FEEDING YOUR DATABASE

Document identification

Feeding Your Database	

Status of reviews

Review n.	*Reason to review*	*Revised by*	*Date*
1	Updated for composite key generation	pabloj@users.sourceforge.net	26/01/2004

FEEDING YOUR DATABASE

This writing is a quick guide to DBMonster, a nice tool useful when you need to load you database with data.

I've recently used it to do check a "soon to be" datawarehouse performance.

Essentials (and where to get them)

In order to follow this tutorial you'll need to download and install DBMonster:

- DBMonster (http://dbmonster.kernelpanic.pl/), current version is **dbmonster-core-1.0.1.tar.gz** (25-05-2004)
- DBMonster has been updated to v 1.0.2, named **dbmonster-core-1.0.2.tar.gz** (08-01-2005), the Oracle example has been built with that version
- JDK 1.4 (that's what I'm using)

Installation

Installing DBMonster is very easy, just uncompressed it to my desktop. Another thing to do is copy the needed jdbc drivers to the ./lib folder, I did that for PostgreSQL's driver, as my datawarehouse is based on it.

Basic tasks

By referencing to "Basic tasks" I mean:

- Get the schema of one or more tables to be loaded
- Load it with data

Getting the schema:

The first thing to do is to create a dbmonster.properties in the main folder, i.e. in the same folder of dbmonster-core-1.0.1.tar.gz.

Mine looks like:

```
dbmonster.jdbc.driver=org.postgresql.Driver
dbmonster.jdbc.url=jdbc:postgresql://127.0.0.1/MONDRIAN_FOODMART
dbmonster.jdbc.username=postgres
dbmonster.jdbc.password=postgres
# for Oracle and other schema enabled databases
dbmonster.jdbc.schema=public
# maximal number of (re)tries
dbmonster.max-tries=1000
# default rows number for SchemaGrabber
dbmonster.rows=1000
```

```
# progres monitor class
dbmonster.progress.monitor=pl.kernelpanic.dbmonster.ProgressMonitorAdapter
```

This basically tells dbmonster:

1. The jdbc driver to be used
2. The database URL to connect to (complete of the database name)
3. The database user
4. The database password
5. The database schema to be used (for databases that have the concept of schema
6. The number of tries for inserts
7. The number of rows
8. The class used to monitor progress of operations

Once this file is completed we'll connect to the database and load two tables employee and department. The employee table contains a foreign key referencing the department table.

The two tables also have one primary key.

Having described the table structure it's time to fire up the command line:

We will use the supplied "dbmonster.bat" file to manage the program, it helps a lot by simplifying the whole classpath thing.

```
C:\Documents and Settings\Administrator\Desktop\dbmonster-core-1.0.1\bin>dbmonst
er --grab -t department employee -o c:/schema_dept_emp.xml

C:\Documents and Settings\Administrator\Desktop\dbmonster-core-1.0.1\bin>rem Bat
ch file to run dbmonster under Windows

C:\Documents and Settings\Administrator\Desktop\dbmonster-core-1.0.1\bin>rem Con
tributed by Peter De Bruycker
2005-01-04 15:08:10,418 INFO  SchemaGrabber - Grabbing schema from database. 2 t
ables to grab.
2005-01-04 15:08:10,629 INFO  SchemaGrabber - Grabbing table department. 50% don
e.
2005-01-04 15:08:10,889 INFO  SchemaGrabber - Grabbing table employee. 100% done
.
2005-01-04 15:08:10,889 INFO  SchemaGrabber - Grabbing schema from database
complete.

C:\Documents and Settings\Administrator\Desktop\dbmonster-core-1.0.1\bin>
```

We can take a peek at the generated file:

```
C:\>type schema_dept_emp.xml
<?xml version="1.0"?>
<!DOCTYPE dbmonster-schema PUBLIC "-//kernelpanic.pl//DBMonster Database
Schema
DTD 1.1//EN" "http://dbmonster.kernelpanic.pl/dtd/dbmonster-schema-1.1.dtd">
<dbmonster-schema>
  <name>Change me!</name>
  <table name="public.department" rows="1000">
    <key databaseDefault="false">
```

```
<generator type="pl.kernelpanic.dbmonster.generator.MaxKeyGenerator">
  <property name="columnName" value="department_id"/>
</generator>
```

But the real interesting thing is:

```
<column name="department_id" databaseDefault="false">
 <generator type="pl.kernelpanic.dbmonster.generator.ForeignKeyGenerator">
  <property name="columnName" value="department_id"/>
  <property name="fastMode" value="false"/>
  <property name="nulls" value="0"/>
  <property name="tableName" value="department"/>
 </generator>
</column>
```

DBMonster correctly recognized that column “department_id” of table “employee” has a foreign key constraint pointing to column “department_id” of table “department”.

Other niceties can be found reading the xml file:

```
<column name="birth_date" databaseDefault="false">
 <generator type="pl.kernelpanic.dbmonster.generator.DateTimeGenerator">
  <property name="endDate" value="1970-01-17 12:24:13.127 +0100"/>
  <property name="nulls" value="0"/>
  <property name="returnedType" value="timestamp"/>
  <property name="startDate" value="1970-01-01 01:00:00.0 +0100"/>
 </generator>
</column>
```

You can tune parameters like the generated date range, or in:

```
<column name="position_title" databaseDefault="false">
 <generator type="pl.kernelpanic.dbmonster.generator.StringGenerator">
  <property name="allowSpaces" value="true"/>
  <property name="excludeChars" value=""/>
  <property name="maxLength" value="255"/>
  <property name="minLength" value="0"/>
  <property name="nulls" value="10"/>
 </generator>
</column>
```

You can tune the maximum and minimum length of the generated string (the default values are taken from the field specifications found on the database).

The table defintion tag contains also the number of rows that the table will contain, even this parameter can be customized:

```
<table name="public.department" rows="1000">
```

In this example the table employee will have 1000 records in it, but the department table will only have 100 records in it, so I’m changing the row above to read:

```
<table name="public.department" rows="100">
```

Now I'm ready to run it:

```
C:\Documents and Settings\Administrator\Desktop\dbmonster-core-1.0.1\bin>dbmonst
er -s c:/schema_dept_emp.xml

C:\Documents and Settings\Administrator\Desktop\dbmonster-core-1.0.1\bin>rem Bat
ch file to run dbmonster under Windows

C:\Documents and Settings\Administrator\Desktop\dbmonster-core-1.0.1\bin>rem Con
tributed by Peter De Bruycker
2005-01-04 15:23:08,760 INFO  DBMonster - Let's feed this hungry database.
2005-01-04 15:23:08,990 INFO  DBCPConnectionProvider - Today we are feeding: Pos
tgreSQL 8.0.0beta1
2005-01-04 15:23:09,391 INFO  Schema - Generating schema <public>.
2005-01-04 15:23:09,391 INFO  Table - Generating table <public.department>.
2005-01-04 15:23:13,958 INFO  Table - Generation of table <public.department> fi
nished.
2005-01-04 15:23:13,968 INFO  Table - Generating table <public.employee>.
pl.kernelpanic.dbmonster.schema.SchemaException: No table <department> in this s
chema!
        at pl.kernelpanic.dbmonster.generator.ForeignKeyGenerator.generate(Forei
gnKeyGenerator.java:170)
        at pl.kernelpanic.dbmonster.schema.Column.generate(Column.java:174)
        at pl.kernelpanic.dbmonster.schema.Table.generate(Table.java:361)
        at pl.kernelpanic.dbmonster.schema.Schema.generate(Schema.java:181)
        at pl.kernelpanic.dbmonster.DBMonster.doTheJob(DBMonster.java:265)
        at pl.kernelpanic.dbmonster.Launcher.run(Launcher.java:193)
        at pl.kernelpanic.dbmonster.Launcher.main(Launcher.java:102)
```

The first lauch ended in this error, so I slightly modified the schema, adding:

```
<dbmonster-schema>
  <name>public</name>
```

and

```
    <column name="department_id" databaseDefault="false">
      <generator type="pl.kernelpanic.dbmonster.generator.ForeignKeyGenerator">
        <property name="columnName" value="department_id"/>
        <property name="fastMode" value="false"/>
        <property name="nulls" value="0"/>
        <property name="tableName" value="public.department"/>
      </generator>
    </column>
```

After this I run it again:

```
C:\Documents and Settings\Administrator\Desktop\dbmonster-core-1.0.1\bin>dbmonst
er -s c:/schema_dept_emp.xml

C:\Documents and Settings\Administrator\Desktop\dbmonster-core-1.0.1\bin>rem Bat
```

```
ch file to run dbmonster under Windows

C:\Documents and Settings\Administrator\Desktop\dbmonster-core-1.0.1\bin>rem Con
tributed by Peter De Bruycker
2005-01-04 15:24:03,409 INFO  DBMonster - Let's feed this hungry database.
2005-01-04 15:24:03,639 INFO  DBCPConnectionProvider - Today we are feeding: Pos
tgreSQL 8.0.0beta1
2005-01-04 15:24:04,030 INFO  Schema - Generating schema <public>.
2005-01-04 15:24:04,030 INFO  Table - Generating table <public.department>.
2005-01-04 15:24:08,416 INFO  Table - Generation of table <public.department> fi
nished.
2005-01-04 15:24:08,416 INFO  Table - Generating table <public.employee>.
2005-01-04 15:25:07,561 INFO  Table - Generation of table <public.employee> fini
shed.
2005-01-04 15:25:07,561 INFO  Schema - Generation of schema <public> finished.
2005-01-04 15:25:07,561 INFO  DBMonster - Finished in 1 min. 4 sec. 152 ms.

C:\Documents and Settings\Administrator\Desktop\dbmonster-core-1.0.1\bin>
```

The most notable thing is that the foreign key generator added some overhead.

Now I'm checking that the rows have been inserted in the expected number and according to the defined constraints.

```
select count(*) from department;
```

Leads to a result of 100 records.

```
select count(*) from employee;
```

Leads to a result of 1000 records, as expected, and a:

```
select count(*) from employee inner join department on employee.department_id =
department.department_id
```

Leads to a result of 1000 as expected, as the foreign key constraint has been respected by the load process.

With a database full of data it's now possible to test query performance and impact of massive load of data on it.

Part II – Struggling with Oracle

A more throughtful test run on an Oracle database lead to new experiences (and some troubles) that I'm describing here:

The first challenge I faced is that dbmonster does not generate an usable schema for Oracle, the –grab option generates a schema with only the table list, I had to add all table s by hand (The author is working on this).

Then I had to load a table with a PK of type varchar (string for java), so I added it to the schema with the appropriate generator:

```
<table name="TD" rows="100">
 <key databaseDefault="false">
  <generator type="pl.kernelpanic.dbmonster.generator.StringKeyGenerator">
   <property name="columnName" value="COD_TD"/>
        <!-- if start value is set to 0 (zero)
             start value is grabbed from database
             using select max(columnName) from table -->
        <property name="startValue" value="aaaaaaaa"/>
  </generator>
 </key>
```

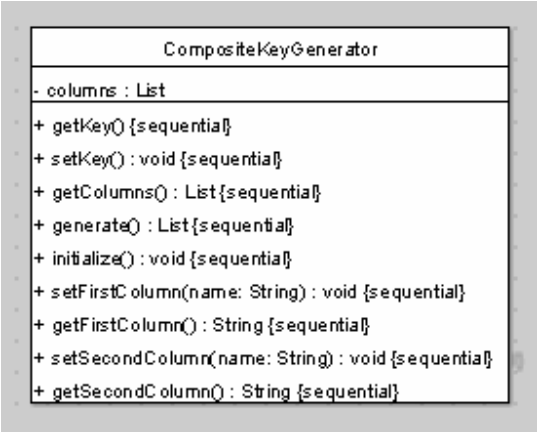
This helped discover a bug that made the generator unusable, promptly fixed by the author (see http://sourceforge.net/forum/forum.php?thread_id=1216239&forum_id=249327 for reference).

Next I found out that every referenced table must be in the schema, even if it's not considered in the load job, this error came out because of lazy me adding only some tables by hand, but it was very useful, as I learned that you can actually skip loading a table by simply specifying rows="0".

The last and most "fun" problem was the need for a multiple key generator, i.e. I had to generate a PK for a table made of three columns, the first and the second column were both FK to two other tables and together made the PK.

I started out from the example found on site and extended it to suit my needs, as it refers to a case in which the key is made of two fields of which only one is a FK. Another little challenge was to find out the way to extract a random record in Oracle, which is different from the supplied example.

An UML schema of the resulting work:



After building the correct class and adding it to the jar file I faced more errors, all due to lacking parameters in the schema file, after some trial and error I came to the following solution:

```
<table name="OSP" rows="10">
 <key databaseDefault="false">
  <generator type="pl.kernelpanic.dbmonster.misc.CompositeKeyGenerator">
        <property name="firstColumn" value="IR"/>
        <property name="secondColumn" value="IU"/>
  </generator>
 </key>
```

In which the generator type points to my custom class CompositeKeyGenerator and the two properties set the names of the two columns that make the key.

With all this in place, and writing the schema by hand, I've finally been able to set up the loading process for my brand new Oracle based datawarehouse :-)

Note that in this case the "rows" parameter can be meaningless since the maximum number of rows depends on the maximum number of possible combinations of key values.

Listing for CompositeKeyGenerator.java (adapted from the example provided by Piotr Maj):

```
/*
 * Copyright 2004 Piotr Maj
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *   http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package pl.kernelpanic.dbmonster.misc;

import java.sql.ResultSet;
import java.util.ArrayList;
import java.util.List;

import pl.kernelpanic.dbmonster.DBMonster;
import pl.kernelpanic.dbmonster.DBMonsterContext;
import pl.kernelpanic.dbmonster.connection.ConnectionProvider;
import pl.kernelpanic.dbmonster.connection.Transaction;
import pl.kernelpanic.dbmonster.generator.Initializable;
import pl.kernelpanic.dbmonster.generator.KeyGenerator;
import pl.kernelpanic.dbmonster.schema.Column;
import pl.kernelpanic.dbmonster.schema.Key;
import pl.kernelpanic.dbmonster.schema.Table;
```

```
/**
 * @author Piotr Maj &lt;ant@kernelpanic.pl&gt;
 *
 * $Revision: 1.1 $ $Date: 2004/06/11 10:37:59 $
 */
public class CompositeKeyGenerator implements KeyGenerator, Initializable {

    private Key key;
    private DBMonsterContext context;
    private Column firstColumn;
    private Column secondColumn;
    private List columns;
    private ConnectionProvider connProvider;

    //pabloj@users.sourceforge.net
    //added to try to solve the problem I reported on sourceforge
    public Key getKey()
    {
        return key;
    }

    public void setKey(Key key) {
        this.key = key;
    }

    public List getColumns() {
        if (columns == null) {
            columns = new ArrayList();
            columns.add(firstColumn);
            columns.add(secondColumn);
        }
        return columns;
    }

    public List generate() throws Exception {

        // first: be sure that table invoices is
        //        already generated
        //Table invTable = key.getTable().getSchema().findTable("invoices");
        //check that referenced tables are already generated
        Table risSpotTable = key.getTable().getSchema().findTable("RS");
        //invTable.generate();
        risSpotTable.generate();
        Table uTable = Key.getTable().getSchema().findTable("U");
        uTable.generate();
        //checking for both tables will avoid the null pointer exception I mistakenly reported
        //in the project's forums

        // Transaction is user-friendly wraper for pure JDBC obscure methods.
        Transaction tx = null;
        try {
            tx = new Transaction(connProvider);
            tx.begin();
```

```
                //second: get the random invoice id from the invoices
                //        table.
                //ResultSet rs = tx.executeQuery("SELECT invno FROM invoices ORDER BY
random() LIMIT 1");
                //rs.next();
                //int invno = rs.getInt("invno"); // this is our random invoice id
                        //pabloj@users.sourceforge.net
                        //query to get one random record in Oracle
                        ResultSet rs = tx.executeQuery("SELECT * FROM (SELECT rs.id_r AS
id_r FROM rs rs  ORDER BY dbms_random.value) WHERE ROWNUM < 2");
                        rs.next();
                        int risSpotno = rs.getInt(1); //this is our random id_risposta

                // third: generate itemno using MAX + 1 strategy
                //rs = tx.executeQuery("SELECT max(itemno) + 1 FROM invoice_items where
invno = " + invno);
                //int itemno = 1;
                //if (rs.next()) {
                //    itemno = rs.getInt(1);
                //}
                //pabloj@users.sourceforge.net
                //get a random record from the second table
                rs = tx.executeQuery("SELECT * FROM (SELECT u.id_u AS id_u FROM u u
ORDER BY dbms_random.value) WHERE ROWNUM < 2");
                        rs.next();
                        int usrno = rs.getInt(1);

                // we have all what we need so just set the column values
                firstColumn.setValue(new Integer(risSpotno));
                secondColumn.setValue(new Integer(usrno));

                tx.commit();
            } catch (Exception e) {
                throw e;
            } finally {
                tx.close();
            }

            return getColumns();
        }

        public void initialize(DBMonsterContext ctx) throws Exception {
            context = ctx;
            connProvider                    =                    (ConnectionProvider)
context.getProperty(DBMonster.CONNECTION_PROVIDER_KEY);
        }

        public void setFirstColumn(String name) {
            firstColumn = new Column();
            firstColumn.setName(name);
            firstColumn.reset();
        }
        public String getFirstColumn() {
            if (firstColumn != null) {
                return firstColumn.getName();
            }
```

```
        return null;
    }

    public void setSecondColumn(String name) {
        secondColumn = new Column();
        secondColumn.setName(name);
        secondColumn.reset();
    }
    public String getSecondColumn() {
        if (secondColumn != null) {
            return secondColumn.getName();
        }
        return null;
    }

}
```