ETL with json-autotype and other Haskell tools

Michał J. Gajda mjgajda@migamake.com

Migamake Pte Ltd

Aug 22 2019

Extract-Transform-Load

1. Discover the data sources.

Extract-Transform-Load

1. Discover the data sources.
2. Extract data from input:

Extract-Transform-Load

1. Discover the data sources.
2. Extract data from input:

- json-autotype

Extract-Transform-Load

1. Discover the data sources.
2. Extract data from input:

- json-autotype
- DataFrames

Extract-Transform-Load

1. Discover the data sources.
2. Extract data from input:

- json-autotype
- DataFrames
- ...

Extract-Transform-Load

1. Discover the data sources.
2. Extract data from input:

- json-autotype
- DataFrames
- ...

3. Transform:

Extract-Transform-Load

1. Discover the data sources.
2. Extract data from input:

- json-autotype
- DataFrames
- ...

3. Transform:

- extract key information

Extract-Transform-Load

1. Discover the data sources.
2. Extract data from input:

- json-autotype
- DataFrames
- ...

3. Transform:

- extract key information
- correlate information from different sources

Extract-Transform-Load

1. Discover the data sources.
2. Extract data from input:

- json-autotype
- DataFrames
- ...

3. Transform:

- extract key information
- correlate information from different sources

4. Load into database

Dataset

You can pick any JSON file for this tutorial, but I give you some hints for fun:

First schema

1. Create your data project with `stack`

```
stack new my-data-project
```

```
stack install json-autotype
```

In `package.yaml`:

```
build-tool: json-autotype
dependencies:
- json-alt
- aeson
```

First schema

1. Create your data project with `stack`

```
stack new my-data-project
```

2. Add build tool for generating schema:

```
stack install json-autotype
```

In `package.yaml`:

```
build-tool: json-autotype
dependencies:
- json-alt
- aeson
```

Look into schema

1. “‘sh json-autotype input/data.json -o MyFormat.hs

2.
```json
```

Look into schema

1. “‘sh json-autotype input/data.json -o MyFormat.hs

2.
```json
```

3.

Look at the schema - contd

-

```
stack ghci
```

```
import MyFormat
:info TopLevel
```

```
import Data.Aeson
:info TopLevel
:info FromJSON
```

Look at the schema - contd

-

```
stack ghci
```

-

```
import MyFormat
:info TopLevel


import Data.Aeson
:info TopLevel
:info FromJSON
```

Look at the schema - contd

- ```
  stack ghci
  ```
- ```
  import MyFormat
  :info TopLevel
  ```
- ```
  import Data.Aeson
  :info TopLevel
  :info FromJSON
  ```
  ```

Note: building with preprocessors

If you want to embed `.json` *example as template for your data declaration in* `.cabal` *package:* MyFormat.lhs

```
{-# GHC_OPTIONS -pgmL json-autotype --preprocessor #-}
```

Summarize data

```
module Main where

import MyFormat

main = do
  result <- parse
  let extracted = concatMap extractCol result
      avg = sum extracted/length extracted
  print $ "Average: " ++ show avg
```

Better summarization

```
dependencies:
- lfold

import qualified Control.Foldl as L

main = do
  input <- parse "inputs/data.json"
  let average = (/) <$> L.sum <*> L.genericLength
  print <$> lfold average input
```

Reading CSV summaries

```
dependencies:
  - Frames
```

```haskell
{-# LANGUAGE TemplateHaskell #-}
import Frames.TH
import Data.Vinyl.Derived
import Frames
import Frames.Categorical (declareCategorical)
import qualified Pipes.Prelude as P

tableTypes' (rowGenCat "input/data.csv") { rowTypeName = "S

main = do
  print <$> runSafeEffect loader
  where
     loader :: MonadSafe m => Producer Small m ()
     loader = readTable "input/data.csv"
```

Charting