


CARLSON SCHOOL  
OF MANAGEMENT

UNIVERSITY OF MINNESOTA



## Extending Hive: JSON, Custom Scripts, & UDFs

MSBA 6330 Prof Liu

Slides credits: Cloudera Academic Partners Program

---

---

---

---

---


---

---

---

CARLSON SCHOOL  
OF MANAGEMENT

UNIVERSITY OF MINNESOTA



### Goals

- In this module, you will learn
  - How to handle JSON-formatted data
  - How to query JSON-encoded fields with Hive
  - How to use TRANSFORM for custom record processing
  - How to add support for a User Defined Function (UDF)

---

---

---

---

---


---

---

---

CARLSON SCHOOL  
OF MANAGEMENT

UNIVERSITY OF MINNESOTA



## How to load JSON tables

Extending Hive

---

---

---

---

---

---

---

---

## JSON (JavaScript Object Notation) Table

- Hive can work with JSON tables with JsonSerDe
- A **JSON table** is a collection of JSON documents:
  - Each JSON document must fit in a single line of the text file.
  - Arrays and maps are supported
  - Nested data structures are also supported
  - Read the data stored in JSON format

### JSON Table Example:

```
{ "nationkey": "5", "name": "ETHIOPIA", "regionkey": "0", "comment": "ven packages wake quickly. regu" }
{ "nationkey": "6", "name": "FRANCE", "regionkey": "3", "comment": "refully final requests. regular. iron" }
{ "nationkey": "7", "name": "GERMANY", "regionkey": "3", "comment": "platelets. regular accounts x-ray: unusual, regular acco" }
```

### The data could contain nested JSON elements like this

```
{ "country": "Switzerland", "languages": ["German", "French", "Italian"], "religions": { "catholic": {10,20}, "protestant": {40,50} } }
```

---

---

---

---

---

---

---

---

## Create a JSON table

### • Example

```
{ "country": "Switzerland", "languages": ["German", "French", "Italian"], "religions": { "catholic": {10,20}, "protestant": {40,50} } }
```

```
CREATE EXTERNAL TABLE json_nested_test (
  country string,
  languages array<string>,
  religions map<string,array<int>>)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
STORED AS TEXTFILE
LOCATION '...';
```

JSON serde is available in hive 0.12.0 and later. In some distributions, a reference to hive-hcatalog-core.jar is required. ADD JAR /usr/lib/hive-hcatalog/lib/hive-hcatalog-core.jar; In Hive 4.0, you can use STORED AS JSONFILE. On our VM, you need to install it using, wget <http://idsdl.csom.umn.edu/c/share/msba6330/json-serde-1.3.8-jar-with-dependencies.jar> then ADD JAR /home/cloudera/json-serde-1.3.8-jar-with-dependencies.jar;

---

---

---

---

---

---

---

---

## How to query JSON-encoded fields with Hive

Extending Hive

---

---

---

---

---

---

---

---

### Use GET\_JSON\_OBJECT to query JSON-formatted string fields

- In some cases, some fields, but not the entire row, are JSON encoded.
  - e.g.: John Doe | 25 | ["soccer", "video game", "ski"]
- GET\_JSON\_OBJECT can be used to parse the field and extract information from it.
  - Such a field has a string type
  - It is different from a complex field, which has predefined the field's structure.

---

---

---

---

---

---

---

---

### Use GET\_JSON\_OBJECT to parse json fields

- The function takes the form of `get_json_object(src_json, json_path)`
  - `$`: root object
  - `[ ]`: subscript operator for array
  - `.`: child operator

```
{ "store": { "fruit": [ { "weight": 8, "type": "apple" }, { "weight": 9, "type": "pear" } ], "bicycle": { "price": 19.95, "color": "red" } }, "email": "amy@only_for_json_udf_test.net", "owner": "amy" }
```

Suppose one column called X stores JSON blob strings like this.

```
get_json_object(X, '$.store.fruit[0]')      { "weight": 8, "type": "apple" }
get_json_object(X, '$.owner')                amy
get_json_object(X, '$.store.bicycle.price')  19.95
get_json_object(X, '$.name')                 NULL
```

---

---

---

---

---

---

---

---

## Data Transformation With Custom Scripts

---

---

---

---

---

---

---

---

## Using External Scripts with TRANSFORM

- Hive allows you to transform data through external scripts or programs
  - These can be written in nearly any language
- Done with HiveQL's **TRANSFORM** & **USING** clauses
  - One or more fields are supplied as arguments to **TRANSFORM()**
  - The external script is identified by a **USING** clause
  - It receives each record, processes it, and returns the result

```
hive> ADD FILE myscript.py;
hive> SELECT TRANSFORM(*) USING 'myscript.py' FROM employees;
```

- Added a custom script
- Use the script to transform rows

## Using External Scripts with TRANSFORM

- Similar to Hadoop Streaming
- Your external program will receive one record per line on standard input
  - Each field in the supplied record will be a tab-separated string
  - NULL values are converted to the literal string \N
- You may need to convert values to appropriate types within your program
  - For example, converting to numeric types for calculations
- Your program must return tab delimited fields on standard output
  - Output fields can optionally be named and cast using the syntax below

```
hive> SELECT TRANSFORM(product_name, price)
       USING 'tax_calculator.py'
       AS (item_name STRING, tax INT)
       FROM products;
```

## Hive TRANSFORM Example (1 Of 3)

- Here is a complete example of using **TRANSFORM** in Hive
  - Our python script parses an e-mail address, determines to which country it corresponds, then returns an appropriate greeting
- Here's a sample of the input data

```
hive> SELECT name, email FROM employees;
Antoine  antoine@example.fr
Kai      kai@example.de
Pedro    pedro@example.mx
Joel     joel@example.us
```

- Here's the corresponding HiveQL code

```
hive> ADD FILE greeting.py;
hive> SELECT TRANSFORM(name, email)
       USING 'greeting.py' AS greeting
       FROM employees;
```

## Hive TRANSFORM Example (2 Of 3)

- The Python script for this example is shown below

```
#!/usr/bin/env python
import sys
import re
greetings = {'de':'Hallo', 'fr':'Bonjour', 'mx':'Hola'}

for line in sys.stdin:
    name, email = line.strip().split('\t')
    match = re.search(r'\.(\w+)', email)
    if match and greetings.has_key(match.group(1)):
        print
        "{0}\t{1}".format(greetings[match.group(1)], name)
    else:
        print "Hello\t{0}".format(name)
```

## Hive TRANSFORM Example (11 Of 11)

- Here is the result of our transformation

```
hive> ADD FILE greeting.py;
hive> SELECT TRANSFORM(name, email)
        USING 'greeting.py' AS greeting
        FROM employees;
Bonjour Antoine
Hallo Kai
Hola Pedro
Hello Joel
```

- Caveat: TRANSFORM is not allowed when SQL authorization is enabled in Hive
  - Due to security risks, Hive will not execute a script on a secure cluster
  - Workaround: use Hadoop Streaming instead of Hive to invoke the script

Source: <https://goo.gl/aj06b>

## User-Defined Functions

Extending Hive

## Overview Of User-Defined Functions (UDFs)

- User Defined Functions (UDFs) are custom functions
  - Invoked with the same syntax as built-in functions
- There are three types of UDFs in Hive
  - Standard UDFs
    - One row of input, one row of output (1:1), e.g. TRIM, UPPER
  - User-Defined Aggregate Functions (UDAFs)
    - Many rows of input, one row of output (\*:1), e.g. SUM, MAX
  - User-Defined Table Functions (UDTFs)
    - One row of input, many rows of output (1:\*), e.g. EXPLODE

```
hive>SELECT CALC_SHIPPING_COST(order_id, 'OVERNIGHT')
FROM orders WHERE order_id = 5742354;
```

---

---

---

---

---

---

---

---

## Developing Hive UDFs

- Hive user-defined functions are written in Java
  - Currently no support for writing them in other languages
  - Using TRANSFORM may be an alternative
- Open source user-defined functions are plentiful on the web/github
- There are three steps for using a user-defined function in Hive
  1. Copy the function's JAR file to HDFS
  2. Register the function
  3. Use the function in your query

---

---

---

---

---

---

---

---

## Example: Using an UDF in Hive

- First, copy the JAR file to HDFS
  - Same step as with a custom SerDe
- Next, register the function and assign an alias
  - The quoted value is the fully qualified Java class for the UDF
- Hive persists the function in the metastore database
  - To remove the function, use DROP FUNCTION url\_decode;

```
hadoop fs -put url-decode-udf.jar /myscripts/
```

```
CREATE FUNCTION url_decode
AS 'com.example.hive.udf.URLDecode'
USING JAR '/myscripts/url-decode-udf.jar';
```

```
SELECT url_decode(your_url);
```

---

---

---

---

---

---

---

---

### Bibliography

- Hive Transform Syntax
  - <https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Transform>
- Hive UDFs
  - <https://www.linkedin.com/pulse/hive-functions-udfudaf-udtf-examples-gaurav-singh>
- Hive get\_json\_object documentation
  - [https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-get\\_json\\_object](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-get_json_object)
- Load json-format data in Hive
  - <https://docs.aws.amazon.com/athena/latest/ug/json.html#openjson>

---

---

---

---

---

---

---