch sytuacji.
- Nie pokazano funkcje do zapytań asynchronicznych, które są również obsługiwane.

2.4. Dostęp z PHP

Moduł dostępowy dla języka PHP nie wchodzi w skład dystrybucji, ponieważ nie jest on rozwijany przez programistów PostgreSQL. Znajduje się natomiast w standardowej dystrybucji źródłowej PHP i co więcej jest standardowo instalowany na większości serwerów oferujących dostęp do tego języka skryptowego.

Podobnie jak większość modułów do PHP, interfejs do PostgreSQL jest opakowaniem biblioteki dostępowej w języku C. Oto jak wyglądają przedstawione wcześniej funkcje opakowane dla PHP:

```
// utworzenie połączenia z bazą danych
resource pg_connect(string $connection_string [, int $connect_type ] )

// zamknięcie połączenia
bool pg_close([ resource $connection ] )

// wykonanie synchronicznego zapytania
resource pg_query([ resource $connection ], string $query)
resource pg_query_params([ resource $connection ], string $query, array $params)

// konwersja danych binarnych w celu wstawienia do zapytania
string pg_escape_string([ resource $connection ], string $data)

// pytanie o liczbę pól i wierszy
int pg_num_fields(resource $result)
```

```
int pg_num_rows(resource $result)

// pobranie pojedynczej wartości
string pg_fetch_result(resource $result, int $row, mixed $field)
string pg_fetch_result(resource $result, mixed $field)
int pg_field_is_null(resource $result, mixed $field)

// pobranie całego wiersza do tablicy
array pg_fetch_row(resource $result [, int $row ] )
array pg_fetch_assoc(resource $result [, int $row ] )
array pg_fetch_array(resource $result [, int $row [, int $result_type ]] )

// pobranie całego wiersza do obiektu
object pg_fetch_object(resource $result [, int $row [, int $result_type ]] )
object pg_fetch_object(resource $result [, int $row [, string $class_name
        [, array $params ]]] )
```

Wnioski jakie można wyciągnąć z obserwacji tego interfejsu:

- Opcjonalne parametry ułatwiają programowanie. Część aplikacji WWW pracuje tylko z jednym połączeniem z bazą danych i wtedy nie trzeba go przekazywać jako argument do każdej z funkcji.
- Nie trzeba już ręcznie zwalniać zasobów, ponieważ interpreter PHP robi to automatycznie. Pomimo tego dostępna jest funkcja do zamknięcia połączenia.
- Do ustanowienia połączenia dostępna jest jedynie funkcja, do której argumenty przekazywane są zakodowane w napisie.
- W zapytaniach z parametrami wartości przekazywane są w tablicy, co upraszcza interfejs i znacznie ułatwia programowanie.
- Funkcje dostępu do danych operują na całych wierszach wyniku, oprócz pg_fetch_result dostarczonej prawdopodobnie do zachowania zgodności interfejsu z libpq. Pole można zidentyfikować numerem lub nazwą, stąd ostatni parametr ma typ mixed.
- Wszystkie typy odwzorowują się na napisy. NULL z bazy danych przekształca się na NULL w języku PHP. Zatem wywołanie pg_field_is_null jest nadmiarowe.
- Funkcja pg_fetch_row indeksuje wartości numerami kolejnymi, natomiast pg_fetch_assoc nazwami pól. Obydwie funkcje są nadmiarowe, gdyż takie same efekty można osiągnąć wołając pg_fetch_array z odpowiednimi flagami.
- Funkcja pg_fetch_object tworzy obiekt, którego atrybuty są ustawiane na wartości odpowiednich pól w pobieranym rekordzie wyniku. Może tworzyć instancję określonej klasy konstruując ją z podanymi w tablicy parametrami.
- Pobieranie danych jest nadal przystosowane do modelu relacyjnego, jednak w PHP nie jest to tak mocno zauważalne, dzięki tablicom i obiektom.
- Obsługiwane są również asynchroniczne połączenia.

- Można nawiązywać trwałe połączenia, czyli takie które nie są zamykane w momencie zakończenia skryptu i mogą być wykorzystane ponownie w późniejszych uruchomieniach. Mechanizm ten służy podniesieniu wydajności aplikacji.

2.5. Panel administracyjny

Najpopularniejszym panelem administracyjnym WWW do bazy PostgreSQL jest phpPgAdmin. Jest to niezależny projekt Open Source dostępny na licencji GNU GPL. Pierwsza jego wersja powstała jako port aplikacji phpMyAdmin, czyli narzędzia do zarządzania bazą MySQL. Obecna wersja jest napisana od nowa.

Zdaniem autorów panel posiada następujące cechy:

- obsługa serwerów w wersjach 7.x i 8.x
- zarządzanie użytkownikami, bazami danych, schematami, tabelami, indeksami, widokami, etc.
- łatwa manipulacja danych: wykonywanie zapytań, przeglądanie, dodawanie, edycja oraz usuwanie
- konwersja wyników do wielu formatów, jak np. SQL, COPY, XML, CSV, TSV
- import danych zapisanych w większości formatów, do których można je eksportować
- obsługa wielu języków (obecnie 27)
- łatwość instalacji i konfiguracji

Korzystając z wersji demonstracyjnej, na którą można się dostać ze strony domowej projektu, można się przekonać o słuszności tych stwierdzeń, a także znaleźć drobne usterki, jak np. niekompletne polskie tłumaczenie. Wsparcie zarządzania obiektami i manipulacji danymi polega na tym, że użytkownik nie musi ręcznie wpisywać zapytań, ale ma gotowe formularze, z których aplikacja po wypełnieniu i sprawdzeniu generuje zapytania.

phpPgAdmin posiada dobrze zaprojektowany i wygodny interfejs graficzny. Składa się z paska po lewej stronie i głównej treści. Pasek zawiera kontrolkę z widokiem drzewiastym, na którym można przeglądać lub wybrać odpowiednio: serwer, bazę danych, schemat, typ obiektu (np. tabela), konkretny obiekt, i ostatecznie jego własności (np. kolumna lub indeks). Oczywiście elementy można zwijać i rozwijać. Po pierwszym rozwinięciu elementu, jego zawartość jest pobierana bez przeładowywania całej strony. W głównej części strony jest pasek stanu, pasek lokalizacji (pokazuje ścieżkę do wybranego węzła w drzewie), pasek nawigacji (pokazuje dzieci wybranego węzła w drzewie) oraz zasadnicza treść strony. Pasek stanu oprócz informacji o połączeniu zawiera przyciski pozwalające na otworzenie okien zapytania, historii i wyszukiwania oraz wylogowanie się. Dzięki takiemu układowi programu, dostosowanemu do struktury relacyjnej bazy danych, często wykorzystywane funkcję są łatwo dostępne, a nawigacja intuicyjna, co pozwala na szybkie i sprawne wykonywanie typowych czynności.

2.6. Wnioski

Dzięki obserwacji przedstawionego w tym rozdziale rozwiązania można wyciągnąć wnioski przydatne w projektowaniu interfejsów programistycznych, które wzięto pod uwagę przy budowie panelu administracyjnego dla LoXiM i odpowiednich bibliotek dostępowych. Oto one:

- W języku C należy dostarczyć funkcje do zwalniania pamięci. W PHP nie powinno się, jeśli będzie to jedyny efekt funkcji, gdyż interpreter zwolni pamięć automatycznie.
- Dobrym pomysłem jest przekazywanie parametrów połączenia zakodowanych jako napis, ponieważ będzie można je wówczas modyfikować bez zmian w API.
- Dane binarne nie muszą być obsługiwane w treści zapytania, natomiast powinny w wartościach parametrów oraz w wyniku.
- Dla zapytań z parametrami istotne jest jakiego rodzaju metadane są dostarczone.
- Funkcje pobierające dane powinny operować na większych częściach danych. Odpowiednia reprezentacja danych powinna zapewnić kontrolę typów.
- Należy używać domyślnych wartości parametrów w PHP wszędzie gdzie to ma sens.

Podobnie dzięki obserwacji wyglądu i sposobu działania panelu administracyjnego nasuwają się następujące wnioski:

- Panel powinien być zwykłą aplikacją w PHP, czyli korzystać jedynie z oficjalnie dostępnego API.
- Oferowane funkcjonalności powinny być takie same dla różnych rodzajów użytkowników. Dla uproszczenia można pokazywać różne zestawy operacji w zależności od uprawnień korzystającego.
- Należy zapewnić wsparcie dla podstawowych czynności: przeglądania, dodawania, edycji oraz usuwania danych.
- Użyteczną cechą jest możliwość eksportu i importu danych.
- Aplikacja powinna się łatwo instalować oraz mieć minimalną konfigurację.
- Widok powinien być zaprojektowany w zorganizowany sposób, czyli z podziałem na części, przy zachowaniu spójności w różnych miejscach aplikacji.
- Często wykonywane operacje powinny być łatwo dostępne.
- Możliwie dużo akcji należy wykonywać bez przeładowywania strony.

Zastosowanie się do powyższych spostrzeżeń powinno umożliwić zaprojektowanie dobrego interfejsu programistycznego, czyli o dużej funkcjonalności połączonej z małym rozmiarem, a także wymagającego w przyszłości niewielkich zmian. Podobne stwierdzenie zachodzi też dla panelu administracyjnego, gdzie wymaganiami są głównie: łatwość obsługi, komfort pracy, estetyczny wygląd oraz dostępność mechanizmów wspomagających pracę.

Rozdział 3

Architektura rozwiązania

3.1. Wstęp

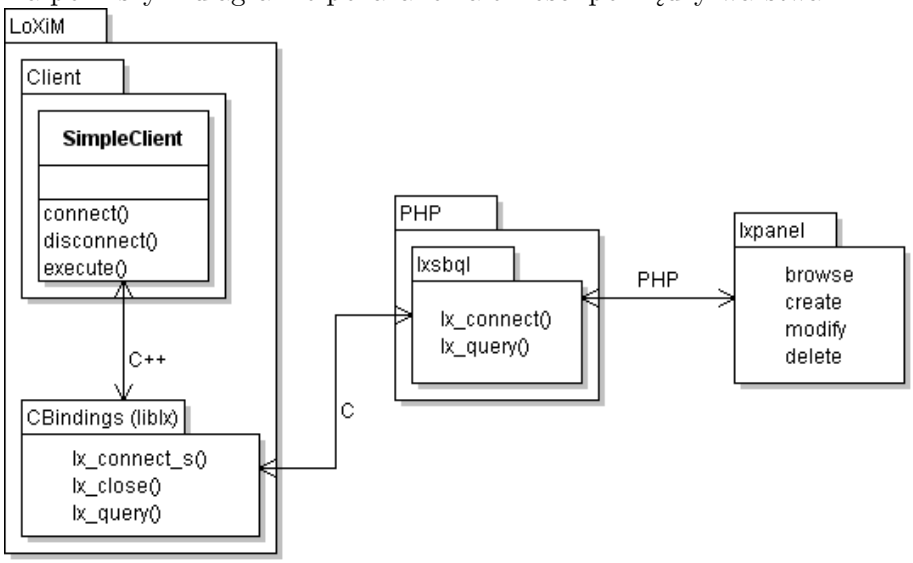
W niniejszym rozdziale znajduje się opis projektu architektury rozwiązania, wraz z ogólnym omówieniem poszczególnych komponentów. Dla czytelności dołączono również diagram klas. Przedstawiono także zalety zastosowanego rozwiązania.

3.2. Projekt

System zaprojektowano w sposób wielowarstwowy. Poszczególne warstwy dobrano w ten sposób, aby każda odpowiadała za możliwie jak najprostszą funkcjonalność. Wyróżnia się następujące warstwy:

- biblioteka dostępowa dla języka C – liblx
- moduł dostępowy dla języka PHP – lxsbql
- aplikacja panelu administracyjnego – lxpanel

Na poniższym diagramie pokazano zależności pomiędzy warstwami:



Zewnętrzne pakiety reprezentują aplikacje, a wewnętrzne moduły. W module Client utworzono nową klasę, której przykładowe metody są wymienione. Pozostałe widoczne moduły

również zawierają przykładowe metody. Aplikacja panelu nie ma podziału na odrębne moduły, a jako jej treść wymieniono przykładowe przypadki użycia.

Zadania poszczególnych warstw są następujące:

- SimpleClient umożliwia wykonanie złożonych akcji (z punktu widzenia programowania bazy danych) w pojedynczych wywołaniach. Akcje te są wspólne dla interfejsu programistycznego i klienta tekstowego, dlatego należało stworzyć klasę która je udostępnia. Dodatkowe objaśnienia znajdują się w rozdziale 8.

- liblx jest interfejsem programistycznym dla języka C. Udostępnia zatem programiście funkcje i typy danych zgodne z tym językiem. Oprócz koniecznej konwersji, zapewnia niezależność od wewnętrznej implementacji bazy danych.

- lxsbql jest rozszerzeniem języka PHP. Odpowiada za odwzorowanie funkcji i typów danych języka C na zgodne z PHP. Pociąga to za sobą drobne zmiany w interfejsie oraz bardziej znaczące w modelu danych.

- lxpanel to ostateczna aplikacja WWW, napisana głównie w języku PHP i wykorzystująca jego rozszerzenie umożliwiające komunikację z bazą danych. Jest odpowiedzialna za możliwość zarządzania bazą danych z poziomu przeglądarki internetowej.

3.3. Zalety

Wybrane rozwiązanie wielowarstwowe ma szereg zalet, których nie posiadałoby rozwiązanie z mniejszą liczbą warstw, wymienionych poniżej:

- Każda z warstw ma ograniczony zakres kompetencji, który jest dostatecznie prosty, aby logika warstw nie musiała być zbyt skomplikowana.

- Zmiana (poprawna) implementacji warstwy nie zakłóca działania całości.

- Każda z warstw zależy tylko od sąsiedniej warstwy niższego rzędu, dzięki czemu zmiana organizacji dalszych warstw niższych nie zakłóca wyższych. W szczególności zmiany w implementacji serwera nie są nigdzie zauważalne. Również zmiany interfejsu jednej warstwy wymagają jedynie przystosowania do nich sąsiedniej warstwy wyższej.

- Warstwy mogą zostać użyte w innych projektach. Alternatywna implementacja jednej z warstw zapewni działanie komponentów korzystających z niej, w tym także wyższych warstw.

- Biblioteka udostępniająca API w języku C ze względu na jego powszechną dostępność może wspierać inne zastosowania, takie jak samodzielne aplikacje lub interfejsy programistyczne dla innych języków programowania.

- Dzięki wspomaganiu pisania aplikacji WWW w języku PHP, nie ma potrzeby dbania o niskopoziomową komunikację z przeglądarką użytkownika, gdyż odpowiedzialność ta spoczywa na serwerze HTTP. Język ten wspiera również wywoływanie skryptów z wiersza poleceń oraz tworzenie interfejsów graficznych.

- Ponieważ aplikacja panelu administracyjnego została wyodrębniona, zastosowanie jej dla innego obiektowego systemu zarządzania bazą danych nie powinno wymagać wielkiego wysiłku (jeśli istnieje API dla PHP).

Jako skrajny przypadek alternatywny dla zaprezentowanego rozwiązania można wyobrazić sobie panel administracyjny zaimplementowany bezpośrednio w języku C++, który byłby zbiorem modułów w projekcie LoXiM. Wówczas żadna z wyżej wymienionych własności nie jest spełniona. Oczywiście istnieją też rozwiązania pośrednie (z mniejszą liczbą warstw), jednak nie zachowują one wszystkich z powyższych zalet.

Z drugiej strony można się zastanowić nad wprowadzeniem kolejnych warstw. Problem tkwi w tym, że nie jest jasne gdzie takie warstwy można by dodać i za co miałyby one odpowiadać. Obecny podział jest wystarczający, by poszczególne warstwy były nieskomplikowane, a jego rozszerzenie spowodowałoby, że nowe warstwy stałyby się trywialne. Wyjątkiem może być jednak sama aplikacja lxpanel, którą napisano bezpośrednio w języku PHP, bez wykorzystania żadnego frameworka. Oparcie jej o framework mogło by poprawić jej strukturę lub czytelność kodu, aczkolwiek nie jest ono niezbędne do osiągnięcia celów, jakie są tej aplikacji stawiane.

Rozdział 4

Biblioteka C – liblx

4.1. Wstęp

Bibliotekę liblx zaprojektowano aby umożliwić dostęp do bazy danych programom pisanym w języku C. Udostępnia ona w tym celu funkcje oraz typy danych służące komunikacji z serwerem bazy danych. Ponadto dzięki niej programy klienckie nie muszą korzystać bezpośrednio z kodu serwera, co zapewnia niezależność od aktualnej jego implementacji.

4.2. Organizacja

Biblioteka jest jednym z modułów wewnątrz projektu LoXiM. Aby odpowiadała ona wyłącznie za tłumaczenie kodu i danych z C na C++, należało utworzyć klasę zajmującą się komunikacją z serwerem. Klasę nazwano SimpleClient, a dodano ją do modułu Client, gdyż powinien z niej również korzystać klient tekstowy. Oto jej interfejs:

```
// sprawdzenie wersji
static const std::string version;

// konstrukcja
SimpleClient(const std::string &params);
SimpleClient(const std::string &host, int port,
        const std::string &login, const std::string &password);

// ustanawianie i zamykanie połączenia
void connect();
void disconnect();

// wykonywanie zapytań
void execute(const std::string &query,
        const Protocol::Package *&result);
void executeParams(const std::string &query,
        const Protocol::Package *&result,
        int nparams,
        const char* const *keys,
        const char* const *values,
```

```
        const int *lengths);

// sprawdzanie stanu połączenia
int statusConnected();
int statusTime();
std::string statusHostname();
std::string statusHostaddr();
int statusPort();
std::string statusUser();
```

Dla uproszczenia założono, że jedna instancja reprezentuje połączenie z określonym serwerem i nie można tego zmienić. Stąd parametry połączenia dostarcza się w konstruktorze. Wprowadzono jednak rozróżnienie stanu połączonego i rozłączonego, co można zbadać metodą statusConnected, oraz metody connect i disconnect do zmiany tego stanu.

Jedyną funkcjonalnością udostępnianą przez tę klasę jest wykonanie zapytania. Służy do tego blokująca metoda execute. Metoda executeParams nie jest zaimplementowana w bieżącej wersji, ponieważ zapytania z parametrami nie były zrealizowane po stronie serwera w chwili pisania kodu.

W celach informacyjnych wprowadzono również rodzinę funkcji do sprawdzania stanu obecnego połączenia. Zakres informacji, które można sprawdzić obejmuje: czas połączenia, nazwę, adres i port serwera oraz nazwę użytkownika.

Przedstawiony powyżej interfejs wystarczył do realizacji biblioteki liblx.

4.3. API

Udostępniono następujące typy danych:

- LXCONN – reprezentuje połączenie z bazą danych
- LXRES – informacja o stanie zapytania wraz z wynikiem
- LXDATA – dane
- LXERR – opis błędu
- LXREF – typ danych: referencja do obiektu

Ze względu na brak struktury dane mają charakter rekurencyjny. Wyróżnia się typy proste (liście w drzewie) i złożone (węzły). Typy danych rozróżnia się dzięki predefiniowanym stałym. Istnieją również makra stwierdzające, czy dany typ jest prosty czy złożony. Omówienie typów danych nastąpi poprzez prezentację funkcji dostępowych:

```
// zwraca typ - jedną z predefiniowanych stałych
int lx_get_type(LXDATA *d);

// pobieranie wartości typu prostego
char* lx_get_varchar(LXDATA *d);
int lx_get_varchar_len(LXDATA *d);
int lx_get_int(LXDATA *d);
```

```
int lx_get_bool(LXDATA *d);
double lx_get_double(LXDATA *d);
LXREF lx_get_ref(LXDATA *d);

// uniwersalne pobieranie wartości typu złożonego
LXDATA* lx_get_children(LXDATA *d);
int lx_get_length(LXDATA *d);

// konkretne pobieranie wartości typu złożonego
LXDATA* lx_get_bag(LXDATA *d);
LXDATA* lx_get_struct(LXDATA *d);
LXDATA* lx_get_seq(LXDATA *d);
char* lx_get_bind_key(LXDATA *d);
int lx_get_bind_key_len(LXDATA *d);
LXDATA* lx_get_bind_val(LXDATA *d);
```

Funkcje udostępniane przez bibliotekę mają następujące sygnatury:

```
// ustanawianie połączenia
LXCONN lx_connect_p(const char *host, int port,
        const char *user, const char *password);
LXCONN lx_connect_s(const char *params);

// wykonanie zapytania
LXRES lx_query(LXCONN c, const char *query);
LXRES lx_query_params(LXCONN c, const char *query,
        length_t nparams, const char * const *keys,
        const char * const *values, const length_t *lengths);

// pobranie wyników zapytania
LXDATA* lx_fetch(LXRES r);

// sprawdzanie błędów
LXERR lx_error_conn(LXCONN c);
LXERR lx_error_res(LXRES r);
#define lx_error_code(e) ((e)->code)
#define lx_error_msg(e) ((e)->msg)

// sprawdzanie stanu połączenia
int lx_status(LXCONN c, const char *code, char *buf, int maxlen);
int lx_status_int(LXCONN c, const char *code, int *res);
int lx_status_codes(char *buf, int maxlen);

// zwalnianie zasobów
void lx_close(LXCONN c);
void lx_dispose(LXRES r);
```

Zmiany w interfejsie wymuszone są przejściem z obiektowego stylu programowania na styl strukturalny, więc wprowadzono typy danych reprezentujące obiekty, które trzeba przekazywać przy wywoływaniu metod.

Wyjątkowo przeprojektowano funkcje do sprawdzania stanu połączenia. Nowy interfejs oparto na założeniu, że każdy element stanu reprezentowany jest przez napis. Obecnie rozpoznawane napisy, to: „hostname", „hostaddr", „user", „connected", „time" i „port". Odpowiadają one wywołaniom metod klasy SimpleClient. Ponieważ mogły one zwracać napisy lub liczby, pojawiło się rozróżnienie na lx_status i lx_status_int, przy czym lx_status przy pytaniu o liczbę zwróci wartość liczbową jako napis. Funkcja lx_status_codes pozwala uzyskać listę wszystkich rozumianych napisów.

4.4. Cechy

Przedstawiony interfejs charakteryzuje się następującym własnościami:

- Parametry połączenia zakodowane w napisie. Jest to bardziej rozszerzalny sposób niż przekazywanie ich jako argumentów funkcji. Aby umożliwić szybkie testowanie dostarczono również drugą wersję.
- Informacje o stanie połączenia jako napisy. Analogicznie, w porównaniu do informacji jako osobnych funkcji, metoda ta umożliwia łatwe rozszerzanie dostępnego zestawu informacji.
- Wsparcie dla nazywanych parametrów zapytań. Nie są jednak obsługiwane przez aktualną implementację niższej warstwy.
- Bezpieczeństwo dla danych binarnych. Osiągnięto je dzięki przekazywaniu napisów wraz z informacją o długości. Nie dotyczy to treści zapytań, w których nie mogą pojawić się dane binarne, ponieważ nie obsługuje tego serwer.
- Obsługa wielowątkowości. Zapewnia ją brak zmiennych globalnych.

Zaobserwowany wzrost rozmiaru API w porównaniu do klasy SimpleClient spowodowany jest inną specyfiką stylu programowania. Semantycznie obydwa interfejsy są ze sobą zgodne. Dotyczy to również typów danych.

4.5. Przykład wykorzystania

Poniżej znajduje się fragment programu, który nawiązuje połączenie z bazą danych, wykonuje proste zapytania i wypisuje ich wyniki.

```
LXCONN c = lx_connect_p("localhost", 2000, "root", "");
if (lx_error_code(lx_error_conn(c)) != 0)
        exit(0);
execute(c, "begin");
execute(c, "2;");
execute(c, "3.14;");
execute(c, "4 < 87;");
execute(c, "7 union 6");
execute(c, "(4 as x, 5 as y)");
execute(c, "end");
lx_close(c);
```

Definicja funkcji execute jest następująca:

```
void execute(LXCONN c, const char *q)
{
        LXDATA *d;
        LXRES r;
        r = lx_query(c, q);
        if (lx_error_code(lx_error_res(r)) != 0)
                return;
        d = lx_fetch(r);
        if (d != NULL)
                print_data(d, 4);
        lx_dispose(r);
}
```

Przykładowe wypisywanie zaimplementowano tak:

```
void print_data(LXDATA *d, int lev)
{
        int i;
        LXDATA *ch;
        for (i = 0; i < lev; i++) putchar(' ');
        switch (lx_get_type(d)) {
        case LX_DATATYPE_INT:
                printf("<int %d>\n", lx_get_int(d));
                break;
        case LX_DATATYPE_DOUBLE:
                printf("<double %g>\n", lx_get_double(d));
                break;
        /* ... */
        case LX_DATATYPE_BAG:
                printf("<bag> {\n");
                ch = lx_get_bag(d);
                for (i = 0; i < lx_get_length(d); i++)
                        print_data(&ch[i], lev+1);
                for (i = 0; i < lev; i++) putchar(' ');
                printf("}\n");
                break;
        case LX_DATATYPE_BIND:
                printf("<bind> %s:\n", lx_get_bind_key(d));
                print_data(lx_get_bind_val(d), lev+1);
                break;
        default:
                printf("<unknown type>\n");
        }
}
```

Rozdział 5

Moduł PHP – lxsbql

5.1. Wstęp

PHP jest w chwili pisania tego tekstu najbardziej popularnym, powszechnie dostępnym językiem skryptowym wspierającym programowanie aplikacji WWW. Projekt powstał w 1995 roku i już dwa lata później miał bardzo szeroką społeczność, co sprzyjało jego szybkiemu rozwojowi. Obiektowy styl programowania jest wspierany od wersji 5, ale nie jest on wykorzystywany przez to rozszerzenie, które napisano dla właśnie tej wersji.

Rdzeniem PHP jest interpreter zwany ZEND Engine. Obsługuje on podstawowe konstrukcje językowe (w tym tablice i słowniki), ale nie dostarcza żadnych funkcji. Najczęściej używane są zaimplementowane w bibliotece standardowej. Dużą elastyczność uzyskano poprzez mechanizm rozszerzeń, które mogą być dynamicznie doładowywane. Rozszerzenie może definiować abstrakcyjne typy danych, funkcje lub klasy, które implementuje się w języku C lub C++. Zazwyczaj (szczególnie w przypadku dostępu do bazy danych) korzysta ono z zewnętrznej biblioteki dostępowej dla języka C. Rolą takiego rozszerzenia jest więc głównie odwzorowanie niskopoziomowych struktur danych języka C na obiekty bardziej złożone, wygodniejsze w użyciu przez PHP. Podobne założenie przyjęto przy projektowaniu modułu dostępowego dla bazy danych LoXiM.

5.2. API

Abstrakcyjne typy danych w języku PHP określa się jako zasoby. Dostęp do nich z poziomu języka ogranicza się do ich kopiowania (jako referencji), w tym przekazywania jako argument funkcji. Warto także zwrócić uwagę, że programista nie musi dbać o zwalnianie zasobów, ponieważ są one wyposażone w liczniki referencji, co umożliwia ich usuwanie gdy przestają być potrzebne, czemu towarzyszy wywołanie destruktora zdefiniowanego przez dany moduł.

W opisywanym rozszerzeniu wyróżnia się następujące typy danych:

- Połączenie – reprezentuje sesję z bazą danych
- Wynik – dane zwrócone przez zapytanie
- Referencja – typ danych będący odwołaniem do innego obiektu z bazy danych

Za tworzenie oraz manipulację zasobami odpowiadają funkcje danego rozszerzenia. Udostępnia się następujące wywołania:

```
// nawiązanie połączenia
resource lx_connect(string params)
resource lx_connect_params(string host, int port, string user, string password)

// wykonanie zapytania
resource lx_query(string q [, resource conn])
resource lx_query_params(string q [, array params [, resource conn]])

// pobranie wyników
mixed lx_fetch(resource res [, int howto])

// sprawdzenie błędów
string lx_error_conn([resource conn])
string lx_error(resource res)

// zwalnianie zasobów
void lx_close([resource conn])
void lx_dispose(resource res)

// sprawdzenie stanu połączenia
array lx_status([resource conn])

// operacje na referencjach
mixed lx_deref(resource res [, int howto [, resource conn]])
string lx_refid(resource res)
resource lx_mkref(mixed id)
```

Rozszerzenie udostępnia także wpisy w widoku informacyjnym umożliwiające sprawdzenie wersji oprogramowania. Aktualnie nie są obsługiwane żadne wpisy w pliku konfiguracyjnym. Dostępne są następujące stałe:

- LXSBQL_VERSION – informacja o wersji
- LXSBQL_FETCH_DATA – pobieranie właściwych danych wyniku
- LXSBQL_FETCH_STRUCT – pobieranie informacji o strukturze wyniku
- LXSBQL_FETCH_ALL – pobieranie danych i wszystkich metadanych

Stałe LXSBQL_FETCH podaje się jako argument howto dla funkcji lx_fetch oraz lx_deref i określają jakiego typu dane należy zwrócić. Domyślną wartością jest LXSBQL_FETCH_DATA.

5.3. Odwzorowanie danych

W poniższej tabeli znajduje się opis zastosowanego odwzorowania danych:

liblx / C	lxsbql / PHP
void	NULL
int	int (wewnętrznie long)
bool	bool
double	double
varchar	string
reference	resource (referencja)
bag	array
struct	array
sequence	array
collection(bind(k, v))	array[k] = v
collection(v)	array[] = v

Tablice w PHP można indeksować napisami i liczbami, co zostało wykorzystane w dwóch ostatnich wierszach tabeli z konwersjami. NULL jest, podobnie jak w większości języków programowania, stałą mogącą być dowolnego typu.

Odwzorowanie to dobrze sprawdza się dla typów prostych, ale pewne problemy pojawiają się przy kolekcjach. Rozróżnienie rodzaju kolekcji osiągnięto poprzez dodanie do tablicy z jej reprezentacją metainformacji „_type", której wartość wskazuje na typ kolekcji. Informacja taka dodawana jest, jeśli poda się flagę LXSBQL_FETCH_STRUCT.

Istnieją także dane, które nie mają dobrego odwzorowania, takie jak:

- bind(k, v) → v – jeśli pojawia się bezpośrednio w korzeniu. W bieżącej implementacji bazy danych sytuacja taka nie zachodzi.
- Jeśli w kolekcji występują dwa elementy bind o tych samych kluczach, późniejszy nadpisze wcześniejszy, ponieważ tablice w PHP to słowniki, co uniemożliwia stosowanie tych samych kluczy do różnych wartości. Możliwe obejścia tego problemu zaprezentowano w rozdziale 8.
- bind(k1, bind(k2, v)) → bind(k1, v) – podczas odwołań do utrwalonych obiektów taka sytuacja nie zdarza się, chociaż istnieją zapytania zwracające taki wynik. Rozwiązania tego problemu są analogiczne jak w poprzednim przypadku (opisane w rozdziale 8).

5.4. Cechy

Przedstawiony interfejs charakteryzuje się następującymi własnościami:

- Parametry połączenia przekazywane w napisie – dzięki temu zmiana ich zestawu nie wymaga zmian w interfejsie.
- Wartości parametrów zapytania przekazywane w tablicy – co upraszcza ich przekazanie, gdyż wystarcza do tego jedna kolekcja.
- Pobieranie całego wyniku jedną funkcją – dzięki czemu wynik może być przeglądany niezależnymi mechanizmami, a także unika się narzutu na kolejne wywołania. Z tego powodu typ zwracanej wartości oznaczony jest jako mixed.
- Bezpieczeństwo dla danych binarnych – jest zapewnione dla wyników i wartości parametrów, ale nie dla treści zapytań.
- Obsługa wielowątkowości – ponieważ nie występują zmienne globalne.

- Pamiętanie stanu (połączenie domyślne) – co ułatwia programowanie, a zrealizowane jest w sposób bezpieczny dla wielowątkowości.

Skrócenie się liczby argumentów niektórych funkcji spowodowane jest użyciem wysokopoziomowych typów danych, które niosą wiele informacji w jednej wartości.

5.5. Przykład wykorzystania

Następujący fragment programu łączy się z bazą danych, wykonuje kilka zapytań, po czym zamyka połączenie (czego nie trzeba robić, ponieważ zostanie to zrobione automatycznie po zakończeniu wykonywania skryptu).

```
$c = lx_connect_params("localhost", 2000, "root", "");
if (!$c) {
        echo lx_error_conn() . "\n";
        exit;
}
query($c, "begin");
query($c, "2;");
query($c, "3.14;");
query($c, "4 < 87;");
query($c, "7 union 6");
query($c, "(4 as x, 5 as y)");
query($c, "end");
lx_close($c);
```

Funkcja query może być zaimplementowana następująco:

```
function query($q)
{
        $r = lx_query($q);
        if (lx_error($r) != "") {
                echo "Error: " . lx_error($r) . "\n";
                return;
        }
        $f = lx_fetch($r);
        var_dump($f);
        lx_dispose($r);
}
```

Także w tym przypadku nie trzeba pamiętać o zwolnieniu wyniku zapytania, ponieważ zostanie to zrobione przez interpreter po wyjściu z funkcji. Na uwagę zasługuje fakt, że wynik zapytania można wypisać korzystając z wbudowanej funkcji var_dump.

Rozdział 6

Aplikacja lxpanel

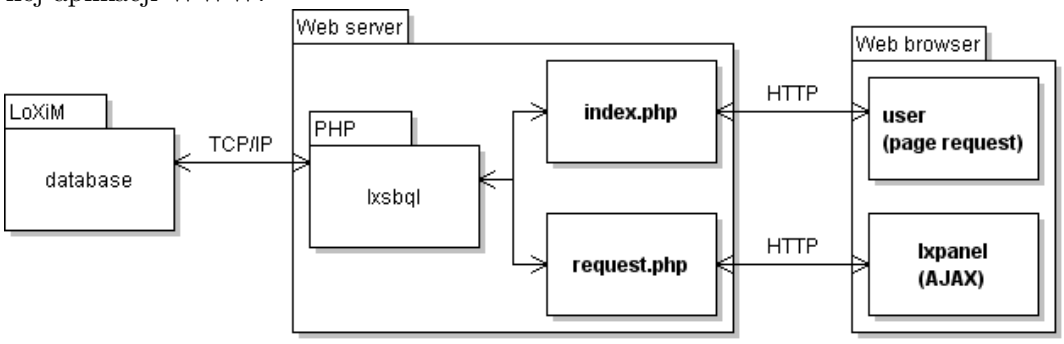
6.1. Wstęp

Panel administracyjny jest zwykłą aplikacją w języku PHP, z zastrzeżeniem, że używa rozszerzenia umożliwiającego komunikację z bazą LoXiM. Instalacja polega na przekopiowaniu plików aplikacji katalogu udostępnianego przez serwer WWW. Obecna wersja działa jako skrypt CGI, więc należy wpisać poprawną ścieżkę do programu php-cgi w pierwszych liniach plików w katalogu głównym. Konfiguracja polega na edycji pliku config.cgi.

Wykorzystane technologie obejmują:

- HTTP 1.1 – protokół komunikacji przeglądarki z serwerem WWW
- XHTML 1.0 – jako język opisu stron
- CSS 2.0 – zapewnia kontrolę nad wyglądem stron
- mechanizm sesji w PHP – aby zapamiętywać stany poszczególnych połączeń
- JavaScript – umożliwiający dynamiczne zmiany w wyglądzie strony
- AJAX – dający możliwość wysłania asynchronicznego zapytania

Na poniższym diagramie pokazano schemat komunikacji, który jest typowy dla interaktywnej aplikacji WWW:



Zwykłe żądanie pobrania pliku obsługiwane jest przez skrypt index.php, a zlecenie asynchroniczne – przez request.php.

6.2. Przypadki użycia

Uwierzytelniony użytkownik aplikacji może wykonać następujące akcje:

- Przeglądanie zawartości bazy danych
- Zmienianie obiektów: tworzenie, modyfikacja i usuwanie
- Wykonywanie zapytań
- Przeglądanie wyników zapytań
- Pobieranie obiektów wskazywanych przez referencje
- Ponowne wykonywanie wcześniejszych zapytań
- Oglądanie statystyk
- Import/eksport danych z/do XML

6.3. Interfejs użytkownika

Po pierwszym wejściu na stronę ukazuje się ekran logowania. Użytkownik może podać dane wypełniając pola, lub wpisując gotowy napis z zakodowanymi parametrami połączenia. Jeśli zaistnieje problem z brakiem obsługi JavaScript, AJAX lub modułu lxsbql, pojawia się stosowna informacja. Wpisanie poprawnych danych i zatwierdzenie powoduje przejście na stronę główną aplikacji.

Widok programu składa się z paska znajdującego się z lewej strony oraz głównej części. Pasek zawiera menu oferujące wykonanie dostępnych operacji oraz listę ulubionych zapytań i historię. Główna część widoku składa się z małej części zawierającej informację o stanie połączenia oraz treści bieżącej strony. Po zalogowaniu się jest to ekran powitalny.

W menu dostępne są dwie specjalne akcje: zakończenie sesji oraz otwarcie okienka z zapytaniem. Pozostałe powodują przejście na odpowiednią stronę do wykonania danej czynności. Przykładowo przeglądanie bazy danych wyświetla listę nazw obiektów będących korzeniami.

Po wykonaniu zapytania pokazuje się widok z wynikami. Składa się ze statystyk, widoku wyniku, ukrytego okna z treścią zapytania oraz przycisków umożliwiających wykonywanie czynności, takich jak dodanie zapytania do ulubionych, działające bez przeładowania strony. Statystyki zawierają informacje o czasie wymagania, a także o liczbie pomyślnie wykonanych zapytań przy wykonywaniu więcej niż jednego. Widok wyników zapytania został zrealizowany jako przeglądanie drzewa, gdzie kolekcje są węzłami, a wartości proste – liśćmi. Dla każdego węzła można ukryć jego zawartość klikając w przycisk „-”, a pokazać wybierając „+”. Są również dostępne skróty służące do rozwijania i chowania wszystkich węzłów. Domyślnie wszystkie węzły są rozwinięte. W przypadku gdy w wynikach znajduje się referencja, można zlecić pokazanie wskazywanego obiektu poprzez kliknięcie w odnośnik. Spowoduje to wykonanie asynchronicznego zlecenia, które przy powodzeniu zaktualizuje widok na stronie, uzupełniając go o obiekt pobrany z bazy danych. O takich obiektach wiadomo, że reprezentują utrwalone byty, więc można je modyfikować. Dla wartości atomowych istnieje możliwość zmiany wartości lub typu, a dla struktur można usuwać, zmieniać i dodawać pola.

6.4. XML – import i eksport

Aplikacja umożliwia wyeksportowanie wyników dowolnego zapytania do formatu XML, a także możliwość późniejszego ich zaimportowania. Dla referencji eksportowane są identyfikatory obiektów wskazywanych, aby podczas importowania wskazać na te same obiekty. Definicja typu dokumentu opiera się na modelu danych z biblioteki liblx i jest następująca:

```
<!ENTITY % datapart "void|int|double|string|bool|ref|bag|struct|seq|bind" >

<!ELEMENT data (part+) >
<!ELEMENT part (%datapart;) >
<!ELEMENT void EMPTY >
<!ELEMENT int (#PCDATA) >
<!ELEMENT double (#PCDATA) >
<!ELEMENT string (#PCDATA) >
<!ELEMENT bool (#PCDATA) >
<!ELEMENT ref (#PCDATA) >
<!ELEMENT bag ((%datapart;)*) >
<!ELEMENT struct ((%datapart;)*) >
<!ELEMENT seq ((%datapart;)*) >
<!ELEMENT bind (k, v) >
<!ELEMENT k (#PCDATA) >
<!ELEMENT v (%datapart;) >

<!ATTLIST data xmlns CDATA #IMPLIED>
<!ATTLIST part version CDATA #FIXED "1">
<!ATTLIST part id ID #IMPLIED>
```

W programie wyodrębniono następujące funkcje do konwersji danych:

- PHP → XML
- XML → PHP
- PHP → SBQL

Aby wykonać poprawną konwersję, dane wejściowe w formacie PHP muszą zawierać metainformacje.

6.5. Statystyki

W bazie LoXiM istnieje ogólny interfejs tworzenia widoków systemowych, a także widok pokazujący statystyki zgromadzone przez inny ogólny mechanizm. Obydwa zostały opisane w pracy [Kla08]. Organizacja taka sprzyja podejściu polegającemu na wymyśleniu ogólnego mechanizmu do prezentowania zebranych statystyk, który zostanie omówiony.

Zadaniem prezentowanego mechanizmu jest zautomatyzowanie generowania kodu HTML wizualizującego hierarchie obiektów z wartościami statystyk, oparte na spostrzeżeniu, że podobne struktury w widokach systemowych powtarzają się. Zatem kluczowym elementem jest opis struktury, wskazujący na sposób wyświetlania tabel na podstawie danych.

Hierarchia statystyk jest opisana przez zagnieżdżone tablice PHP. Główny obiekt deskryptora zawiera pary (klucz, wartość) opisujące dostępne rodzaje statystyk, z których każda wyświetla się na oddzielnej stronie. Klucz jest identyfikatorem, a wartość specyfikuje dany typ statystyki. Posiada pola „title” (wypisywany tytuł), „query” (zapytanie do wykonania) oraz „struct” (opis struktury). Początkowo przetwarzanym elementem jest wynik zapytania. Wyróżnia się następujące typy struktury:

- table1 – posiada tytuł. Wyświetla zawartość elementu w dwukolumnowej tabeli, w lewej kolumnie umieszczając klucze, a w prawej wartości.

- table1kv – posiada tytuł i nazwy pól z etykietą i treścią. Wyświetla taką samą tabelę jak wcześniejszy typ, ale informacje pobiera ze wskazanych pól elementów kolekcji, po której iteruje.
- table2 – posiada tytuł i opis pól, czyli dla każdego pola nazwę z wyniku zapytania oraz etykietę dla użytkownika. Tworzy dwuwymiarową tabelę, gdzie kolumny są polami, a wiersze elementami kolekcji. Struktura wyniku zapytania jest więc dokładnie taka sama jak dla danych relacyjnych.
- legend – posiada tytuł i opis pól. Tworzy legendę, czyli pionową tabelę, której wiersze to elementy statycznego opisu pól. Klucze, czyli pojęcia, umieszcza po lewej, a wartości – wyjaśnienia – po prawej.
- sequence – posiada informacje skąd brać tytuł, gdzie są dane oraz podstrukturę. Iteruje po bieżącej kolekcji, wyświetlając jej elementy według podstruktury. Tytuł może być brany z klucza lub z wybranego pola wyniku, dane mogą znajdować się w wartości albo we wskazanym polu wyniku. Różnica pomiędzy tymi dwoma podejściami jest analogiczna jak pomiędzy table1 a table1kv.
- combo – posiada podstruktury. Wyświetla wszystkie z nich dla bieżącego elementu. Przydatne dla mieszanej zawartości oraz w celu dodania legendy.

Na przykład struktura:

```
array("type" => "legend", "title" => "Komunikacja", "info" => array(
        "A" => "Autobus",
        "T" => "Tramwaj",
        "M" => "Metro",
));
```

Wygeneruje następujący kod (niezależnie od przetwarzanych danych):

```
<table class="statl">
  <tr class="title">
    <td colspan="2">Komunikacja</td>
  </tr>
  <tr>
    <th>A</th>
    <td>Autobus</td>
  </tr>
  <tr>
    <th>T</th>
    <td>Tramwaj</td>
  </tr>
  <tr>
    <th>M</th>
    <td>Metro</td>
  </tr>
</table>
```

Dzięki takiemu projektowi, dodanie obsługi dla nowego rodzaju statystyk jest łatwe, bo polega na uzupełnieniu odpowiednich wpisów w deskryptorze. Dostarczone typy struktury powinny wystarczać dla większości typowych zastosowań, jednak zawsze można dodać nowe.

Rozdział 7

Modyfikacje serwera

Aby możliwe było pełne zaimplementowanie panelu administracyjnego, niezbędne okazało się rozszerzenie bazy danych o nowe funkcjonalności, opisane w tym rozdziale. Wymagane było także usunięcie kilku drobnych błędów.

7.1. Perspektywa %Roots

%Roots jest to nowy widok systemowy, służący do wydobywania informacji o obiektach bazy danych znajdujących się w korzeniu. Powstanie tej perspektywy przewidział autor mechanizmu widoków systemowych w pracy [Kla08].

Widok ten zwraca strukturę, której elementy reprezentują korzenie bazy danych. Kluczami są ich nazwy, a wartościami – referencje do nich. Bez tego rozszerzenia nie byłoby możliwe przeglądanie bazy danych.

7.2. Leksem refid

refid jest nową wartością prostą w języku zapytań bazy LoXiM. Pełna składnia opisywana jest przez wyrażenie regularne: refid([0-9]+). Służy do tego, aby zaadresować obiekt o znanym identyfikatorze logicznym.

Składnię tę zaimplementowano, aby umożliwić dereferencję obiektów na życzenie, ponieważ nie istniał inny sposób na wykonanie tej czynności.

7.3. Pseudozmienna _position

_position jest to pseudozmienna, czyli zamiast faktycznych danych przechowywanych w bazie zwraca metadane. Służy do kolejnego numerowania elementów kolekcji, począwszy od 0. Może wystąpić po prawej stronie każdego operatora niealgebraicznego nierekurencyjnego i przesłania ewentualne pole obiektu o tej samej nazwie. W typowych zastosowaniach lewy argument powinien być kolekcją posortowaną. Przykłady użycia:

```
// pracownik zarabiający najmniej
(Emp ORDER BY sal) WHERE _position = 0

// ponumerowana lista nazwisk pracowników
(Emp ORDER BY name).(_position+1 as nr, name as nazwisko)
```

```
// ponumerowana lista pracowników, posortowana alfabetycznie
(Emp ORDER BY name) JOIN _position
```

Rozszerzenie to, wbrew przewidywaniom, nie znalazło zastosowania w implementacji panelu administracyjnego. Zaimplementowano je, ponieważ okazało się to bardzo proste.

Rozdział 8

Podsumowanie

8.1. Wstęp

W rozdziale opisano, o jakie dodatki została wzbogacona baza danych LoXiM. Zaimplementowano jedynie podstawową funkcjonalność, stąd istnieją pewne braki, które także tu przedstawiono.

8.2. Korzyści

Dzięki napisaniu pracy, baza danych zyskała dodatkowe komponenty:

- bibliotekę programistyczną dla języka C – liblx
- rozszerzenie dla języka PHP – lxsbql
- panel administracyjny WWW – lxpanel

Zadaniem interfejsów programistycznych jest umożliwienie pisania aplikacji klienckich bazy danych LoXiM. Zaprojektowanie ich w oparciu o analogiczne biblioteki dla bazy PostgreSQL, przy dostosowaniu do odmiennego modelu danych, powinno zapewnić późniejszą ich użyteczność.

Celem powstania panelu administracyjnego jest natomiast ułatwienie przeciętnemu użytkownikowi dostępu do bazy danych poprzez wizualizację danych i mechanizmy wspomagające wykonywanie typowych czynności. Także ta aplikacja wykorzystuje dobre cechy odpowiednika dla bazy PostgreSQL, dzięki czemu powinna być intuicyjna i prosta w obsłudze.

8.3. Zagadnienia do rozwiązania

W tej części znajduje się spis aktualnie znanych problemów w projekcie lub implementacji, wraz z propozycjami ich rozwiązań. Nie usunięto ich, ponieważ nie było to konieczne w celu zapewnienia poprawnej podstawowej funkcjonalności, a byłoby to zbyt czasochłonne.

8.3.1. Grupowanie alokacji w liblx

Biblioteka alokuje dwa rodzaje danych: struktury LXDATA oraz napisy. Bieżąca implementacja używa w tym celu funkcji bibliotecznej malloc, dodatkowo przeplatając alokacje dla różnych typów obiektów, co może prowadzić do nadmiernej fragmentacji pamięci.

Rozwiązaniem byłaby zmiana alokatora w taki sposób, aby dane tych samych typów były zgrupowane w większych blokach. Powinno się zaprojektować ogólny interfejs, parametryzowany rozmiarem obiektu, a następnie wykorzystać go do obydwu rodzajów danych.

8.3.2. Implementacja liblx

Biblioteka jest zaimplementowana jako jeden z modułów serwera. Wynikają z tego różne problemy, takie jak nadmierne zależności od bibliotek zewnętrznych, czy wydłużony czas kompilacji projektu.

Jednym z rozwiązań jest wyodrębnienie biblioteki do zewnętrznego projektu. Będzie to wymagało synchronizacji tych projektów, co prawdopodobnie doprowadzi do mniejszej spójności dwóch interfejsów i większych problemów przy późniejszych zmianach.

Alternatywnie można próbować skonfigurować narzędzia odpowiedzialne za kompilację projektu w ten sposób, aby ograniczyć zależności biblioteki do minimum. Będzie to prawdopodobnie wymagało ręcznego śledzenia zależności, co może okazać uciążliwe przy refaktoryzacjach.

8.3.3. Wykonywanie asynchronicznych zapytań

Bieżąca implementacja wykonuje wszystkie działania synchronicznie. Z tego powodu dostarczenie obsługi asynchronicznej jest kłopotliwe, a dodatkowo pojawia się problem z odpowiedzią na pakiety keepalive.

Modyfikacja powinna objąć najbardziej niskopoziomową część klienta, czyli klasę SimpleClient. Należy kod odpowiedzialny za komunikację sieciową przepisać w ten sposób, aby wykorzystywał asynchroniczne usługi systemu operacyjnego, takie jak funkcja poll. Zmiany powinno się rozpropagować także do wyższych warstw.

Po rozwiązaniu tego problemu możliwe będzie scalenie nadmiarowej klasy SimpleClient z klasą Client, która w momencie pisania projektu nie nadawała się do użycia, dlatego stworzono jej odpowiednik z odpowiednim API.

8.3.4. Trwałe połączenia z bazą danych w lxsbql

Język PHP obsługuje tworzenie trwałych zasobów, czyli istniejących nawet po zakończeniu pracy skryptu, mogących zostać powtórnie użytych przez nowe skrypty. Aby skorzystać z tej możliwości, interpreter musi działać pod nadzorem serwera WWW. Mechanizm ten szczególnie wykorzystuje się przy obsłudze połączeń z bazą danych, których nawiązywanie jest kosztowne.

W celu zaimplementowania funkcji lx_pconnect należy wykonać drobną refaktoryzację polegającą na wydzieleniu części wspólnej z lx_connect do osobnej funkcji. Taki kod będzie zdecydowanie lepiej zorganizowany, niż gdyby został bezmyślnie powielony.

8.3.5. Przekazywanie parametrów do zapytań

Wykonywanie zapytań z parametrami nie jest obsługiwane. Brakuje implementacji metody SimpleClient::executeParams.

Możliwe, że wymagane będą zmiany w sygnaturach niektórych funkcji odpowiedzialnych za to zadanie. Wartości można dostarczać w konkretnych typach, a nie jako napisy, co wymaga stwierdzenia jakiego są typu oraz zastosowania odpowiedniej konwersji. Nie jest oczywiste, która część interfejsu ma wykonywać te dodatkowe akcje.

8.3.6. Problem z odwzorowaniem danych do PHP

W praktyce jedynym występującym problemem jest obsługa struktur mających wiele wartości o takich samych kluczach. Obecnie następna wartość nadpisuje poprzednią, co może prowadzić do wrażenia utraty danych, co ewidentnie wpływa na poprawność.

Najbardziej oczywistym obejściem jest taka konstrukcja zapytań, aby grupować elementy o tych samych kluczach w kolekcje. Może to jednak wymagać dużego wysiłku, ponieważ nie można tego robić automatycznie, a przetworzone zapytania są dużo bardziej skomplikowane niż oryginalne.

Automatycznym rozwiązaniem jest dodanie nowej flagi do wywołania lx_fetch. Powodowałaby ona, że przy tworzeniu tablicy z reprezentacją kolekcji, każdy wielokrotny klucz będzie automatycznie przetworzony na podkolekcję. Wadą tego rozwiązania jest zależność struktury wyniku od konkretnych danych.

Innym rozwiązaniem jest stworzenie zamiennika tablic wbudowanych w PHP, który dopuszczałby dodawanie wielu elementów o tych samych kluczach. Niezbędne będzie zaprojektowanie sensownego API dla takiej klasy. Wadą takiego podejścia jest obniżona wydajność, w porównaniu do tablic PHP.

Drugim problemem, występującym znacznie rzadziej, jest możliwość wystąpienia dwóch bezpośrednio zagnieżdżonych elementów bind. Sytuacja taka zachodzi jedynie dla wyników zapytania nie reprezentujących utrwalonych obiektów, jak np. (1 as x) as y. Ręcznie zapytanie takie można poprawić następująco: (1 as x, 0 as null) as y. Automatyczna konwersja powinna utworzyć dodatkowy pośredni kontener pomiędzy elementami bind, gdy wykryje opisywaną sytuację.

Dodatek A

Dołączona płyta

A.1. Zawartość

Na płycie znajdują się następujące dane:

- praca.pdf – ten dokument w formacie PDF.
- loxim/ – obraz repozytorium projektu LoXiM, gdzie znajdują się pliki źródłowe zarówno bazy danych, jak i opisywanych interfejsów oraz aplikacji.

A.2. Instalacja bazy danych

Wymagany jest system operacyjny typu UNIX. W celu zainstalowania programu należy:

1. Skopiować folder z płyty lub pobrać z repozytorium do katalogu z uprawnieniami zapisu. Polecenia należy wykonywać w tym katalogu.
2. cmake -DCMAKE_INSTALL_PREFIX=/opt/loxim
3. make
4. make install
5. /opt/loxim/bin/loxim_server

A.3. Instalacja PHP

Rozszerzenie dla języka PHP powstawało dla wersji 5.2.9 i aby je zainstalować należy:

1. Rozpakować dystrybucję PHP i przejść do jej katalogu głównego.
2. Utworzyć katalog ext/lxsbql i skopiować tam wszystkie pliki z katalogu interfaces/php projektu.
3. ./buildconf --force
4. ./configure --with-lxsbql=/opt/loxim
5. make
6. Interpreter CGI powinien znaleźć się w ścieżce sapi/cli/php-cgi, a jego domyślny plik konfiguracyjny sapi/cli/php-cgi.ini

A.4. Instalacja panelu administracyjnego

Panel administracyjny instaluje się w następujący sposób:

1. Należy skopiować wszystkie pliki z katalogu interfaces/lxpanel do miejsca udostępnianego przez serwer WWW.
2. W pierwszych liniach wszystkich plików z rozszerzeniem .cgi w katalogu głównym należy umieścić poprawną pełną ścieżkę do interpretera PHP-CGI.
3. Konfiguracja aplikacji polega na modyfikacji pliku config.cgi.
4. Minimalna wymagana zawartość pliku konfiguracyjnego PHP jest taka:

   ```
   [php]
   short_open_tag = off
   magic_quotes_gpc = off
   ```

5. Należy zadbać aby uprawnienia wszystkich plików były prawidłowe, co jest zależne od konfiguracji serwera WWW.
6. W wyniku funkcji phpinfo() powinna pojawić się informacja o module lxsbql.

Bibliografia

[Sub04] Kazimierz Subieta, *Teoria i konstrukcja obiektowych języków zapytań*, Wydawnictwo PJWSTK, Warszawa 2004, ISBN 83-89244-29-2.

[OMG07] Object Management Group, *Next-Generation Object Database Standardization*, http://www.omg.org/docs/mars/07-09-13.pdf.

[Ros07] Tomasz Rosiek, *Przezroczyste odwzorowanie semistrukturalnej bazy danych na Javę*, Warszawa 2007.

[Kla08] Damian Klata, *Implementacja dynamicznych widoków systemowych w systemie LoXiM*, Warszawa 2008.

[pgref] The PostgreSQL Global Development Group, *PostgreSQL Reference Manual - Volume 2 Programming Guide*, Network Theory Ltd., 2007, ISBN 0-9546120-3-5.

[phppgsql] Ewald Geschwinde, Hans-Juergen Schoenig, *PHP and PostgreSQL Advanced Web Programming (Developer's Guide)*, Sams, 2008, ISBN 0672323826.

[prphp] Rasmus Lerdorf, Kevin Tatroe, Peter MacIntyre, *Programming PHP*, O'Reilly Media, Inc.; 2nd edition, 2006, ISBN 0596006810.

[phpopp] Matt Zandstra, *PHP Objects, Patterns, and Practice*, Apress; 2nd edition, 2007, ISBN 1590599098.