# Bazy danych 2 - projekt

*Autor:*
Tymon Tobolski (181037)
Jacek Wieczorek (181043)

*Prowadzący:*
Mgr inż. Mariusz Słabicki

Wydział Elektroniki
III rok
Cz 09.15 - 11.00

12 stycznia 2012

# 1 Cel projektu

Celem projektu jest zaprojektowanie i zaimplementowanie systemu do zarządzania projektem opartego na obiektowych bazach danych.

# 2 Założenia projektu

System do zarządzania projektami IT będzie oferował następujące funkcjonalności : możliwość dodawania użytkowników wraz ze statusami, prawa poszczególnych użytkowników, harmonogramowanie zadań, listy TODO, punkty kontrolne/kamienie milowe. Projekt oparty jest na obiektowej bazie danych Neodatis [1], zaimplementowanej w języku Java, interfejs użytkownika we frameworku Ruby on Rails [2].

# 3 Opis funkcjonalności

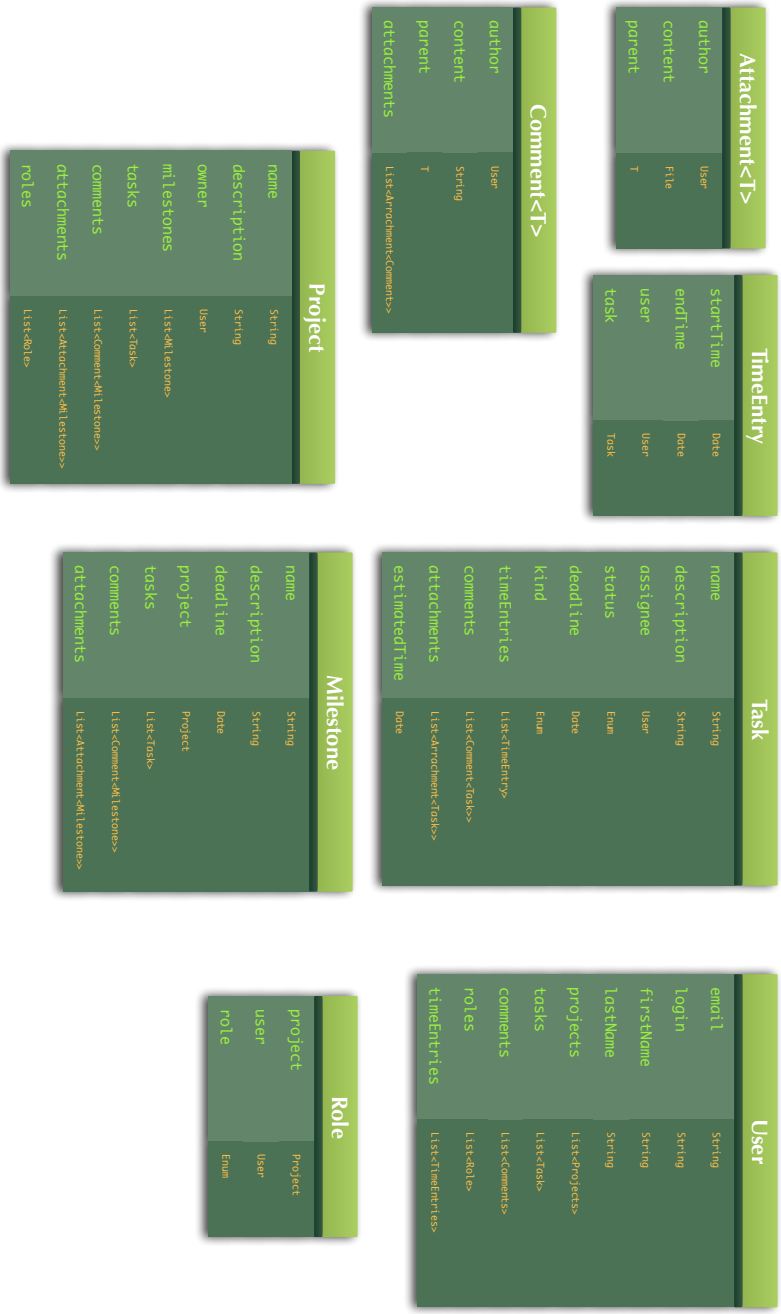
- Tworzenie projektów
  - Definiowanie nazwy, terminów, osób odpowiedzialnych, kamieni milowych
- Dodawanie nowych zadań do wykonania
  - Przypisanie do projekt/kaminienia milowego
  - Zdefiniowanie osoby odpowiedzialnej za zadanie
  - Estymacja i logowanie czasu spędzonego nad zadaniem
- Zarządzanie zadaniami do wykonania
  - Edycja zadań
  - Dodawanie komentarzy i plików
  - Zmiana stanu
- Zarządzanie uprawnieniami użytkowników
  - Dostęp do projektów
  - Pozwolenie na edycje / tylko do wglądu (konto klienta)
  - Komentowanie zadań

[1] Obiektowa baza danych Neodatis - `http://neodatis.org`

[2] Framework sieciowy Ruby on Rails - `http://rubyonrails.org/`

  - Publikacja załączników
- Statystyka
  - Przebieg czasowy zadań (czas rozpoczęcia, zakończenia)
  - Opóźnienia względem terminów

# 4 Model bazy danych




Rysunek 1: Model bazy danych

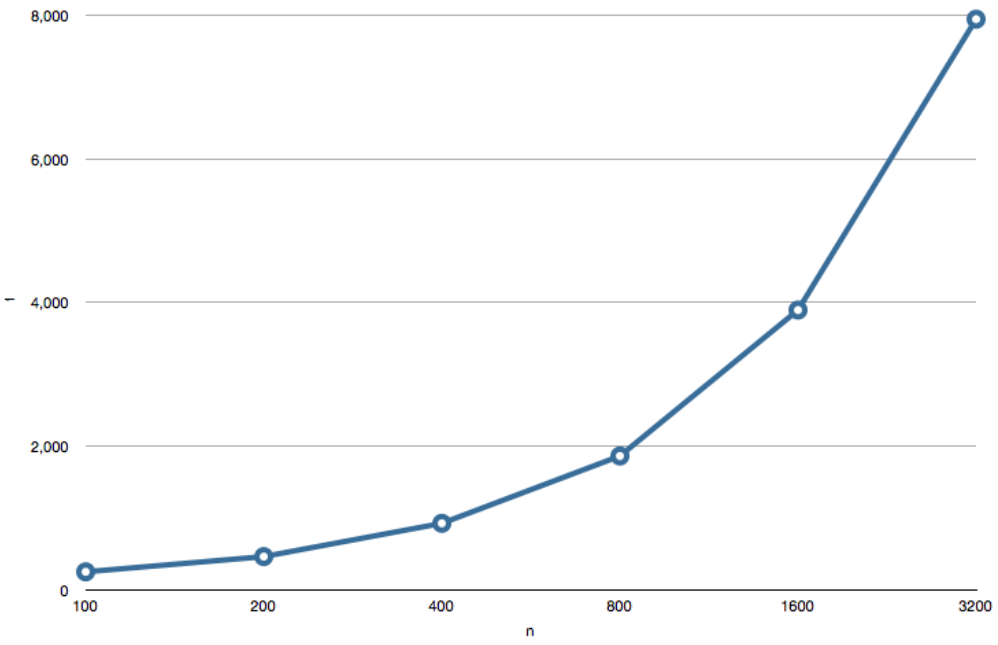
# 5 Testy wydajnościowe

## 5.1 Wydajność wyszukiwania

Przeprowadzony test polegał na wyszukaniu 1 000 pojedynczych rekordów na podstawie parametru *id* dla różnych wielkości bazy danych.

| $n$ | $t[ms]$ | $n/t$ |
|---|---|---|
| 100 | 251 | 2,5100 |
| 200 | 463 | 2,3150 |
| 400 | 926 | 2,3150 |
| 800 | 1 863 | 2,3288 |
| 1600 | 3 894 | 2,4338 |
| 3200 | 7 942 | 2,4819 |

Tabela 1: Czas wyszukiwania w zależności od wielkości bazy danych




Rysunek 2: Czas wyszukiwania w zależności od wielkości bazy danych

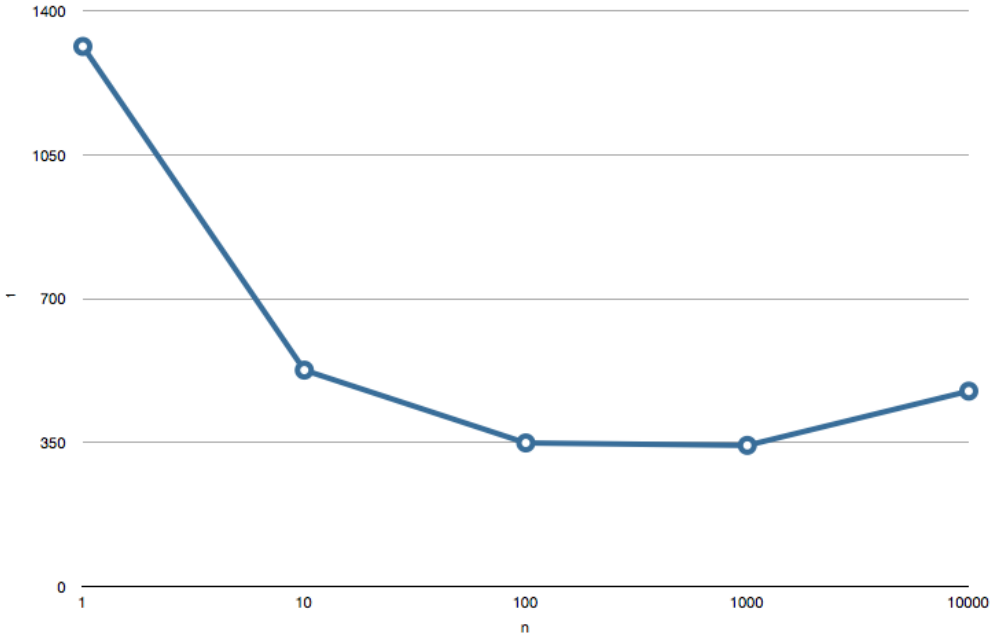
Jak widać w Tabeli 1 oraz na Rysunku 1 czas wyszukiwania jest liniowo zależny od ilości rekordów w bazie. Wpływ na to ma algorytm wyszukiwania, który korzysta z przeglądu zupełnego wszystkich obiektów bazy danych.

## 5.2 Wydajność zapisu

Zapis danych w obiektowej bazie danych NeoDatis dzieli sie na dwa etapy. Pierwszym etapem jest operacja `odb.store(obj)`, a kolejnym operacja `odb.commit()` przypominająca nieco operacje `COMMIT` znaną z operacji na transakcjach w relacyjnych bazach danych. Dodatkowo, w połączeniu klient-serwer używanym przez aplikacje dopiero operacja `odb.commit()` przesyła dane do serwera bazy danych. W związku z narzutem czasowym na transfer informacji po sieci zapisywanie i wysyłanie każdego obiektu osobno może okazać się mało wydajne. Test polegał na zapisaniu (za pomocą `odb.store(obj)`) do bazy danych 10 000 obiektów w grupach po 1, 10, 100, 1 000 oraz 10 000 obiektów. Po zapisaniu grupy obiektów wywoływana była metoda `odb.commit()` odpowiedzialna za przesłanie zapisanych danych do serwera. Table 2 oraz Rysunek 2 prezentują wyniki testu.

| $n$ | $t[ms]$ |
|---|---|
| 1 | 1 313 |
| 10 | 526 |
| 100 | 349 |
| 1000 | 343 |
| 10000 | 475 |

Tabela 2: Czas zapisu w zależności od wielkości paczki




Rysunek 3: Czas zapisu w zależności od wielkości paczki

Po przeprowadzeniu testów okazało się, że najlepsze rezultaty daje zapisywanie obiektów w paczkach po 100 - 1 000 obiektów jednocześnie. Jednorazowe wysłanie 10 000 obiektów okazało się dłuższe, niż np. 10 razy po 1 000 obiektów.

# 6 Wnioski

W porównaniu do relacyjnych baz danych bazy obiektowe nie są ograniczone przez tabele z danymi typu prostego (liczba, ciąg znaków, data) lecz pozwalają na trzymania danych dowolnych typów w dowolnie zagnieżdżonej strukturze. W łatwy sposób można przechować relacje jeden do wielu (za pomocą listy) czy wiele do wiele (za pomocą dwóch list). Z założenia obiektowych baz danych przechowywane obiekty nie wymagają specjalnego klucza głównego. W przypadku bazy danych NeoDatis działającej na maszynie wirtualnej Javy kluczami obiektów są wewnętrzne identyfikatory obiektów maszyny wirtualnej. Pozwala to na przeźroczyste przechowywanie danych obiektowych.

Wykorzystanie obiektowej bazy danych w aplikacji sieciowej pociąga za somą pewne konsekwencje. Ze względu na bezstanowość protokołu HTTP, wszysktie obiekty, na których mają być wykonywane operacja muszą posiadać niezmienny identyfikator, przesyłany jako część adresu URL lub w parametrach POST zapytania HTTP. Jest to równoznaczne ze sztucznym wprowadzeniem unikalnego identyfikatora dla każdego obiektu zapisanego w bazie danych. Obiektowa baza danych NeoDatis nie posiada mechanizmu automatycznej inkrementacji pola (`AUTO INCREMENT`). W związku z tym taką funkcjonalnośc należało napisać samemu. Przydzielenie nowego identyfikatora numerycznego (*id*) podczasu zapisu nowego obiektu sprowadza się do przeszukania całej bazy danych i wybranie maksymalnej wartości parametru *id*, a następnie przypisanie do nowego obiektu wartości o 1 większej. Taka operacja jest mało wydajna w porównaniu do podobnych mechanizmów w relacyjnych bazach danych.

W celu wykorzystania potencjału obiektowej bazy danych, należy też pamiętać, żeby przy relacjach uzupełniać dane po obu stronach relacji. Dla przykładu, przy relacji jeden do wielu w bazach relacyjnych wystarczy zapisać tylko identyfikator rekordu A w polu klucza obcego rekordu B. W obiektowej bazie danych należy przypisać referencje do obiektu A w obiekcie B, ale też dodać obiekt B do kolekcji obiektów typu B w obiekcie A. Bez zapisania obiektu B na liście zatraca się idea bazy obiektowej i wyszukiwanie elementów prowadza się do przeszukania całej bazy w poszukiwaniu obiektu o zadanym kluczu obcym.

Ze względu na bezstanowość protokołu HTTP traci się wiele zalet obiektowych baz danych, a także zwiększa się czas odczytu i zapisu danych. Nie

bez znaczenia jest konieczność implementacji własnych mechanizmów kontroli poprawności (unikalne identyfikatory, relacje).

Biorąc pod uwagę specyfikę i zasadę działania obiektowych baz danych, znajdą one szersze zastosowanie w aplikacjach desktopowych (wszędzie tam, gdzie można w prosty sposób połączyć interfejs użytkownika z modelem danych) niż w aplikacjach sieciowych opeartych o bezstanowy protokół HTTP.