

# Baza danych Oracle jako otwarte, rozszerzalne i potężne środowisko obliczeniowe. Integracja Maszyny Wirtualnej Perla z jądrem Oracle

Piotr Jarmuż  
ITTI sp. z o.o.

*piotr.jarmuz@itti.com.pl*

**Abstrakt.** Projekt i artykuł mają na celu pokazanie, że Oracle to coś znacznie więcej niż tylko czarna skrzynka przetwarzająca zapytania SQL. W bazie danych Oracle istnieje szereg otwartych i udokumentowanych API umożliwiających dowolne rozszerzanie jądra Oracle. Parę lat temu zająłem się projektem hobbystycznym mającym na celu zintegrowanie środowiska języka Perl i bazy danych Oracle, umożliwienie pisania procedur składowanych w Perlu. Wykorzystanie doskonałych zdolności Perla do przetwarzania tekstów ze skalowalnością Oracle pozwoliłoby na uzyskanie całkiem nowej jakości. Artykuł opisuje ogólną architekturę Oracle'a, podstawy maszyny wirtualnej Perla, pomysł, sposób i szczegóły implementacyjne realizacji projektu integracyjnego. Artykuł dotyczy bazy danych Oracle 9i r2 i wersji maszyny Perla 5.8.0.

**Informacja o autorze.** Piotr Jarmuż w latach 1991-1993 studiował na Politechnice Poznańskiej na kierunku Elektrotechnika. W latach 1993-1996 kontynuował studia w EFP - Francusko-Polskiej Wyższej Szkole Technik Informatyczno-Komunikacyjnych w Poznaniu. W 1995 roku odbył półroczny staż w instytucie badawczo rozwojowym Forschung und Technik Daimler-Benz w Berlinie. Stopień magistra inżyniera uzyskał w 1996 roku w dziedzinie systemów rozproszonych. Od 1998 roku pracował na stanowisku programisty aplikacji baz danych w firmie Praendle Software GmbH w Berlinie. Od roku 2000 pracował jako kierownik polskiej grupy programistów przy projekcie Messe Frankfurt Online w Bertelsman Nionex w Guetersloh w Niemczech. Od 2002 do 2004 jako administrator i deweloper baz danych Oracle w firmie ASCO GmbH w Braunschweig, Niemcy. Po powrocie do Polski przez pół roku pracował jako konsultant w firmie DGA S.A. w Poznaniu. Do końca 2007 roku w ITTI na stanowisku konsultanta realizował projekty związane z analizą, projektowaniem i budową systemów informatycznych oraz z zagadnieniami bezpieczeństwa informacji. Od stycznia 2007 pracuje jako Database Analyst w GSK Technical Centre i zajmuje się administrowaniem baz danych Oracle i MS SQL Server w korporacji GSK.

## Audytarium

Dokument ten przeznaczony jest dla średnio i wysokozaawansowanych użytkowników Oracle'a. Wymagana jest elementarna wiedza na temat systemu UNIX/Linux, bardzo dobra znajomość architektury serwera baz danych Oracle w rozdziałach dotyczących struktury procesów i struktur globalnej dzielonej pamięci. Wymagana jest zaawansowana znajomość PL/SQL i Perla w szczególności w kontekście integracji obu języków z środowiskiem runtimowym „C”, API OCI i perlembed, elementarna znajomość zasad funkcjonowania nowoczesnych interpreterów bytcodeowych jak Java i Perl oraz ogólna wiedza na temat technik kompilacji, architektury kompilatorów w systemów POSIX-owych.

## 1. Rozszerzalna architektura Oracle

Serwer bazy danych Oracle to nie tylko zwykły serwer SQL, czarna skrzynka, do której wrzuca się zapytania SQL-owe i oczekuje na tabularyczne rezultaty. Serwer Oracle to potężne środowisko obliczeniowe umożliwiające rozwiązywanie nawet najbardziej skomplikowanych projektów w dużej mierze w samym serwerze [TKYTE1]. Serwer bazy danych Oracle oferuje wiele dobrze opisanych i otwartych API umożliwiających łatwą integrację z istniejącym oprogramowaniem z wykorzystaniem standardowych protokołów. Serwer Oracle posiada oprócz standardowego engina SQL zgodnego ze standardem ANSI 1999, szereg wbudowanych języków programowania 3 i 4 generacji. W jądrze Oracle zintegrowany z SQL jest język 4 generacji PL/SQL, od wersji 8.x w skład serwera Oracle wchodzi własna implementacja maszyny wirtualnej Java „Aurora” oraz standardowe środowisko runtimowe JRE (3 - generacji) oraz do programowania na najniższym poziomie język „C” do rozwiązywania najbardziej nietypowych zadań, dla których wymagania czasowe są krytyczne oraz dla tych zadań, gdzie wymagana jest integracja z istniejącymi systemami dla których istnieją tylko interfejsy binarne w języku „C”.

### 1.1 PL/SQL

Język PL/SQL jest językiem 4 generacji bazowanym na języku Ada wbudowanym w engine SQL-owy w jądrze serwera. PL/SQL jest językiem proceduralnym z rozszerzeniami obiektowymi obsługującym wyjątki i transakcje Oracle'a. Wspiera wszystkie standardowe typy danych SQL na poziomie natywnym, tzn. nie wymagana jest żadna konwersja pomiędzy SQL-em a PL/SQL-em. Konstrukty PL/SQL-a mogą istnieć w postaci procedur, funkcji zarówno niezależnych jak i zgrupowanych w logiczne i funkcjonalne pakiety.

### 1.2 Java

Od wersji 8.x Oracle oferuje nowy język 3 – generacji Java. Oracle zaimplementował własną wersję JVM Aurora, która jest zdecydowanie szybsza i lepiej nadaje się do rozwoju oprogramowania w Javie niż jakakolwiek inna implementacja JVM. Dla osób zajmujących się profesjonalnym rozwojem oprogramowania w Javie najważniejszy jest fakt, iż nie wymagany jest restart serwera, przy ponownej kompilacji i instalacji pakietów i klas Javy. Maszyna wirtualna Java jest zintegrowana ściśle z jądrem Oracle'a tzn. bytecode i dane Javy znajdują się w tej samej przestrzeni adresowej co dane serwera, w tym zachowane bloki baz danych, co umożliwia bezpośrednią manipulację kodowi Javy na tych danych bez konieczności drogiego przełączania kontekstów na poziomie systemu operacyjnego. Kod Javy można tworzyć bezpośrednio za pomocą konstruktorów DDL „create or replace and compile” albo przygotować na zewnątrz i zainstalować poprzez pakiet zarządczy dbms\_java z poziomu SQL-a albo za pomocą narzędzia load\_java z linii komendy

### 1.3 „C” i model external stored procedures

Oracle oferuje możliwość pisania procedur składowanych w języku „C” od bardzo wczesnych wersji. Procedury pisane w „C” mają największe możliwości. Ze względu na bezpieczeństwo danych procedury składowane pisane w „C” są implementowane zupełnie odmiennie niż procedury w PL/SQL i Javy, które są zintegrowane bezpośrednio w jądrze Oracle’a. PL/SQL i Java mają swój własny wbudowany mechanizm dostępu do danych i nie umożliwiają by design manipulacji na wskaźnikach co nie może skutkować przypadkowym zniszczeniem danych serwera i/lub (co gorsza) danych persystentnych w samej bazie. Kod w „C” nie ma takich zabezpieczeń i umożliwia zrobienie wszystkiego na co pozwala sam język i biblioteki w kontekście użytkownika systemowego. W przypadku procedur pisanych w „C”, Oracle tworzy nowy model komunikacji danych. Procedury pisane w „C” są technicznie zawarte w bibliotece dynamicznej linkowanej z zewnętrznym w stosunku do serwera Oracle procesem o nazwie extproc w celu separacji przestrzeni adresowych. Taka separacja powoduje, iż niemożliwa jest bezpośrednia manipulacja na danych bloków bazodanowych i ich przypadkowe zniszczenie ale także zwiększa to koszty pojedynczego wywołania, ze względu na przełączanie kontekstów i konieczność stosowania stosunkowo „ciężkich” metod komunikacji IPC. Procedury w „C” można pisać na poziomie niskim używając mechanizmów OCI (Oracle Call Interface) albo z wykorzystaniem prekompilatora i bibliotek Pro\*C.

## 2. Perl i maszyna wirtualna

Wbrew powszechnej opinii język Perl nie jest językiem czysto interpretowanym. Można wyróżnić typowe fazy charakterystyczne dla języków kompilowanych takich jak „C” lub Java. Interpreter Perla składa się z parsera i wykonawcy. Parser dokonuje analizy syntaktycznej modułu a wynikiem jego pracy jest tymczasowa, istniejąca tylko w pamięci struktura bytcodeu (tzw. p-code). Struktura taka reprezentuje w runtimie kompilowaną funkcję i jest przekazywana do modułu wykonawcy. Bytecode Perla jest zasadniczo różny od bytcodeu Javy. Bytecode Javy odpowiada mniej więcej językom maszynowym dla istniejących architektur procesorów i zawiera ponad 200 prostych instrukcji manipulujących stosem i stanem maszyny operujących na prostych operandach 1, 2, 4 i 8 bytowych. Bytecode Perla zawiera ponad 600 opcodów, które odpowiadają wbudowanym funkcjom i operatorom jak i operacjom podstawowym na dużo wyższym poziomie, np. jednym opcodem w Perlu jest referencja do tablicy haszującej (łącznie z obliczeniem wartości haszującej dla klucza) lub pobranie elementu z tablicy, co w Javie odpowiadało by w najlepszym przypadku kilkadziesiąt rozkazom. Opcody w module wykonawczym odpowiadają wysoce zoptymalizowanym funkcjom napisanym ręcznie w „C”. Taki model sprawia, że uzasadnione jest myślenie o Perlu jak o skryptowalnym „C”. Maszyna wirtualna Perla jest zaimplementowana w bibliotece libperl, wchodzącej w skład standardowej instalacji Perla na każdej platformie. Tak więc interpreter Perla w linii komendy jest prostym programem zlinkowanym z tą biblioteką, a całe skomplikowane przetwarzanie danych i kodu Perla zaimplementowane jest w bibliotece libperl i udostępnione programom w „C” poprzez dobrze zdefiniowany i otwarty interfejs. W szczególności rozdzielone są kroki tworzenia i inicjalizacji maszyny wirtualnej, kompilacji modułów Perla oraz przekazywania parametrów, właściwego wykonania kodu i zwrotu wartości. W tym projekcie wykorzystaliśmy API **perlembed**.

## 3. Założenia i ograniczenia systemu

Cały projekt zdecydowano się zaimplementować w następującym środowisku:

- System operacyjny Linux Debian 3.0
- Serwer bazy danych Oracle 9i r2
- Perl wersja 5.8.0
- Kompilator GNU „C” 2.8.0

Ze względu na ograniczenia czasowe projektu jak i potencjalne problemy z wydajnością zdecydowano iż integracja wirtualnej maszyny Perla z serwerem Oracle ograniczona będzie tylko do konfiguracji z serwerem dedykowanym. Wtedy, gwarantowane jest istnienie zewnętrznych procesów extproc w stosunku **1:1** do dedykowanych serwerów (procesów użytkownika), a tym samym nie istnieje konieczność zapamiętywania stanu maszyny wirtualnej Perla wewnątrz kontekstu Oracle [TKYTE2], co byłoby o tyle nieefektywne co prawie niemożliwe jako, że stan maszyny wirtualnej Perla jest dostępny tylko poprzez „opaque handle” z poziomu języka „C”. Oracle oferuje sposób na przechowywanie globalnych danych „C” skojarzonych z sesją użytkownika w strukturze OCIContext, nie mniej jednak nie istnieje łatwy sposób na serializację stanu maszyny i zapisanie jej w postaci strumienia bajtów. Byłoby to jak wspomniano wyżej bardzo nieefektywne. Zatem ograniczenie projektu do konfiguracji z serwerem dedykowanym ma sens i jest całkiem rozsądne. Ponadto nawet przy konfiguracji z serwerem dzielonym istnieje możliwość „wymuszenia” dedykowanego połączenia i co za tym idzie dedykowanego procesu serwera i procesu extproc poprzez odpowiedni string inicjalizujący połączenie lub odpowiedni wpis TNS w pliku konfiguracyjnym po stronie klienta.

## 4. Interfejsy

System posiada dwa interfejsy. Pierwszy interfejs to standardowy interfejs SQL i PL/SQL Oracle’a oferowany każdemu klientowi bazy danych. Podsumowując kod w Perlu może być w sposób przezroczysty wywoływany z dowolnej aplikacji Oracle’owej. Drugi interfejs to interfejs zarządczy służący do definiowania nowych funkcji, procedur i pakietów w Perlu oraz ich wrapperów widocznych dla klienta.

Przykładowy kod klienta:

```
declare
  l_length number;
  l_file varchar2(30) := 'file.txt';
begin
  l_length:=perl_demo(l_file);
  dbms_output.put_line('Length of file' || l_file || ' is ' || l_length);
end;
```

Powyższy kod wywoływany jest dynamicznie dowolną ilość razy bez konieczności wykonywania dodatkowych kroków.

Kod procedury wrappera konwertujący parametry wejściowe do tablicy typu varchar2 i oddający sterowanie do generycznego punktu wejścia do maszyny wirtualnej Perla. Ewentualne parametry wyjściowe (lub wartość zwracana przez funkcję) są pakowane w tablicę stringów i konwertowane do właściwego typu po powrocie do kodu wywołującego.

```
create or replace function perl_demo(p_file varchar2)
authid current_user
return number as
l_inarr: t_perl_array;
l_outarr: t_perl_array;
begin
  l_inarr[1]:=p_file;
  perl_entry_point(l_inarr, l_outarr);
  return outarr[1];
end;
```

Powyższy kod wykonywany jest jednokrotnie po zainstalowaniu nowej funkcji.procedury Perla. Pełni rolę pośrednika między kodem klienta, a generycznym wrapperem.

Trzy kolejne instrukcje wykonywane są jednokrotnie podczas instalacji systemu.

Oto deklaracja punktu wejścia do maszyny wirtualnej Perla.

```
create or replace procedure perl_entry_point(p_inarray t_perl_array,
p_outarray t_perl_array) as language C name "C_perl_entry_point" library
oraperl with context
parameters(context, p_inarray OCICOLL, p_outarray OCICOLL);
```

Kod implementujący powyższą procedurę napisany jest w „C” i zawarty jest w bibliotece dzielonej oraperl. Biblioteka oraperl pełni rolę kodu łączącego Oracle i Perl, wykorzystuje API OCI Oracle’a i perlembd Perla.

Definicja biblioteki oraperl.

```
create library oraperl as '/var/lib/oraperl.so';
```

Przykładowe polecenie DDL zezwalające użytkownikowi **scott** na wywoływanie procedur w Perlu.

```
grant execute on oraperl to scott;
```

Kod Perla realizujący funkcję perl\_demo.

```
package Demo;

sub perl_demo
{
    my p_file=shift;
    my @arr=stat(p_file);
    return $arr[7];
}
```

Interfejs zarządczy ma na celu definiowanie, usuwanie, modyfikację pakietów Perla i w przyszłości automatyczną generację kodu PL/SQL wrappera dla funkcji i/lub procedury oraz jej automatyczną kompilację i instalację w bazie danych. Poprzez interfejs zarządczy można zdefiniować, czy kod Perla będzie przechowywany w systemie plików czy też w specjalnych tablicach w dedykowanym schemacie w samej bazie Oracle. Istnieje nawet możliwość zdecydowania czy kod Perla wykonywany będzie w kontekście właściciela czy też użytkownika słowo kluczowe AUTHID przy definicji wrappera.

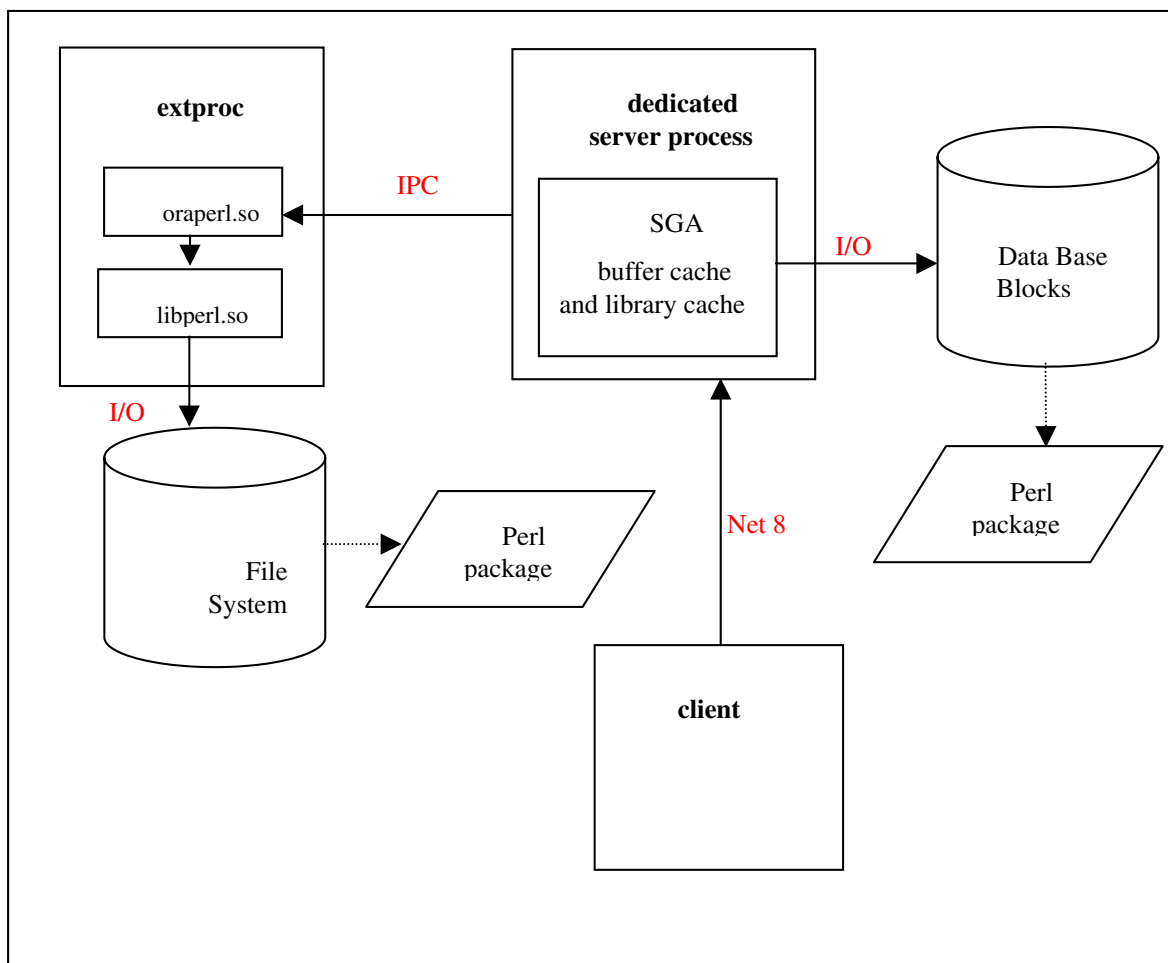
```
create type t_perl_array as table of varchar2;

create or replace package dbms_perl as
    procedure setDBSource(p_flag boolean);
    function getDBSource return boolean;
    procedure addSearchPath(p_path varchar2);
    procedure deleteSearchPath(p_path varchar2);
    procedure addPerlSource(p_file varchar2);
    procedure deletePerlSource(p_path varchar2);
    procedure compileFunction(p_name varchar2, p_body varchar2);
    procedure dropFunction(p_name varchar2);
    procedure compileProcedure(p_name varchar2, p_body varchar2);
    procedure dropProcedure(p_name varchar2);

    function listSearchPaths() return t_perl_array;
    function listPerlFunctions() return t_perl_array;
    function listPerlProcedures() return t_perl_array;
end;
```

## 5. Architektura systemu

Na rysunku 1. przedstawiona została ogólna architektura implementowanego systemu.



Rys. 1

### Opis systemu:

Z punktu widzenia Oracle'a maszyna wirtualna Perla dostępna jest poprzez standardowy interfejs zewnętrznych procedur składanych w „C”. Interfejs ten realizuje proces o nazwie **extproc**. Sama maszyna wirtualna Perla znajduje się w bibliotece **libperl.so**. Biblioteka ta wchodzi w skład runtime'u każdej instalacji Perla. Nasz kod sprzęgający zawarty jest w bibliotece o nazwie **oraperl.so**. Biblioteka ta definiuje punkt wejścia do maszyny wirtualnej *C\_perl\_entry\_point* oraz zajmuje się inicjalizacją, sterowaniem przepływu i przekazywaniem danych Oraclowych do Perla i danych zwrotnych w kierunku przeciwnym. Procedury z biblioteki **oraperl** zajmują się też bardzo podstawową obsługą błędów. Błędy runtime'u Perla są wychwytywane w bibliotece **oraperl** a do Oracle'a zwracany jest wyjątek poprzez mechanizm **OCIExtProcRaiseExcp**. Reasumując po otwarciu sesji przez klienta Oracle w konfiguracji serwera dedykowanego tworzony jest dokładnie jeden proces/wątek użytkownika. Gdy aplikacja kliencka po raz pierwszy odwołuje się do kodu Perla (czyli z punktu widzenia Oracle jest to kod „C”) to serwer tworzy zewnętrzny proces **extproc**, który dynamicznie linkuje się z naszą biblioteką **oraperl** poprzez wiązania zdefiniowane przy deklaracji generycznego wrappera. Implementacja funkcji *C\_perl\_entry\_point* sprawdza czy jest wywołana po raz pierwszy w sesji i jeśli tak to inicjalizuje maszynę wirtualną Perla poprzez mechanizm *perl\_alloc*, *perl\_construct*. Dla każdego wywołania wykonywany jest kod *perl\_parse* i *call\_argv* przekazujący argumenty z *inarray* do funkcji Perla oraz ładujący wartości zwrotne do tablicy *outarray* po powrocie z funkcji Perla. Biblioteka **oraperl** czyta z bazy danych parametry

inicjalizujące aby wiedzieć gdzie ma szukać i jak lokalizować pakiety Perla (w systemie plików PERL\_PATH lub w bazie danych z tablicy perl\_code w schemacie **perl**). Wywołanie *perl\_parse* dokonuje analizy syntaktycznej, a wynikiem jego jest zoptymalizowany bytecode Perla odpowiadający kompilowanej funkcji. Wywołanie *perl\_parse* jest wewnętrznie zoptymalizowane tzn. wielokrotne wywołanie *perl\_parse* na tym samym kodzie Perla nie powoduje wielokrotnej kompilacji. Wywołanie *call\_argv* powoduje wejście do maszyny Perla i wykonanie naszej funkcji.

### Przeływ danych i sterowania dla przykładowego wywołania:

Program klienta (może to być dowolny klient Oracle'a np. SQL\*Plus albo TOAD, lub dowolna aplikacja np. używająca JDBC Javy albo DBI Perla) wysyła zapytanie SQL-owe zawierające pośrednią referencję do zewnętrznej procedury składowanej w Perlu. Klient może widzieć tylko funkcje SQL-a, funkcje PL/SQL-a albo anonimowe bloki zawierające procedury PL/SQL-a. Procedury jak i funkcje PL/SQL-a mogą być samodzielne jak i należeć do pakietu. Klient widzi naszą funkcję PL/SQL, która jest wrapperem dla kodu w „C” i pośrednio kodu w Perlu, przekazuje parametry wejściowe do generycznego punktu wejścia do maszyny wirtualnej Perla, gdzie następuje warunkowa kompilacja (jeśli kod jest dostarczony po raz pierwszy do tej instancji maszyny wirtualnej) i wykonanie, następnie ewentualne wartości zwrotne przekazywane są z powrotem do funkcji wrappera i oryginalnego kodu klienta. Przekazywane parametry mogą być dowolnego typu skalarnego (nie istnieje możliwość przekazywania kolekcji, rekordów i obiektów SQL-a), wszystkie parametry przed przekazaniem do Perla konwertowane są do stringów i pakowane do tablicy. Perl jest językiem traktującym dane polimorficznie, tzn. zależnie od kontekstu dana może być widziana jako tekst, liczba stała- lub zmiennoprzecinkowa, data, pojedynczy znak. Ewentualna konwersja dokonywana jest „leniwie” tzn. bezpośrednio przed użyciem, a wynik konwersji jest zapamiętany. Oczywiście taka konwersja spowalnia dodatkowo cały proces nie mniej jednak jest to konieczny krok w naszej implementacji. Wartości powrotne traktowane są również jako stringi. Konwersja zwrotna dokonywana jest automatycznie przez kompilator PL/SQL na granicy wywołania oryginalnego kodu klienta i kodu wrappera dla kodu Perla.

## 6. Korzyści i problemy wynikające z implementacji systemu

Zasadniczym problemem przy implementacji były problemy z konfiguracją i kompilacją projektu. Niestety nie udało się przenieść projektu poza środowisko linuxa (debian 3.0). Jeśli chodzi o korzyści z używania systemu to można by je opisać poprzez zmodyfikowaną mantrę Toma Kyta.

- Jeśli problem da się rozwiązać w SQL-u, zrób to w SQL-u
- Jeśli problem nie da się rozwiązać w SQL-u, zrób to w PL/SQL-u
- Jeśli problem nie da się rozwiązać w PL/SQL-u, zrób to w Javie
- Jeśli problem nie da się rozwiązać w Javie, zrób to C (**lub w Perlu**)
- Jeśli problem nie da się rozwiązać w C (**lub w Perlu**) wróć do deski kreślarskiej

Ogólnie klasę problemów dających i nadających się do rozwiązania przy użyciu ogólnego mechanizmu procedur składowanych da się podzielić na kilka rodzajów:

1. Mała ilość obliczeń na małej ilości danych
2. Mała ilość obliczeń na dużej ilości danych
3. Duża ilość obliczeń na małej ilości danych
4. Duża ilość obliczeń na dużej ilości danych

W przypadku procedur składowanych w Perlu, ze względu na ograniczenia implementacyjne wymagające użycia co najmniej 2 przestrzeni adresowych i co najmniej 2 procesów, a co za tym idzie relatywnie wysoki koszt przełączania kontekstów i kopiowania danych między przestrzeniami adresowymi, koszt ten może być częściowo skompensowany jeśli problem należy do klasy

3. Tzn. istnieje stosunkowo duża liczba obliczeń na niewielkiej ilości danych stricte bazodanowych. Wtedy gdy komunikacja między enginem SQL, a Perlem sprowadza się do przekazania kilku parametrów skalarnych i zwrotu kilku wartości skalarnych, a wykonanie samego algorytmu jest niejako delegowane do Perla, który wykonuje obliczenia np. na danych zewnętrznych w systemie plików czy poprzez sieć. To ma sens, albowiem programowanie dostępu **I/O** i sieci w Perlu jest o wiele prostsze i bardziej naturalne niż w nieco akademickim języku jakim jest PL/SQL (oczywiście chodzi o redundantną i generalnie niepraktyczną składnię PL/SQL-a, a nie o możliwości tego języka, zwłaszcza jeśli chodzi o integrację z SQL-em, zarządzanie transakcjami, itp)

## 7. Wnioski oraz perspektywy zastosowania

Przykładowy projekt pokazuje iż integracja tak różnorodnych środowisk jak Oracle i Perl jest możliwa. Celem projektu była demonstracja że taka integracja jest możliwa choć nie łatwa ani nie wydajna. Najnowsza wersja Serwera Oracle'a 10g zawiera już wiele pakietów wbudowanych m.in. obsługujących wyrażenia regularne zgodne z Perl OROMatcher, a więc główny atut Perla został niejako wytracony. Z drugiej strony Oracle podobnie jak inni wielcy gracze na rynku serwerów baz danych i aplikacji zaczynają powoli dostrzegać zalety języków skryptowych takich jak PHP i Perl oraz ich niewątpliwą przewagę nad Java mianowicie efektywność rozwoju oprogramowania. Można więc mieć nadzieję, że następna wersja serwera Oracle'a będzie miała już w swym jądrze zintegrowany interpreter i maszynę wirtualną Perla i/lub PHP, a co za tym idzie wydajność w runtimie nie będzie już tak drastycznie odbiegać od procedur składowanych pisanych bezpośrednio w PL/SQL i Javie.

## Bibliografia

- [TKYTE1] Thomas Kyte, Expert One-on-One Oracle, Apress
- [TKYTE2] Thomas Kyte, Effective Oracle by Design, Apress
- [BKUHN] Bradley M. Kuhn, "perljvm: Using B to Facilitate a Perl Port To the Java Virtual Machine"., San Diego CA, 23-27 July, 2001