Loading Data into Relations

Overview

Introduce relations, the basic structure to hold data on which operations are performed

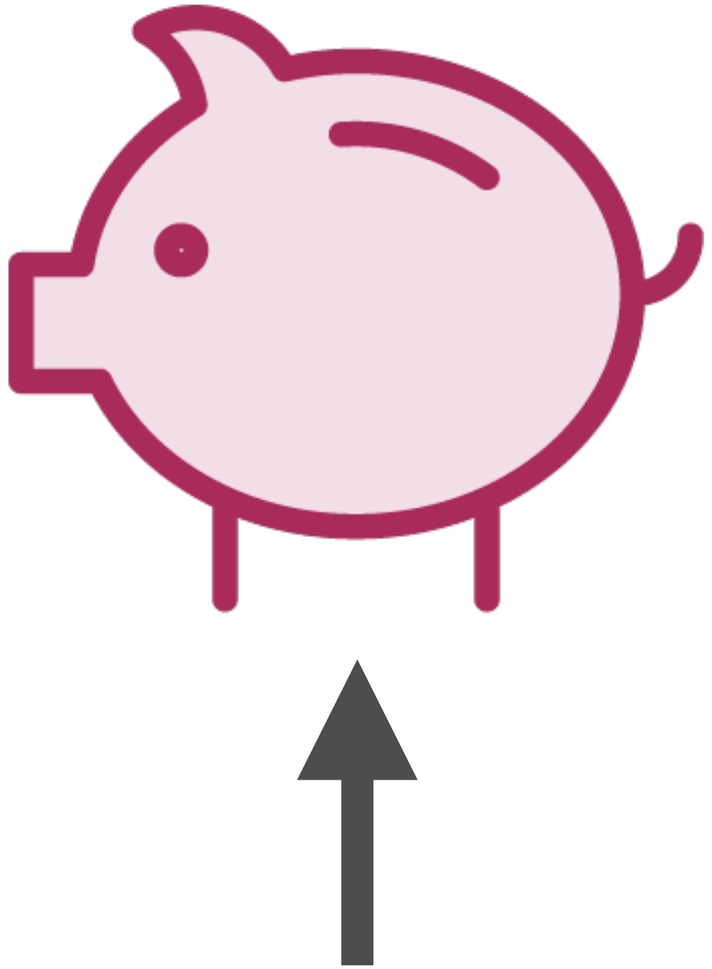
Understand and implement the load, store and dump commands

Know the scalar and complex data types that Pig supports

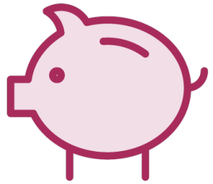
Introduce Pig commands to create complex data types

Basic Building Blocks: Relations

Load data into Pig

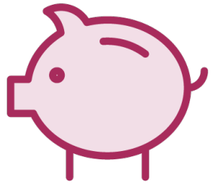


Load data into Pig



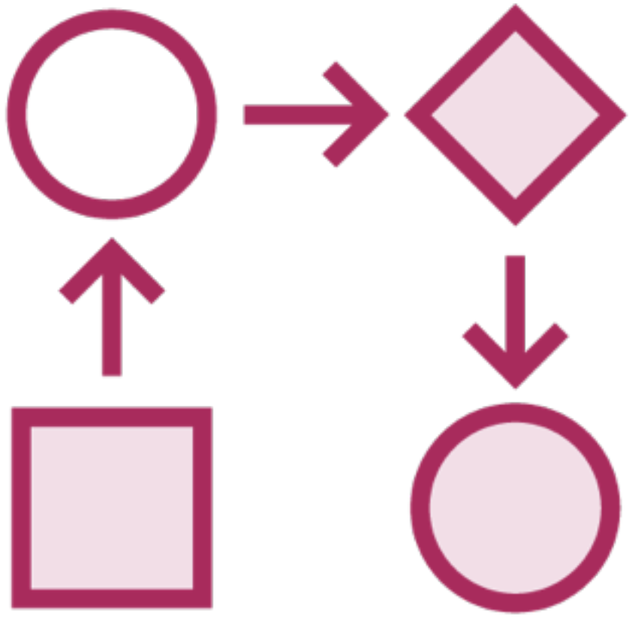
Data is stored in a relation

Load data into Pig

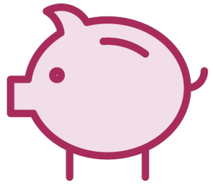


Data is stored in a relation

Transform and update the data

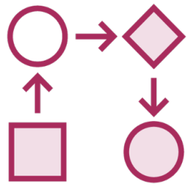


Load data into Pig



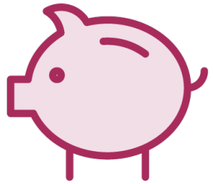
Data is stored in a relation

Transform and update the data



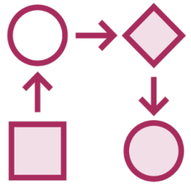
Store the data to file or display it to screen

Load data into Pig



Data is stored in a relation

Transform and update the data

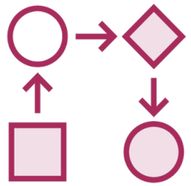


Store the data to file or display it to screen

Load data into Pig

Data is stored in a relation

Transform and update the data



Store the data to file or display it to screen

What is a relation and how do transformations affect it?

Relations

A dataset with a name i.e. like variables in Java, Python

May or may not have a schema associated with it

Immutable, updates to a relation creates a new relation

Relations exist for the duration of a single Pig session

Immutable Relations

Operations transform data and create new relations

Immutable Relations

Not evaluated till we display the results on screen or store them to a file

Immutable Relations

Lazy evaluation

Immutable Relations

On edits the original relations remain unchanged i.e. immutable

Pig as a Data Flow Language

relation_1 = load data from file into Pig

relation_2 = pig latin commands to transform relation_1

relation_3 = pig latin commands to transform relation_2

relation_4 = pig latin commands to transform relation_3

store relation_4 to file or display results to screen

Pig as a Data Flow Language

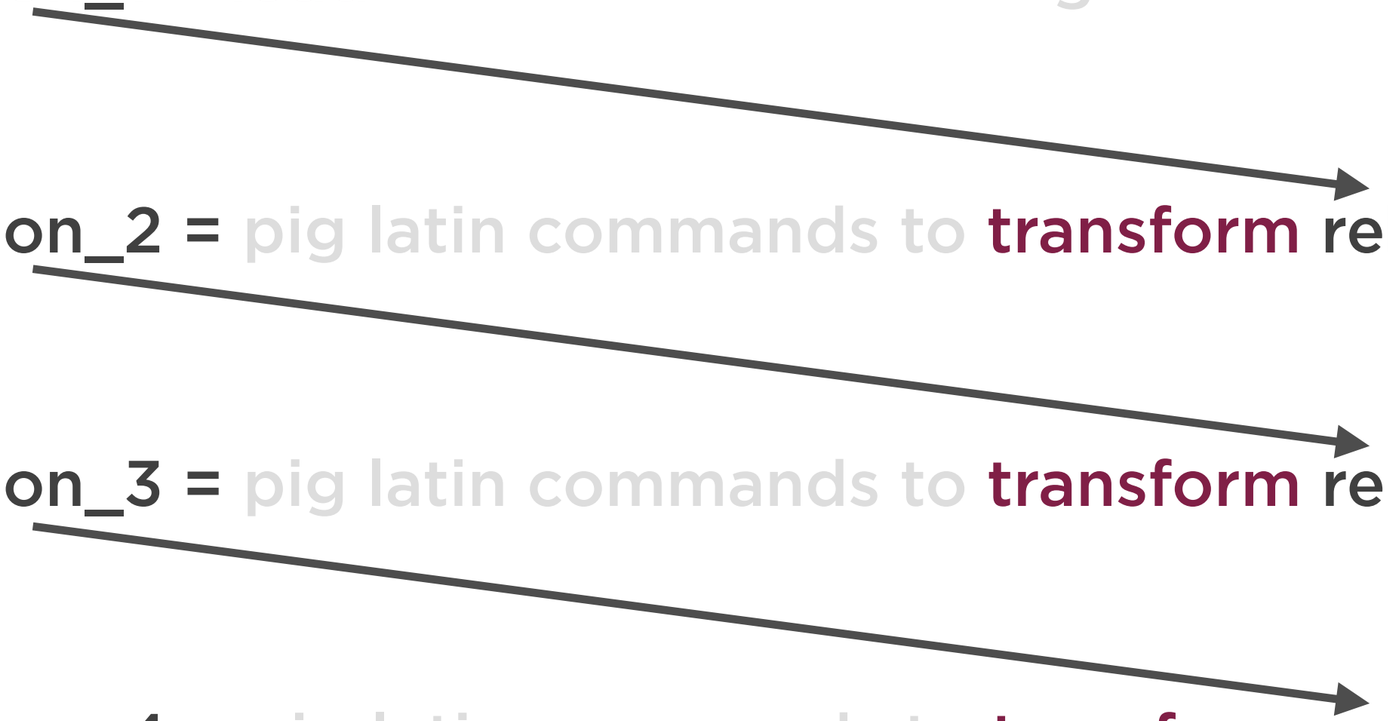
relation_1 = load data from file into Pig

relation_2 = pig latin commands to transform relation_1

relation_3 = pig latin commands to transform relation_2

relation_4 = pig latin commands to transform relation_3

store relation_4 to file or display results to screen



Demo

Load data into Pig using the PigStorage() function

Load from files as well as directories

Demo

Specify a schema for the loaded data

Demo

Store data into files in a directory

Case-sensitivity in Pig

What Is Case-sensitive and What Is Not?

Case-sensitive

Relation names

Field names within relations

Function names such as PigStorage(), SUM(), COUNT()

Case-insensitive

Keywords in Pig such as load, store, foreach, generate, group by, order by, dump

Data Types in Pig

Pig Data Types

Scalar

Primitive types to represent a single entity or field

Complex

Collection types to represent a group of entities

Pig Data Types

Scalar

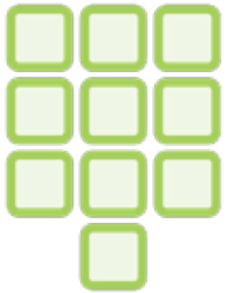
Primitive types to represent a single entity or field

Complex

Collection types to represent a group of entities

Scalar Data Types

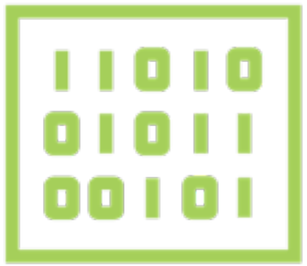
Boolean



Numeric

String

Date/Time

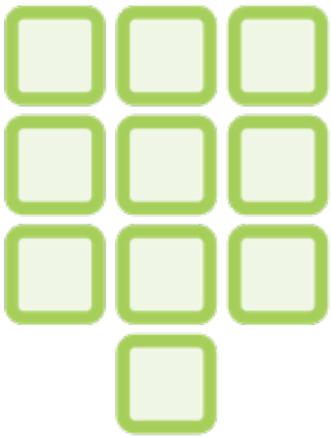


Bytes

Boolean

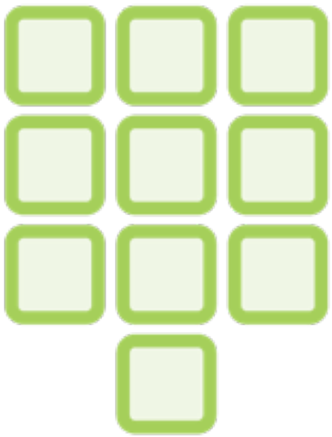
true or false

yes/no questions



Numeric

Integers or Decimals



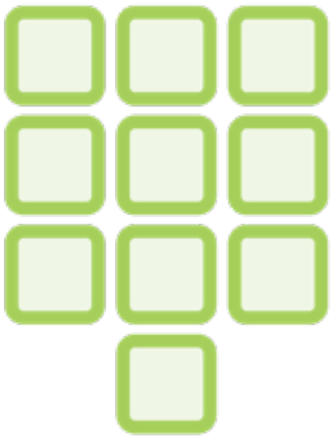
Numeric

Integers

Int: 4 bytes, range -2^{31} to $2^{31} - 1$

Long: 8 bytes, range -2^{63} to $2^{63} - 1$

Decimals



Numeric

Float: 4 bytes

Double: 8 bytes

String

String

Chararray: **Unbounded, variable length character string**

Date/Time

Date/Time

Datetime: Time specified in the date, hour, minute, seconds, milliseconds, nanoseconds format

Bytes

Bytes

Bytearray: A blob used to represent any kind of data

Default type when none of the other types are specified

Pig Data Types

Scalar

Primitive types to represent a single entity or field

Complex

Collection types to represent a group of entities

Pig Data Types

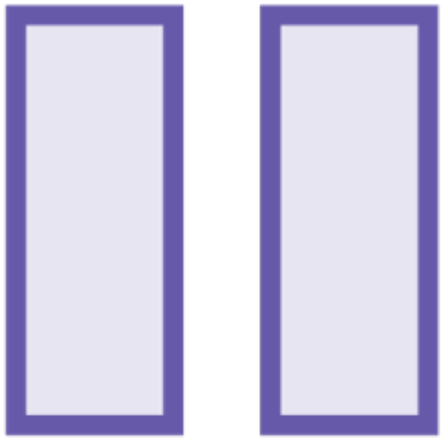
Scalar

Primitive types to represent a single entity or field

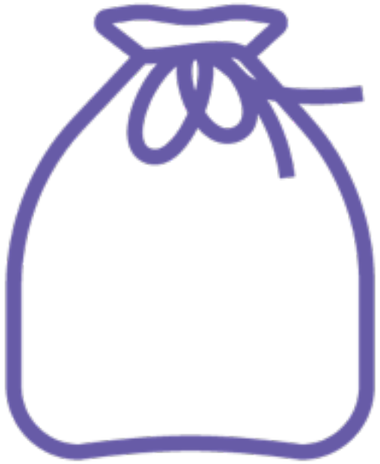
Complex

Collection types to represent a group of entities

Complex Data Types



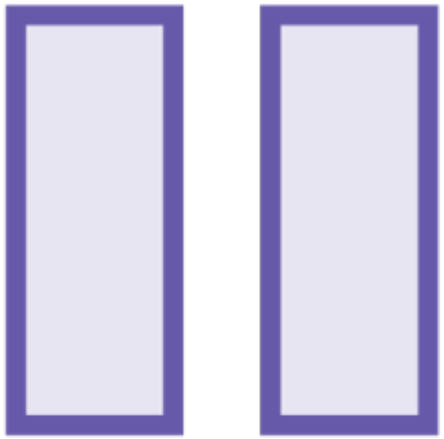
Tuple



Bag

Map

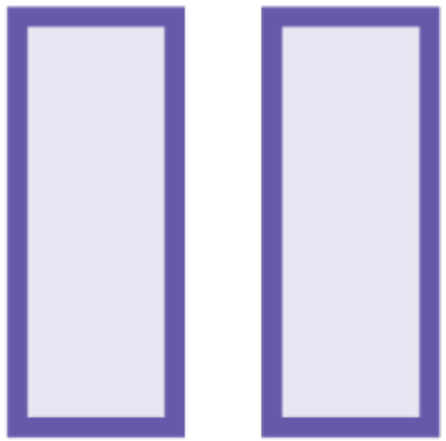
Complex Data Types



Tuple

Bag

Map

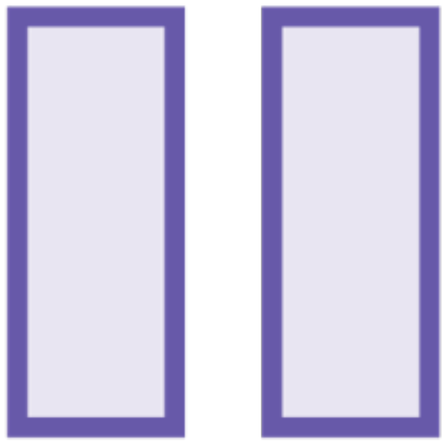


Tuple

Tuple

```
(134, “John”, “Smith”, “HR”, 9)
```

An ordered collection of fields

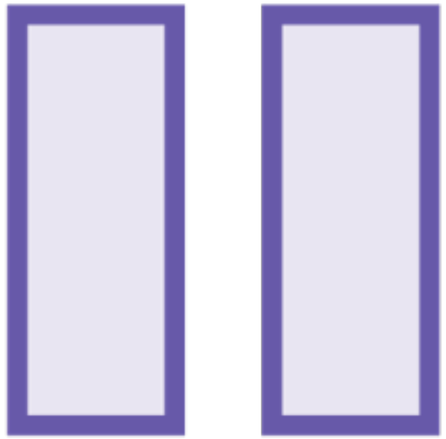


Tuple

Tuple

```
(134, “John”, “Smith”, “HR”, 9)
```

Enclosed within parenthesis



Tuple

Tuple

```
(134, “John”, “Smith”, “HR”, 9)
```

Each field has its own data type

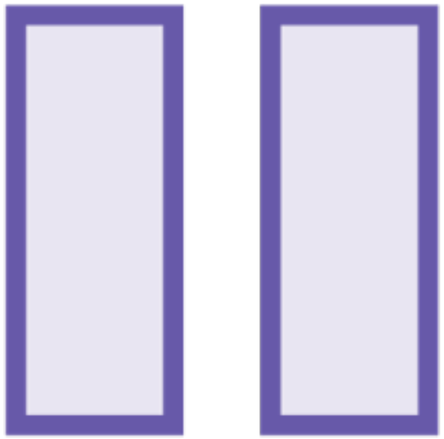
The data type is optional, they default to bytearray

Demo

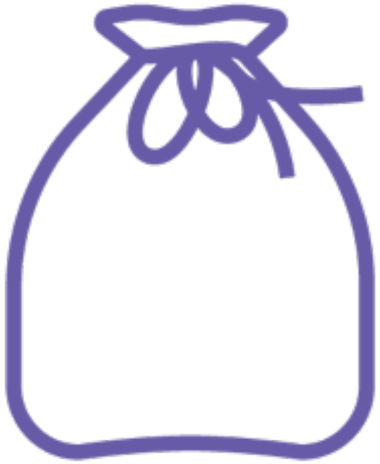
Specify a tuple as a part of the schema definition for a relation

Use the TOTUPLE() function to generate a tuple in a relation

Complex Data Types



Tuple

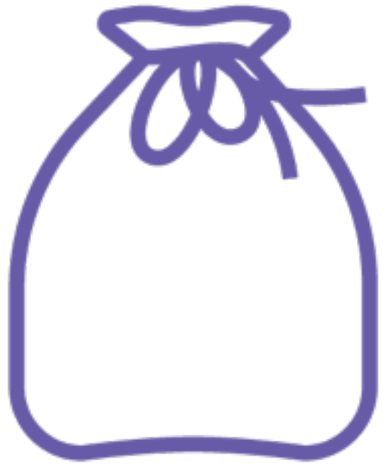


Bag

Map

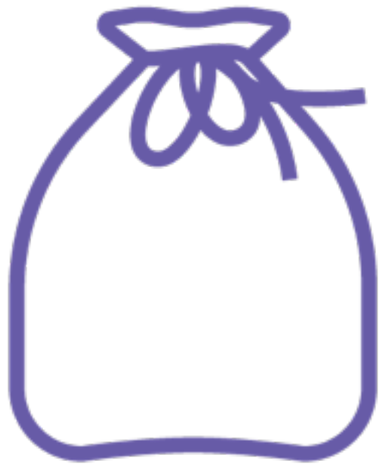
Complex Data Types

Tuple



Bag

Map



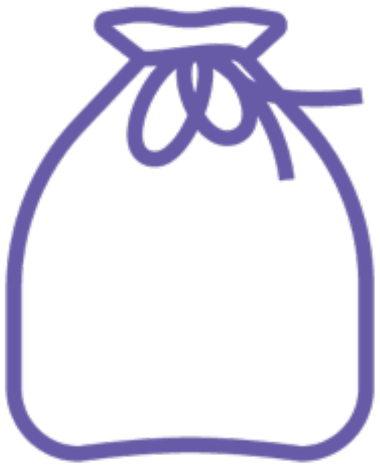
Bag

Bag

```
{(134, “John”, “Smith”, “HR”, 9)
(238, “Jill”, “Paul”, “Engg”, 8)
(561, “Nina”, “Tang”, “Engg”, 9)}
```

An unordered collection of tuples

Duplicates may be present

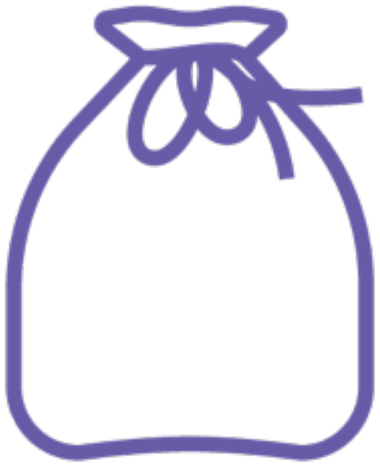


Bag

Bag

```
{(134, “John”, “Smith”, “HR”, 9)
(238, “Jill”, “Paul”, “Engg”, 8)
(561, “Nina”, “Tang”, “Engg”, 9)}
```

Enclosed within curly braces



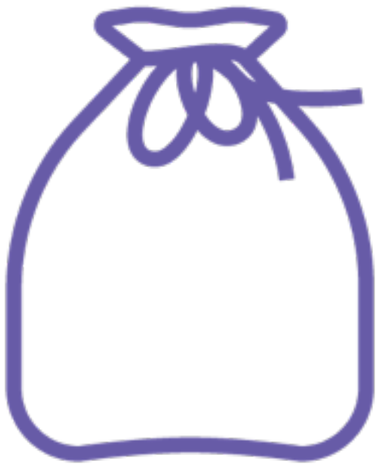
Bag

Bag

```
{(134, “John”, “Smith”, “HR”, 9)
(238, “Jill”, “Paul”, “Engg”, 8)
(561, “Nina”, “Tang”, “Engg”, 9)}
```

Each tuple can have a different number and type of fields

Absent fields will be nulls when accessed

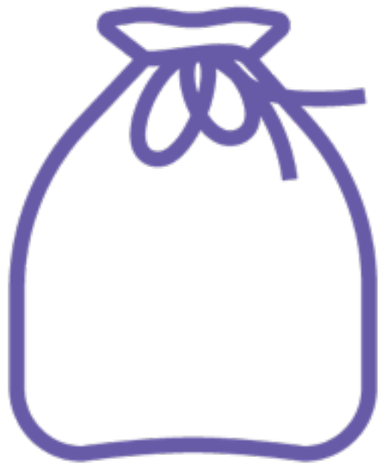


Bag

Bag

```
{(134, “John”, “Smith”, “HR”, 9)
(238, “Jill”, “Paul”, “Engg”, 8)
(561, “Nina”, “Tang”, “Engg”, 9)}
```

A relation is a bag of tuples, the outer bag



Bag

Bag

```
{(134, “John”, “Smith”, “HR”, 9)
(238, “Jill”, “Paul”, “Engg”, 8)
(561, “Nina”, “Tang”, “Engg”, 9)}
```

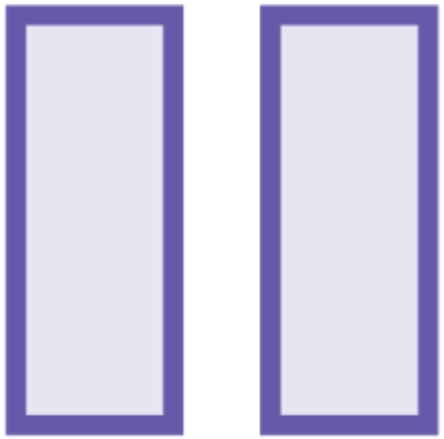
A bag within the fields of the relation is an inner bag

Demo

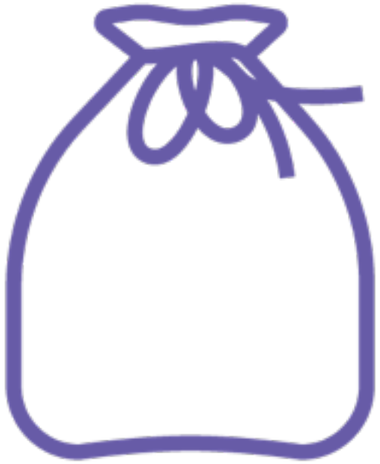
Specify a bag as a part of the schema definition for a relation

Use the TOBAG() function to generate a bag in a relation

Complex Data Types



Tuple



Bag

Map

Complex Data Types

Tuple

Bag

Map

Map

Map

[John#HR

Jill#Engg

Nina#Engg]

Key-value pairs, values can be looked up by key

Keys are unique

Map

Map

```
[John#HR
Jill#Engg
Nina#Engg]
```

Map key-values are specified in square brackets

Map

Map

```
[John#HR

Jill#Engg

Nina#Engg]
```

Keys are always of type chararray

Map

Map

```
[John#HR
Jill#Engg
Nina#Engg]
```

The values can be of any data type

Map

Map

```
[John#HR

Jill#Engg

Nina#Engg]
```

The # is the delimiter when specifying maps in files

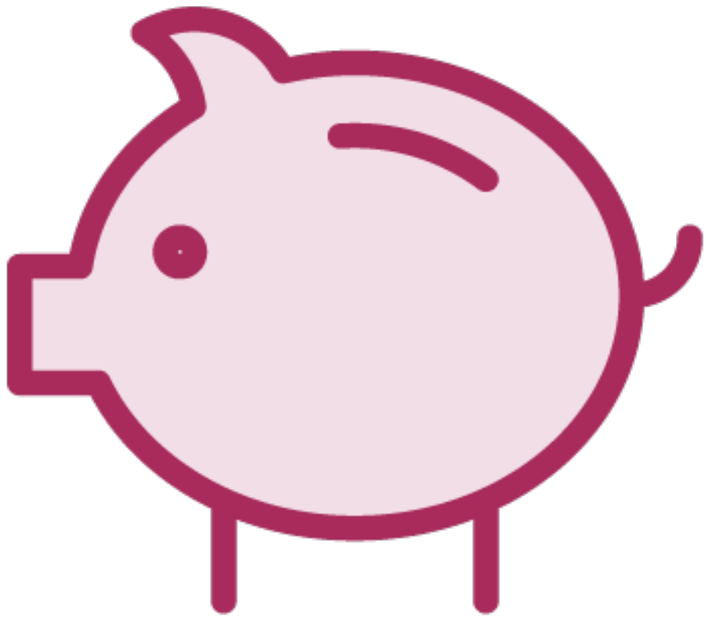
Demo

Specify a map as a part of the schema definition for a relation

Use the TOMAP() function to generate a map in a relation

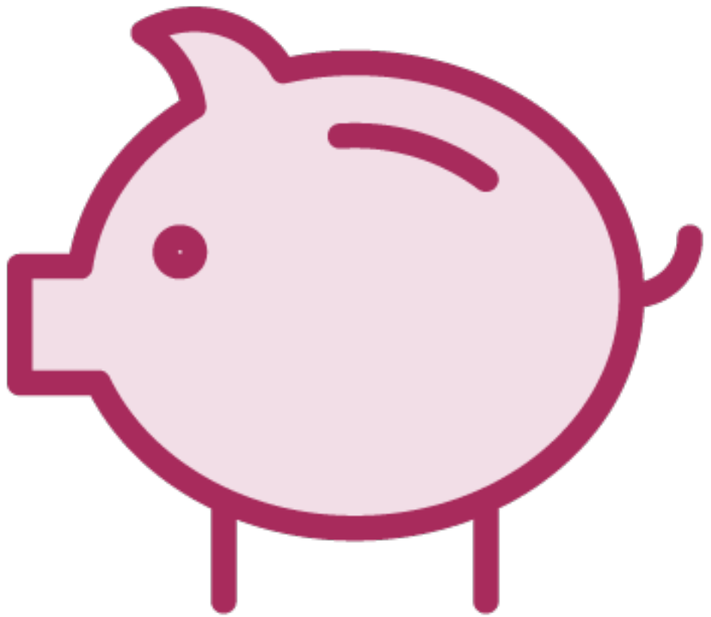
Partial Schema Specification

Pig Is Omnivorous



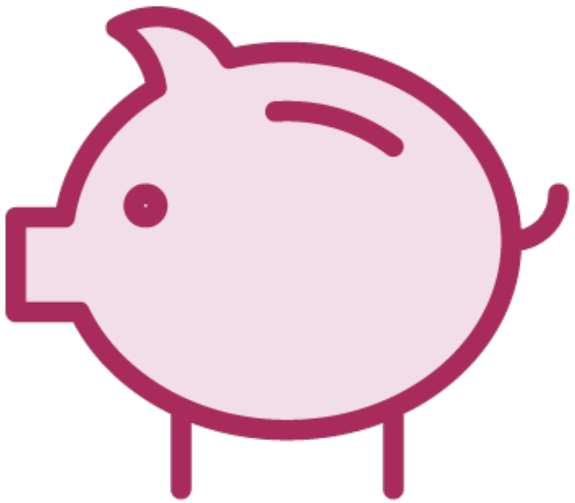
It consumes any kind of data

Pig Is Omnivorous



If schema is not known, it will still accept data, guessing along the way

Pig and Schema Definitions



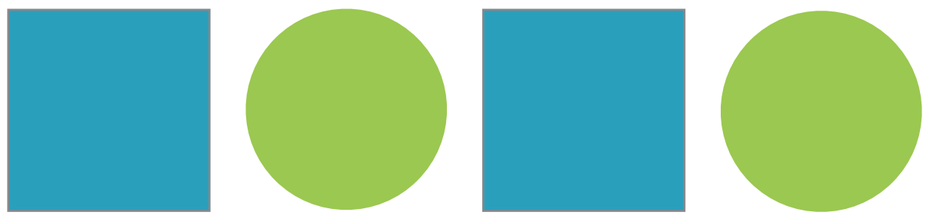
No schema definition: Pig will assume every field to be of type bytearray

Partial schema definition: Can leave out data types for fields

Complete schema definition: All fields and data types known

Casting and Conversion

Schema defined for the relation

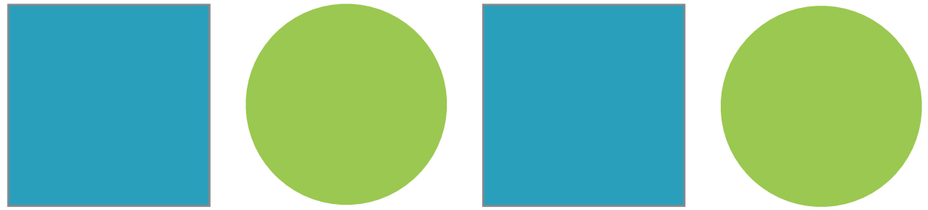


Schema found by the Pig loader

Casting and Conversion

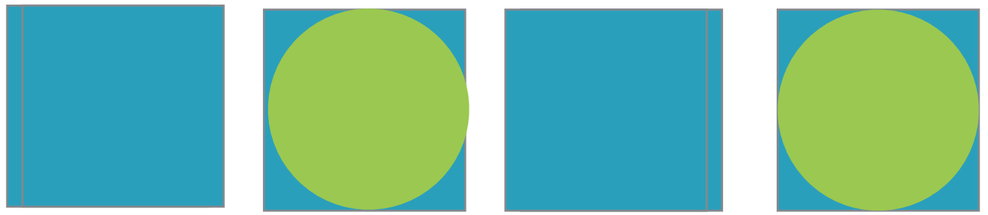
Schema defined

Schema found



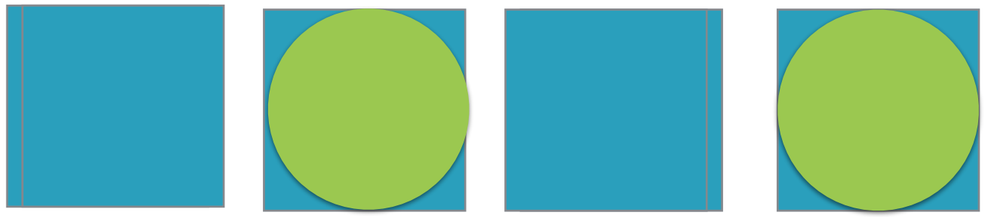
Pig will try and convert the data it found by casting it to the specified schema

Casting and Conversion



Pig will try and convert the data it found by casting it to the specified schema

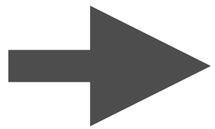
Casting and Conversion



Not all conversions are permitted

Casting and Conversion

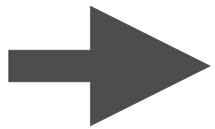
bytearray



int
long
float
double
chararray
bag
tuple
map

Casting and Conversion

long



int
float
double
chararray

Casting and Conversion

http://pig.apache.org/docs/r0.9.1/basic.html#cast

Demo

Specifying partial schemas for a relation

Summary

Understood how relations work, the basic dataset on which Pig operates

Implemented the load, store and dump commands

Worked with both scalar and complex data types supported by Pig