Command Line DataShaper, version 1.0.0

User Guide

©2010-2011 Wan Lee,
wan5332@gmail.com

DataShaper User Guide

Contents

DataShaper User Guide

Thank you for choosing Command Line DataShaper. Please review license.txt file to make sure you understand and agree to the terms and conditions before using the software.

Introduction

Command Line DataShaper version 1.0 (`datashaper.jar`) is a runtime engine used for populating volume data in JDBC-compliant relational database management systems (RDBMS).

The runtime has been lab-tested and certified to run against ORACLE, IBM DB2, MYSQL, POSTGRESQL and MSSQLSERVER databases.

The data shape definition is expressed as XML configuration file, which is passed into DataShaper runtime. The XML configuration file is structured according to DataShape.dtd.

For a quick overview of the application, you can look at the PPT slides in DataShaper Introduction.pdf file.

Download

The zip file `datashaper-v1.0.0-bin.zip` can be downloaded from www.psrtoolkit.com.
The source code of DataShaper can be downloaded from
https://github.com/wanlee/datashaper.git

Installation

No special installer is required. Simply unzip the zip file to the target directory. The directory should have the following files and sub-directories after unzipping:

```
.
|-- DataShape.dtd
|-- datashaper.jar
|-- docs
|   |-- DataShaper Introduction.pdf
|   `-- DataShaper User Guide.pdf
|-- drivers
|-- license.txt
|-- readme.txt
|-- samples
|   |-- db2
|   |   |-- DB2Example*.xml files
|   |   |-- DB2HowToStart.txt
|   |   `-- dbscript
|   |       |-- actproj-db2-identity-column-option.ddl
|   |       `-- actproj-db2.ddl
|   |-- mssqlserver
|   |   |-- MSSQLServerExample*.xml files
|   |   |-- MSSQLServerHowToStart.txt
|   |   `-- dbscript
|   |       `-- actproj-mssqlserver.ddl
```

```
|   |-- mysql
|   |   |-- MYSQLExample*.xml files
|   |   |-- MYSQLHowToStart.txt
|   |   `-- dbscript
|   |       `-- actproj-mysql.ddl
|   |-- oracle
|   |   |-- ORCLExample*.xml files
|   |   |-- ORCLHowToStart.txt
|   |   `-- dbscript
|   |       |-- actproj-orcl-alphanumeric-rowid-implicit-sequence-with-
trigger.ddl
|   |       `-- actproj-orcl.ddl
|   `-- postgresql
|       |-- POSTGRESQLExample*.xml files
|       |-- POSTGRESQLHowToStart.txt
|       `-- dbscript
|           |-- actproj-postgres-implicit-sequence.ddl
|           `-- actproj-postgres.ddl
```

System Dependencies

DataShaper runtime requires JRE version 6 or later and the vendor-specific JDBC driver.

The table below shows the version of drivers that are currently certified:

DB Name	JDBC Driver
Oracle	ojdbc5.jar
IBM	db2jcc4.jar
PostgreSQL	postgresql-8.4-701.jdbc4.jar
MySQL	mysql-connector-java-5.1.13-bin.jar
MSSQLServer	sqljdbc4.jar

Note: Please go to the vendor's site to download the driver and agree to their licensing terms and conditions before using their software.

Command Line Execution

It is easier to explain by using an example below:

```
java -classpath datashaper.jar:drivers/ojdbc5.jar
com.psrtoolkit.datashaper.DataShaper
-c samples/oracle/ORCLExampleUseCase1SmallScale.xml -u actproj -p actproj
```

Where `com.psrtoolkit.datashaper.DataShaper` is the DataShaper runtime's java class name.

Where the command line option:
`-c` is the data shape definition XML configuration file
`-u` is the user login name
`-p` is the user password

Where the JVM option:

`-classpath`, a parameter used with java command, is where the JDBC driver is located; the example shows ojdbc5.jar from Oracle is located in `/drivers` directory under the location where `datashaper.jar` is launched from.

`datashaper.jar` and the driver jar are separated with character ';' (on Windows) or ":" (on Linux).

The location of the driver can be in any directory (not necessarily `/drivers`), but it must be referenced in the `-classpath` option.

Command Line Output

A snapshot of an execution run is shown below:

```
Copyright(C) 2010-2012, Wan Lee, wan5332@gmail.com

com.psrtoolkit.datashaper.DataShaper -c
samples/mysql/MYSQLExampleSimpleCreateCustomers.xml -u actproj -p
*******

Time elapsed: 365 milliseconds
Connected to DB, begin populating data in tables...
*************** Summary of records created: **************
Completed run and reached maxTotalCountLimit: 100
      CUSTOMER: 100
DataShaper completed in 447 milliseconds
May 15, 2012 8:36:59 PM
```

Functionality

DataShaper performs the following functions:

1. Create new records based on new ids from the database.
2. Create new records based on existing ids provided by user.
3. Create new records based on the specified filters and percentage of parent records and choose sequentially or randomly the parent ids for the association.
4. Perform query for existing records in parent tables and supply the ids to child tables for the association.
5. Support 1:1, 1:M, and M:M relationship data population based on data shape needs.
6. Support de-normalized column and List of Values (LOVs) column population.
7. Support both numeric and alphanumeric ids.

Samples

You can learn how to use the configuration and runtime by using the samples.

Samples are available under `/samples` directory. Below `/samples` are sub-directories such as `oracle, db2, mysql, postgresql, and mssqlserver`. Each sub-directory contains the sample database schema DDL file, a number of XML configuration files, and `how-to-`

start.txt file. Copy the DataShape.dtd into the directory where you create or maintain your own XML configuration files.

The sample XML configuration files that can be found in the sub-directories are as follows:

Name	Remarks
xxxExampleSimpleCreateCustomers.xml	Simple data population example
xxxExampleMany2ManyRelationship.xml	M:M relationship data population
xxxExampleOne2ManyRelationship.xml	1:M relationship data population
xxxExampleOne2OneRelationship.xml	1:1 relationship data population
xxxExampleSeedData.xml	Seed data population
xxxExampleDenormColumnAndPickListColumn.xml	De-normalized columns and pick-list
xxxExampleUseQuery.xml	Using query in data population
xxxExampleUseCasesDataPreparation.xml	Preparing data for other runs
xxxExampleUseCase1SmallScale.xml	Use case #1 data shape
xxxExampleUseCase2SmallScale.xml	Use case #2 data shape
xxxExampleUseCase3SmallScale.xml	Use case #3 data shape
xxxExampleUseCase1and2and3Combined.xml	Combined data shape from use case 1, 2 and 3

In the samples, the database schema is called 'actproj', which has about thirteen tables currently. The respective xxxExamplexxx XML configuration files have references to those tables.

Some examples of alpha-numeric record ids usage can be found in xxx**Example**xxx**AlphaNumericRowId**xxx.xml files also.

Data Shape Definition - XML Configuration

It defines the data shape requirements for the runtime engine to execute. The definition is expressed in the form of XML configuration. The configuration file XML structure conforms to DataShape.dtd. Again, you can refer to the sample XML configuration files on how the structures look like.

DataShape Markups

At the highest level, the definition consists of three major areas as follows:

```
<datashape>
  <database type="vendor-name">
  </database>
  <population-spec>
  </population-spec>
  <tables-spec>
  </tables-spec>
</datashape>
```

Where <database> specifies the vendor type and connection info to the database

Where <population-spec> specifies the target tables, size of population, computational method, table relationship, foreign keys selection method, and etc.

Where <tables-spec> specifies details of INSERT or SELECT statement, bind variables for input data, place-holders for runtime data substitution , optional use of global or local sequence, optional query spec, and etc.

Definition of <database> Usage

It is easier to explain with an example here:

```
<database type="ORCL">
  <connection>jdbc:oracle:thin:@localhost:1521:xe</connection>
</database>
```

The examples specifies that the vendor database is ORCL (which stands for Oracle) and its JDBC's connection string is `jdbc:oracle:thin:@localhost:1521:xe` where the hostname or ip address is localhost or 127.0.0.1, and the port number is 1521. The SID is `xe`.

Currently the legitimate type values are: *DB2, ORCL, MSSQLSERVER, MYSQL, POSTGRESQL*

The table below illustrates more examples of JDBC connection strings:

Database	JDBC Connection String
DB2	jdbc:db2://localhost:50000/actproj
ORCL	jdbc:oracle:thin:@localhost:1521:xe
MSSQLSERVER	jdbc:sqlserver://localhost:1433;DatabaseName=actproj
MYSQL	jdbc:mysql://localhost:3306/ACTPROJ
POSTGRESQL	jdbc:postgresql://localhost:5432/actproj

Definition of <population-spec>

It is easier to show an example and explain what it means.

```
<population-spec>
  <table-populationname="USR">
    <size>100</size>
    </table-population>
  <table-populationname="LOC">
    <size>100</size>
  </table-population>
  <table-populationname="USR_LOC">
    <relationship>M:M</relationship>
    <size>by-derivation</size>
    <fkey-providers>
      <fkey-providername="USR">
        <percent>10</percent>
        <random-select>true</random-select>
      </fkey-provider>
      <fkey-providername="LOC">
        <percent>10</percent>
        <random-select>true</random-select>
      </fkey-provider>
    </fkey-providers>
```

```
  </table-population>
</population-spec>
```

The example shows the population of data will occur in three tables – USR, LOC, and USR_LOC. The USR and LOC tables will populate 100 new records each.

The size of population for USR_LOC, which is the intersection table to represent the M:M relationship (as specified by `<relationship>M:M</relationship>)`, between USR AND LOC (which are the designated foreign key providers, as specified in `<fkey-providers>`), will be computed (as specified by `<size>by-derivation</size>)`, based on 10% percent of the total new records in USR and 10% of the total new records in LOC (as specified by `<percent>10</percent>`).

The selection of parent IDs from USR and LOC will be random (as specified by `<random-select>true</random-select>`). The USR_LOC table will end up with 10 x10 = 100 new records; and for each randomly selected parent id from USR, there are 10 randomly selected records from LOC to be associated with. Due to performance reason, the percentage value, if greater than 50%, will cause the `<random-select>true</random-select>` value to be reset to `false` at runtime. When `<random-select>` is false, the selection is sequential and the first 10 records from the parent table will be linked. In this example, if both parent tables have `<random-select>` set to false, the first ten records from USR will be associated with first ten records from LOC, producing a total of 100 new records in USR_LOC. There are four possible combinations of parent id selection between the two tables.

The `<table-population name="">` attribute specifies the logical name for the table to be populated with data. The convention is to use the physical table name; however, if the same table is specified in multiple table populations, it is necessarily to distinguish them by giving a different name for each of the table populations.

The order of `<fkey-provider>` is significant in an M:M relationship. The first one always represents the table on the left side of the relationship. The order determines the direction and implicitly influences how many records from the right will be associated with the records from the left.

There are times when the users want to leverage existing records in the database to provide the parent ids to the child tables, the table-population can specify `<size create-new="false">by-query</size>` as shown in the example here:

```
      <table-population name="USR">
        <size create-new="false">by-query</size>
      </table-population>
```

When seeded-data with user-supplied fixed ids are desired, the table-population can specify `<size>by-input-data</size>` as shown in the example below:

```
      <table-population name="USR">
        <size>by-input-data</size>
      </table-population>
```

In the 1:M relationship, the population spec can look like this:

```
    <table-population name="ACTIVITY">
      <size>100</size>
      <relationship>1:M</relationship>
      <fkey-providers>
        <fkey-provider name="USR">
          <!-- or you can specify <percent>10</percent> -->
          <size>10</size>
          <random-select>true</random-select>
        </fkey-provider>
      </fkey-providers>
    </table-population>
```

The child table ACTIVITY will have 100 new records created, and they will reference 10 different parent records (i.e. 10 % of 100 USR). That works out to be 10 children records referencing a common parent record.

In a 1:1 relationship, the population spec can look like this:

```
<table-population name="PROFILE">
  <relationship>1:1</relationship>
  <size>100</size>
  <fkey-providers>
    <fkey-provider name="USR">
      <percent>100</percent>
      <random-select>false</random-select>
    </fkey-provider>
  </fkey-providers>
</table-population>
```

The child table PROFILE will have 100 new records created, and they will reference 100 parent records (100% of 100 USR. That works out to be one child record referencing one unique parent record. It will be a configuration error if the number of children records does not match exactly one to one with those of the parent.

Notion of <fkey-secondary-provider>

M:M or 1:M relationship involves at least three or two tables respectively in the main relationship. The use of `<fkey-provider>` is for the entities involved in the main relationship. In the example below, USR and LOC are the main relationships to USR_LOC. However, the use of `<fkey-secondary-provider>` is to capture additional relationship with other entities. In the example, ADDRESS is the secondary relationship to USR_LOC; and only one randomly selected address will be associated with a USR_LOC record.

```
    <table-population name="USR_LOC">
      <size>by-derivation</size>
      <relationship>M:M</relationship>
      <fkey-providers>
        <fkey-provider name="USR">
          <percent>10</percent>
          <random-select>true</random-select>
        </fkey-provider>
        <fkey-provider name="LOC">
          <!-- <filter>TIMEZONE='UTC+0'</filter> -->
          <percent>10</percent>
          <random-select>true</random-select>
```

```
        </fkey-provider>
        <!-- USR_LOC can have relationship with other entities such as
ADDRESS, but such relationship is not the main one -->
        <fkey-secondary-provider name="ADDRESS">
            <size>1</size>
          <random-select>true</random-select>
        </fkey-secondary-provider>
      </fkey-providers>
    </table-population>
```

List of Values (LOVs)

LOVs are common features in applications; for examples, a list of State Abbreviations, Country Codes, and etc. Picklist is a foreign value provider that is having a pseudo relationship to the consuming entity. The selection of an attribute value from a field in the picklist and storing of the selected value in a column in the consuming table can be achieved with `<picklist>` configuration.

In `<population-spec>` , the configuration specifies that each of the 100 USR records will have a randomly selected foreign value provided by *`USER_TYPE_PICKLIST`* :

```
    <table-population name="USR">
      <size>100</size>
      <relationship>1:1</relationship>
      <fkey-providers>
        <picklist name="USER_TYPE_PICKLIST">
          <size>1</size>
          <random-select>true</random-select>
        </picklist>
      </fkey-providers>
    </table-population>
```

In `<tables-spec>`, the configuration specifies that the picklist field `NAME` in *`USER_TYPE_PICKLIST`* will be picked and stored in the bind variable for `USR_TYPE` column:

```
    <table name="USR">
      <sql-statement>INSERT INTO ACTPROJ.USR (USR_ID, EMAIL, NAME, USR_TYPE,
SYS_TENANT_ID, DESCRIPTION) VALUES (?, ?, ?, ?, ?, ?)</sql-statement>
      <data-type>...</data-type>
      <input-data>
        <row>'${ds-nextval}', 'johndoe533@yahoo.com', 'John Doe'
, '${ds-picklist}= USER_TYPE_PICKLIST.NAME', '0', 'A software developer from
Sunnyvale, California.'</row>
      </input-data>
    </table>
```

Notion of Filter

Filters are rules to produce a fine-grain data set within a foreign key provider. It is a powerful feature to associate a filtered list of records with other entities. Below is an example from `ExampleUseCase1SmallScale.xml` configuration file:

```
    <table-population name="USR_SN_1">
      <size>by-derivation</size>
      <relationship>M:M</relationship>
```

```
      <fkey-providers>
        <fkey-provider name="USR">
          <filter>AGE &gt; 18 AND AGE &lt; 55</filter>
          <percent>10</percent>
          <random-select>true</random-select>
        </fkey-provider>
        <fkey-provider name="SN">
          <filter>TYPE=&apos;technology&apos;</filter>
          <percent>5</percent>
          <random-select>true</random-select>
        </fkey-provider>
      </fkey-providers>
    </table-population>
```

In the example, USR records are filtered by demographic criteria such as AGE before a 10% selection on the pool is applied. Likewise, SN records can be filtered based on a different criterion. DataShaper rules engine supports simple AND and OR operations on rules.

Definition of <tables-spec>

At the high level, the <tables-spec> element has a structure that looks like this:

```
<tables-spec>
  <shared-sequence name="some-global-sequence"></shared-sequence>
  <table name="SOME-TABLE-NAME">
    <local-sequence name="some-local-sequence"></local-sequence>
    <sql-statement>some insert or select statement<sql-statement>
    <data-type>some data types</data-type>
    <input-data>
      <row>some sample data</row>
            …
      <row>some other sample data</row>
    </input-data>
  </table>
  <tablename="ANOTHER-TABLE-NAME">
      …
  </table>
</tables-spec>
```

The <tables-spec> has at least one and optionally numerous table definitions and an optional <shared-sequence> if it exists in the schema. Each table definition can specify an optional <local-sequence> as an override on the shared sequence (which is quite common in PostgreSQL schema using SERIAL per table). Some databases don't support Sequence, hence the <shared-sequence> and <local-sequence> are optional. The <sql-statement> can specify either an INSERT or SELECT statement. The insert SQL statement can have a number of bind variables marked with '?' and separated by ','. The <data-type> specifies the equivalent JDBC/JAVA data types that the bind variables will be mapped to, which correspond to the SQL types expected by the columns bound to the variables. A listing of the data type mappings between Java and SQL can be found in the appendix. The <input-data> consists of numerous rows of sample data or reference data, which are acting as the source for data generation. The <row> specifies the data that will be used by the bind variables in the SQL statement. There is

currently no hard limit on the number of <row> in <input-data>. The rows of data are repeatedly used, in a sequential fashion, during the data population. One should carefully consider the characteristics of the data and craft a representative sample for mass production.

It is easier to understand the above description by looking at the example below:

```
<tables-spec>
  <shared-sequence name="ACTPROJ"></shared-sequence>
  <table name="USR">
    <sql-statement>INSERT INTO ACTPROJ.USR (USR_ID, EMAIL, NAME, USR_TYPE,
SYS_TENANT_ID, DESCRIPTION) VALUES (?, ?, ?, ?, ?, ?)</sql-statement>
    <data-type>java.math.BigDecimal, java.lang.String, java.lang.String,
java.lang.String, java.math.BigDecimal, java.lang.String</data-type>
    <input-data>
      <row>'${ds-nextval}', 'johndoe123@yahoo.com', 'John Doeee ${ds-current-
record-id}', 'user', '0', 'A developer from Sunnyvale, California.'</row>
     <row>'${ds-nextval}', 'johndoe345@yahoo.com', 'John Doerrr ${ds-current-
record-id}', 'user', '0', 'A musician from San Francisco, California.'</row>
     <row>'${ds-nextval}', 'johndoe678@yahoo.com', 'John Doeoe ${ds-current-
record-id}', 'user', '0', 'A baseball player from Oakland, California.'</row>
    </input-data>
  </table>
  <table name="LOC">
    <sql-statement>INSERT INTO ACTPROJ.LOC (LOC_ID, NAME) VALUES (?, ?)</sql-
statement>
    <data-type>java.math.BigDecimal, java.lang.String</data-type>
    <input-data>
      <row>'${ds-nextval}', 'Main Residence${ds-default-signature}'</row>
    </input-data>
  </table>
  <table name="USR_LOC">
    <sql-statement>INSERT INTO ACTPROJ.USR_LOC (USR_LOC_ID, LOC_ID, USR_ID)
VALUES (?, ?, ?)</sql-statement>
    <data-type>java.math.BigDecimal, java.math.BigDecimal,
java.math.BigDecimal</data-type>
    <input-data>
      <row>'${ds-nextval}', '${ds-fkey-provider}=LOC', '${ds-fkey-
provider}=USR'</row>
    </input-data>
  </table>
</tables-spec>
```

Where appropriate, the special characters in the data are properly 'escaped' to comply with XML rules; e.g. apostrophe character is escaped with `&apos;` and double quote character is escaped with `&quot;`. In this document, they are deliberately left 'un-escaped' for readability. The `/samples` directory has more examples of vendor-specific <tables-spec> configuration.

Where sequence is specified, the <row> will have `${ds-nextval}` as the first item in the list of data items and an extra bind variable '?' in the INSERT statement. The Runtime Engine will pre-fetch a block of 100 new ids, one block at a time from the sequence in the database and assign them to the new rows for batch insertion. `${ds-nextval}` is the placeholder to be replaced with the row id. Place holders such as `${ds-fkey-provider}=LOC` and `${ds-fkey-provider}=USR`

indicate that the row will get its foreign key values from the parents, which in this example, are LOC and USR.

Place Holders

Place holders in the form of `${place-holder-name}` are used widely in the input data `<input-data>` section of the XML configuration file. They are substituted with data at runtime. They serves the purpose of achieving uniqueness of data, resolving foreign-key references, and providing information about the run in the records. The table below shows the complete list:

Place Holder	Purpose	Example
${ds-nextval}	Bound to first bind variable in INSERT statement; to be assigned a new row id. Used when a Sequence object is available.	<row>'${ds-nextval}', 'MayDoee123@yahoo.com&apos</row>
${ds-fkey-provider}=TARGET-TABLE	Bound to a bind variable relative to its position in the SQL statement; to be assigned a parent id from the specified table. The provider's relationship to the consuming table is main or primary.	<row>'${ds-nextval}', **'${ds-fkey-provider}=SN', '${ds-fkey-provider}=USR'**</row>
${ds-row-id}	Bound to first bind variable in INSERT statement; to be assigned a user-supplied fixed id. ${ds-row_id} and ${ds-nextval} are mutually exclusive.	<row>**'${ds-row-id}'**, ' MayDoee123@yahoo.com' </row>
${ds-uuid}	To be assigned a uuid generated by java.util.UUID; only the absolute value of the hashcode (an integer value) will be used.	<row>'${ds-nextval}', 'May Doee **${ds-uuid}**'</row>
${ds-current-record-id}	To be assigned the same new id as the record; only applicable if block of new ids can be pre-fetched by using a sequence. Not applicable to tables using Identity column or Auto Increment. Default value 0.	<row>'${ds-nextval}', 'May Doee **${ds-current-record-id}**'</row>
${ds-default-signature}	Default signature string generated by runtime engine. To distinguish data generated by DataShaper from others. Default value " -DataShaper	<row>'${ds-nextval}', 'A software developer from Sunnyvale, California. **${ds-default-signature}** '</row>

	" Notice the space character at both ends.	
${ds-user-signature}	User-defined signature, specified through optional command line parameter –s.	-s run_by_john
${ds-current-date}	To be assigned current date	<row>'${ds-nextval}', 'A software developer from Sunnyvale, California. **${ds-current-date}** '</row>
${ds-current-datetime}	To be assigned current date and time	<row>'${ds-nextval}', 'A software developer from Sunnyvale, California. ${**ds-current-datetime}** '</row>
${ds-fkey-secondary-provider}=TARGET-TABLE	Bound to a bind variable relative to its position in the SQL statement; to be assigned a parent id from the specified table. The provider's relationship to the consuming table is secondary.	<row>'${ds-fkey-provider}=LOC', '${ds-fkey-provider}=USR', '${ds-denorm-val}=LOC.NAME', '${ds-denorm-val}=LOC.TIMEZONE', '**${ds-fkey-secondary-provider}=ADDRESS**', '${ds-denorm-val}=ADDRESS.ZIPCODE'</row>
${ds-denorm-val}= TARGET-TABLE.TARGET-COLUMN	Bound to a bind variable for the de-normalized column that will take and store a foreign attribute value from the provider. The placeholder should only appear after ${ds-fkey-provider} or ${ds-fkey-secondary-provider}. This will ensure the foreign key id is captured first before any foreign attribute values can be stored as de-normalized values in the consuming table.	<row>'${ds-fkey-provider}=LOC', '${ds-fkey-provider}=USR', '**${ds-denorm-val}=LOC.NAME**', '${**ds-denorm-val}=LOC.TIMEZONE**', '${ds-fkey-secondary-provider}=ADDRESS', '**${ds-denorm-val}=ADDRESS.ZIPCODE**'</row>
${ds-picklist}= TARGET-TABLE.TARGET-COLUMN	Bound to a bind variable for the LIST OF VALUES (LOVs) column that will take and store a foreign attribute value from the provider. Although its assignment format looks similar to ${ds-denorm-val}, it differs in not requiring that the record id value from the foreign provider be used.	<row>'johndoe533@yahoo.com', 'John Doe ${ds-uuid}', '**${ds-picklist}=USER_TYPE_PICKLIST.NAME**', '0', 'A software developer from Sunnyvale, California'</row>
${ds-textfile}=TEXTFILE.extension	Read the content of the text file and bound it to the bind variable mapped to the CLOB column in the table. Consume memory if file is large. May degrade	<row>'${ds-nextval}', '**${ds-textfile}=sampleBigText1.txt**', '${ds-binaryfile}=design-doc\DataShaper User

	performance.	Guide.pdf'</row>
${ds-binaryfile}=BINFILE.extension	Read the content of the binary file and bound it to the bind variable mapped to the BLOB column in the table. Consume memory if file is large. May degrade performance.	<row>'${ds-nextval}', '${ds-textfile}=sampleBigText1.txt', '**${ds-binaryfile}=design-doc\DataShaper User Guide.pdf**'</row>

Performance Tips

Where the database tables are using Sequence, which is a record id generation database object, DataShaper runtime will always pre-fetch a block of new ids from the sequence and consume them by assigning one by one to new rows in the target tables. Once the block is exhausted, the runtime will continue to fetch the next block. Batch insert using the pre-fetched ids is the most optimal way of populating large volume of data. Sequence increment of 100 is the optimal value for the runtime. Prior to the run, it is recommended that the sequence increment should be set to 100 by issuing an alter statement; for example, alter sequence actproj increment by 100, where 'actproj' is the sequence name, 100 is the new increment. After the run, the increment should be reset back to its original value.

Wherever applicable, indexes should not be added prior to the bulk data population.

Wherever applicable, the DataShape Runtime should be installed and run from a server closest to the db server on the network. Network latency can add significant timing cycles to the interactions between the client and server.

If the configuration is using query to retrieve existing data, ensure that indexes are created prior to the run.

Lab-Test Performance Data

Based on the lab tests conducted under the environment: `Intel® Core™ 2 Duo CPU P3750 @2.00G Hz, 4.00GB RAM, 198GB Disk Space` and with the five db servers running on the same box as DataShaper Runtime and with no indexes in the schema during population.

The table below shows the timing statistics of run (in milliseconds) based on sample `ExampleMany2ManyRelationship.xml` configuration file against each of the supported database servers in creating a total of `10110000` (ten million and eleven hundred thousand) records:

Database	With Sequence at increment of 100	With Sequence at increment of 1	With Identity Column or Auto Increment
Oracle	493,976 ms (< 9 mins)	3,252,271 ms (< 57 mins)	Simulated with a trigger and using ExecuteUpdate()/getReturnedIndexes(): 4,458,707 ms (< 1 hr 15 mins)
IBM DB2	899,665 ms (< 15 mins)	9,703,388 ms (< 2 hrs 42 mins)	3,850,949 ms (< 1 hr 5 mins)

PostgreSQL	1,882,830 ms (< 32 mins)	4,450,313 ms (< 1 hr 15 mins)	Not applicable. PostgreSQL uses implicit Sequence.
MySQL	Not supported by database vendor	Not supported by database vendor	6,140,684 ms (< 1 hr 43 mins)
MSSQLServer	Not supported by database vendor	Not supported by database vendor	2,648,674 ms (< 45 mins)

Appendixes

Appendix – Mapping between JDBC/Java and SQL Data Types

SQL	JDBC/Java	Remarks
VARCHAR	java.lang.String	
CHAR	java.lang.String	
LONGVARCHAR	java.lang.String	
BIT	boolean	
NUMERIC	java.math.BigDecimal	
TINYINT	byte	
SMALLINT	short	
INTEGER	int	
BIGINT	long	
REAL	float	
FLOAT	float	
DOUBLE	double	
VARBINARY	byte[]	
BINARY	byte[]	
DATE	java.sql.Date	
TIME	java.sql.Time	
TIMESTAMP	java.sql.Timestamp	
CLOB	java.sql.Clob	
BLOB	java.sql.Blob	
ARRAY	java.sql.Array	Not supported
REF	java.sql.Ref	Not supported
STRUCT	java.sql.Struct	Not supported

Appendix - Examples of Runtime Engine Run

(note: Windows uses ; and backward slash \)

```
java -classpath datashaper.jar:drivers/ojdbc5.jar com.psrtoolkit.datashaper.DataShaper
-c samples/oracle/ORCLExampleSimpleCreateCustomers.xml -u actproj -p actproj

java -classpath datashaper.jar:drivers/postgresql-8.4-701.jdbc4.jar
com.psrtoolkit.datashaper.DataShaper
-c samples/postgresql/POSTGRESQLExampleSimpleCreateCustomers.xml -u actproj -p actproj

java -classpath datashaper.jar:drivers/mysql-connector-java-5.1.13-bin.jar
com.psrtoolkit.datashaper.DataShaper -c samples/mysql/MYSQLExampleSimpleCreateCustomers.xml -u
actproj -p actproj

java -classpath datashaper.jar:drivers/db2jcc4.jar com.psrtoolkit.datashaper.DataShaper
-c samples/db2/DB2ExampleSimpleCreateCustomers.xml -u actproj -p actproj

java -classpath datashaper.jar:drivers/sqljdbc4.jar com.psrtoolkit.datashaper.DataShaper
-c samples/mysqlserver/MSSQLServerExampleSimpleCreateCustomers.xml -u actproj -p actproj
```