STUDIA INFORMATICA 2009
Volume 30 Number 2B (84)

Dariusz KARPISZ
Politechnika Krakowska, Instytut Informatyki Stosowanej

DEDYKOWANY SYSTEM MONITORINGU INSTANCJI BAZY DANYCH DLA SYSTEMU ORACLE

Streszczenie. Sprawne utrzymanie systemów baz danych wymaga stosowania dodatkowych narzędzi do monitoringu serwera baz danych. Mimo bogatej oferty rynkowej takich systemów warto jest spróbować utworzyć własną aplikację o funkcjonalności wymaganej dla konkretnego przypadku użytkowania. W artykule przedstawiono system oparty na witrynie WWW przeznaczony do monitoringu instancji serwera Oracle 10g użytkowanej w ramach laboratorium baz danych.

Słowa kluczowe: monitoring, zarządzanie, administracja, DBA, Oracle

DEDICATED MONITORING SYSTEM OF INSTANCE FOR THE ORACLE DATABASE SERVER

Summary. Efficient maintain of database systems require additional tools for database server monitoring. Sometimes is better to use dedicated monitoring system than commercial tools. Dedicated application prepared for specific system will fulfill all required functions. In this paper web based system for monitoring instance of Oracle database, used for education purpose was presented.

Keywords: monitoring, management, administration, DBA, Oracle

1. Wprowadzenie

W ostatnich latach można zaobserwować znaczny rozwój systemów baz danych. Zarządzanie nimi nie jest prostym zadaniem, które administrator mógłby wykonać bez dodatkowych narzędzi. Podstawowe narzędzia, które ułatwiają administratorowi sprawne utrzymywanie dużych systemów baz danych to zaawansowane aplikacje do zarządzania i monitoringu. Takie systemy umożliwiają monitorowanie zarówno instancji bazy danych, jak

i serwera. Potrafią one analizować zebrane dane, np. podpowiadając czynności niezbędne do wykonania w przypadku awarii. Automatyzują pewne powtarzalne czynności, a nawet potrafią podjąć samodzielną decyzję i zrealizować ją na działającej instancji.

Na rynku są dostępne systemy realizujące wszystkie powyższe zadania. Jednak są one bardzo kosztowne oraz nie zawsze możliwe jest ich dostosowanie do specyficznych zastosowań. W zależności od platformy DBMS (ang. *Database Management System*) oraz systemu operacyjnego serwera/klienta możliwy jest wybór rozwiązań oferowanych przez dostawcę DBMS, np. *Oracle Enterprise Manager* będącego integralną częścią pakietu *Oracle Database* lub pozyskanie i adaptację narzędzi dostawców zewnętrznych (np. *TOAD*, *ManageEngine – Oracle Management*, *eG Oracle Monitor*).

Większość dostępnych na rynku systemów do monitoringu i zarządzania instancją bazy danych ma podobne funkcjonalności, z których najważniejsze to:

- informacje ogólne o systemie (stan instancji, SID, ...),
- stopień użycia procesora,
- stopień użycia przestrzeni składowania danych,
- ostrzeżenia,
- wskazówki diagnostyczne,
- aktywność użytkowników, sesje,
- wydajność plików danych,
- śledzenie zapytań i transakcji.

Udostępnianie dużej liczby funkcji z jednej strony jest zaletą, ale ten fakt można uznać również za wadę. Często, aby dotrzeć do właściwych informacji, należy przejść określoną ścieżkę z menu. Najczęściej należy również zalogować się jako użytkownik z uprawnieniami do schematu *SYSTEM*, co automatycznie wyklucza zastosowanie takich systemów do prezentacji on-line wybranych parametrów instancji.

W ramach niniejszego artykułu zostanie przedstawiona koncepcja i implementacja systemu do zarządzania serwerem baz danych *Oracle 10g*, obsługującym dużą liczba użytkowników, by umożliwić zastosowanie go do prowadzenia zajęć dydaktycznych z zakresu baz danych.

2. System OraLabManager

Przedstawione we wstępie motywacje dały podstawy do zbudowania systemu dedykowanego do monitorowania instancji bazy danych przeznaczonej do wykorzystania w trakcie zajęć dydaktycznych. W dalszej części artykułu zostanie omówiona implementacja takiego rozwiązania.

2.1. Motywacje do pracy nad dedykowanym systemem zarządzania i monitoringu

Wszystkie przeanalizowane systemy mimo rozbudowanej funkcjonalności nie spełniały wymagań, niezbędnych do użytkowania w przypadku udostępnienia systemu do wsparcia laboratorium baz danych. Żaden system nie zaoferował możliwości „hurtowego" tworzenia użytkowników z plików tekstowych, co znacznie przyspieszyłoby tworzenie kont dla studentów na początku każdego semestru. Również żaden system nie przedstawiał ogólnie dostępnej informacji na temat aktualnego stanu instancji, bez konieczności logowania, co mogłoby pomóc studentom w określeniu chwilowego obciążenia systemu i realizacji własnych zadań, np. z zakresu optymalizacji baz danych. Informacja taka jest niezbędna w tworzeniu własnych schematów testowych i wypełnianiu ich dużą ilością danych z generatorów *PL/SQL* (powyżej 1 miliona wierszy na tabelę). Najbardziej zbliżoną do oczekiwanej funkcjonalność oferuje system firmy Manage Engine, który udostępnia możliwość tworzenia grupy użytkowników z różnymi prawami dostępu. Oracle Enterprise Manager jest systemem dołączonym do Oracle Database, jednak jego użytkowanie jest zbyt skomplikowane i nie jest dopuszczalne udostępnianie jego funkcjonalności poza kompetencjami DBA (ang. *Database Administrator*). W innych przypadkach koszty licencji wydają się być zbyt wysokie. Dlatego też podjęto próbę realizacji systemu opierając się na bezpłatnych narzędziach posiadających funkcje niezbędne do ich adaptacji na potrzeby prowadzenia laboratorium baz danych.

2.2. Framework systemu OraLabManger

Do budowy docelowego systemu wykorzystano nowoczesne języki programowania z ich najnowszymi rozszerzeniami. Aplikacja została osadzona na platformie *Apache* jako witryna WWW. Do jej budowy wykorzystano między innymi takie technologie, jak np.:

- PHP5,
- sterowniki dostępu do danych PDO (PHP Data Object),
- szablony Smarty,
- bibliotekę generowania wykresów pChart,
- framework jQuery do generacji funkcjonalności JavaScript [8].

Do przygotowania systemu docelowego wykorzystano wzorzec projektowy MVC (*Model–View–Controller*). Według [1] wzorzec to pomysł, który okazał się użyteczny w jednym rzeczywistym kontekście i prawdopodobnie będzie użyteczny w innym. Taka definicja i jej implementacja w postaci aplikacji WWW, przy uproszczeniu, iż wzorzec to sprawdzone rozwiązanie, dla powtarzających się problemów, daje możliwość budowy elastycznego systemu gotowego do ciągłej ewaluacji [6]. Ze względu na założenie możliwości swobodnego ste-

rowania zakresem funkcjonalności użyto np. wcześniej wspomnianej technologii *jQuery* uniezależniającej kod *JavaScript* od rodzaju przeglądarki internetowej [8].

MVC jest obecnie najczęściej stosowanym wzorcem projektowym przy tworzeniu aplikacji internetowych. Swoją popularność zyskuje dzięki temu, iż wymusza podzielenie aplikacji na warstwy. Taki podział służy uporządkowaniu architektury systemu. Dzięki niemu zmiana w dowolnej części systemu nie musi powodować zmian w innych modułach. Zapewnia on również odseparowanie warstwy danych (modelu) od sposobu ich pobierania (kontrolera) oraz od sposobu ich prezentacji (widoku). Aby taka separacja była możliwa, obiekty w warstwie modelu nie mogą wpływać na sposób ich prezentacji.

Warstwa widoku powinna tylko i wyłącznie pokazywać wcześniej przygotowane dane, a kontroler w zależności od akcji użytkownika pobierać i przetwarzać dane pochodzące z modelu, a następnie przekazywać je do warstwy widoku w celu prezentacji użytkownikowi.

Analizując dostępne na rynku rozwiązania typu framework (np. *Zend Framework*), ustalono, iż większość z nich służy budowie aplikacji webowych skierowanych do wykorzystania przez dużą liczbę użytkowników. Ponadto, nie współpracują z najnowszym i najszybszym na rynku sterownikiem dostępu do danych, jakim jest *PDO* [5]. *OraLabManager*, z założenia, miał być skierowany do administratora laboratorium oraz DBA. Większość funkcjonalności biznesowych nie jest potrzebna w przypadku tak specyficznego systemu, dlatego zdecydowano się na własną implementację. Przykład implementacji klasy *Model* tworzącej statyczny obiekt *PDO* (np. [5]) przedstawia poniższy kod *PHP*.

```
class Model
{
    protected static $DBh = null;
    protected function connect()
    {
        if(self::$DBh == null) {
            try {
                    self::$DBh = new PDO("oci:", "system", "xyz");
                    self::$DBh->setAttribute(PDO::ATTR_ERRMODE,
                                                PDO::ERRMODE_EXCEPTION);
              } catch (PDOException $e) {
                 throw new Exception("Error!: " . $e->getMessage());
              }
        }
    }
}
```

Implementacja warstwy kontrolera występuje w postaci klasy *Controller* oraz wszystkich klas, dziedziczących po klasie *Controller*. Na poniższym listingu przedstawiono uproszczony kod klasy *Controller* bez ciał metod składowych. Znajdują się w niej takie metody, jak: renderowanie widoku, pobieranie danych z pliku konfiguracyjnego, dodawanie odnośnika do pliku procedur *JavaScript*, dodawanie odnośnika do pliku styli *CSS*, dodawanie elementu składowego podmenu oraz reprezentacja obiektu klasy *Smarty* (widoku).

```
class Controller {
    public $javaScripts = array();
```

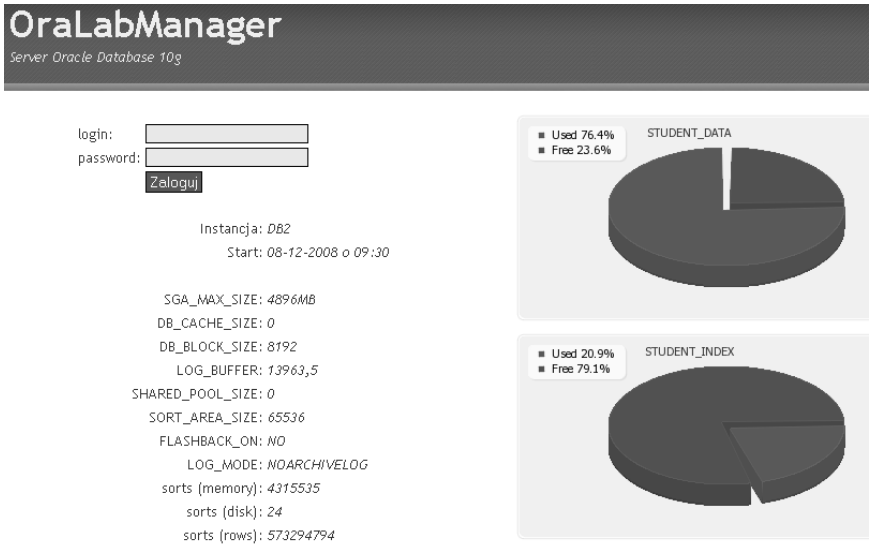
```
    public $csses = array();
    public $subMenu = array();
    public $smarty;
    protected function addJS($js);
    protected function addCSS($css);
    protected function addSubmenu($href, $text);
    protected function renderView($content);
    protected function getConfig();
    public function __construct();
}
```

Istotnym elementem systemu jest tzw. *Front Controller* (wzorzec architektoniczny), będący pojedynczym punktem dostępu do systemu, przez który przechodzą wszystkie wywołania *HTTP* [3]. Dzięki temu wspólne dla wszystkich akcji funkcjonalności są obsługiwane z poziomu jednego miejsca wywołania – np. ładowanie pliku konfiguracyjnego oraz uwierzytelnianie użytkowników.

2.3. Budowa systemu

Pod względem funkcjonalnym system *OraLabManager* został podzielony na dwie funkcjonalności:

- zarządzanie użytkownikami (lista użytkowników bazy danych, lista aktualnie zalogowanych użytkowników, tworzenie nowego użytkownika, tworzenie użytkowników wg pliku, lista zdefiniowanych ról, tworzenie nowej roli),
- monitoring serwera Oracle Database (monitoring przestrzeni tabel, dodawanie pliku danych do przestrzeni tabel, lista plików danych, monitoring I/O, kontrola braku przestrzeni TEMP dla użytkowników, monitoring zajętości przestrzeni przez obiekty schematów użytkowników, monitoring ekstentów, kontrola dużych segmentów, kontrola dużych tabel, kontrola segmentów wycofania, sprawdzanie warunku optymalnego rozmiaru (od 1024 do 4096 ekstentów) dla przestrzeni tymczasowych, kontrola plików kontrolnych, segmentów wycofań, logów i śledzenia, kontrola i audyt sesji, monitoring parametrów SGA).



Rys .1. Okno logowania do systemu OraLabManger
Fig. 1. Login window of the OraLabManager system

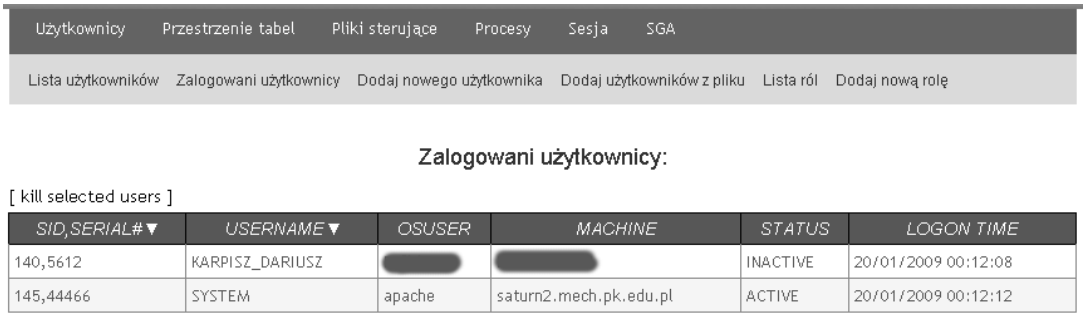
Samo otworzenie witryny internetowej (rys. 1) udostępnionej pod adresem:
https://saturn2.mech.pk.edu.pl/OraLabManager/htdocs/,
umożliwia wgląd w aktualny stan instancji, a z punktu widzenia DBA, wzrokowe ustalenie konieczności przydziału dodatkowego miejsca w przestrzeniach tabel obsługujących prywatne schematy studentów (tu: *STUDENT_DATA* i *STUDENT_INDEX*).

Z punktu widzenia przedmiotowego zastosowania, z praktyki dydaktycznej wynika, że przydatną funkcją jest zamykanie zablokowanych sesji. Aby wykonać taką operację, niezbędne jest najpierw zidentyfikowanie użytkownika, a następnie zamknięcie jego sesji (np. [2, 4, 7]), co odzwierciedla poniższy kod *SQL*.

```
select sid, serial# as serial, username, osuser, machine, status,
 to_char(LOGON_TIME, 'DD/MM/YYYY HH24:MI:SS') as logon_time
from v$session where username is not null;

alter system kill session 'sid, serial#';
```

W przypadku użycia narzędzia *OraLabManager* możliwe jest wykonanie tej operacji w sposób intuicyjny, zaznaczając odpowiedniego użytkownika na liście, a następnie wykonując opcję „kill selected users”. Interfejs takiej operacji przedstawiony został na rys. 2. Na rysunku widoczne jest również menu systemu.



Rys .2. Moduł obsługi sesji użytkowników
Fig. 2. Module of operation of sessions of database users

Inną, niezwykle istotną funkcją omawianego systemu jest graficzna reprezentacja zajętości obszaru składowania dla poszczególnych przestrzeni tabel. Liczba przestrzeni w rozbudowanych systemach może być znaczna, dlatego najlepiej jest wykorzystać do ich szybkiego monitoringu wykresy kołowe. Przykład niniejszej implementacji prezentuje rys. 1. Taka funkcjonalność dostępna jest dodatkowo z poziomu menu „Przestrzenie tabel", prezentująca zestawienia i wykresy dla wszystkich przestrzeni określonych poniższym zapytaniem.

```
select nvl(b.tablespace_name,
       nvl(a.tablespace_name,'UNKOWN')) name,
 kbytes_alloc total,
    kbytes_alloc-nvl(kbytes_free,0) used,
    nvl(kbytes_free,0) free,
    ((kbytes_alloc-nvl(kbytes_free,0))/kbytes_alloc)*100 pct_used
from   ( select sum(bytes)/1024 Kbytes_free, tablespace_name
    from  sys.dba_free_space
    group by tablespace_name ) a,
    ( select sum(bytes)/1024 Kbytes_alloc,
          tablespace_name
      from sys.dba_data_files
      group by tablespace_name )b
where a.tablespace_name (+) = b.tablespace_name
order by 1;
```

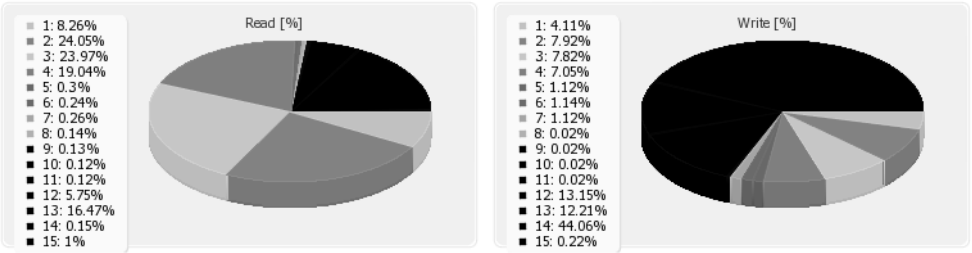
Kolejną ważną opcją wymagającą monitorowania, by uniknąć problemów z wydajnością jest obciążenie wejścia–wyjścia (I/O) dla przestrzeni składowania danych. Należy bowiem co jakiś czas analizować statystyki dysków podczas ich zwykłego oraz szczytowego obciążenia np. podczas zajęć laboratoryjnych. Nie należy polegać na jednokrotnych pomiarach, ponieważ system zmienia się dynamicznie i dyski fizyczne muszą być monitorowane. System *OraLabManager* umożliwia monitorowanie całego zakresu I/O instancji, co zaprezentowano na rys. 3.

Physical I/O by filesystem

READ	WRITE
390,821	2422,952

Physical I/O by files

Lp.	NAME ▼	READ ▼	[%]	WRITE ▼	[%]
1	b0_student_data01.dbf	32292	8.26	99581	4.11
2	b0_student_data02.dbf	94005	24.05	191834	7.92
3	b0_student_data03.dbf	93679	23.97	189410	7.82
4	b0_student_data04.dbf	74412	19.04	170837	7.05
5	b0_student_index01.dbf	1173	0.3	27241	1.12
6	b0_student_index02.dbf	922	0.24	27608	1.14
7	b0_student_index03.dbf	1000	0.26	27171	1.12
8	b0_system_work01.dbf	545	0.14	531	0.02
9	xample01.dbf	490	0.13	484	0.02
10	wingbench_cc_data.dbf	488	0.12	484	0.02
11	wingbench_cc_index.dbf	488	0.12	484	0.02
12	ysaux01.dbf	22470	5.75	318685	13.15
13	ystem01.dbf	64356	16.47	295797	12.21
14	ndotbs01.dbf	596	0.15	1067462	44.06
15	sers01.dbf	3905	1	5343	0.22



Rys .3. Rozkład I/O dla poszczególnych plików danych
Fig. 3. Physical I/O by the selected data files

Przykład zapytania do monitoringu liczby operacji I/O dla systemu plików [4] pokazano na kolejnym listingu kodu *SQL*.

```
select
   sum(s.phyblkrd)/1024 read, sum(s.phyblkwrt)/1024 write
from v$datafile d, v$filestat s
where s.file# = d.file#;
```

Opis pełnej funkcjonalności systemu *OraLabManager* jest zbyt obszerny do przedstawienia w ramach niniejszej pracy. Ponadto, ze względu na wysoką elastyczność rozwiązania możliwa jest szybka ewaluacja w zakresie dostępnych funkcji, dlatego zakres opcji menu może podlegać zmianom.

3. Podsumowanie

Dobór właściwych narzędzi do monitoringu i eksploatacji serwera baz danych może być trudny. W szczególności, zastosowania specjalizowane, niezwiązane z dużym rynkiem odbiorców biznesowych mogą (ale nie muszą) wymagać utworzenia własnych, wygodniejszych niż już dostępne pakietów oprogramowania dla DBA. Tak też się stało w przypadku systemu *OraLabManager*, dedykowanego do monitoringu serwera *Oracle 10*g użytkowanego w zakresie laboratorium baz danych.

Udało się utworzyć system do monitoringu serwera przystosowanego do prowadzenia zajęć laboratoryjnych, znacznie skracający czas wprowadzania nowych użytkowników oraz czas reakcji na ewentualne zagrożenia wydajności i bezpieczeństwa. Ponadto, możliwe jest przystosowanie systemu do innego zakresu zastosowań. Taką adaptację umożliwia odpowiednia konstrukcja sytemu wykorzystująca model MVC i inne rozwiązania natury technicznej (np. *PDO*, *JQuery*).

System został zainstalowany na serwerze użytkowanym w Instytucie Informatyki Stosowanej na Wydziale Mechanicznym Politechniki Krakowskiej, a poszczególne jego funkcje, a przede wszystkim bezpieczne użytkowanie zostało zweryfikowane przez najbardziej wymagających (pomysłowych) użytkowników – studentów.

BIBLIOGRAFIA

1. Fowler M.: Refactoring – Improving the Design of Existing Code. Addison Wesley Pub Co Inc, 1999.
2. Gutta R.: Oracle skrypty administracyjne. Mikom, Warszawa 2002.
3. Komar B.: Administracja sieci TCP/IP dla każdego. Helion, Gliwice 2000.
4. Loney K.: Oracle Database 10g Kompendium administratora. Helion, Gliwice 2005.
5. McLaughlin M.: Oracle Database 10g Express Edition. Tworzenie aplikacji internetowych w PHP. Helion, Gliwice 2007.
6. Shalloway A., Trott J.R.: Projektowanie zorientowane obiektowo. Wzorce projektowe. Helion, Gliwice 2002.
7. Whalen E.: Oracle Database 10g Administracja bazy danych w Linuksie. Helion 2007.
8. Zakas N.: JavaScript dla webmasterów. Zaawansowane programowanie. Helion, Gliwice 2006.

Recenzent: Dr inż. Bożena Małysiak-Mrozek

Wpłynęło do Redakcji 20 stycznia 2009 r.

Abstract

Efficient maintain of database systems required additional tools for database server monitoring. Sometimes is better to use dedicated monitoring system than commercial tools. Dedicated application prepared for specific system will fulfill all required functions.

The system supports large number of users by creating users from text file. The most important factors are present as circle charts, what makes identification of problems much easier.

In this paper web based system for monitoring instance of Oracle database, use for education purpose was presented.

Adres

Dariusz KARPISZ: Politechnika Krakowska, Instytut Informatyki Stosowanej,
al. Jana Pawła II 37, 31-864 Kraków, Polska, drejku@poczta.onet.pl.