Write a hive UDF that implements functionality of string concat\_ws(string SEP, array<string>).

This UDF will accept two arguments, one string and one array of string.

It will return a single string where all the elements of the array are separated by the SEP.

package org.apache.hadoop.hive.ql.udf.generic;

import org.apache.hadoop.hive.ql.exec.Description;

import org.apache.hadoop.hive.ql.exec.UDFArgumentException;

import org.apache.hadoop.hive.ql.exec.UDFArgumentLengthException;

import org.apache.hadoop.hive.ql.exec.UDFArgumentTypeException;

import org.apache.hadoop.hive.ql.metadata.HiveException;

import org.apache.hadoop.hive.serde.serdeConstants;

import org.apache.hadoop.hive.serde2.objectinspector.ListObjectInspector;

import org.apache.hadoop.hive.serde2.objectinspector.ObjectInspector;

import org.apache.hadoop.hive.serde2.objectinspector.ObjectInspector.Category;

import org.apache.hadoop.hive.serde2.objectinspector.PrimitiveObjectInspector;

import org.apache.hadoop.hive.serde2.objectinspector.PrimitiveObjectInspector.PrimitiveCategory;

import org.apache.hadoop.hive.serde2.objectinspector.primitive.PrimitiveObjectInspectorFactory;

import org.apache.hadoop.hive.serde2.objectinspector.primitive.PrimitiveObjectInspectorUtils;

import org.apache.hadoop.hive.serde2.objectinspector.primitive.PrimitiveObjectInspectorUtils.PrimitiveGrouping;

import org.apache.hadoop.hive.serde2.objectinspector.primitive.StringObjectInspector;

import org.apache.hadoop.io.Text;

@Description(name = "concat\_ws",

value = "\_FUNC\_(separator, [string | array(string)]+) - "

+ "returns the concatenation of the strings separated by the separator.",

extended = "Example:\n"

+ " > SELECT \_FUNC\_('.', 'www', array('facebook', 'com')) FROM src LIMIT 1;\n"

+ " 'www.facebook.com'")

public class GenericUDFConcatWS extends GenericUDF {

private transient ObjectInspector[] argumentOIs;

@Override

public ObjectInspector initialize(ObjectInspector[] arguments) throws UDFArgumentException {

if (arguments.length < 2) {

throw new UDFArgumentLengthException(

"The function CONCAT\_WS(separator,[string | array(string)]+) "

+ "needs at least two arguments.");

}

// check if argument is a string or an array of strings

for (int i = 0; i < arguments.length; i++) {

switch(arguments[i].getCategory()) {

case LIST:

if (isStringOrVoidType(

((ListObjectInspector) arguments[i]).getListElementObjectInspector())) {

break;

}

case PRIMITIVE:

if (isStringOrVoidType(arguments[i])) {

break;

}

default:

throw new UDFArgumentTypeException(i, "Argument " + (i + 1)

+ " of function CONCAT\_WS must be \"" + serdeConstants.STRING\_TYPE\_NAME

+ " or " + serdeConstants.LIST\_TYPE\_NAME + "<" + serdeConstants.STRING\_TYPE\_NAME

+ ">\", but \"" + arguments[i].getTypeName() + "\" was found.");

}

}

argumentOIs = arguments;

return PrimitiveObjectInspectorFactory.writableStringObjectInspector;

}

protected boolean isStringOrVoidType(ObjectInspector oi) {

if (oi.getCategory() == Category.PRIMITIVE) {

if (PrimitiveGrouping.STRING\_GROUP

== PrimitiveObjectInspectorUtils.getPrimitiveGrouping(

((PrimitiveObjectInspector) oi).getPrimitiveCategory())

|| ((PrimitiveObjectInspector) oi).getPrimitiveCategory() == PrimitiveCategory.VOID) {

return true;

}

}

return false;

}

private final Text resultText = new Text();

@Override

public Object evaluate(DeferredObject[] arguments) throws HiveException {

if (arguments[0].get() == null) {

return null;

}

String separator = PrimitiveObjectInspectorUtils.getString(

arguments[0].get(), (PrimitiveObjectInspector)argumentOIs[0]);

StringBuilder sb = new StringBuilder();

boolean first = true;

for (int i = 1; i < arguments.length; i++) {

if (arguments[i].get() != null) {

if (first) {

first = false;

} else {

sb.append(separator);

}

if (argumentOIs[i].getCategory().equals(Category.LIST)) {

Object strArray = arguments[i].get();

ListObjectInspector strArrayOI = (ListObjectInspector) argumentOIs[i];

boolean strArrayFirst = true;

for (int j = 0; j < strArrayOI.getListLength(strArray); j++) {

if (strArrayFirst) {

strArrayFirst = false;

} else {

sb.append(separator);

}

sb.append(strArrayOI.getListElement(strArray, j));

}

} else {

sb.append(PrimitiveObjectInspectorUtils.getString(

arguments[i].get(), (PrimitiveObjectInspector)argumentOIs[i]));

}

}

}

resultText.set(sb.toString());

return resultText;

}

@Override

public String getDisplayString(String[] children) {

assert (children.length >= 2);

return getStandardDisplayString("concat\_ws", children);

}

}