Exercise 1

Using Jaql

Contents

Lab 1 Using Jaql

In this exercises you will access JSON arrays and records using Jaql.

After completing this hands-on lab, you'll be able to:

- Manipulate arrays and records usnig Jaql

Allow 15 minutes to complete this lab.

This version of the lab was designed using the InfoSphere BigInsights 2.1 Quick Start Edition, but has been tested on the 2.1.2 image. Throughout this lab you will be using the following account login information. If your passwords are different, please note the difference.

	Username	Password
VM image setup screen	root	password
Linux	biadmin	biadmin

1.1 Start the BigInsights components

__ 1. Log into your BigInsights image with a userid of **biadmin** and a password of **biadmin.**

__ 2. Start Hadoop and its components using the icon on the desktop.

1.2 Setup to run Jaql

You have two choices for your Jaql environment. You can start your Jaql shell from a command prompt or you can use the Eclipse environment. The nice thing about using the Eclipse environment is that you can cut and paste previously executed Jaql commands into the Jaql command prompt. We will use the Eclipse environment.

__ 1. If you want to run Jaql from a command window, do the following steps.

__ a. Open a command window. (Right click the desktop and select **Open Terminal**.)

__ b. Change to the Jaql bin directory.

```
cd $BIGINSIGHTS_HOME/jaql/bin
```

__ c. Start the Jaql shell.

```
./jaqlshell
```

Note:

You can code your Jaql script in a text file and then execute the jaqlshell command, passing one or more script files.

```
./jaqlshell script1 script2 ...
```

Note:

If you do not have a BigInsights server defined to your Eclipse system, do the following to create one:

__ a. In the BigInsights Servers frame, right-click *BigInsights Servers* and select **New.**

__ b. Type in the *URL* of the BigInsights server. For this exercise it is **http://bivm:8080.**

__ c. You can keep the default *Server name.*

__ d. *User ID* is **biadmin.**

__ e. *Password* is **biadmin.**

__ f. Select to save the password.

__ g. Test your connection. If the test is successful, click **OK** and the **Finish.**

__ h. If prompted to enter a password, enter **biadmin.**

__ i. If asked, click **No** on the *Secure Storage* dialog.

__ j. Click **Finish.**

__ 2. To run Jaql from within the Eclipse environment, do the following steps.

__ a. Start Eclipse and when prompted for a workspace, click **OK.**

__ b. Make sure that you are using the BigInsights perspective. Click **Window->Open Perspective->Other.** Select *BigInsights* and click **OK.**

__ c. A *BigInsights Server* has already been defined**.**

__ d. In the *BigInsights Servers* view, expand **BigInsights Servers.**

__ e. Right click *bivm-bivm:8080...* and select **Open JAQL Shell.**

Note:

If you want to create a Jaql application that you would later publish to a BigInsights server, then you would want to create a BigInsights project and a Jaql program.

1. In the *Overview* tab of the *Task Launcher for Big Data,* click **Create a new BigInsights project.** Give your project a name and click **Finish.**
2. Click the *Develop* tab of the *Task Launcher for Big Data* and click **Create a BigInsights program.** Then you can select either *JAQL Script* or *JAQL Module,* depending on your needs. Then click **OK.**

1.3 Some Jaql basics

__ 1. At your Jaql prompt, create an array;

```
a1 = [1,2,3,4];
```

__ 2. Access the last element in the array.

```
a1[3];
```

__ 3. Access the first three elements in the array/

```
a1[0:2];
```

__ 4. Next update the first element in the array. Try the following and see what happens.

```
a1[0]=9;
```

__ 5. Well, that did not work successfully, so try it the correct way.

```
a1 = replaceElement(a1,0,9);
a1;
```

__ 6. Create an array, *a2,* whose values *range* from 1 to 20.

```
a2 = range(1,20);
a2;
```

__ 7. Next *reverse* all of the values in that array.

```
reverse(a2);
```

__ 8. And finally determine the number of elements in the array

```
count(a2);
```

1.4 Work with records

__ 1. Create an array that has multiple records:

```
a3=[{name: 'John', age: 40, children:['Katie','Will']}, {name:'Mary',
age:21}];
```

__ 2. What happens if you access the field called *name?*

```
a3.name;
```

__ 3. Both *name* field values were displayed. The student guide also stated that there was another way of getting all values of a field that was more self-documenting.

```
a3[*].name;
```

__ 4. What if you only wanted the value of the *name* field for the first record?

```
a3[0].name;
```

__ 5. And if you wanted the name of the second child from the first record:

```
a3[0].children[0];
```

__ 6. List only the *name* and *age* fields for the first record.

```
a3[0]{.name,.age};
```

__ 7. Add a new field called *gender* to the second record and give it a value of 'F';

```
{a3[1].*,gender:'F'};
```

__ 8. Remove the *age* field from the first record.

```
a3[0]{*-.age};
```

__ 9. Remove the *age* field from the first record and at the same time add a new field called *gender* with a value of 'M';

```
{a3[0]{*-.age},gender:'M'};
```

__ 10. List the field *names* in the first record;

```
names(a3[0]);
```

1.5 Jaql functions

With Jaql, it is easy to create user-defined functions written entirely in Jaql. Since you have not yet been exposed to most of the features and functions of Jaql, the functions that you will now code will be rather simplistic. In spite of this, you should still be able to get the concept of coding functions in Jaql.

__ 1. Create a function called *myabs* that calls the built-in absolute function.

```
myabs = fn(x) abs(x);
```

__ 2. Test it with both a positive and a negative value.

```
myabs(4);
myabs(-4);
```

__ 3. Next create a function, *evenodd,* that checks if the input value is even or odd and returns an appropriate string value.

```
evenodd = fn(x) if (mod(x,2) == 0) "even" else "odd";
```

__ 4. Test your function.

```
evenodd(5);
evenodd(6);
```

__ 5. Finally create a function, called *myfunc,* that accepts other functions that support a single value.

```
myfunc = fn(op, val) op(val);
```

__ 6. Test your function.

```
myfunc(myabs, -5);
myfunc(evenodd,5);
```

End of exercise

NOTES

NOTES