Processing Data with U-SQL - Basics

Updated on: 1/31/2017

# Introduction

In this lab you will learn the basics of how to use U-SQL, the new query language for big data.

# Sample Data

We will be using a subset of Github data from the GHTorrent project. This project monitors the Github public event time line and retrieves the contents of events and their dependencies. With this subset of data we will be looking at projects, users, and commits for Github so we can try and answer some interesting questions about the dataset.

You can find the schema of the dataset here: http://ghtorrent.org/files/schema.pdf.

The data is already uploaded on the **adltrainingsampledata** ADLS account.

# Prerequisites

* Access to an ADLA account
* Access to an Azure Data Lake Store

# Exercise 1: Data & Basic Input Output

In this first exercise, we will simply read data from the GitHub project and write it back out as an ordered dataset based on the updated column. The purpose here is to get used to the basics of extracting and outputting data with U-SQL.

Task: There’s an input file at this location:

adl://adltrainingsampledata.azuredatalakestore.net/GHData/Projects.csv

Write a script that takes any 50K rows and writes them to your ADLS store as this path @"/GHDataSmall/projectssmall.csv". This output CSV should be sorted in increasing order on the **updated\_a** fied,

The schema of the input file is

id int?,

url string,

owner\_id int?,

name string,

descriptor string,

language string,

created\_a DateTime?,

forked\_from int?,

deleted int?,

updated\_a DateTime?

The types that end with question marks are called **nullable types** in .NET. Please read this [MSDN article](https://msdn.microsoft.com/en-us/library/1t3y8s4s.aspx).

# Exercise 2: Using scalar variables

Once you have computed Exercise 1 and change the script such that the input and output files use DECLARE variables to specify the input and output paths. Use the DECLARE variables in the EXTRACT and OUTPUT instead of the literal string values for the paths.

As a hint, the general form for a DECLARE statement is:

DECLARE @<id> <type> = <value>;

# Exercise 3: Encapsulating Code with a TVF

Instead of repeating the EXTRACT in the future, wrap it in a TVF.

Remember to first create a database to hold the TVF with the **CREATE DATABASE** statement.

CREATE DATABASE IF NOT EXISTS <dbname>;

The TVF syntax looks like this

CREATE FUNCTION <dbname>.<schema>.<tvfname>()

RETURNS @<rowset>

AS BEGIN

@<rowset> = …

RETURN;

END;

ProTip: the if the code inside the TVF uses a DECLARE variable, the DECLARE variable must be defined inside the body of the TVF or be passed as a TVF parameter.

# Exercise 4: Grouping and Aggregations

Use the GROUP BY Expression to determine the popularity of languages in GitHub projects. Make sure to use the TVF created in Exercise 3.

The schema of the output should be

* language string,
* count long

# Exercise 5: Using code in .NET Assemblies

There’s an assembly at this location:

adl://adltrainingsampledata.azuredatalakestore.net/Assemblies/ADLHandsOnLabLib.dll

It contains a simple helper function called **DoubleIt**. Please take note of the namespace, the name of the static class, and the name of the method itself.

namespace ADLHandsOnLabLib

{

public static class Helpers

{

public static string DoubleIt(string s)

{

return s + s;

}

}

}

Create an Assembly object in your U-SQL DB using that assembly file.

The create assembly syntax is:

CREATE

ASSEMBLY <db>.<assemblyid> // you can pick any meaningful name for the assemblyid

FROM <assemblypath>;

Now write a script that references that assembly using the **REFERENCE ASSEMBLY** statement. In the script, get 100 rows from the projects TVF and return the project name, and the project name that has been transformed by the DoubleIt method from the assembly.

The fully-qualified name of the C# method is **ADLHandsOnLabLib.Helpers.DoubleIt**