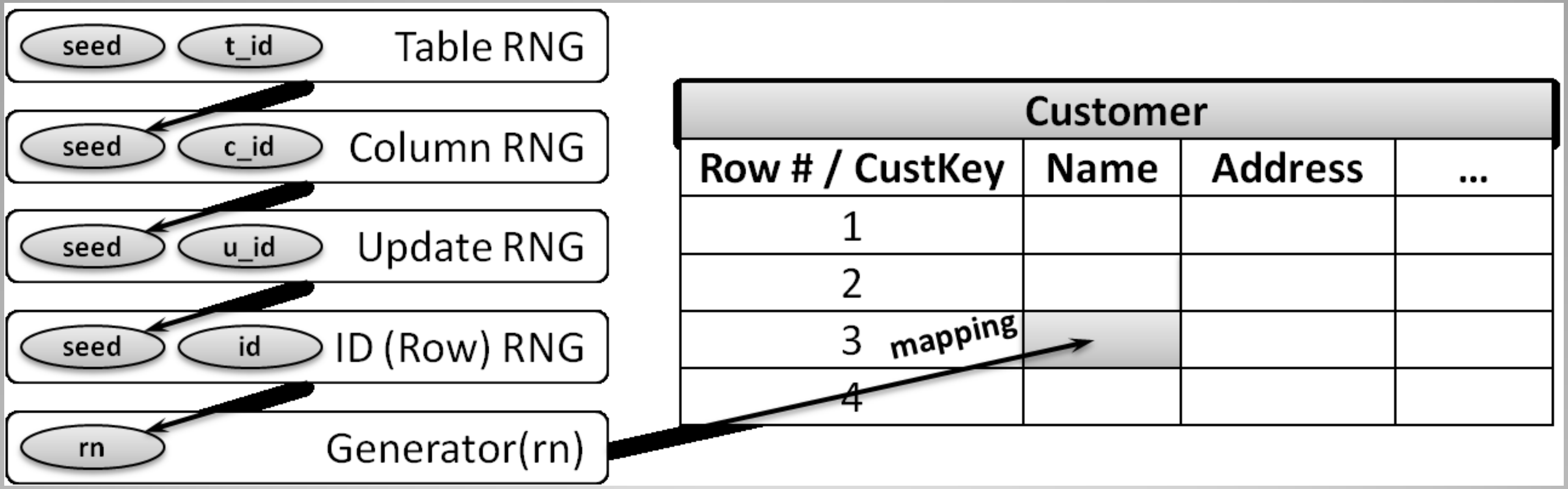
Introducing PDGF
A Framework For Big Data Generation

Tilmann Rabl
Middleware Systems Research Group
University of Toronto
WBDB2012.in, Pune, India, 17.12.12

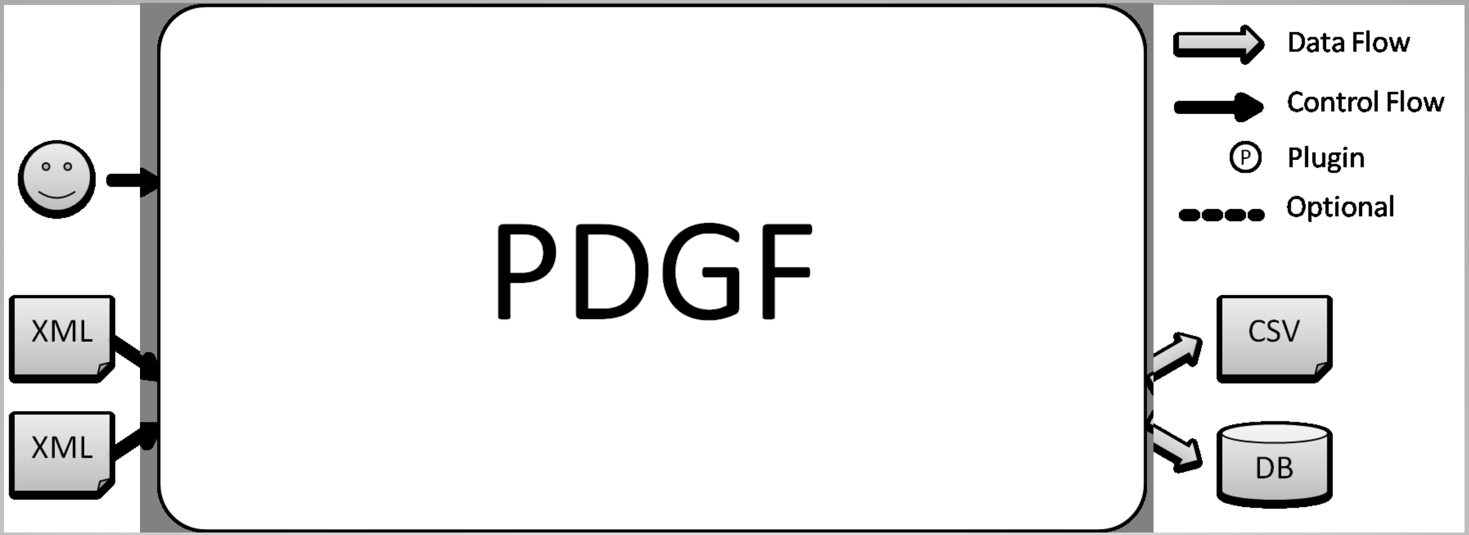
- Generic
- Repeatable
- Configurable
- Extensible
- Fast
- Scalable
- Java based

Parallel Data Generation Framework



- Deterministic random number generation (xorshift)
- Hierarchical seeding strategy
  - Schema → Table → Column → Update→ Row → Generator
  - Uses deterministic seeds
  - Guarantees that n-th random number determines n-th value
  - Even for large schemas all seeds can be cached
- Repeatable, parallel, deterministic generation

Deterministic Data Generation



- Controller → Initialization
- Meta Scheduler → Inter node scheduling
- Scheduler → Inter thread scheduling
- Worker → Blockwise data generation
- Update Black Box → Co-ordination of data updates
- Seeding System → Random sequence adaption
- Generators → Value generation
- Output system → Data formating

Architecture

- XML file
- Project seed
- RNG
- Definition of the database structure
  - Tables, fields, generators
  - Properties
  - Update definition

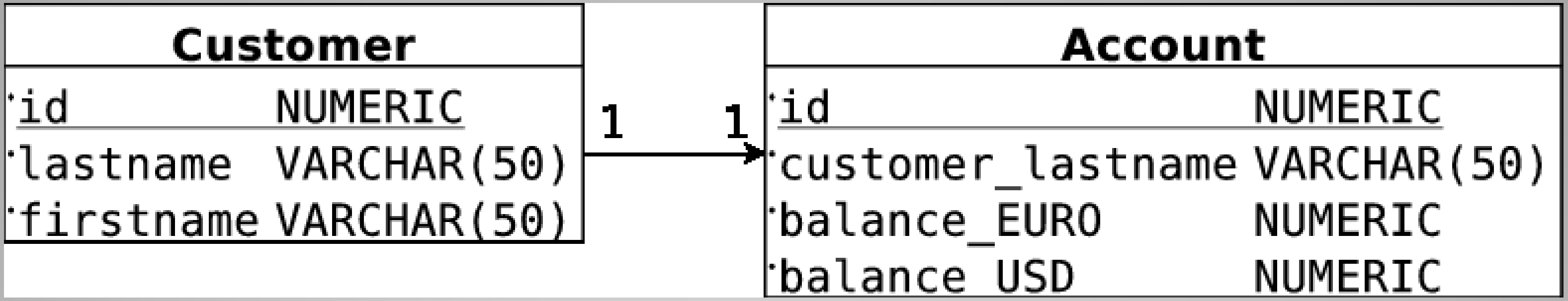
```xml
<?xml version="1.0" encoding="UTF-8"?>
<schema name="demo” [..] >

 <seed>1234567890</seed>
 <rng name="PdgfDefaultRandom"/>

 <!-- Scale factor and properties -->
 <property name="SF" type="double">
   5000
 </property>
 <property name="EXCHANGE_RATE" type="double">
  1.25
 </property>

 <table name="Customer">
  <size>1000 * ${SF}</size>
  <field name="id" size="" type="NUMERIC">
   <gen_IdGenerator/>
  </field>
    [..]
 </table>

 <table name="Account">
  [..]
 </table>
</schema>
```

Schema Configuration File

Generation Configuration File

- XML file
- Defines the output
  - Scheduling
  - Data format
  - Sorting
  - File name and location
- Post processing
  - Filtering of values
  - Merging of tables
  - Splitting of tables
  - Templates (e.g. XML / queries)

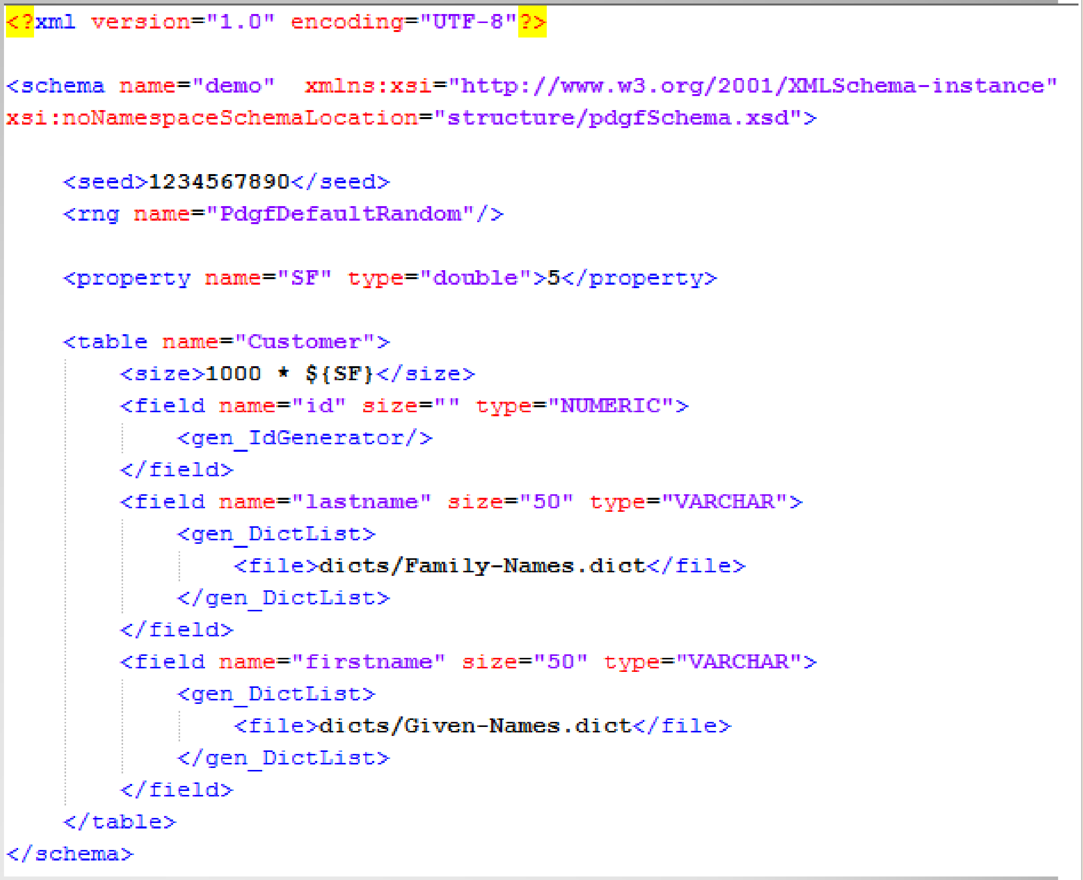
```
<?xml version="1.0" encoding="UTF-8"?>
<generation>
 <scheduler name="DefaultScheduler" />

 <output name="CSVRowOutput">
  <fileTemplate>
    outputDir + table.getName() + fileEnding
  </fileTemplate>
  <outputDir>output/</outputDir>
  <fileEnding>.txt</fileEnding>
  <delimiter>|</delimiter>
 </output>

 <schema name="demo">
  <tables>
  </tables>
 </schema>
</generation>
```



- Customer
  - <u>ID</u>, last name, first name
- Account
  - <u>ID</u>, last name, balance in Euro, balance in USD

Demonstration

- Customer
  - 1000 per SF
  - Id (unique)
  - First name
    - From dictionary
  - Last name
    - From dictionary

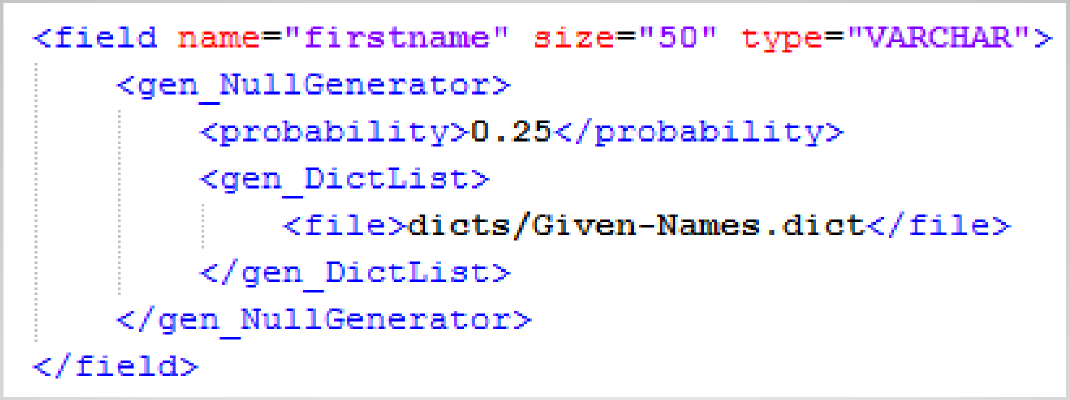


```xml
<?xml version="1.0" encoding="UTF-8"?>

<schema name="demo"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="structure/pdgfSchema.xsd">

    <seed>1234567890</seed>
    <rng name="PdgfDefaultRandom"/>

    <property name="SF" type="double">5</property>

    <table name="Customer">
        <size>1000 * ${SF}</size>
        <field name="id" size="" type="NUMERIC">
            <gen_IdGenerator/>
        </field>
        <field name="lastname" size="50" type="VARCHAR">
            <gen_DictList>
                <file>dicts/Family-Names.dict</file>
            </gen_DictList>
        </field>
        <field name="firstname" size="50" type="VARCHAR">
            <gen_DictList>
                <file>dicts/Given-Names.dict</file>
            </gen_DictList>
        </field>
    </table>
</schema>
```

Customer Table

Null Values

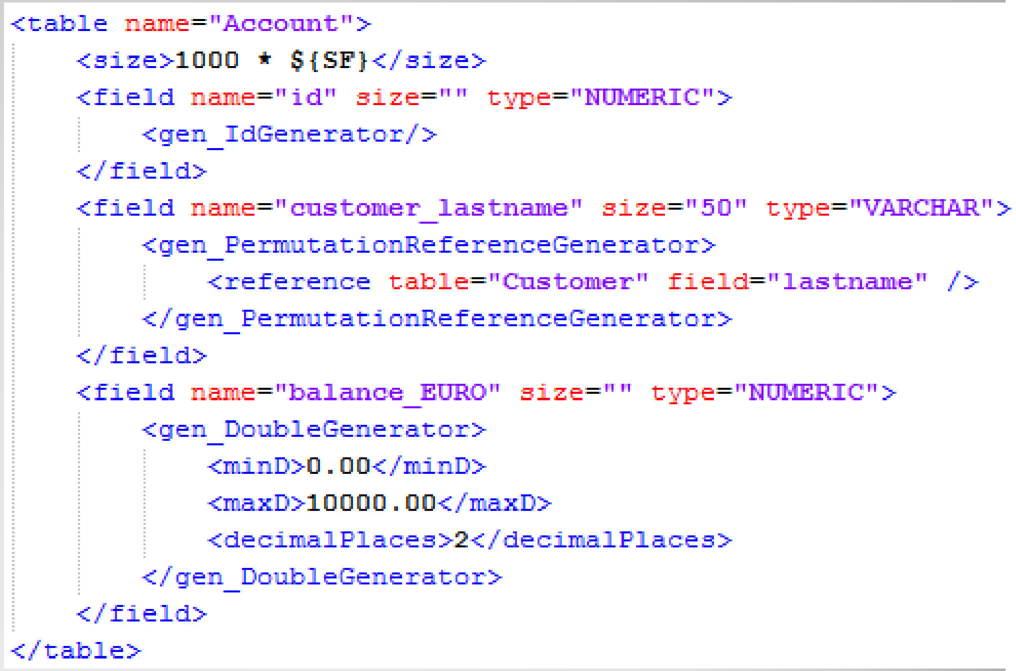
- Some customers have no first name
  - NullGenerator (meta generator)
  - 25% probability of null values



```
<field name="firstname" size="50" type="VARCHAR">
    <gen_NullGenerator>
        <probability>0.25</probability>
        <gen_DictList>
            <file>dicts/Given-Names.dict</file>
        </gen_DictList>
    </gen_NullGenerator>
</field>
```

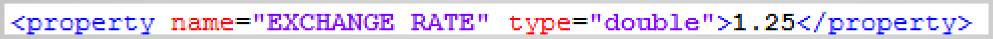
Account Table

- Account table
  - ID
    - Unique
  - Customer last name
    - Unique Reference
  - Balance
    - Numerical value

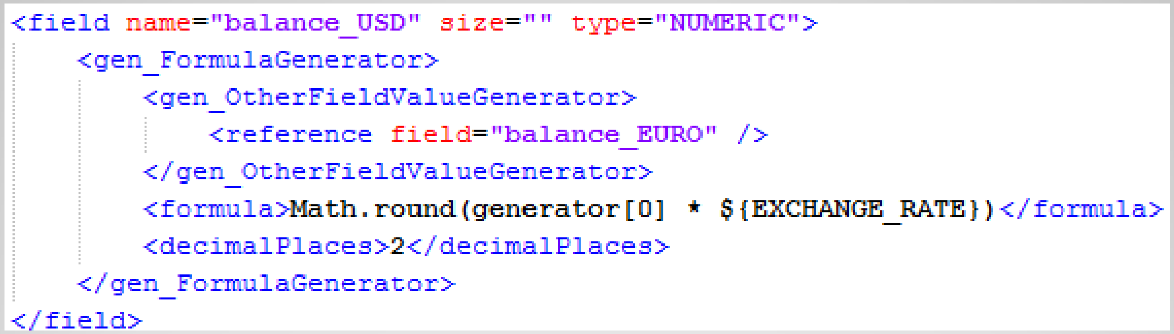


```xml
<table name="Account">
    <size>1000 * ${SF}</size>
    <field name="id" size="" type="NUMERIC">
        <gen_IdGenerator/>
    </field>
    <field name="customer_lastname" size="50" type="VARCHAR">
        <gen_PermutationReferenceGenerator>
            <reference table="Customer" field="lastname" />
        </gen_PermutationReferenceGenerator>
    </field>
    <field name="balance_EURO" size="" type="NUMERIC">
        <gen_DoubleGenerator>
            <minD>0.00</minD>
            <maxD>10000.00</maxD>
            <decimalPlaces>2</decimalPlaces>
        </gen_DoubleGenerator>
    </field>
</table>
```

- Adding a formula based field
    - Balance in USD
    - Calculated from reference and property
    - Properties can be altered from command line



```
<property name="EXCHANGE RATE" type="double">1.25</property>
```



```
<field name="balance_USD" size="" type="NUMERIC">
    <gen_FormulaGenerator>
        <gen_OtherFieldValueGenerator>
            <reference field="balance_EURO" />
        </gen_OtherFieldValueGenerator>
        <formula>Math.round(generator[0] * ${EXCHANGE_RATE})</formula>
        <decimalPlaces>2</decimalPlaces>
    </gen_FormulaGenerator>
</field>
```

More Fancy Features

Implemented Data Generators

- SetQuery (Set Query Benchmark by Pat O'Neil)
  - Single table, 21 columns
  - 250 lines in schema and generation XML files
  - TPC TC’10
- TPC-H & SSB (Star Schema Benchmark by Pat O'Neil)
  - 8 tables, 61 columns
  - 500 lines in schema and generation XML files
  - TPC TC’11
- TPC-DI (aka TPC-ETL)
  - 20 tables, more than 200 columns
  - 6000 lines in schema and generation XML files
  - In progress
- More to come…

- Further information
  - www.paralleldatageneration.org
  - Presentation, tutorial, and PDGF-V2 online
- Pro version in planning
- Contact
  - tilmann.rabl@paralleldatageneration.org

Questions?