

**AKADEMIA EKONOMICZNA IM. KAROLA ADAMIECKIEGO W KATOWICACH  
WYDZIAŁ INFORMATYKI I KOMUNIKACJI**

**INFORMATYKA I EKONOMETRIA**

MICHAŁ PŁONKA

**OPTYMALIZACJA WYDAJNOŚCI  
SERWISÓW INTERNETOWYCH  
WEB SERVICES EFFICIENCY TUNING**

Praca magisterska  
napisana w Katedrze Inżynierii Wiedzy  
pod kierunkiem dr Barbary Filipczyk

Oświadczam, że niniejsza praca została przygotowana pod moim kierunkiem  
i stwierdzam, że spełnia wymogi stawiane pracom magisterskim

.....  
(data)

.....  
(podpis promotora pracy magisterskiej)

KATOWICE 2009

Katowice, dnia 10.06.2009

Michał Płonka

Wydział Informatyki i Komunikacji

Informatyka i Ekonometria

## OŚWIADCZENIE

Świadom odpowiedzialności prawnej oświadczam, że złożona praca magisterska pt.: „Optymalizacja wydajności serwisów internetowych” została napisana przeze mnie samodzielnie.

Równocześnie oświadczam, że praca ta nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz. U. 1994, nr 24, poz. 83) oraz dóbr osobistych chronionych prawem.

Ponadto praca nie zawiera informacji i danych uzyskanych w sposób nielegalny i nie była wcześniej przedmiotem innych procedur związanych z uzyskaniem dyplomów lub tytułów zawodowych uczelni wyższej.

Wyrażam zgodę na przetwarzanie moich danych osobowych oraz nieodpłatne udostępnienie mojej pracy w celu oceny samodzielności jej przygotowania przez system elektronicznego porównywania tekstów oraz przechowywania jej w bazie danych tego systemu.

Oświadczam także, że wersja pracy znajdująca się na przedłożonej przeze mnie płycie CD jest zgodna z wydrukiem komputerowym pracy.

.....

(podpis składającego oświadczenie)

## SPIS TREŚCI

<b>WSTĘP .....</b>	<b>5</b>
<b>ROZDZIAŁ 1. WPROWADZENIE DO PROBLEMATYKI WYDAJNOŚCI SERWISÓW INTERNETOWYCH.....</b>	<b>7</b>
1.1. ZNACZENIE WYDAJNOŚCI SERWISÓW INTERNETOWYCH DLA ORGANIZACJI.....	7
1.2. CZYNNIKI WPŁYWAJĄCE NA WYDAJNOŚĆ SERWISÓW INTERNETOWYCH.....	10
1.3. KORZYŚCI WYNIKAJĄCE Z OPTYMALIZACJI WYDAJNOŚCI SERWISÓW INTERNETOWYCH .....	17
<b>ROZDZIAŁ 2. WARSTWY OPTYMALIZACJI WYDAJNOŚCI SERWISÓW INTERNETOWYCH .....</b>	<b>23</b>
2.1. WARSTWA PREZENTACJI DANYCH .....	23
2.2. WARSTWA LOGIKI APLIKACJI .....	27
2.3. WARSTWA BAZY DANYCH.....	34
<b>ROZDZIAŁ 3. NARZĘDZIA TESTUJĄCE WYDAJNOŚĆ SERWISÓW INTERNETOWYCH .....</b>	<b>41</b>
3.1. SPOSOBY BADANIA WYDAJNOŚCI SERWISÓW INTERNETOWYCH.....	41
3.2. BADANIE WYDAJNOŚCI ZAPYTAŃ SQL ZA POMOCĄ INSTRUKCJI EXPLAIN W BAZACH DANYCH.....	42
3.3. PROFILOWANIE KODU APLIKACJI ZA POMOCĄ BIBLIOTEKI BENCHMARK Z REPOZYTORIUM PEAR.....	47
3.4. TESTOWANIE WYDAJNOŚCI SERWISÓW INTERNETOWYCH ZA POMOCĄ ZAAWANSOWANYCH APLIKACJI.....	51
3.4.1. <i>ApacheBench</i> .....	51
3.4.2. <i>Jmeter</i> .....	55
<b>ROZDZIAŁ 4. PRZYKŁAD OPTYMALIZACJI WYDAJNOŚCI SERWISU INTERNETOWEGO OPROGRAMOWANEGO W JĘZYKU PHP.....</b>	<b>60</b>
4.1. CHARAKTERYSTYKA TESTOWEGO SERWISU INTERNETOWEGO ORAZ ŚRODOWISKA TESTOWEGO .....	60

4.2.	WYJŚCIOWA WERSJA TESTOWEGO SERWISU INTERNETOWEGO.....	65
4.3.	POMIAR WYDAJNOŚCI TESTOWEGO SERWISU INTERNETOWEGO.....	70
4.4.	OPTIMALIZACJA WYDAJNOŚCI TESTOWEGO SERWISU INTERNETOWEGO .....	71
4.5.	POMIAR WYDAJNOŚCI ZOPTYMALIZOWANEGO SERWISU INTERNETOWEGO ....	76
4.6.	PORÓWNANIE WYNIKÓW TESTÓW .....	77
<b>ZAKOŃCZENIE.....</b>		<b>79</b>
<b>LITERATURA.....</b>		<b>81</b>
<b>SPIS RYSUNKÓW I TABEL .....</b>		<b>84</b>

## WSTĘP

Pojęcie optymalizacji wydajności serwisów internetowych jest zagadnieniem bardzo obszernym i można je rozpatrywać patrząc z różnych punktów widzenia, do których zaliczyć można m.in. technologię oraz technikę wykonania, a także zaplecze techniczno-sprzętowe. Rozpatrując problem wydajności serwisów internetowych od strony wykonania należy skupić się na odpowiednim ich zaprojektowaniu oraz zaprogramowaniu. Ponadto ważną kwestią jest to, w jakich technologiach dany serwis internetowy powinien zostać wykonany. Biorąc natomiast pod uwagę zaplecze techniczno-sprzętowe ważną staje się dbałość o dobór odpowiednich komponentów takich jak poszczególne podzespoły serwera, a także jego umiejętna konfiguracja. W celu uzyskania maksymalnej wydajności serwisu internetowego należy zadbać o jak najlepsze dostrojenie obu tych aspektów każdego z osobna ale również o odpowiednie zestrojenie ich ze sobą. Można bowiem serwis internetowy traktowany jako całość porównać do łańcucha; jeżeli którekolwiek z ogniw okaże się zbyt słabe wówczas cały łańcuch również będzie słaby.

Optymalizacja wydajności serwisów internetowych odgrywa bardzo istotną rolę zarówno dla użytkownika końcowego (czyli internauty), jak i dla właściciela (tu: organizacji). Dla każdego z nich optymalizacja wydajności serwisu internetowego przynosi wymierne korzyści. Patrząc z punktu widzenia internauty kluczową sprawą staje się wygoda korzystania z serwisu internetowego, natomiast od strony organizacji ważne są koszty związane z jego eksploatacją, utrzymaniem oraz rozwojem.

Celem niniejszej pracy jest przedstawienie metod optymalizacji wydajności serwisów internetowych oraz zastosowanie ich w celu poprawy wydajności autorskiego oprogramowania sklepu internetowego. Autor skupia się w niej na opisanu zagadnień optymalizacji wydajności związanych z projektowaniem oraz tworzeniem serwisów internetowych wykorzystujących język PHP oraz bazę danych PostgreSQL.

Pierwszy rozdział pracy jest wprowadzeniem do problematyki wydajności serwisów internetowych. Zawiera on wyjaśnienie tego, jakie znaczenie ma optymalizacja wydajności serwisów internetowych, jakie czynniki na nią wpływają, a także jakie korzyści można uzyskać dzięki odpowiedniej optymalizacji serwisów internetowych.

W drugim rozdziale scharakteryzowane zostały podstawowe obszary optymalizacji wydajności serwisów internetowych, na których należy się skupić podczas ich projektowania oraz tworzenia. Rozdział ten przedstawia sposoby optymalizacji warstwy prezentacji danych, logiki aplikacji oraz bazy danych.

Rozdział trzeci prezentuje wybrane narzędzia służące do testowania wydajności serwisów internetowych wraz z ich przykładowym wykorzystaniem oraz z interpretacją uzyskanych wyników. Wśród opisywanych narzędzi znajduje się bazodanowa instrukcja EXPLAIN, biblioteka Benchmark należąca do repozytorium PEAR, a także zaawansowane aplikacje testowe takie jak ApacheBench oraz JMeter.

W czwartym rozdziale został przedstawiony przykład optymalizacji wydajności serwisu internetowego z wykorzystaniem autorskiego systemu sklepu internetowego napisanego w języku PHP z wykorzystaniem bazy danych PostgreSQL. Rozdział ten zawiera informacje dotyczące poziomu wydajności serwisu internetowego przed i po optymalizacji wydajności oraz przykłady dokonanych modyfikacji.

Niniejszą pracę napisano z wykorzystaniem pozycji książkowych w języku polskim oraz materiałów pozyskanych z sieci Internet, a także w oparciu o wiedzę i doświadczenie autora. Do realizacji części praktycznej wykorzystano następujące oprogramowanie: framework Kohana wspomagający programowanie w języku PHP, bazę danych PostgreSQL, a także narzędzie testujące wydajność serwisów internetowych ApacheBench,

## **ROZDZIAŁ 1. WPROWADZENIE DO PROBLEMATYKI WYDAJNOŚCI SERWISÓW INTERNETOWYCH**

W dobie coraz dynamicznej rozwijającej się sieci Internet serwisy internetowe stają się częścią życia globalnego społeczeństwa. To dzięki nim można mieć praktycznie cały świat na wyciągnięcie ręki. Serwisy informacyjne, sklepy internetowe czy też bankowość elektroniczna umożliwiają wykonanie w kilka sekund tego, co do tej pory wymagało zdecydowanie więcej czasu i wysiłku.

Jednak aby serwisy internetowe funkcjonowały w sposób sprawny koniecznym jest zapewnienie ich odpowiedniej wydajności. W poniższym rozdziale autor skupi się na przybliżeniu znaczenia wydajności serwisów internetowych, czynników na nią wpływających oraz korzyści, jakie można osiągnąć w wyniku odpowiedniej optymalizacji wydajności serwisów internetowych.

### **1.1. Znaczenie wydajności serwisów internetowych dla organizacji**

Wydajność serwisów internetowych jest (obok funkcjonalności, niezawodności, użyteczności, modyfikalności oraz przenośności [zob. Ziemia 2005 s.32]) podstawowym elementem, o który powinni zadbać ich projektanci oraz programiści. Serwis, posiadający bogatą szatę graficzną oraz mnogość funkcji, może okazać się kompletnie bezużyteczny jeżeli w parze z tymi elementami nie będzie szła odpowiednia wydajność umożliwiająca jego efektywne wykorzystanie. Bardzo często wydajność serwisów internetowych ma istotne znaczenie dla powodzenia całego projektu. Uzyskanie maksymalnego tempa działania może oznaczać powodzenie lub porażkę, więc planowanie aplikacji w sposób pozwalający na szybkie działanie jest bardzo ważne [zob. Lecky-Thompson 2005 s.621].

Rozpatrując zagadnienie znaczenia wydajności serwisów internetowych można spojrzeć na nie z dwóch perspektyw:

- z punktu widzenia internauty,
- z punktu widzenia organizacji.

Patrząc z punktu widzenia internauty, który jest użytkownikiem danego serwisu internetowego, ogromne znaczenie posiada to, jak szybko serwis ten reaguje na jego polecenia. Nie do przyjęcia jest sytuacja, w której użytkownik zmuszony jest oczekiwać dłuższy okres czasu na to, aby serwis internetowy wykonał jego polecenie. Reakcja powinna być niemal natychmiastowa. Wszelkie opóźnienia z pewnością wpłyną w sposób negatywny na postrzeganie serwisu przez użytkownika. Stanie się on dla niego po prostu kłopotliwy w użytkowaniu. „Cechą systemów e-biznesu jest brak możliwości przewidywania ich granicznych obciążeń oraz ich niejednorodność. Moc transakcyjna systemu powinna zapewnić uzyskanie w maksymalnie krótkim czasie odpowiedzi na działania użytkowników, niezależnie od ich jednoczesnej liczby” [Sroka 2005 s.90].

Podchodząc do problematyki znaczenia wydajności serwisów internetowych z punktu widzenia organizacji należy skupić się na finansowych aspektach prowadzenia serwisu internetowego. Wiązą się one zarówno z wydatkami, jakie należy ponieść na jego utrzymanie i prowadzenie, jak i z przychodami, które może generować. „Dla projektów programistycznych opartych na Internecie istotne znaczenie mają zagadnienia obejmujące uwierzytelnianie, szyfrowanie i kontrolowanie sesji. Stąd dla inżynierów oprogramowania istotne będą takie pojęcia inżynierii oprogramowania jak utrzymywanie, testowanie oraz współpraca. Parametry techniczne ograniczone będą dostępnością konkretnych rozwiązań, jak i możliwościami budżetu przeznaczzonego na zakup infrastruktury technicznej” [Sroka 2005 s.91].

Paradoksalnie wydajność serwisów internetowych ma obecnie większe znaczenie niż miało to miejsce kilka lat temu, gdy prędkość łączy internetowych nie była tak wysoka jak obecnie. Wówczas normalnym było kilkusekundowe (lub wręcz dłuższe) oczekiwanie na wyświetlenie zawartości żądanej strony. Nikt nie myślał wtedy o tym, iż niesie to ze sobą negatywne skutki. Serwisy internetowe były czymś nowym, nieznanym, czymś, co przyciągało i fascynowało. Jednak wraz ze wzrostem popularności sieci Internet rosły również możliwości transferowe łączy internetowych. Równocześnie zwiększały się również wymagania użytkowników, którzy oczekiwali przyspieszenia wczytywania się stron. Był to okres gdy zaczęto intensywnie korzystać z nowych technologii takich jak m.in. animacje Flash lub aplety Java. Elementy te wyglądały bardzo efektownie i stanowiły o nowoczesności witryny jednak skutecznie wydłużały czas jej wczytywania. Po dość krótkim okresie stwierdzono, iż nie nadają się one do zastosowań w serwisach internetowych; efekty wizualne nie są w stanie zrekomensować strat związanych z wygodą użytkownika.

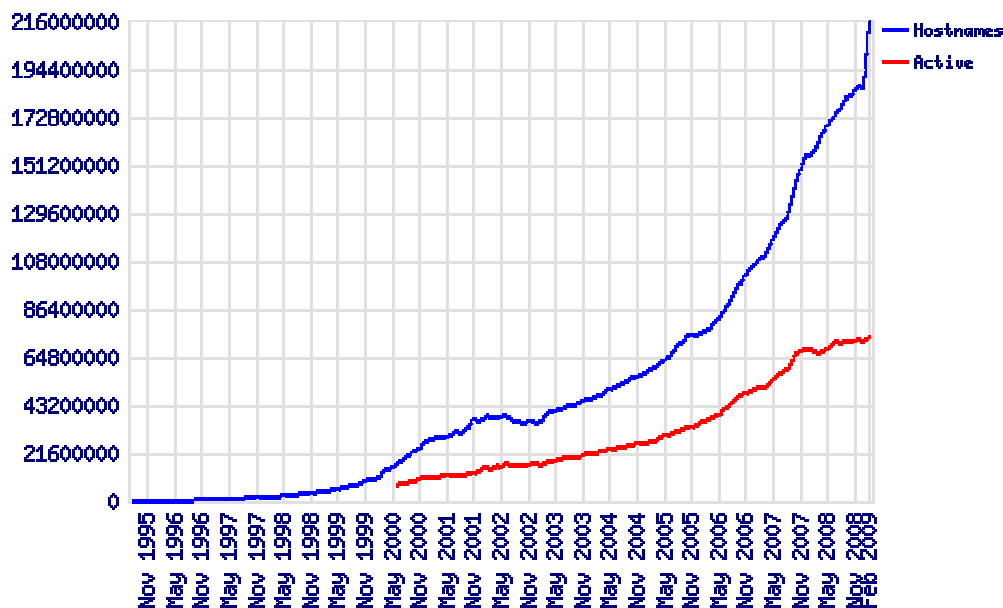
Obecnie użytkownicy zdecydowanie bardziej krytycznie patrzą na zagadnienie wydajności serwisów internetowych. Żądana przez nich informacja powinna być dostarczona w sposób możliwie jak najszybszy. Mnogość serwisów internetowych dostępnych w sieci skutkuje tym, iż w przypadku jakichkolwiek opóźnień użytkownik rezygnuje z przeglądania zasobów danego serwisu i kieruje się do innego, najczęściej konkurencyjnego. Społeczność internetowa stała się zdecydowanie bardziej wymagająca i niecierpliwa, została przyzwyczajona do bardzo szybkiej realizacji procesu żądanie-odpowieź, a większość operacji wykonuje właściwie intuicyjnie [zob. Kłopotek 2001].

Wydajność serwisów internetowych ma również znaczenie w kontekście mnogości urządzeń, z których mogą korzystać użytkownicy przeglądający ich zasoby. Jeszcze kilka lat temu do urządzeń tych zaliczały się praktycznie tylko komputery. Dziś Internauta może korzystać nie tylko z komputera ale i m.in. z telefonu komórkowego. Specjalny protokół o nazwie WAP, który został stworzony z myślą o przeglądaniu zasobów Internetu przez telefon komórkowy, został dziś praktycznie wyparty. Nowoczesne telefony komórkowe są w stanie przeglądać Internet w sposób zbliżony do tego znanego z komputerów. Posiadają wbudowane przeglądarki internetowe, które umożliwiają m.in. wyświetlanie grafiki. Era tworzenia osobnych wersji serwisów internetowych przeznaczonych na różne urządzenia przechodzi powoli do lamusa przez co oczywistym staje się fakt, iż przy ich tworzeniu należy uwzględniać możliwości różnych urządzeń, które potencjalnie mogą zostać użyte do przeglądania ich zawartości. Nieprzebranie tej zasady może skutkować w zmniejszeniu dostępności danego serwisu internetowego.

Poziom wydajności serwisów internetowych ma również znaczenie dla organizacji. Im większe obciążenie generowane przez serwis tym konieczniejsze będą wkłady finansowe w celu jego funkcjonowania. Zależność ta jest wprost proporcjonalna. Oznacza to, iż wraz ze wzrostem obciążenia serwera wzrasta wartość środków finansowych, jakie należy wyłożyć na jego utrzymanie. Oczywistym jest, iż każdy właściciel serwisu internetowego chce przeznaczyć na ten cel jak najmniejszą sumę pieniędzy nieprzekraczającą jego finansowych możliwości. Odpowiednia optymalizacja wydajności serwisu internetowego jest więc jedną z podstawowych kwestii, o którą należy zadbać podczas jego tworzenia.

Odpowiednia wydajność serwisów internetowych posiada także ogromne znaczenie w skali globalnej. Z dnia na dzień w sieci Internet powstają coraz to nowe serwisy oraz małe strony, które nie zawsze są stworzone w prawidłowy sposób. Sumaryczna liczba serwisów internetowych w styczniu 2009 roku wyniosła blisko 216 bilionów (zob. rys. 1),

a miesięcznie odnotowuje się powstawanie więcej niż 30 milionów nowych stron [zob. Netcraft 2009].



Rysunek 1. Wykres sumarycznej liczby serwisów internetowych  
Źródło: Netcraft 2009

Przyrost serwisów internetowych, których słabą stroną jest wydajność, może w stosunkowo krótkim czasie doprowadzić do tego, iż cała sieć Internet zacznie funkcjonować wolniej. Ucierpią na tym również i te serwisy, których programiści poświęcili sporo czasu i wysiłku na uzyskanie dobrej wydajności. W tym kontekście dbanie o odpowiednią wydajność serwisów internetowych leży w interesie każdego, kto taki serwis posiada lub chce posiadać.

## 1.2. Czynniki wpływające na wydajność serwisów internetowych

Wśród czynników wpływających na wydajność serwisów internetowych wyróżnić można dwie grupy: zaplecze techniczne oraz jakość oprogramowania. Pierwsza grupa to cały fizyczny sprzęt komputerowy, na którym oparty jest serwis internetowy, natomiast druga grupa to sposób, w jaki został zaprogramowany ów serwis. W dalszej części pracy omówione zostaną zagadnienia związane z wydajnością serwisów internetowych patrząc z perspektywy programistycznej.

Większość obecnie powstających serwisów internetowych jest kombinacją technologii działających po stronie klienta (przeglądarki internetowej) i serwera (maszyny, na której znajduje się serwis internetowy) z równoczesnym wykorzystaniem różnego rodzaju baz

danych. Każdy z tych elementów z osobna posiada wpływ na ogólną wydajność całego serwisu internetowego, jednak dopiero odpowiednie ich dopasowanie pozwala na uzyskanie maksymalnej lub chociaż zadowalającej wydajności.

Skupiając się jedynie na aspekcie programistycznym podczas optymalizacji wydajności serwisu internetowego nie zawsze jest się w stanie osiągnąć zamierzony efekt. Zdarza się bowiem, iż nawet najlepiej zaprojektowana baza danych i najlepiej napisany kod nie są w stanie sprostać wymaganiom, jakie są stawiane przed aplikacją. Wówczas jedynym rozwiązaniem pozostaje powiększenie zaplecza technicznego o kolejną maszynę serwerową. Jednak bez odpowiedniego przygotowania oprogramowania nie można jednoznacznie stwierdzić, iż jedynym rozwiązaniem będzie inwestycja w nowy serwer. Możliwe jest wówczas, iż zakup kolejnego serwera będzie jedynie środkiem doraźnym, pomagającym chwilowo. Problem wydajności oprogramowania będzie istniał dalej lecz będzie mniej dokuczliwy z racji rozbudowanego zaplecza technicznego. Skupiając się jednak na odpowiedniej optymalizacji wydajności serwisów internetowych od strony programistycznej można mieć pewność, iż uzyskano najlepszą możliwą wydajność kodu jako całości.

Najistotniejszym czynnikiem, na który warto zwrócić uwagę podczas projektowania serwisu internetowego jest odpowiednia baza danych, bowiem to od niej w głównej mierze zależy wydajność całego systemu. Wydawać by się mogło, iż jest to zagadnienie sprowadzające się głównie do odpowiedniego schematu tabel w bazie danych. Faktycznie, właściwe zaprojektowanie bazy danych to kluczowa sprawa jeżeli chodzi o jej wydajność. Nawet najdroższy system bazodanowy nie będzie działał sprawnie i wydajnie jeżeli nie zostanie wykorzystany w należyty sposób. Podczas projektowania schematu bazy danych należy w szczególności zwrócić uwagę na zastosowanie indeksów założonych na odpowiednie kolumny. Należy przy tym zachować umiar, ponieważ zbyt duża liczba indeksów w bazie danych może wpłynąć negatywnie na jej wydajność. Podczas projektowania bazy danych oraz zakładania indeksów należy również mieć na uwadze zapytania, które będą do niej kierowane. Zazwyczaj jest tak, iż zapytania można zrealizować na kilka sposobów. Należy więc wybrać taki wariant, który będzie najbardziej optymalny.

Drugim, często pomijanym, elementem związanym z systemem bazy danych jest wybór odpowiedniego serwera baz danych. Element ten jest ignorowany, gdyż aktualnie zdecydowanie najpopularniejszym systemem do zastosowań w Internecie jest baza danych MySQL. Jest ona powszechnie dostępna na większości serwerów w związku z czym jej wybór jest właściwie automatyczny. Jednak zawsze warto się zastanowić czego się oczekuje

od bazy danych oraz czy lepiej skorzystać z darmowego czy komercyjnego rozwiązania. Na rynku aktualnie dostępnych jest kilka różnych systemów baz danych, z czego na szczególną uwagę zasługują:

- MySQL,
- PostgreSQL,
- Oracle,
- Microsoft SQL Server.

MySQL jest popularną relacyjną bazą danych dystrybuowaną na zasadach *open source*, czyli jako tzw. *otwarte/wolne oprogramowanie*. Sukces bazy danych MySQL został osiągnięty głównie dzięki jej niezawodności, wydajności i dużym możliwościom. Firma Sun Microsystems, która zajmuje się rozwijaniem MySQL, szacuje, że jej baza danych jest wykorzystywana w ponad 6 milionach systemów informatycznych [zob. Dyer 2005 s.23]. Do najbardziej znanych serwisów internetowych korzystających z MySQL można z pewnością zaliczyć m.in. Yahoo!, Google, YouTube oraz Wikipedię [zob. MySQL informacje 2009]. W zależności od użytego mechanizmu składowania danych MySQL umożliwia m.in. obsługę więzów integralności oraz wyszukiwania pełno tekstowego. Oprócz wersji darmowej istnieje również komercyjna wersja bazy danych MySQL nazwana Enterprise, która odróżnia się od swego darmowego odpowiednika m.in. tym, iż zawiera dużo bardziej rozbudowane oprogramowanie służące do monitorowania pracy serwera bazy danych.

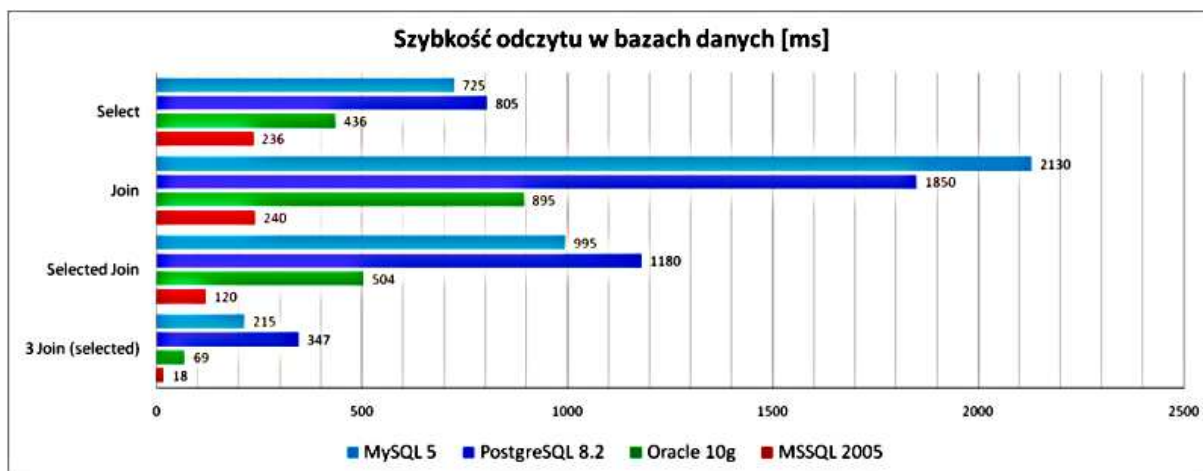
PostgreSQL to darmowa obiektowo-relacyjna baza danych, która jest stale rozwijana i usprawniana od ponad 15 lat. Baza ta posiada wersje dla systemów operacyjnych z rodziny Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64), a także Windows. Pozwala na stosowanie rozmaitych typów danych (włącznie z tworzeniem własnych typów) włączając w to również takie, które potrafią przechowywać dużą ilość danych binarnych (możliwość zapisu grafiki, dźwięków oraz sekwencji video). Prócz darmowej wersji istnieje także PostgreSQL w wersji komercyjnej, który charakteryzuje się m.in. możliwością równoczesnej pracy kilku wersji serwera bazy danych (Multi-Version Concurrency Control – MVCC), mechanizmem asynchronicznej replikacji oraz bardziej zaawansowanym optymalizatorem wykonywania zapytań. PostgreSQL może poszczycić się 5-krotnym zdobyciem tytułu najlepszego systemu bazodanowego przyznawanego przez The Linux Journal Editors' Choice Award [zob. PostgreSQL informacje 2009].

Baza danych Oracle jest komercyjnym systemem bazodanowym przeznaczonym do profesjonalnego zastosowania. Jej cechami charakterystycznymi są ogromne możliwości, elastyczność oraz mnogość narzędzi dostarczanych wraz z bazą danych, które są niezastąpione w codziennej pracy administratora baz danych. Najnowsza (piąta) wersja bazy danych Oracle, oznaczona symbolem 10g, została gruntownie przebudowana i zawiera wiele użytecznych narzędzi jak np. harmonogram zadań [zob. Urman 2007 s.17]. „Podstawowa wersja serwera bazodanowego Oracle (Standard Edition) jest przeznaczona dla systemów średniej i małej wielkości; jest ona kosztowo dostępna dla każdej firmy i organizacji tego sektora. Dla najbardziej wymagających użytkowników oferowana jest wersja Enterprise Edition, która oprócz funkcjonalności podstawowej zaopatrzona jest w wiele opcji dodatkowych. Odmianą tej wersji jest Personal Edition – baza danych dla pojedynczego użytkownika. Dla urzędów bezprzewodowych oferowana jest wersja Oracle Lite. Ułatwia ona tworzenie i wdrażanie aplikacji mobilnych oraz zarządzanie nimi. Baza ta obsługuje transakcje, zapewnia pełną integralność danych, a także umożliwia przetwarzanie zapytań na bieżąco” [SQL Server informacje 2009].

Microsoft SQL Server to, obok Oracle, jedna z najpopularniejszych komercyjnych baz danych, która określana jest jako „kompleksowa platforma do zarządzania danymi i analizy biznesowej zapewniająca skalowalność, magazynowanie danych, zaawansowaną analizę i najwyższy poziom bezpieczeństwa na potrzeby uruchamiania krytycznych dla przedsiębiorstwa aplikacji biznesowych” [SQL Server informacje]. Jest wszechstronną platformą bazodanową umożliwiającą bezpieczne przechowywanie danych oraz tworzenie wysoce wydajnych aplikacji korzystających z bazy danych. Istotną cechą Microsoft SQL Server jest możliwość prostej integracji tej bazy danych ze środowiskiem programistycznym Microsoft Visual Studio oraz z pakietem biurowym Microsoft Office [zob. Bagui 2007 s.15]. Najnowszą wersję systemu charakteryzują m.in. równoległe partycjonowanie tabel, korzystanie z udoskonaleń funkcji klastrowania, stała i przewidywalna wydajność zapytań dzięki sterowaniu priorytetami obciążeń pracą i alokacją zasobów za pomocą narzędzia Resource Governor, a także dostosowywanie, monitorowanie i rozwiązywanie występujących problemów dla wszystkich instancji SQL Server w całym przedsiębiorstwie dzięki funkcji gromadzenia danych o wydajności (Performance Data Collector) [zob. SQL Server informacje].

Pomiędzy tymi systemami istnieje szereg różnic w możliwościach jakie one oferują. Dwa pierwsze systemy (czyli MySQL oraz PostgreSQL) są rozwiązaniami darmowymi i znajdują

zastosowanie w większości serwisów internetowych. Kolejne natomiast (czyli Oracle oraz Microsoft SQL Server) należą do grupy komercyjnych baz danych i posiadają bardzo bogaty zestaw narzędzi wspomagających codzienną pracę i konserwację. Różnice widać również w wydajności tych systemów (zob. rys. 2) m.in. pod względem prędkości odczytu danych [zob. Lesiak 2008].



Rysunek 2. Szybkość odczytu danych w MySQL, PostgreSQL, Oracle i Microsoft SQL Server  
Źródło: [Lesiak 2008]

Prócz wyboru odpowiedniego systemu baz danych czasem, w zależności od tegoż systemu, również należy wybrać odpowiedni mechanizm składowania danych. W przypadku bazy danych MySQL można wyróżnić trzy najważniejsze mechanizmy składowania danych:

- MyISAM,
- InnoDB,
- MEMORY (HEAP).

MyISAM jest domyślnym mechanizmem składowania danych wykorzystywanym przez MySQL. Każda tabela w bazie danych przechowywana jest za pomocą trzech plików, których nazwy są odpowiednikami nazwy tabeli, natomiast rozszerzenia związane są z funkcjami, które te pliki spełniają. Plik \*.frm przechowuje strukturę tabeli, \*.MYD dane, natomiast \*.MYI informacje o indeksach [zob. MySQL MyISAM 2009]. Niewątpliwą zaletą MyISAM jest duża szybkość wykonywanych operacji pobierania oraz modyfikacji danych, która została uzyskana kosztem praktycznie zerowej dbałości o utrzymanie więzów integralności. Inną istotną zaletą tego mechanizmu składowania danych jest możliwość wyszukiwania pełnotekstowego. Do wad MyISAM (prócz wspomnianego wcześniej ignorowania więzów integralności) zaliczyć można z pewnością brak obsługi transakcji, które wydają się być wręcz konieczne w bardziej zaawansowanych serwisach internetowych.

InnoDB jest transakcyjnym mechanizmem składowania danych, który ponadto charakteryzuje się wysokim poziomem dbałości o ochronę danych użytkownika. Istotną zaletą tego mechanizmu jest możliwość blokowania danych na poziomie wiersza, a także (wzorowane na rozwiązaniach Oracle) stosowanie odczytu danych zapobiegającego występowaniu zakleszczeń, co jest bardzo istotne przy wielu konkurencyjnych żądaniach kierowanych do bazy danych. W odróżnieniu od MyISAM, InnoDB oferuje m.in. utrzymywanie więzów integralności między danymi, natomiast nie wspiera wyszukiwania pełnotekstowego. Pod względem wydajnościowym jest również znacznie wolniejszym mechanizmem aniżeli MyISAM [zob. MySQL InnoDB 2009].

Główną cechą mechanizmu składowania danych MEMORY jest przechowywanie danych w pamięci, co ma bezpośredni wpływ na wręcz błyskawiczną szybkość operacji wykonywanych na danych. Negatywnym skutkiem takiego przechowywania danych jest fakt, iż są one tracone podczas każdego restartu serwera bazy danych. Innym ograniczeniem tego mechanizmu składowania danych jest brak możliwości tworzenia kolumn typu BLOB oraz TEXT. Głównym przeznaczeniem tabel MEMORY są tabele tymczasowe, w przypadku których bardzo istotnym aspektem jest prędkość odczytu/modyfikacji danych niż, natomiast ich trwałość ma znaczenie marginalne [zob. MySQL MEMORY 2009].

Na wydajność serwisu internetowego ma również wpływ technologia zastosowana do napisania całego serca serwisu czyli kodu. Również i w tym wypadku wybór nie jest prosty. Do wyboru są takie języki programowania jak m.in.:

- PHP,
- ASP,
- JSP,
- Ruby on Rails.

Zdecydowanie najpopularniejszym językiem służącym do tworzenia serwisów internetowych jest PHP. Jego popularność to w dużej mierze zasługa powszechnej dostępności oraz stosunkowo dużej prostoty programowania aplikacji. „Jest językiem programowania, osadzonym wewnątrz kodu HTML, służącym do tworzenia skryptów wykonywalnych po stronie serwera. Ułatwia on pisanie aktywnych stron WWW odwołujących się do baz danych, oferując bogaty zbiór funkcji i metod do większości systemów SQL” [Kłopotek 2001 s.137].

Przewagę nad nim posiadają ASP oraz JSP (będące notabene językami konkurencyjnymi dla siebie wzajemnie) z racji tego, iż są to języki kompilowane. Skompilowany kod wynikowy jest bardziej wydajny z racji tego, iż jest od razu przystosowany do wykonania przez serwer. W praktyce jednak wykorzystanie tych technologii wiąże się z większymi nakładami finansowymi związanymi z przygotowaniem odpowiedniego środowiska; widać to szczególnie dobrze w przypadku ASP z racji ścisłej zależności z produktami firmy Microsoft oraz gorszej współpracy z serwerem Apache na rzecz Internet Information Server (IIS) [zob. Kłopotek 2001].

Ruby on Rails nie jest językiem programowania w dosłownym tego słowa znaczeniu, a jedynie *frameworkiem*, czyli zbiorem komponentów ściśle ze sobą powiązanych i mających na celu przyspieszenie tworzenia serwisów internetowych. Jest to stosunkowo nowa technologia w związku z czym nie zdobyła jeszcze takiej popularności jak wymienione wcześniej.

Podobnie jak w przypadku wyboru odpowiedniego systemu bazy danych, tak i wybór odpowiedniego języka programowania nie jest prostą decyzją. Przede wszystkim należy jasno określić wymagania, jakie musi spełniać język, a w szczególności to, czy język powinien być kompilowany czy interpretowany. Wybór języka kompilowanego może przynieść spore korzyści jeśli chodzi o wydajność tworzonego kodu. Równocześnie może się okazać, że sam interpreter języka jest odpowiedzialny za jej nieprawidłowe funkcjonowanie [zob. Argreich 2003 s.865]. Ponadto warto zastanowić się na jakiego typu serwerach będzie działało tworzone oprogramowanie. Jeżeli nie ma się co do tego pewności lepiej zastosować uniwersalną technologię, która sprawdzi się w każdych warunkach.

Odpowiednie połączenie systemu bazy danych oraz języka programowania jest kluczem do uzyskania bardzo dobrego punktu wyjścia do stworzenia wydajnego serwisu internetowego. Aktualnie zdecydowanie najczęściej stosowanym rozwiązaniem (nie tylko przez małe i średnie serwisy internetowe) jest połączenie bazy danych MySQL z językiem PHP lecz znaczący wzrost wydajności bazy danych PostgreSQL w wersji 8.3 [zob. PostgreSQL 8.3 zmiany 2009] może doprowadzić do stopniowego tracenia pozycji przez MySQL.

Rozpatrując czynniki wpływające na wydajność serwisów internetowych należy spojrzeć nie tylko na czynniki decydujące o wydajności po stronie serwera, ale i te, które dotyczą bezpośrednio użytkownika, czyli klienta. Chodzi tutaj o odpowiednie przygotowanie prezentowanych dokumentów. W przypadku przeciążenia serwisu internetowego elementami

graficznymi lub animacjami odczuwalny czas wczytywania strony będzie znacznie wydłużony. Nowoczesne przeglądarki internetowe są w stanie gromadzić niektóre elementy serwisu internetowego w pamięci podręcznej jednak nie powinno to być usprawiedliwieniem dla tworzenia przeciążonych witryn. Przy projektowaniu szaty graficznej serwisów internetowych podstawową sprawą powinno być przestrzeganie zasadny minimalizmu koniecznego. Jeżeli z niektórych elementów można zrezygnować bez uszczerbku na funkcjonalności serwisu to należy to uczynić. Ważnym jest aby znaleźć swego rodzaju złoty środek pomiędzy efektami wizualnymi, a prędkością działania serwisu internetowego. Prawdą jest bowiem, iż nawet najlepiej zaprogramowany serwis od strony serwera nie będzie w pełni wydajny, jeżeli zostanie zaopatrzone w zbyt ciężki interfejs.

### **1.3. Korzyści wynikające z optymalizacji wydajności serwisów internetowych**

Korzyści płynące z odpowiedniej optymalizacji wydajności serwisów internetowych należy przede wszystkim rozpatrywać patrząc z punktu widzenia organizacji. Głównym zadaniem większości serwisów internetowych jest generowanie zysku. Serwisy internetowe, których utrzymanie wymaga ciągłych wkładów finansowych stają się po prostu nierentowne, co najczęściej powoduje ich zamknięcie. Zarabianie przez serwis chociażby tylko na własne utrzymanie jest więc bardzo ważną kwestią, z którą muszą uporać się jego opiekunowie.

Omawiając korzyści wynikające z optymalizacji wydajności serwisów internetowych należy również wspomnieć o naczelnej zasadzie, która głosi, iż „wstępna optymalizacja jest źródłem wszelkiego zła”. Tezę tą („Preoptimization is the root of all evils”) postawił Donald Knuth w swojej książce pt. „The Art Of Computer Programming” [zob. Argreich 2003 s.865].

Podstawową korzyścią, do której dąży się podczas optymalizacji wydajności serwisów internetowych, jest zmniejszenie obciążenia serwera. Jest to kluczowy element na drodze do poprawy prędkości funkcjonowania serwisu internetowego. Każdy, nawet najmocniejszy, serwer posiada pewną skończoną moc, którą może wykorzystać do pracy. Uruchomienie na nim oprogramowania, które będzie w sposób znaczny wykorzystywało jego zasoby, może w krótkim czasie doprowadzić do tego, iż przestanie on być wystarczający. Serwis internetowy zacznie funkcjonować w sposób ociężały, a poszczególne podstrony będą wczytywały się bardzo powoli. Skutkiem takiego przeciążenia serwera może być również jego automatyczne restartowanie. Bardzo często jest tak, iż na jednym serwerze funkcjonuje kilka serwisów internetowych. W momencie, gdy któryś z nich zacznie wykorzystywać sporą ilość jego

zasobów, skutki mogą odczuć również pozostałe serwisy internetowe. W przypadku posiadania własnego serwera (np. dedykowanego) można szybko zareagować na ten problem przenosząc pozostałe swoje serwisy na inną maszynę serwerową. Jednak w sytuacji, gdy dany serwis internetowy korzysta z serwera współdzielonego, skutki mogą okazać się bardziej dotkliwe. Firma, u której wynajmowane jest miejsce na serwerze, może wówczas wypowiedzieć umowę jako argument podając działanie, które zakłóca funkcjonowanie serwisów internetowych innych użytkowników. Oznaczać to może zastój w prowadzeniu własnego serwisu internetowego oraz problemy ze znalezieniem nowego partnera, u którego wykupione zostanie miejsce na serwerze. Dbając więc o odpowiednią optymalizację wydajności serwisu internetowego można zyskać jego stabilizację, a co za tym idzie uniknąć również pewnych zbędnych, ale w danej sytuacji koniecznych, wydatków finansowych.

Optymalizacja wydajności kodu serwisu internetowego po stronie klienta polegająca m.in. na ograniczeniu rozmiaru przesyłanych informacji przyniesie pozytywny skutek w redukcji transferu, który generuje owy serwis. Współdzielone konta serwerowe mają z reguły nałożone pewne ograniczenia dotyczące maksymalnego miesięcznego oraz rocznego transferu danych. W przypadku różnych usługodawców wartości te mogą być diametralnie różne, jednak zazwyczaj oscylują w granicach 50GB na miesiąc i 600GB w skali roku. Liczby te zdają się być duże i w spokoju powinny sprostać ruchowi nawet na większych serwisach, jednak w rzeczywistości zdarza się, iż stają się niewystarczalne. Podobnie problem limitu transferu danych wygląda również na serwerach dedykowanych oraz na własnych maszynach serwerowych. Wówczas problem mogą stanowić ograniczenia nałożone przez dostawcę Internetu. Zadbanie o odpowiednią optymalizację przesyłanych do klienta danych może więc znacząco wpłynąć na koszty związane z utrzymaniem serwisu internetowego. Mniejsze zużycie transferu oznacza przecież możliwość obsłużenia większej liczby użytkowników przy takim samym limicie transferu. Następstwem tego jest brak konieczności dokonywania inwestycji polegającej na dokupywaniu dodatkowych GB transferu.

Niewątpliwą zaletą wydajnych serwisów internetowych jest również to, iż znacznie łatwiej jest im przyciągnąć i, co ważniejsze, zatrzymać Internautę. Jak już wspomniano wcześniej obecni użytkownicy Internetu oczekują od serwisów internetowych praktycznie natychmiastowej reakcji na wydawane przez nich polecenia. Mnogość serwisów o podobnej tematyce powoduje, iż w przypadku zbyt wolno działającego serwisu internetowego *A* użytkownik rezygnuje z dalszego jego przeglądania i postanawia odwiedzić konkurencyjny serwis *B*, przez co serwis *A* traci jednego użytkownika na rzecz konkurencji. Takie

zachowania użytkowników w skali miesięcznej czy wręcz rocznej są w stanie w bardzo dotkliwy sposób wpłynąć negatywnie na statystyki odwiedzalności serwisu internetowego. Bezpośrednio z liczbą użytkowników wiążą się następujące kwestie:

- popularność serwisu internetowego,
- liczba reklam zamieszczanych w serwisie internetowym.

Popularność, jaką posiada serwis internetowy, zależy bezpośrednio od liczby jego stałych użytkowników. Nie sztuką bowiem jest przyciągnąć użytkowników na chwilę lecz sztuką jest ich zatrzymanie i nakłonienie do kolejnych wizyt. Spora część użytkowników trafia do serwisu internetowego poprzez różnego rodzaju wyszukiwarki. Im wyższa pozycja w wyszukiwarkach tym większe prawdopodobieństwo, iż potencjalny użytkownik skorzysta właśnie z danego serwisu internetowego. Szczegółowy opis działań związanych z pozycjonowaniem stron w wyszukiwarkach wybiega poza temat tej pracy, jednak zauważyć należy, iż jednym z czynników wpływających na pozycję strony w wyszukiwarkach jest właśnie jej popularność. Popularność ta jest bezpośrednio powiązana z liczbą odwiedzin, jakie odnotowuje dany serwis. Wpisując w wyszukiwarce Google frazę „game online” w rezultacie otrzymuje się listę serwisów udostępniających na swoich stronach gry online (zob. rys. 3).



Rysunek 3. Rezultat wyszukiwania frazy „game online”

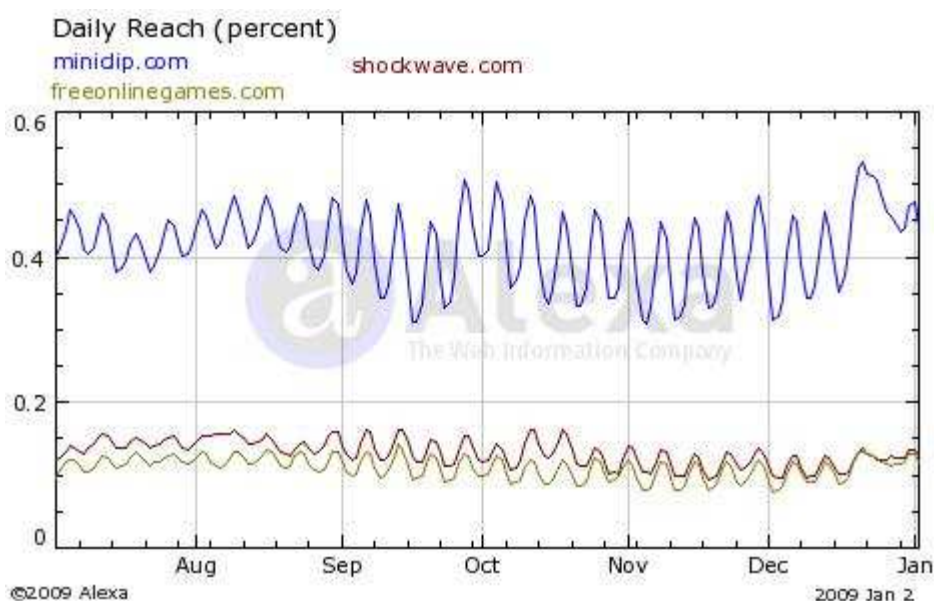
Źródło: Opracowanie własne

Trzy pierwsze miejsca na uzyskanej liście wyników wyszukiwania zajmują strony:

- [www.miniclip.com](#),
- [www.shockwave.com](#),

- [www.freeonlinegames.com](http://www.freeonlinegames.com).

Korzystając następnie z usług serwisu Alexa ([www.alexa.com](http://www.alexa.com)) można sprawdzić jaką odwiedzalność odnotowały te serwisy w przeciągu ostatnich 6 miesięcy (zob. rys. 4).



Rysunek 4. Wykres popularności stron wygenerowany dzięki serwisowi Alexa  
Źródło: [www.alexa.com](http://www.alexa.com)

Zauważyć można, iż ustawienie serwisów w wynikach wyszukiwania jest takie samo, jak ich kolejność pod względem liczby odwiedzin. Reguła ta oczywiście nie zawsze się sprawdza gdyż na pozycję stron w wyszukiwarkach mają wpływ jeszcze inne czynniki. Warto jednak mieć na uwadze, iż popularność serwisu internetowego jest bezpośrednio powiązana z liczbą jego odwiedzin.

Poziom odwiedzalności serwisu internetowego można również przełożyć w bardzo prosty sposób na wartość dochodów generowanych przez ten serwis. Głównym źródłem dochodów serwisów są reklamy. Mogą to być reklamy różnego rodzaju m.in.:

- reklamy kontekstowe,
- linki reklamowe,
- banery reklamowe.

Większość serwisów internetowych korzysta z usług zewnętrznych firm, najczęściej agencji reklamowych, które są pośrednikiem między serwisem, a reklamodawcą. Firmy te zbierają inne firmy czy instytucje, które zainteresowane są reklamowaniem swoich usług w sieci Internet. W wyborze serwisu internetowego, w którym emitowane będą dane reklamy, ogromny wpływ ma liczba jego odwiedzin. Im ta wartość wyższa, tym większa szansa na

umieszczenie reklam właśnie w tym serwisie oraz tym wyższe stawki, które można wynegocjować za pojedyncze odsłony reklamy. Jak wspomniano wcześniej reklamy są zazwyczaj głównym źródłem dochodów generowanych przez serwisy internetowe. W związku z tym liczba użytkowników oraz ich odwiedzalność, będące następstwem odpowiedniej optymalizacji wydajności serwisów internetowych, wywiera mocny wpływ na poziom rentowności biznesu.

Oprócz korzyści czysto materialnych, związanych z kwestiami finansowymi, optymalizacja wydajności serwisów internetowych może również przynieść korzyści na polu marketingowym. Swego czasu hasłem reklamowym popularnej obecnie przeglądarki internetowej Opera było „Fastest browser” czyli „Najszybsza przeglądarka”. Autorzy aplikacji szczycili się tym, iż udało im się uzyskać większą wydajność, a co za tym idzie także większą przyjemność podczas korzystania z niej. Przez wiele lat Opera faktycznie uchodziła za najszybsze rozwiązanie na tle konkurencji. Dbając o zoptymalizowanie wydajności serwisu internetowego można z czystym sumieniem pozwolić sobie na używanie podobnych sloganów reklamowych. Odpowiednia kampania marketingowa, poparta dodatkowo faktycznymi osiągnięciami w dziedzinie wydajności, może być dobrym pomysłem na przyciągnięcie nowych użytkowników.

Podsumowując korzyści płynące z odpowiedniej wydajności serwisów internetowych można stwierdzić, iż są one bezpośrednio związane z niektórymi celami, które stawiane są przed serwisami internetowymi [zob. Ziemia 2005 s.48]:

- korzyści/cele wymierne,
  - generowanie przychodów z handlu elektronicznego,
  - generowanie przychodów ze sprzedaży będącej rezultatem działania serwisu,
  - zwiększenie udziału w rynku,
  - realizacja serwisu zgodnie z projektem (czas i koszty, bezbłądność działania),
- korzyści/cele niewymierne,
  - dotarcie do nowych rynków,
  - wykreowanie marki,
  - osiągnięcie pozycji lidera na rynku,
  - zwiększenie dostępności do organizacji,
  - pozyskanie nowych i utrzymanie dotychczasowych klientów,

- uzyskanie przewagi konkurencyjnej,
- podniesienie prestiżu organizacji.

Uzyskanie odpowiedniej wydajności serwisów internetowych jest możliwe tylko w przypadku właściwej optymalizacji ich poszczególnych warstw. W następnym rozdziale omówione zostaną główne warstwy optymalizacji wydajności serwisów internetowych. Poruszona zostanie problematyka optymalizacji warstwy prezentacji danych, logiki aplikacji, a także bazy danych.

## ROZDZIAŁ 2. WARSTWY OPTIMALIZACJI WYDAJNOŚCI SERWISÓW INTERNETOWYCH

Optymalizację wydajności serwisu internetowego można przeprowadzić w trzech odrębnych warstwach, z których każda odpowiedzialna jest za wydajność różnych składowych tworzących serwis internetowy. Wśród tych warstw można wyróżnić: warstwę prezentacji danych, warstwę logiki aplikacji oraz warstwę bazy danych.

Przeprowadzając optymalizację wydajności serwisu internetowego należy skupić się na wszystkich wymienionych warstwach, gdyż tylko wtedy osiągnięty rezultat będzie możliwie najwyższy. Odpowiednia ich współpraca przyniesie wymierne korzyści w postaci wzrostu szybkości działania serwisu oraz spadku obciążenia serwera, na którym on pracuje. To natomiast zaskutkuje zwiększeniem liczby klientów, którzy mogą zostać obsłużeni równocześnie w jednostce czasu.

W niniejszym rozdziale omówione zostaną wymienione warstwy optymalizacji wydajności serwisów internetowych, a także przedstawione zostaną sposoby umożliwiające dokonanie optymalizacji wydajności w każdej z tych warstw.

### 2.1. Warstwa prezentacji danych

Tworząc lub optymalizując serwis internetowy należy zająć się jego warstwą prezentacji. Co prawda realny wzrost wydajności, wynikający z optymalizacji tej warstwy, jest mniejszy niż w przypadku pozostałych obszarów, jednak również warto o nią zadbać, ponieważ to właśnie na niej skupia się uwaga użytkownika.

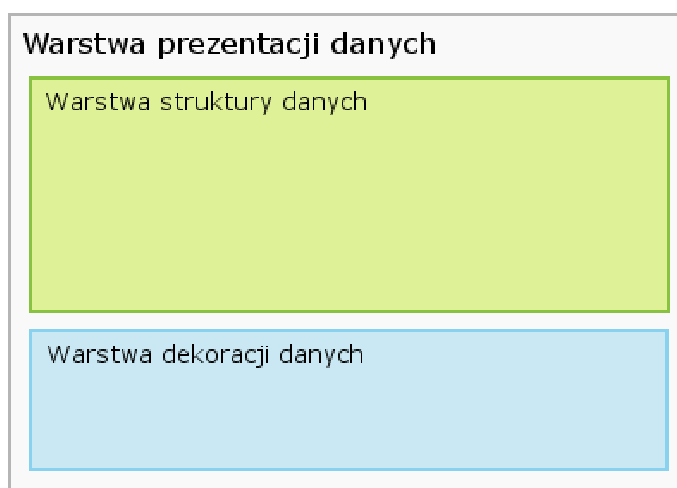
Skupiając się na optymalizacji warstwy prezentacji danych w serwisie internetowym, w pierwszej kolejności należy wyeliminować wszelkie style *inline*, czyli występujące bezpośrednio w formie atrybutu dla znaczników HTML. Kod zawierający deklaracje stylów osadzone w znacznikach HTML posiada kilka dość istotnych wad:

- blisko 80% kodu stanowią same deklaracje stylów, co znacząco zwiększa jego objętość,
- w przypadku konieczności naniesienia pewnych zmian w szablonie (np. dodanie koloru tła dla elementów listy) konieczna jest zmiana kodu w kilku miejscach,

- kod jest mniej przejrzysty.

Wady te można wyeliminować poprzez stosowanie arkuszy stylów CSS, w których to umieszczane są odpowiednie deklaracje. Dzięki temu kod HTML może zostać uproszczony, a znaczniki oczyszczone ze zbędnych atrybutów. Uzyskany wówczas kod warstwy prezentacji danych zostanie poprawnie rozdzielony na dwie dodatkowe podwarstwy (zob. rys. 5):

- warstwę struktury danych (kod HTML),
- warstwę dekoracji danych (kod CSS).



Rysunek 5. Warstwa prezentacji danych  
Źródło: Opracowanie własne

Współpraca obu podwarstw skutkuje w odpowiedniej wizualizacji danych. Rozdzielenie warstwy prezentacji na dwie podwarstwy daje możliwość szybkiej zmiany sposobu prezentacji danych poprzez modyfikację tylko i wyłącznie warstwy dekoracji danych.

Kolejnym krokiem w celu zoptymalizowania warstwy prezentacji danych, jest przeniesienie arkusza stylów, czyli warstwy dekoracji danych, poza dokumenty HTML. Arkusz ten może zostać później wczytany za pomocą znacznika polecenia `@import`. Dzięki wyizolowaniu warstwy dekoracji danych do osobnego pliku osiągnięte zostaną cele:

- rozmiar plików HTML zostanie zmniejszony o rozmiar arkusza stylów CSS, dzięki czemu pliki te będą wczytywane szybciej,
- możliwe będzie przechowywanie arkusza stylów CSS w *cache* przeglądarki internetowej, co również przyczyni się do zwiększenia prędkości wczytywania stron,

- wszystkie dokumenty HTML w serwisie będą korzystały z jednego arkusza stylów CSS, w związku z czym zmiany nanoszone w warstwie dekoracji danych będą automatycznie widoczne we wszystkich stronach serwisu internetowego.

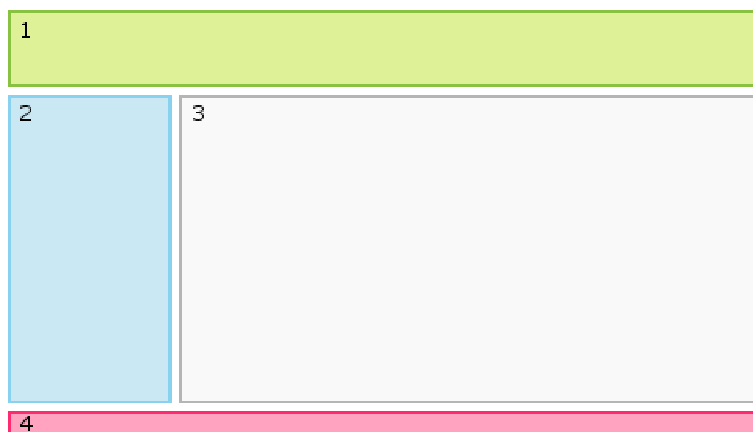
Korzyści wynikające z przechowywania arkusza stylów CSS w osobnym pliku są niezaprzeczalne. Prócz poprawy wydajności, zyskiwana jest również wygoda podczas przyszłej modernizacji serwisu internetowego. Właśnie z tych powodów warstwa dekoracji danych zawsze powinna być umieszczana w osobnym, niezależnym, pliku.

Tworząc szablon serwisu internetowego powinno się zrezygnować z układu tabelarycznego na rzecz układu zbudowanego w oparciu o znaczniki *div*. Znacznik *table*, jak sama nazwa wskazuje, służy do definiowania tabel, w związku z czym nie wydaje się więc rozsądnym stosowanie go w celu budowy szablonu. Prócz użycia nieodpowiedniego znacznika do nieodpowiedniego celu, tabele, jako siatka układu strony, pociągają za sobą kolejne konsekwencje [Osiołki 2008]:

- „mieszają treść i warstwę prezentacyjną. Strony są niepotrzebnie duże — użytkownicy muszą ściągać dane odpowiedzialne za prezentację z każdą stroną, którą odwiedzają, a przecież ruch w Sieci kosztuje,
- zmiana wyglądu strony jest niezwykle pracochłonna (i tym samym droga),
- również trudne (i kosztowne) jest utrzymanie spójnego wyglądu stron w całej witrynie,
- strony oparte o tabelki są o wiele trudniejsze w odbiorze dla osób niepełnosprawnych albo korzystających z Sieci za pomocą urządzeń przenośnych — telefonów komórkowych czy organizatorów (PDA).”

Najczęściej spotykanym układem elementów serwisu internetowego jest szablon składający się z 4 komponentów (zob. rys. 6):

1. nagłówek wraz z logo,
2. menu nawigacyjne,
3. treść właściwa,
4. stopka informacyjna.



Rysunek 6. Najpopularniejszy układ serwisu internetowego  
Źródło: Opracowanie własne

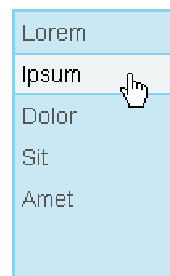
Wykonanie 4-komponentowego szablonu z wykorzystaniem tabel sprowadza się do utworzenia tabeli złożonej z 3 wierszy, z czego 2. Wiersz składa się z 2 kolumn. Definiując odpowiedni arkusz stylów można wyświetlić tabelę w postaci siatki dla szablonu serwisu internetowego. Początkowo kod takiego arkusza stylów nie będzie zbyt skomplikowany ani rozbudowany, natomiast w kodzie HTML wystąpi sporo znaczników (co najmniej *table*, *tr* oraz *td*), które są konieczne do poprawnej interpretacji szablonu przez przeglądarki internetowe, lecz tak naprawdę nie są konieczne do budowy szablonu o takiej strukturze. Modyfikacja szablonu, polegająca np. na zmianie układu komponentów, pociąga za sobą konieczność gruntownej przebudowy struktury tabeli. Ponadto występuje również inny problem. Zgodnie z zasadami semantyki menu powinno być zdefiniowane jako lista *ul*. W przypadku układu tabelarycznego koniecznym więc okazuje się stworzenie dodatkowego elementu w komórce nr 2, co pociąga za sobą kolejne komplikacje szablonu.

Przekształcenie szablonu na kod z użyciem znaczników *div* pozwoli na uzyskanie bardziej czytelnego i skalowalnego dokumentu. Kod zdecydowanie zyska na prostocie i przejrzystości; wystarczy jedynie jeszcze utworzyć odpowiedni arkusz stylów, a cały szablon będzie stanowił doskonały punkt wyjścia do dalszej rozbudowy.

Używając właściwego kodu HTML, który określa strukturę (nie wygląd) treści, oraz opierając układ strony wyłącznie na CSS, można utrzymać treść strony niezależną od sposobu, w jaki jest ona prezentowana. Używanie współczesnych, standardowych technologii WWW pozwala zmniejszyć objętość plików tworzących strony, ponieważ użytkownicy nie będą musieli już ściągać danych prezentacyjnych z każdą odwiedzaną stroną. Arkusze stylów są przechowywane w pamięci podręcznej przeglądarki, więc nie będą ściągane za każdym

razem. Mniejsze pliki to mniej danych do ściągnięcia, czyli szybsze otwieranie się stron i niższe koszty utrzymania serwisu [zob. Osiołki 2008].

Ostatnim elementem warstwy prezentacji, na który warto zwrócić uwagę podczas jej optymalizacji, jest możliwość zastąpienia niektórych elementów graficznych definicjami CSS. Doskonałym przykładem tego typu optymalizacji jest stworzenie menu opartego na odpowiednio określonej liście *ul*. Dzięki temu można uzyskać proste menu o estetycznym wyglądzie, które w całości opiera się na arkuszu stylów CSS i nie zawiera ani jednego pliku graficznego (zob. rys. 7).



Rysunek 7. Menu zdefiniowane za pomocą CSS

Źródło: Opracowanie własne

Korzyści płynące z zastosowania tego rozwiązania są oczywiste:

- rozmiar całego kodu obsługującego menu to ok. 650B, natomiast w przypadku utworzenia menu graficznego rozmiar samej grafiki wyniósłby ok. 2kB, co należałoby jeszcze powiększyć o rozmiar niezbędnego kodu (ok. 1kB),
- modyfikacja zawartości menu nie wymaga użycia programu graficznego – wystarczy dokonać edycji szablonu,
- modyfikacja wyglądu menu również nie wymaga użycia programu graficznego – wystarczy dokonać edycji arkusza stylów CSS.

Odpowiednia i umiejętna optymalizacja warstwy prezentacji serwisu internetowego wpływa nie tylko na jej wydajność, ale również pozwala na jej prostsze i szybsze modyfikacje.

## 2.2. Warstwa logiki aplikacji

Optymalizacja wydajności warstwy logiki aplikacji ma na celu wykrycie oraz eliminację lub ograniczenie występowania wąskich gardeł w części serwisu internetowego działającej po

stronie serwera. Optymalizacja ta wymaga znajomości technik programowania w językach server-side takich, jak PHP czy ASP, oraz sprowadza się do bezpośredniej ingerencji w kod aplikacji obsługującej serwis internetowy. Faktyczny wpływ optymalizacji warstwy logiki może być zauważalny dopiero przy większym obciążeniu serwera, jednak oczywistym wydaje się fakt, iż warto o nią zadbać. Przedstawione przykłady działań optymalizacyjnych bazują na języku PHP.

Podstawową kwestią, na jaką powinno się zwrócić uwagę tworząc oprogramowanie serwisu internetowego, jest to, w jaki sposób tworzone są ciągi dynamiczne. Istnieje bowiem zasadnicza różnica pomiędzy ciągami objętymi w cudzysłowie, a ciągami zawartymi pomiędzy apostrofami. Ujęcie ciągu w cudzysłowie powoduje, iż jest on wcześniej parsowany przez interpreter języka serwerowego w poszukiwaniu zmiennych, pod które należy podstawić przypisane im wartości. Zdecydowanie lepszym rozwiązaniem jest tworzenie ciągów ujętych w apostrofy. W tym przypadku interpreter języka serwerowego wyświetla na ekranie ciąg w takiej formie, w jakiej został utworzony. Ewentualne zmienne występujące w ciągu zostaną wyświetlone w formie ich nazw, a nie przypisanych im wartości.

Sporo operacji w każdym serwisie internetowym realizowanych jest przy pomocy pętli. Pętle najczęściej wykorzystywane są do wyświetlania tablicy zawierającej rekordy pobrane wcześniej z bazy danych. Prędkość wykonania takowej operacji w dużej mierze zależy od tego, w jaki sposób napisano kod pętli.

#### Przykład 1

```
for ($i=0; $i<count($array); $++) {  
    // dalsze operacje  
}
```

Standardowe użycie pętli *for* (zob. przykł. 1) wydaje się być w pełni poprawnym, jednak posiada jedną, ale za to dość istotną, wadę, która wpływa ujemnie na czas jej wykonania. Przy każdej iteracji sprawdzany jest warunek wyjścia z pętli *\$i<count(\$array)*. Wiąże się to z każdorazowym obliczaniem rozmiaru tablicy *\$array*, co znacznie obniża wydajność kodu.

#### Przykład 2

```
for ($i=0, $count=count($array); $i<$count; $++) {  
    // dalsze operacje  
}
```

Wprowadzając do pętli dodatkową zmienną *\$count* (zob. przykł. 2), do której (podczas pierwszego przejścia pętli) zostanie przypisana wartość odpowiadająca rozmiarowi tablicy

*\$array*, skutecznie zwiększana jest wydajność pętli. Przy kolejnych iteracjach sprawdzany jest warunek wyjścia z pętli wykorzystując utworzoną wcześniej zmienną *\$count*.

Optymalizacja pętli nie sprowadza się jedynie do odpowiedniego konstruowania pętli *for*; wiąże się również z wyborem odpowiedniego rodzaju pętli dla konkretnego zbioru danych. Test [zob. PHPBench 2008] przeprowadzony na tablicy asocjacyjnej składającej się z 1000 elementów, posiadającej indeksy 24-bajtowe oraz wartości o długości 10000 znaków każdy, dowodzi, iż najbardziej optymalnym rozwiązaniem dla tego przypadku jest użycie pętli *for* (pomimo konieczności użycia dodatkowej funkcji *array\_keys()*).

Tabela 1. Porównanie wydajności pętli

Rodzaj pętli	Czas [ms]	%
<code>\$keys = array_keys(\$array); \$size = count(\$keys); for (\$i=0; \$i&lt;\$size; \$i++) \$tmp[] = &amp;\$array[\$keys[\$i]];</code>	2	100
<code>\$keys = array_keys(\$array); \$count = count(\$keys); for (\$i=0; \$i&lt;\$size; \$i++) \$tmp[] = &amp;\$array[\$keys[\$i]];</code>	2	140
<code>foreach(\$array as \$key[] =&gt; \$val[]);</code>	3	192
<code>while(list(, \$val) = each(\$array));</code>	3	209
<code>foreach(\$array as \$val);</code>	6	363
<code>while(list(\$key) = each(\$array)) \$tmp[] = &amp;\$array[\$key];</code>	9	565
<code>foreach(\$array as \$key =&gt; \$val) \$tmp[] = &amp;\$array[\$key];</code>	9	586
<code>foreach(\$array as \$key =&gt; \$val);</code>	13	804
<code>while(list(\$key, \$val) = each(\$array));</code>	14	906

Źródło: [PHPBench 2008]

Analizując wyniki testu (zob. tab. 1) można zauważyć, iż różnica pomiędzy najszybszym, a najwolniejszym wykonaniem pętli jest znaczna. Faktem jest więc, że wybór właściwej pętli może przyczynić się do znacznego wzrostu wydajności systemu.

Mając na uwadze optymalizację wydajności serwisu internetowego powinno się zwrócić szczególną uwagę na ograniczenie liczby operacji wyjścia, gdyż to właśnie one w głównej mierze są odpowiedzialne za szybkość działania całego systemu. W związku z tym należy zminimalizować ilość wywołań funkcji *echo()* na rzecz konkatencji kolejnych ciągów z dotychczas utworzonymi komunikatami.

Istotną sprawą podczas optymalizacji warstwy logiki jest również zastępowanie wywołań funkcji na rzecz użycia konstrukcji języka. Część operacji, które zazwyczaj realizowane są przy użyciu odpowiednich funkcji, można z powodzeniem zrealizować również w inny sposób, który jest lepszy z punktu widzenia wydajności. Przykładowo chcąc sprawdzić czy

dana tablica posiada określony rozmiar można skorzystać ze standardowego rozwiązania polegającego na zbadaniu jej rozmiaru. Sprawdzenie to realizowane jest przy pomocy funkcji *count()*, która zwraca liczbę elementów tablicy przekazanej jako parametr. Dokładnie ten sam efekt osiągnąć można z wykorzystaniem konstrukcji *isset()*. Sprawdzając czy istnieje element tablicy o indeksie 100 (czyli 101. Element tablicy, gdyż indeksowanie tablic rozpoczyna się od 0) uzyskiwana jest równocześnie informacja na temat tego, czy rozmiar tablicy jest większy od 100.

Podobne rozwiązanie można zastosować także przy sprawdzaniu czy ciąg posiada określoną długość. Przyjmując, że zmienna *\$string* zawiera ciąg znaków, można sprawdzić długość tego ciągu za pomocą funkcji *strlen()*. Jednak ten sam rezultat zostanie osiągnięty poprzez sprawdzenie, czy w ciągu istnieje dowolny znak znajdujący się na pierwszym (zerowym) miejscu (*isset(\$string[0])*).

Język PHP udostępnia kilka funkcji, które wymagają sporych nakładów obliczeniowych. Zwykle można je zastąpić ich uproszczonymi odpowiednikami:

- *strstr()* zamiast *ereg()* – jeżeli nie jest konieczne wyszukiwanie danych w ciągu z wykorzystaniem wyrażeń regularnych, nie potrzeba funkcji *ereg()* – do wyszukiwania podciągów nadaje się również dużo szybsza funkcja *strstr()*,
- *str\_replace()* zamiast *ereg\_replace()* – jak wyżej, jeżeli operacja zastępowania podciągów nie wymaga wyszukiwania wyrażeń regularnych, warto zastosować szybszą funkcję *str\_replace()*.

W sytuacjach, w których korzystanie z wyrażeń regularnych jest bezwzględnie konieczne, warto wykorzystać funkcje *PCRE (Perl Compatible Regular Functions)*, szybsze niż standardowe funkcje *ereg\_\**(*)* wbudowane w PHP [zob. Argreich 2003 s.873].

Z pozoru nieprzydatne operacje bitowe również mogą przyczynić się do poprawy wydajności serwisu internetowego. Idealnym przykładem ich użycia jest sprawdzanie, czy kolejna liczba, będąca wartością iteracji pewnej pętli, jest liczbą parzystą, czy nieparzystą. Standardowe rozwiązanie tego problemu opiera się na operacji dzielenia modulo. Dokładnie ten sam rezultat, lecz w sposób bardziej wydajny, można osiągnąć korzystając z operatora bitowego *AND (&)*. Binarne reprezentacje kolejnych wartości zmiennej *\$i* (czyli *0, 1, 10, 11* itd.) porównywane są z binarnym odpowiednikiem liczby 1 (czyli z binarnym *1*). W przypadku, gdy na ostatnim miejscu aktualnej reprezentacji znajduje się bit jedynekowy oznacza to, iż jest to liczba nieparzysta. Jeżeli natomiast ostatnim bitem jest bit zerowy jest to

liczba parzysta. Operacje sprawdzania parzystości liczb wykonywane są najczęściej w celu generowania kolejnych wierszy tabeli wypełnionych kolorami w sposób naprzemienny. Zadanie to można zrealizować w sposób bardziej wydajny korzystając z operacji bitowych zamiast z dzielenia modulo.

„Dane generowane przez skrypt PHP przesyłane są do przeglądarki jako ciągi języka HTML sformatowane zgodnie z wartością nagłówka *Content-Type* protokołu HTTP. Nowoczesne przeglądarki akceptują również dane skompresowane, dzięki czemu serwer może dokonać kompresji (m.in. za pomocą programu *gzip*) danych wyjściowych skryptu przed ich przesłaniem do przeglądarki; przeglądarka poddaje otrzymany ciąg dekompresji i wyświetla użytkownikowi uzyskany w ten sposób dokument HTML” [Argreich 2003 s.877].

Zastosowanie skompresowanych danych może „znaczaco zwiększyć wydajność skryptów i generatorów stron WWW produkujących obszerne ciągi wyjściowe. Testy tej techniki wykazały 60-procentowe skrócenie czasu wykonania skryptu“ [Argreich 2003 s.877].

Zajmując się optymalizacją wydajności serwisu internetowego w obszarze warstwy logiki nie można zapomnieć o bardzo istotnym elemencie jakim jest system cache. Odpowiedzialny jest on za przechowywanie na dysku serwera danych, które zmieniają się bardzo rzadko. System cache może zapisywać wygenerowaną zawartość całej strony, jej poszczególne elementy składowe (np. menu), a także wszelkie dane tworzone w trakcie wykonywania się skryptów server-side (np. tablice, obiekty). Prostym przykładem demonstrującym zasadę działania systemu cache jest klasa, odpowiedzialna za mechanizm składowania, pobierania oraz aktualizowania danych (zob. przykł. 3).

### Przykład 3

```
abstract class Cache {
    protected $id;
    protected $content;

    public function __construct($id) {
        $this->id = $id;
    }

    public function isFresh() {
        $file = $this->getPath();

        return (file_exists($file) && filemtime($file) + self::LIFETIME >
time());
    }
}
```

```

abstract public function start();

abstract public function save();

public function load() {
    return file_get_contents($this->getPath());
}

public function setContent($content) {
    $this->content = $content;
}

protected function getPath() {
    return self::DIRECTORY.$this->id.self::EXTENSION;
}

const DIRECTORY = './cache/';
const LIFETIME = 10;
const EXTENSION = '.cphp';
}

```

Abstrakcyjna klasa *Cache* stanowi dobry punkt wyjścia do budowy kolejnych, bardziej wyspecjalizowanych, podklas, które będą zajmowały się przechowywaniem danych konkretnego typu. Klasę tą można więc rozszerzyć do kolejnych klas (zob. przykł. 4), których zadaniem będzie cache'owanie wygenerowanej zawartości serwisu internetowego (klasa *Cache\_Content*) oraz cache'owanie tablic i obiektów (klasa *Cache\_Serialized*).

#### Przykład 4

```

class Cache_Content extends Cache {
    public function start() {
        ob_start();
    }

    public function save() {
        if (is_null($this->content)) {
            $this->content = ob_get_contents();
        }

        file_put_contents($this->getPath(), $this->content);
    }
}

```

```

class Cache_Serialized extends Cache {
    public function start() {
    }

    public function save() {
        file_put_contents($this->getPath(), serialize($this->content));
    }

    public function load() {
        return unserialize(parent::load());
    }
}

```

Obie przedstawione klasy dokonują specjalizacji odpowiednich metod, dzięki czemu mogą w pełni realizować powierzone im zadania. Wykorzystanie ich w praktyce sprowadza się do włączenia do kodu instrukcji warunkowej sprawdzającej aktualność przechowywanych danych (zob. przykł. 5 I 6).

#### Przykład 5

```

$c = new Cache_Content('my_cache_content');

if ($c->isFresh()) {
    echo $c->load();
}
else {
    $c->start();
    echo date('Y-m-d H:i:s');
    $c->save();
}

```

#### Przykład 6

```

$c = new Cache_Serialized('my_cache_serialized');

if ($c->isFresh()) {
    print_r($c->load());
}
else {
    $array = array('time' => date('Y-m-d H:i:s'));
    print_r($array);

    $c->setContent($array);
    $c->save();
}

```

W przypadku, gdy dane są nadal aktualne skrypt pobiera je bezpośrednio z dysku; w przeciwnym razie zostają one wygenerowane, po czym zapisane na dysku. Wykorzystanie mechanizmu cache powinno być podstawową kwestią przy optymalizacji wydajności serwisu internetowego w warstwie logiki, gdyż przynosi najbardziej wymierne korzyści w porównaniu do wcześniej przedstawionych metod. Przy stosunkowo niewielkich nakładach pracy można uzyskać zdecydowany wzrost wydajności serwisu internetowego.

### 2.3. Warstwa bazy danych

Tworząc serwis internetowy powinno się również skupić na odpowiednim projekcie bazy danych, gdyż to właśnie od jej wydajności w dużej mierze zależy wydajność całego systemu. Źle zaprojektowana baza danych jest w stanie skutecznie spowolnić nawet najlepiej zaprogramowany serwis.

Tworząc bazę danych należy zdefiniować typy danych, które będą przetrzymywane w poszczególnych polach tabel (zob. tab. 2). „Stosowanie typów danych o mniejszych rozmiarach może wpłynąć korzystnie na wydajność przyszłych zapytań” [Argreich 2003 s.883].

Tabela 2. Rozmiary wybranych typów danych w bazach danych

Typ danych	Rozmiar (w bajtach)		
	MySQL	PostgreSQL	MS SQL Server
BOOLEAN	-	1	-
TINYINT	1	-	1
SMALLINT	2	2	2
MEDIUMINT	3	-	-
INT, INTEGER	4	4	4
BIGINT	8	8	8
REAL	8	4	4
DOUBLE	8	8	-
DATE	3	4	-
TIME	3	8 lub 12	-
DATETIME	8	-	8
TIMESTAMP	4	8	8
ENUM	1 lub 2	-	-

Źródło: [MySQL typy danych 2008, PostgreSQL typy danych 2008, SQL Server typy danych 2008]

„W przypadku liczb całkowitych zalecane jest stosowanie jak najkrótszych typów danych. Dla przykładu, typ *MEDIUMINT* jest niekiedy przetwarzany wydajniej niż *INT*” [Argreich 2003 s.883].

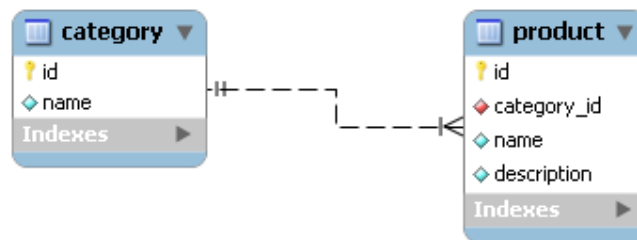
Dla pól typu znakowego wybór pomiędzy typem *VARCHAR*, a typem *CHAR* zależy od tego, czy dane w nich przetrzymywane będą zmiennej czy stałej długości. „Zaletą typu *VARCHAR(N)* jest określona maksymalna długość ciągu znaków oraz to, że zapisywanych jest przy jego użyciu tylko tyle znaków, ile potrzeba. W przypadku typu *CHAR(N)* długość ciągu jest stała, zatem typ ten może być w pewnych okolicznościach nieco bardziej wydajny. W zasadzie dla krótkich ciągów znaków o znanej długości najlepiej wykorzystać typ *CHAR(N)*. W przypadku znaczącego zróżnicowania długości dla różnych wierszy należy wybrać typ *VARCHAR(N)*. Lepiej stosować go także w przypadku wątpliwości co do rozmiaru ciągu znaków“ [Stones 2002 s.221]. „Jeżeli w danej tabeli musi występować kolumna o zmiennej długości, inne jej kolumny również można swobodnie definiować jako pola o zmiennej długości – i tak nie uda się już wykorzystać w tej tabeli zalet stałej długości pól. Jeżeli tabela zawiera pomiędzy grupą kolumn o stałej długości pojedynczą kolumnę dla danych typu *BLOB (Binary Large Objects)*, warto rozważyć umieszczenie tej kolumny w osobnej tabeli i odwoływać się do niej za pośrednictwem identyfikatora umieszczonego w kolumnie o stałej długości w tabeli pierwotnej“ [Argreich 2003 s.883].

Projektując bazę danych należy zwrócić szczególną uwagę na to, czy jej struktura jest *znormalizowana*. Wyróżnia się 5 postaci normalnych, z czego w praktyce stosowane są tylko 3 pierwsze [zob. Vieira 2007 s.237-242]:

- pierwsza postać normalna,
  - dotyczy eliminacji powtarzających się grup danych i gwarantuje *niepodzielność* danych (dane są samodzielne i niezależne),
- druga postać normalna,
  - tabela musi spełniać zasady pierwszej postaci normalnej,
  - każda kolumna musi zależeć od *całego* klucza,
- trzecia postać normalna,
  - tabela musi spełniać zasady drugiej postaci normalnej,
  - żadna kolumna nie może zależeć od żadnej innej kolumny niebędącej kluczem,
  - tabela nie może zawierać wyliczonych danych.

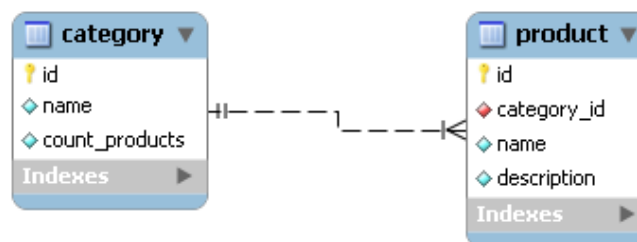
„Chociaż doprowadzenie bazy danych do trzeciej postaci normalnej (co oznacza, że jej struktura będzie zgodna z wszystkimi trzema regułami normalizacji) jest prawie zawsze rozwiązaniem pożądanym, istnieją sytuacje, w których trzeba złamać te zasady. Nazywa się to *denormalizacją* bazy danych i jest czasami konieczne w celu polepszenia wydajności. Jednak zawsze powinno się najpierw projektować bazę danych w pełni znormalizowaną, a potem denormalizować ją, jeżeli występują poważne problemy w wydajnością“ [Stones 2002 s.373].

Prostym przykładem obrazującym sposób denormalizacji bazy danych jest baza przetrzymująca dane o asortymencie sklepu, czyli o produktach oraz kategoriach, w których te produkty się znajdują. Wyjściowa wersja bazy danych zawiera 2 tabele: *category* oraz *product* (zob. Rys. 8).



Rysunek 8. Schemat znormalizowanej bazy danych  
Źródło: Opracowanie własne

Chcąc uzyskać raport przedstawiający podsumowanie listy kategorii oraz liczby produktów w nich się znajdujących, należy wykonać zapytanie wykorzystujące złączenie tabel oraz funkcję agregującą *COUNT()*. Czas uzyskania wyniku zależy od liczby kategorii oraz ilości produktów przechowywanych w bazie danych. W przypadku dużych baz danych może okazać się on zbyt długi i nieakceptowalny z racji konieczności częstego tworzenia raportów. W takiej sytuacji można zdenormalizować strukturę bazy poprzez dodanie kolumny *count\_products* w tabeli *category* (zob. Rys. 9).



Rysunek 9. Schemat zdenormalizowanej bazy danych  
Źródło: Opracowanie własne

Zadaniem nowej kolumny będzie przechowywanie liczby określającej aktualną ilość produktów przypisanych do danej kategorii. Aktualizację kolumny można zautomatyzować poprzez stworzenie odpowiedniego wyzwalacza. Wyzwalacz ten uruchamiany byłby na tabeli *product* po każdej operacji wstawienia, aktualizacji lub usunięcia rekordu, a jego praca polega na inkrementacji i/lub dekrementacji (w zależności od wykonanej operacji) wartości pola *count\_products* dla danej kategorii.

Poprzez dodanie nowej kolumny oraz utworzenie odpowiedniego wyzwalacza można w znacznym stopniu przyspieszyć generowanie listy kategorii wraz z liczbą produktów do nich przypisaną.

Z upływem czasu w bazie danych gromadzą się zdezaktualizowane wiersze, które zajmują niepotrzebnie miejsce, ale równocześnie nie można z nich skorzystać. Zazwyczaj są one wynikiem cofania transakcji. Bazy danych potrafią samodzielnie radzić sobie w takich przypadkach. Przykładowo baza danych PostgreSQL udostępnia proste w użyciu, ale potężne w możliwościach, polecenie *VACUUM*, służące do odzyskiwania utraconego miejsca. Aby oczyścić całą bazę danych ze zbędnych danych wystarczy wywołać polecenie bez podawania żadnego parametru. W celu odzyskania miejsca z konkretnej tabeli należy podać jej nazwę jako parametr wywoływanego polecenia.

Innym zadaniem polecenia *VACUUM* jest aktualizacja statystyk dotyczących poszczególnych tabel. Statystyki te wykorzystywane są przez optymalizator bazy danych w celu określenia sposobu wykonania danego zapytania. „W zależności od struktury bazy danych, kluczy podstawowych oraz liczby wierszy w tabelach, jeden sposób może być znacznie szybszy od innego. PostgreSQL próbuje określić, który ze sposobów wykonania zapytania jest najszybszy. Właśnie to jest zadaniem optymalizatora. Tworzy on plan zapytania przed jego wykonaniem. Zazwyczaj opiera się to na strukturze bazy danych oraz rozmiarze tabel, których dotyczy zapytanie [...]“ [Stones 2002 s.343].

W celu utrzymania statystyk aktualnymi należy skorzystać z polecenia *VACUUM* z opcją *ANALYZE*. Spowoduje to, iż baza danych ponownie przeliczy i zaktualizuje wszystkie statystyki, co przyniesie wymierne korzyści przy późniejszym wykonywaniu zapytań. Zaleca się, aby wykonywać operację *VACUUM* systematycznie, na przykład jako część codziennej obsługi. Dzięki temu miejsce zajmowane przez dane będzie miało minimalną objętość, a statystyki wykorzystywane przez optymalizator zapytań będą aktualne, co przyczyni się do zapewnienia maksymalnej wydajności [zob. Stones 2002 s.344].

Niestety, najpopularniejsza baza danych wykorzystywana przy tworzeniu serwisów internetowych, czyli MySQL, nie posiada polecenia *VACUUM*. Udostępnia natomiast inne polecenie, *OPTIMIZE TABLE*, którego systematyczne wykonywanie polepsza wydajność bazy danych. „Podczas usuwania danych z tabeli MySQL tworzy listę pustych pozycji, które potem są wypełniane zapytaniami INSERT. Dlatego po przeprowadzeniu na tabeli dużej liczby operacji usuwania lub operacji manipulujących rekordami o zmiennej długości należy tabelę poddać defragmentacji. Defragmentacja inicjowana jest poleceniem *OPTIMIZE TABLE*“ [Argreich 2003 s.883]. Zadaniem tego polecenia jest wykonanie następujących operacji [zob. Argreich 2003 s.883]:

- scalanie wierszy oraz redukcja rozproszenia luk po usuniętych wierszach,
- sortowanie wszystkich nieposortowanych stron indeksowych,
- aktualizacja statystyk tabeli.

Systematyczne wykonywanie operacji defragmentacji tabel wpłynie korzystnie na szybkość wykonywania zapytań kierowanych do bazy danych. Należy jednak przy tym pamiętać, że na czas wykonywania defragmentacji tabela jest blokowana, w związku z czym nie należy wykonywać tej operacji w momentach natężonej komunikacji z bazą danych.

Najważniejszym czynnikiem wpływającym na wydajność bazy danych są indeksy. „Indeksy to podstawowe elementy wspomagające mechanizm wyszukiwawczy bazy danych. Przeglądanie indeksu jest operacją wielokrotnie szybszą niż przeglądanie całej tabeli. Utworzenie jednego lub dwóch indeksów na danej tabeli optymalizuje 90% zapytań kierowanych do tej tabeli“ [Argreich 2003 s.884].

Indeks w bazie danych można porównać do indeksu haseł znajdującego się na końcu książki. Wyszukiwanie strony zawierającej pożądaną informację jest zdecydowanie szybsze z użyciem tegoż indeksu, aniżeli bez niego. Wówczas przeszukane musiałyby zostać wszystkie strony po kolei; wyszukiwanie takie nosi nazwę wyszukiwania sekwencyjnego lub, w bazach danych, skanowania tabel. „Skanowanie tabeli to całkiem prosty proces – polega na przeglądaniu wszystkich wierszy tabeli, zaczynając od jej fizycznego początku. Wiersze odpowiadające kryteriom zapytania zostają dodane do zbioru wyników“ [Vieira 2007 s.287]. Korzystając z indeksów wyszukiwanie przebiega w sposób podobny, jednak wykorzystywane są dodatkowe informacje. „Podczas procesu optymalizacji zapytań optymalizator przygląda się dostępnym indeksom i wybiera najlepszy (z reguły na podstawie informacji określonych w złączeniach i klauzuli *WHERE* w połączeniu ze statystycznymi informacjami

gromadzonymi podczas tworzenia indeksów). Po wybraniu indeksu SQL Server nawiguje przez strukturę drzewa do punktu danych spełniającego zadane kryterium i wyodrębnia tylko te rekordy, których potrzebuje. Różnica jest taka, że ponieważ dane są posortowane, to mechanizm zapytań wie, kiedy osiągnie koniec bieżącego poszukiwanego zakresu. Może wtedy zakończyć wykonanie zapytania lub przejść do następnego zakresu danych, jeśli to konieczne“ [Vieira 2007 s.287].

Powszechnie przyjętą zasadą jest, aby zakładać indeksy na kolumny będące kluczami obcymi odwołującymi się do innych tabel. Równocześnie należy zauważyć, iż kolumny oznaczone jako klucze główne zostają automatycznie zindeksowane. Nie ma więc potrzeby, lub wręcz nie powinno się, tworzyć indeksów na takich kolumnach.

Typ danych, jakie przechowywane są w danej kolumnie, również ma wpływ na użyteczność indeksów, a co za tym idzie również na wydajność samej bazy danych. Kolumny o typach numerycznych zdecydowanie lepiej radzą sobie z indeksami niż kolumny o typach znakowych. Zakładanie indeksów na kolumny charakteryzujące się niewielkim zróżnicowaniem danych (np. Przechowujące dane typu *BOOLEAN*) może przynieść efekty odwrotne do zamierzonych. Stanie się tak dlatego, iż dane przechowywane w takich indeksach będą właściwie wierną kopią danych z tabel przy równoczesym zwiększeniu rozmiaru bazy danych wynikłym z potrzeby fizycznego przechowywania zindeksowanych danych.

Indeksy można zakładać na pojedyncze kolumny lub na grupy kolumn. W przypadku indeksu obejmującego kilka kolumn ważna jest kolejność, w jakiej kolumny te zostały wymienione. „To, że indeks obejmuje dwie kolumny, nie oznacza, iż jest on użyteczny dla dowolnego zapytania, które odnosi się do każdej z nich. Indeks jest wybierany do wykorzystania tylko wtedy, jeśli pierwsza kolumna z wymienionych w indeksie zawarta jest w zapytaniu. Zaletą jest to, że nie musi być dokładnego dopasowania wszystkich kolumn – tylko tej pierwszej. Oczywiście, im więcej kolumn jest dopasowanych (w kolejności), tym lepiej, ale tylko pierwsza jednoznacznie wyznacza sytuację, w której indeks zostanie użyty“ [Vieira 2007 s.308].

Omawiając zagadnienie dotyczące indeksów w bazie danych należy również zaznaczyć, iż definiowanie nadmiernej ilości indeksów wpłynie niekorzystnie na wydajność całej bazy danych. „Indeksy znakomicie zwiększają wydajność zapytań *SELECT*, ale zmniejszają efektywność zapytań *UPDATE*, *INSERT* i *DELETE*“ [Argreich 2003 s.884]. Ponadto definiowanie jednego dużego indeksu, zawierającego wszystkie kolumny, w nadziei iż będzie

on pomocny we wszystkich sytuacjach sprawi, iż jedyne co zostanie osiągnięte to duplikacja przechowywanych danych. Indeksy mogą szkodzić tak samo, jak pomagają – wszystko zależy od tego w jaki sposób zostały stworzone oraz w jaki sposób są wykorzystywane [zob. Vieira 2007 s.314].

Chcąc przetestować wydajność serwisów internetowych można skorzystać z narzędzi, które są w stanie wygenerować szczegółowe raporty dotyczące obciążenia generowanego przez dany serwis internetowy. W następnym rozdziale omówione zostaną przykładowe narzędzia, dzięki którym możliwe jest przeprowadzanie testów wydajności serwisów internetowych. Ponadto przedstawione zostaną przykłady wykorzystania tych narzędzi.

## **ROZDZIAŁ 3. NARZĘDZIA TESTUJĄCE WYDAJNOŚĆ SERWISÓW INTERNETOWYCH**

W celu przetestowania wydajności serwisów internetowych można skorzystać z gotowych narzędzi dostępnych na rynku. Wśród nich znaleźć można narzędzia sprawdzające sposób wykonywania zapytań kierowanych do bazy danych, narzędzia do testowania wydajności poszczególnych fragmentów kodu serwisu internetowego, a także narzędzia, które są w stanie symulować odpowiedni ruch generowany przez określoną liczbę użytkowników.

W niniejszym rozdziale omówione zostaną wybrane narzędzia, dzięki którym można przeprowadzić testy wydajności serwisów internetowych od strony zapytań kierowanych do bazy danych, a także samego kodu źródłowego serwisu internetowego. Ponadto przedstawione zostaną dwa zaawansowane narzędzia potrafiące sprawdzić wydajność serwisu internetowego przy określonej liczbie odwiedzających.

### **3.1. Sposoby badania wydajności serwisów internetowych**

Badając wydajność kodu serwisu internetowego można skorzystać z gotowych bibliotek, które najczęściej są tworzone i rozwijane przez samą społeczność programistów danego języka. Do bibliotek tych można zaliczyć m.in. bibliotekę Benchmark z repozytorium PEAR będącego zbiorem oficjalnych rozszerzeń dla języka PHP.

Do testowania wydajność poszczególnych zapytań kierowanych do bazy danych działającej w serwisie internetowym warto wykorzystać z instrukcję EXPLAIN, która potrafi dostarczyć bardzo cennych informacji na temat sposoby, w jaki baza danych realizuje dane zapytanie.

W celu przetestowania wydajności serwisu internetowego jako całości można użyć programów, które specjalizują się w symulacji sporego obciążenia serwisu internetowego. Do takich narzędzi należą m.in. ApacheBench oraz Jmeter.

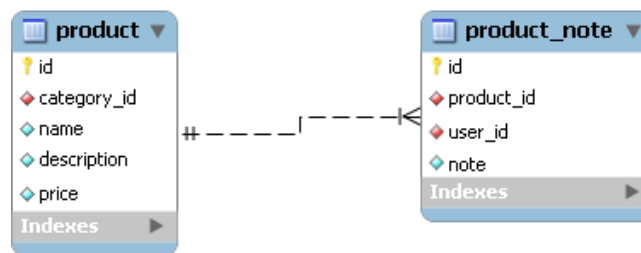
### 3.2. Badanie wydajności zapytań SQL za pomocą instrukcji EXPLAIN w bazach danych

Chcąc uzyskać informacje o tym, w jaki sposób baza danych wykonuje kierowane do niej zapytanie, można skorzystać z instrukcji *EXPLAIN*. Instrukcja ta dostarcza wielu cennych informacji, które mogą pomóc w optymalizacji lub ewentualnej modyfikacji zapytania. Wśród nich znaleźć możemy takie dane jak:

- koszt wykonywania poszczególnych fragmentów zapytania,
- tryb przeszukiwania tabel,
- informacje o użytych indeksach.

Analizując powyższe informacje można zlokalizować krytyczne miejsca danego zapytania oraz podjąć próbę ich eliminacji.

W celu przedstawienia informacji dostarczanych przez instrukcję *EXPLAIN*, zmodyfikowano strukturę przykładowej bazy danych przedstawionej w poprzednim rozdziale (zob. rys. 10).



Rysunek 10. Dodanie tabeli *product\_note* do przykładowej bazy danych  
Źródło: Opracowanie własne

Do tabeli *product* dodano nową kolumnę *price*, w której przechowywana jest cena danego produktu. Ponadto utworzono nową tabelę o nazwie *product\_note*, której zadaniem jest przechowywanie ocen użytkowników dla danego produktu. Pozostałe tabele nie uległy zmianie, w związku z czym nie zostały uwzględnione na diagramie.

Aby uzyskane wartości pomiarowe różniły się znacząco od siebie w poszczególnych etapach optymalizacji zapytania, tabele bazy danych zostały wypełnione rekordami:

- *product* – 10 000 rekordów,
- *product\_note* – 200 000 rekordów.

Warto w tym miejscu zauważyć, iż dane były generowane w sposób losowy w celu uzyskania środowiska zbliżonego w sposób maksymalny do rzeczywistego.

## Przykład 7

```
SELECT
  product.id, name, price,
  (SELECT AVG(note) FROM product_note WHERE product_id=product.id) AS avg_note
FROM
  product
WHERE
  price BETWEEN 700 AND 1000
ORDER BY
  price ASC,
  avg_note DESC
LIMIT 20
```

Zadaniem zapytania testowego (zob. przykł. 7) jest pobranie listy takich produktów, których cena znajduje się w przedziale między 700, a 1000. Na liście wynikowej powinny się znaleźć takie informacje o produkcie jak jego identyfikator, nazwa, cena oraz średnia ocena użytkowników. Produkty powinny zostać posortowane według ceny rosnąco oraz, w przypadku jednakowych cen kilku produktów, średniej oceny użytkowników malejąco. Ponadto lista powinna zawierać tylko 20 pierwszych rekordów wynikowych.

Wykonanie zapytania testowego z użyciem instrukcji *EXPLAIN* (zob. przykł. 8) zwróci informacje na temat tego, w jaki sposób zapytanie jest fizycznie wykonywane przez bazę danych.

## Przykład 8

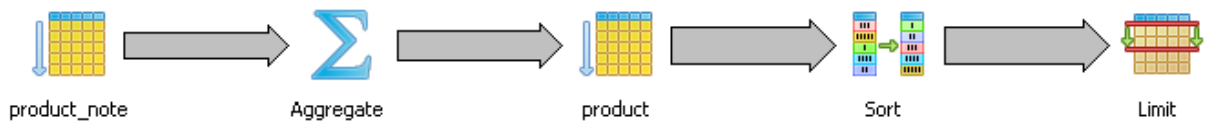
```
EXPLAIN SELECT ...
```

Należy w tym miejscu zaznaczyć, iż wszelkie wyniki przedstawione w niniejszej pracy, dotyczą bazy danych PostgreSQL; w innych systemach bazodanowych sposób prezentacji informacji zwróconych przez instrukcję *EXPLAIN* może odbiegać od tu przedstawionego.

## Przykład 9

```
Limit (cost=1023524.04..1023524.09 rows=20 width=30)
-> Sort (cost=1023524.04..1023524.75 rows=282 width=30)
    Sort Key: product.price, ((subplan))
-> Seq Scan on product (cost=0.00..1023516.54 rows=282 width=30)
    Filter: ((price >= 700::numeric) AND (price <= 1000::numeric))
    SubPlan
        -> Aggregate (cost=3628.46..3628.47 rows=1 width=2)
            -> Seq Scan on product_note (cost=0.00..3628.40 rows=23
width=2)
                Filter: (product_id = $0)
```

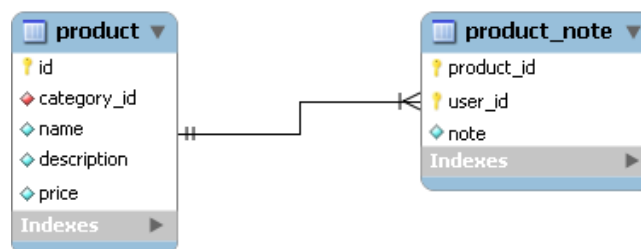
Zrozumienie uzyskanych informacji (zob. przykł. 9) z pewnością ułatwi diagram planu zapytania (zob. rys. 11), generowany automatycznie przez program PgAdmin III.



Rysunek 11. EXPLAIN SELECT... nr 1  
Źródło: Opracowanie własne

Pierwszym etapem wykonywania zapytania jest sekwencyjne przeglądanie (*Seq Scan*) tabeli *product\_note* wykorzystaniem filtra (nie indeksu) na kolumnie *product\_id*. Koszt tej operacji został oszacowany od 0,00 do 3628,40. Następnie następuje odpowiednie przetworzenie uzyskanych wyników w celu uzyskania odpowiedniej wartości średniej. W tym momencie oszacowany koszt został określony na poziomie od 3628,46 do 3628,47. Kolejnym etapem jest, również sekwencyjne, przeglądanie tabeli *product* w celu znalezienia tych produktów, których cena spełnia zadane kryteria. Koszt tej operacji może osiągnąć wartość nawet do 1023516,54. Dalej następuje odpowiednie posortowanie otrzymanych rekordów oraz ograniczenie zbioru wynikowego do określonej liczby rekordów. Całkowity czas wykonania zapytania (niewidoczny w raporcie instrukcji *EXPLAIN*) to średnio 14300 ms.

Posiadając wiedzę z zakresu w jaki sposób baza danych wykonuje zapytanie, można przystąpić do działań mających na celu jego przyspieszenie. Jak zauważono pierwszym etapem wykonywania zapytania jest sekwencyjne przeglądanie tabeli z ocenami produktów. Warto więc zastanowić się nad założeniem indeksu na kolumnie odnoszącej się do produktu (czyli *product\_id*). Przyjmując jednak, iż jeden użytkownik może oddać ocenę na dany produkt tylko raz, można utworzyć ograniczenie unikalności obejmujące kolumny *product\_id* oraz *user\_id*. Jeżeli więc takowa para będzie unikalna można z powodzeniem stworzyć z niej klucz podstawowy, na który, jak wiadomo, automatycznie nadawany jest indeks. W takim przypadku kolumna *id* okaże się zbędna i będzie można ją usunąć, przenosząc indeks na kolumny *product\_id* oraz *user\_id* (zob. rys. 12).

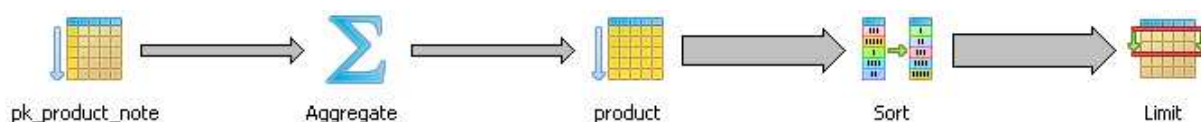


Rysunek 12. Zmiana klucza głównego w tabeli *product\_note*  
Źródło: Opracowanie własne

Po naniesieniu zmian można przystąpić do ponownego przetestowania szybkości realizowania zapytania, a także do dalszego badania planu zapytania.

### Przykład 10

```
Limit (cost=13618.54..13618.59 rows=20 width=30)
-> Sort (cost=13618.54..13619.25 rows=282 width=30)
    Sort Key: product.price, ((subplan))
    -> Seq Scan on product (cost=0.00..13611.04 rows=282 width=30)
        Filter: ((price >= 700::numeric) AND (price <= 1000::numeric))
        SubPlan
            -> Aggregate (cost=47.23..47.24 rows=1 width=2)
                -> Index Scan using pk_product_note on product_note
                    (cost=0.00..47.17 rows=23 width=2)
                    Index Cond: (product_id = $0)
```



Rysunek 13. EXPLAIN SELECT... nr 2

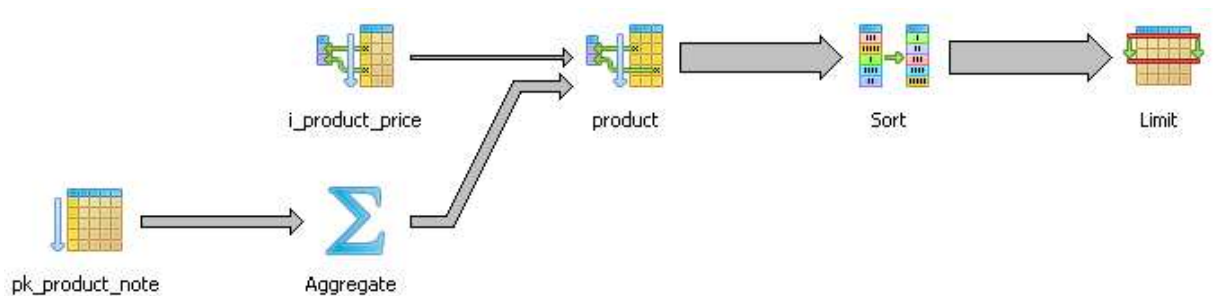
Źródło: Opracowanie własne

Korzyść z wprowadzonych modyfikacji widać od razu, tuż po wykonaniu zapytania. Zapytanie realizowane jest obecnie w czasie średnio 47 ms, czyli ok. 300 razy szybciej niż to miało miejsce przed zmianami. Dokładna analiza planu zapytania również dostarcza nowych informacji (zob. przykł. 10 i rys. 13). Zauważyć można, że przeglądanie tabeli *product\_note* ma teraz charakter indeksowy (*Index Scan*). Używany zostaje indeks obejmujący klucz główny tabeli (*pk\_product\_note*). Dzięki temu koszt tej operacji zmniejszono z poziomu 3628,40 do poziomu 47,17. Kolejne operacje również wykonywane są zdecydowanie szybciej, co ma wpływ na ogólną szybkość realizacji zapytania.

Ponowna analiza raportu instrukcji *EXPLAIN* pozwala zauważyć jeszcze jedno miejsce w zapytaniu, które można by poddać lekkiej optymalizacji w celu uzyskania jego większej wydajności. Chodzi o sekwencyjne przeglądanie tabeli *product* w celu selekcji tylko tych rekordów, które spełniają zadane warunki cenowe. Wartości w polu *price* są na tyle zróżnicowane, iż warto pomyśleć o założeniu indeksu na tą kolumnę. Pomimo, iż zapytanie i tak wykonuje się już znacznie szybciej niż pierwotnie, to nadal warto sprawdzić kolejną możliwość jego przyspieszenia.

## Przykład 11

```
Limit (cost=13483.92..13483.97 rows=20 width=30)
  -> Sort (cost=13483.92..13484.62 rows=282 width=30)
      Sort Key: product.price, ((subplan))
      -> Bitmap Heap Scan on product (cost=11.14..13476.41 rows=282
width=30)
          Recheck Cond: ((price >= 700::numeric) AND (price <=
1000::numeric))
          -> Bitmap Index Scan on i_product_price (cost=0.00..11.07
rows=282 width=0)
              Index Cond: ((price >= 700::numeric) AND (price <=
1000::numeric))
          SubPlan
              -> Aggregate (cost=47.23..47.24 rows=1 width=2)
                  -> Index Scan using pk_product_note on product_note
(cost=0.00..47.17 rows=23 width=2)
                      Index Cond: (product_id = $0)
```



Rysunek 14. EXPLAIN SELECT...nr 3

Źródło: Opracowanie własne

Na podstawie danych szacunkowych (zob. przykł. 11) zauważyć można jedynie niewielką poprawę w stosunku do poprzedniego raportu, jednak fizyczny czas wykonania został zmniejszony nieco ponad 3 razy i aktualnie wynosi średnio ok. 15 ms. Na diagramie planu zapytania (zob. rys. 14) można dokładnie zaobserwować, iż do jego realizacji wykorzystywane są 2 indeksy:

- `pk_product_note` (bazujący na kluczu głównym tabeli `product_note`),
- `i_product_price`.

To właśnie dzięki nim możliwe było tak znaczne przyspieszenie wykonywania zapytania. Przy niewielkim nakładzie pracy uzyskano bardzo dobre efekty finalne w postaci ogromnych zysków wydajnościowych (zob. tab. 3).

Tabela 3. Zyski wydajnościowe po analizie EXPLAIN SELECT...

	Przed zmianami	Po zmianach	
Koszt zapytania	1023524,09	13483,97	1,32%
Czas realizacji zapytania	14300 ms	15 ms	0,1%

Źródło: Opracowanie własne

Przeglądanie oraz analizowanie raportów planów zapytań kierowanych do bazy danych powinno być pierwszym krokiem do optymalizacji warstwy komunikacji z bazą danych. Dzięki temu można m.in. zlokalizować miejsca, w których warto by było pomyśleć o założeniu indeksu lub takie, w których indeks nie jest używany. Opisany w pracy przykład jest bardzo prosty i nie wymaga zbyt wielu operacji na tabelach. Jednak dzięki swej prostocie w bardzo dobry sposób obrazuje korzyści, jakie daje odpowiednia struktura bazy danych.

### 3.3. Profilowanie kodu aplikacji za pomocą biblioteki Benchmark z repozytorium PEAR

Biblioteka Benchmark, wchodząca w skład repozytorium PEAR, jest zbiorem 3 klas umożliwiających śledzenie obciążenia, jakie generują poszczególne fragmenty kodu serwisu internetowego. Wśród tych klas znaleźć można *Benchmark\_Iterate*, *Benchmark\_Profiler*, *Benchmark\_Timer*. Głównym zadaniem biblioteki jest pomiar czasu, jaki jest potrzebny do wykonania poszczególnych fragmentów kodu serwisu internetowego. Dzięki temu można zlokalizować te fragmenty, które zużywają najwięcej zasobów i na wykonanie których potrzeba najwięcej czasu.

Instalacja biblioteki sprowadza się do wykonania prostego polecenia z linii komend (zob. przykł. 12).

Przykład 12

```
pear install --alldeps Benchmark
```

Zaleca się skorzystanie podczas instalacji z przełącznika `--alldeps`, dzięki któremu automatycznie zostaną również zainstalowane inne biblioteki, od których zależna jest biblioteka Benchmark.

Biblioteka Benchmark składa się 3 klas:

- *Benchmark\_Iterate*,
- *Benchmark\_Profiler*,

- *Benchmark\_Timer*.

Klasa *Benchmark\_Iterate* odpowiedzialna jest za cykliczne wykonywanie określonych sekcji kodu. Doskonałym przykładem wykorzystania tej klasy jest badanie obciążenia, jakie generuje dana funkcja lub metoda klasy (zob. przykł. 13).

#### Przykład 13 <sup>1</sup>

```
require_once 'Benchmark/Iterate.php';

$benchmark = new Benchmark_Iterate;

function foo($string) {
    print $string . '<br>';
}

$benchmark->run(100, 'foo', 'test');
$result = $benchmark->get();
```

W wyniku uruchomienia kodu funkcja *foo()* zostanie wykonana 100 razy, po czym do zmiennej *\$result* zostanie przypisana tablica zawierająca informacje na temat czasu wykonywania poszczególnych iteracji (zob. przykł. 14).

#### Przykład 14

```
Array
(
    [1] => 0.000076
    [2] => 0.000044
    [3] => 0.000042
    ...
    [98] => 0.000042
    [99] => 0.000043
    [100] => 0.000043
    [mean] => 0.000045
    [iterations] => 100
)
```

Dodatkowo raport zawiera informacje na temat liczby wykonanych iteracji, a także średniego czasu, jaki był potrzebny do wykonania funkcji. Klasa *Benchmark\_Iterate* może znaleźć zastosowanie szczególnie podczas testowania tych funkcji w serwisie internetowym,

---

<sup>1</sup> Źródło: [PHP Benchmark 2008]

które korzystają z wielu zasobów danych oraz często komunikują się z bazą danych w celu pobrania, wstawienia, zmodyfikowania lub usunięcia danych.

Kolejną klasą wchodzącą w skład biblioteki PEAR jest klasa *Benchmark\_Profiler*. Służy ona również do pomiaru czasu wykonania danej funkcji lub metody klasy, jednak tym razem jej egzemplarz osadzany jest bezpośrednio w ciele badanej funkcji. Dzięki temu można w bardzo prosty sposób dokonać pomiarów wydajnościowych już istniejącego oprogramowania serwisu internetowego, bez konieczności tworzenia osobnych testów (zob. przykł. 15).

#### Przykład 15<sup>2</sup>

```
require_once 'Benchmark/Profiler.php';

$profiler = new Benchmark_Profiler(TRUE);

function myFunction() {
    global $profiler;
    $profiler->enterSection('myFunction');

    //do something

    $profiler->leaveSection('myFunction');

    return;
}

//do something

myFunction();
```

Uruchomienie kodu spowoduje utworzenie funkcji *myFunction()*, w ciele której dostępny będzie egzemplarz klasy *Benchmark\_Profiler*. Obiekt ten będzie miał za zadanie mierzenie czasu wejścia oraz wyjścia z funkcji. Funkcja *myFunction()* zostanie uruchomiona, a obiekt *\$profiler* będzie gromadził potrzebne informacje.

	total ex. time	netto ex. time	#calls	% calls	callers
<b>myFunction</b>	4.2915344238281E-5	4.2915344238281E-5	1	N/A	Global (1)

Rysunek 15. Raport *Benchmark\_Profiler*  
Źródło: Opracowanie własne

---

<sup>2</sup> Źródło: [PHP Benchmark 2008]

Raport wygenerowany przez klasę *Benchmark\_Profiler* (zob. rys. 15) zawiera m.in. takie informacje, jak:

- łączny czas wykonania wszystkich wywołań funkcji,
- czas potrzebny na wykonanie funkcji 1 raz,
- liczba wywołań funkcji,
- źródła wywołań funkcji wraz z liczbą wywołań.

Dzięki tym informacjom można prześledzić narzuty czasowe, jakie niesie ze sobą wywoływanie danej funkcji. Można również zdiagnozować skąd wywoływana jest dana funkcja; możliwe, że część wywołań jest zupełnie zbędna i niepotrzebnie wpływa na osłabienie wydajności serwisu internetowego. Możliwe jest także wykrycie niepożądanych wywołań rekurencyjnych.

Klasa *Benchmark\_Timer* jest najistotniejszą klasą wchodzącą w skład biblioteki *Benchmark*. Umożliwia ona określanie punktów kontrolnych w kodzie serwisu internetowego, na podstawie których sporządzony zostanie odpowiedni raport wynikowy. Klasa ta, pomimo swej prostoty, może okazać się bardzo przydatnym narzędziem w procesie lokalizacji krytycznych miejsc serwisu internetowego.

#### Przykład 16<sup>3</sup>

```
require_once 'Benchmark/Timer.php';

$timer = new Benchmark_Timer(true);
$timer->setMarker('Marker 1');

$timer->display();
```

Tworząc nowy obiekt *Benchmark\_Timer* inicjowany jest automatycznie (dzięki pierwszemu parametrowi o wartości *true*) pomiar czasu (zob. przykł. 16). Dodając punkty kontrolne za pomocą metody *setMarker()* można badać różnice czasowe pomiędzy poszczególnymi sekcjami kodu. Dzięki temu można dowiedzieć się m.in. ile czasu zajmuje pobranie danych z bazy danych lub jak długo trwa wykonywanie skomplikowanych obliczeń.

---

<sup>3</sup> Źródło: [PHP Benchmark 2008]

	time index	ex time	%
Start	1219774402.57812800	-	0.00%
Marker 1	1219774402.57820300	0.000075	84.27%
Stop	1219774402.57821700	0.000014	15.73%
total	-	0.000089	100.00%

Rysunek 16. Raport Benchmark\_Timer  
Źródło: Opracowanie własne

Rezultatem działania kodu z wykorzystaniem klasy *Benchmark\_Timer* jest wygenerowanie raportu końcowego w postaci tabeli. Tabela ta zawiera informacje na temat czasów wejścia do danego punktu kontrolnego, czas jego wykonania, a także procentowy udział w całkowitym czasie wykonania kodu.

Biblioteka *Benchmark* może znaleźć zastosowanie w bieżącej kontroli wydajności kodu. Jej implementacja w aplikacji jest bardzo prostym zadaniem, a co za tym idzie zajmującym niewiele czasu. Dzięki niej można na bieżąco kontrolować wydajność poszczególnych elementów kodu. Można również wybrać najwydajniejsze rozwiązanie spośród kilku możliwych do zastosowania poprzez empiryczne przetestowanie każdego z nich.

### 3.4. Testowanie wydajności serwisów internetowych za pomocą zaawansowanych aplikacji

W celu określenia wydajności serwisu internetowego jako całości (tj. oprogramowania, bazy danych, a także samego serwera) można skorzystać z zaawansowanych aplikacji testowych. Aplikacje te potrafią wykonać określoną liczbę żądań kierowanych do serwera i na tej podstawie oszacować poziom wydajności serwisu internetowego. Uzyskane rezultaty mogą zostać przedstawione w postaci tabeli lub wykresu graficznego.

#### 3.4.1. ApacheBench

ApacheBench jest przykładem bardzo prostej w obsłudze, ale równocześnie bardzo użytecznej, aplikacji, dzięki której można przeprowadzić wstępne testy wydajności całego serwisu internetowego. Jest on dostarczany wraz z serwerem Apache i stanowi jego integralną część. ApacheBench pracuje tylko i wyłącznie w trybie tekstowym. W celu uruchomienia programu należy w oknie konsoli wpisać polecenie *ab*, będące nazwą aplikacji. Wyświetlona

zostanie lista dostępnych parametrów, których można używać w celu wykonania najbardziej odpowiedniego testu (zob. przykł. 17).

#### Przykład 17

```
-n requests      Number of requests to perform
-c concurrency   Number of multiple requests to make
-t timelimit     Seconds to max. wait for responses
-p postfile      File containing data to POST
-T content-type  Content-type header for POSTing
-v verbosity     How much troubleshooting info to print
-w              Print out results in HTML tables
-I              Use HEAD instead of GET
-x attributes    String to insert as table attributes
-y attributes    String to insert as tr attributes
-z attributes    String to insert as td or th attributes
-C attribute     Add cookie, eg. 'Apache=1234. (repeatable)
-H attribute     Add Arbitrary header line, eg. 'Accept-Encoding: gzip'
                 Inserted after all normal header lines. (repeatable)
-A attribute     Add Basic WWW Authentication, the attributes
                 are a colon separated username and password.
-P attribute     Add Basic Proxy Authentication, the attributes
                 are a colon separated username and password.
-X proxy:port    Proxyserver and port number to use
-V              Print version number and exit
-k              Use HTTP KeepAlive feature
-d              Do not show percentiles served table.
-S              Do not show confidence estimators and warnings.
-g filename      Output collected data to gnuplot format file.
-e filename      Output CSV file with percentages served
-h              Display usage information (this message)
```

Najbardziej istotnymi opcjami są `-n` oraz `-c`. Pierwsza definiuje łączną liczbę żądań, jakie zostaną przesłane do serwera, natomiast druga określa liczbę żądań wykonywanych w tym samym czasie. Opcje te są w zupełności wystarczające do wykonania prostych testów wydajnościowych polegających na symulacji odwiedzin serwisu internetowego. Pozostałe parametry umożliwiają m.in. przesyłanie dodatkowych nagłówek oraz danych, a także odpowiednie formatowanie i zapisywanie danych wynikowych (np. w postaci pliku HTML).

Aby wykonać test wydajności wybranego serwisu internetowego należy skorzystać z polecenia `ab` wraz z określeniem koniecznych opcji, a także z podaniem samego adresu testowanego serwisu internetowego (zob. przykł. 18).

## Przykład 18

```
ab -n 100 -c 10 http://example.com/
```

ApacheBench wykona 100 żądań do strony internetowej `http://example.com/`, przy czym równocześnie wykonywane będzie zawsze po 10 żądań. W rezultacie serwis internetowy zostanie przetestowany pod kątem wydajności podczas symulacji 10 serii żądań po 10 pojedynczych żądań w każdej serii.

## Przykład 19

```
Server Software:      Apache/2.2.3
Server Hostname:     example.com
Server Port:         80

Document Path:      /
Document Length:    438 bytes

Concurrency Level:   10
Time taken for tests: 25.687500 seconds
Complete requests:   100
Failed requests:     0
Write errors:        0
Total transferred:   70200 bytes
HTML transferred:    43800 bytes
Requests per second: 3.89 [#/sec] (mean)
Time per request:    2568.750 [ms] (mean)
Time per request:    256.875 [ms] (mean, across all concurrent requests)
Transfer rate:       2.65 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median  max
Connect:     218  253  23.6    250   421
Processing:   250 2179 412.3   2265  2437
Waiting:     234 1264 668.7   1265  2437
Total:       500 2433 413.6   2515  2687

Percentage of the requests served within a certain time (ms)
 50%    2515
 66%    2515
 75%    2531
 80%    2671
 90%    2671
 95%    2671
 98%    2687
 99%    2687
```

100% 2687 (longest request)

Wynikiem wykonania polecenia jest szczegółowy raport dotyczący zrealizowanych operacji oraz czasu, w jakim zostały one wykonane (zob. przykł. 19). Raport ten zawiera ponadto liczbę pomyślnie zrealizowanych żądań oraz takich, które zakończyły się niepowodzeniem. Dodatkowo można z niego odczytać także informacje dotyczące rozmiaru przesłanych danych, a także szybkości transferu (zob. tab. 4).

Tabela 4. Analiza raportu ApacheBench

Nazwa wyniku:	Opis wyniku:	Wartość wyniku:
Server Software	Nazwa i wersja serwera, spod którego został uruchomiony test	Apache/2.2.3
Server Hostname	Nazwa hosta serwera testowanego	example.com
Server Port	Numer portu, na którym uruchomiony jest serwer, spod którego został uruchomiony test	80
Document Path	Ścieżka do testowanego dokumentu	/
Document Length	Rozmiar testowanego dokumentu	438 B
Concurrency Level	Liczba żądań uruchamianych w tym samym czasie	10
Time taken for test	Całkowity czas realizacji testu	25,687500 s
Complete requests	Liczba żądań zakończonych sukcesem	100
Failed requests	Liczba żądań zakończonych niepowodzeniem	0
Write errors	Liczba zapisanych błędów żądań	0
Total transferred	Łączna liczba odebranych danych (kod HTML oraz nagłówki)	70200 B
HTML transferred	Łączna liczba odebranych danych (tylko kod HTML)	43800 B
Requests per second	Średnia liczba żądań obsłużona w czasie 1 sekundy	3,89 #/s
Time per request	Średni czas potrzebny na realizację jednej serii żądań	2568,750 ms
Time per request (across all concurrent requests)	Średni czas potrzebny na realizację jednego żądania w jednej serii żądań	256,875 ms
Transfer rate	Szybkość transferu przy odbieraniu danych	2,65 kB/s

Źródło: Opracowanie własne

Analizując uzyskane dane można stwierdzić, iż wszystkie żądania kierowane do serwera zostały pomyślnie zrealizowane. W przypadku, gdyby któreś z nich zakończyło się niepowodzeniem mógłby to być znak, iż serwer nie jest w stanie zrealizować danej liczby żądań do danego serwisu internetowego, czego przyczyną mogłaby być słaba optymalizacja wydajności serwisu. Bardzo istotnym parametrem jest *Requests per second* określa on liczbę

zadań możliwych do zrealizowania w czasie 1 sekundy. Im wyższa wartość tego parametru tym większa zdolność do masowej realizacji ządań.

Raport uzyskany dzięki testowi z wykorzystaniem ApacheBench pokazuje również czasy potrzebne do realizacji ządań z wyszczególnieniem nawiązywania połączenia, przetwarzania danych oraz oczekiwania na odpowiedź (zob. przykł. 20).

#### Przykład 20

Connection Times (ms)				
	min	mean[+/-sd]	median	max
Connect:	218	253 23.6	250	421
Processing:	250	2179 412.3	2265	2437
Waiting:	234	1264 668.7	1265	2437
Total:	500	2433 413.6	2515	2687

Raport czasów połączeń udostępnia dodatkowo wyniki obliczeń obrazujące najkrótszy oraz najdłuższy czas w danej sekcji, wartość średnią, a także dane typowo statystyczne czyli odchylenie standardowe oraz medianę.

ApacheBench potrafi również określić procentową liczbę zrealizowanych ządań w kolejnych odstępach czasu (zob. przykł. 21).

#### Przykład 21

Percentage of the requests served within a certain time (ms)	
50%	2515
66%	2515
75%	2531
80%	2671
90%	2671
95%	2671
98%	2687
99%	2687
100%	2687 (longest request)

Analizując uzyskane dane można zauważyć, iż 66% ządań zostało obsłużonych w czasie wynoszącym 2515 ms (nie odnotowano różnic czasowych między wykonaniami 50% a 66% ządań). Uzyskanie proporcjonalnych wyników może świadczyć o tym, iż testowany serwis internetowy działa w sposób stabilny.

### 3.4.2. Jmeter

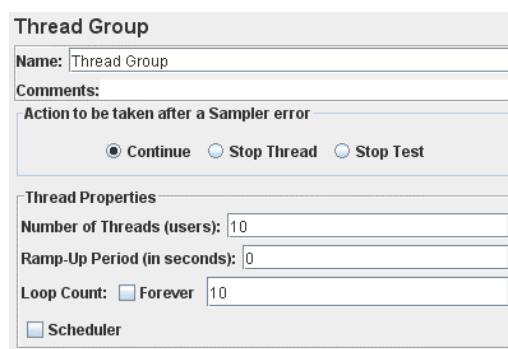
W celu zbadania wydajności serwisu internetowego można również skorzystać z innej, bardziej zaawansowanej, aplikacji jaką jest Jmeter. Początkowo Jmeter miał służyć jedynie do

testowania serwisów internetowych, jednak z biegiem czasu został rozbudowany o dodatkowe funkcje i możliwości. Można go zastosować do sprawdzenia wydajności statycznych oraz dynamicznych zasobów takich jak m.in. pliki, serwlety czy bazy danych. Można go użyć do symulowania wysokiego obciążenia występującego na serwerze, sieci lub innych testowanych obiektach. Istotną zaletą programu jest to, że potrafi przedstawić uzyskane wyniki w sposób graficzny w postaci wykresów [zob. Jmeter 2008].

Wśród mnogości funkcji, jakie oferuje Jmeter, warto wyszczególnić możliwość przeprowadzania testów serwerów następujących typów:

- HTTP oraz HTTPS,
- FTP,
- SOAP,
- bazy danych za pośrednictwem JDBC,
- LDAP,
- JMS,
- pocztowych typu POP3.

Aby wykonać test wydajności serwisu internetowego jako całości, należy w menu programu wybrać opcję *Add* → *ThreadGroup*. Zostanie utworzona nowa grupa wątków, w której będzie można określić takie parametry jak testowa liczba użytkowników oraz częstotliwość wysyłania żądań (zob. rys. 17).



Rysunek 17. Tworzenie grupy wątków w Jmeter

Źródło: Opracowanie własne

Określając liczbę wątków na wartość 10 oraz liczbę powtórzeń również na poziomie 10 uzyskano symulację 10 serii odwiedzin po 10 użytkowników w każdej. Opóźnienie wykonywanych żądań pomiędzy kolejnymi wątkami ustawiono na 0, co oznacza, iż kolejny wątek będzie uruchamiany natychmiast po zakończeniu poprzedniego.

Następnym krokiem jest ustalenie domyślnych ustawień dla wysyłanych żądań. W celu utworzenia tych danych należy wybrać z menu opcję *Add* → *Config Element* → *HTTP Request Defaults*.

Name	Value	Encode?	Include Equ...
------	-------	---------	----------------

Rysunek 18. Określanie domyślnych ustawień żądań HTTP w Jmeter  
Źródło: Opracowanie własne

W wyświetlonym oknie można zdefiniować m.in. takie parametry żądania jak adres serwera, port czy protokół jakim będą wysyłane żądania (zob. rys. 18). Ponadto można określić również dodatkowe parametry, które zostaną uwzględnione w wykonywanych żądaniach.

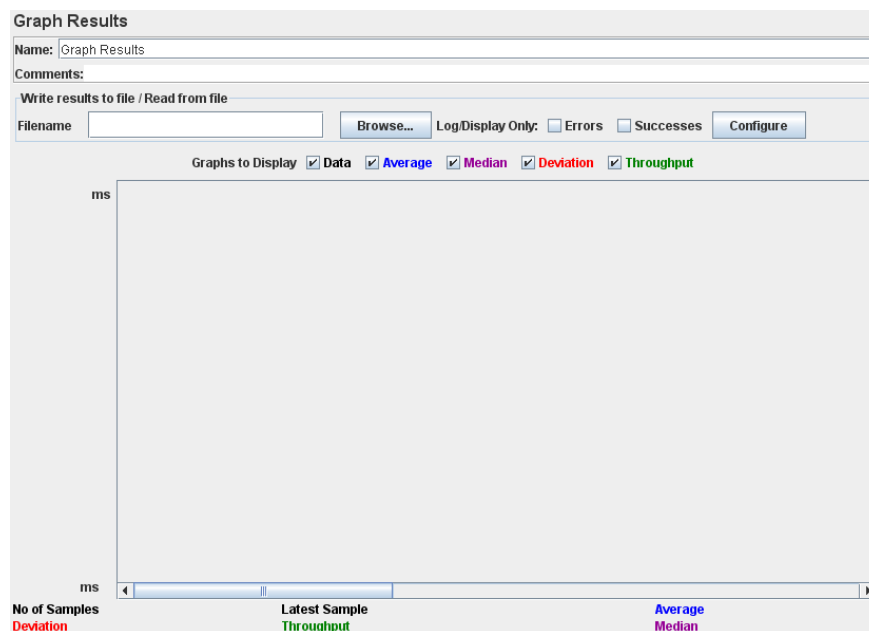
Po utworzeniu domyślnych parametrów żądań należy utworzyć przykładowe żądanie kierowane do serwera. W tym celu trzeba wybrać z menu opcję *Add* → *Sampler* → *HTTP Request*.

Name	Value	Encode?	Include Equ...
------	-------	---------	----------------

Rysunek 19. Tworzenie nowego żądania HTTP w Jmeter  
Źródło: Opracowanie własne

W tym momencie można określić parametry specyficzne dla konkretnego żądania (zob. rys. 19). Jeśli jednak zdefiniowane wcześniej parametry domyślne nie muszą być nadpisane można wówczas pozostawić stworzone żądanie bez nanoszenia jakichkolwiek zmian.

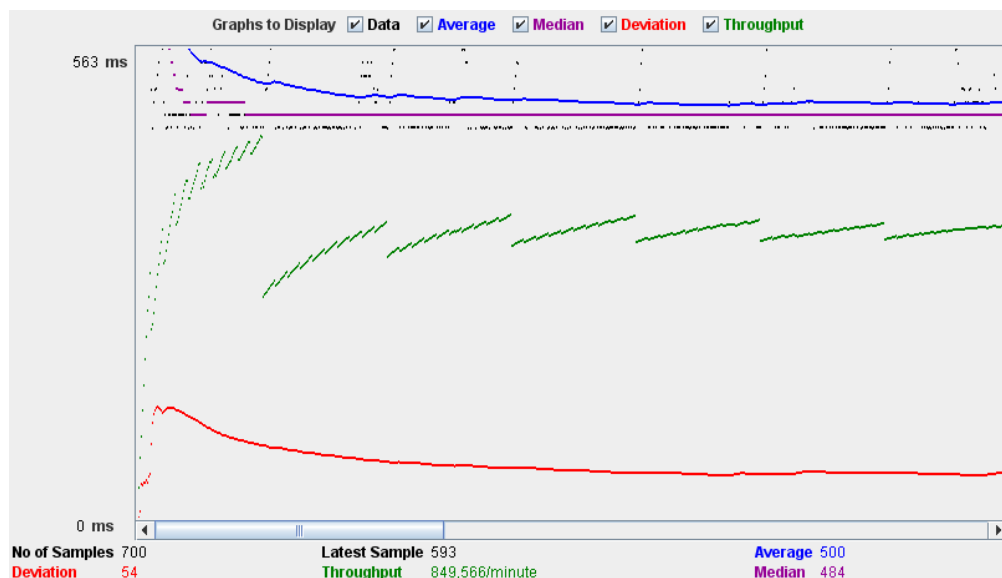
Aby uzyskać graficzną reprezentację uzyskanych wyników należy dodać do tworzonego planu komponent wykresu. Uczynić to można wybierając z menu opcję *Add* → *Listener* → *Graph Results*.



Rysunek 20. Dodawanie grafu wyników w Jmeter  
Źródło: Opracowanie własne

Dodany komponent wykresu przedstawiać będzie na bieżąco uzyskiwane wartości w teście (zob. rys. 20). Po zakończeniu testu możliwe będzie zapisanie grafu do pliku.

Aby wykonać przygotowany plan testu należy z menu wybrać opcję *Run* → *Start*. Test rozpocznie się, a na wykresie wyników będą pojawiać się kolejne krzywe reprezentujące uzyskiwane wyniki (zob. rys. 21).



Rysunek 21. Graf żądań HTTP w Jmeter  
Źródło: Opracowanie własne

Wykonując test kilkakrotnie można zaobserwować, iż uzyskiwane wyniki są do siebie bardzo zbliżone. Jedynie wyniki otrzymane w pierwszym teście są wyraźnie różne od pozostałych. Kolejne serie żądań wydają się być realizowane w sposób coraz bardziej stabilny. Przy wykonaniu łącznej liczby 700 żądań średni czas realizacji żądania wyniósł 500ms, natomiast przepustowość została określona na prawie 850 jednostek na minutę.

Porównując wyniki uzyskane przy pomocy ApacheBench oraz Jmeter można zauważyć ogromne rozbieżności. Przyczyny tego należy szukać w zupełnie innych technologiach, w których wykonano obie aplikacje. ApacheBench jest w rzeczywistości prostym klientem HTTP, którego dodatkową cechą jest możliwość mierzenia czasów realizacji żądań. Praca w trybie tekstowym sprawia, iż program wymaga bardzo niewielką ilość zasobów komputera. Jmeter natomiast jest rozbudowaną aplikacją o różnorodnym zastosowaniu. Kod źródłowy programu został wykonany w języku Java, co pociąga za sobą pewien wzrost zapotrzebowania na zasoby komputera, ponieważ każdy wątek stanowi osobny obiekt w pamięci; w przypadku testu bazującego na wielu próbkach liczba tworzonych przez program obiektów może wzrosnąć do olbrzymiej liczby. Dodatkowo Jmeter posiada graficzny interfejs użytkownika, który również ma wpływ na uzyskiwane rezultaty testów. [zob. Jmeter vs ApacheBench 2008]. Z tych właśnie powodów zaleca się przeprowadzanie testów z wykorzystaniem tylko jednego z tych narzędzi.

W kolejnym rozdziale przedstawiony zostanie przykład optymalizacji wydajności serwisu internetowego. Jako testowy serwis internetowy posłuży autorskie oprogramowanie sklepu internetowego.

## **ROZDZIAŁ 4. PRZYKŁAD OPTIMALIZACJI WYDAJNOŚCI SERWISU INTERNETOWEGO OPROGRAMOWANEGO W JĘZYKU PHP**

Posiadając wiedzę teoretyczną z zakresu optymalizacji wydajności serwisów internetowych można zająć się jej praktycznym zastosowaniem. Jako testowy serwis internetowy posłuży oprogramowanie sklepu internetowego.

Dokonane zostaną pomiary wydajności zarówno przed, jak i po dokonaniu odpowiednich zabiegów optymalizacyjnych. Zestawiając ze sobą uzyskane wyniki będzie można zobaczyć w jakim stopniu zwiększyła się wydajność testowego serwisu internetowego.

### **4.1. Charakterystyka testowego serwisu internetowego oraz środowiska testowego**

Testowy serwis internetowy został w pełni napisany w języku PHP. Do jego zaprogramowania został wykorzystany framework Kohana będący jednym z popularniejszych frameworków dla tego języka. Wśród cech charakteryzujących framework Kohana można wymienić m.in. [zob. Kohana informacje 2009]:

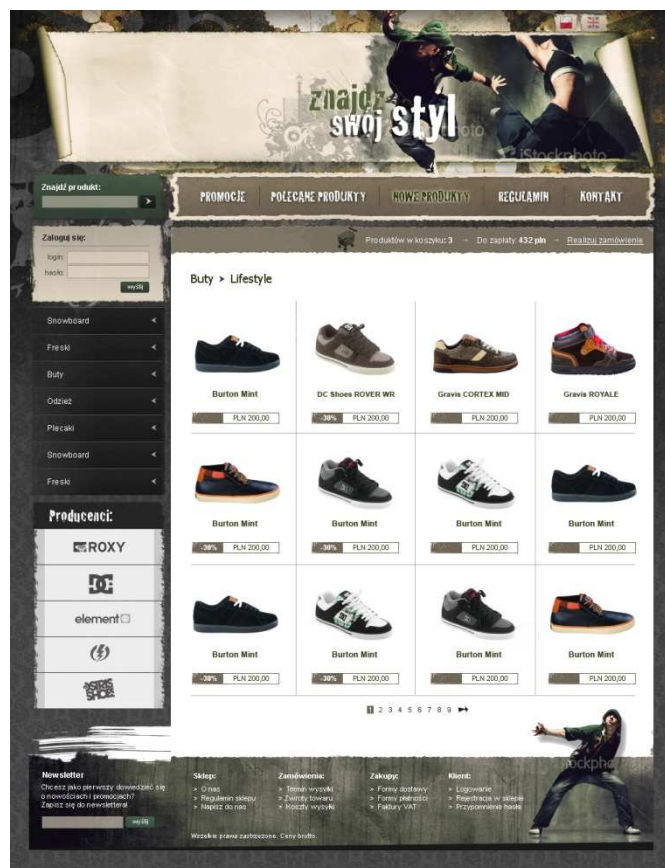
- wysoki poziom bezpieczeństwa tworzonych serwisów internetowych,
- ekstremalną szybkość działania,
- korzystanie z architektury MVC (Model-View-Controller, czyli Model-Widok-Kontroler),
- pełne wsparcie dla kodowania UTF-8,
- luźno sprzężoną architekturę,
- prostotę dalszego rozwijania i rozbudowywania,
- wykorzystywanie technik programowania obiektowego w PHP 5,
- prostą obsługę baz danych przy wykorzystaniu sterowników SQL,
- sesje wielokrotne (natywna, bazy danych i cookie),
- rozbudowany system obsługi zdarzeń pozwalający na modyfikacje,
- ogólna idea oraz filozofia bazująca na frameworku CodeIgniter.

Decyzja o wyborze Framework Kohana została podjęta głównie z racji tego, iż jest on bezpieczny, wydajny oraz łatwy w użyciu. Konkurencyjne rozwiązania (takie jak m.in. Zend Framework lub Symfony) mogą z pewnością pochwalić się większym wachlarzem możliwości, natomiast ustępują Kohana miejsca na polu wydajności. Przykładowy serwis internetowy nie wykorzystywałby w pełni ich możliwości, w związku z czym wybór rozwiązania mniejszego i lżejszego, a co za tym idzie zdecydowanie bardziej wydajnego, wydaje się być właściwym. Ponadto Kohana jest stale rozwijana, bardzo często publikowane są jej kolejne wydania. Aktualna wersja tego frameworka oznaczona jest numerem 2.3.1 i nosi nazwę kodową „Accipiter”. Dodatkowym atutem Kohana jest prostota instalacji. Aby uruchomić serwis internetowy napisany przy jej pomocy, wystarczy przenieść pliki na serwer i ustawić odpowiednie prawa dla kilku katalogów oraz dane dostępowe do bazy danych.

Wybierając bazę danych dla testowego serwisu internetowego rozważano wybór jedynie między darmowymi rozwiązaniami, czyli pomiędzy MySQL oraz PostgreSQL. Pomimo ogromnej popularności bazy MySQL dla zastosowań w Internecie, wybrano bazę danych PostgreSQL w najnowszej wersji, tj. 8.3. Wyboru dokonano opierając się w dużej mierze na porównaniu obu tych serwerów (zob. MySQL vs PostgreSQL 2009). Głównym czynnikiem, który wpłynął na wybór PostgreSQL była pełna obsługa transakcji oraz więzów integralności. MySQL posiada co prawda taką funkcjonalność ale tylko i wyłącznie w przypadku wyboru InnoDB jako mechanizmu składowania danych (który jest wyraźnie wolniejszy od MyISAM). W takiej sytuacji różnice wydajnościowe między tymi bazami zacierają się i mają znaczenie raczej marginalne. Kolejną istotną przewagą PostgreSQL jest fakt, iż lepiej obsługuje on sporą ilość konkurencyjnych połączeń co może mieć spore znaczenie w przypadku osiągnięcia przez serwis internetowy dużej popularności. MySQL ustępuje PostgreSQL również na polu możliwości tworzenia procedur i funkcji składowanych oraz triggerów. Jest tak za sprawą tego, iż MySQL aż do wersji 5 nie posiadał takich możliwości i aktualnie są one niejako nowością. Istnieje bardzo duża rozbieżność w tworzeniu i funkcjonowaniu procedur oraz funkcji w zależności od konkretnej wersji MySQL. W przypadku bazy danych PostgreSQL procedury oraz funkcje posiadają bogatą historię, w związku z czym pracują zdecydowanie bardziej stabilnie, a ich działanie w zdecydowanie mniejszym stopniu zależy od konkretnej wersji serwera bazy danych. Omawiając funkcje składowane należy zauważyć również, iż PostgreSQL umożliwia przypisanie praktycznie dowolnej funkcji jako domyślną wartość pola w tabeli, podczas gdy MySQL dopuszcza użycie tylko funkcji *NOW()*, co jest znacznym ograniczeniem. Ostatnim czynnikiem, który miał wpływ na wybór bazy

PostgreSQL była domyślna obsługa wyszukiwania pełnotekstowego z wykorzystaniem Tsearch2 (we wcześniejszych wersjach instalowane jako osobny moduł). W przypadku MySQL wyszukiwanie pełnotekstowe również jest dostępne ale tylko dla mechanizmu składowania danych MyISAM, który wyklucza inne istotne funkcjonalności takie jak m.in. obsługa transakcji. Konieczność wyboru „ważniejszej” funkcjonalności wpłynęło negatywnie na ocenę MySQL jako serwera bazy danych.

Testowy serwis internetowy to aplikacja z rodziny e-commerce będąca stosunkowo prostym sklepem internetowym. Posiada on budowę oraz układ poszczególnych komponentów typowe dla większości podobnych rozwiązań dostępnych na rynku (zob. rys. 22).



Rysunek 22. Wygląd testowego serwisu internetowego  
 Źródło: Opracowanie własne

Istotną cechą systemu jest możliwość prowadzenia sprzedaży w kilku wersjach językowych i z wykorzystaniem kilku różnych walut. Wpływa to w sposób znaczący na schemat tabel w bazie danych, ponieważ większość z nich musi zostać zdublowana w celu uwzględnienia tłumaczeń dla poszczególnych wersji językowych. W wyniku tego zdecydowana część zapytań kierowanych do bazy danych opiera się na złączeniach tabel.

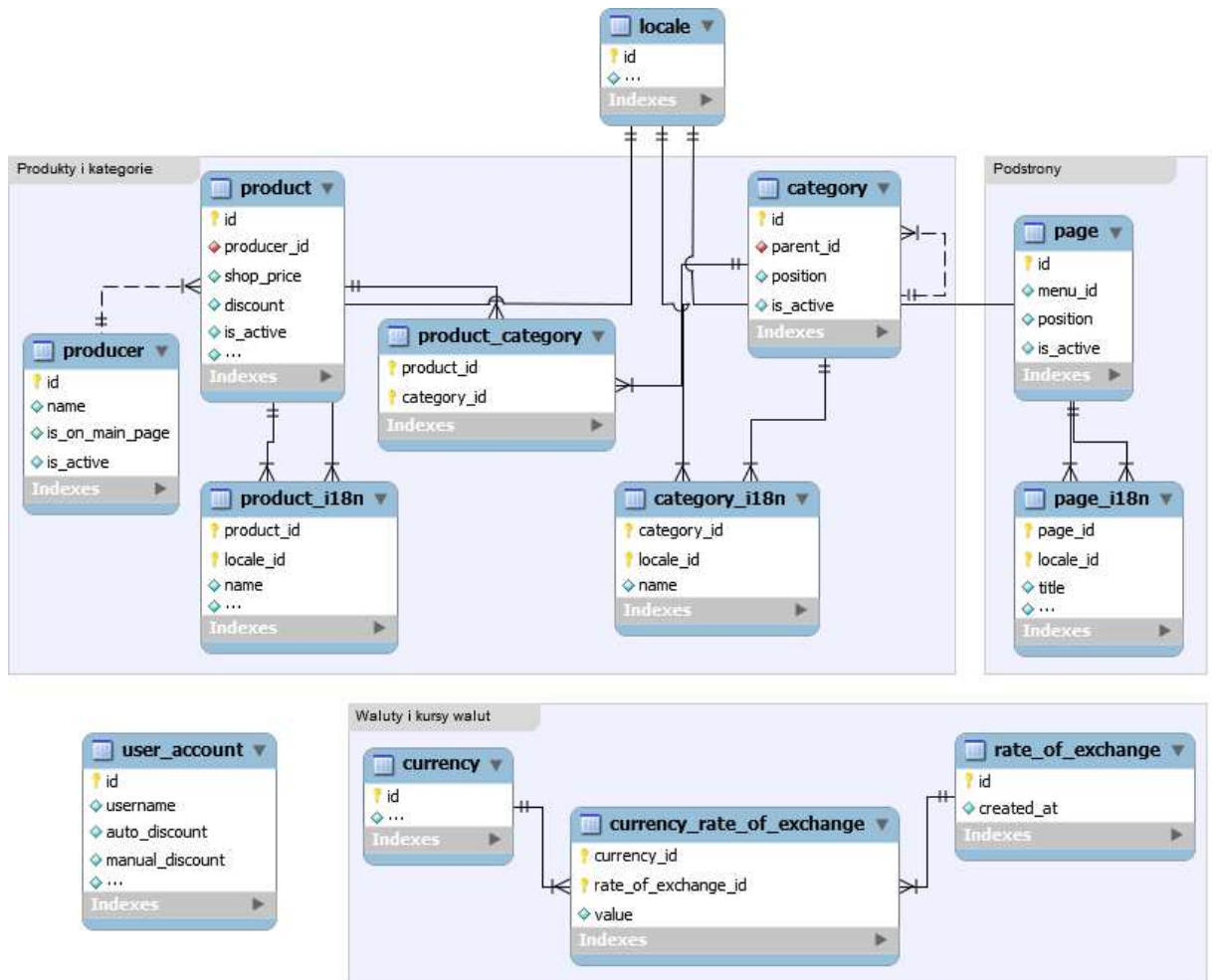
Kolejną istotną cechą jest umożliwienie w maksymalny sposób zarządzania treścią oraz danymi z poziomu panelu administracyjnego. Skutkuje to w tym, iż praktycznie każdy element sklepu generowany jest na podstawie bazy danych co, w połączeniu z wieloma wersjami językowymi, prowadzi do generowania sporej liczby zapytań do bazy danych.

Analizując rysunek 22 przedstawiający wygląd testowego serwisu internetowego można sporządzić listę elementów, które muszą być tworzone na podstawie bazy danych:

- wybór wersji językowej,
- wybór waluty,
- menu górne oraz menu dolne (4 osobne segmenty),
- drzewo kategorii,
- lista producentów,
- liczba produktów w koszyku wraz z podsumowaniem jego łącznej wartości,
- *breadcrumb* czyli ścieżka aktualnie przeglądanej kategorii,
- lista produktów z danej kategorii.

W przypadku wystąpienia jakichkolwiek problemów z wydajnością sklepu podejrzenia powinny więc paść w pierwszej kolejności właśnie na te elementy, gdyż to komunikacja z bazą danych jest najczęstszą przyczyną niskiej wydajności całego serwisu internetowego.

Schemat bazy danych testowego serwisu internetowego składa się z kilkudziesięciu tabel, z których najważniejsze (czyli te, z których pobierane będą dane podczas testu) to m.in. tabela produktów, kategorii oraz kursów walut (zob. rys. 23).



Rysunek 23. Schemat bazy danych testowego serwisu internetowego  
 Źródło: Opracowanie własne

Wykonywany test wydajności serwisu internetowego będzie dotyczył cyklicznego wyświetlania strony prezentującej produkty przypisane do wybranej kategorii i będzie składał się z dwóch części. W pierwszej do serwera wysyłane będzie 50 żądań rozdzielonych na 5 partii co zasymuluje natężenie jednoczesnych odwiedzin użytkowników na poziomie 10 użytkowników. Drugi test będzie polegał na wysyłaniu 25 konkurencyjnych żądań w 20 partiach ale z ograniczeniem czasowym 60 sekund na cały test.

Tabele bazy danych zostaną wypełnione następującą liczbą danych:

- 3 wersje językowe,
- 100 000 produktów (oraz 300 000 rekordów dla wersji językowych),
- 100 kategorii (oraz 300 rekordów dla wersji językowych),
- 500 000 przypisań produkt-kategoria (każdy produkt będzie należał do 5 kategorii),
- 500 producentów,

- 50 podstron (oraz 150 rekordów dla wersji językowych),
- 100 000 użytkowników,
- 3 waluty,
- 1 000 kursów walut (oraz 3 000 wartości kursów walut).

Wszystkie dane zostaną wygenerowane w sposób losowy za pomocą specjalnie do tego celu napisanych skryptów PHP.

Środowisko testowe, w którym odbywać się będą testy wydajnościowego serwisu internetowego, to komputer przenośny HP 510 o następujących parametrach:

- processor Intel Pentium M 2.26 GHz,
- 1GB pamięci RAM.

Na komputerze tym zainstalowany został system operacyjny Windows XP, a w celu uruchomienia środowiska zbliżonego do serwera zainstalowano:

- Apache (2.2.8),
- PHP (5.2.6) jako moduł,
- PostgreSQL (8.3.1).

W tym miejscu należy zaznaczyć, iż wyniki, które będą uzyskane w czasie trwania testu, nie mogą w żadnym stopniu być utożsamiane z faktycznymi, tj. z takimi, które zostałyby uzyskane na prawdziwym serwerze. Wyniki te mają jedynie charakter pogładowy.

Podczas wykonywania testów pozamykane zostaną zbędne aplikacje, a także zakończono zostaną procesy, które mogłyby zafałszować uzyskane wyniki. Każdy test zostanie wykonany kilkakrotnie aż do momentu ustabilizowania się otrzymywanych wyników. Do wykonania testów wykorzystany zostanie wbudowany we framework Kohana profiler (bardzo zbliżony do opisywanej wcześniej klasy Benchmark pakietu PEAR) oraz narzędzie ApacheBench.

## **4.2. Wyjściowa wersja testowego serwisu internetowego**

Pozornie wyjściowa wersja testowego serwisu internetowego wygląda poprawnie. Patrząc na schemat bazy danych nie można mieć do niego większych zastrzeżeń, tabele posiadają pozakładane odpowiednie indeksy, wykorzystanie szybkiego frameworka do oprogramowania serwisu również powinno budzić zaufanie. Samo przeglądanie asortymentu

testowego sklepu internetowego również nie nastęca żadnych problemów; system wydaje się chodzić dość szybko i stabilnie.

Przeglądając kody źródłowe można jednak mieć pewne obawy co do wydajności testowego serwisu internetowego. Największe zastrzeżenia może budzić kod odpowiedzialny za budowę drzewa kategorii pełniącego rolę menu nawigacyjnego po kategoriach sklepu. Jest ono bowiem budowane z wykorzystaniem rekurencyjnych zapytań do bazy danych (zob. przykł. 22).

### Przykład 22

```
public static function html($parent_id = 0) {
    $return = '';

    $mdl_category = new Category_Model();
    $list = $mdl_category->get_childs($parent_id);

    if (count($list) > 0) {
        $return .= '<ul>';

        foreach ($list as $c) {
            $return .=
            '<li>'.html::anchor('category/'. $c['id']. '/' . url::title($c['name']), $c['name'],
            array('title' => $c['name']));

            $return .= self::html($c['id']);

            $return .= '</li>';
        }

        $return .= '</ul>';
    }

    return $return;
}
```

Widać wyraźnie rekurencyjne wywołanie metody *self::html()*, co wiąże się każdorazowo z wykonaniem metody *get\_childs()* klasy *Category\_Model* (zob. przykł. 23), w celu pobrania kategorii będących podkategoriami tej, która aktualnie jest iterowana.

### Przykład 23

```
public function get_childs($parent_id) {
    $this->db
        ->select('category.id, category.parent_id')
        ->select('category_il8n.name')
```

```

        ->from('category')
        ->join('category_il8n', array('category.id' =>
'category_il8n.category_id'))
        ->where('category.parent_id', $parent_id)
        ->where('category_il8n.locale_id', locale::get('id'))
        ->where('category.is_active', pgsql::btos(TRUE))
        ->orderby('category.position', 'ASC')
    ;

    return $this->db->get()->result_array(FALSE);
}

```

Taki sposób generowania drzewa kategorii może prowadzić do drastycznego wzrostu liczby zapytań wyrażonej wzorem  $N+1$ , gdzie  $N$  to liczba kategorii.

W podobny sposób, tj. z wykorzystaniem rekurencyjnych zapytań zostało zaprogramowane pobieranie danych potrzebnych do wyświetlenia *breadcrumb*, czyli ścieżki aktualnie przeglądanej kategorii (zob. przykł. 24).

#### Przykład 24

```

public static function get_category_path($category_id) {
    $return = array();

    $db = Database::instance();

    while(TRUE) {
        $db
            ->select('category.id, category.parent_id')
            ->select('category_il8n.name')
            ->from('category')
            ->join('category_il8n', array('category.id' =>
'category_il8n.category_id'))
            ->where('category.id', $category_id)
            ->where('category_il8n.locale_id', locale::get('id'))
            ->where('category.is_active', pgsql::btos(TRUE))
            ->limit(1)
        ;

        $r = $db->get()->result_array(FALSE);

        if (isset($r[0])) {
            $return[] = $r[0];

            $category_id = $r[0]['parent_id'];
        }
    }
}

```

```

    }
    else {
        break;
    }
}

return $return;
}

```

Podobnie jak w przypadku kodu tworzącego drzewo kategorii, tak i w tym przypadku występuje duży narzut liczby wykonywanych zapytań. Liczba ta będzie również określona wzorem  $N+1$ , gdzie  $N$  to poziom zagłębienia kategorii wyjściowej.

Pewien niepokój może także wzbudzać kod odpowiedzialny za podsumowanie wartości koszyka zakupów (zob. przykł. 25).

#### Przykład 25

```

public function get_total_sum() {
    $sum = 0;

    $mdl_product = new Product_Model();

    foreach ($this->products as $k => $i) {
        $sum += $mdl_product->get_product_price($k) * $i;
    }

    return $sum;
}

```

Można od razu zauważyć, iż skrypt będzie pobierał cenę dla każdego produktu z osobna, co będzie wiązać się z liczbą zapytań do bazy danych równą  $N$ , gdzie  $N$  to liczba produktów aktualnie przechowywanych w koszyku. Zagłębiając się dalej w kod związany z podsumowaniem wartości koszyka można zauważyć, iż metoda *get\_product\_price()* klasy *Product\_Model* również będzie mogła sprawiać pewne kłopoty (zob. przykł. 26).

#### Przykład 26

```

public function get_product_price($product_id) {
    $q = 'SELECT a.value FROM `.$this->table_prefix.`currency_rate_of_exchange a
LEFT JOIN `.$this->table_prefix.`currency b ON a.currency_id=b.id WHERE
a.currency_id=.`currency::get(`id`).` AND a.rate_of_exchange_id=(SELECT MAX(id)
FROM `.$this->table_prefix.`rate_of_exchange)';
}

```

```

    $currency_rate = $this->db->query($q)->current()->value;

    $session = Session::instance();
    $user_id = (int)$session->get_user_data('id');

    $q = `SELECT GREATEST(auto_discount, manual_discount) AS discount FROM
    `.$this->table_prefix.`user_account WHERE id=`.`.$user_id;
    $r = $this->db->query($q)->result_array();

    if (is_array($r) && isset($r[0])) {
        $user_discount = $r[0]->discount;
    }
    else {
        $user_discount = 0;
    }

    $this->db
        ->select(`((`.`.$this->table_prefix.`product.shop_price - (`.`.$this-
->table_prefix.`product.shop_price * `.$this->table_prefix.`product.discount /
100)) / `.$currency_rate.`) * `.`.(100 - $user_discount) / 100).` AS
final_price`, FALSE)
        ->from('product')
        ->where('id', $product_id)
        ->where('is_active', pgsql::btos(TRUE))
        ->limit(1)
    ;

    $r = $this->db->get()->result_array(FALSE);

    return isset($r[0]) ? $r[0]['final_price'] : 0;
}

```

W metodzie tej, prócz pobrania ceny produktu, wywoływane są dodatkowo 2 zapytania, które mają na celu pobranie aktualnego kursu dla wybranej przez użytkownika waluty oraz wartości rabatu, jaki został przyznany użytkownikowi (zapytanie wykonywane nawet jeśli użytkownik nie jest zalogowany). W wyniku takiej budowy metody *get\_product\_price()* klasy *Product\_Model* liczba zapytań potrzebnych do obliczenia łącznej wartości produktów znajdujących się w koszyku wyrażona będzie poprzez  $N*3$ , gdzie  $N$  to liczba produktów w koszyku.

Analizując kody źródłowe testowego serwisu internetowego można również zauważyć, iż nie został zastosowany żaden mechanizm *cache*. Wszystkie dane pobierane są każdorazowo

z bazy danych, co również będzie mogło mieć negatywny wpływ na wydajność całego systemu.

### 4.3. Pomiar wydajności testowego serwisu internetowego

Pierwsza część testu wydajności serwisu internetowego nie przebiegła pomyślnie, a uzyskane wyniki były bardzo słabe (zob. przykł. 27).

#### Przykład 27

```
Concurrency Level:      10
Time taken for tests:   133.64203 seconds
Complete requests:     50
Failed requests:       28
    (Connect: 0, Length: 28, Exceptions: 0)
Write errors:           0
Total transferred:     1626650 bytes
HTML transferred:     1582500 bytes
Requests per second:   0.38 [#/sec] (mean)
Time per request:      26612.842 [ms] (mean)
Time per request:      2661.284 [ms] (mean, across all concurrent requests)
```

Ponad połowa żądań kierowanych do serwera nie została w ogóle obsłużona. 28 żądań zostało odrzuconych, natomiast jedynie 22 zostały zrealizowane. Dla użytkowników oznacza to tyle, że tylko co drugi z nich będzie w stanie korzystać z testowego serwisu internetowego. Liczba żądań realizowanych w przeciągu 1 sekundy wynosiła 0.38 żądania. Należy w tym miejscu zauważyć, iż liczba ta dotyczy wszystkich żądań, nie tylko tych faktycznie zrealizowanych. Można więc przyjąć, że rzeczywista wartość tego wskaźnika powinna zostać zmniejszona do wartości ok. 0.17 żądań na sekundę.

Wyniki drugiego testu wydają się prezentować nieco lepiej, a to za sprawą wprowadzenia ograniczenia czasowego na wykonywany test (zob. przykł. 28).

#### Przykład 28

```
Concurrency Level:      25
Time taken for tests:   73.906250 seconds
Complete requests:     28
Failed requests:       0
Write errors:           0
Total transferred:     915304 bytes
HTML transferred:     889697 bytes
```

```
Requests per second: 0.38 [#/sec] (mean)
Time per request: 65987.723 [ms] (mean)
Time per request: 2639.509 [ms] (mean, across all concurrent requests)
```

W tym przypadku wszystkie żądania zostały poprawnie zrealizowane, a liczba żądań realizowanych w ciągu sekundy wyniosła dokładnie tyle samo co w pierwszym teście. Świadczyć to może o tym, iż wraz ze wzrostem natężenia odwiedzin użytkowników system będzie miał coraz większe kłopoty z ich prawidłowym obsłużeniem.

Używając Profilera wbudowanego we framework Kohana można zmierzyć czas wykonywania poszczególnych fragmentów kodu oraz sprawdzić ile oraz jakie zapytania kierowane są do bazy danych (zob. rys. 24).

Benchmarks	Time	Memory
Kohana Loading	0.068	0.49MB
Environment Setup	0.027	0.08MB
System Initialization	0.512	1.08MB
Controller Setup	0.037	0.27MB
Controller Execution	1.473	0.72MB
Breadcrumb	0.011	0.04MB
Wersje Językowe Oraz Waluty	0.030	0.15MB
Drzewo Kategorii	0.164	0.10MB
Lista Producentów	0.370	0.05MB
Total Execution	2.099	2.57MB

Queries	Time	Rows
Total: 136	0.950	401

Rysunek 24. Profiler dla wyjściowej wersji testowego serwisu internetowego  
Źródło: Opracowanie własne

Liczba zapytań kierowanych do bazy danych jest wręcz zatrważająca. 136 zapytań potrafi skutecznie spowolnić nawet najlepiej zaprojektowaną bazę danych. Ponadto zauważyć można, iż samo generowanie strony zajmuje nieco ponad 2 sekundy. Najwięcej czasu zajmuje wykonanie kontrolera, gdyż to właśnie on jest odpowiedzialny za pobranie (z modeli) oraz przekazanie (do widoku) odpowiednich danych. Pewnym zaskoczeniem może być fakt, iż pobranie drzewa kategorii trwa zdecydowanie krócej niż pobranie listy producentów.

#### 4.4. Optymalizacja wydajności testowego serwisu internetowego

Zajmując się optymalizacją wydajności testowego serwisu internetowego skupiono się na eliminacji zbędnych zapytań kierowanych do bazy danych. To właśnie w tym miejscu upatrywano szanse na zwiększenie wydajności całego systemu.

Zmieniając koncepcję tworzenia drzewa kategorii wyeliminowano dwa najdrażliwsze miejsca w testowym serwisie internetowym. Owa koncepcja miała teraz polegać na *jednorazowym* pobraniu zawartości całej tabeli, odpowiednim jej przetworzeniu oraz zapisaniu w takim miejscu, aby zarówno kod odpowiedzialny za wyświetlanie drzewa kategorii jak i kod odpowiedzialny za wyświetlanie ścieżki kategorii, mogły bez problemu korzystać z tych danych (zob. przykł. 29).

#### Przykład 29

```
public function get_tree() {
    $return = array();

    $q = $this->db
        ->select('category.id, category.parent_id, category_il8n.name')
        ->select('(SELECT COUNT(product_id) FROM `.$this->table_prefix.`product_category WHERE category_id=id) AS count_product', FALSE)
        ->from('category')
        ->join('category_il8n', array('category.id' => 'category_il8n.category_id'), null, 'INNER')
        ->where('category.is_active', pgsql::btos(TRUE))
        ->where('category_il8n.locale_id', locale::get('id'))
        ->orderby('category.parent_id', 'ASC')
        ->orderby('category.position', 'ASC');

    $categories = $q->get()->result_array(false);

    foreach ($categories as $c) {
        if (!isset($return[$c['parent_id']]) || !is_array($return[$c['parent_id']])) {
            $return[$c['parent_id']] = array();
        }

        $return[$c['parent_id']][] = $c;
    }

    return $return;
}
```

Zapisując listę kategorii w tablicy dwuwymiarowej znacznie zredukowano liczbę zapytań. Tablica ta posiada budowę, która ułatwia poruszanie się po jej strukturze (zob. przykł. 30).

### Przykład 30

```
(int)parent_id => (array)items
```

Kluczem dla poszczególnych elementów jest identyfikator kategorii nadrzędnej, natomiast wartością jest tablica kategorii znajdujących się na danym poziomie.

Utworzona w ten sposób tablica może być z powodzeniem wykorzystywana przy tworzeniu drzewa kategorii (zob. przykł. 31).

### Przykład 31

```
public static function html(&$categories, $parent_id = 0) {
    $return = '';

    if (isset($categories[$parent_id]) && is_array($categories[$parent_id])) {
        $return .= '<ul>';

        foreach ($categories[$parent_id] as $c) {
            $return .=
            '<li>'.html::anchor('category/'. $c['id']. '/' . $c['url']::title($c['name']), $c['name'],
            array('title' => $c['name'])).' ('.$c['count_product'].)';

            if (isset($categories[$c['id']]) &&
            is_array($categories[$c['id']])) {
                $return .= self::html($categories, $c['id']);
            }

            $return .= '</li>';
        }

        $return .= '</ul>';
    }

    return $return;
}
```

Jako pierwszy parametr metody przekazywana jest (przez referencję) utworzona wcześniej tablica. Z wykorzystaniem pętli *foreach* następuje iterowanie po poszczególnych elementach tablicy w celu wypisania konkretnej gałęzi drzewa. Równocześnie nie następują kolejne rekurencyjne wywoływania jeśli dana kategoria nie ma żadnej podkategorii (można to stwierdzić dzięki strukturze tablicy).

Ta sama tablica została również wykorzystana do określenia ścieżki, pod którą znajduje się aktualnie przeglądana kategoria (zob. przykł. 32).

### Przykład 32

```
public static function get_category_path(&$category_tree, $category_id) {
    $return = array();

    foreach ($category_tree as $level1) {
        foreach ($level1 as $v) {
            if ($v['id'] == $category_id) {
                $return[] = $v;

                if ($v['parent_id'] != 0) {
                    $return = array_merge($return,
category_tree::get_category_path($category_tree, $v['parent_id']));
                }

                break;
            }
        }
    }

    return $return;
}
```

Tym razem kod polega na iterowaniu po elementach tablicy „od góry”, tj. od kategorii najgłębiej zagnieżdżonej, czyli od tej, do której ścieżkę chcemy poznać. Skrypt przerywa działanie jeżeli dojdzie do sytuacji, gdy w całej tablicy nie zostanie znaleziona kategoria o danym identyfikatorze.

Kolejnym zabiegiem, który zastosowano w celu poprawy wydajności testowego serwisu internetowego, jest przeniesienie często wykorzystywanych danych do listy parametrów konfiguracyjnych, których wartości są ustalane dynamicznie. Do danych tych można zaliczyć:

- listę wersji językowych,
- listę walut,
- wartość kursu aktualnie wybranej waluty,
- rabat przyznany aktualnemu użytkownikowi.

Jako elementy dostępne z dowolnego miejsca systemu mogą one być bez przeszkód wykorzystywane. Doskonałym przykładem tego może być wykorzystanie wartości kursu

aktualnie wybranej waluty oraz rabatu przyznanego aktualnemu użytkownikowi podczas określania sumarycznej wartości towarów, które znajdują się w koszyku (zob. przykł. 33).

### Przykład 33

```
public function get_products_price(array $ids) {
    $currency_rate = currency::get('rate');
    $user_discount = user::get_discount_value();

    $return = array();

    $this->db
        ->select('id')
        ->select('((\'.'.$this->table_prefix.'product.shop_price - (\'.'.$this->table_prefix.'product.shop_price * \\'.'.$this->table_prefix.'product.discount / 100)) / \\'.'.$currency_rate.'') * \\'.'.$user_discount.' AS final_price', FALSE)
        ->from('product')
        ->in('id', $ids)
        ->where('is_active', pgsql::btos(TRUE))
    ;

    $r = $this->db->get()->result_array(FALSE);

    foreach ($r as $item) {
        $return[$item['id']] = $item['final_price'];
    }

    return $return;
}
```

Równocześnie należy zauważyć, iż zmianie uległo samo zapytanie kierowane do bazy danych. Aktualnie jednorazowo pobierane są ceny wszystkich produktów, które znajdują się w koszyku. Metoda *75ro\_products\_price()* przyjmuje obecnie jako parametr nie identyfikator pojedynczego produktu, a tablicę z identyfikatorami produktów.

Wprowadzając te oraz inne drobne modyfikacje zdecydowanie się także na wprowadzenie mechanizmu cache, który objął takie elementy jak:

- listę wersji językowych,
- listę walut,
- listę pozycji menu podstron,
- listę producentów,

- tablicę z drzewem kategorii.

Zapis danych do cache ma na celu maksymalne ograniczenie liczby zapytań kierowanych do bazy danych. Dane, które nie ulegają zmianom zbyt często mogą zostać zapisane w pliku na dysku serwera i być z niego pobierane. Komunikacja z bazą danych polegająca na pobieraniu za każdym razem tych samych danych wydaje się być nie najlepszym pomysłem.

#### 4.5. Pomiar wydajności zoptymalizowanego serwisu internetowego

Dokonując ponownego pomiaru wydajności testowego serwisu internetowego odnotowano znacznie lepsze wyniki, które świadczą o tym, że wprowadzone zmiany warte były poświęconego czasu oraz pracy (zob. przykł. 34).

##### Przykład 34

```
Concurrency Level:      10
Time taken for tests:   30.578125 seconds
Complete requests:     50
Failed requests:       0
Write errors:          0
Total transferred:     2699144 bytes
HTML transferred:     2654953 bytes
Requests per second:   1.64 [#/sec] (mean)
Time per request:      6115.625 [ms] (mean)
Time per request:      611.563 [ms] (mean, across all concurrent requests)
```

Czas, w którym został wykonany pierwszy test, wyniósł ok. 30.5 sekundy, co można uznać za wartość satysfakcjonującą. Jednak bardziej istotną wartością uzyskaną w tym teście jest brak nieobsłużonych żądań co oznacza, że wszystkie żądania zostały poprawnie zrealizowane. Równocześnie należy zauważyć, iż liczba żądań możliwych do zrealizowania w czasie 1 sekundy kształtuje się na wartości 1.64 żądania.

Wyniki uzyskane po przeprowadzeniu drugiego testu także można potraktować jako sukces (zob. przykł. 35).

##### Przykład 35

```
Concurrency Level:      25
Time taken for tests:   60.125000 seconds
Complete requests:     89
Failed requests:       0
Write errors:          0
```

```

Total transferred:      4900893 bytes
HTML transferred:      4821423 bytes
Requests per second:    1.48 [#/sec] (mean)
Time per request:       16889.045 [ms] (mean)
Time per request:       675.562 [ms] (mean, across all concurrent requests)

```

W przeciągu 60 sekund system był w stanie obsłużyć 89 żądań przy równoczesnym natężeniu 25 konkurencyjnych żądań. Liczba żądań, które testowy serwis internetowy może zrealizować w przeciągu 1 sekundy nieco spadła w porównaniu do pierwszego testu ale nadal oscyluje w granicach 1.5 żądania na sekundę.

Zapoznając się z danymi uzyskanymi dzięki profilerowi można zauważyć istotną zmianę w stosunku do wyjściowej wersji testowego serwisu internetowego (zob. rys. 25).

Benchmarks	Time	Memory
Kohana Loading	0.024	0.53MB
Environment Setup	0.012	0.12MB
System Initialization	0.060	0.50MB
Controller Setup	0.023	0.33MB
Controller Execution	0.452	1.18MB
Wersje Językowe Oraz Waluty	0.021	0.15MB
Drzewo Kategorii	0.047	0.05MB
Lista Producentów	0.020	0.02MB
Total Execution	0.569	2.55MB
Queries	Time	Rows
Total: 4	0.231	26

Rysunek 25. Profiler dla zoptymalizowanej wersji testowego serwisu internetowego  
Źródło: Opracowanie własne

Liczba zapytań kierowanych do bazy danych została ograniczona do 4 zapytań. Główną przyczyną tak gwałtownego spadku tej liczby jest zastosowanie mechanizmu cache. Dane, które nie ulegają częstym modyfikacjom zostały zapisane w odpowiedniej formie na serwerze. Czas wykonania samego kontrolera również uległ znacznemu skróceniu, dzięki czemu cały system funkcjonuje zdecydowanie szybciej.

#### 4.6. Porównanie wyników testów

Wyniki uzyskane w testach przeprowadzonych przed i po dokonaniu optymalizacji wydajności testowego serwisu internetowego odbiegają od siebie w sposób znaczący. Poprawnie uległy praktycznie wszystkie parametry wydajnościowe systemu (zob. tab. 5, 6 i 7).

Tabela 5. Porównanie wyników testów końcowych (test 1)

Parametr:	Przed optymalizacją:	Po optymalizacji:	Poprawa o:
Czas wykonania testu:	133.64203 s.	30.578125 s.	77 %
Liczba zrealizowanych żądań:	22	50	127 %
Żądania zakończone błędem:	56 %	0 %	100 %
Liczba żądań na sekundę:	0.38 (0.17)	1.64	332 % (865 %)

Tabela 6. Porównanie wyników testów końcowych (test 2)

Parametr:	Przed optymalizacją:	Po optymalizacji:	Poprawa o:
Liczba zrealizowanych żądań:	28	89	218 %
Liczba żądań na sekundę:	0.38	1.48	289 %

Tabela 7. Porównanie wyników testów końcowych (profiler)

Parametr:	Przed optymalizacją:	Po optymalizacji:	Poprawa o:
Czas wykonywania kontrolera:	1.473 s.	0.452 s.	69 %
Czas generowania strony:	2.099 s.	0.569 s.	73 %
Liczba zapytań do bazy danych:	136	4	97 %

Optymalizacja wydajności testowego serwisu internetowego przyniosła spore korzyści związane ze zmniejszeniem obciążenia generowanego przez serwis internetowy. Relatywnie niewielkim nakładem poświęconego czasu oraz pracy dokonano znaczącej poprawy funkcjonowania serwisu internetowego. Dzięki temu zmniejszono obciążenie serwera (zarówno Apache jak i bazy danych PostgreSQL), a co za tym idzie zwiększono możliwości obsłużenia większej liczby użytkowników. Umiejętne spojrzenie na kod jako całość pozwoliło wyeliminować nadmierne obciążenia generowane przez system. Bez konieczności inwestycji w dodatkowy sprzęt osiągnięto to, co powinno być celem już na początku tworzenia oprogramowania serwisu internetowego.

## ZAKOŃCZENIE

Optymalizacja wydajności serwisów internetowych jest jednym z ważniejszych czynników wpływających na ich prawidłowe funkcjonowanie. Odpowiednie zaprojektowanie oraz wykonanie ich od strony programistycznej, a także zapewnienie dobrego zaplecza techniczno-sprzętowego powinno więc być stawiane jako cel nadrzędny przy realizacji każdego projektu serwisu internetowego. Możliwość obsługi wielu użytkowników w niewielkich jednostkach czasu, zwiększenie popularności, odwiedzalności i dostępności, a równocześnie zmniejszenie kosztów utrzymania to główne korzyści, jakie można uzyskać poprzez odpowiednią optymalizację wydajności serwisów internetowych. Optymalizacja wydajności serwisów internetowych ma również duże znaczenie w ujęciu globalnym, ponieważ pozwala w pewnym, choćby niewielkim, stopniu zredukować obciążenie całej sieci Internet.

Patrząc od strony programistycznej ową wydajność serwisów internetowych można osiągnąć w głównej mierze poprzez wybór odpowiedniego języka programowania oraz systemu bazy danych. Ogromne znaczenie ma tutaj również stosowanie właściwych algorytmów oraz struktur danych. Korzystanie z funkcji, które w mniejszym stopniu obciążają procesor serwera, umiejętne stosowanie różnych konstrukcji języka programowania, odpowiedni schemat bazy danych (ze szczególnym uwzględnieniem prawidłowo założonych indeksów) odgrywają tutaj kluczowe znaczenie. Bardzo ważnym elementem, bez którego osiągnięcie odpowiedniej wydajności serwisu internetowego jest wręcz niemożliwe, jest mechanizm cache, czyli zapis na dysku serwera lub w jego pamięci danych, które rzadko ulegają zmianom.

Stosując relatywnie proste zabiegi, polegające na przebudowie newralgicznych punktów systemu, można w znacznym stopniu przyspieszyć pracę całego serwisu internetowego. Posiadając wiedzę z zakresu optymalizacji wydajności serwisów internetowych można również uniknąć powstawania tzw. wąskich gardeł już w fazie samego projektowania systemu. Należy przy tym jednak zachować umiar i rozsądek oraz mieć na uwadze stwierdzenie Donalda Knutha mówiące, iż „wstępna optymalizacja jest źródłem wszelkiego zła”.

Jako główny cel niniejszej pracy została postawiona optymalizacja wydajności autorskiego systemu sklepu internetowego. Po zastosowaniu wybranych technik opisanych w pracy uzyskano znaczną poprawę wydajności testowego serwisu internetowego. Można więc stwierdzić, iż cel pracy został osiągnięty.

W przypadku gdy przedstawione w pracy metody optymalizacji wydajności serwisów internetowych okazałyby się niewystarczające, wówczas należałoby dodatkowo skupić się m.in. na odpowiedniej konfiguracji oprogramowania, przy użyciu którego funkcjonuje serwis internetowy, a także na ewentualnej modernizacji zaplecza technicznego. Zadbanie o odpowiednią konfigurację serwera powinno przynieść wymierne korzyści w postaci zwiększenia wydajności serwisu internetowego.

## LITERATURA

- [ApacheBench 2008] Apache Bench  
<http://httpd.apache.org/docs/2.2/programs/ab.html>  
Data dostępu: 09.21.2008
- [Argreich 2003] Argerich L., Choi W., Coggershall J., Egervari K., Giesler M., Greant Z., Hill A., Hubbard Ch., Moore J., O'Dell D., Parise J., Rawat H., Sani T., Scollo Ch., Thomas D., Ullman Ch.: *PHP4. Zaawansowane programowanie*, wyd. Helion, Gliwice, 2003.
- [Bagui 2007] Bagui S., Earp R.: *SQL dla SQL Server 2005. Wprowadzenie*, wyd. Helion, Gliwice, 2003.
- [Dyer 2005] Dyer R.: *MySQL. Almanach*, wyd. Helion, Gliwice, 2005.
- [Jmeter 2008] Jmeter  
<http://jakarta.apache.org/jmeter>  
Data dostępu: 09.21.2008
- [Jmeter vs ApacheBench 2008] Różnice w wynikach między Jmeter, a ApacheBench  
<http://osdir.com/ml/jakarta.jmeter.user/2003-09/msg00140.html>  
Data dostępu: 09.21.2008
- [Kłopotek 2001] Kłopotek M. A.: *Inteligentne wyszukiwarki internetowe*, Akademicka Oficyna Wydawnicza EXIT, Warszawa 2001.
- [Kohana informacje 2009] Ogólne informacje o frameworku Kohana  
<http://kohanaphp.pl/home>  
Data dostępu: 27.02.2009
- [Lecky-Thompson 2005] Lecky-Thompson E., Eide-Goodman H., Nowicki S., Cove A.: *PHP5. Zaawansowane programowanie*, wyd. Helion, Gliwice 2005.
- [Lesiak 2008] Porównanie wydajności 4 systemów bazodanowych  
<http://adamlesiak.com/articleview/3>  
Data dostępu: 21.12.2008
- [MySQL informacje 2009] Ogólne informacje o bazie danych MySQL  
<http://www.mysql.com/about/>  
Data dostępu: 03.02.2009
- [MySQL InnoDB 2009] Mechanizm składowania danych InnoDB w MySQL  
<http://dev.mysql.com/doc/refman/5.1/en/innodb.html>  
Data dostępu: 05.02.2009
- [MySQL MEMORY 2009] Mechanizm składowania danych MEMORY w MySQL  
<http://dev.mysql.com/doc/refman/5.0/en/memory-storage-engine.html>

- Data dostępu: 05.02.2009
- [MySQL MyISAM 2009] Mechanizm składowania danych MyISAM w MySQL  
<http://dev.mysql.com/doc/refman/5.0/en/myisam-storage-engine.html>  
 Data dostępu 05.02.2009
- [MySQL typy danych 2008] Typy danych w MySQL  
<http://dev.mysql.com/doc/refman/5.1/en/storage-requirements.html>  
 Data dostępu: 06.05.2008
- [MySQL vs PostgreSQL 2009] Różnice między MySQL, a PostgreSQL  
[http://www.wikivs.com/wiki/MySQL\\_vs\\_PostgreSQL](http://www.wikivs.com/wiki/MySQL_vs_PostgreSQL)  
 Data dostępu: 27.02.2009
- [Netcraft 2009] Statystyka liczby serwisów internetowych  
[http://news.netcraft.com/archives/web\\_server\\_survey.html](http://news.netcraft.com/archives/web_server_survey.html)  
 Data dostępu: 02.02.2009
- [Osiołki 2008] Wady tabelarycznego układu szablonu serwisu internetowego  
<http://osiolki.net/tabelki/index.html>  
 Data dostępu: 05.10.2008
- [Oracle informacje 2009] Ogólne informacje o bazie danych Oracle  
<http://www.oracle.com/global/pl/corporate/index.html>  
 Data dostępu: 03.02.2009
- [PHPBench 2008] Testy wydajności PHP  
<http://www.php.lt/benchmark/phpbench.php>  
 Data dostępu: 05.10.2008
- [PHP Benchmark 2008] Biblioteka Benchmark  
<http://pear.php.net/package/Benchmark>  
 Data dostępu: 09.21.2008
- [PostgreSQL 8.3 zmiany 2009] Zmiany w PostgreSQL 8.3  
<http://www.postgresql.org/about/press/features83>  
 Data dostępu: 28.02.2009
- [PostgreSQL informacje 2009] Ogólne informacje o bazie danych PostgreSQL  
<http://www.postgresql.org/about/>  
 Data dostępu: 03.02.2009
- [PostgreSQL typy danych 2008] Typy danych w PostgreSQL  
<http://www.postgresql.org/docs/8.3/interactive/datatype.html>  
 Data dostępu: 06.05.2008
- [SQL Server typy danych 2008] Typy danych w SQL Server  
<http://msdn2.microsoft.com/en-us/library/ms187752.aspx>  
 Data dostępu: 05.10.2008
- [Sroka 2005] Sroka H.: *Strategie i metodyka budowy systemów e-biznesu*,  
 Wydawnictwo Uczelniane Akademii Ekonomicznej im. Karola

Adameckiego, Katowice 2005.

- [Stones 2002] Stones R., Matthew N.: *Bazy danych i PostgreSQL. Od podstaw*, wyd. Helion, Gliwice 2002.
- [Urman 2007] Urman S., Hardman R., McLaughlin M.: *Oracle Database 10g. Programowanie w języku PL/SQL*, wyd. Helion, Gliwice, 2007
- [Vieira 2007] Vieira S.: *SQL Server 2005. Programowanie. Od podstaw*, wyd. Helion, Gliwice 2007.
- [Ziemba 2005] Ziemba E.: *Metodologia budowy serwisów internetowych dla zastosowań gospodarczych*, Wydawnictwo Uczelniane Akademii Ekonomicznej im. Karola Adameckiego, Katowice 2005.

## SPIS RYSUNKÓW I TABEL

### Spis tabel

TABELA 1. PORÓWNANIE WYDAJNOŚCI PĘTLI	29
TABELA 2. ROZMIARY WYBRANYCH TYPÓW DANYCH W BAZACH DANYCH	34
TABELA 3. ZYSKI WYDAJNOŚCIOWE PO ANALIZIE EXPLAIN SELECT...	47
TABELA 4. ANALIZA RAPORTU APACHEBENCH	54
TABELA 5. PORÓWNANIE WYNIKÓW TESTÓW KOŃCOWYCH (TEST 1)	78
TABELA 6. PORÓWNANIE WYNIKÓW TESTÓW KOŃCOWYCH (TEST 2)	78
TABELA 7. PORÓWNANIE WYNIKÓW TESTÓW KOŃCOWYCH (PROFILER)	78

### Spis rysunków

RYSUNEK 1. WYKRES SUMARYCZNEJ LICZBY SERWISÓW INTERNETOWYCH	10
RYSUNEK 2. SZYBKOŚĆ ODCZYTU DANYCH W MYSQL, POSTGRESQL, ORACLE I MICROSOFT SQL SERVER	14
RYSUNEK 3. REZULTAT WYSZUKIWANIA FRAZY „GAME ONLINE”	19
RYSUNEK 4. WYKRES POPULARNOŚCI STRON WYGENEROWANY DZIĘKI SERWISOWI ALEXA	20
RYSUNEK 5. WARSTWA PREZENTACJI DANYCH	24
RYSUNEK 6. NAJPOPULARNIEJSZY UKŁAD SERWISU INTERNETOWEGO	26
RYSUNEK 7. MENU ZDEFINIOWANE ZA POMOCĄ CSS	27
RYSUNEK 8. SCHEMAT ZNORMALIZOWANEJ BAZY DANYCH	36
RYSUNEK 9. SCHEMAT ZDENORMALIZOWANEJ BAZY DANYCH	36
RYSUNEK 10. DODANIE TABELI PRODUCT_NOTE DO PRZYKŁADOWEJ BAZY DANYCH	42
RYSUNEK 11. EXPLAIN SELECT... NR 1	44
RYSUNEK 12. ZMIANA KLUCZA GŁÓWNEGO W TABELI PRODUCT_NOTE	44
RYSUNEK 13. EXPLAIN SELECT... NR 2	45
RYSUNEK 14. EXPLAIN SELECT...NR 3	46
RYSUNEK 15. RAPORT BENCHMARK_PROFILER	49
RYSUNEK 16. RAPORT BENCHMARK_TIMER	51

RYSUNEK 17. TWORZENIE GRUPY WĄTKÓW W JMETER	56
RYSUNEK 18. OKREŚLANIE DOMYŚLNYCH USTAWIEŃ ŻĄDAŃ HTTP W JMETER	57
RYSUNEK 19. TWORZENIE NOWEGO ŻĄDANIA HTTP W JMETER	57
RYSUNEK 20. DODAWANIE GRAFU WYNIKÓW W JMETER	58
RYSUNEK 21. GRAF ŻĄDAŃ HTTP W JMETER	59
RYSUNEK 22. WYGLĄD TESTOWEGO SERWISU INTERNETOWEGO	62
RYSUNEK 23. SCHEMAT BAZY DANYCH TESTOWEGO SERWISU INTERNETOWEGO	64
RYSUNEK 24. PROFILER DLA WYJŚCIOWEJ WERSJI TESTOWEGO SERWISU INTERNETOWEGO	71
RYSUNEK 25. PROFILER DLA ZOPTYMALIZOWANEJ WERSJI TESTOWEGO SERWISU INTERNETOWEGO	77