**Read and Write JSON-formatted Data With Apache Pig**

In this Exercise, Wewill explain how to use the JsonStorage and JsonLoader objects in Apache Pig to read and write JSON-formatted data.

**Reading JSON-Formatted Data With JsonLoader**

Apache Pig can read JSON-formatted data if it is in a particular format. Each row in the file has to be a JSON dictionary where the keys specify the column names and the values specify the table content.

For example, supposed our data had three columns called food, person, and amount. We can store this data in first\_table.json as:

{"food":"Tacos", "person":"Alice", "amount":3}

{"food":"Tomato Soup", "person":"Sarah", "amount":2}

{"food":"Grilled Cheese", "person":"Alex", "amount":5}

We can then load the file using JsonLoader as:

second\_table = LOAD '/user/maria\_dev/json\_test/second\_table.json'

USING JsonLoader('food:chararray, person:chararray, amount:int');

Here, 'food:chararray, person:chararray, amount:int' is the Pig schema for the data.

This creates the expected table:

| **food** | **person** | **amount** |
| --- | --- | --- |
| Tacos | Alice | 3 |
| Tomato Soup | Sarah | 2 |
| Grilled Cheese | Alex | 5 |

**Reading Nested Data**

What is nice is that JSON and Pig both support nesting data. We can store both bags of data and tuples in JSON and have them read into Pig. Pig expects tuples to be stored in JSON as dictionaries and bags as lists of dictionaries. In our next example, third\_table.json contains rows with both a bag and a tuple:

{"recipe":"Tacos","ingredients":[{"name":"Beef"},{"name":"Lettuce"},{"name":"Cheese"}],"inventor":{"name":"Alex","age":25}}

{"recipe":"TomatoSoup","ingredients":[{"name":"Tomatoes"},{"name":"Milk"}],"inventor":{"name":"Steve","age":23}}

Notice that for the first row, the ingredients bag is stored as a list of dictionaries

([{"name":"Beef"},{"name":"Lettuce"},{"name":"Cheese"}]). Similarly, the inventor tuple is stored as a dictionary ({"name":"Alex","age":25}).

We can read this data in Pig by specifying a more complicated schema:

third\_table = LOAD '/user/maria\_dev/json\_test/third\_table.json'

USING JsonLoader('recipe:chararray,

ingredients: {(name:chararray)},

inventor: (name:chararray, age:int)');

We can DUMP this data using Pig to ensure that the data is loaded correctly:

(Tacos,{(Beef),(Lettuce),(Cheese)},(Alex,25))

(Tomato Soup,{(Tomatoes),(Milk)},(Steve,23))

**Writing JSON-Formatted Data With JsonStorage**

Finally, we can write JSON-formatted data using JsonStorage. Imagine we had a simple [CSV](http://en.wikipedia.org/wiki/Comma-separated_values) file first\_table.dat:

cat > first\_table.dat

Tacos

Tomato Soup

Grilled Cheese

We can read it into Pig using PigStorage and then save it out using JsonStorage:

first\_table = LOAD 'first\_table.dat'

USING PigStorage()

AS (col1:chararray);

...

STORE first\_table

INTO 'first\_table.json'

USING JsonStorage();

As is the convention in HDFS, the output is a folder called first\_table.json. Inside the folder is a file called part-m-00000 that contains the data in JSON format:

{"col1":"Tacos"}

{"col1":"Tomato Soup"}

{"col1":"Grilled Cheese"}

If the job had lots of output data, it would be spread across additional files like part-m-00001.

Pig also wrote out an intermediate file in the folder called .pig\_schema that explicitly specifies the schema of the output data:

{"fields":[{"name":"col1","type":55,"description":"autogenerated from Pig Field Schema","schema":null}],"version":0,"sortKeys":[],"sortKeyOrders":[]}

This file allows the table to be read in by subsequent Pig jobs without explicitly specifying the schema.