Anna Kowalczyk-Niewiadomy
Wydział Informatyki i Zarządzania
Wyższa Szkoła Informatyki i Umiejętności
ul. Rzgowska 17a 93-008 Łódź
email: anna_kowalczyk@wsinf.edu.pl

SYSTEM GENERACJI DANYCH TESTOWYCH OPARTYCH O STATYSTYKI ODWIEDZIN WITRYN INTERNETOWYCH

Streszczenie – Niniejsza publikacja opisuje system pozwalający na generacje w pełni zanonimizowanych testowych danych dotyczących statystyk ruchu w witrynie internetowej. W artykule pokazano jak w prosty sposób zintegrować aplikację Java z przykładowym systemem Google oraz jak przenieść dane z systemu Google do bazy danych Oracle. Zadanie to zrealizowano na podstawie systemu monitoringu i analizy witryn internetowych Google Analytics, a migrację oparto na przykładzie standalone'owej aplikacji napisanej w języku Java.

Słowa kluczowe: bazy danych, pozyskiwanie danych, Google Analytics, Java Google Analytics API

1 Wprowadzenie

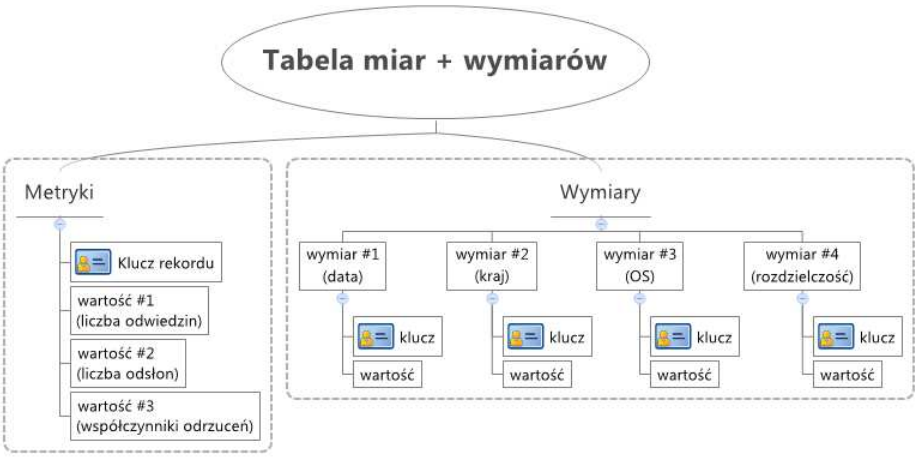
Wszelkie badania naukowe wymagają weryfikacji eksperymentalnej. W większości przypadków już na wstępnym etapie badań pojawia się potrzeba zdefiniowania testowego zbioru danych. Należy zwrócić uwagę, iż dane testowe nie zawsze są dostępne bez konieczności ponoszenia kosztów przez jednostkę naukową. Dodatkowo zważywszy na fakt, iż autorka niniejszego rozdziału zajmuje się zagadnieniami bazodanowymi (zapytania rozmyte [1][2], grupowanie danych[3], a także zapytania audio[4]) zbiór testowy powinien być bazą danych odzwierciedlającą dane pochodzące z rzeczywistego systemu bazodanowego. Takie podejście pozwala na weryfikację eksperymentów w naturalnym środowisku oddającym rzeczywiste tendencje użytkowników.

2 Projekt systemu

Bazując na poprzednim paragrafie pojawia się potrzeba automatycznego wygenerowania struktur bazodanowych odpowiadających rzeczy-

wistemu zbiorowi danych zachowując przy tym naturalność danych (brak wykorzystywania generatorów pseudolosowych) oraz dokonując ewentualnej anonimizacji danych. Docelowo na podstawie zadanych kryteriów zbioru wyjściowego system powinien móc wygenerować:

a) Pojedynczą tabelę danych (Rys. 1.) zawierającą kombinację:
   - Klucza rekordu na potrzeby modyfikacji danych
   - Zestawu metryk zdefiniowanych w kryteriach tworzenia zbioru danych
   - Zestawu wymiarów zdefiniowanych w kryteriach tworzenia zbioru danych

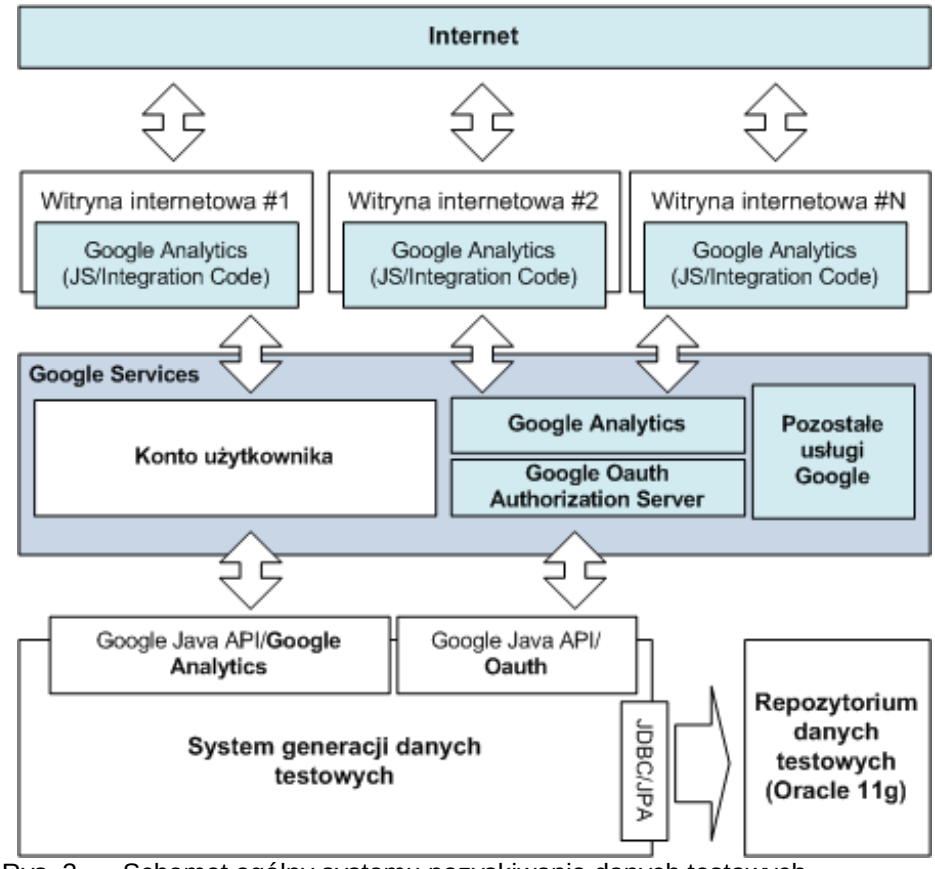


Rys. 1. Model danych oparty o pojedynczą tabelę

b) Zestaw tabel (Rys. 2.) odpowiadających swoją strukturą hurtowni danych opartej o model gwiazdy zawierających:
   - Centralną tabelę miar, w której znajdują się wartości poszczególnych metryk, klucz rekordu oraz klucze obce do tabel wymiarów.
   - Zestaw tabel pomocniczych zawierających dane słownikowe (wartości liczbowe, wartości tekstowe lub wartości w formie daty) oraz klucze służące do powiązania ich z danymi umieszczonymi w tabeli miar.

Rys. 2. Model danych oparty o schemat gwiazdy

Bazując na powyższych wymaganiach, stworzony został system generacji zbiorów testowych (Rys. 3.) oparty o dane pochodzące z Google Analytics [5], czyli hurtowni danych zawierającej dane dotyczące ruchu witryn internetowych.



Rys. 3. Schemat ogólny systemu pozyskiwania danych testowych

Schemat zaprezentowany na rysunku 3 prezentuje elementy składowe systemu generacji danych testowych.

Najważniejsze elementy systemu to:

a) **element wykonawczy**, zawierający logikę budującą mini-kostki danych (Rys. 2) na podstawie danych otrzymanych z Google Analytics. W celu przygotowania danych niezbędna jest integracja z centralnym repozytorium danych źródłowych, poprzez wykorzystanie bibliotek:

- Java OAuth, czyli bibliotekę języka Java odpowiedzialną za autoryzację użytkownika w systemach Google (dostęp do kont, dostęp do zasobów)
- Java Google Analytics API, czyli zestaw bibliotek Java odpowiedzialnych za dostęp do zasobów konta Google Analytics.

b) **konto Google** z którym związane są usługi Analytics, które umożliwia korzystanie z API Analytics (zdefiniowane parametry dostępu).

c) **szereg witryn**, bądź pojedyncza witryna internetowa z zainstalowanym kodem javascript służącym do zbierania danych na potrzeby Google Analytics.

Podsumowując niniejszy paragraf, architektura systemu pozwala na generowanie zbiorów testowych na podstawie danych o ruchu internetowym poprzez wykonywanie zapytań do hurtowni danych Google Analytics. Na podstawie tak zebranych danych wykonywana jest transformacja danych do postaci nowo tworzonych tabel słownikowych (odpowiedzialnych za przechowywanie danych o wymiarach) oraz tabeli centralnej. Tak stworzone dane pozwalają na generowanie pewnej formy mikro-kostek danych na podstawie zapytań generowanych przez użytkownika.

3 Dostęp do usług Google Analytics

W celu rozpoczęcia pracy nad powyżej opisanym projektem niezbędne było posiadanie testowych stron internetowych (witryny z naturalnym ruchem), a także założenie konta związanego z usługą Google Analytics. Po stworzenia konta w GA, na każdej ze stron zainstalowany został skrypt monitorujący ruch użytkowników.

Samo posiadanie ww. konta nie daje pełnej możliwości korzystania z API GA, stąd wyniknęła konieczność stworzenia odpowiedniego projektu z poziomi konsoli Google API's [6]. Dla tak stworzonego projektu, z poziomu panelu zarządzania dostępem do interfejsów (API Access), zdefiniowano klienta dzięki któremu stworzona przez nas aplikacja może się autoryzować. Aby nie wpisywać za każdym razem danych logowania pobieramy dane klienta w zakodowanej formie w postaci pliku JSON i wskazujemy je w naszej aplikacji jako client_secrets.json. Tak zbudo-

wane ustawienia dostępu pozwalają na implementację kodu pobierającego uchwyt do usługi GA z poziomu języka Java (Rys. 4.).

```
private Analytics initializeGA() throws Exception
{
    GoogleClientSecrets clientSecrets =  GoogleClientSecrets.load(
         JSON_FACTORY,
GaService.class.getResourceAsStream("/client_secrets.json")
    );

    FileCredentialStore credentialStore = new FileCredentialStore(
           new
File("D:\\CREADENTIAL_STORE\\credentials\\analytics.json"),JSON_FACTORY);

    GoogleAuthorizationCodeFlow flow = new
GoogleAuthorizationCodeFlow.Builder(
           HTTP_TRANSPORT, JSON_FACTORY, clientSecrets,
          Collections.singleton(AnalyticsScopes.ANALYTICS_READONLY)
    ).setCredentialStore(credentialStore).build();

    Credential credential = new AuthorizationCodeInstalledApp(
         flow, new LocalServerReceiver()
    ).authorize("user");

    Builder builder = new Analytics.Builder(
         HTTP_TRANSPORT, JSON_FACTORY, credential);

    return builder.setApplicationName( GOOGLE_API_APP_NAME).build();
}
```

Rys. 4. Pobranie uchwytu do usługi Google Analytic

Wykonanie powyższego kodu skutkuje zalogowaniem i pobraniem uchwytu do usługi Google Analytics. Poprawne wykonanie powyższego scenariusza pozwala na dalszą pracę z danymi poprzez zdefiniowanie odpowiednich zapytań o dane. W ramach procesu definiowania zapytania należy zdefiniować następujące kryteria:

a) identyfikator profilu (wskazanie witryny, ponieważ pod jedno konto GA może być przypięte wiele niezależnych domen);
b) zakres czasowy wyszukiwania statystyk (data od oraz data do);
c) lista metryk (głównie wartości numerycznych) poddanych agregacji w obrębie wymiarów;
d) lista wymiarów, według których budowane są metryki;
e) filtr danych;
f) sposób sortowania (wskazanie kolumn oraz kierunku sortowania)
g) limit rekordów możliwych do pobrania.

Zaprezentowana metoda executeQuery (Rys. 5.) na bazie uchwytu do usługi analytics dla aktualnie zalogowanego użytkownika (na bazie

konfiguracji JSON) oraz identyfikatora profilu zwraca: liczbę odwiedzin od pierwszego grudnia 2012 do pierwszego stycznia 2013 w kontekście miast zlokalizowanych w Polsce. Zwrócony rezultat ograniczony jest do 5 miast posortowanych malejąco względem liczby odwiedzin.

```
private GaData executeQuery(Analytics analytics, String profileId)
throws IOException
{
    Get query = analytics.data().ga().get(
                    "ga:" + profileId,   //profil
                    "2012-12-01",         // data od
                    "2013-01-01", // data do
                    "ga:visits" // metryki

                )
               .setDimensions("ga:city") //ustalenie wymiarów
               .setSort("-ga:visits,ga:city") //ustalenie kolejności
sortowania
               .setFilters("ga:country==Poland") //ustawienie filtrów
               .setMaxResults(5); //ustawienie limitu wierszy

    return query.execute();
}
```

Rys. 5. Kod budowy i wywołania zapytania

Powyższy kod źródłowy zwraca obiekt GaData zawierający zbiór rekordów będących rezultatem zapytania wykonanego po stronie serwerów Google. Wynik ten w prosty sposób można przełożyć na inną formę danych np. na tekstową formę (Rys. 6) lub w sposób bardziej złożony zamienić na dane w strukturze gwiazdowej.

Ostatnim krokiem niezbędnym do wykonania systemu generującego testowe zbiory danych było zamienienie wartości zgromadzonych w wyniku zapytania do GA do postaci jednej, bądź kilku tabel w lokalnym repozytorium (baza danych Oracle).

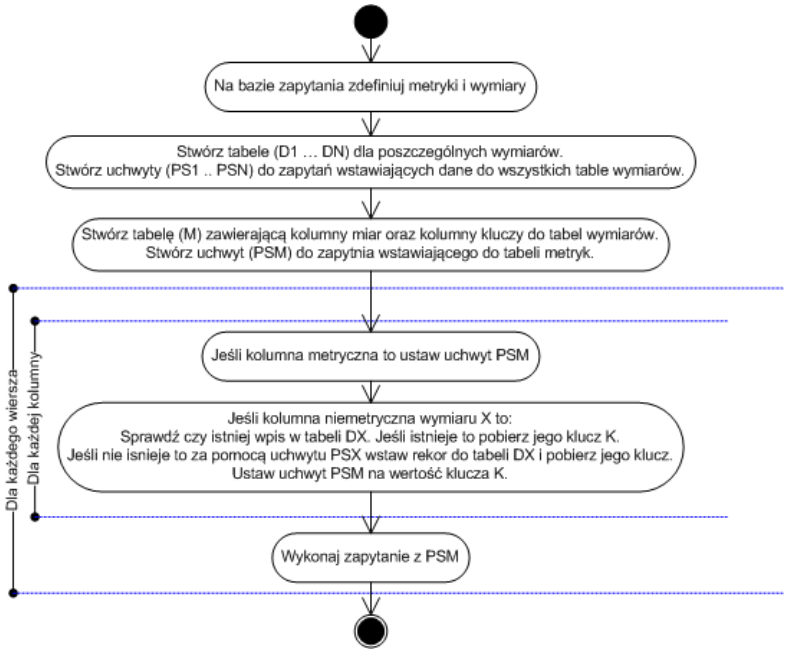
W tym celu w systemie wykonywana jest transformacja danych z postaci struktury GaData do postaci rekordów w nowo stworzonych tabelach danych w lokalnym repozytorium. Cała operacja wykonywana jest zgodnie ze schematem zaprezentowanym na rysunku 7 i wykorzystuje mechanizmy JDBC.

```
private static void printGaData(GaData results) {

    if (results.getRows() == null || results.getRows().isEmpty())
    {
      System.out.println("Brak danych.");
    }
    else
    {
```

```
        for (ColumnHeaders header : results.getColumnHeaders())
        {
            System.out.printf("%50s", header.getName());
        }
        System.out.println();
        for (List<String> row : results.getRows())
        {
            for (String column : row)
            {
                System.out.printf("%30s", column);
            }
            System.out.println();
        }
      System.out.println();
    }
  }
```

Rys. 6. Iteracja po zbiorze wynikowym zapytania do GA



Rys. 7. Transformacja danych GA do bazy danych

4 Podsumowanie

Podsumowując należy zwrócić uwagę na fakt, iż stworzono system generujący mikro-kostki danych oparte o dane rzeczywiste. Jako dane źródłowe przyjęto statystyki ruchu dla witryny internetowej, otrzymywane na bazie integracji autorskiego systemu z hurtownią danych Google

Analytics. Integracja ta była możliwa dzięki wykorzystaniu bibliotek udostępnionych w pakiecie Google API's. Zbiory wygenerowane za pomocą ww. systemu mogą stanowić alternatywne źródło danych testowych do ogólnie uznanych zbiorów testowych.

5 Literatura

[1] Kowalczyk-Niewiadomy A., Pelikant A., *Przetwarzanie Zapytań W Metajęzyku Naturalnym Przy Pomocy Algorytmów Rozmytych*, Studia Informatica, vol. 31, no. 2A(89) ISSN 0208-7286, pp. 477-488, 2010.

[2] Kowalczyk-Niewiadomy A., Pelikant A., *Fuzzy clustering algorithms for imprecise database queries*, NTAV/SPA, Łódź, 2012.

[3] Pelikant A., Kowalczyk-Niewiadomy A., *Zagadnienie grupowania w kontekście budowania zapytań rozmytych*, Bazy Danych Rozwój Metod i Technologii. pp. 175-186, 2008

[4] Niewiadomy D., Kowalczyk-Niewiadomy A., Pelikant A., *Automatyczne wyzwalacze audio dla mowy polskiej na bazodanowej platformie Oracle*, Studia Informatica, vol. 33, no.2B, ISSN 0208-7286, 2012

[5] Google Analytics official website: http://www.google.com/analytics/

[6] Konsola Google APIs website: https://code.google.com/apis/console/

TEST SET GENERATOR BASED ON ANONIMISED WEBSITE TRAFFIC DATA

Summary – The chapter presents flexible and comprehensive test set generator based on website traffic acquired from Google Analytics via Java Google Analytics API. This paper takes into consideration the problem of test set data preparation. Every research needs an experimental verification. In most cases, there is a need to define own test data set or use a commercial one. The chapter presents flexible and comprehensive test set generator based on real user website traffic acquired from Google Analytics via Java Google Analytics API. Due to the fact that the author of this chapter deals with the issues of imprecise queries (fuzzy queries [1] [2], clustering algorithms [3]), a set of test data should be reflecting the actual data from the real database system.

Presented approach allows every scientist to create his own test data set that reflects the real user trends.

Keywords: data acquisition, databases, Google Analytics, Java Google Analytics API