Relacyjne Bazy Danych

Andrzej M. Borzyszkowski
PJATK/ Gdańsk

materiały dostępne elektronicznie
http://szuflandia.pjwstk.edu.pl/~amb

Integracja ze środowiskami programistycznymi API

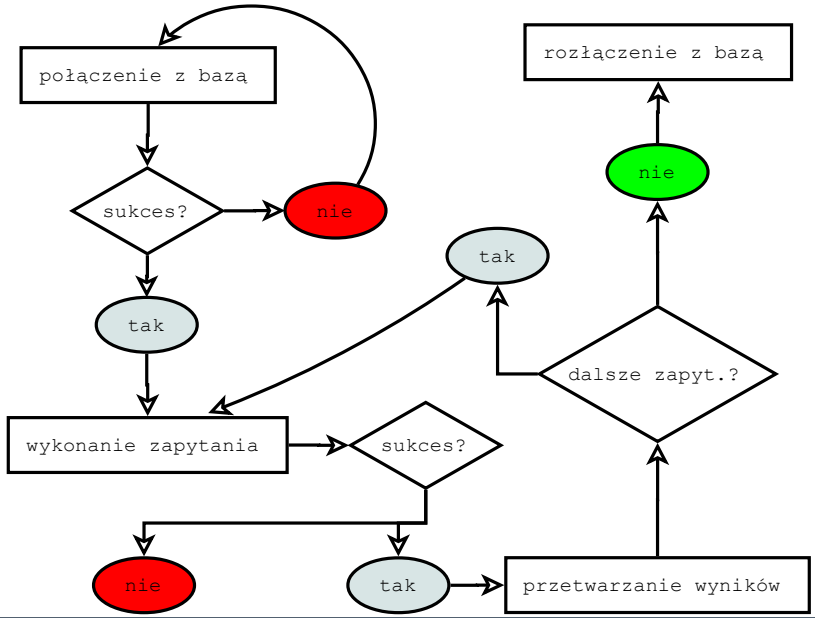
© Andrzej M. Borzyszkowski

Programowanie

- Po stronie serwera
  - PL/pgSQL
  - inne języki
  - procedury przechowywane i wyzwalane (trigger)
- Po stronie klienta
  - libpq, libpgeasy, libpqxx (C/C++)
  - ecpg (embedded C)
  - ODBC (open database connectivity)
  - JDBC (Java database connectivity)
  - perl, php, pgtcl, PyGreSQL (python)

© Andrzej M. Borzyszkowski

Struktura aplikacji klienckiej



© Andrzej M. Borzyszkowski

Dostęp z użyciem języka C (libpq)

- Biblioteka libpq
  - Wskazanie kompilatorowi cc katalogów z biblioteką i plikami nagłówkowymi
  - INC=/usr/local/pgsql/include
  - LIB=/usr/local/pgsql/lib -lpq
- Działanie programu:
  - połączenie z bazą danych
  - wykonanie (wielu instrukcji SQL)
  - m.in. BEGIN/ROLLBACK/COMMIT
  - rozłączenie z bazą danych

Krok 1. połączenie z bazą

```
#include <stdlib.h>
#include <libpq-fe.h>
int main()
{
  PGconn *mojepo;
  PGresult  *wynik;
/* połączenie z bazą danych*/
   mojepo = Pqconnectdb ( "dbname=____ password =' 123' user=ja" );
   if ( Pqstatus ( mojepo ) == NULL) printf ( "brak połączenia\n" );
   else if ( Pqstatus(mojepo ) == CONNECTION_OK ) printf ( "połączono\
    n" );
/* zakończenie połączenia z bazą danych ( na końcu, -- nie teraz!! ) */
  PQfinish ( mojepo );
  return EXIT_SUCCESS;
 }
```


Krok 2. pobranie wyników

```
/* pobranie informacji z bazy danych poprzez zapytanie SQL*/
    wynik = Pqexec ( mojepo, "SELECT * FROM klient" );
/* drukowanie nazw atrybutów */
    kolumny = PQnfields ( wynik );
    for ( i = 0; i < kolumny; i++ ) printf ( "%s\n", PQfname ( wynik, i ) );
    printf ( "\n\n" );
/* drukowanie wartości */
    for ( i = 0; i < PQntuples ( wynik); i++ ) {
        for (j = 0; j < kolumny; j++)
        printf( "%s", PQgetvalue ( wynik, i, j ) );
            printf( "\n" );
    }
/* oprócz zakończenia połączenia jeszcze czyszczenie pamięci */
  PQclear (wynik);
```


Funkcje z biblioteki libpq

- PQresultStatus
  - być może zapytanie było puste, być może zwróciło pusty wynik, być może wynik jest tabelą o zerowej liczbie wierszy, być może odpowiedź serwera jest błędna lub niezrozumiała
- PQresStatus
  - zamienia powyższe na czytelny napis
- PQresultErrorMessage
- PQgetisnull (wynik, i, j)
  - bez tej funkcji nie odróżni się pustego napisu od wartości NULL
- PQftype, PQfmod, PQfsize, PQgetlength,
  - pomagają ustalić typ i wielkość otrzymanego wyniku

Bazy danych via Internet

Dostęp sieciowy przez Web

- HTML
  - wyświetlanie tekstu/ grafiki
  - formularze do wprowadzania tekstu
- Powody:
  - łatwość integracji ze środowiskiem użytkownika
- Funkcje:
  - udostępnianie danych statystycznych (raporty)
  - udostępnianie danych konkretnych
    - (użytkownik wysyła dokładne zapytanie, przepływ dwustronny)
  - udostępnianie całych aplikacji
    - formularze, wprowadzanie danych
    - zapytania, wyświetlanie danych

© Andrzej M. Borzyszkowski
Relacyjne Bazy Danych

Dostęp sieciowy, narzędzia

- PHP
  - wersja 4 i wyżej
  - zainstalowana współpraca z serwerem (PostgreSQL w naszym przypadku)
- Java servlet
- JSP – Java Server Pages
- ASP – Active Server Pages

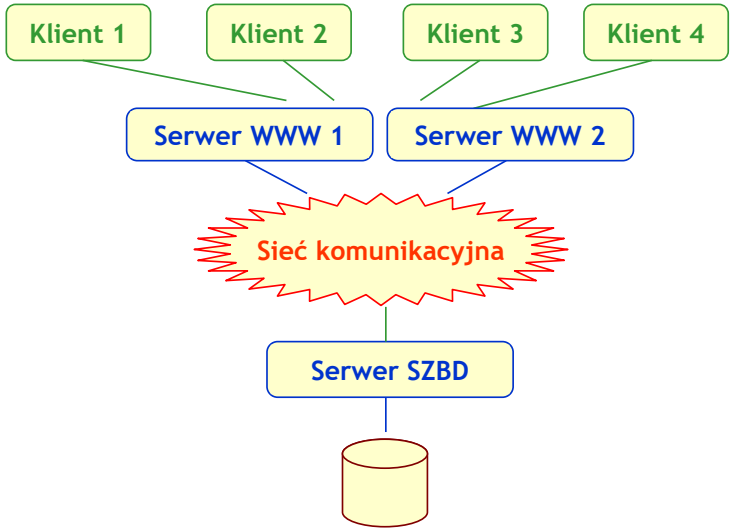
© Andrzej M. Borzyszkowski
Relacyjne Bazy Danych

HTML (*hypertext markup language*)

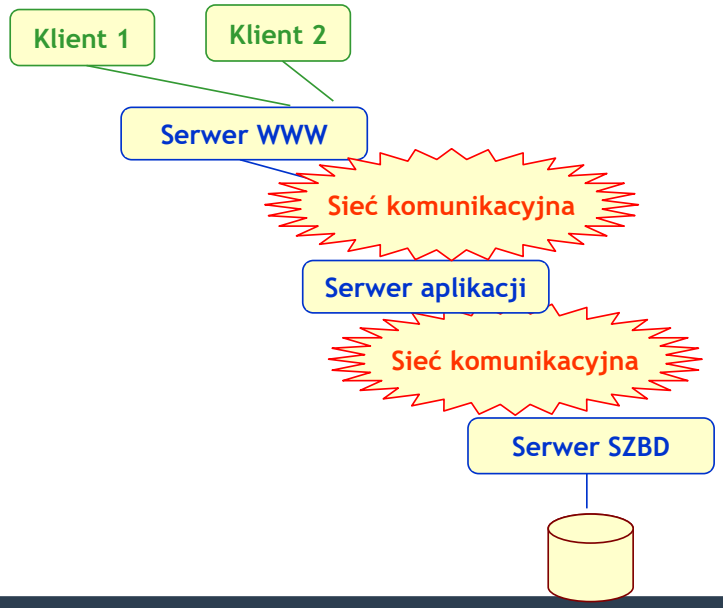
- Zasadniczo tekst statyczny
- Idea 1:
  - kod wykonywalny po stronie klienta
  - np. sprawdzanie poprawności wprowadzanych danych
  - narzędzia: javascript, flash, shockwave, applety
- Idea 2:
  - kod wykonywalny po stronie serwera
  - dynamiczne generowanie tekstu
  - w zależności od potrzeb użytkownika
  - w zależności od bieżącego czasu

© Andrzej M. Borzyszkowski
Relacyjne Bazy Danych

Architektura wielowarstwowa



Architektura wielowarstwowa –3 warstwy



Funkcje warstw: klient, WWW

- Klient (przeglądarka www, np. Mozilla)
  - wyświetlanie (renderowanie)
  - skrypty klienta
- Klient – serwer WWW: protokół http (*hypertext transfer protocol*)
- Serwer WWW (np. Apache)
  - przetwarzania argumentów adresu strony html
  - komunikacja z serwerem aplikacji
  - CGI (common gateway interface)
  - inne interfejsy
  - skrypty serwera
- Serwer WWW może być zintegrowany z serwerem aplikacji

Funkcje warstw: aplikacji, SZBD

- Serwer aplikacji (osobny lub zintegrowany z serwerem WWW)
  - przesyłanie zapytań do serwera bazodanowego
  - interpretacja odpowiedzi serwera bazodanowego
  - skrypty serwera
- Serwer WWW – serwer SZBD: protokół ODBC, JDBC (open/Java database connectivity)
  - Serwer SZBD (np. PostgreSQL)
  - przetwarzanie zapytań SQL
  - wysyłanie wyników

Sesje a HTTP, maszyna bezstanowa

- Protokół http nie przewiduje sesji
  - np. w uniksie login wykonywany jest jeden raz
  - połączenie z bazą danych nie jest rozłączane bez wyraźnej potrzeby
  - w przeciwieństwie, serwer WWW odczytuje podany adres, przetwarza go, odsyła tekst i zapomina o połączeniu
- Problem:
  - uwierzytelnianie przy połączeniu
  - modyfikacja ustawień
- Rozwiązanie:
  - ciasteczka (cookie)
  - są wysyłane przez serwer WWW, przechowywane u klienta, odczytywane przy kolejnych połączeniach

servlet (=server applet)

- servlet – komunikacja pomiędzy serwerem WWW a serwerem aplikacji
  - np. definiuje metody pozwalające serwerowi generować kod html
  - często metody do obsługi (zakładania/odczytywania) ciasteczek
- Kod servleta jest przekazywany od serwera aplikacji do serwera WWW
  - żądanie od klienta przesłania tekstu może wywołać servlet po stronie serwera WWW
  - może spowodować żądanie przesłania servleta, jeśli nie był jeszcze obecny

Java servlet, przykład

```
Public class BankQuery(Servlet extends HttpServlet {
  public void doGet(HttpServletRequest request,
  HttpServletResponse result)
          throws ServletException, IOException {
          String type =
  request.getParameter("type");
          String number =
  request.getParameter("number");
          result.setContentType("text/html");
          PrintWriter out = result.getWriter( );
          out.println("<HEAD><TITLE>Query
  Result</TITLE></HEAD>");
          out.println("<BODY>");
          out.println("Balance on " + type + number +
  "=" + balance);
          out.println("</BODY>");
          out.close ( );
  }}
```


Skrypty serwera

- Problem:
  - czy kod programu generujący tekst
  - czy kod html z fragmentami programistycznymi
- Idea: kod html z zanurzonymi servletami
- Nowsze języki:
  - Javascript, JScript, JSP, ColdFusion (cfml), Zope
- Klasyczne rozwiązania:
  - php, perl, python, VBScript

Przykład skryptu PHP

```
<?php
header('Content-Type: text/html; charset=iso-8859-2');
session_start();
echo('<HTML><HEAD>');
echo('<META http-equiv="content-type" content="text/html;
  charset=ISO-8859-2">');
echo('<TITLE>System zapisów na zabiegi</TITLE>');
echo("</HEAD><BODY><TABLE background=\"op.gif\"
  width=\"650\" border=\"0\">");
  $password=$_GET['password'];
if((strlen($password)!==11) || !is_numeric($password)) {
   echo('<TR><TD><center>Hasło ma niepoprawną budowę...
  </td></tr><tr><td><center><A
  href="index.html">Powrót</A></TD></TR></TABLE>');
    }
else {
```


Przykład skryptu PHP, c.d.

```
if(pg_connect("host=___ port=5432 dbname=___ user=___
  password=___")) {
  $data1=pg_exec("SELECT * FROM lekarz WHERE
  peselL='$password';");
  $is_lekarz=pg_num_rows($data1);
  if($is_lekarz) {
      echo ("<tr><td><H3>Witaj!!!</h3></td></tr>");
      while($row=pg_fetch_row($data1))
      {
        echo("<tr><td>dr ".$row[1]." ".$row[2]."<BR>".
  $row[3]."</td></tr>");
      echo("<TR><TD><form method=\"post\"
  action=\"terminarz.php\">");
      echo("<input type=\"hidden\" name=\"password\"
  value=\"$password\">");
      echo ("<SELECT name=\"dzien\">");
              for($i=1;$i<32;$i++) {
              echo ("<OPTION value=\"$i\">".$i);
              }
```


Przykład dla języka Python

```
#!/usr/bin/python
# -*- coding: iso-8859-2 -*-

import menu
from escape import ES

mn = menu.menu('news.cgi')

mn.header()
print '<CENTER><TABLE align="middle" border="0">'
if mn.dbIsOk():
  if mn.checkAccess('MSG ADD'):
    print '<a href="addnews.cgi">Dodaj
  wiadomość</a> '
```


Przykład z JSP

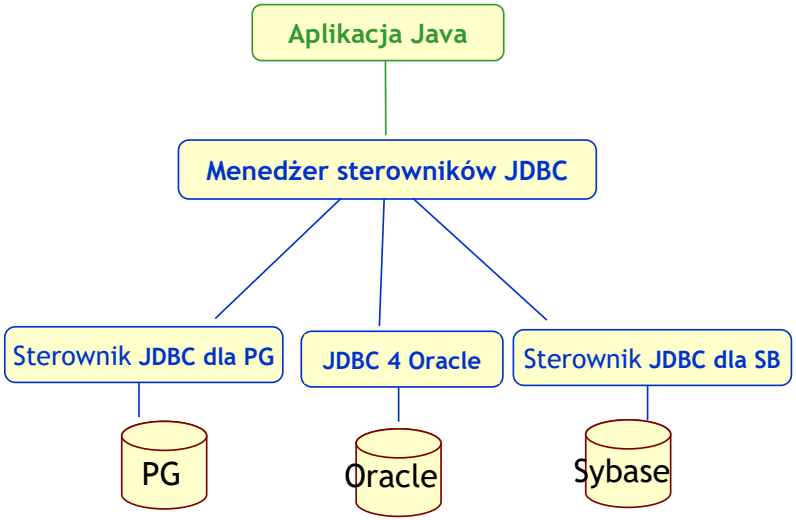
```
<HTML>
<BODY>
<FORM METHOD=POST ACTION="SaveName.jsp">
What's your name? <INPUT TYPE=TEXT NAME=username
          SIZE=20 VALUE="<%= user.getUsername() %>">
<P><INPUT TYPE=SUBMIT>
</FORM>
</BODY>
</HTML>
```

- wartość odczytana w formularzu jest przekazywana do metody
- można też generować dynamicznie tagi

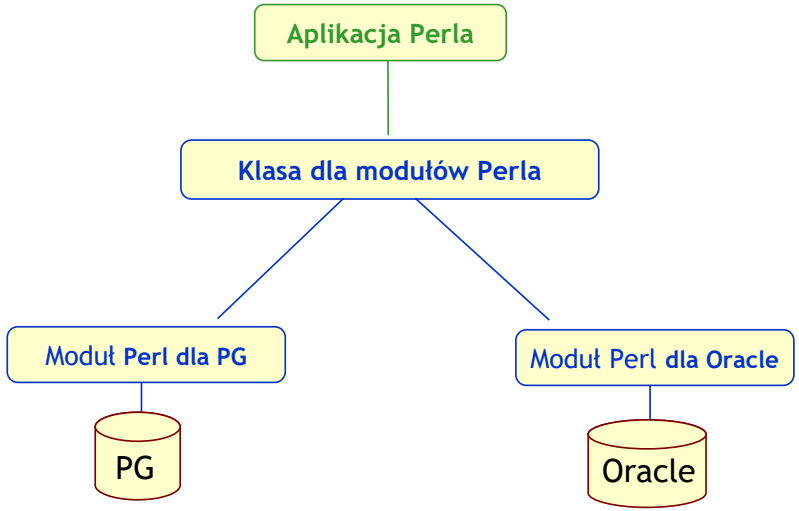
ODBC/JDBC

- Być może dwa najważniejsze interfejsy do użycia bazy danych
  - NIE są ograniczone tylko do PostgreSQL
  - są przeznaczone do komunikacji z każdą bazą danych
  - słowa kluczowe to np. SQLConnect, SQLError, executeStmt , itp.
- ODBC jest oparty na języku C, JDBC na Javie
  - tzn. program napisany z użyciem *DBC będzie współpracował z *każdą* bazą danych
  - ale nie będzie mógł wykorzystać specyficznych cech konkretnej bazy danych
  - a jeśli jakieś rozszerzenie potrafi je wykorzystać, to nie będzie działać z inną bazą danych

JDBC – architektura wielowarstwowa



Perl – również architektura wielowarstwowa



Atak *SQL injection*

Atak *SQL injection*

- Przykład PHP

```
$data1=pg_exec("
      SELECT * FROM lekarz WHERE peselL='$password';
      ");
```

  - jako parametr zapytania ma być przekazany numer pesel np. `12345678910` i ma być wykonane wykonane zapytanie `SELECT * FROM lekarz WHERE peselL='12345678910';`
  - złośliwy użytkownik zapytany o numer pesel być może wstawi wartość `'; DELETE FROM lekarz; SELECT '` i spowoduje wykonanie zapytania SQL `SELECT * FROM lekarz WHERE peselL=''; DELETE FROM lekarz; SELECT '';`
  - złośliwy użytkownik może na każdy parametr podawać `12345678910' OR '1'='1` i obserwować wywołane efekty

Atak *SQL injection* 2

- Możliwe szkody:
  - nieautoryzowane zmiany w zawartości bazy danych
  - dostęp do danych poufnych/masowych
  - atak DOS (*denial of service*, odmowa usługi) czyli przeciążenie serwera bazodanowego
  - zbadanie dokładnej struktury bazy danych
  - wykonanie poleceń systemowych
- Warunki wykonania ataku
  - precyzyjna znajomość oprogramowania przez atakującego

Obrona przed *SQL injection*

- Zabezpieczenie na poziomie aplikacji:
  - używanie znaków specjalnych, na pewno apostrofów
    - każdy z języków będzie miał swoje rozwiązania
    - np. w PHP `$data1=pg_exec("SELECT * FROM lekarz WHERE peselL='addslashes($password)';");`
    - zapytanie `SELECT * FROM lekarz WHERE peselL='\ 12345678910\ ';` będzie równoważne poprzedniemu
    - zapytanie `SELECT * FROM lekarz WHERE peselL='\'; DELETE FROM lekarz; SELECT \'';` nie będzie groźne i nie zwróci żadnych wyników

Obrona przed *SQL injection* 2

- Zabezpieczenie na poziomie aplikacji:
  - można pilnować poprawności typów,
  - sprawdzać postać parametrów,
  - nie dopuszczać by zawierały pewne znaki
    - apostrof,
    - nawiasy (podejrzenie użycia funkcji),
    - średniki (kilka poleceń),
    - dwa minusy (komentarz),
    - nazwy tabel systemu bazodanowego,
    - itp.
  - sama długość parametrów też powinna być niewielka

Obrona przed *SQL injection* 3

- Zabezpieczenie na poziomie serwera bazodanowego
  - minimalne uprawnienia użytkownika
  - mechanizm zapytań może dopuszczać parametry – należy wówczas z tego korzystać
- Zabezpieczenia na poziomie serwera aplikacji
  - analiza uprawnień użytkownika końcowego
  - analiza zapytań przesyłanych do bazy danych
    - zakaz wykonania wielu zapytań w jednym
    - zakaz wywoływania poleceń systemowych