High Utility Data Generation Using DataXplod

Vijayanand Banahatti  
TCS Innovation Labs - TRDDC  
54B Hadapsar Industrial Estate  
Pune 411013 India  
91-20-66086333  
vijayanand.banahatti@tcs.com

Srinivasan Iyengar  
TCS Innovation Labs - TRDDC  
54B Hadapsar Industrial Estate  
Pune 411013 India  
91-20-66086333  
venkatachary.srinivasaniyengar@tcs.com

Sachin Lodha  
TCS Innovation Labs - TRDDC  
54B Hadapsar Industrial Estate  
Pune 411013 India  
91-20-66086333  
sachin.lodha@tcs.com

ABSTRACT
Despite the persistent need for an appropriate volume of high utility data across all TCS projects, there was no economical, efficient, flexible, person-independent and reusable methodology for data generation and supporting tool available to satisfy this need.

In this paper, we propose a novel, flexible methodology called Xplod for generation of any volume of high utility data that can exploit the production data, if available, and has low expectations when it comes to inputting metadata/data- semantics related information. We also describe our data generation tool DataXplod that implements this methodology in economical, efficient, and reusable manner, eliminating the need to have a highly skilled human resource, allowing high utility data generation on-demand even at off-shore, thus giving TCS project teams/clients substantial cost, quality and productivity improvements.

1. INTRODUCTION
Need for an appropriate volume of high utility data is ubiquitous in TCS [12, 14]. This need could be explicit in certain types of projects that require us to do, say, performance testing, load testing, or capacity planning, or it could be implicit in some other types of projects where such data could help validate or fine-tune the underlying application, process, or system, for example, query optimization, data migration, and knowledge discovery. Even in case of functional testing of applications, which is a routine activity across TCS, having more data would certainly help to improve the effectiveness of testing [2, 7, 9].

1.1 Current Practice
TCS project teams follow one of the following strategies to fulfill this need:

1. Use production data: Since the production data already has the right syntax and inborn semantics that capture business rules, it might be the best choice as far as utility of the data is concerned.

   A major problem with this approach is that there could be privacy issues that need to be addressed [19]. This may necessitate use of data masking tools (like our in-house tool Masketeer [6]) to appropriately mask the given production data so that it can be shared at off-shore. Another option is to execute the project entirely at on-site to avoid non-compliance of privacy regulations, which might not be an ideal situation for both the client and TCS.

   Moreover, this approach does not work if the required data volume is more than the volume of available production data, for example, as in a capacity planning or a load testing project. Similarly, in case of fresh development when there is no existing production environment to provide the data, this approach would not work.

2. Use synthetic data: This approach eliminates the risk of privacy breach as the data generated is fictitious [4, 15, 18]. Its biggest drawback is that it demands near perfect knowledge about data semantics which is seldom readily available; otherwise it generates random data that does not lead to meaningful results. It is possible to gather more insights into data and its semantics either by analyzing the related application code, or by interviewing client/data- owner, or by running some data profiling tools [1, 10]. Note that each of them can be a time-consuming and/or costly activity; and many a times without full assurance about the completeness of the task.

   a) A number of off-the-shelf products are available in the market, for example, [3, 13], that can generate synthetic data. Better ones cost around USD 30,000 to USD 50,000 per licensed copy – not a small amount. An additional consultancy/support fee charge is also applicable while the product is in use. Moreover, even the best of these tools may not be able to handle all requirements of high utility data generation (we shall study these requirements in detail in Section 2). For many of them, the task of inputting metadata and/or data semantics knowledge is a completely manual process that takes time and introduces errors at the input stage itself.

   b) Writing custom scripts to generate data is a popular approach in TCS. In many projects that we have come across, such a data generation mini-project is explicitly agreed upon with the client. Note that this kind of scripting activity often requires a highly skilled human resource. And yet it does not ensure that the data generated would be as per requirements. If the requirements themselves keep evolving such as in fresh development projects, then one need to constantly evolve these scripts as well. Moreover, this work is
typically non-reusable, even within the same account team. From TCS view-point, this is not a happy situation since productivity, reusability and quality all are low whereas dependency on a skilled human resource is high.

Thus, despite the persistent need for appropriate volume of high utility data being critical across TCS projects, there is no economical, efficient, flexible, human-independent, reusable methodology and supporting tool available to satisfy this need.

In this paper, we propose a novel, flexible methodology called Xplod for generation of any volume of high utility data that can exploit the production data, if available, and has low expectations when it comes to inputting metadata and data semantics related information. We also describe our data generation tool DataXplod that implements this methodology in economical, efficient, and reusable manner, eliminating the need to have a highly skilled human resource, allowing high utility data generation on-demand even at off-shore, thus giving TCS project teams/clients substantial cost, quality and productivity improvements.

1.2 Outline of the Paper
In Section 2 we have discussed the idea of data utility. The Xplod methodology is described in Section 3. It uses seed data and repeatedly applies various transformations on the same. These transformations are described in Section 4 and Section 5. Description of DataXplod tool that provides these transformations and enables the Xplod methodology appears in Section 6. We discuss its small experimental study Section 7. DataXplod has received a very enthusiastic response from TCS project teams. We have provided a quick summary of its success stories in Section 8. We conclude with discussions in Section 9.

2. CAPTURING DATA UTILITY
Utility of data is extremely contextual and depends on the underlying task at hand. For example, it could just be certain number of records adhering to some specific format in the context of load testing or it may require data satisfying all kinds of statistics when the data is to be used for validating/testing knowledge discovery applications. Between these two extremes, there may be a scenario like functional testing where the utility of data could be measured in terms of its testing effectiveness/code coverage.

We have tried to broadly categorize the salient characteristics of data that would have significant effect in determining its utility for essentially any task at hand. Please note that these characteristics would also help in capturing the syntax and inherent semantics of the production data, the highest utility data possible in many scenarios. To the best of our knowledge, this is first of its kind of efforts to explicitly state such characteristics of data that directly impact its utility.

2.1 Cell Data Characteristics
These are essentially micro level characteristics of the data.

2.1.1 Syntactic Integrity of the Data
Syntactic considerations can be present, such as data element has to start with a certain combination of numbers or needs to be of a fixed length. For example, in India, bank account numbers are generally 15 digits in length where each bank has first 3 digits reserved for specifying its own code.

2.1.2 Familiar Feel of the Data
Look-n-feel can play an important role in determining data utility. For example, in India, names like Vijay, Srinivasan and Sachin are more familiar than Didier and Adebayor which can be common in Africa. One may want to use the Indian names in India and the African names in Africa while training respective HR teams on the same software.

2.2 Column Data Characteristics
These are essentially macro level characteristics of the data.

2.2.1 Statistical Integrity of the Data
There could be statistical considerations such as maintaining average for the account balance field in a bank database, or maintaining correlations between age and disease columns in a medical database. Similarly, maintaining frequency distributions of data values in different columns can play an important role in correctly capturing the data utility. For example, some mobile handsets may be very popular in some geography while not so popular in some other geography. Also, the sales of products vary throughout the year.

2.2.2 Checks on the Dataset
Please recall that checks are defined to represent some conditionality of certain columns with some constant values. These are generally enforced in the production environments through external applications. Such checks can be applied to see if a numeric field is between two constant values, or a date field is greater than some specific date or a string is never equal to NULL value. For example, account balance can never be less than zero in a bank database.

2.3 Inter Column Data Characteristics
The inter column data characteristics define the dependencies across columns that need to be maintained.

2.3.1 Referential Integrity
For example, account number, a primary key in the account table, is a foreign key in the transaction table. Referential integrity is one of the most important considerations. If given data cannot meet referential integrity considerations, it is going to be of no use in most of the practical scenarios.

2.3.2 Associations between Columns
Associations between columns are very difficult to capture. For example, if the age of a person is less than 25 years, then there is a high possibility that he cannot have Parkinson’s disease. Also for any two records A and B in the HR database, if A.emp_id>B.emp_id, then A.joining_date >B.joining_date could be the case. For many complex knowledge discovery applications, data rich with such associations is a must to correctly validate/test/fine-tune their working.

2.3.3 Derived Columns
Often there are some columns in the database that are derived from other columns. For example, the gross salary column can be derived from the summation of basic salary and bonus money. Similarly, international calling number can be derived by concatenating country code and individual’s telephone number. Like referential integrity, if the given data cannot meet derived columns considerations; it is not going to be of much use in most of the practical scenarios.
3. XPLOD METHODOLOGY

In Section 1, we have already discussed strengths and limitations of the currently prevalent data generation practices in TCS. Here we present a novel and flexible methodology called Xplod that can harness strengths of those methods while overcoming their many limitations, and helps generate data consisting of all characteristics discussed in Section 2, thus providing the user data with high utility for the given task.

Before we proceed further, please note that if the data requirement can be met by sampling and suitably masking production data, we already have a successful data masking tool called Masketeer [6] to cater to that scenario. In what follows, we essentially address the complementary scenario wherein the data requirement is bigger than what can be availed from the production environment.

The main idea here is to let data itself be used for generating more data. Sample data from the production environment or hand created data, which we call seed data can be used for this purpose. An approach for generating voluminous data for testing can be obtained by copying this seed data one by one till the required quantity of data is generated. But this is not possible as relational databases will not accept collated copies of the original data. Also mere copy of the seed data will not always be sufficient to satisfy all the requirements of the test data. For example, in case of functional testing, copies of same data will not improve the code coverage. Hence there is a need to transform data every time a copy is created. Figure 1 shows how this approach works.

![Figure 1: Xplod Methodology](image)

Even without getting into further details of how to do the data transformation, it is easy to see several advantages of this methodology:

1. Production data, if available, can be considered to have the highest utility. By making multiple copies of it, we preserve much of the same. For example, average of a column does not change, correlations between columns scale proportionately. Even other data characteristics such as associations, if present in the seed data, can just carry over to the generated data.

2. In the case when production data is not available, there is a provision to feed in relatively small size seed data which could very well be hand-created with great care to maintain right characteristics (described in Section 2), and therefore, utility.

3. The overall approach is truly scalable as data of the size of seed data is generated in each run, and in a sequential manner. Further this approach is amenable to parallelization which can result in more speed up by generating disjoint data on different machines.

4. It is not required to explicitly add/describe complex data characteristics such as associations between columns and statistical integrity constraints. Thus there is no real need of complete data profiling. Moreover, inputting metadata/data characteristics to the data generator should be less demanding and less effort intensive.

5. By applying appropriate transformations to the seed data, one can ensure that the generated data is privacy compliant, and thus shareable even at off-shore [19].

This brings us to the important process of data transformation which has to be done in a careful manner such that all the business rules, syntactic / statistical / relational integrity constraints, and other data semantics are maintained and carried over from the seed data. In the next two sections we describe different base transformers and synthesizers that can help us achieve this objective.

4. BASE TRANSFORMERS

The base transformers are designed to preserve the cell and column level data characteristics. There are two main classes of base transformers, namely, Randomization and Noise Addition.

Please note that the Xplod methodology proceeds in multiple, say \( R \), rounds, and in each round each data item of certain pre-decided columns of input seed data is being transformed to meet the utility requirements. In what follows, \( X_i \) represents the \( i \)th data item of the column to be transformed in the seed data, where \( X_i^j \) represents the corresponding generated data item in the \( j \)th round. Once we fix the transformer to be applied to the given column, then \( X_i^j \) is simply a result of applying that transformer to \( X_i \), where position value \( i \), the round number \( r \), the total number of rounds \( R \), size of the column, and some other auxiliary inputs may also play a role.

4.1 Randomization

The methodology of randomization and random number generation is well-researched [11, 5] and it has been frequently used as a key method in synthetic data generation [3, 4, 15, 18]. We have designed the following sub-classes of randomization based transformers to help generate high utility data as characterized in Section 2.

4.1.1 List-based Randomization

In this sub-class, values from a list are used to generate data. Thus it can help transform both alpha-numeric and date-time data types. This sub-class can be represented in the form of a mathematical function in the following manner:

\[ X_i^j = L[m*(r-1) + i] \]

Here \( m \) is the number of records in the input column, and...
\[ L = \{l_p, l_{p-1}, \ldots, l_1\} \] is the list containing \( p \geq R \cdot m \) values. \( L[q] \) returns \( l_q \) for \( 1 \leq q \leq p \).

Table 1 demonstrates the use of this transformer. Here \( m=3 \), \( R=3 \), and \( L = \{\text{Kolkata, Lucknow, Chennai, Kolhapur, Shimla, Chandigarh, Bengaluru, Jaipur, Surat}\} \).

<table>
<thead>
<tr>
<th>CITY_NAME</th>
<th>ACC_NO</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pune</td>
<td>310143</td>
<td>1</td>
</tr>
<tr>
<td>Mumbai</td>
<td>310157</td>
<td>1</td>
</tr>
<tr>
<td>Delhi</td>
<td>310182</td>
<td>1</td>
</tr>
<tr>
<td>Kolhapur</td>
<td>310193</td>
<td>2</td>
</tr>
<tr>
<td>Shimla</td>
<td>310138</td>
<td>2</td>
</tr>
<tr>
<td>Chandigarh</td>
<td>310138</td>
<td>2</td>
</tr>
<tr>
<td>Bengaluru</td>
<td>310139</td>
<td>2</td>
</tr>
<tr>
<td>Jaipur</td>
<td>310191</td>
<td>3</td>
</tr>
<tr>
<td>Surat</td>
<td>310139</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 1: Example for List-based Randomization

4.1.2 Range-based Randomization

In this sub-class, values from a range are used to generate data. Thus it can help transform numeric and date-time data types. This sub-class can be represented in the form of a mathematical function in the following manner:

\[
X'_i = \text{lowval} + ((R \cdot (r-1) + i) \cdot \text{incrval}) \% (\text{highval} - \text{lowval})
\]

Here \( \text{lowval} \) is the lower limit of the range, \( \text{highval} \) is the upper limit of the range, and \( \text{incrval} \) is the increment value. If increments result in values beyond \( \text{highval} \), we reset it back using modulus (\( \% \)) operator.

Table 2 demonstrates the use of this transformer. Here \( m=3 \), \( R=3 \), \( \text{lowval} = 5000 \), \( \text{highval} = 5020 \), with \( \text{incrval} = 2 \).

<table>
<thead>
<tr>
<th>BRANCH_ID</th>
<th>ACC_NO</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>310143</td>
<td>1</td>
</tr>
<tr>
<td>1002</td>
<td>310157</td>
<td>1</td>
</tr>
<tr>
<td>1003</td>
<td>310182</td>
<td>1</td>
</tr>
<tr>
<td>1004</td>
<td>310193</td>
<td>2</td>
</tr>
<tr>
<td>1005</td>
<td>310138</td>
<td>2</td>
</tr>
<tr>
<td>1006</td>
<td>310139</td>
<td>2</td>
</tr>
<tr>
<td>1007</td>
<td>310191</td>
<td>3</td>
</tr>
<tr>
<td>1008</td>
<td>310139</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 2: Example for Range-based Randomization

4.1.3 Regular-Expression-based Randomization

In this sub-class, values from a range specified by a regular expression [16] are used to generate data. Thus it can help transform alpha-numeric data type. For columns having uniqueness requirement, the first value is picked randomly from the range and the subsequent values are generated similar to an odometer in a sequential manner. Otherwise the values are generated in a purely random manner.

Table 3 demonstrates the use of this transformer. Here \( m=4 \), \( R=3 \), regular expression used is \( 301[0-9]/3 \) which specifies range between 301000 and 301999.

<table>
<thead>
<tr>
<th>Original seed data</th>
<th>Final exploded data where ACC_NO is unique</th>
<th>Final exploded data where ACC_NO is non-unique</th>
<th>Generated in round</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACC_NO</td>
<td>ACC_NO</td>
<td>ACC_NO</td>
<td></td>
</tr>
<tr>
<td>301443</td>
<td>301383</td>
<td>301624</td>
<td>1</td>
</tr>
<tr>
<td>301567</td>
<td>301384</td>
<td>301549</td>
<td>1</td>
</tr>
<tr>
<td>301882</td>
<td>301385</td>
<td>301345</td>
<td>1</td>
</tr>
<tr>
<td>301913</td>
<td>301386</td>
<td>301442</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>301387</td>
<td>301711</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>301388</td>
<td>301267</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>301389</td>
<td>301104</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>301390</td>
<td>301249</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>301391</td>
<td>301995</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>301392</td>
<td>301965</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>301393</td>
<td>301298</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>301394</td>
<td>301087</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 3: Example for Regular-Expression-based Randomization

4.2 Noise Addition

Like randomization the methodology of noise addition has also been well explored and it has already been found very useful in maintaining the statistical properties of data [17, 8]. But unlike in randomization here exists more direct relationship between the input data and the generated data. If \( X \) were the input data, then the generated data \( Y \) would be \( Y = X + e \), where \( e \) is the noise which itself could be a function of \( X \) itself, or some standard function like a Gaussian, etc. Often the type of noise depends upon the characteristics that we want to maintain in the generated data. Note that we can apply this class of transformation only to the numeric or date-time data types.

We have designed the following sub-classes of noise addition based transformers to help generate high utility data as characterized in Section 2.

4.2.1 Gaussian-based Noise Addition

In this sub-class, Gaussian noise \( G \) is added to the input data where \( G \) is given by the formula

\[
G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}.
\]

The shape of the Gaussian is decided by its variance value \( \sigma \). By controlling this value, one can make more or less amount of noise addition. This sub-class can be represented in the form of a mathematical function in the following manner:

\[
x'_i = x_i + G(x) \cdot \text{rand}().
\]

Here variance value \( \sigma \) is already fixed, that is why \( G(x) \) and \( \text{rand}() \) function is used to spit out a random value \( x \) to be input to the Gaussian distribution function \( G(x) \).
Table 4 demonstrates the use of this transformer. Here \( m=3, \ R=3 \). Please note more variation in the third column compared to that in the second column with respect to the original seed data.

<table>
<thead>
<tr>
<th>Original seed data</th>
<th>Final exploded data (variance 0.6)</th>
<th>Final exploded data (variance 0.9)</th>
<th>Generated in round</th>
</tr>
</thead>
<tbody>
<tr>
<td>BALANCE</td>
<td>BALANCE</td>
<td>BALANCE</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>99</td>
<td>188</td>
<td>1</td>
</tr>
<tr>
<td>3450</td>
<td>3454</td>
<td>3530</td>
<td></td>
</tr>
<tr>
<td>45000</td>
<td>44991</td>
<td>44873</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>71</td>
<td>3584</td>
<td>2</td>
</tr>
<tr>
<td>3448</td>
<td>45005</td>
<td>44843</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>88</td>
<td>3426</td>
<td>3</td>
</tr>
<tr>
<td>3457</td>
<td>44996</td>
<td>45147</td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Example for Gaussian-based Noise Addition

4.2.2 Range-based Noise Addition

In this sub-class, noise from a range is added to the input. This sub-class can be represented in the form of a mathematical function in the following manner:

\[
X'_i = \alpha X_i + R,
\]

\[
R = \text{rand}(\text{lowval}, \text{highval}).
\]

Here \( \text{rand}(\cdot, \cdot) \) is a function that will return a random value between the two parameters provided as input, namely, \( \text{lowval} \) and \( \text{highval} \) that define the range.

Table 5 demonstrates the use of this transformer. Here \( m=3, \ R=3 \). Please note more variation in the third column compared to that in the second column with respect to the original seed data.

<table>
<thead>
<tr>
<th>Original seed data</th>
<th>Final exploded data with range +100 to +200</th>
<th>Final exploded data with range -100 to +500</th>
<th>Generated in round</th>
</tr>
</thead>
<tbody>
<tr>
<td>BALANCE</td>
<td>BALANCE</td>
<td>BALANCE</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>251</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>3450</td>
<td>3623</td>
<td>3779</td>
<td></td>
</tr>
<tr>
<td>45000</td>
<td>45121</td>
<td>44973</td>
<td></td>
</tr>
<tr>
<td>236</td>
<td>442</td>
<td>3818</td>
<td>2</td>
</tr>
<tr>
<td>3491</td>
<td>45155</td>
<td>45297</td>
<td></td>
</tr>
<tr>
<td>284</td>
<td>3567</td>
<td>237</td>
<td>3</td>
</tr>
<tr>
<td>3593</td>
<td>45111</td>
<td>45112</td>
<td></td>
</tr>
</tbody>
</table>

Table 5: Example for Range-based Noise Addition

4.2.3 Percentage-based Noise Addition

In this sub-class, noise is added from a percentage range specified. This sub-class can be represented in the form of a mathematical function in the following manner:

\[
X'_i = \alpha X_i + P,
\]

\[
P = \text{Rand}((\text{lowval}^* \cdot X_i), (\text{highval}^* \cdot X_i)).
\]

Here \( P \) stands for the percent based noise added, and \( \text{rand}(\cdot, \cdot) \) is a function that will return a random value between the two parameters provided as input. Note \( \text{lowval} \) and \( \text{highval} \) are the inputs that define the percentage range. This sub-class can only be used for numeric data type.

Table 6 demonstrates the use of this transformer. Here \( m=3, \ R=3 \). Please note more variation in the third column compared to that in the second column with respect to the original seed data.

<table>
<thead>
<tr>
<th>Original seed data</th>
<th>Final exploded data with range +10% to +80%</th>
<th>Final exploded data with range -50% to +50%</th>
<th>Generated in round</th>
</tr>
</thead>
<tbody>
<tr>
<td>BALANCE</td>
<td>BALANCE</td>
<td>BALANCE</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>169</td>
<td>88</td>
<td>1</td>
</tr>
<tr>
<td>3450</td>
<td>5217</td>
<td>3589</td>
<td></td>
</tr>
<tr>
<td>45000</td>
<td>69841</td>
<td>43217</td>
<td></td>
</tr>
<tr>
<td>112</td>
<td>112</td>
<td>143</td>
<td></td>
</tr>
<tr>
<td>5913</td>
<td>5913</td>
<td>3555</td>
<td></td>
</tr>
<tr>
<td>77920</td>
<td>77920</td>
<td>47602</td>
<td></td>
</tr>
</tbody>
</table>

Table 6: Example for Percent-based Noise Addition

4.2.4 Shift-based Noise Addition

In this sub-class, the data is shifted by a constant which itself is incremented in every run. This sub-class can be represented in the form of a mathematical function in the following manner:

\[
X'_i = \alpha X_i + r \cdot \text{shiftval}.
\]

Here \( \text{shiftval} \) is the constant value that gets multiplied by the round number \( r \), and the resultant noise value is added to the input data.

Table 7 demonstrates the use of this transformer. Here \( m=3, \ R=3 \), and \( \text{shiftval}=30 \).

<table>
<thead>
<tr>
<th>Original seed data</th>
<th>Final exploded data</th>
<th>Generated in round</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACC_OPEN_DATE</td>
<td>ACC_OPEN_DATE</td>
<td></td>
</tr>
<tr>
<td>10/12/1998</td>
<td>9/01/1999</td>
<td>1</td>
</tr>
<tr>
<td>12/12/1999</td>
<td>11/01/2000</td>
<td></td>
</tr>
<tr>
<td>03/05/1998</td>
<td>01/06/1998</td>
<td></td>
</tr>
<tr>
<td>8/02/1999</td>
<td>10/02/2000</td>
<td>2</td>
</tr>
<tr>
<td>02/07/1999</td>
<td>10/03/1999</td>
<td></td>
</tr>
<tr>
<td>11/03/2000</td>
<td>01/08/1998</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 7: Example for Shift-based Noise Addition

4.3 Remarks

Empirically we find that Randomization and Noise Addition class of base transformers are sufficient to preserve most of the cell-level and column-level data characteristics. Randomization class of base transformers can be applied to string, number, alphanumeric, and date-time data types. This class is also best suited for transforming primary and unique keys. Noise Addition class of base transformers can be applied on numeric and date-time data types only and it is ideal for preserving temporal details.
Yet there is a need of a framework for adding more base transformers to this list depending on the requirement of the data generation process. This framework must have an extensible interface to plug-in customized transformers. Customized transformers as the name suggests can be developed for any data type.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Transformations</th>
<th>Randomization</th>
<th>Noise Addition</th>
<th>Customized</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numeric</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>String</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Alpha</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Numeric (Non-integer)</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Alpha (Non-integer)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Date</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Time</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Binary</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

**Table 8: Transformers for different data types**

Applicability of all the base transformers to various data types is summarized in Table 8 above.

### 5. SYNTHESIZERS

Base transformers are used to meet the data characteristics requirements at the cell level and/or column level. But they by themselves cannot maintain inter-column data characteristics like referential integrity, association between columns, etc. Therefore there is a need of more sophisticated and powerful functionalities that can work with the base transformers and achieve the same. We call them synthesizers as they synthesize data from the outcome of base transformations.

The synthesizers are of two types – Relational Integrity synthesizers and Business Logic synthesizers. We make use of the following schema shown in Figure 2 to point out their role in the Xplosion process. Their working, that is, the final exploded data is shown in Figure 3 on the next page.

![Figure 2: Original Seed Data](image)

**Figure 2: Original Seed Data**

There are two tables in this schema - namely Customer table and Account table. <CUSTOM_ID> is the primary key for the Customer table and <ACC_NO> is the primary key in the Account table. <CUSTOM_ID> is also a foreign key in the Account table which refers to the <CUSTOM_ID> from the Customer table.

### 5.1 Relational Integrity Synthesizers

Relational integrity synthesizers help in preserving the relational properties of the data.

#### 5.1.1 Primary Key Synthesizer

In relational databases, primary key is used to uniquely identify individual records. While generating data, it is imperative to generate unique values for a primary key column(s). This particular synthesizer helps meet this requirement irrespective of whichever underlying base transformer we use for the data generation of such column(s).

For example, in Figure 2, this synthesizer would come in play while generating data for <CUSTOM_ID> and <ACC_NO> columns which are the primary keys for the Customer table and the Account table respectively.

#### 5.1.2 Unique Key Synthesizer

In relational databases, unique key is one where no two records have the same value for the column where it is enforced. This is similar to the primary key; the only difference being that unique keys can have NULL values. Hence while generating data, it is imperative to generate unique values for a unique key column(s).

This particular synthesizer helps meet this requirement irrespective of whichever underlying base transformer we use for the data generation of such column(s).

For example, in Figure 2, this synthesizer would come in play while generating data for <CONTACT_NO> column which is a unique key in the Customer table.

#### 5.1.3 Foreign Key Synthesizer

In relational databases, foreign key identifies a column or a set of columns in one table (referencing table) that refers to a column or set of columns in another table (referenced table). The columns in the referencing table must be the primary key in the referenced table. This makes it possible to maintain cascading relationships between tables. This particular synthesizer helps meet this requirement by enforcing data generation of the primary key columns in the referenced table first, and then propagating those data values to the corresponding columns in the referencing table.

For example, in Figure 2, this synthesizer would come in play while generating data for <CUSTOM_ID> in the Account table since it is a foreign key which refers to <CUSTOM_ID> in the Customer table.

### 5.2 Business Logic Synthesizers

Business logic synthesizers help in preserving the business logic semantics of the data.

#### 5.2.1 Check Synthesizer

This synthesizer helps meet Checks on the Dataset requirement discussed in Subsection 2.2.2.

For example, in Figure 2, it would come in play while generating data for <OPEN_BALANCE> column in the Account table to ensure that the generated data is never less than zero.

#### 5.2.2 Derivation Synthesizers

This synthesizer helps meet Associations between Columns, Derived Columns requirements discussed in Subsections 2.3.2 and 2.3.3.

For example, in Figure 2, <ACC_TYPE> from the Account table is dependent on <OPEN_BALANCE> from the same table. <ACC_TYPE> can have three values which are derived from the value in the <OPEN_BALANCE> column in the following way: If OPEN_BALANCE ≤ 1000, then the ACC_TYPE is called “SILVER”. If 1000 < OPEN_BALANCE ≤ 10000, then the ACC_TYPE is called “GOLD”. If OPEN_BALANCE > 10000, then the ACC_TYPE is called “PLATINUM”. Derivation synthesizer would come in play while generating data for <ACC_TYPE> column as it would derive this column by
applying the above logic on the already generated data for <OPEN_BALANCE> column.

The final exploded data after application of the above mentioned synthesizers along with some base transformers is shown in Figure 3 below.

---

Now the entire Xplod methodology of data generation boils down to an appropriate data processing using base transformers and synthesizers which is shown in Figure 4. Initially the original seed data passes through the base transformers wherein randomization and/or noise addition and/or custom classes of base transformers are applied to it. Then such transformed data passes through synthesizers wherein relational integrity synthesis and, if required, business logic synthesis steps take place, and finally the transformed data is output. This entire processing on the original seed data repeats for multiple rounds in order to provide the required size output.

Note that relational integrity synthesizers are drawn in a vertical fashion to indicate that the data has to compulsorily flow through them as they are mandatory constraints. The base transformers and the business logic synthesizers are optional and the user can use them in the way it is required.

---

6. DATAXPLOD

To implement the Xplod methodology in efficient and reusable manner, we have developed a data generation tool called DataXplod. It comes packed with many features, including all base transformers and synthesizers described in the previous section, to help execute the above mentioned data processing steps in flawless and frugal manner.

In this section, we provide a brief overview of some important features of DataXplod. We also describe a three phase methodology that TCS project teams / clients can follow for high utility data generation using DataXplod in their respective projects. Please refer to Figure 5 for the picture of DataXplod start-up screen.

---

6.1 Key Features

6.1.1 Data Sampling

The tool has a sampling and sub-setting capability wherein a small sample can be taken from the source data.

6.1.2 Constraint Editor

Constraints such as unique, primary, foreign key can be read from the schema or can be input through the UI of the tool. Also, some DataXplod defined constraints such as reference, derivation and check can also be specified.

6.1.3 Data Transformations

Data can be transformed by specifying datasets in the form of range of values, regular expressions, lists of values, input from an external text file, or by adding noise in the form of Gaussian noise, constant noise, noise in terms of range and percent. Also transformation of data can be brought about by using derivation functions, and referencing from another column.

To take care of referential integrity, the data transformation is initially applied on the primary key values and then the effect is cascaded to the child tables. Also as we try to transform data at each run taking the seed data as input, we maintain the cardinality that was originally present in the seed data.

6.1.4 Visual Validations

Comparative viewer is available to see the source data vis-à-vis the transformed data.

6.1.5 Support Files

Ready to load data is generated in the form of flat files. The tool also generates necessary support files to load data into the target DBMS.
6.1.6 Pluggable Interface
An interface has been provided to add and extend data transformation logic.

6.1.7 Multi Database Platform Support
DataXplod supports all popular databases like CacheDB, DB2, Informix, MS SQL Server, MySQL, Oracle, Sybase and also inputs from flat files. It can be readily extended to support any database for which JDBC driver is available.

6.1.8 Schema Refresh
Changes, if any, in the source schema can be read by the tool using its schema refresh functionality. This provides a great amount of flexibility in the process of data generation, especially in the development environment.

6.2 DataXplod Three Phase Methodology
We have designed a three phase methodology for data generation using DataXplod. Please refer to Figure 6 below.

![Figure 6: DataXplod Three Phase Methodology](image)

6.2.1 Phase 1 - Analysis and Planning
Analysis and planning phase includes requirement analysis, generating initial seed data often called as base volume and designing the Xplosion strategy.

6.2.1.1 Requirement Analysis
In this step, we clearly write the final result that is expected from the tool. Here we need to mention how much data needs to be generated, what kind of relationship exists between the different tables, and the patterns that have to be present which are required to effectively perform the necessary tasks like performance testing, functional testing, query optimization, etc. This necessitates analysis of the data syntax, semantics, business logic and the relationships among different columns.

6.2.1.2 Designing the Xplosion Strategy
Depending on the expected output, base volume or initial seed data is obtained. Production data can serve as an ideal seed data. When it is not available, seed data has to be generated from scratch by taking into consideration all required data characteristics. The various ways to use the base transforms and synthesizers are considered and most optimal way to use the tool is decided so that time taken for the data generation is reduced.

6.2.2 Phase 2 – Data Xplosion
In this phase, the data transformation configuration is carried out and its validation is done though mini explosion. After this step, actual data xplod is carried out.

6.2.2.1 Configuration Set-up
Based on the Xplosion strategy, DataXplod set-up is done by configuring transformation strategies for various columns. Since the tool takes care of referential integrity constraints automatically, we need to configure only base transformers and business logic synthesizers.

6.2.2.2 Validation through Mini-Xplosion
To avoid rework, a mini xplod is carried out and the generated data is reviewed against all requirements.

6.2.2.3 Data Xplosion
This is the all important step where data gets generated. Here the time taken for data generation depends on the size of output data and also on the complexity of the database schema.

6.2.3 Phase 3 – Deployment
In this phase, the generated data is loaded in to the target database.

6.2.3.1 Loading Data into the Database
DataXplod generates output in the form of flat files. It also generates the necessary support files depending on the target database. By running those support files, data can be conveniently loaded into the target database.

6.2.3.2 Final Validation
A final validation is performed on the generated data against all the requirements using pre-designed test cases.

7. EXPERIMENTAL STUDY
A bank has envisaged that it will implement a core banking solution for all its branches. So the load on its systems will increase by a factor 3 in the coming two years. It wants to check if its existing applications can handle the increased load.

![Figure 7: Initial Seed Data](image)

7.1 Schema Information
Figure 7 shows schema of input seed data of a bank database. Customer and Branch tables are parents to the Account table, which in turn is the parent to the Transaction table. <CUST_ID> is the primary key in the Customer table and is a foreign key in the Account table. Similarly, <BRANCH_ID> is the primary key in the Branch table and is a foreign key in the Account table. <ACC_NO> is the primary key in the Account table and is a foreign key in the Transaction table.

7.2 Requirements
The generated data is expected to meet the following requirements:

1. Customer’s personal data should look realistic.
2. The proportion of different account types have to be similar to that in the seed data.
3. <OPEN_BALANCE> should be always greater than zero and the average of this field over the whole column has to be maintained.
4. Branch names should be from a master list.
5. Referential integrity across tables has to be maintained.

Copyright 2010
6. Different fields such as <CUSTOMER_ID> and <ACC_NO> should be generated in a format as seen in the seed data.

7.3 Base Transformers
Here is a transformation plan to generate realistic and voluminous data. Please refer to Table 9 to Table 12 for the same.

<table>
<thead>
<tr>
<th>COLUMN NAME</th>
<th>Base Transform</th>
<th>Rule or parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUST_ID</td>
<td>Regular-Expression-based Randomization</td>
<td>C-22[0-9][3]</td>
</tr>
<tr>
<td>CUST_NAME</td>
<td>List-based Randomization</td>
<td>External text file containing American names</td>
</tr>
<tr>
<td>CONTACT_NO</td>
<td>Regular-Expression-based Randomization</td>
<td>9[0-9][9]</td>
</tr>
</tbody>
</table>

Table 9: Base Transform on Customer table

<table>
<thead>
<tr>
<th>COLUMN NAME</th>
<th>Base Transform</th>
<th>Rule or parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRANCH_ID</td>
<td>Range-based Randomization</td>
<td>lower=1; Upper=2; increment=1</td>
</tr>
<tr>
<td>BR_NAME</td>
<td>List-based Randomization</td>
<td>External text file containing branch names</td>
</tr>
<tr>
<td>BR_ADDRESS</td>
<td>List-based Randomization</td>
<td>External text file containing branch addresses</td>
</tr>
</tbody>
</table>

Table 10: Base Transform on Branch table

<table>
<thead>
<tr>
<th>COLUMN NAME</th>
<th>Base Transform</th>
<th>Rule or parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACC_NO</td>
<td>Regular-Expression-based Randomization</td>
<td>301[0-9][3]</td>
</tr>
<tr>
<td>OPEN_DATE</td>
<td>Range-based Noise Addition</td>
<td>lower = -5; higher = +25</td>
</tr>
<tr>
<td>OPEN_BALNACE</td>
<td>Gaussian-based Noise Addition</td>
<td>80%</td>
</tr>
</tbody>
</table>

Table 11: Base Transform on Account table

<table>
<thead>
<tr>
<th>COLUMN NAME</th>
<th>Base Transform</th>
<th>Rule or parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRANSACTION_ID</td>
<td>Range-based Randomization</td>
<td>lower=800001; Upper=800006; increment=1</td>
</tr>
<tr>
<td>AMOUNT</td>
<td>Percentage-based Noise Addition</td>
<td>lower = -10%; higher = +10%</td>
</tr>
<tr>
<td>CONTACT_NO</td>
<td>Shift-based Noise Addition</td>
<td>Shift parameter = 30</td>
</tr>
</tbody>
</table>

Table 12: Base Transform on Transaction table

7.4 Relational Integrity Synthesizers
While generating data, DataXplod automatically enforces the relational integrity constraints on primary keys <CUST_ID>, <BRANCH_ID>, and <ACC_NO> of Customer Table, Branch Table and Account Table respectively, and these changes are cascaded down to the referencing tables.

7.5 Business Logic Synthesizers
7.5.1 Check
This constraint is applied on the <OPEN_BALANCE> field – “OPEN_BALANCE > 0”. This is done as the <OPEN_BALANCE> field is always a positive number.

7.5.2 Derivation
This constraint is applied to populate the <ACC_TYPE> field as this field is dependent on the <OPEN_BALANCE> field. The logic for <ACC_TYPE> is decided in the following way – “OPEN_BALANCE <= 1000” then the <ACC_TYPE> is called “SILVER”. If “1000 < OPEN_BALANCE <= 10000” then the <ACC_TYPE> is called “GOLD”. And if the “OPEN_BALANCE > 10000”, then the <ACC_TYPE> is called “PLATINUM”.

7.6 Data Xplision
Run count for the tool is an input given from the user to give the xplision factor. This factor decides the volume of data to be generated. In this example, the xplision factor is 3 as the bank wants 3 times the data volumes to check its systems. The generated data is shown in Figure 8.

![Figure 8: Final Exploded Data](image)

8. SUCCESS STORIES
A stable version of DataXplod got ready in early 2009. Thanks to our colleagues in Assurance Services Group, Corporate Tools Group, Performance Engineering Lab, and Technology Excellence Group, DataXplod could reach out to many projects/account teams. Here is a summary of its proliferation and impact as on 20-January-2010:

- Customer connects: 45
- As a TCS differentiator: 8 RFPs
- Active deployments/case-studies: 21
- DataXplod odometer reading: 1.1+ billion records generated

Scenarios catered to: anomaly detection testing, capacity planning testing, data migration testing, functional testing, performance testing, query optimization testing, and storage consolidation in test environs.

Average productivity gain: 50% to 80% person hours saved compared to developing custom script-based solution.

The use of the tool has helped the TCS account teams to perform proper testing of different systems and applications for their respective clients because of the quality data that was generated by the tool. Easy-to-use interface and robustness of the tool meant...
that it was an ideal solution for the real-time projects where the requirement of the clients keep on changing and agility of the data generation process is very important. Note that the observed data generation rate has been around 3 to 5 GB/hour on a typical desktop machine. This number can vary depending on the amount of base transformers and synthesizers applied and the complexity of the schema in terms of integrity constraints.

Refer Table 13 for various case studies that we have published till 20-January-2010. For more information on different success stories kindly visit the following link: https://knowmax.ultimatix.net/sites/ctgnew-corpfn/tool/DataXplod/default.aspx.

![Image](Image)

<table>
<thead>
<tr>
<th>Case study name</th>
<th>Type of assignment</th>
<th>Records generated</th>
<th>Productivity gained</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance Testing for a Data Warehouse</td>
<td>Performance Testing</td>
<td>83 Million</td>
<td>932 person hours saved</td>
</tr>
<tr>
<td>Performance Testing of a New Web Applications</td>
<td>Performance Testing</td>
<td>452 Million</td>
<td>472 person hours saved</td>
</tr>
<tr>
<td>Data Migration testing for operational reporting</td>
<td>Data Migration testing</td>
<td>30 Million</td>
<td>252 person hours saved</td>
</tr>
<tr>
<td>Voluminous Data Generation for Fraud detection and Prediction Framework</td>
<td>Knowledge Discovery / Functional testing</td>
<td>0.2 Million</td>
<td>702 person hours saved</td>
</tr>
<tr>
<td>Test Data Generation for Systematic Load Testing</td>
<td>Load Testing</td>
<td>0.42 Million</td>
<td>116 person hours saved</td>
</tr>
<tr>
<td>Realistic Data Generation for Performance Benchmarking</td>
<td>Performance Benchmarking</td>
<td>80 Million</td>
<td>603 person hours saved</td>
</tr>
<tr>
<td>Performance Testing of a Web 2.0 Application</td>
<td>Performance Testing</td>
<td>17.3 Million</td>
<td>201 person hours saved</td>
</tr>
<tr>
<td>Performance Testing for Customer Program Management 1.0 Application</td>
<td>Performance Testing</td>
<td>62.2 Million</td>
<td>825 person hours saved</td>
</tr>
</tbody>
</table>

Table 13: Case Study Highlights

Following are comments from a couple of DataXplod customers:

“Our project required to generate huge volume of data and to include complex functionality and business rules in the generated data. DataXplod with its unique features helped to effectively and efficiently generate the data from scratch. Features like importing all database constraints, possibility of adding user defined constraints on data, including business rules as constraints, data uploading capabilities, support for different data types helped us in a big way. The tool is rugged, portable, easy to use and stable. The efficiency of DataXplod in carrying out things correctly and in less time is worthy for any real-time project.”

– TCS Healthcare Account Team.

“This (DataXplod) tool exactly meets our requirements and it is capable of addressing all the needs of high volume data generation. As far as I know, I really do not see any other tool available in the market for data generation. This (DataXplod) tool is very efficient and can generate high volume of data in a short period of time.”

– TCS Retail Account Team Member.

9. DISCUSSIONS

Data driven testing is a paradigm where both applications and systems are thoroughly tested using data similar to the production environment. It is essential to achieve standard benchmarks for keeping ‘Experience certainty’ promise. Hence the need for high utility, high volume, and privacy compliant data will continue to grow across TCS projects as more and more of our clients want more for less, and demand better quality, more certainty at cheaper rates. This situation gets more interesting with the emerging pay-per-use cloud computing paradigm wherein the clients can now be expected to rent infrastructure for the test purpose on per-use basis rather than investing up-front in a dedicated testing facility. Here easy availability of vast computing resources on rent basis invites possibility of using high volume of data, and its only show stopper could be privacy concerns. Thus high utility privacy compliant data generators will play a major role in assurance/data management services provided in the clouds.

In this paper, we proposed a novel, flexible methodology called Xplod for generation of any volume of high utility data that can exploit the production data, if available, and has low expectations when it comes to inputting metadata/data semantics related information. We also described our data generation tool DataXplod that implements this methodology in economical, efficient, and reusable manner, eliminating the need to have a highly skilled human resource, allowing high utility data generation on-demand even at off-shore, thus giving TCS project teams/clients substantial cost, quality and productivity improvements. We are happy to note that DataXplod has received a very enthusiastic response from TCS project teams and hope that it becomes a useful asset for every TCS project in addressing their data needs.

10. ACKNOWLEDGMENTS

We would like to thank Anshula Dhar, Rahul Kelkar, Amol Khanapurkar, Aniket Kulkarni, Amol Limaye, Rekha Pathak, Nikhil Patwardhan, Kalaiyaran Periasamy, Ellora Praharaj, Ravishankar Rajamony, Aarti Ramakrishnan, Ashim Roy, Rajat Satender, Aakash Shroff, Suparna Soman, and Leena Walavalkar for their valuable inputs and active support.

We are grateful to all our account teams who have used DataXplod and given us valuable feedback that has helped us develop useful functionalities in DataXplod resulting in great value addition.

Special thanks to Siva Ganesan (Head, Assurance Services), Vijayalakshmi Gopal (Head, Corporate Tools Group), Dr Rajesh Mansharamani (Head, Performance Engineering Lab), and Dr Harrick Vin (Head, Systems Research Lab) for their continued guidance, support and encouragement for this activity.

11. REFERENCES


