4th April

Variable Performance: Declared vs Let

Declared Variables:

1. Only evaluated if they are referenced.
2. Evaluation starts when referenced.
3. Execution is halted when its first referenced.
4. Only evaluated once.
5. Global scoped

Now lets dive into some declare variable examples and demonstrating these properties. I needed a way to create variables that take different amounts of time to evaluate. To do this I created the local:slow() function that accepts how many seconds to wait. The function spawns a xdmp:sleep command to the task server with the <result> option set to true. The result option causes the caller to wait for the task to finish and return the result.

```
declare function local:slow($i as xs:int) {
  xdmp:spawn-function(
    function() {
      xdmp:sleep($i * 1000), $i
    },
    <options xmlns="xdmp:eval">
      <result>{fn:true()}</result>
    </options>
  )
};
```

local:slow.xqy view raw [https://gist.github.com/williammsawyer/9969147/raw/local:slow.xqy] [https://gist.github.com/williammsawyer/9969147#file-local-slow-xqy] hosted with ❤ by GitHub [https://github.com]

1. Only Evaluated if they are used. - Notice no time was taken because none of the variables were referenced.

```
declare variable $a := local:slow(5);
declare variable $b := local:slow(4);
declare variable $c := local:slow(3);
declare variable $d := local:slow(2);
declare variable $e := local:slow(1);

xdmp:elapsed-time()

(: Result: 0 Seconds :)
```

declareVariable1.xqy view raw [https://gist.github.com/williammsawyer/9969147/raw/declareVariable1.xqy] [https://gist.github.com/williammsawyer/9969147#file-declarevariable1-xqy] hosted with ❤ by GitHub [https://github.com]

2. Evaluation starts when referenced. - It took 15 seconds because each one started when it was referenced.

3. Evaluation is halted when its first referenced. - The Execution had to wait for each variable as it was referenced.

```
declare variable $a := local:slow(5);
declare variable $b := local:slow(4);
declare variable $c := local:slow(3);
declare variable $d := local:slow(2);
declare variable $e := local:slow(1);

$a || $b || $c || $d || $e || " " || xdmp:elapsed-time()

(: Result: 15 Seconds :)
```

declareVariable2.xqy view raw [https://gist.github.com/williammsawyer/9969147/raw/declareVariable2.xqy] [https://gist.github.com/williammsawyer/9969147#file-declarevariable2-xqy] hosted with ❤ by GitHub [https://github.com]

4. Only evaluated once. - Each variable was referenced twice and it still took 15 seconds because they were only evaluated once.

```
declare variable $a  := local:slow(5);
declare variable $b := local:slow(4);
declare variable $c := local:slow(3);
declare variable $d := local:slow(2);
declare variable $e := local:slow(1);

$a || $a || $b || $b || $c ||  $c || $d || $d || $e || $e || " " || xdmp:elapsed-ti

(: Result: 15 Seconds :)
```

declareVariable3.xqy view raw [https://gist.github.com/williammsawyer/9969147/raw/declareVariable3.xqy] [https://gist.github.com/williammsawyer/9969147#file-declarevariable3-xqy] hosted with ❤ by GitHub [https://github.com]

Summary - Lets see if we can get these rules to work for us. By referencing all the variables at once they all start evaluating at the same time and in parallel because they have no dependencies to each other. Then when we actually use them in then return we find it only took 5 seconds to execute.

```
declare variable $a := local:slow(5);
declare variable $b := local:slow(4);
declare variable $c := local:slow(3);
declare variable $d := local:slow(2);
```

```
declare variable $e := local:slow(1);

let $referenced := ($a, $b, $c, $d, $e)
return
  $a || $b || $c || $d || $e || " " || xdmp:elapsed-time()

(: Result: 5 Seconds :)
```

declareVariable4.xqy view raw [https://gist.github.com/williammsawyer/9969147/raw/declareVariable4.xqy] [https://gist.github.com/williammsawyer/9969147#file-declarevariable4-xqy] hosted with ❤ by GitHub [https://github.com]

Let Variables:

1. Always evaluated.
2. Evaluation starts when defined.
3. Execution is halted if and when its first referenced.
4. Only evaluated once.
5. Local scope.

1. Always evaluated. - Even though $a was never referenced it was still evaluated and throw an error.

```
let $a := (local:slow(5), fn:error())
let $b := local:slow(4)
let $c := local:slow(3)
let $d := local:slow(2)
let $e := local:slow(1)
return
  $b  || $c || $d || $e || " " || xdmp:elapsed-time()

(: Result: error :)
```

letVariable1.xqy view raw [https://gist.github.com/williammsawyer/9969147/raw/letVariable1.xqy] [https://gist.github.com/williammsawyer/9969147#file-letvariable1-xqy] hosted with ❤ by GitHub [https://github.com]

2. Evaluation starts when defined. - This has really interesting results when you compare it to the second declared example. This example only took 5 seconds where the declared took 15 seconds. The lets executed in parallel as they were defined and didn't stop and wait until they were referenced.

```
let $a := local:slow(5)
let $b := local:slow(4)
let $c := local:slow(3)
let $d := local:slow(2)
let $e := local:slow(1)
return
```

```
  $a || $b  || $c || $d || $e || " " || xdmp:elapsed-time()

(: Result: 5 Seconds :)
```

letVariable2.xqy view raw [https://gist.github.com/williammsawyer/9969147/raw/letVariable2.xqy] [https://gist.github.com/williammsawyer/9969147#file-letvariable2-xqy] hosted with ❤ by GitHub [https://github.com]

3. Execution is halted if and when its first referenced. - $z references $a so it couldn't start evaluating until $a finished. Causing the execution to take 10 seconds.

```
let $a := local:slow(5)
let $z := local:slow($a)
let $b := local:slow(4)
let $c := local:slow(3)
let $d := local:slow(2)
let $e := local:slow(1)
return
  $z || $b  || $c || $d || $e || " " || xdmp:elapsed-time()

(: Result: 10 Seconds :)
```

letVariable3.xqy view raw [https://gist.github.com/williammsawyer/9969147/raw/letVariable3.xqy] [https://gist.github.com/williammsawyer/9969147#file-letvariable3-xqy] hosted with ❤ by GitHub [https://github.com]

4. Only evaluated once. - Each variable was referenced twice and it still took 5 seconds because they were only evaluated once.

```
let $a := local:slow(5)
let $b := local:slow(4)
let $c := local:slow(3)
let $d := local:slow(2)
let $e := local:slow(1)
return
$a || $a || $b || $b || $c ||  $c || $d || $d || $e || $e || " " || xdmp:elapsed-ti

(: Result: 5 Seconds :)
```

letVariable4.xqy view raw [https://gist.github.com/williammsawyer/9969147/raw/letVariable4.xqy] [https://gist.github.com/williammsawyer/9969147#file-letvariable4-xqy] hosted with ❤ by GitHub [https://github.com]

Summary - The return waited for 6 seconds before it needed the variables and by the time it was done the other variables were evaluated and no extra time was taken waiting for them.

```
let $a := local:slow(5)
let $b := local:slow(5)
let $c := local:slow(5)
let $d := local:slow(5)
let $e := local:slow(5)
return (
  local:slow(6) || $a || $b || $c || $d || $e || " " || xdmp:elapsed-time()
)

(: Result: 6 seconds :)
```

letVariable5.xqy view raw [https://gist.github.com/williammsawyer/9969147/raw/letVariable5.xqy] [https://gist.github.com/williammsawyer/9969147#file-letvariable5-xqy] hosted with ❤ by GitHub [https://github.com]

Conclusion

Now these examples I have shown are skewed to the extreme when it comes to slow performance problems. But what I found is the type of variable and how it is used can affect performance. Most of the time it will be negligible. But when you know you have a function or script that is going take some time to evaluate assign it to let variable as soon as possible. And then don't reference it until you absolutely need the value or if you don't need the value don't reference it. This will allow it execute in parallel of everything else and not slow down the request. If you are unsure which to use I feel its always best to start with lets. Declare variables do have their place. They are nice for global constants and if you need a variable outside a function in a library module its your only option.

Posted 4th April by William Sawyer

Labels: marklogic, performance, xquery

0 Add a comment