DataSynth: Democratizing property graph generation

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

LDBC

Oracle Labs

Joan Guisado Gámez and Arnau Prat Pérez

DAMA-UPC

01/09/2017

10th LDBC TUC Meeting

Why generating property graphs?

- We need data for:
  - testing
  - benchmarking
  - prototyping
- Real data is not always available
  - Privacy issues
  - Valuable asset
  - Not large enough
  - etc.
- Synthetic property graph generation can be an alternative

Democratizing?

*“**Democratization of technology** refers to the process by which access to technology rapidly continues to become more **accessible** to more people” – Wikipedia*

Our goal

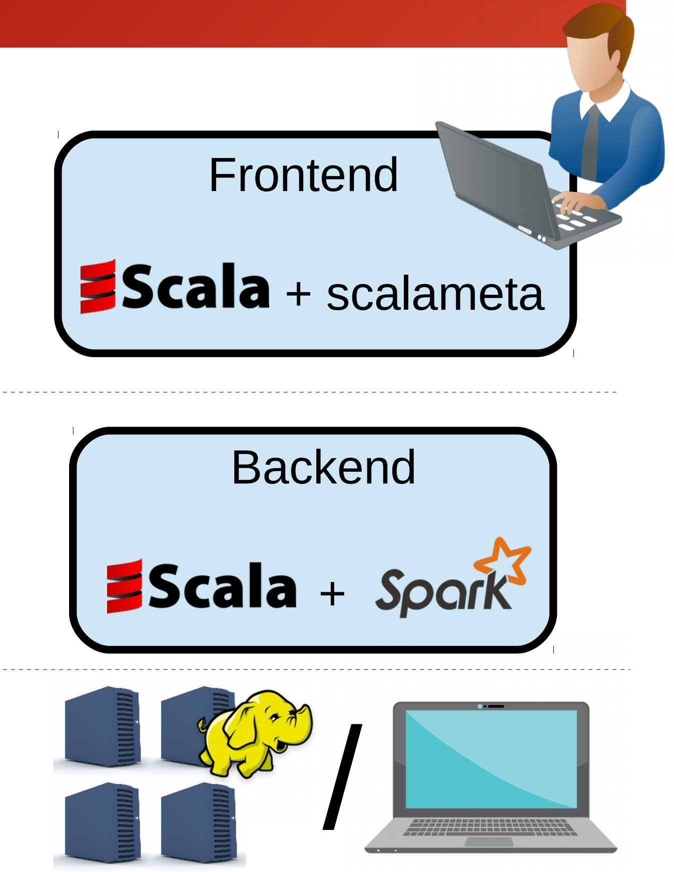
Make property graph generation **accessible** to developers, so they do not need to manually create their own tools

DataSynth

- Is a framework/library for the creation of property graph generators
  - A **DSL** to specify the property graph to generate **declaratively**
  - Hooks to allow **customizing** parts of the property graph generation process, but reusing the other stuff
  - Execute transparently on a cluster to generate **large** amounts of data
- https://github.com/DAMA-UPC/DataSynth
- Highly based on techniques learnt from other projects (e.g. Myriad – https://github.com/TU-Berlin-DIMA/myriad-toolkit)

DataSynth

- DSL on top of Scala + Scalameta
  - Take advantage of all IDE support already available
  - Backend is written in Scala as well
- Backend written in Scala + Spark
  - Easy to execute/deploy on a Yarn cluster or laptop



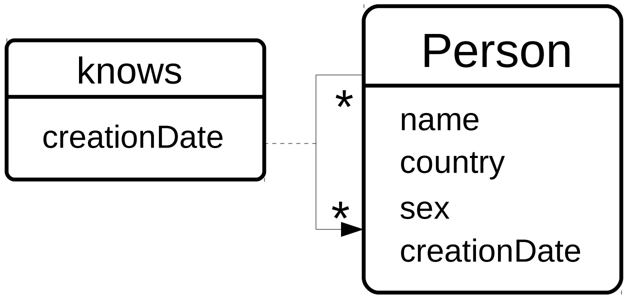
What to support?

- Different **node** and edge **types**
- **Graph size**: Node vs Edge based
- **Properties**:
  - **Types** (Int, Long, String, Timestamp, UUID, etc.)
  - **Constraints**: distributions, correlations, dependencies (>=, <, etc.)
  - **Format requirements**: a given number of decimals, a specific date format, etc.
- **Structural properties**:
  - Degree distributions
  - Clustering coefficient distribution
- **Property-Structure correlations**
  - Nodes tend to be connected to others with specific property values

Example

Social Network

- *Person.country* → $P_{country}(X)$
- *Person.name* → $P_{name}(X \mid country, sex)$
- *Knows.creationDate* → is greater than two connected persons' *creationDate*
- *Knows* degree distribution follows a power-law
- $P_{knows}(X_{country}, Y_{country})$ should be realistic



Frontend

```
@Node
case class Person ( country      : TypeString,
                    sex          : TypeString,
                    name         : TypeString,
                    interest     : TypeString,
                    creationDate : TypeTimestamp
) {

  country is empirical from "path/to/countries/file"

  sex is empirical from "path/to/sex/file"

  name is empirical from "path/to/names/file" dependsOn country dependsOn
sex

  creationDate is uniform withMin "2010-01-01" withMax "2013-01-01"
}


@Edge
case class Knows ( source : Person,
                   target : Person,
                   creationDate : TypeTimestamp) {

  creationDate is generated path.to.KnowsCreationDate.getClass with
initParameter Timestamp("2013/01/01")

  structure BTER with degrees "path/to/degrees/file" with ccs
"path/to/ccs/file"

  correlates source.country and target.country from "path/to/prob/file"
}

DataSynth(config).add(Person(),1000000).add(Knows()).run()
```

https://github.com/DAMA-UPC/Babel/

Frontend

```
@Node
case class Person ( country      : TypeString,
                    sex          : TypeString,
                    name         : TypeString,
                    interest     : TypeString,
                    creationDate : TypeTimestamp
) {

  country is empirical from "path/to/countries/file"

  sex is empirical from "path/to/sex/file"

  name is empirical from "path/to/names/file" dependsOn country dependsOn
sex

  creationDate is uniform withMin "2010-01-01" withMax "2013-01-01"
}
```

← Node Definition

```
@Edge
case class Knows ( source : Person,
                   target : Person,
                    creationDate : TypeTimestamp) {

  creationDate is generated path.to.KnowsCreationDate.getClass with
initParameter Timestamp("2013/01/01")

  structure BTER with degrees "path/to/degrees/file" with ccs
"path/to/ccs/file"

   correlates source.country and target.country from "path/to/prob/file"
}

DataSynth(config).add(Person(),1000000).add(Knows()).run()
```

Frontend

```
@Node
case class Person ( country      : TypeString,
                    sex          : TypeString,
                    name         : TypeString,
                    interest     : TypeString,
                    creationDate : TypeTimestamp
) {

  country is empirical from "path/to/countries/file"

  sex is empirical from "path/to/sex/file"

  name is empirical from "path/to/names/file" dependsOn country dependsOn
sex

  creationDate is uniform withMin "2010-01-01" withMax "2013-01-01"
}


@Edge
case class Knows ( source : Person,
                   target : Person,
                    creationDate : TypeTimestamp) {

  creationDate is generated path.to.KnowsCreationDate.getClass with
initParameter Timestamp("2013/01/01")

  structure BTER with degrees "path/to/degrees/file" with ccs
"path/to/ccs/file"

   correlates source.country and target.country from "path/to/prob/file"
}

DataSynth(config).add(Person(),1000000).add(Knows()).run()
```

Attributes and their types

Frontend

```
@Node
case class Person ( country      : TypeString,
                    sex          : TypeString,
                    name         : TypeString,
                    interest     : TypeString,
                    creationDate : TypeTimestamp
) {

  country is empirical from "path/to/countries/file"

  sex is empirical from "path/to/sex/file"

  name is empirical from "path/to/names/file" dependsOn country dependsOn
sex

  creationDate is uniform withMin "2010-01-01" withMax "2013-01-01"
}


@Edge
case class Knows ( source : Person,
                   target : Person,
                   creationDate : TypeTimestamp) {

  creationDate is generated path.to.KnowsCreationDate.getClass with
initParameter Timestamp("2013/01/01")

  structure BTER with degrees "path/to/degrees/file" with ccs
"path/to/ccs/file"

  correlates source.country and target.country from "path/to/prob/file"
}

DataSynth(config).add(Person(),1000000).add(Knows()).run()
```

How-to-generate block

Frontend

```
@Node
case class Person ( country      : TypeString,
                    sex          : TypeString,
                    name         : TypeString,
                    interest     : TypeString,
                    creationDate : TypeTimestamp
) {
  country is empirical from "path/to/countries/file"

  sex is empirical from "path/to/sex/file"

  name is empirical from "path/to/names/file" dependsOn country dependsOn
sex

  creationDate is uniform withMin "2010-01-01" withMax "2013-01-01"
}


@Edge
case class Knows ( source : Person,
                  target : Person,
                   creationDate : TypeTimestamp) {

  creationDate is generated path.to.KnowsCreationDate.getClass with
initParameter Timestamp("2013/01/01")

  structure BTER with degrees "path/to/degrees/file" with ccs
"path/to/ccs/file"

  correlates source.country and target.country from "path/to/prob/file"
}

DataSynth(config).add(Person(),1000000).add(Knows()).run()
```

Property

Frontend

```
@Node
case class Person ( country      : TypeString,
                    sex          : TypeString,
                    name         : TypeString,
                    interest     : TypeString,
                    creationDate : TypeTimestamp
) {

  country is empirical from "path/to/countries/file"

  sex is empirical from "path/to/sex/file"

  name is empirical from "path/to/names/file" dependsOn country dependsOn
sex

  creationDate is uniform withMin "2010-01-01" withMax "2013-01-01"
}


@Edge
case class Knows ( source : Person,
                   target : Person,
                    creationDate : TypeTimestamp) {

  creationDate is generated path.to.KnowsCreationDate.getClass with
initParameter Timestamp("2013/01/01")

  structure BTER with degrees "path/to/degrees/file" with ccs
"path/to/ccs/file"

   correlates source.country and target.country from "path/to/prob/file"
}

DataSynth(config).add(Person(),1000000).add(Knows()).run()
```

How-to

Frontend

```
@Node
case class Person ( country      : TypeString,
                    sex          : TypeString,
                    name         : TypeString,
                    interest     : TypeString,
                    creationDate : TypeTimestamp
) {

  country is empirical from "path/to/countries/file"

  sex is empirical from "path/to/sex/file"

  name is empirical from "path/to/names/file" dependsOn country dependsOn
sex

  creationDate is uniform withMin "2010-01-01" withMax "2013-01-01"
}


@Edge
case class Knows ( source : Person,
                   target : Person,
                    creationDate : TypeTimestamp) {
|
  creationDate is generated path.to.KnowsCreationDate.getClass with
initParameter Timestamp("2013/01/01")

  structure BTER with degrees "path/to/degrees/file" with ccs
"path/to/ccs/file"

   correlates source.country and target.country from "path/to/prob/file"
}

DataSynth(config).add(Person(),1000000).add(Knows()).run()
```

How-to parameters

Frontend

```
@Node
case class Person ( country      : TypeString,
                    sex          : TypeString,
                    name         : TypeString,
                    interest     : TypeString,
                    creationDate : TypeTimestamp
) {

  country is empirical from "path/to/countries/file"

  sex is empirical from "path/to/sex/file"

  name is empirical from "path/to/names/file" dependsOn country dependsOn
sex

  creationDate is uniform withMin "2010-01-01" withMax "2013-01-01"
}

@Edge
case class Knows ( source : Person,
                   target : Person,
                    creationDate : TypeTimestamp) {

  creationDate is generated path.to.KnowsCreationDate.getClass with
initParameter Timestamp("2013/01/01")

  structure BTER with degrees "path/to/degrees/file" with ccs
"path/to/ccs/file"

   correlates source.country and target.country from "path/to/prob/file"
}

DataSynth(config).add(Person(),1000000).add(Knows()).run()
```

Edge

Frontend

```
@Node
case class Person ( country      : TypeString,
                    sex          : TypeString,
                    name         : TypeString,
                    interest     : TypeString,
                    creationDate : TypeTimestamp
) {

  country is empirical from "path/to/countries/file"

  sex is empirical from "path/to/sex/file"

  name is empirical from "path/to/names/file" dependsOn country dependsOn
sex

  creationDate is uniform withMin "2010-01-01" withMax "2013-01-01"
}


@Edge
case class Knows ( source : Person,
                  target : Person,
                   creationDate : TypeTimestamp) {

  creationDate is generated path.to.KnowsCreationDate.getClass with
initParameter Timestamp("2013/01/01")

  structure BTER with degrees "path/to/degrees/file" with ccs
"path/to/ccs/file"

  correlates source.country and target.country from "path/to/prob/file"
}

DataSynth(config).add(Person(),1000000).add(Knows()).run()
```

Source, target and attributes

Frontend

```
@Node
case class Person ( country      : TypeString,
                    sex          : TypeString,
                    name         : TypeString,
                    interest     : TypeString,
                    creationDate : TypeTimestamp
) {

  country is empirical from "path/to/countries/file"

  sex is empirical from "path/to/sex/file"

  name is empirical from "path/to/names/file" dependsOn country dependsOn
sex

  creationDate is uniform withMin "2010-01-01" withMax "2013-01-01"
}


@Edge
case class Knows ( source : Person,
                   target : Person,
                    creationDate : TypeTimestamp) {

  creationDate is generated path.to.KnowsCreationDate.getClass with
initParameter Timestamp("2013/01/01")

  structure BTER with degrees "path/to/degrees/file" with ccs
"path/to/ccs/file"

  correlates source.country and target.country from "path/to/prob/file"
}

DataSynth(config).add(Person(),1000000).add(Knows()).run()
```

Property configuration

Frontend

```
@Node
case class Person ( country      : TypeString,
                    sex          : TypeString,
                    name         : TypeString,
                    interest     : TypeString,
                    creationDate : TypeTimestamp
) {

  country is empirical from "path/to/countries/file"

  sex is empirical from "path/to/sex/file"

  name is empirical from "path/to/names/file" dependsOn country dependsOn
sex

  creationDate is uniform withMin "2010-01-01" withMax "2013-01-01"
}


@Edge
case class Knows ( source : Person,
                   target : Person,
                    creationDate : TypeTimestamp) {

  creationDate is generated path.to.KnowsCreationDate.getClass with
initParameter Timestamp("2013/01/01")

  structure BTER with degrees "path/to/degrees/file" with ccs
"path/to/ccs/file"

  correlates source.country and target.country from "path/to/prob/file"
}

DataSynth(config).add(Person(),1000000).add(Knows()).run()
```

Structure configuration

Frontend

```
@Node
case class Person ( country      : TypeString,
                    sex          : TypeString,
                    name         : TypeString,
                    interest     : TypeString,
                    creationDate : TypeTimestamp
) {

  country is empirical from "path/to/countries/file"

  sex is empirical from "path/to/sex/file"

  name is empirical from "path/to/names/file" dependsOn country dependsOn
sex

  creationDate is uniform withMin "2010-01-01" withMax "2013-01-01"
}


@Edge
case class Knows ( source : Person,
                   target : Person,
                    creationDate : TypeTimestamp) {

  creationDate is generated path.to.KnowsCreationDate.getClass with
initParameter Timestamp("2013/01/01")

  structure BTER with degrees "path/to/degrees/file" with ccs
"path/to/ccs/file"

  correlates source.country and target.country from "path/to/prob/file"
}

DataSynth(config).add(Person(),1000000).add(Knows()).run()
```

Correlation

Frontend

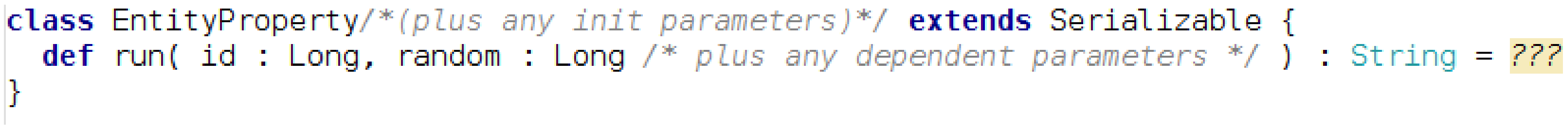
```
@Node
case class Person ( country      : TypeString,
                    sex          : TypeString,
                    name         : TypeString,
                    interest     : TypeString,
                    creationDate : TypeTimestamp
) {

  country is empirical from "path/to/countries/file"

  sex is empirical from "path/to/sex/file"

  name is empirical from "path/to/names/file" dependsOn country dependsOn
sex

  creationDate is uniform withMin "2010-01-01" withMax "2013-01-01"
}


@Edge
case class Knows ( source : Person,
                  target : Person,
                   creationDate : TypeTimestamp) {

  creationDate is generated path.to.KnowsCreationDate.getClass with
initParameter Timestamp("2013/01/01")

  structure BTER with degrees "path/to/degrees/file" with ccs
"path/to/ccs/file"

  correlates source.country and target.country from "path/to/prob/file"
}
```

```
DataSynth(config).add(Person(),1000000).add(Knows()).run()
```

Commit

Frontend

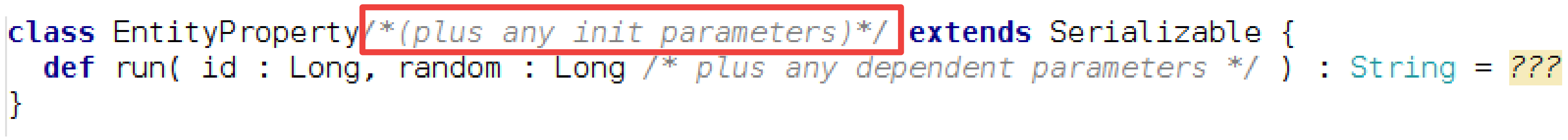
For each pair <entity,property>, the frontend generates a class of this form



```
class EntityProperty/*(plus any init parameters)*/ extends Serializable {
  def run( id : Long, random : Long /* plus any dependent parameters */ ) : String = ???
}
```

Frontend

For each pair <entity,property>, the frontend generates a class of this form

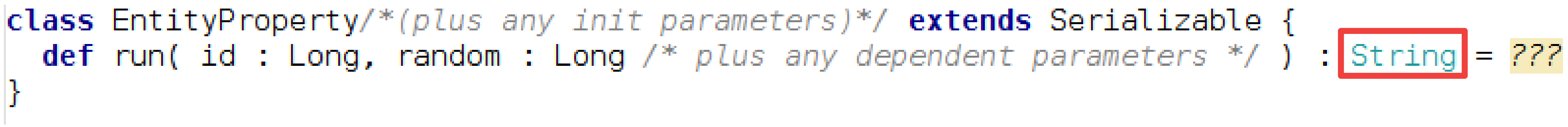


```
class EntityProperty/*(plus any init parameters)*/ extends Serializable {
  def run( id : Long, random : Long /* plus any dependent parameters */ ) : String = ???
}
```

Can contain constructor parameters

Frontend

For each pair <entity,property>, the frontend generates a class of this form

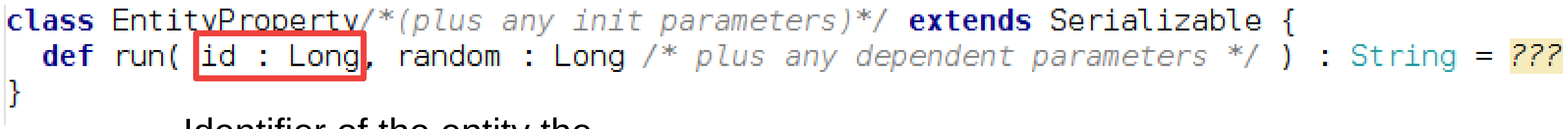


```
class EntityProperty/*(plus any init parameters)*/ extends Serializable {
  def run( id : Long, random : Long /* plus any dependent parameters */ ) : String = ???
}
```

The return value type must match that of the property (e.g. Int, String, Long, Float, etc.)

Frontend

For each pair <entity,property>, the frontend generates a class of this form

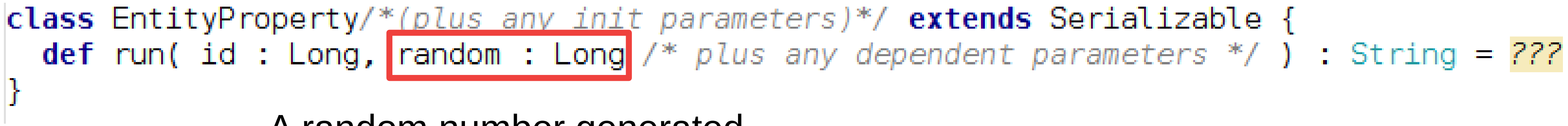


```
class EntityProperty/*(plus any init parameters)*/ extends Serializable {
  def run( id : Long, random : Long /* plus any dependent parameters */ ) : String = ???
}
```

Identifier of the entity the
property is being generated for

Frontend

For each pair <entity,property>, the frontend generates a class of this form

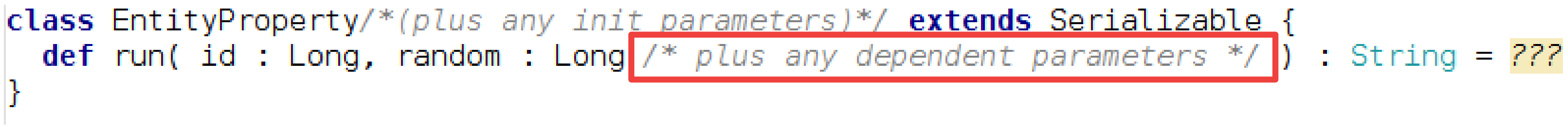


```
class EntityProperty/*(plus any init parameters)*/ extends Serializable {
  def run( id : Long, random : Long /* plus any dependent parameters */ ) : String = ???
}
```

A random number generated
deterministically from ***id***

Frontend

For each pair <entity,property>, the frontend generates a class of this form

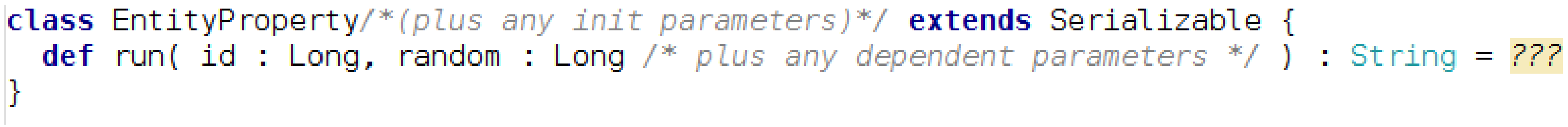


```
class EntityProperty/*(plus any init parameters)*/ extends Serializable {
  def run( id : Long, random : Long /* plus any dependent parameters */ ) : String = ???
}
```

Can take additional dependant parameters

Frontend

For each pair <entity,property>, the frontend generates a class of this form



```
class EntityProperty/*(plus any init parameters)*/ extends Serializable {
  def run( id : Long, random : Long /* plus any dependent parameters */ ) : String = ???
}
```

“*run*” must be a **pure function !!!**

Frontend

For each pair <entity,property>, the frontend generates a class of this form

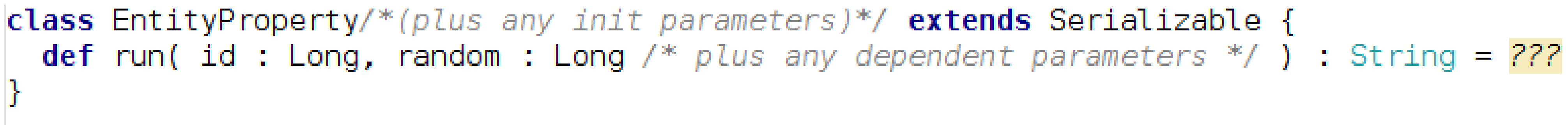
```
class EntityProperty/*(plus any init parameters)*/ extends Serializable {
  def run( id : Long, random : Long /* plus any dependent parameters */ ) : String = ???
}
```

Or users can provide their manually implemented classes

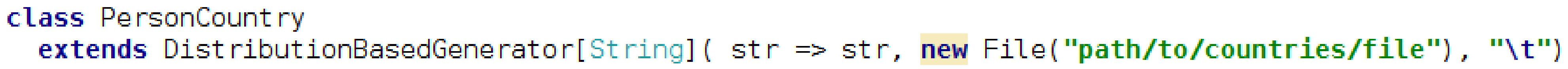
```
class KnowsDate( max : Long ) extends Serializable {
  def run( id : Long, random : Long, date1 : Long, date2 : Long ) : Long = {
    val min = Math.max(date1,date2)
    val ratio = random / Long.MaxValue.toDouble
    ((max - min)*ratio + min).toLong
  }
}
```

Frontend

For each pair <entity,property>, the frontend generates a class of this form



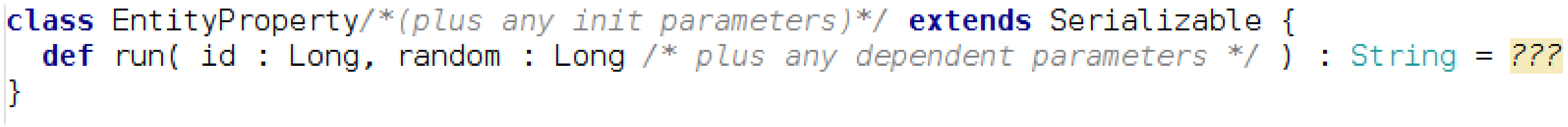
```
class EntityProperty/*(plus any init parameters)*/ extends Serializable {
  def run( id : Long, random : Long /* plus any dependent parameters */ ) : String = ???
}
```



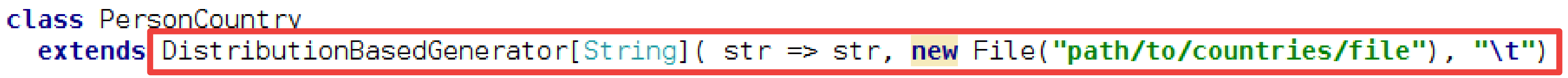
```
class PersonCountry
  extends DistributionBasedGenerator[String]( str => str, new File("path/to/countries/file"), "\t")
```

Frontend

For each pair <entity,property>, the frontend generates a class of this form



```
class EntityProperty/*(plus any init parameters)*/ extends Serializable {
  def run( id : Long, random : Long /* plus any dependent parameters */ ) : String = ???
}
```

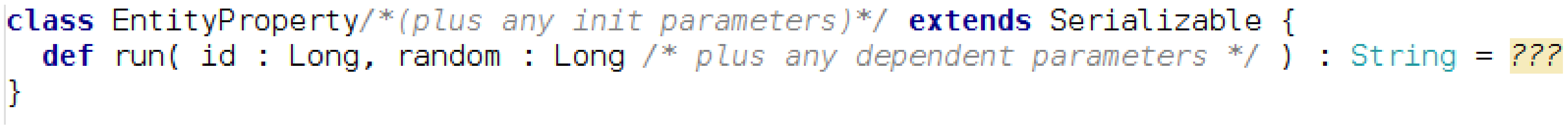


```
class PersonCountry
  extends DistributionBasedGenerator[String]( str => str, new File("path/to/countries/file"), "\t")
```

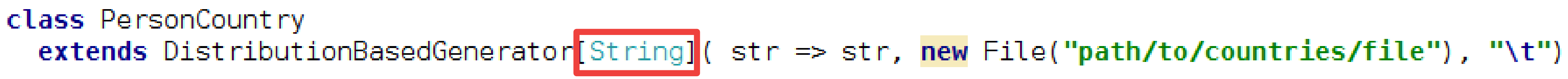
This class already has “*run*” defined

Frontend

For each pair <entity,property>, the frontend generates a class of this form



```
class EntityProperty/*(plus any init parameters)*/ extends Serializable {
  def run( id : Long, random : Long /* plus any dependent parameters */ ) : String = ???
}
```

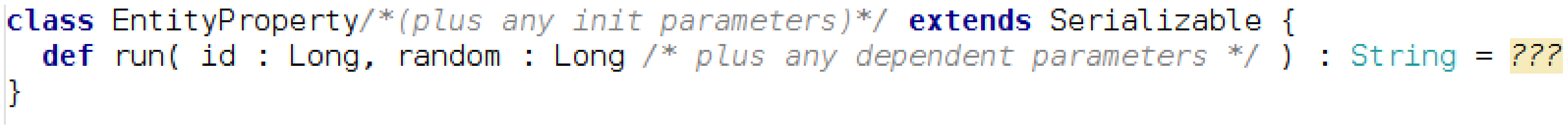


```
class PersonCountry
  extends DistributionBasedGenerator[String]( str => str, new File("path/to/countries/file"), "\t")
```

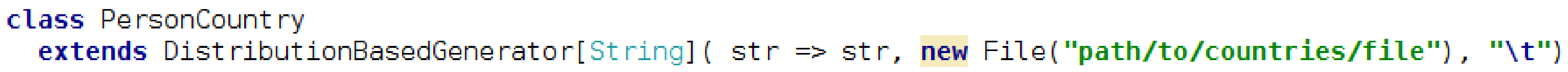
We use the generics parameter
to tune the return type of “*run*”

Frontend

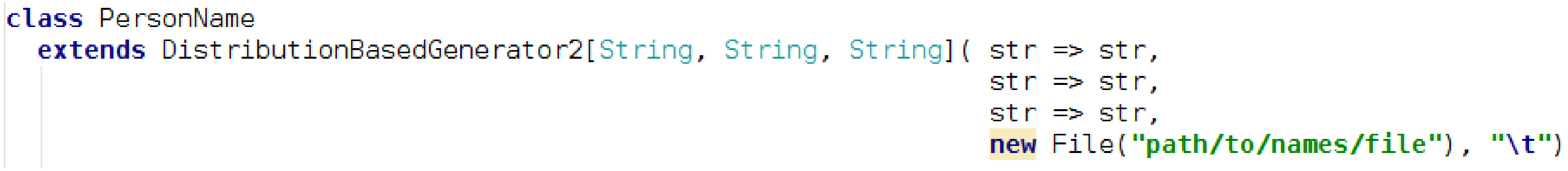
For each pair <entity,property>, the frontend generates a class of this form



```
class EntityProperty/*(plus any init parameters)*/ extends Serializable {
  def run( id : Long, random : Long /* plus any dependent parameters */ ) : String = ???
}
```



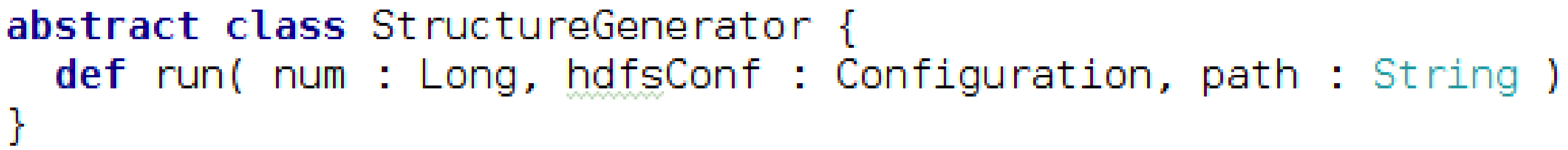
```
class PersonCountry
  extends DistributionBasedGenerator[String]( str => str, new File("path/to/countries/file"), "\t")
```



```
class PersonName
  extends DistributionBasedGenerator2[String, String, String]( str => str,
                                                                str => str,
                                                                str => str,
                                                                new File("path/to/names/file"), "\t")
```

Frontend

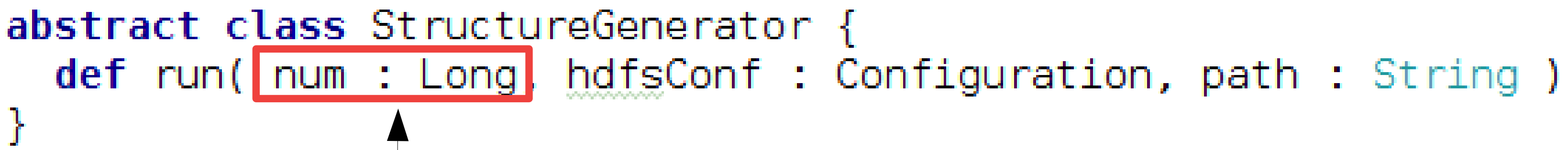
Structure generators are provided by the framework



```
abstract class StructureGenerator {
  def run( num : Long, hdfsConf : Configuration, path : String )
}
```

Frontend

Structure generators are provided by the framework

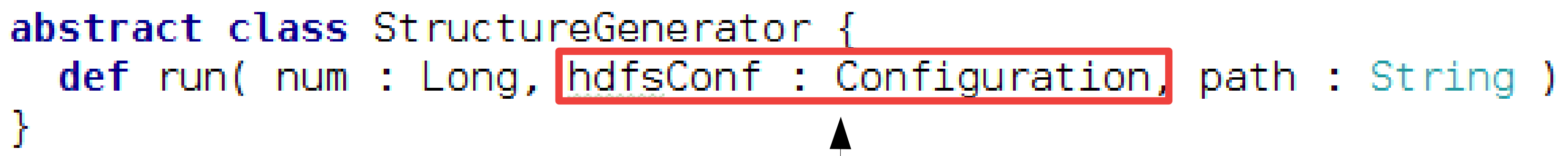


```
abstract class StructureGenerator {
  def run( num : Long, hdfsConf : Configuration, path : String )
}
```

Number of nodes

Frontend

Structure generators are provided by the framework

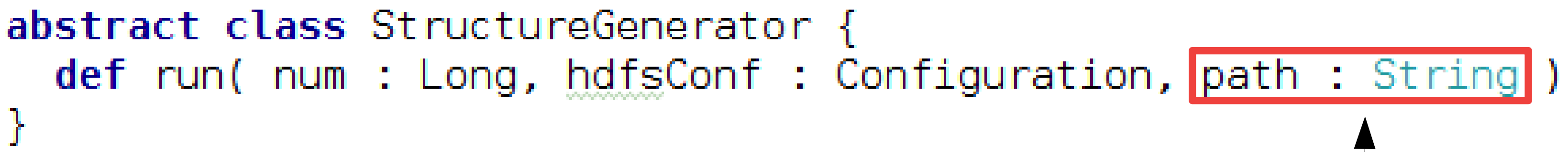


```
abstract class StructureGenerator {
  def run( num : Long, hdfsConf : Configuration, path : String )
}
```

HDFS configuration

Frontend

Structure generators are provided by the framework

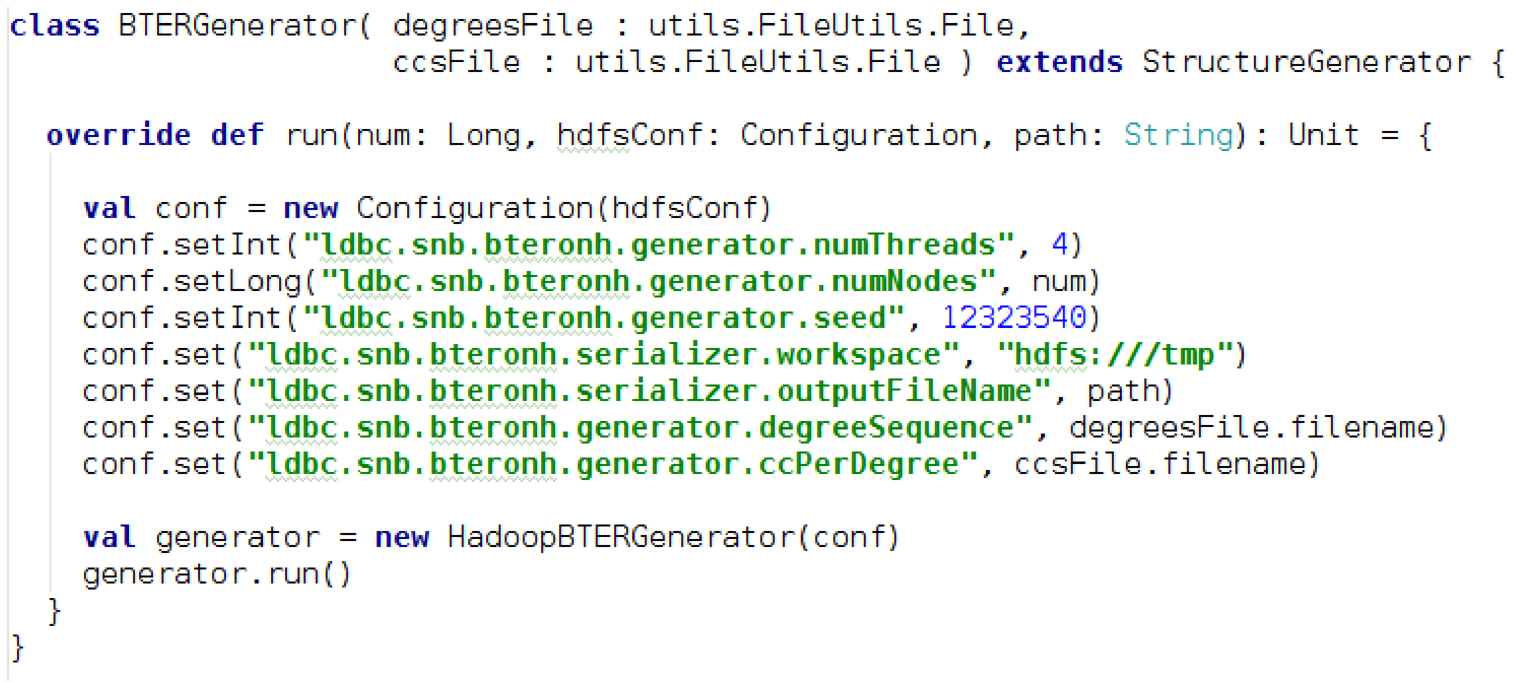


```
abstract class StructureGenerator {
  def run( num : Long, hdfsConf : Configuration, path : String )
}
```

Path to output file

Frontend

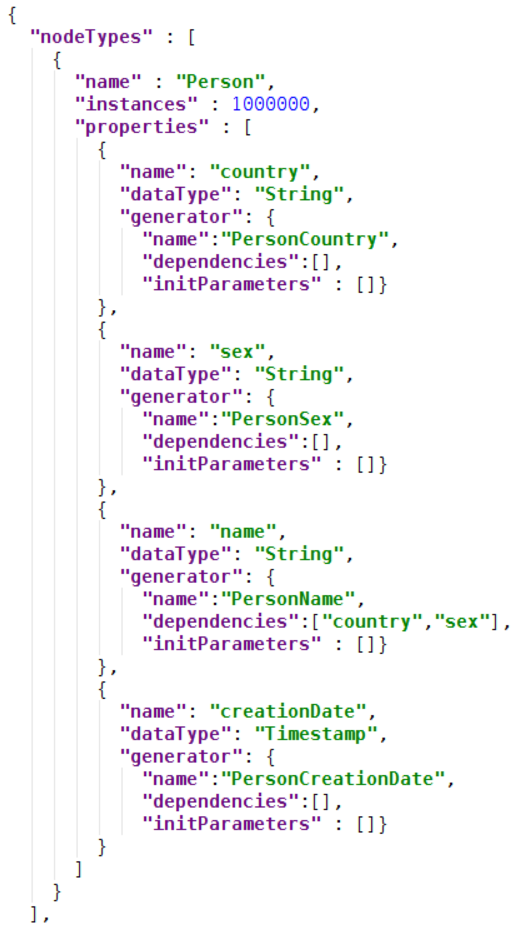
Structure generators are provided by the framework



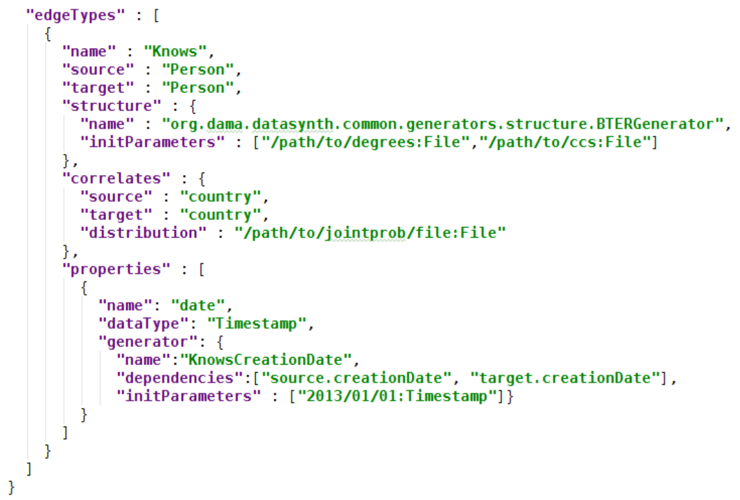
```scala
class BTERGenerator( degreesFile : utils.FileUtils.File,
                     ccsFile : utils.FileUtils.File ) extends StructureGenerator {

  override def run(num: Long, hdfsConf: Configuration, path: String): Unit = {

    val conf = new Configuration(hdfsConf)
    conf.setInt("ldbc.snb.bteronh.generator.numThreads", 4)
    conf.setLong("ldbc.snb.bteronh.generator.numNodes", num)
    conf.setInt("ldbc.snb.bteronh.generator.seed", 12323540)
    conf.set("ldbc.snb.bteronh.serializer.workspace", "hdfs:///tmp")
    conf.set("ldbc.snb.bteronh.serializer.outputFileName", path)
    conf.set("ldbc.snb.bteronh.generator.degreeSequence", degreesFile.filename)
    conf.set("ldbc.snb.bteronh.generator.ccPerDegree", ccsFile.filename)

    val generator = new HadoopBTERGenerator(conf)
    generator.run()
  }
}
```

https://github.com/DAMA-UPC/BTERonH

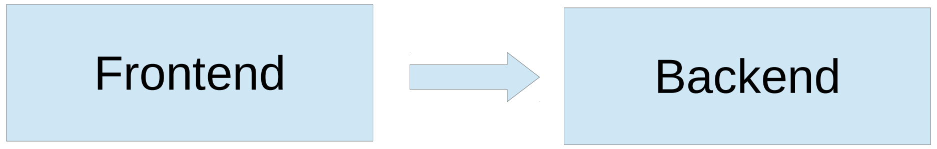
Frontend/Backend interface



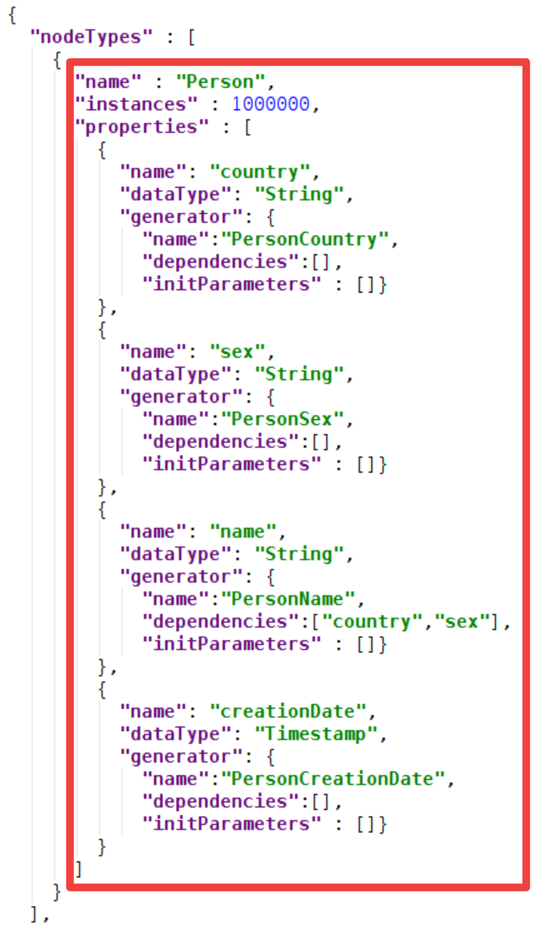
```
{
  "nodeTypes" : [
    {
      "name" : "Person",
      "instances" : 1000000,
      "properties" : [
        {
          "name": "country",
          "dataType": "String",
          "generator": {
            "name":"PersonCountry",
            "dependencies":[],
            "initParameters" : []}
        },
        {
          "name": "sex",
          "dataType": "String",
          "generator": {
            "name":"PersonSex",
            "dependencies":[],
            "initParameters" : []}
        },
        {
          "name": "name",
          "dataType": "String",
          "generator": {
            "name":"PersonName",
            "dependencies":["country","sex"],
            "initParameters" : []}
        },
        {
          "name": "creationDate",
          "dataType": "Timestamp",
          "generator": {
            "name":"PersonCreationDate",
            "dependencies":[],
            "initParameters" : []}
        }
      ]
    }
  ],
```



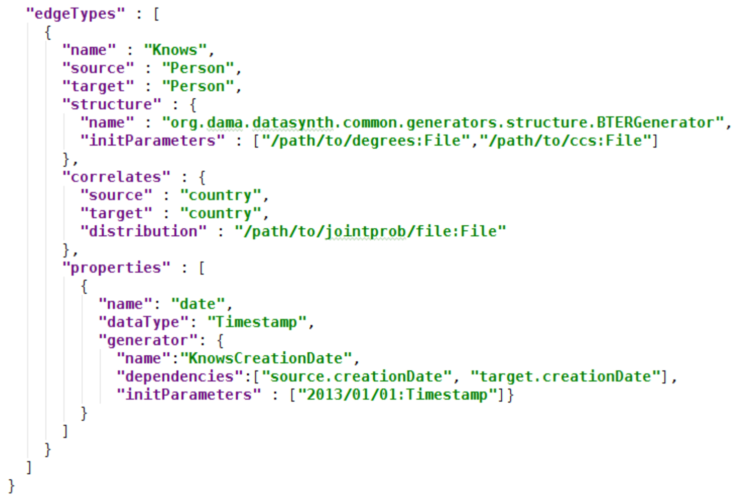
```
  "edgeTypes" : [
    {
      "name" : "Knows",
      "source" : "Person",
      "target" : "Person",
      "structure" : {
        "name" : "org.dama.datasynth.common.generators.structure.BTERGenerator",
        "initParameters" : ["/path/to/degrees:File","/path/to/ccs:File"]
      },
      "correlates" : {
        "source" : "country",
        "target" : "country",
        "distribution" : "/path/to/jointprob/file:File"
      },
      "properties" : [
        {
          "name": "date",
          "dataType": "Timestamp",
          "generator": {
            "name":"KnowsCreationDate",
            "dependencies":["source.creationDate", "target.creationDate"],
            "initParameters" : ["2013/01/01:Timestamp"]}
        }
      ]
    }
  ]
}
```



Frontend/Backend interface



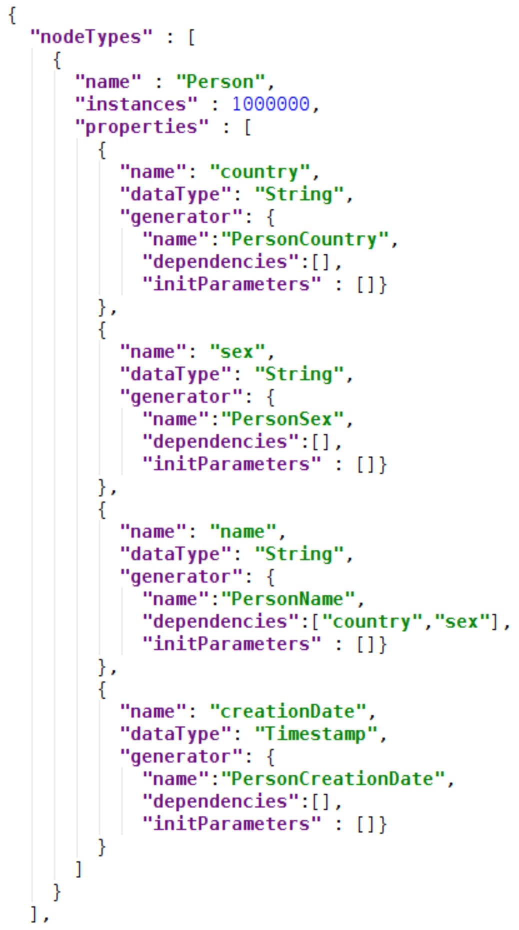
```
{
  "nodeTypes" : [
    {
      "name" : "Person",
      "instances" : 1000000,
      "properties" : [
        {
          "name": "country",
          "dataType": "String",
          "generator": {
            "name":"PersonCountry",
            "dependencies":[],
            "initParameters" : []}
        },
        {
          "name": "sex",
          "dataType": "String",
          "generator": {
            "name":"PersonSex",
            "dependencies":[],
            "initParameters" : []}
        },
        {
          "name": "name",
          "dataType": "String",
          "generator": {
            "name":"PersonName",
            "dependencies":["country","sex"],
            "initParameters" : []}
        },
        {
          "name": "creationDate",
          "dataType": "Timestamp",
          "generator": {
            "name":"PersonCreationDate",
            "dependencies":[],
            "initParameters" : []}
        }
      ]
    }
  ],
```



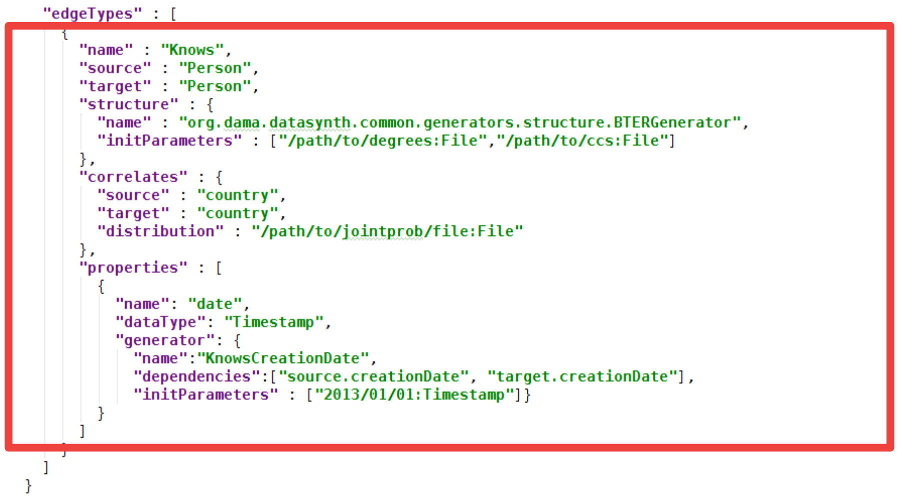
```
  "edgeTypes" : [
    {
      "name" : "Knows",
      "source" : "Person",
      "target" : "Person",
      "structure" : {
        "name" : "org.dama.datasynth.common.generators.structure.BTERGenerator",
        "initParameters" : ["/path/to/degrees:File","/path/to/ccs:File"]
      },
      "correlates" : {
        "source" : "country",
        "target" : "country",
        "distribution" : "/path/to/jointprob/file:File"
      },
      "properties" : [
        {
          "name": "date",
          "dataType": "Timestamp",
          "generator": {
            "name":"KnowsCreationDate",
            "dependencies":["source.creationDate", "target.creationDate"],
            "initParameters" : ["2013/01/01:Timestamp"]}
        }
      ]
    }
  ]
}
```

Node Type

Frontend/Backend interface



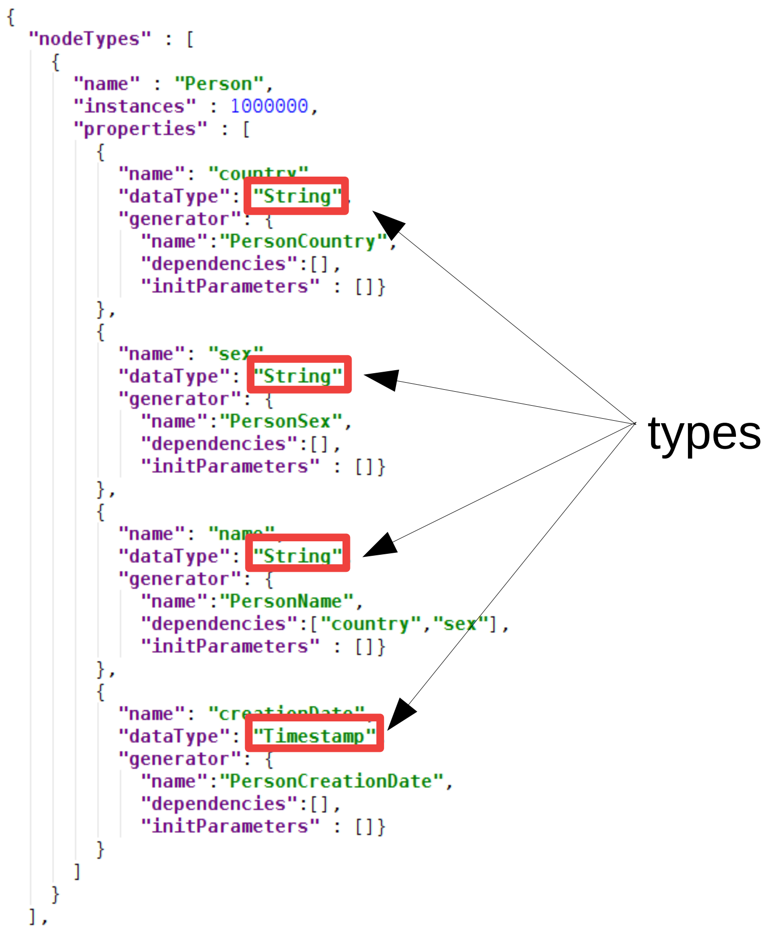
```
{
  "nodeTypes" : [
    {
      "name" : "Person",
      "instances" : 1000000,
      "properties" : [
        {
          "name": "country",
          "dataType": "String",
          "generator": {
            "name":"PersonCountry",
            "dependencies":[],
            "initParameters" : []}
        },
        {
          "name": "sex",
          "dataType": "String",
          "generator": {
            "name":"PersonSex",
            "dependencies":[],
            "initParameters" : []}
        },
        {
          "name": "name",
          "dataType": "String",
          "generator": {
            "name":"PersonName",
            "dependencies":["country","sex"],
            "initParameters" : []}
        },
        {
          "name": "creationDate",
          "dataType": "Timestamp",
          "generator": {
            "name":"PersonCreationDate",
            "dependencies":[],
            "initParameters" : []}
        }
      ]
    }
  ],
```



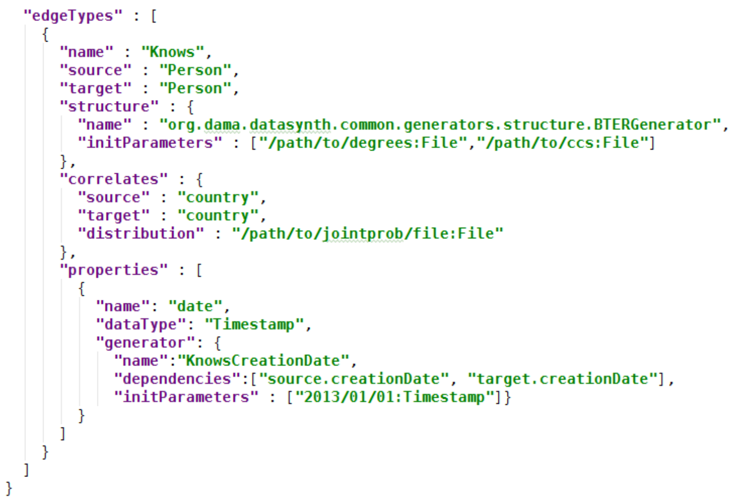
```
  "edgeTypes" : [
    {
      "name" : "Knows",
      "source" : "Person",
      "target" : "Person",
      "structure" : {
        "name" : "org.dama.datasynth.common.generators.structure.BTERGenerator",
        "initParameters" : ["/path/to/degrees:File","/path/to/ccs:File"]
      },
      "correlates" : {
        "source" : "country",
        "target" : "country",
        "distribution" : "/path/to/jointprob/file:File"
      },
      "properties" : [
        {
          "name": "date",
          "dataType": "Timestamp",
          "generator": {
            "name":"KnowsCreationDate",
            "dependencies":["source.creationDate", "target.creationDate"],
            "initParameters" : ["2013/01/01:Timestamp"]}
        }
      ]
    }
  ]
}
```

Edge Type

Frontend/Backend interface

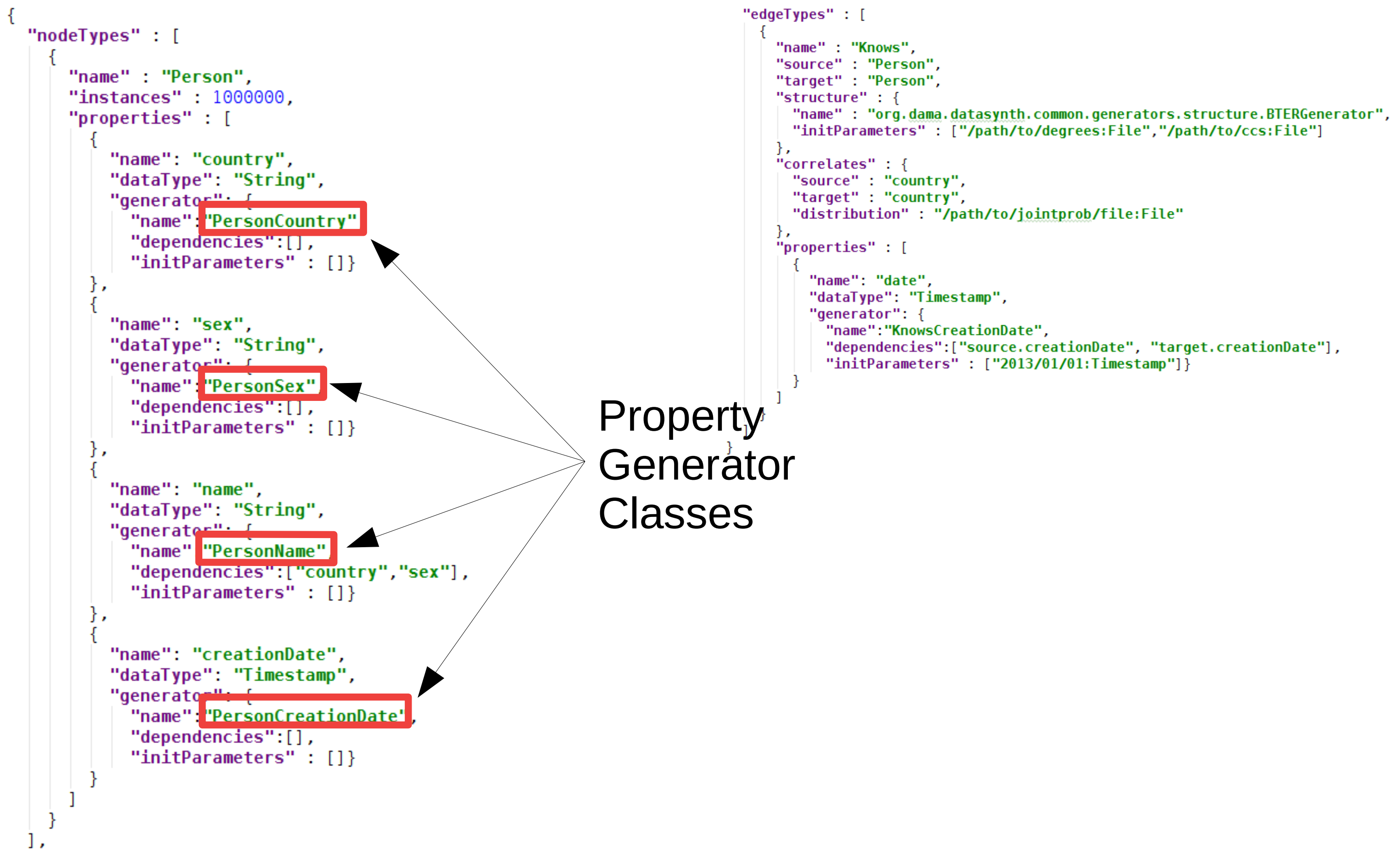


```
{
  "nodeTypes" : [
    {
      "name" : "Person",
      "instances" : 1000000,
      "properties" : [
        {
          "name": "country",
          "dataType": "String",
          "generator": {
            "name":"PersonCountry",
            "dependencies":[],
            "initParameters" : []}
        },
        {
          "name": "sex",
          "dataType": "String",
          "generator": {
            "name":"PersonSex",
            "dependencies":[],
            "initParameters" : []}
        },
        {
          "name": "name",
          "dataType": "String",
          "generator": {
            "name":"PersonName",
            "dependencies":["country","sex"],
            "initParameters" : []}
        },
        {
          "name": "creationDate",
          "dataType": "Timestamp",
          "generator": {
            "name":"PersonCreationDate",
            "dependencies":[],
            "initParameters" : []}
        }
      ]
    }
  ],
```



```
"edgeTypes" : [
  {
    "name" : "Knows",
    "source" : "Person",
    "target" : "Person",
    "structure" : {
      "name" : "org.dama.datasynth.common.generators.structure.BTERGenerator",
      "initParameters" : ["/path/to/degrees:File","/path/to/ccs:File"]
    },
    "correlates" : {
      "source" : "country",
      "target" : "country",
      "distribution" : "/path/to/jointprob/file:File"
    },
    "properties" : [
      {
        "name": "date",
        "dataType": "Timestamp",
        "generator": {
          "name":"KnowsCreationDate",
          "dependencies":["source.creationDate", "target.creationDate"],
          "initParameters" : ["2013/01/01:Timestamp"]}
      }
    ]
  }
]
}
```

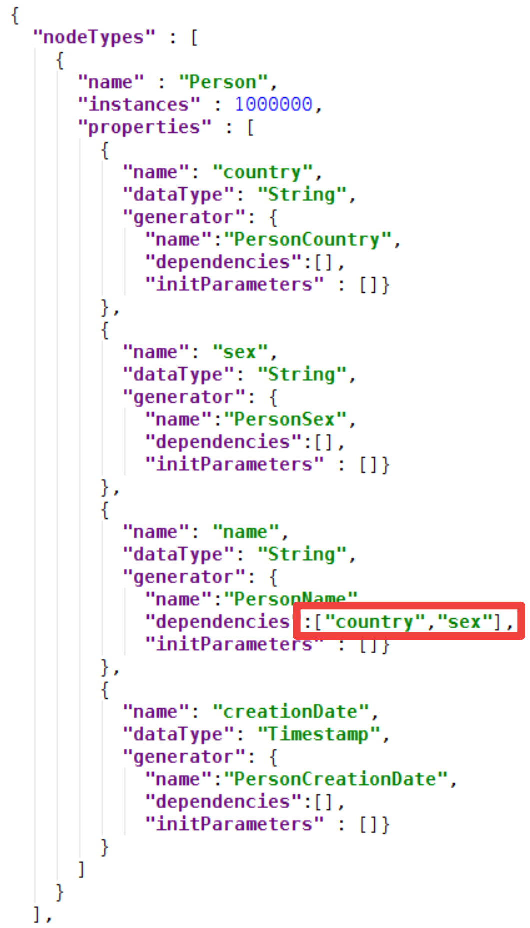
Frontend/Backend interface



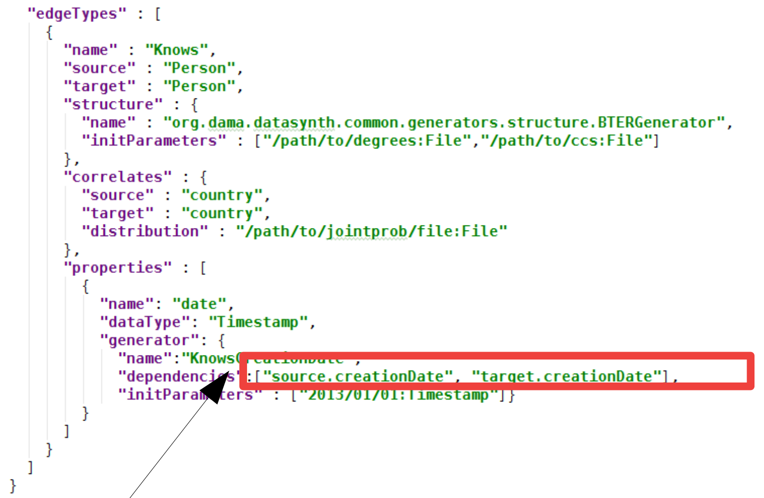
```
{
  "nodeTypes" : [
    {
      "name" : "Person",
      "instances" : 1000000,
      "properties" : [
        {
          "name": "country",
          "dataType": "String",
          "generator": {
            "name":"PersonCountry",
            "dependencies":[],
            "initParameters" : []}
        },
        {
          "name": "sex",
          "dataType": "String",
          "generator": {
            "name":"PersonSex",
            "dependencies":[],
            "initParameters" : []}
        },
        {
          "name": "name",
          "dataType": "String",
          "generator": {
            "name":"PersonName",
            "dependencies":["country","sex"],
            "initParameters" : []}
        },
        {
          "name": "creationDate",
          "dataType": "Timestamp",
          "generator": {
            "name":"PersonCreationDate",
            "dependencies":[],
            "initParameters" : []}
        }
      ]
    }
  ],
```

```
"edgeTypes" : [
  {
    "name" : "Knows",
    "source" : "Person",
    "target" : "Person",
    "structure" : {
      "name" : "org.dama.datasynth.common.generators.structure.BTERGenerator",
      "initParameters" : ["/path/to/degrees:File","/path/to/ccs:File"]
    },
    "correlates" : {
      "source" : "country",
      "target" : "country",
      "distribution" : "/path/to/jointprob/file:File"
    },
    "properties" : [
      {
        "name": "date",
        "dataType": "Timestamp",
        "generator": {
          "name":"KnowsCreationDate",
          "dependencies":["source.creationDate", "target.creationDate"],
          "initParameters" : ["2013/01/01:Timestamp"]}
      }
    ]
  }
]
}
```

Frontend/Backend interface



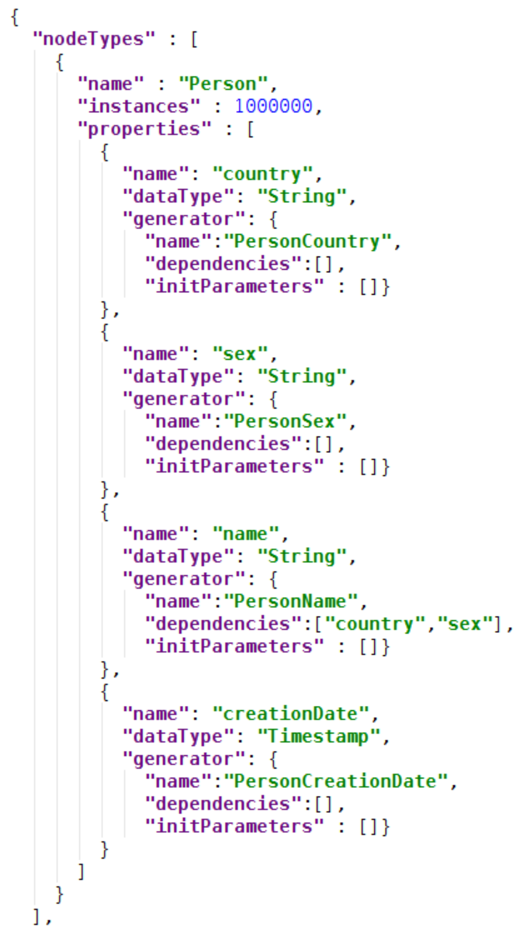
```
{
  "nodeTypes" : [
    {
      "name" : "Person",
      "instances" : 1000000,
      "properties" : [
        {
          "name": "country",
          "dataType": "String",
          "generator": {
            "name":"PersonCountry",
            "dependencies":[],
            "initParameters" : []}
        },
        {
          "name": "sex",
          "dataType": "String",
          "generator": {
            "name":"PersonSex",
            "dependencies":[],
            "initParameters" : []}
        },
        {
          "name": "name",
          "dataType": "String",
          "generator": {
            "name":"PersonName",
            "dependencies":["country","sex"],
            "initParameters" : []}
        },
        {
          "name": "creationDate",
          "dataType": "Timestamp",
          "generator": {
            "name":"PersonCreationDate",
            "dependencies":[],
            "initParameters" : []}
        }
      ]
    }
  ],
```



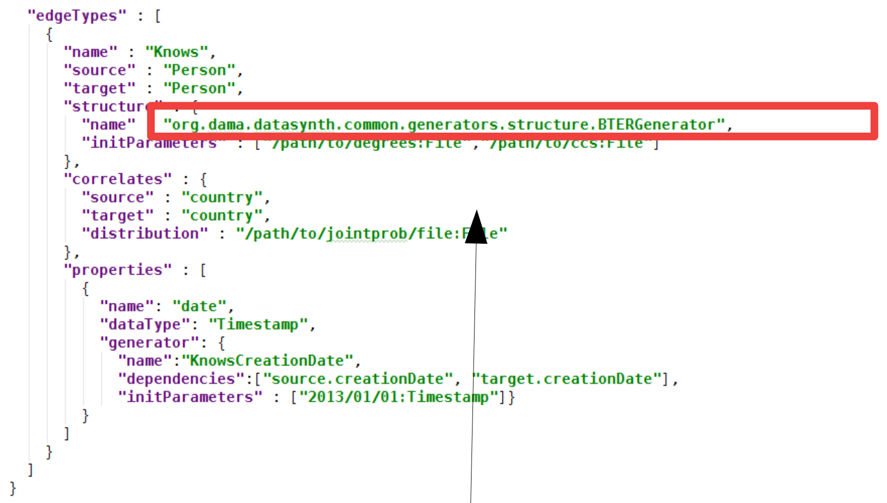
```
  "edgeTypes" : [
    {
      "name" : "Knows",
      "source" : "Person",
      "target" : "Person",
      "structure" : {
        "name" : "org.dama.datasynth.common.generators.structure.BTERGenerator",
        "initParameters" : ["/path/to/degrees:File","/path/to/ccs:File"]
      },
      "correlates" : {
        "source" : "country",
        "target" : "country",
        "distribution" : "/path/to/jointprob/file:File"
      },
      "properties" : [
        {
          "name": "date",
          "dataType": "Timestamp",
          "generator": {
            "name":"KnowsCreationDate",
            "dependencies":["source.creationDate", "target.creationDate"],
            "initParameters" : ["2013/01/01:Timestamp"]}
        }
      ]
    }
  ]
}
```

Dependencies

Frontend/Backend interface



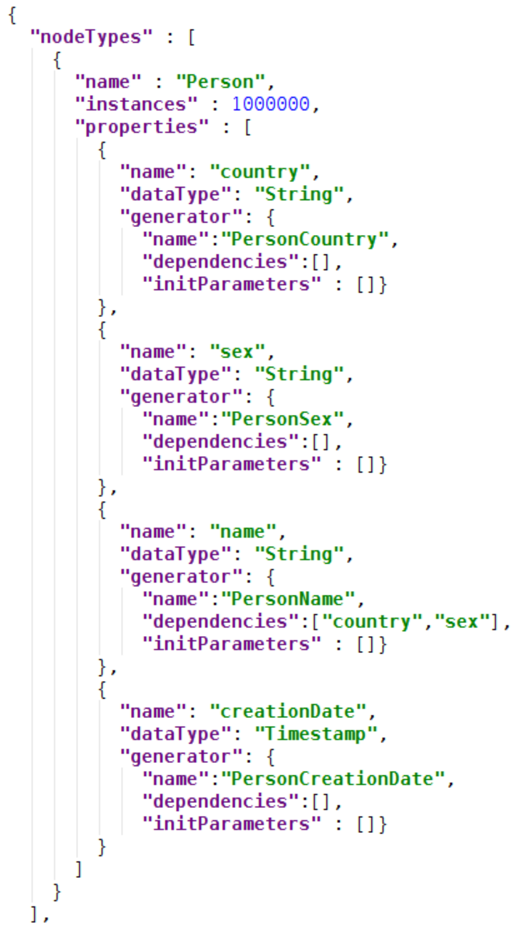
```
{
  "nodeTypes" : [
    {
      "name" : "Person",
      "instances" : 1000000,
      "properties" : [
        {
          "name": "country",
          "dataType": "String",
          "generator": {
            "name":"PersonCountry",
            "dependencies":[],
            "initParameters" : []}
        },
        {
          "name": "sex",
          "dataType": "String",
          "generator": {
            "name":"PersonSex",
            "dependencies":[],
            "initParameters" : []}
        },
        {
          "name": "name",
          "dataType": "String",
          "generator": {
            "name":"PersonName",
            "dependencies":["country","sex"],
            "initParameters" : []}
        },
        {
          "name": "creationDate",
          "dataType": "Timestamp",
          "generator": {
            "name":"PersonCreationDate",
            "dependencies":[],
            "initParameters" : []}
        }
      ]
    }
  ],
```



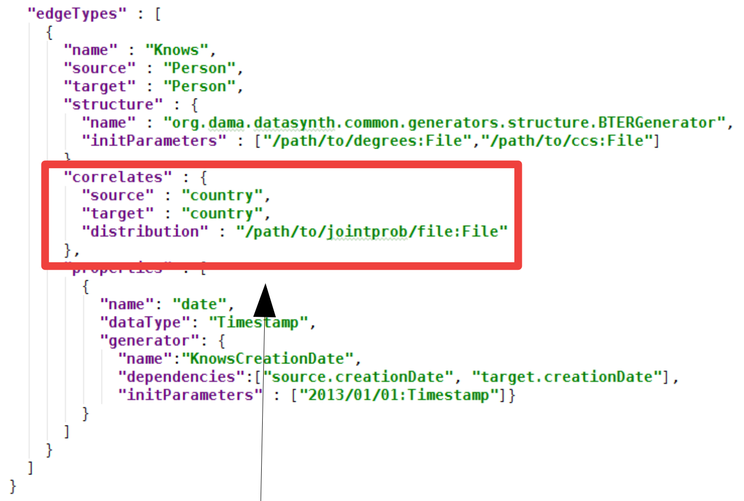
```
  "edgeTypes" : [
    {
      "name" : "Knows",
      "source" : "Person",
      "target" : "Person",
      "structure" : {
        "name" : "org.dama.datasynth.common.generators.structure.BTERGenerator",
        "initParameters" : ["/path/to/degrees:File", "/path/to/ccs:File"]
      },
      "correlates" : {
        "source" : "country",
        "target" : "country",
        "distribution" : "/path/to/jointprob/file:File"
      },
      "properties" : [
        {
          "name": "date",
          "dataType": "Timestamp",
          "generator": {
            "name":"KnowsCreationDate",
            "dependencies":["source.creationDate", "target.creationDate"],
            "initParameters" : ["2013/01/01:Timestamp"]}
        }
      ]
    }
  ]
}
```

Structure Generator

Frontend/Backend interface



```
{
  "nodeTypes" : [
    {
      "name" : "Person",
      "instances" : 1000000,
      "properties" : [
        {
          "name": "country",
          "dataType": "String",
          "generator": {
            "name":"PersonCountry",
            "dependencies":[],
            "initParameters" : []}
        },
        {
          "name": "sex",
          "dataType": "String",
          "generator": {
            "name":"PersonSex",
            "dependencies":[],
            "initParameters" : []}
        },
        {
          "name": "name",
          "dataType": "String",
          "generator": {
            "name":"PersonName",
            "dependencies":["country","sex"],
            "initParameters" : []}
        },
        {
          "name": "creationDate",
          "dataType": "Timestamp",
          "generator": {
            "name":"PersonCreationDate",
            "dependencies":[],
            "initParameters" : []}
        }
      ]
    }
  ],
```



```
  "edgeTypes" : [
    {
      "name" : "Knows",
      "source" : "Person",
      "target" : "Person",
      "structure" : {
        "name" : "org.dama.datasynth.common.generators.structure.BTERGenerator",
        "initParameters" : ["/path/to/degrees:File","/path/to/ccs:File"]
      }
      "correlates" : {
        "source" : "country",
        "target" : "country",
        "distribution" : "/path/to/jointprob/file:File"
      },
      "properties" : [
        {
          "name": "date",
          "dataType": "Timestamp",
          "generator": {
            "name":"KnowsCreationDate",
            "dependencies":["source.creationDate", "target.creationDate"],
            "initParameters" : ["2013/01/01:Timestamp"]}
        }
      ]
    }
  ]
}
```

Structure Generator

Backend

- Our goal is to generate one table per <node,property> pair, <edge, property> pair, and edge type.
- We have property tables and edge tables

ID	Property Value
0	
1	
2	
...	

Property Table

ID	tail	head
0		
1		
2		
...		

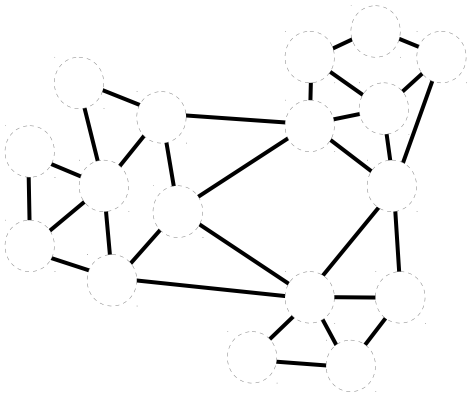
Edge Table

Backend – Approach

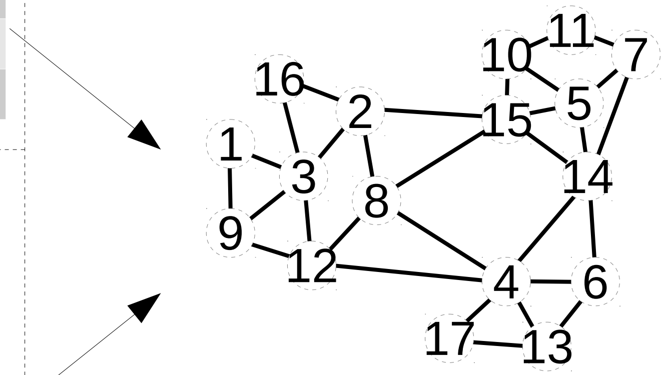
Id	PersonCountry
0	China
1	India
2	France
...	...
17	Germany

Id	PersonName
1	Bo
2	Sidharta
3	Julie
...	...
17	Annie

structure generation



Matching preserving given joint probability distributions



e.g. P(China,China) ≈ 0.2

ID	KnowsCreationDate
0	2011/08/08
1	2010/07/15
2	2012/06/30
...	...
30	2010/11/12

TIME

Backend – Generating Properties

- Property tables are created by mapping the corresponding run method over a range(0,n-1)

Backend – Generating Properties

- Property tables are created by mapping the corresponding run method over a range(0,n-1)

ID
0
1
2
...
n-1

Backend – Generating Properties

- Property tables are created by mapping the corresponding run method over a range(0,n-1)

ID
0
1
2
...
n-1

$\longrightarrow$ PersonCountry.run(**0**, random(**0**))

Backend – Generating Properties

- Property tables are created by mapping the corresponding run method over a range(0,n-1)

ID
0
1
2
...
n-1

→ PersonCountry.run(**0**, random(**0**)) →

ID	PersonCountry
0	China

Backend – Generating Properties

- Property tables are created by mapping the corresponding run method over a range(0,n-1)

ID
0
1
2
...
n-1

$\longrightarrow$ PersonCountry.run(**1**, random(**1**)) $\longrightarrow$

ID	PersonCountry
0	China
1	India

Backend – Generating Properties

- Property tables are created by mapping the corresponding run method over a range(0,n-1)

ID
0
1
2
...
n-1

$\longrightarrow$ PersonCountry.run(**2**, random(**2**)) $\longrightarrow$

ID	PersonCountry
0	China
1	India
2	France

Backend – Generating Properties

- Property tables are created by mapping the corresponding run method over a range(0,n-1)

ID
0
1
2
...
n-1

$\longrightarrow$ PersonCountry.run(**n-1**, random(**n-1**)) $\longrightarrow$

ID	PersonCountry
0	China
1	India
2	France
...	...
n-1	Germany

Backend – Generating Properties

- What if there are dependencies?
  - e.g PersonName has a dependency on PersonCountry and PersonSex

Backend – Generating Properties

- What if there are dependencies?
  - e.g PersonName has a dependency on PersonCountry and PersonSex

ID	PersonCountry
0	China
1	India
2	France
...	...
n-1	Germany

$\bowtie$

ID	PersonSex
0	M
1	M
2	F
...	...
n-1	F

=

ID	PersonCountry	PersonSex
0	China	M
1	India	M
2	France	F
...		...
n-1	Germany	F

Backend – Generating Properties

- What if there are dependencies?
  - e.g PersonName has a dependency on PersonCountry and PersonSex

ID	PersonCountry	PersonSex
0	China	M
1	India	M
2	France	F
...		...
n-1	Germany	F

Backend – Generating Properties

- What if there are dependencies?
  - e.g PersonName has a dependency on PersonCountry and PersonSex

ID	PersonCountry	PersonSex
0	**China**	**M**
1	India	M
2	France	F
...		...
n-1	Germany	F

PersonSex.run(**0**, random(**0**),”**China**”, “**M**”)

ID	PersonName
0	Bo

Backend – Generating Properties

- What if there are dependencies?
  - e.g PersonName has a dependency on PersonCountry and PersonSex

ID	PersonCountry	PersonSex
0	China	M
1	**India**	M
2	France	F
...		...
n-1	Germany	F

PersonSex.run(**1**, random(**1**),"**India**", "**M**")

ID	PersonName
0	Bo
1	Sidharta

Backend – Generating Properties

- What if there are dependencies?
  - e.g PersonName has a dependency on PersonCountry and PersonSex

ID	PersonCountry	PersonSex
0	China	M
1	India	M
2	**France**	**F**
...		...
n-1	Germany	F

PersonSex.run(**2**, random(**2**),"**France**", "**M**")

ID	PersonName
0	Bo
1	Sidharta
2	Julie

Backend – Generating Properties

- What if there are dependencies?
  - e.g PersonName has a dependency on PersonCountry and PersonSex

ID	PersonCountry	PersonSex
0	China	M
1	India	M
2	France	F
...		...
n-1	**Germany**	**F**

PersonSex.run(**n-1**, random(**n-1**), ”**Germany**”, “F”)

ID	PersonName
0	Bo
1	Sidharta
2	Julie
...	...
n-1	Annie

Backend – Generating Properties

- What if there are dependencies?
  - e.g PersonName has a dependency on PersonCountry and PersonSex

```
PersonSex.run(0, random(0), “China”, “M”)
```

Backend – Generating Properties

- What if there are dependencies?
  - e.g PersonName has a dependency on PersonCountry and PersonSex

PersonSex.run(0, random(0), “China”, “M”)

PersonSex.run(0, random(0), PersonCountry.run(0,random(0)), “M”)

Backend – Generating Properties

- What if there are dependencies?
  - e.g PersonName has a dependency on PersonCountry and PersonSex

```
PersonSex.run(0, random(0), “China”, “M”)

PersonSex.run(0, random(0), PersonCountry.run(0,random(0)), “M”)

PersonSex.run(0, random(0), PersonCountry.run(0,random(0)), PersonSex.run(0,random(0))
```

Backend – Generating Properties

- What if there are dependencies?
  - e.g PersonName has a dependency on PersonCountry and PersonSex

```
PersonSex.run(0, random(0), “China”, “M”)

PersonSex.run(0, random(0), PersonCountry.run(0,random(0)), “M”)

PersonSex.run(0, random(0), PersonCountry.run(0,random(0)), PersonSex.run(0,random(0))
```

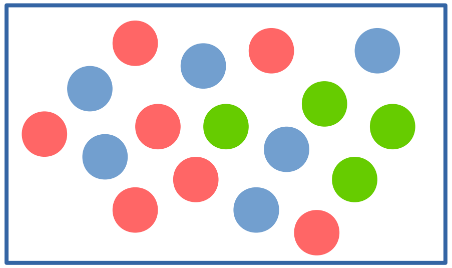
- We avoid performing an expensive join
- The generation of a property value depends only on the ID!

Backend – Generating Edges

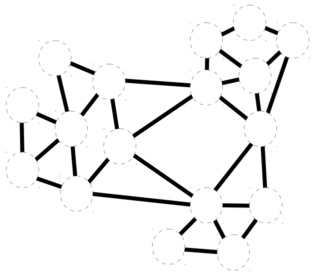
- The generation of edges is delegated to Structure Generators
- Just calling a Hadoop/Spark based library implementing the method
- Produces an HDFS file with the edge table
- If necessary, a STRUCTURE – PROPERTY matching is executed

Backend – Generating Edges

Input

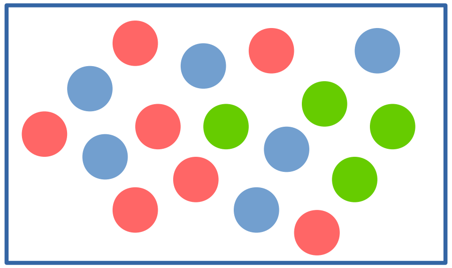


P(X,Y)	🟦	🟥	🟩
🟦	0.3	0.067	0.067
🟥	0.067	0.33	0.067
🟩	0.067	0.067	0.17



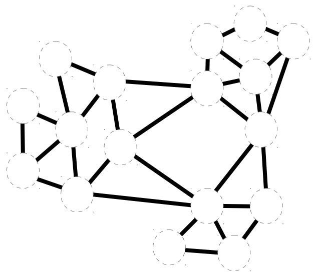
Backend – Generating Edges

Input



P(X,Y)

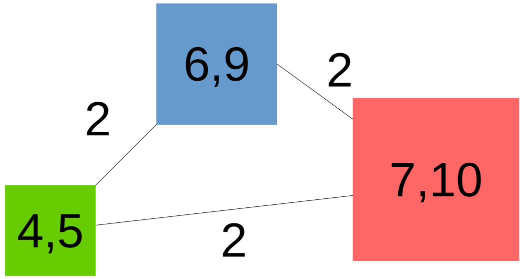
	🟦	🟥	🟩
🟦	0.3	0.067	0.067
🟥	0.067	0.33	0.067
🟩	0.067	0.067	0.17



Block Model

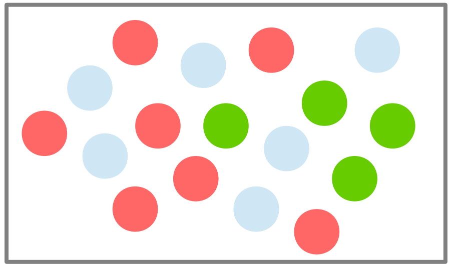
🟦	6
🟥	7
🟩	4

	🟦	🟥	🟩
🟦	9	2	2
🟥	2	10	2
🟩	2	2	5



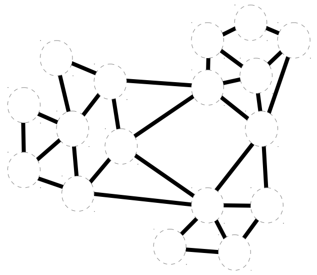
Backend – Generating Edges

Input



P(X,Y)

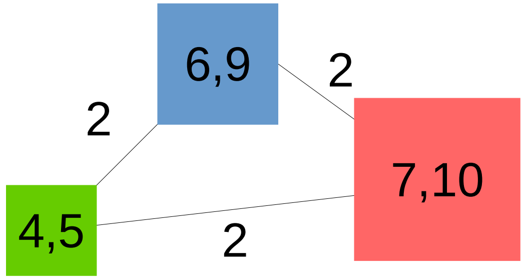
	🟦	🟥	🟩
🟦	0.3	0.067	0.067
🟥	0.067	0.33	0.067
🟩	0.067	0.067	0.17

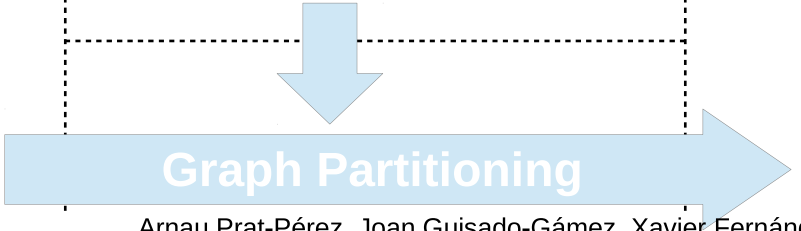


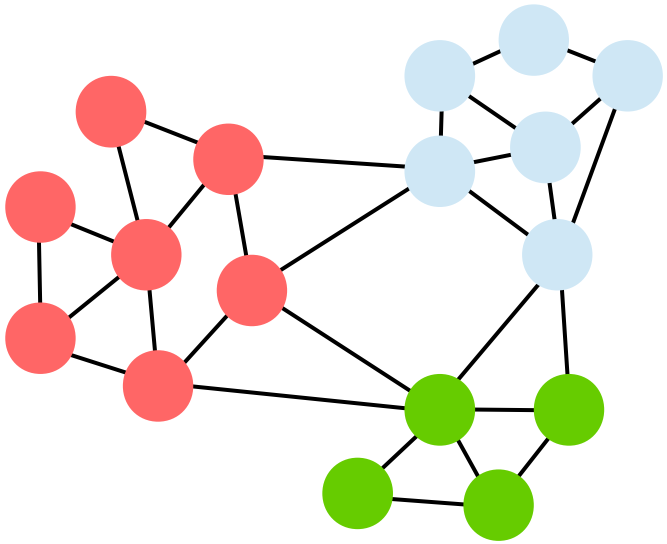
Block Model

🟦	6
🟥	7
🟩	4

	🟦	🟥	🟩
🟦	9	2	2
🟥	2	10	2
🟩	2	2	5







Arnau Prat-Pérez, Joan Guisado-Gámez, Xavier Fernández Salas, Petr Koupy, Siegfried Depner, Davide Basilio Bartolini: Towards a property graph generator for benchmarking. GRADES@SIGMOD/PODS2017: 6:1-6:6

Backend – Generating Edges

ID	tail	head
0	0	2
1	0	3
2	1	4
...	...	...
m-1	999998	999999

Backend – Generating Edges

ID	tail	head
0	**0**	**2**
1	0	3
2	1	4
...	...	...
m-1	999998	999999

```
PersonSex.run(0, random(0),
                    PersonCreationDate.run(0,random(0)),
                    PersonCreationDate.run(2,random(2)))
```

ID	PersonCreationDate
0	2013/08/08

Backend – Generating Edges

ID	tail	head
0	0	2
1	**0**	**3**
2	1	4
...	...	...
m-1	999998	999999

```
PersonSex.run(1, random(1),
              PersonCreationDate.run(0,random(0)),
              PersonCreationDate.run(3,random(3)))
```

ID	PersonCreationDate
0	2011/08/08
1	2010/07/15

Backend – Generating Edges

ID	tail	head
0	0	2
1	0	3
2	1	4
...	...	...
m-1	999998	999999

```
PersonSex.run(2, random(2),
                    PersonCreationDate.run(1,random(1)),
                    PersonCreationDate.run(4,random(4)))
```

ID	PersonCreationDate
0	2011/08/08
1	2010/07/15
2	2012/06/30

Backend – Generating Edges

ID	tail	head
0	0	2
1	0	3
2	1	4
...	...	...
m-1	**999998**	**999999**

ID	PersonCreationDate
0	2011/08/08
1	2010/07/15
2	2012/06/30
...	...
m-1	2010/11/12

```
PersonSex.run(m-1, random(m-1),
              PersonCreationDate.run(999998,random(999998)),
              PersonCreationDate.run(999999,random(999999)))
```

Conclusions

- We want to make property graph generation easier
- We are accepting contributions!