PROTODEMOS

PD/DBL CODE EXAMPLES

These are actual examples of code that can run in PD/DBL today.

1. "Hello World" in PD/DBL

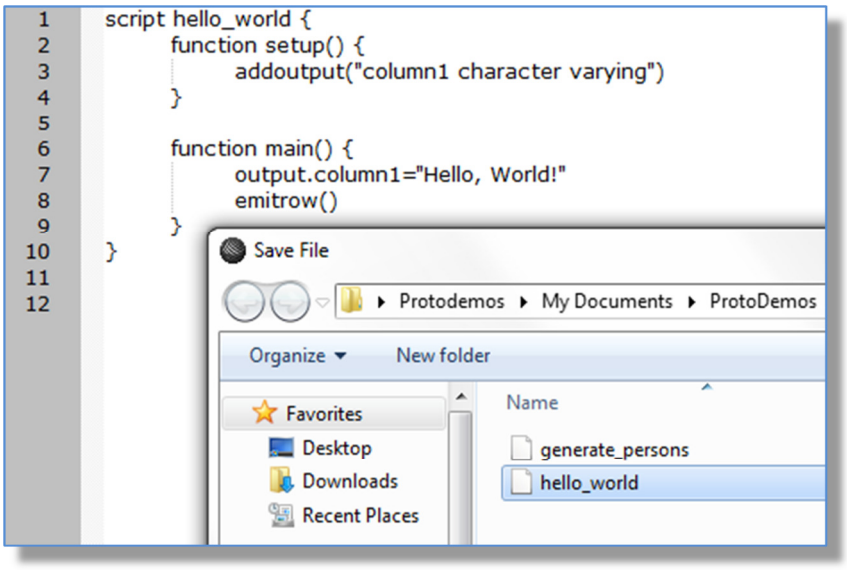
```
mydatabase=>SELECT * FROM pddbl(
mydatabase->ON (SELECT 1)
mydatabase->
mydatabase->SCRIPT('
mydatabase'>script hello_world {
mydatabase'>        function setup() {
mydatabase'>                addoutput("column1 character varying")
mydatabase'>        }
mydatabase'>
mydatabase'>        function main() {
mydatabase'>                output.column1="Hello, World!"
mydatabase'>                emitrow()
mydatabase'>        }
mydatabase'>}
mydatabase'>'))
mydatabase->;
    column1
---------------
 Hello, World!
(1 row)
```

2. "Hello World" as a PD/DBL stored procedure: Save the text of the program as a file, then use "\install" to upload it to nCluster.

```
mydatabase=>
mydatabase=>\install hello_world
```

Then execute the stored procedure using "@" and the name of the file for the "SCRIPT" argument.

```
mydatabase=>
mydatabase=>SELECT * FROM pddbl(
mydatabase->ON (SELECT 1)
mydatabase->
mydatabase->SCRIPT('@hello_world')
mydatabase->
mydatabase->);
    column1
---------------
 Hello, World!
(1 row)
```



3. Example: PD/DBL stored procedure to print out text of a PD/DBL stored procedure (or any text-based installed file).

```
#printfile - Print the lines of an installed file
script printfile {

    function setup() {
            removeoutput("*")
            addoutput("LineNumber integer")
            addoutput("ScriptText character varying")
    }

    function main() {
            #Do nothing.
    }

    function drain() {
            var myfile file
            if(argc==0) then {
                    output.ScriptText="ERROR: Name of file must be passed in as first argument of DRAINARGS"
                    emitrow()
                    return()
            } endif

            if(fileexists(argv[0])==false) then {
                    output.ScriptText="ERROR: file " + argv[0] + " does not exist"
                    emitrow()
                    return()
            } endif

            fileopen(myfile,argv[0])

            var myline string
            var linenum integer
            linenum=1
            for(myline=readline(myfile);isanull(myline)==false;myline=readline(myfile)) {
                    output.LineNumber=linenum
                    output.ScriptText = myline
                    emitrow()
                    linenum=linenum+1
            }
    }
}
```

Call using the “@scriptname” syntax. Pass in file name as argument to the drain() function by adding a “DRAINARGS” clause.

```
mydatabase=>
mydatabase=>SELECT * FROM pddbl(
mydatabase->ON (SELECT 1)
mydatabase->
mydatabase->DRAINARGS('hello_world')
mydatabase->
mydatabase->SCRIPT('@printfile')
mydatabase->
mydatabase->);
 LineNumber |                ScriptText
------------+-------------------------------------------
          1 | script hello_world {
          2 |   function setup() {
          3 |           addoutput("column1 character varying")
          4 |   }
          5 |
          6 |   function main() {
          7 |           output.column1="Hello, World!"
          8 |           emitrow()
          9 |   }
         10 | }
         11 |
(11 rows)
```

4. Code Snippet: Iterate over the contents of an array.

```
var array1 array[] integer
#...some code that populates array1...#
var item integer
var total integer
var total=0
foreach(item : array1) {
    total=total+item
}
```

5. Code snippet: Iterate over the contents of SEVERAL arrays.

```
var array1 array[] integer
var array2 array[] integer
var array3 array[] integer
#...some code that populates array1, array2 & array3...#
var item integer
var total integer
var total=0
foreach(item : array1, array2, array3) {
    total=total+item
}
```

6. Code Snippet: Declare a JDBC connection and use it to execute some arbitrary SQL.

```
connection asterqueen is
    driver "com.asterdata.ncluster.Driver"
    url "jdbc:ncluster://192.168.100.100/mydatabase"
open connection asterqueen
    login "mylogin"
    password "mypassword"
execsql "delete from junk where j2=2"
    using connection asterqueen
close connection asterqueen
```

7. Code Snippet: Get data from multiple inputs (i.e. operate as multi-input Sql-MR function).

```
#input1 is a DIMENSION input.
#Store contents as a map variable for later lookup:
var lookup map[string,string]
while(consume(input1,inp0rec)
    lookup[input1.natural_key]=input1.surrogate_key
}

#input0 are some fact table rows.
#We  need to look up the dimension surrogate_key
#based on a natural key on the fact.
while(consume(input0,factrec)) {
    factrec.dim_sk = lookup[input0.dim_natural_key)
    emitrow(factrec)
}
```

8. Code snippet: User-defined function, with recursion, (also demonstrate “case” statement).

```
function pyramid(a integer) returns integer {
      return(case a
               when 1 then 1
               else a+pyramid(a-1)
             end)
      }
```

9. Code snippet: Exception handling using try/catch/finally, (also demonstrate rollback).

```
#Read in records from nNcluster cursor
#Attempt to emit them to a JDBC cursor
#Rollback and quit after 10,000 bad records
while(consume(input0,factrec)) {
    factrec.dim_sk = lookup[input0.dim_natural_key)
    try {
            insertrow(jdbc_cur1,factrec)
            writerow(jdbc_cur1)
    } catch(errtext, stackarr) {
            if(regexfind(errtext,'(SOME ERROR TEXT|SOME OTHER ERROR TEXT)')>0) {
                    if(getcount(jdbc_cur1,"ERRORS") > 10000) then {
                            rollback connection jdbc_con1
                            close cursor jdbc_cur1
                            close connection jdbc_con1
                            break
                    } endif
            }
    }
}
```

10. Code snippet: User-defined function, with recursion, (also demonstrate "case" statement).

```
function pyramid(a integer) returns integer {
      return(case a
               when 1 then 1
               else a+pyramid(a-1)
             end)
      }
```

11. Code snippet: Coalesce function.

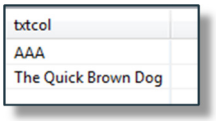
```
    function example11() {
            var a integer
            var b integer
            var c integer

            c=23
            output.intcol=coalesce(a,b,c)
            emitrow() #emits 23 in column intcol
    }
```

12. Code snippet: Use minic() to find min string, ignoring case (also demonstrate strupper() to upper-case and strproper() to proper-case a string).

```
    function example11() {
            output.txtcol=strupper(minic("aaa","bbb","aAa","bBb"))
            emitrow() #emits "AAA" in column intcol
            output.txtcol=strproper("the quick brown dog")
            emitrow() #emits "The Quick Brown Dog")
    }

    #Generate rows
    function drain() {
            if(taskindex()==0) then
                    example11()
            endif
    }
```



txtcol
AAA
The Quick Brown Dog

13. Semi-Complex Example: Generate test data using “pickone” function

```
SELECT * FROM pddbl(
ON (SELECT * FROM persons WHERE 1=0)
DRAINARGS('10000') --Number of records to generate

SCRIPT('
#Generate Persons - a simple script for generating test "person" records
script genpersons {

    #Set up output columns (ON clause generates no records, but can be
    #used to pick up table structure definition
    function setup() {
            addoutput("input0.*") #Add all input columns to output definition
    }

    #Ignore input data (WHERE clause has "1=0" as a predicate)
    function main() {
    }

    #Generate rows
    function drain() {
            #Only run on one vWorker
            if(taskindex() != 0) then return() endif

            #Set up variables for random name generator
            var firstnames file #Installed file containing most common first names in USA
            var lastnames file  #Installed file containing most common last names in USA
            var firstnamesarr array[] string #array to hold the first names
            var lastnamesarr array[] string #array to hold the last names

            #open the files (previously uploaded with \install command)
            fileopen(firstnames, "firstnames.txt")
            fileopen(lastnames, "lastnames.txt")

            #read in comma-separated lists of names
            filesplit(firstnames, firstnamesarr, ",")
            filesplit(lastnames, lastnamesarr, ",")

            #Pick up the row count from the "DRAINARGS" option
            var numrows integer
            numrows=tointeger(argv[0])

            #Generate records
            var i integer #loop iteraator
            for(i=1;i<=numrows;i=i+1) {
                    output.lastname = pickone(lastnamesarr)
                    output.firstname = pickone(firstnamesarr)
                    output.ssn = randint(1,9) * 100000000 + randint(11111111,99999999)
                    emitrow()
            }
    }
}'))
```