Lec 4

Index

- An index can help you quickly locate one or more observations that you want to read.
- Indexes provide direct access to observations in SAS data sets based on values of one or more key variables.

- Index applications include the following:

- Yield faster access to small subsets of observations for WHERE processing
- Perform table lookup operations
- Join observations

- Without an index, SAS accesses observations sequentially, in the order in which they are stored in a data set.

- An index consists of value/identifier pairs that enable you to locate an observation by value.

Types of Indexes

- You can create two types of indexes:
  - a simple index
  - a composite index

- A simple index consists of the values of one key variable, which can be character or numeric. When you create a simple index, SAS names the index after the key variable.

- A composite index consists of the values of multiple key variables, which can be character, numeric, or a combination. The values of these key variables are concatenated to form a single value.

- When you create a composite index, you must specify a unique index name that is not the name of any existing variable or index in the data set.

Creating Indexes in the DATA Step

- To create an index at the same time that you create a data set, use the INDEX= data set option in the DATA statement.

- You can create multiple indexes on a single SAS data set.

- The UNIQUE option guarantees that values for the key variable or the combination of a composite group of variables remain unique for every observation in the data set.

- In an existing data set, if the variable or variables on which you attempt to create a unique index have duplicate values, the index is not created.
- If an update tries to add an observation with a duplicate value for the index variable to that data set, the update is rejected.

```
data simple (index=(division));
set sasuser.empdata;
run;

data simple2 (index=(division empid/unique));
set sasuser.empdata;
run;

data composite (index=(Empdiv=(division empid)));
set sasuser.empdata;
run;
```

- An index is not used in these circumstances:
  - With a subsetting IF statement in a DATA step
  - With particular WHERE expressions
  - If SAS determines it is more efficient to read the data sequentially

- You can use the MSGLEVEL=N|I option to determine whether SAS used an index.

- You can use the DATASETS procedure to manage indexes on an existing data set.
  - MODIFY statement with the INDEX CREATE statement to create indexes on a data set.
  - MODIFY statement with the INDEX DELETE statement to delete indexes from a data set.
  - INDEX CREATE statement and the INDEX DELETE statement can be used in the same step.

PROC DATASETS LIBRARY= *libref* <NOLIST>**;**
MODIFY *SAS-data-set-name***;**
INDEX DELETE *index-name***;**
INDEX CREATE *index-specification***;**
QUIT;

- The INDEX CREATE statement in PROC DATASETS cannot be used if the index to be created already exists.
- PROC DATASETS executes statements in order.

```
proc datasets library=sasuser nolist;
modify sale2000;
index create origin;
quit;

proc datasets library=sasuser nolist;
modify sale2000;
index delete origin;
```

```
index create flightid;
index create Fromto=(origin dest);
quit;
```

- You can also create or delete indexes from an existing data set within a PROC SQL step.
  - The CREATE INDEX statement enables you to create an index on a data set.
  - The DROP INDEX statement enables you to delete an index from a data set.

```
proc sql;
create index origin on sasuser.sale2000(origin);
quit;

proc sql;
drop index origin from sasuser.sale2000;
create index Tofrom
on sasuser.sale2000(origin, dest);
quit;
```

- Indexes are stored in the same SAS library as the data set that they index, but in a separate SAS file from the data set.

- Index files have a member type of INDEX. There is only one index file per data set; all indexes for a data set are stored together in a single file.

- The following table describes the effects on an index file or an index file that result from several common maintenance tasks.

Task	Effect
Add an observation or observations to a data set.	Value/identifier pairs are added to the index or indexes.
Delete an observation or observations from a data set.	Value/identifier pairs are deleted from the index or indexes.
Update an observation or observations in a data set.	Value/identifier pairs are updated in the index or indexes.
Delete a data set.	The index file is deleted.
Rebuild a data set with the DATA step.	The index file is deleted.
Sort the data in place with the FORCE option in PROC SORT.	The index file is deleted.

- When you use PROC COPY to copy a data set that has an associated index, a new index file is automatically created for the new data file.

- If you use the MOVE option in the COPY procedure, the index file is deleted from the original location and rebuilt in the new location.

Note: If you copy and paste a data set in either SAS Explorer or in SAS Enterprise Guide, a new index file is automatically created for the new data file.

- Another common task is to rename an indexed data set. To preserve the index, you can use the CHANGE statement in PROC DATASETS to rename a data set. The index file is automatically renamed as well.

- In order to preserve any indexes that are associated with the data set, you can use the RENAME statement in the DATASETS procedure to rename variables when you want to rename one or more variables within an indexed data set.

  - When you use the RENAME statement to change the name of a variable for which there is a simple index, the statement also renames the index.
  - If the variable that you are renaming is used in a composite index, the composite index automatically references the new variable name.
  - However, if you attempt to rename a variable to a name that has already been used for a composite index, you receive an error message.

Example 49:
The SAS data set WORK.TEST has an index on the variable Id and the following SAS program is submitted.

```
data WORK.TEST;
  set WORK.TEST( keep=Id Var_1 Var_2 rename=(Id=Id_Code));
  Total=sum(Var_1, Var_2);
run;
```

Which describes the result of submitting the SAS program?

A. The index on Id is deleted.
B. The index on Id is updated as an index on Id_Code.
C. The index on Id is deleted and an index on Id_Code is created.
D. The index on Id is recreated as an index on Id_Code.

Example 50:
The SAS data set WORK.CHECK has an index on the variable Code and the following SAS program is submitted.

```
proc sort data=WORK.CHECK;
  by Code;
run;
```

Which describes the result of submitting the SAS program?
A. The index on Code is deleted.
B. The index on Code is updated.
C. The index on Code is uneffected.
D. The sort does not execute.

Example 51:
The SAS data set WORK.TEMP is indexed on the variable Id:

Id	Amount
P	52
P	45
A	13
A	56
R	34
R	12
R	78

The following SAS program is submitted:

```
proc print data=WORK.TEMP;
   [_insert_BY_statement_]
run;
```

Which BY statement completes the program, creates a listing report that is grouped by Id, and completes without errors?

A. by Id;
B. by Id grouped;
C. by Id descending;
D. by descending Id;

Using the IDXWHERE= and IDXNAME= Data Set Options

- In most situations, it is best to let SAS determine whether to use an index for WHERE processing.
  - If your query selects a small subset and there are multiple available indexes, you can make sure that SAS uses a particular index to process your WHERE statement.
  - You might want to force SAS to use (or not use) an index when you are benchmarking.
- The data set options IDXWHERE= and IDXNAME= control index usage.

Option	Action
IDXWHERE=	specifies whether SAS should use an index to process the WHERE expression, no matter which access method SAS estimates is faster.
	You cannot use IDXWHERE= to override the use of an index for processing a BY statement.
IDXNAME=	causes SAS to use a specific index.

Note: You can use either IDXWHERE= or IDXNAME=, but not both at the same time.

Using Multidimensional Arrays

➢ General form, multidimensional ARRAY statement:

ARRAY *array-name* {*rows,cols,...*} <*$*> <*length*><*array-elements*> <(*initial values*)>**;**

➢ When you work with arrays, remember the following:

- The name of the array must be a SAS name that is not the name of a SAS function or variable in the same DATA step.
- The variables listed as array elements must all be the same type (either all numeric or all character).
- The initial values that are specified can be numbers or character strings. You must enclose all character strings in quotation marks.

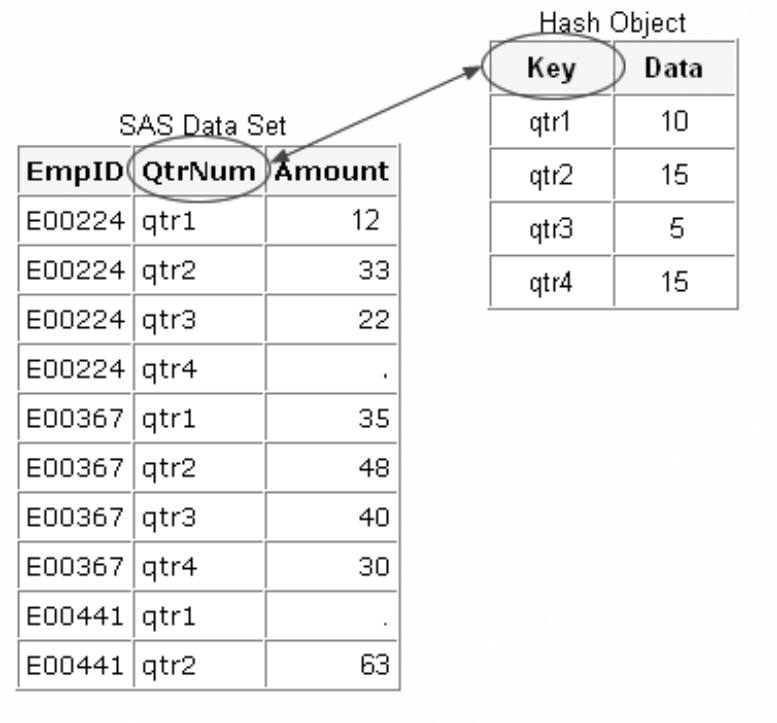
Example 52:
Which of the following ARRAY statements is similar to the statement array Yr{1974:2007} Yr1974-Yr2007; and will compile without errors?
A. array Yr{34} Yr1974-Yr2007;
B. array Yr{74:07} Yr1974-Yr2007;
C. array Yr{74-07} Yr1974-Yr2007;
D. array Yr{1974-2007} Yr1974-Yr2007;

Hash Object

Overview

➢ The hash object provides an efficient, convenient mechanism for quick data storage and retrieval.

- Hash objects can return multiple values from a given lookup.
- A hash object can be loaded from hardcoded values or a SAS data set, is sized dynamically, and exists for the duration of the DATA step.
- The hash object is a good choice for lookups that use unordered data that can fit into memory because it provides in-memory storage and retrieval and does not require the data to be sorted or indexed.

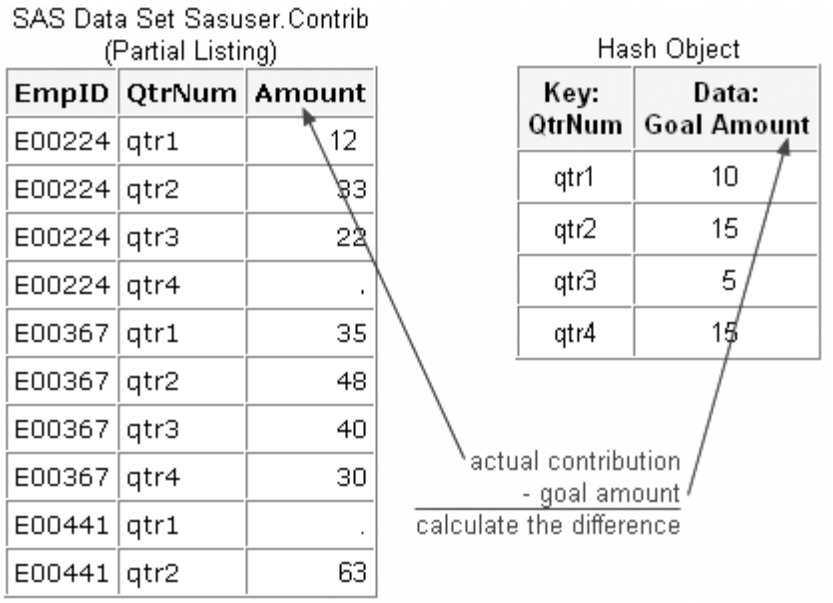


The Structure of a Hash Object

- The key component has these characteristics:
  - Might consist of numeric and character values
  - Maps key values to data rows
  - Must be unique
  - Can be a composite
- The data component has these characteristics:
  - Can contain multiple data values per key value
  - Can consist of numeric and character values

Example 53:
Which statement is true for Data step HASH objects?
A. The key component must be numeric.
B. The data component may consist of numeric and character values.
C. The HASH object is created in one step and referenced in another.
D. The HASH object must be smaller than 2 to the 8th power bytes.

- A Hash object Example



EmpID	QtrNum	Amount
E00224	qtr1	12
E00224	qtr2	33
E00224	qtr3	22
E00224	qtr4	.
E00367	qtr1	35
E00367	qtr2	48
E00367	qtr3	40
E00367	qtr4	30
E00441	qtr1	.
E00441	qtr2	63

Key: QtrNum	Data: Goal Amount
qtr1	10
qtr2	15
qtr3	5
qtr4	15

```
data work.difference (drop= goalamount);

if _N_ = 1 then do;
declare hash goal( );
goal.definekey("QtrNum");
goal.definedata("GoalAmount");
goal.definedone( );
call missing(goalamount);
goal.add(key:'qtr1', data:10 );
goal.add(key:'qtr2', data:15 );
goal.add(key:'qtr3', data: 5 );
goal.add(key:'qtr4', data:15 );
end;
set sasuser.contrib;
goal.find();
Diff = amount - goalamount;
run;
```

Using Custom Formats

Review Proc Format

- After creating a custom format, you can permanently assign the format to a variable in a DATA step, or you can temporarily specify a format in a PROC step to determine how the data values appear in output.

- Another way to assign, change, or remove the format that is associated with a variable in an existing SAS data set is to use the DATASETS procedure to modify the descriptor portion of a data set.

- General form, DATASETS procedure with the MODIFY and FORMAT statements:

PROC DATASETS LIB=*SAS-library* <NOLIST>**;**
MODIFY *SAS-data-set***;**
FORMAT *variable(s) format***;**
QUIT;

Note: The DATASETS procedure is interactive and remains in memory until you issue the QUIT statement.

```
proc datasets lib=Mylib;
modify flights;
format dest $dest.;
quit;
```

Using a Permanent Storage Location for Formats

- When you permanently associate a format with a variable in a data set, it is important to ensure that the format that you are referencing is stored in a permanent location.

- The storage location for the format is determined when the format is created in the FORMAT procedure.

- When you create formats that you want to use in subsequent SAS sessions, it is useful to take these steps:

  1. Assign the **Library** libref to a SAS library in the SAS session in which you are running the PROC FORMAT step.
  2. Specify LIB=LIBRARY in the PROC FORMAT step that creates the format.
  3. Include a LIBNAME statement in the program that references the format to assign the Library libref to the library that contains the permanent format catalog.

- You can store formats in any catalog that you choose. However, you must identify the format catalogs to SAS before you can access them.

- When a format is referenced, SAS automatically looks through the following libraries in this order:
  1. Work.Formats
  2. Library.Formats

Example 54:
In what order does SAS search for format definitions by default?
A. 1. WORK.FORMATS 2. LIBRARY.FORMATS
B. 1. LIBRARY.FORMATS 2. WORK.FORMATS
C. There is no default order, it must be defined by the user.
D. All user defined libraries that have a catalog named FORMATS, in alphabetic order.

- If you store formats in libraries or catalogs other than those in the default search path, you must use the FMTSEARCH= system option to tell SAS where to find your formats.

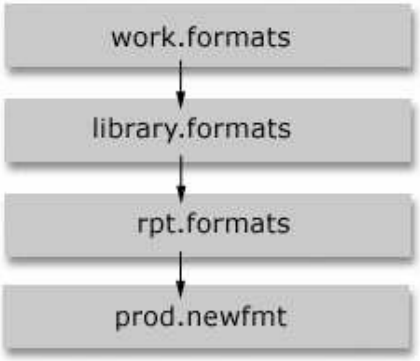
OPTIONS FMTSEARCH= *(catalog-1 catalog-2...catalog-n)*;

- The Work.Formats catalog is always searched first, and the Library.Formats catalog is searched next, unless one or both catalogs appear in the FMTSEARCH= list.

- Suppose you have formats that are stored in the Rpt library and in the Prod.Newfmt catalog. The following OPTIONS statement tells SAS where to find your formats:

```
options fmtsearch=(rpt prod.newfmt);
```

- Because no catalog is specified with the Rpt libref, the default catalog name Formats is assumed. This OPTIONS statement creates the following search order:



- Because the Work and Library librefs were not specified in the FMTSEARCH= option, they are searched in default order.

Using the FILEVAR= Option

- You can make the process of concatenating raw data files more flexible by using an INFILE statement with the FILEVAR= option.
- The FILEVAR= option enables you to dynamically change the currently opened input file to a new input file.

INFILE *file-specification* **FILEVAR**= *variable*;

- When you use an INFILE statement with the FILEVAR= option, the file specification is a **placeholder**, not an actual filename or fileref that had been assigned previously to a file.

- SAS uses this placeholder for reporting processing information to the SAS log. The file specification must conform to the same rules as a fileref.
- When the INFILE statement executes, it reads from the file that the FILEVAR= variable specifies. Like automatic variables, this variable is not written to the data set.

```
data work.quarter;
infile temp filevar=nextfile;
input Flight $ Origin $ Dest $
Date : date9. RevCargo : comma15.;
```

Assigning the Names of the Files to Read

- The next step is to assign the names of the three files to read to the variable Nextfile:

Month9.dat
Month10.dat
Month11.dat

- You can use an iterative DO loop and the PUT function to automatically change the values that are assigned to Nextfile.

```
data work.quarter;
do Month = 9, 10, 11;
nextfile="c:\sasuser\month"
!!put(Month,2.)!!".dat";
infile temp filevar=nextfile;
input Flight $ Origin $ Dest $
Date : date9. RevCargo : comma15.;
end;
```

The following table shows the value of Nextfile as the value of Month changes.

When Month=	Nextfile=
9	c:\sasuser\Month 9.dat
10	c:\sasuser\Month10.dat
11	c:\sasuser\Month11.dat

Example 55:
The following SAS program is submitted:

```
data WORK.NEW;
    do i=1, 2, 3;
      Next=cats('March' || i );
      infile XYZ
        filevar=Next
        end=Eof;
      do until (Eof);
        input Dept $ Sales;
      end;
    end;
 run;
```

The purpose of the FILEVAR=option on the INFILE statement is to name the variable Next, whose value:
A. points to a new input file.

B. is output to the SAS data set WORK.NEW.
C. is an input SAS data set reference.
D. points to an aggregate storage location.

Dictionary Tables

- Dictionary tables are commonly used to monitor and manage SAS sessions.
- Dictionary tables are special, read-only SAS tables that contain information about SAS libraries, SAS macros, and external files that are in use or available in the current SAS session.
- Dictionary tables also contain the settings for SAS system options and SAS titles and footnotes that are currently in effect.
- For example, the Dictionary.Columns table contains information (such as name, type, length, and format) about all columns in all tables that are known to the current SAS session.

Example 56:
Which dictionary table provides information on each occurrence of the variable named LastName?
A. DICTIONARY.TABLES
B. DICTIONARY.COLUMNS
C. DICTIONARY.MEMBERS
D. DICTIONARY.VARIABLES

Dictionary tables are

- created each time they are referenced in a SAS program
- updated automatically
- limited to Read-Only access.

Dictionary table	Sashelp view	Contains
Catalogs	Vcatalg	information about catalog entries
Columns	Vcolumn	detailed information about variables and their attributes
Extfiles	Vextfl	currently assigned filerefs
Indexes	Vindex	information about indexes defined for data files
Macros	Vmacro	information about both user and system defined macro variables
Members	VmemberVsaccesVscatlgVslibVstableVstabvwVsview	general information about data library members
Options	Voption	current settings of SAS system options
Tables	Vtable	detailed information about data sets
Titles	Vtitle	text assigned to titles and footnotes
Views	Vview	general information about data views