Processing Data with U-SQL - Basics

Updated on: 1/13/2017

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.

Download the Sample Data from here

https://microsoft.sharepoint.com/teams/adlcustomertraining/Shared%20Documents/TechReady/TechReady24/PreTR24/SampleData

Prerequisites

To complete this lab you'll need:

- A copy of Visual Studio with the Azure Data Lake Tools for Visual Studio installed, or access to the ADLA Portal.
- Access to an ADLA account (MyAnalyticsAccount)
- Access to an Azure Data Lake Store (MyStoreAccount)

Basic Read & Write

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.

Run this script:

```
@projects =
    EXTRACT id int?,
            url string,
            owner_id int?,
            name string,
            descriptor string,
            language string,
            created_a DateTime?,
            forked_from int?,
            deleted int?,
            updated_a DateTime?
    FROM @"/TR24/GHData/projectssmall.csv"
    USING Extractors.Csv();

@projects =
    SELECT *
    FROM @projects;


OUTPUT @projects
    TO @"/TR24/Users/[[username]]/output/projectsordered.csv"
    ORDER BY updated_a ASC
    USING Outputters.Csv();
```

2. Change the name of the output file from [[username]] to something unique.

3. Submit your script.

NOTE: This U-SQL script has no transformation step. It reads from an input file, schematizes the data during the read process, and then outputs the intermediate rowset back into the file whose name you specified. The id field could be null or of type int, while the url field cannot be null. Note that the C# string type is always nullable.

This script illustrates the following concepts:

- Rowset variables. Each query expression that produces a rowset can be assigned to a variable. Variables in U-SQL follow the T-SQL variable naming pattern of an ampersand (@) followed by a name (@searchlog in this case). Note that the assignment statement does not execute the query. It merely names the expression and gives you the ability to build-up more complex expressions.
- The EXTRACT keyword. This gives you the ability to define a schema as part of a read operation. For each column, the schema specifies a paired value consisting of a column name and a C# type name. It uses a so-called extractor, which can be created or customized by the user. However, in this case we are using the built-in Tsv (tab-separated value) extractor that is provided by the Extractors class, since the input data is not comma but TAB separated.
- The OUTPUT keyword. This takes a rowset and serializes it as a comma-separated file into the specified location. Like extractors, outputters can be created or customized by the user. However, in this case we are using the built-in Csv (comma-separated value) outputter provided by the Outputters class.

In this next script, we will build upon the previous script by adding in additional datasets, combining the datasets, and producing output.

4. Copy the following U-SQL script into a new visual studio U-SQL file.

```
@projects =
    EXTRACT id int?,
            url string,
            owner_id int?,
            name string,
            descriptor string,
            language string,
            created_a DateTime?,
            forked_from int?,
            deleted int?,
            updated_a DateTime?
    FROM "/TR24/GHData/projectssmall.csv"
    USING Extractors.Csv();

@users =
    EXTRACT id int?,
            login string,
            name string,
            company string,
            city_country string,
            email string,
            created DateTime?,
            type string,
            fake int?,
            deleted int?,
            longitude decimal?,
            latitude decimal?,
            country_code string,
            state string,
            city string
    FROM "/TR24/GHData/userssmall.csv"
    USING Extractors.Csv();

@result_set =
    SELECT p.name,
           u.country_code,
           COUNT(u.id) AS NumberOfUsers
    FROM @projects AS p
         JOIN
             @users AS u
         ON u.id == p.owner_id
    GROUP BY p.name,
             u.country_code;

 OUTPUT @result_set
    TO "/TR24/output/CountofProjectUsers.csv"
    ORDER BY NumberOfUsers DESC
    USING Outputters.Csv();
```

Transforming Using the Catalog

It will often be necessary to transform data from one semi-structured data format, like JSON or XML to another format – performing transformations along the way. In this exercise you will submit a U-SQL script that reads data from an input file in JSON format, extracts and schematizes data, and writes the results into an output file.

U-SQL provides the ability to use .Net assemblies in U-SQL's metadata catalog in order to encapsulate more complex expressions to extend the processing capabilities of U-SQL with user-defined operators. If a .Net assembly is required during script compilation, then it needs to be registered with CREATE ASSEMBLY and then referenced with REFERNECE ASSEMBLY. In this section we will be extracting data from a JSON file that will require us to use a JSON extractor example from Github. We will need to CREATE and REFERENCE those assemblies in our script.

Using Azure Data Lake Explorer, explore to https://adltrainingsampledata.azuredatalakestore.net/TR24/Assemblies. Here you will see two dll's Microsoft.Analyitdcs.Samples.Formats.dll and Newtonsoft.Json.dll. Both of these are required for the next step and we need to CREATE ASSEMBLY if they don't exist. There are two ways to CREATE assemblies, either through the GUI or by referencing them in your U-SQL script. In our lab, we will create the assemblies in our script.

7. Copy the following U-SQL script into the tool of your choice (either a U-SQL file in Visual Studio or the query window on the ADLA Portal.):

```
CREATE ASSEMBLY IF NOT EXISTS [Newtonsoft.Json] FROM @"/TR24/Assemblies/Newtonsoft.Json.dll";
CREATE ASSEMBLY IF NOT EXISTS [Microsoft.Analytics.Samples.Formats] FROM
@"/TR24/Assemblies/Microsoft.Analytics.Samples.Formats.dll";

REFERENCE ASSEMBLY [Newtonsoft.Json];
REFERENCE ASSEMBLY [Microsoft.Analytics.Samples.Formats];

//Extract the Json string using a default Text extractor. This is ideal if you file size is <128 KB.
@json =
    EXTRACT repo_id int?,
            user_id int?,
            created DateTime?,
            ext_ref_id string
      FROM @"/TR24/GHData/ProjectMembers.json"
    USING new Microsoft.Analytics.Samples.Formats.Json.JsonExtractor("[*]");


//Output the file to a tool of your choice.
OUTPUT @json
    TO @"/TR24/output/ProjectMembers.csv"
   USING Outputters.Csv();
```

9. Change the run environment from local to your Data Lake Analytics Account (adlfielddemos).

10. Submit your script.

Exercise 3: Using scalar variables and basic aggregations

Before we load data into tables, we will often want to examine the data and relationships between datasets. In this exercise, we will be extracting two datasets and applying a basic aggregation to them for analysis. The ability to get answers before we've loaded the data into tables or operationalized it in any way is one of the main benefits of U-SQL. In this case we are looking to get the number of project users by country.

1. Edit your U-SQL script to resemble the following:

```
@projects =
    EXTRACT id int?,
            url string,
            owner_id int?,
            name string,
            descriptor string,
            language string,
            created_a DateTime?,
            forked_from int?,
            deleted int?,
            updated_a DateTime?
    FROM "/TR24/GHData/Projects.csv"
    USING Extractors.Csv();

@users =
    EXTRACT id int?,
            login string,
            name string,
            company string,
            city_country string,
            email string,
            created DateTime?,
            type string,
            fake int?,
            deleted int?,2.
            longitude decimal?,
            latitude decimal?,
            country_code string,
            state string,
            city string
    FROM "/TR24/GHData/Users.csv"
    USING Extractors.Csv();

@result_set =
    SELECT p.name,
           u.country_code,
           COUNT(u.id) AS NumberOfUsers
    FROM @projects AS p
         JOIN
             @users AS u
         ON u.id == p.owner_id
    GROUP BY p.name,
             u.country_code;

 OUTPUT @result_set
    TO "/TR24/Users/[[username]]/output/CountofProjectUsers.csv"
    ORDER BY NumberOfUsers DESC
    USING Outputters.Csv();
```

2. As before, change the name of the output file from [[username]] to [username].

3. Change the run environment from local to your Data Lake Analytics Account (adlfielddemos).

4. Submit your script and check out the results by opening the CountofProjectUsers file.

Note that the WHERE clause is using a boolean C# expression and thus the comparison operation is == (and not the = sign you may be familiar with from traditional SQL).

You can apply more complex filters by combining them with logical conjunctions (ANDs) and disjunctions (ORs), and you can even use the full power of the C# programming language to create your own expressions and functions. U-SQL supports both AND and OR keywords (which may reorder the predicate arguments) and && and || operators (which provide order guarantee and short cutting).

In this next script, you will introduce scalar variables to make your scripts easier to maintain. In this script we will define the input and output directories at the top of the script, making the rest of the scrip much more readable. This is very useful when you have many files in a script all coming from the same directory. If that directory ever needs to change, you can change it in one place as apposed to many times throughout the script.

1. Create a new U-SQL script to resemble the following, then resubmit the script:

```
// Initial parameters for constants
DECLARE @ALIAS = "[[username]]";
DECLARE @INPUTDIR string = "/TR24/GHData/";

DECLARE @INPUT_PROJECTS string = @INPUTDIR + "Projects.csv";
DECLARE @INPUT_COMMITS string = @INPUTDIR + "Commits.csv";

DECLARE @OUTPUTDIR string = "/TR24/Users/" + @ALIAS + "/output/";
DECLARE @OUTPUT string = @OUTPUTDIR + " NumberOfCommitters.csv";

@projects =
    EXTRACT id int?,
            url string,
            owner_id int?,
            name string,
            descriptor string,
            language string,
            created_a DateTime?,
            forked_from int?,
            deleted int?,
            updated_a DateTime?
    FROM @INPUT_PROJECTS
    USING Extractors.Csv();


@commits =
    EXTRACT id int?,
            sha string,
            author_id int?,
            committer int?,
            project_id int?,
            created DateTime?
    FROM @INPUT_COMMITS
    USING Extractors.Csv();

@result_set =
    SELECT p.name,
           COUNT(c.id) AS NumberOfCommitters
    FROM @projects AS p
         JOIN
             @commits AS c
         ON c.project_id == p.id
    GROUP BY p.name;


 OUTPUT @result_set
    TO @OUTPUT
    ORDER BY NumberOfCommitters DESC
    USING Outputters.Csv();
```

2. As before, change the name of the output file from [[username]] to [username].

3. Change the run environment from local to your Data Lake Analytics Account (adlfielddemos).

4. Submit your script and check out the results by opening the NumberOfCommitters file.