

XIII Konferencja PLOUG
Kościelisko
Październik 2007

Automatyczne generowanie zawartości bazy danych

Sebastian Wajrych, Adam Pelikant
Politechnika Łódzka, Instytut Mechatroniki i Systemów Informatycznych

sebastian.wajrych@gmail.com, apelikant@p.lodz.pl

Abstrakt. Wszystkie projekty (internetowe, desktopowe, itd.) zawierające bazę danych borykają się na pewnym etapie z problemem braku danych. Zarówno programista-developer jak i tester chciałby sprawdzić działanie nowo dodanej funkcjonalności. Oczywiście prawdopodobnie obaj będą zmuszeni uzupełnić bazę porcją danych potrzebnych do sprawdzenia fragmentu programu, którym aktualnie się zajmują. Takie rozwiązanie ma dwa poważne minusy. Po pierwsze: z dużym prawdopodobieństwem programista nie uczestniczył w procesie analizy i wygenerowane przez niego dane w większości nie będą odpowiadały danym docelowym obsługiwanego biznesu, np. Order1, Order2, Order3. Drugim, równie poważnym problemem jest brak spójności danych, tzn. że Johna Smitha znajdziemy w jednym module jako konsultanta Call Center, a w drugim jako dyrektora tej samej jednostki. Takie dane kompletnie nie nadają się do prezentacji aplikacji klientowi. Nie może on zobaczyć działania systemu w swoim kontekście, a co za tym idzie może odczuć, że nie rozumiemy jego biznesu, zatem nie będziemy w stanie spełnić jego wymagań. Dodatkowo straciliśmy cenne zasoby dla wygenerowania bezużytecznych danych.

Informacja o autorach. Mgr inż. Sebastian Wajrych, absolwent Wydziału Elektrotechniki, Elektroniki, Informatyki i Automatyki Politechniki Łódzkiej kierunku Informatyka, specjalność Bazy danych i systemy ekspertowe.
Profesor nadzw. dr hab. inż. Adam Pelikant: jedna z najbardziej doświadczonych osób w administracji i obsłudze systemów bazodanych na Politechnice Łódzkiej. Przewodniczący i założyciel koła naukowego o tematyce baz danych, zrzeszającego studentów chcących rozwijać swoje umiejętności i zainteresowania na tym polu. Bardzo życzliwa osoba, która często bardziej wierzy w umiejętności swych podopiecznych niż oni sami. Zawsze skory do pomocy przy rozwiązywaniu nowych problemów studentów związanych z bazami danych. Zawsze przedstawia jasno swoje cele i oczekiwania względem danej pracy, pomimo że często są to trudne zagadnienia. Nastawiony na praktyczne stosowanie nowych narzędzi i technologii informatycznych. Jego dewizą są słowa Alberta Einsteina "Rozwiązuj wszystko prosto, ale nie prościej".

Wszystkie projekty (internetowe, desktopowe, itd.) zawierające bazę danych borykają się na pewnym etapie z problemem braku danych. Zarówno programista-developer jak i tester chciałby sprawdzić działanie nowo dodanej funkcjonalności. Oczywiście prawdopodobnie obaj będą zmuszeni uzupełnić bazę porcją danych potrzebnych do sprawdzenia kawałka programu, którym aktualnie się zajmują.

Takie rozwiązanie ma dwa poważne minusy. Po pierwsze z dużym prawdopodobieństwem programista nie uczestniczył w procesie analizy i wygenerowane przez niego dane w większości nie będą odpowiadały danym docelowym obsługiwanego biznesu, np. Order1, Order2, Order3. Drugim równie poważnym problemem jest brak spójności danych, tzn. że Johna Smitha znajdziemy w jednym module jako konsultanta Call Center, a w drugim jako dyrektora tej samej jednostki. Takie dane kompletnie nie nadają się do prezentacji aplikacji klientowi. Nie może on zobaczyć działania systemu w swoim kontekście, a co za tym idzie może odczuć, że nie rozumiemy jego biznesu, zatem nie będziemy w stanie spełnić jego wymagań. Dodatkowo straciliśmy cenne zasoby dla wygenerowania bezużytecznych danych.

Kolejnym problemem jest brak możliwości wykrycia i usunięcia wielu błędów zarówno na etapie developmentu jak i na etapie testów. I w tym przypadku również tracimy cenne zasoby

Rozwiązanie mógłby być system automatycznej generacji danych. Aby mógł on zaspokoić w pełni nasze potrzeby powinien spełnić kilka podstawowych założeń:

Dane w jak największym stopniu muszą odpowiadać danym realnym

Przykładowo, jeżeli przygotowujemy polską witrynę zajmującą się sprzedażą „topowej” muzyki to średnia wieku jej użytkowników wyniesie 20 lat z odchyleniem standardowym 5 lat, a użytkownicy o nazwisku Nowak będą stanowić 7%, a Kowalski 5% wszystkich użytkowników serwisu. Takie informacje możemy uzyskać bezpośrednio od konsultantów, na podstawie badań rynku lub wykorzystując istniejące bazy danych klienta.

System musi być wydajny

Generacja dużej ilości danych jest czasochłonna. Dlatego należy tak opracować algorytmy i ich implementację, aby zminimalizować czas generacji. Ponadto powinien on być możliwy do obliczenia i w najgorszym przypadku zależeć liniowo od liczby generowanych danych.

System musi umożliwiać łatwą rozbudowę

W przypadku tworzenia tego typu systemu nie jest możliwe na etapie analizy przewidzenie wszystkich typów oraz sposobów generowania danych. Rozbudowa może polegać na przykład na dodaniu tak prostej funkcjonalności jak nowy typ rozkładu statystycznego obsługiwany przez system. Jeżeli chcielibyśmy do generacji danych wykorzystać bazę danych klienta bądź serwis zewnętrzny, system również musi dawać możliwość łatwego dodania takiego rozwiązania.

System musi wystawiać API

W celu lepszej integracji z narzędziami developerskimi takimi jak np. Eclipse system musi wystawiać API umożliwiające wykonanie większości operacji. Na jego podstawie będzie można stworzyć odpowiednie wtyczki.

Słownik struktury – Structure Dictionary

Należy sobie odpowiedzieć na pytanie jakie dane wejściowe są nam potrzebne, abyśmy mogli wygenerować dane spełniające powyższe wymagania. Analizując przypadki użycia i pierwsze

wymaganie systemu łatwo zauważyć, że potrzebujemy zwięzłego opisu charakterystyki danych. Możemy w tym celu wykorzystać popularne rozkłady statystyczne i ich superpozycję. Potrzebujemy również „pojemnika” na zapisanie modelu danych. Mając cały czas w pamięci drugie wymaganie dotyczące łatwości rozbudowy oraz elastyczności systemu możemy zauważyć, że jedną z podstawowych struktur spełniających te założenia są słowniki. Proponuję zastosowanie słownika umożliwiającego utworzenie hierarchii. Przykładową strukturę przedstawia tabela 1.

Tabela 1. Struktura słownika

Id kolumny	Nazwa kolumny	Typ danych	Wartość domyślna	NULL
1	DIC_ID	NUMBER	Brak	NOT NULL
2	DIC_PARENT_ID	NUMBER	Brak	NOT NULL
3	DIC_TYPE	VARCHAR2(255 BYTE)	Brak	NOT NULL
4	DIC_DESCRIPTION	VARCHAR2(255 BYTE)	Brak	NULL
5	DIC_DATA_TYPE	CHAR(1 BYTE)	Brak	NULL
6	DIC_NUMBER_VALUE	NUMBER	Brak	NULL
7	DIC_STRING_VALUE	VARCHAR2(4000 BYTE)	Brak	NULL
8	DIC_DATE_VALUE	DATE	Brak	NULL

DIC_ID – identyfikator pozycji słownika; Kolumna klucza głównego.

DIC_PARENT_ID – identyfikator rodzica; wiersze w słowniku mogą tworzyć hierarchie.

DIC_TYPE – typ słownika; jest to unikatowy łańcuch tekstowy nie większy niż 255 bitów określający typ danego słownika, np.: S.T określa typ słownika definiujący węzeł tabeli.

DIC_DESCRIPTION - opis słownika; jest to łańcuch tekstowy do 255 bitów stanowiący opis znaczenia danego słownika. Parametr nieobowiązkowy.

DIC_DATA_TYPE - typ danych słownika; jest to pole typu char wielkości jednego znaku. Może przyjmować jedną z poniższych wartości:

- V – dane typu znakowego
- N – dane typu numerycznego
- D – dane typu datowego
- O – dane innego typu;

DIC_NUMBER_VALUE – dane typu numerycznego; w przypadku gdy typ danych słownika jest ustawiony na 'N' bądź 'O' przechowuje dane numeryczne słownika. Jest to parametr nieobowiązkowy, ale zależny od wybranego typu słownika.

DIC_STRING_VALUE - dane typu znakowego; w przypadku gdy typ danych słownika jest ustawiony na 'V' bądź 'O' przechowuje dane znakowe do 4000 bitów. Jest to parametr nieobowiązkowy, ale zależny od wybranego typu słownika.

DIC_DATE_VALUE - dane typu daty; w przypadku gdy typ danych słownika jest ustawiony na 'D' bądź 'O' przechowuje dane daty słownika. Jest to parametr nieobowiązkowy, ale zależny od wybranego typu słownika.

Przykładowa struktura zawierająca definicję tabeli CUSTOMRS, dla której będzie odbywała się generacja i definicję generacji dla kolumny BIRTH_DATE przedstawia diagram 1.

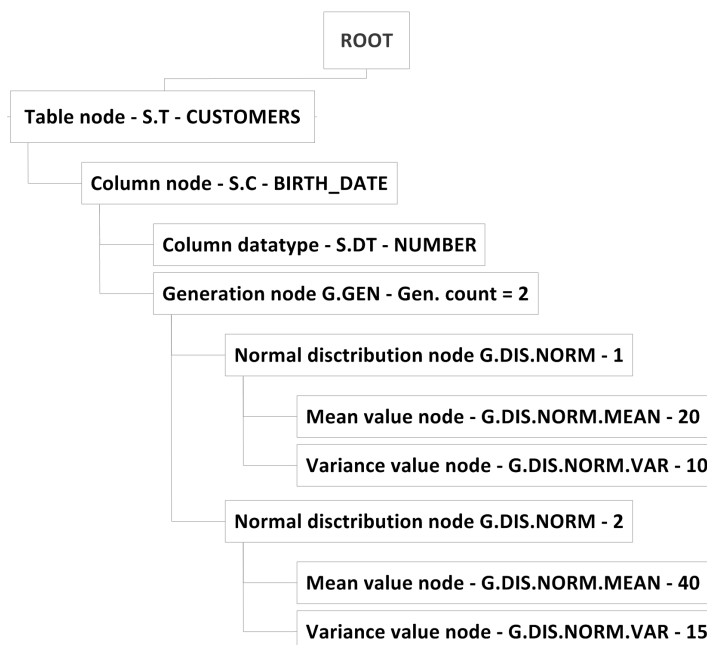


Diagram 1. Definicja podstawowej struktury w SD

Stosując tego typu strukturę dodanie nowego rozkładu nie stanowi żadnego problemu. Wystarczy zdefiniować nowy typ słownika i jego węzły podrzędne. W dalszej części artykułu stosować będę skrót SD (Structure dictionary) w odniesieniu do powyższego słownika struktury.

Słownik danych – Data Dictionary

Kolejnym ważnym elementem jest miejsce gdzie przechowujemy dane ewidencyjne takie jak elementy adresu, imiona, nazwiska, itd. Pożądanym jest, aby dane takie dawały się również hierarchizować, np. państwa → miasta → ulice.

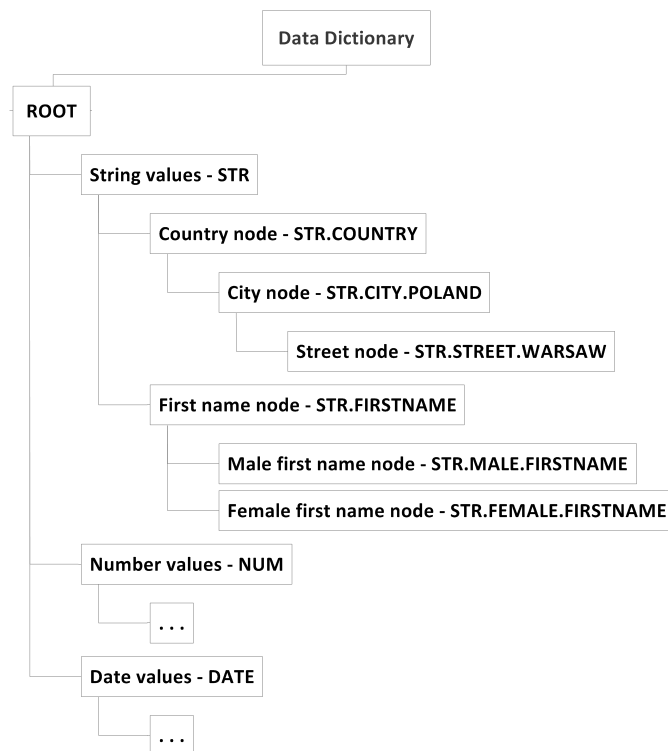


Diagram 2. Definicja podstawowej struktury w DD

W celu przechowywania tego typu danych możemy również wykorzystać struktury słownikowe. Wykorzystamy rozbudowany o pola trzech atrybutów SD.

Wystarczy to jednak dla zapisania wartości pozycji słownika z częstotliwością występowania i sumą skumulowaną dla danego typu. Jeżeli chcielibyśmy np. zapisać częstotliwość występowania poszczególnych nazwisk w każdym z województw zasadnym byłoby utworzenie osobnej tabeli na atrybuty bądź użycie macierzy. Tak powstałą strukturę będziemy określać mianem słownika danych (Data Dictionary) w skrócie DD.

Algorytm generacji danych

Mamy już zapis modelu danych i definicję generowanych danych. Zastanówmy się teraz nad algorytmem generacji. Powinien składać się z minimum trzech faz:

- przygotowania
- generacji danych
- oczyszczenia struktury

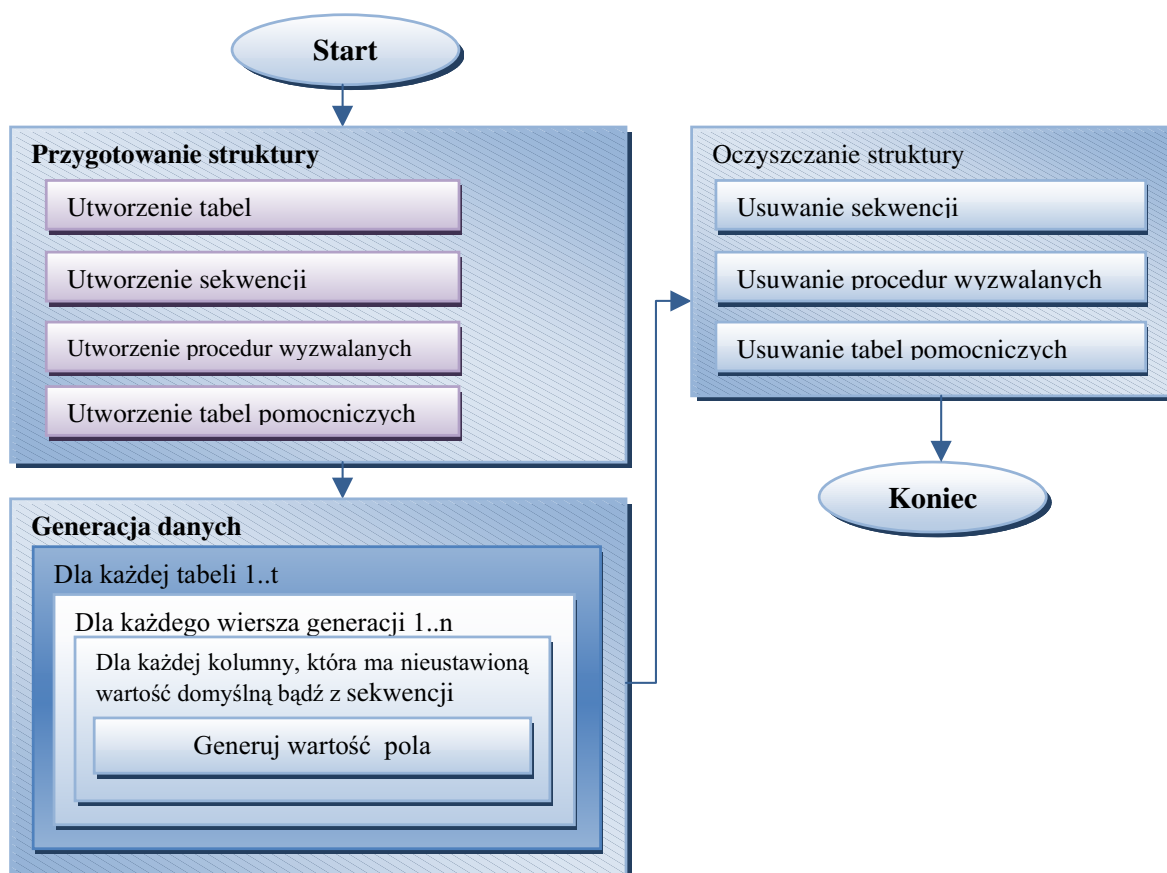
Przygotowanie struktury

W tej fazie na podstawie definicji modelu danych tworzona jest struktura nie zawierająca jeszcze żadnych danych. Dla pól, dla których zdefiniowano wartość z sekwencji tworzone są odpowiednie sekwencje i procedury wyzwalane. Tworzone są również tabele pomocnicze.

Generacja danych

W tej fazie odbywa się właściwa generacja danych. Elementarnym zdarzeniem jest generacja danych dla jednego pola w jednym wierszu dla danej tabeli. I tak dla każdej tabeli n razy (gdzie n

to liczba generowanych wierszy) zostaje wywołana generacja elementarna dla każdej kolumny tabeli.



System umożliwia nakładanie generacji, tzn., że możemy wygenerować dane z kilku różnych typów słowników lub rozkładów statystycznych. Powiedzmy, że przytoczona wcześniej, jako przykład, witryna internetowa rozszerza ofertę o muzykę rockową lat 80'. Prawdopodobnie zyskamy również klientów około 50 roku życia, zatem musimy nałożyć drugi rozkład statystyczny. W tym celu SD umożliwia definiowanie dla danego pola tabeli kilku rodzajów generacji generujących oczywiście ten sam typ danych. Natomiast sam algorytm generacji zostaje wzbogacony o moduł decydujący, który ze zdefiniowanych rozkładów dla danego wiersza generacji w n -tym kroku wybrać.

Oczyszczanie struktury

Ostatnią fazą jest oczyszczania struktury ze zbędnych danych potrzebnych jedynie do procesu generacji. Takimi obiektami są np. tabele pomocnicze czy sekwencje i triggerzy.

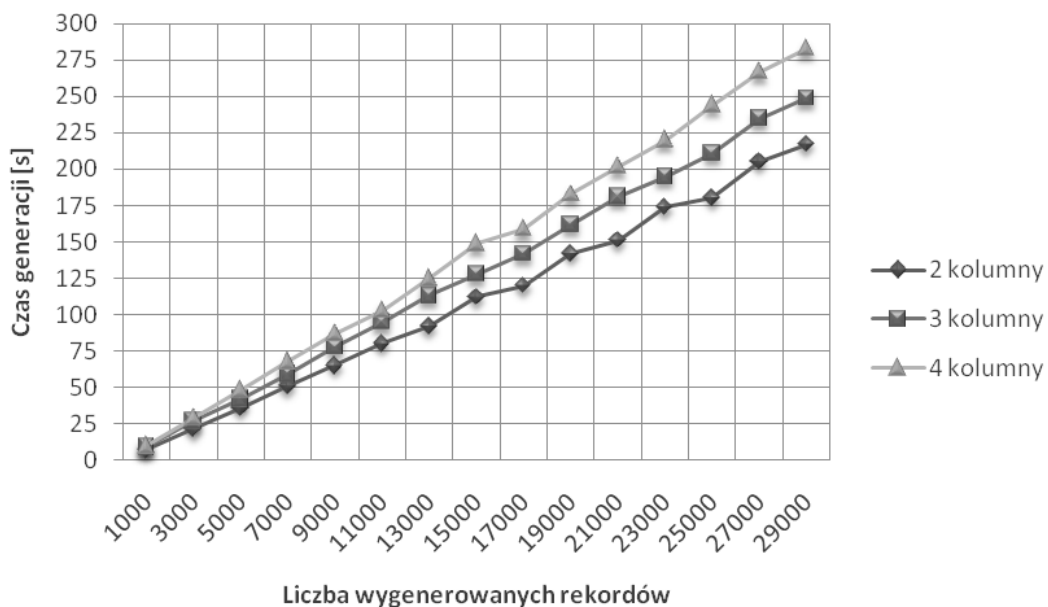
Testy prototypów

Zaproponowane rozwiązanie zostało sprawdzone w praktyce wykorzystując platformę Oracle 10g (10.2.01). Przeprowadzono również testy wydajnościowe systemu oraz zbadano zależność czasu generacji od liczby generowanych rekordów.

Tabela 1. Konfiguracja środowiska testowego

Parametr	Wartość
Baza danych	Oracle 10g (10.2.01) z Apache HTTP Server i mod_plsql
System operacyjny	Microsoft Windows XP Professional Wersja 2002 Dodatek Service Pack 2
Procesor	AMD Athlon XP 2200+ (1.81GHz)
Pamięć operacyjna	1 GB

Jako przykład omówiona zostanie generacja liczb z rozkładem normalnym. Zadaniem testu było zobrazowanie przyrostu czasu generacji liczb z rozkładem normalnym w zależności od liczby kolumn i liczby generowanych rekordów. W czasie testu wygenerowano łącznie 3 375 000 rekordów dla 2, 3 i 4 kolumn. Zaczynając od 1000 rekordów na 29000 kończąc z krokiem 1000. Przybliżony czas testu wyniósł 4 godziny 47 minuty.



Wykres 1. Zależności czasu generacji liczb z rozkładem normalnym dla 2-4 kolumn od liczby generowanych wierszy

Z otrzymanych wyników widać, że czas generacji zależy wprost proporcjonalnie od liczby generowanych wierszy.

Ciekawe wyniki dają obliczenia procentowego przyrostu czasu generacji c kolumn względem generacji dla jednej kolumny. Najprościej wyrazić to wzorem:

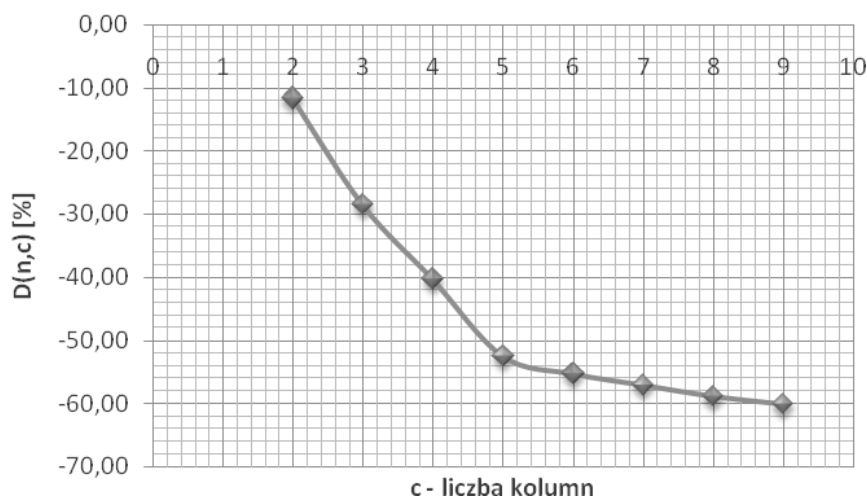
$$D(n, c) = \frac{t(n, c) - t(n, 1)}{t(n, 1)} * 100\%$$

Gdzie:

$D(n, c)$ – procentowa zmiana czasu generacji jednej kolumny przy generowaniu n wierszy z c kolumnami względem czasu generacji n wierszy jednokolumnowych.

$t(n, c)$ – czas generacji n wierszy o c kolumnach.

$t(n, 1)$ – czas generacji n wierszy jednokolumnowych



Wykres 2. Zmiana czasu generacji jednej kolumny w zależności od liczby kolumn

Zakończenie

Zbudowanie prototypu systemu umożliwiającego automatyczną generację danych spełniającego początkowe założenia pozwala wnioskować, że możliwe jest stworzenie pełnowartościowego systemu, który posiada bardzo szeroki obszar zastosowania. Należałoby postawić nacisk na stworzenie modułu umożliwiającego integrację z zewnętrznymi systemami analitycznymi, ponieważ proces ręcznego wprowadzania definicji struktury jest ponownie procesem czasochłonnym. System docelowy winien umożliwiać podłączenie do systemu analitycznego i pobranie odpowiednich informacji i na ich podstawie wygenerować definicję struktury i generacji.

Bibliografia

Steven Feuerstein, Charles Dye, John Beresiewicz, 2000. Oracle Builtin Packages. O'REILLY,
<http://www.unix.org.ua/orelly/oracle/bipack/index.htm>

Joe Celko, 2005-2007. Morgan's Library. Pages Sound Oracle Users Group,
<http://www.psoug.org/library.html>

Eric W. Weisstein at MathWorld. Electronic document, 2005
<http://mathworld.wolfram.com>

Oracle 10g Documentation, 2007
<http://www.oracle.com/technology/documentation/index.html>

Internetowy podręcznik statystyki, StatSoft, Inc. 1984-2005
<http://www.statsoft.pl/textbook/stathome.htm>