MANAGE BIG DATA WITH SAS DS2

WISCONSIN ILLINOIS SAS USERS GROUP
MILWAUKEE
24 JUNE 2015

CHARU SHANKAR
SAS INSTITUTE INC.

AGENDA

what you will learn in this session

1. Basic DS2 syntax
2. Creating reusable code modules with methods
3. Determine when DS2 provides a processing advantage for your data manipulation programs

BASIC DS2 SYNTAX

1.1 WHAT IS DS2?

- New SAS proprietary programming language
- Object-based syntax – user-defined methods and packages
- Intersection of SAS Data step and ANSI SQL : 1999
- Appropriate for advanced data manipulation
- DS2 is included with Base SAS
- ANSI SQL data type support
- Runs anywhere – Base, In-Database (via SAS Code Accelerator), HPA (via HPDS2)

1.1 BASIC DS2 SYNTAX

```
PROC DS2;
data _null_;
   method init();
      dcl varchar(20) foo;
      foo = '**> Starting';
      put foo;
   end;
   method run();
      set ds2_sas.banks;
      put _all_;
   end;
   method term();
      dcl char(11) bar;
      bar = '**> I quit!';
      put bar;
   end;
run; quit;
```

Initial processing

Execution loop

Final processing

1.2 DS2 COMMONALITY WITH SAS DATA STEP

- SAS statements such as SET, BY, RETAIN, PUT, OUTPUT, DO, IFTHEN/ELSE, and others
- the implicit loop of the SET statement
- the implicit declaration of global variables
- ability to call SAS functions and formats

1.2 DS2 EXTENDS THE SAS DATA STEP

- enabling SQL in the SET statement
- in-database execution
- supporting new data types such as INTEGER and VARCHAR
- adding user-defined methods and packages
- providing variable scoping
- enabling parallel execution of entire programs through threading

1.2 EASILY CONVERT A DATA STEP TO DS2

```
data _null_;
   /* Section 1 */
   if _n_ =1 then do;
      Text='**> Starting';
      put Text;
   end;

   /* Section 2 */
   set orion.banks end=last;
   put _all_;

   /* Section 3 */
   if last then do;
      Text='**> All done!';
      put Text;
   end;
run;
```

```
proc ds2;
data _null_;









enddata;
run;
quit;
```

ds02d05

1.2 EASILY CONVERT A DATA STEP TO DS2

Converting Section 1

```
data _null_;
   /* Section 1 */
   if _n_ =1 then do;
      Text='**> Starting';
      put Text;
   end;

   /* Section 2 */
   set orion.banks end=last;
   put _all_;

   /* Section 3 */
   if last then do;
      Text='**> All done!';
      put Text;
   end;
run;
```

```
proc ds2;
data _null_;
   method init();
      Text='**> Starting';
      put Text;
   end;









enddata;
run;
quit;
```

ds02d05

1.2 EASILY CONVERT A DATA STEP TO DS2

Converting Section 2

```
data _null_;
   /* Section 1 */
   if _n_ =1 then do;
      Text='**> Starting';
      put Text;
   end;

   /* Section 2 */
   set orion.banks end=last;
   put _all_;

   /* Section 3 */
   if last then do;
      Text='**> All done!';
      put Text;
   end;
run;
```

```
proc ds2;
data _null_;
   method init();
      Text='**> Starting';
      put Text;
   end;

   method run();
      set orion.banks;
      put _all_;
   end;




enddata;
run;
quit;
```

ds02d05

1.2 EASILY CONVERT A DATA STEP TO DS2

Converting Section 3

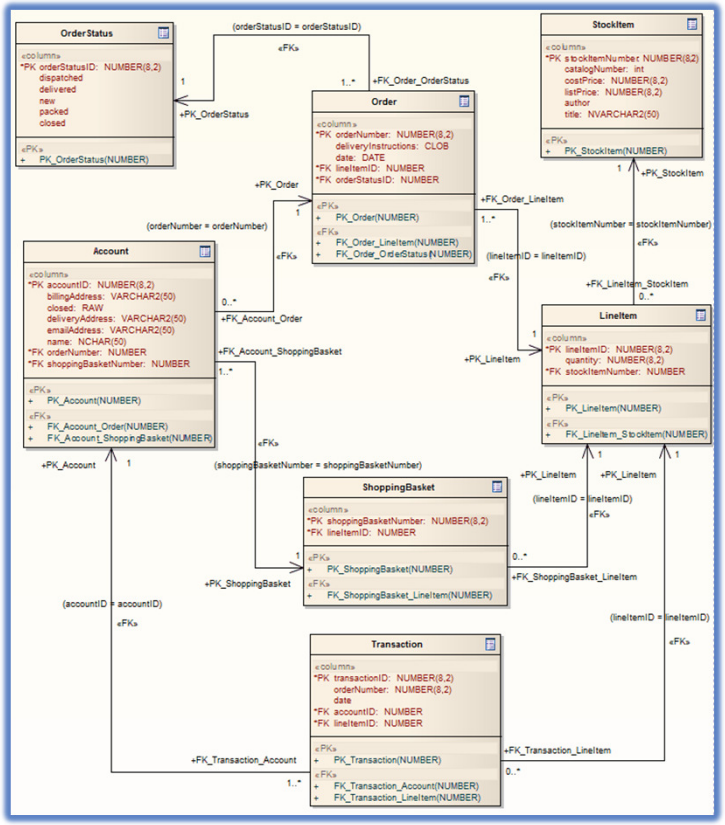
```
data _null_;
   /* Section 1 */
   if _n_ =1 then do;
      Text='**> Starting';
      put Text;
   end;

   /* Section 2 */
   set orion.banks end=last;
   put _all_;

   /* Section 3 */
   if last then do;
      Text='**> All done!';
      put Text;
   end;
run;
```

```
proc ds2;
data _null_;
   method init();
      Text='**> Starting';
      put Text;
   end;

   method run();
      set orion.banks;
      put _all_;
   end;

   method term();
      dcl char(11) Text;
      Text='**> All done!';
      put Text;
   end;
enddata;
run;
quit;
```

ds02d05

1.3 DS2 DATA TYPES

BIGINT
BINARY(*n*)
CHAR(*n*)
DATE
DOUBLE
FLOAT(*p*)
INTEGER
NCHAR(*n*)



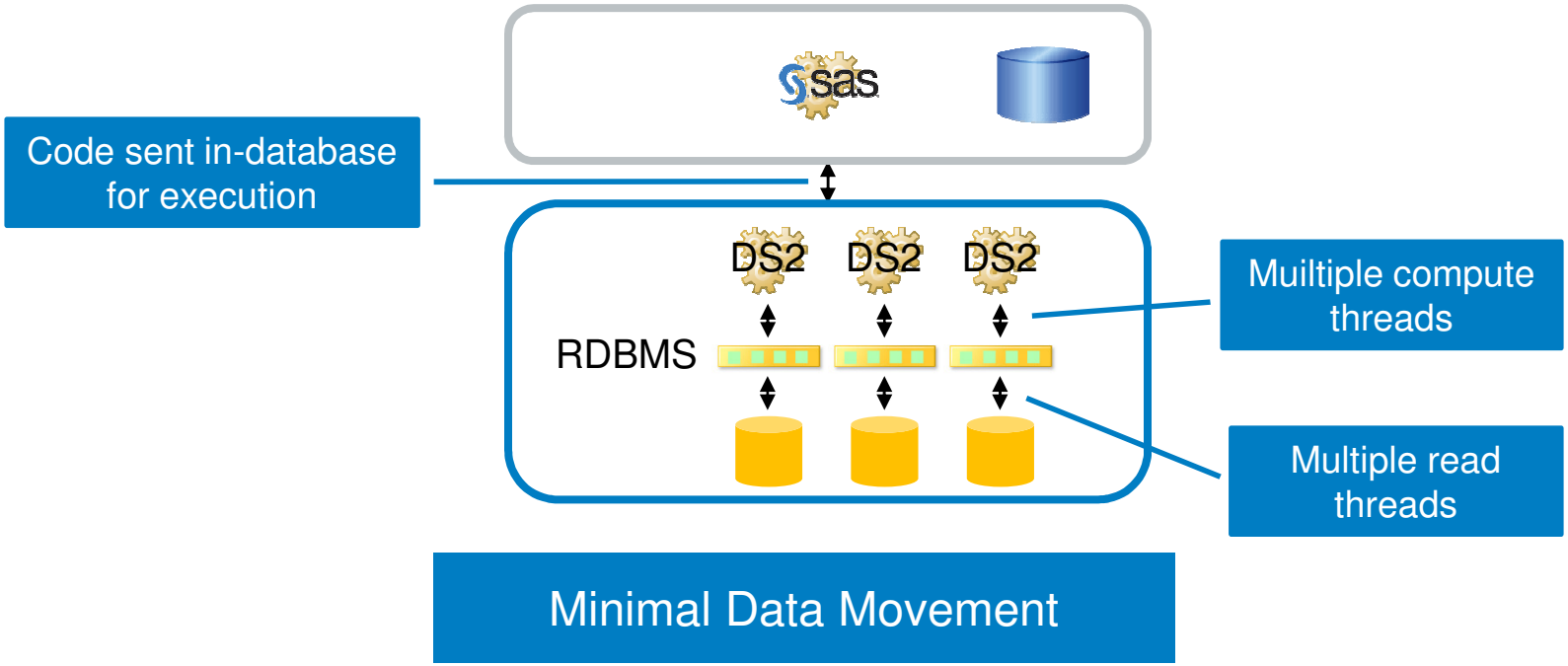
NVARCHAR(*n*)
REAL
SMALLINT
TIME(*p*)
TIMESTAMP(*p*)
TINYINT
VARBINARY(*n*)
VARCHAR(*n*)

1.3 PARALLEL PROCESSING IN BASE SAS

- The Base SAS DATA step processes each observation sequentially.
- DS2 can process observations in parallel.

1.3 PARALLEL PROCESSING IN-DATABASE

- With the SAS In-Database Code Accelerator, DS2 can also be executed in-database.



1.3 EMBEDDED SQL IN DS2

```
proc ds2;
data sales (overwrite=YES);
   keep Customer_ID Total;
   method run();
      set {select c.Customer_ID
                  ,Total_Retail_Price
            from ds2_sas.order_fact f
                  full join
                  ds2_sas.customer_dim c
            on f.Customer_ID=c.Customer_ID
            order by 1};
      by customer_id;
      if first.customer_id then Total=0;
      Total+total_retail_price;
      if last.customer_id then output;
   end; enddata;
run;quit;
```

Returns SQL result set as input stream

BY group processing on results

CREATING REUSABLE CODE MODULES WITH METHODS

2.1 MODERN PROGRAMMING STRUCTURE

Packages

- User–defined: Create collections of re-usable, user-defined methods
- Pre-defined:
  - FCMP – Import PROC FCMP functions
  - Hash / Hash Iterator
  - Matrix – Do matrix math in DS2
  - SQLStmnt – Execute data-driven SQL statements from within a DS2 program

USER-DEFINED METHODS

```
PROC DS2;
data _null_;
   method c2f(double Tc) returns double;
   /* Celsius to Farenheit */
      return (((Tc*9)/5)+32);
   end;
   method init();
      dcl double Degc DegF;
      do DegC=0 to 30 by 15;
         DegF=c2f(DegC);
         PUT DegC= DegF=;
      end;
   end;
enddata;
run;
quit;
```

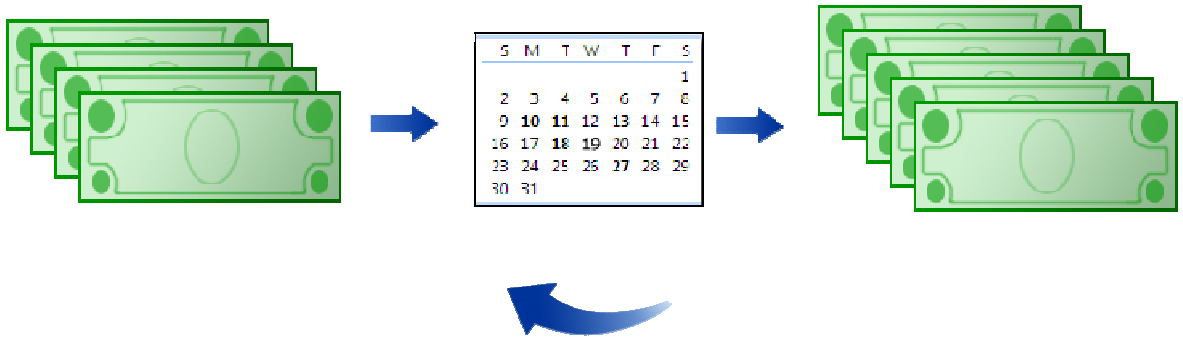
Define method

Call method

2.2 USER DEFINED METHODS

Management wants a standardized method for estimating final values of accounts that accrue compound interest.

For annually compounded interest, the resulting value is equal to the sum of the starting value plus the starting value times the rate. The process should be repeated once for each year.



2.2 USER DEFINED METHODS

Business Scenario: Considerations

- The agreed-upon formula for the interest method is as follows:

  Amount=SUM(**Amount**, **Amount***Rate**)

- The code is reused in all DS2 DATA step programs that perform interest calculations.

2.2 USER DEFINED METHODS

Methods have the following characteristics:

- contain all executable code
- group related program statements into a single, reusable named unit
- leverage global and local variables

```
METHOD MethodName ([IN_OUT] parameter <, parameter,… > )
                                  <RETURNS data-type >;
      … method-body …
END;
```

2.2 USER DEFINED METHODS

User-defined methods have the following characteristics:

- can accept arguments
- can return a value
- execute when they are referenced (called) by name
- can be called multiple times

```
METHOD SumPlus (double V1,double V2 ) RETURNS double;
      RETURN (sum(v1,v2)+1);
END;
```

2.2 USER DEFINED METHODS

Parameter Behavior

- The method can either modify one or more parameters at the call site or return a value.
- IN_OUT parameters are modified at the call site (similar to a DATA step CALL routine).

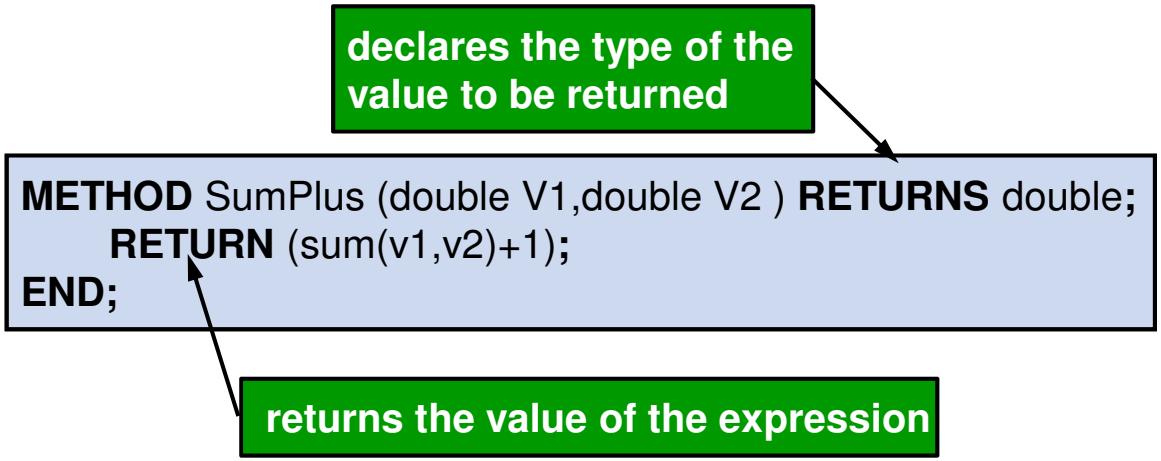
```
METHOD SumPlus (IN_OUT double V1,double V2 );
      V1=sum(v1,v2)+1;
END;
```


continued...

2.2 USER DEFINED METHODS

Parameter Behavior

- If no IN_OUT parameters are defined, the method can return a value (like a DATA step function).



5.01 Multiple Choice Poll

How many parameters are required by a method calculating SUM(**Amount**,**Amount***__Rate__) for a specified number of years?

a. 1
b. 2
c. 3
d. 4
e. other

5.01 Multiple Choice Poll – Correct Answer

How many parameters are required by a method calculating SUM(**Amount**,**Amount***Rate**) for a specified number of years?

a. 1

b. 2

c. 3

d. 4

e. other

One parameter is required for each variable in the equation, and one to specify the number of years.

2.2 USER DEFINED METHODS

Designing the Interest Method

- The interest method accepts three parameters.
  - amount
  - number of years
  - interest rate
- The value of **Amount** is modified at the call site.

  Amount=SUM(**Amount**, **Amount***Rate**)
  - The formula is executed once for every year.
- Other parameter values are not modified.

2.2 USER DEFINED METHODS

This method accepts three arguments and modifies the value of the first parameter at the call site.

```
proc ds2;
data _null_;
   method interest(in_out double Amount,double Rate,int Years);
      dcl int i;
      do i=1 to Years;
         amount=sum(Amount,Amount*Rate);
      end;
   end;
   method init();
      dcl double Total;
      dcl double Duration;
      do Duration=1 to 5 by 2;
         Total=5000;
         interest(Total,0.04,Duration);
         put Duration= Total= dollar10.2;
      end;
   end;
enddata;
run;
quit;
```

```
Duration=1 Total= $5,200.00
Duration=3 Total= $5,624.32
Duration=5 Total= $6,083.26
```

ds05d02

DETERMINE WHEN DS2 PROVIDES A PROCESSING ADVANTAGE

3.1 TRADITIONAL DATA MINING

- The increasing volume, velocity, and variety of data requires extensive data movement, which results in very poor performance.
- Computationally intensive model fitting or simulations can cause slow execution.
- Many data mining algorithms require multiple passes through the data for training models, exacerbating the performance problem.

3.2 WHY USE DS2

DS2 is beneficial in applications that

- need to use the extended data types
- can make use of packages and methods to re-use common functions or operations
- can utilize parallel processing (for computationally intense processes)
- require large data movement. Basically DS2

3.2 FINAL WORD

When to Use DS2?

DS2 programs are best suited for applications that do the following:

- take advantage of threaded processing
  - are computationally complex
  - can execute in massively parallel processing (MPP) databases
- require the precision that the DS2 data types offer
- leverage the convenient reusability of DS2 methods and packages

3.2 HANDY RESOURCES

An introduction to DS2

DS2 threaded processing

Jedi SAS Tricks by Mark Jordan at the SAS training post

Questions & Comments

EMAIL Charu.Shankar@sas.com

SAS BLOG http://blogs.sas.com/content/sastraining/author/charushankar/

TWITTER CharuYogaCan

LINKEDIN https://ca.**linkedin**.com/in/**charushankar**

www.sas.com