

## **Randomly Accessing Files Using the KEY= Option**

*Kim L. Kolbe Ritzow of Systems Seminar Consultants, Kalamazoo, MI*

### **Abstract**

The KEY= option on the SET or MODIFY statement is new starting with Version 6.07 of the BASE SAS® System and allows direct access to an observation nonsequentially based an indexed value.

This paper will provide examples as to how the KEY= option is used and compare this type of table lookup technique with more traditional methods in terms of efficiency.

### **Background**

The KEY= option is used to locate observations in one file based on the values in another file. This is also often referred to as a table lookup.

Other methods that could be used instead of the KEY= option to perform a table lookup would be to use an UPDATE statement, or a SAS MERGE statement in conjunction with an IN= operator, or perform a similar type of join by using PROC SQL. While PROC FORMAT and arrays are also often used to perform table lookups, they only retrieve a single value at a time, whereas the KEY= option, a SAS MERGE, an UPDATE statement and PROC SQL statements can retrieve more than one value at a time. The use of PROC FORMAT and arrays will not be discussed as an alternative table lookup method for the purpose of this paper. (For a detailed discussion of these various methods see Dana Rafiee's article in the 'Related Literature' section at the end of this paper).

### **Preparing the Data**

In order to use the KEY= option, the data set must first be indexed. It is not required that both files be indexed, only the file upon which values will be looked up must be indexed. In SAS, an index can be created one of several ways:

- ☐ PROC DATASETS
- ☐ PROC SQL
- ☐ a SAS data set option (INDEX=)
- ☐ via some Display Manager Windows
- ☐ SAS/IML software
- ☐ SCL (Screen Control Language) available in SAS/AF and SAS/FSP products

PROC DATASETS, PROC SQL, or the INDEX= SAS data set option are probably the most common methods used to create indexes on a SAS data set:

```
PROC DATASETS LIB=SASUSER;  
  MODIFY SALES_86;  
  INDEX CREATE ACCT_NUM;  
RUN;
```

(or)

```
PROC SQL;  
  CREATE INDEX ACCT_NUM  
  ON SASUSER.SALES_86(ACCT_NUM);  
QUIT;
```

(or)

```
DATA SASUSER.SALES_86  
  (INDEX=(ACCT_NUM));  
  INFILE.....;  
  INPUT....;  
RUN;
```

There are two types of indexes in SAS: **simple** and **composite**. A simple index is when variables are indexed independently of one another. For example, ACCT\_NUM, SALESID, and STATE could be indexed but each would be indexed separately from one another. In the case of simple indexes, there name is the same as the variable name. A composite index, however, uses a unique name that refers to two or more variables together that make up an index as a whole, or composite. For example, a composite index called CUSTID could index upon STATE, BRANCH, and REFNO as a single entity. Examples of how composite indexes are created and used can be found in the BASE SAS® manuals. Composite indexes will not be discussed in further detail in the course of this paper.

### **Special Considerations Regarding Indexing**

Indexes are special pointers that allow, in most cases, access data values more quickly. Because not every SAS data set is a good candidate for

indexing, it is highly recommended that you read some of the literature relating to the guidelines for indexing SAS data sets listed in the 'Related Literature' section at the end of this paper.

While indexing supposedly allows faster access to a small subset of data, it has been noted that indexing a SAS data set under certain circumstances can actually reduce the efficiency rather than improve it. It is important to note that indexed data sets sometimes:

- ❑ are more CPU and I/O intensive in order to create and maintain the index
- ❑ require extra memory to load the index table
- ❑ require extra disk space on a separate file to store the index's data structure
- ❑ require extra resources to rebuild indexes for a new version of a data set with indexes

Whether or not a SAS data set should be indexed depends on the structure of the data set itself, the version of SAS being used, and platform in which SAS is being run under. Careful testing should be performed on each data set to be indexed before deciding on if the data set should be permanently indexed or not.

#### Using the KEY= Option

Once the lookup data set has been indexed, the KEY= option can then be used to locate select information.

The KEY= option works much like the POINT= option in that it allows us nonsequential access to our data sets. The main difference between the two is that the POINT= option allows us to access our data nonsequentially based on an observation number, whereas the KEY= option allows us access to our data nonsequentially based on a value.

When the KEY= option is specified on a SET statement it will read only the records from the lookup data set (in our example, the SASUSER.SALES\_86 data set) that are also found on the search file (the FINDIT data set). The result of this lookup will be a file that contains all

the information from both data sets for those records that match:

| SASUSER.SALES_86 |          |     |
|------------------|----------|-----|
| OBS              | ACCT_NUM | AGE |
| 1                | 13388383 | 23  |
| 2                | 33322828 | 55  |
| 3                | 32818181 | 42  |

| FINDIT |          |       |
|--------|----------|-------|
| OBS    | ACCT_NUM | T_JOB |
| 1      | 13388383 | 12    |
| 2      | 33322828 | 31    |

```
PROC DATASETS LIB=SASUSER;
  MODIFY SALES_86;
  INDEX CREATE ACCT_NUM;
RUN;
```

```
DATA COMBINE;
  SET FINDIT(KEEP=ACCT_NUM T_JOB)
  SET SASUSER.SALES_86(KEEP=AGE)
  KEY=ACCT_NUM;
  IF _IORC_=0 THEN OUTPUT;
RUN;
```

```
PROC PRINT DATA=COMBINE;
RUN;
```

| COMBINE |          |       |     |
|---------|----------|-------|-----|
| OBS     | ACCT_NUM | T_JOB | AGE |
| 1       | 13388383 | 12    | 23  |
| 2       | 33322828 | 31    | 55  |

Using the KEEP= or DROP= option behind the data set name not only controls which variables will appear on the resulting output SAS data set, but it will also improve the efficiency of the lookup process. KEY= variables do not need to be specified on the KEEP= statement because they are automatically kept.

The order in which the data sets are specified on the SET statements IS IMPORTANT. The first

data set specified should be the search data set (FINDIT) and the second data set specified should be the lookup data set (SASUSER.SALES\_86). If the order of the SET statements are switched, you WILL get output, but it WILL NOT be correct:

```
DATA COMBINE;
  SET SASUSER.SALES_86 KEY=ACCT_NUM;
  PUT ALL;
  SET FINDIT;
  PUT '***' ALL;
  IF _IORC_ = 0 THEN OUTPUT COMBINE;
  ELSE _ERROR_ = 0;
RUN;
```

```
PROC PRINT DATA=COMBINE;
RUN;
```

| COMBINE |          |       |     |
|---------|----------|-------|-----|
| OBS     | ACCT_NUM | T_JOB | AGE |
| 1       | 33322828 | 31    | 23  |

The result produced here is the last record from the search data set matched with the first record from the lookup data set.

#### What Happened?

By looking at the results of the PUT ALL in our SAS LOG, it helps explain why the order of the SET statements are important:

```
ACCT_NUM= AGE= T_JOB= _ERROR_=1
_IORC_ =1230015 N =1
***ACCT_NUM=13388383 AGE= T_JOB=12 _ERROR_=1
_IORC_ =1230015 N =1
ACCT_NUM=13388383 AGE=23 T_JOB=12 _ERROR_=0
_IORC_ =0 N =2
***ACCT_NUM=33322828 AGE=23 T_JOB=31 _ERROR_=0
_IORC_ =0 N =2
ACCT_NUM=33322828 AGE=55 T_JOB=31 _ERROR_=0
_IORC_ =0 N =3
```

Because no value has been read in for the KEY = value, the result of the first PUT ALL the first time through the Data Step is a dump of the PDV in the SAS LOG with all variable's values set to missing. Values are set into these variables after the data set on the second SET statement has

been read (AGE is not present on this data set, so its value is still set to missing). Because the I/O process did not complete successfully, the value of \_IORC\_ is something other than zero (the purpose of the \_IORC\_ variable is discussed in greater detail below) and therefore this record is not output to the data set, but the value of \_ERROR\_ for this record will be reset to zero. The second time through the DATA Step, the values from the previous iteration are automatically retained and age is now added as a result of the first SET statement. After the second SET statement is read a new KEY = and T\_JOB value are read and the record is then output to the SAS data set because the most recent I/O process completed successfully (\_IORC\_ = 0). Be careful, just because a record is successful it does not necessarily mean it is correct, as is demonstrated in this example. Upon the third iteration of the DATA Step, the values for ACCT\_NUM and T\_JOB are being retained with a new value for AGE being provided. Because the implicit end-of-file indicator has been reached, the process ends.

#### Lessons To Be Learned

The two lessons to be learned by this example are this: first, the order in which the data sets are specified on the SET statements is important (the data set with the KEY = option should always be the second one specified). And secondly, just because you are outputting records only when \_IORC\_ = 0 does not mean you will always get only the perfect matches- so be careful!

#### The \_IORC\_ Pseudo Variable

The \_IORC\_ pseudo variable is automatically created whenever the MODIFY statement or a SET statement with the KEY = option is used. It helps determine if the most recent I/O operation performed completed successfully or not. If the \_IORC\_ pseudo variable returns a value of 0, the process has completed successfully; any value other than 0 indicates the most recent I/O process was unsuccessful.

This is not foolproof though, notice in the previous example even though the value of the \_IORC\_ pseudo variable was checked for 0 (to determine if the I/O process was successful or not) prior to outputting or writing the record out, it still produced incorrect results. In this case, in addition to only writing out those records where

the value of `_IORC = 0` you should also be checking your SAS LOG. Always check the SAS LOG whenever using the `KEY =` option on the `MODIFY` or `SET` statements because if there is a problem in the table lookup process a dump of the PDV will be present. Even if only those records where `_IORC = 0` are being output NEVER trust the results of the output SAS data set if a dump of the PDV is present in the SAS LOG when using the `KEY =` option.

An automatic way to prevent these types of problems would be to do something like the following:

```
DATA COMBINE;  
  SET SASUSER.SALES_86 KEY=ACCT_NUM;  
  SET FINDIT;  
  IF _IORC_ NE 0 THEN ABORT; * OR STOP;  
RUN;
```

The `_IORC` pseudo variable is typically used in conjunction with the `SYSRC` autocall macro to test specific I/O error conditions. See 'Related Literature' for more information on this feature.

#### **Duplicate Records and Records Not Found**

The lookup table (`SASUSER.SALES_86`) should not contain duplicate records. While duplicate records are allowed in the lookup table, only the first indexed value found will be used in the table lookup process- other duplicate records in the lookup table will be ignored. SAS will build an output SAS data set, the SAS LOG will NOT contain a dump of the PDV, and no error or warning messages will be present. Yet, the output SAS data set created may not contain entirely correct information. Because of this, I strongly recommend that whenever creating an index on a data set that is to be used specifically with the `KEY =` option that the `UNIQUE` option be defined upon creating the index. The `UNIQUE` option prevents an index from being created if duplicate index values exist. Duplicate records will have to be eliminated from the data set by using `FIRST./LAST.` logic or by using the `NODUPKEY` option on `PROC SORT` before the index can be created when this option is in effect:

```
PROC DATASETS LIB=SASUSER;  
  MODIFY SALES_86;  
  INDEX CREATE ACCT_NUM/UNIQUE;  
RUN;
```

Similarly, if it is the search data set (`FINDIT`) that contains the duplicate records rather than the lookup data set, the same holds true: only the first record in the search data set will be matched with the corresponding record in the lookup table. However, in this scenario, unlike the previous one, a dump of the PDV will be present in the SAS LOG indicating an I/O process has not completed successfully. Hopefully, if you are in the habit of always checking your SAS LOG, this scenario would be detected by the presence of the PDV in the LOG which may prompt you to correct the lookup process with one of the two methods listed below.

One way to correct this is to simply eliminate the duplicate records from the search data set through a de-duping process (discussed above). Another possibility is that if you wish to keep the duplicate records in the search data set and match each of the duplicate records with the corresponding record in the lookup data set, then use the `UNIQUE` option specified on the `KEY =` option (this is NOT the same as the `UNIQUE` option used in the example above when creating indexes- see discussion below for further details on the `UNIQUE` option).

Duplicates in both files should be avoided in all cases. When both files contain the same number of duplicate indexed values, they will match up and the results will look reasonable. If, however, one file contains more records for a particular indexed value than the other file, you will get some weird results and your SAS LOG will not indicate that anything is wrong. Duplicates should be especially eliminated from the lookup table (`SASUSER.SALES_86`). If you choose not to eliminate the duplicates from the search file (`FINDIT`) then be sure to use the `UNIQUE` option on the `KEY =` option before performing the lookup process in order to get correct results.

When a key value is located on the search data set, but cannot be found on the indexed lookup data set, a dump of the PDV will be present in the SAS LOG, but the resulting data created set should be correct in terms of information. See Table 1 for a summarization of the information.

#### **The UNIQUE Option**

Starting with Version 6.08, the `UNIQUE` option is available on the `KEY =` statement (don't confuse

it with the UNIQUE option used in creating indexes- they are NOT the same). The UNIQUE option on the KEY= statement specifies that the search should begin at the top of the index for the data set being read rather than the default which is to start at the top of the index for each new KEY= value.

This option is particularly useful as noted above when the search data set has multiple observations with the same value that are to be located in the lookup data set. By default, without this option, only the first value from the search data set will be matched with its corresponding record on the lookup data set- duplicate records will be ignored because it does not start the search over from the top. However, with the option specified, multiple records for the same value on the search data set will be matched up with its corresponding value from the lookup data set.

#### KEY= Option on the MODIFY Statement

The MODIFY statement is new starting with Version 6.07 of the SAS System. The MODIFY main difference between the MODIFY statements and other comparable SAS statements is that the MODIFY statement allows for in place data set manipulation rather than creating an additional copy of the data sets. Therefore, the use of the MODIFY statement saves a substantial amount of temporary work space, particularly if the data sets being manipulated are large.

The one major downside to this statement is that because it is in place data set manipulation, if the system should terminate abnormally during an updating process, it can result in an unrecoverable loss of data.

In earlier examples we saw how coding two SET statements with a KEY= option on the lookup data set works more like a merge in that it "joins" information together to create an output data set containing all the information from both data sets. The use of the KEY= option on the MODIFY statement, however, is different in that it is used to update or replace information existing information on the lookup data set.

While the MODIFY statement can do many different things, I am only going to show an example that specifically deals with the MODIFY statements as it relates to using the KEY= option:

| SASUSER.SALES_86 |          |     |
|------------------|----------|-----|
| OBS              | ACCT_NUM | AGE |
| 1                | 13388383 | 23  |
| 2                | 33322828 | 55  |
| 3                | 32818181 | 42  |

| FINDIT2 |          |     |
|---------|----------|-----|
| OBS     | ACCT_NUM | AGE |
| 1       | 13388383 | 15  |
| 2       | 33322828 | 36  |

```
DATA SASUSER.SALES_86;
  SET FINDIT2(RENAME=(AGE=NAGE)
    KEEP=AGE ACCT_NUM);
  MODIFY SASUSER.SALES_86
    KEY=ACCT_NUM;
  AGE=NAGE;
RUN;
```

| SASUSER.SALES_86 |          |     |
|------------------|----------|-----|
| OBS              | ACCT_NUM | AGE |
| 1                | 13388383 | 15  |
| 2                | 33322828 | 36  |
| 3                | 32818181 | 42  |

The master data set (SASUSER.SALES\_86) still contains all its original records, but for those records that were present in the transaction file (FINDIT2) the master file has now been modified or updated accordingly based on the values found in the transaction data set; those records not present in the transaction data set remain on the master file unchanged.

When using the MODIFY statement and KEY= option, it is important to rename the variable on the transaction data set to something different so it can be uniquely referred to when explicitly defining the updating action to take place.

When using the MODIFY statement, be careful about the presence of duplicate records in one or both files. For best results, always de-dup both of the data sets before performing the MODIFY statement with a KEY = option.

#### Other Methods

PROC SQL or a SAS MERGE statement can also be used to perform a similar type of table lookup as was done with the SET statements and KEY = option:

```
DATA MERGE;
  MERGE SASUSER.SALES_86(IN=ONA
    KEEP=ACCT_NUM AGE)
    FINDIT (IN=ONB
    KEEP=ACCT_NUM T_JOB);
  BY ACCT_NUM;
  IF ONA AND ONB;
RUN;
```

(or)

```
PROC SQL;
  CREATE TABLE MERGE AS
  SELECT B.ACCT_NUM, AGE, T_JOB
  FROM SASUSER.SALES_86 A
    RIGHT JOIN FINDIT B
    ON A.ACCT_NUM=B.ACCT_NUM;
QUIT;
```

The following SAS MERGE or UPDATE statement performs a similar type of modifying or updating of information as was done with the SET and MODIFY statements using the KEY = option:

```
DATA SASUSER.SALES_86;
  MERGE SASUSER.SALES_86
    FINDIT2(KEEP=ACCT_NUM AGE);
  BY ACCT_NUM;
RUN;
```

(or)

```
DATA SASUSER.SALES_86;
  UPDATE SASUSER.SALES_86
    FINDIT2(KEEP=ACCT_NUM AGE);
  BY ACCT_NUM;
RUN;
```

#### Comparing Efficiencies

Using an IBM 3290, I ran some tests comparing the various table lookup methods. Table 2 summarizes the findings and explains the sampling methodology used.

In terms of performing a table lookup using the SET statement and KEY = option versus using a SAS MERGE or PROC SQL, I found that PROC SQL was the most efficient in performing the table lookup when the data set was not indexed. If the data set was already indexed, then the KEY = option on the SET statement is A LOT more efficient than the other two methods (on average .03 CPU seconds to perform the table lookup with SET and KEY = versus .46 CPU seconds for PROC SQL and .54 CPU seconds for the MERGE statement). The most expensive part of using the SET and KEY = is the building of the index, which will take on average up to 95 percent of the total job time to build it, the other 5 percent of the job time is spent actually performing the lookup itself.

Another interesting note, with the MERGE and PROC SQL statements, CPU time increased the further down the search records were in the indexed lookup table, whereas with the KEY = option, CPU time remained constant regardless of where the search records were located in the indexed lookup table.

Similarly, the use of the SET/MODIFY statements with the KEY = option when performing an update was the most efficient method only when the data set was already indexed. If the data set was not indexed, then the MERGE statement was more efficient in performing the update/modification than the UPDATE statement, which kind of surprised me.

These findings suggest that if your data set is already indexed, the KEY = option should be used to perform table lookups and updates, but an index should not be created solely for the purpose of using the KEY = option because it may end up using more resources. When the data set is not previously indexed, it is more efficient to use PROC SQL to perform a table lookup and a MERGE statement to perform an update.

### **In Recognition**

A very special thanks to Denise Moorman at the SAS Institute for her help in clarifying some of the technical issues relating to the use of the KEY= option.

### **Related Literature**

SAS Indexes (how to create them, types, criteria, guidelines, managing them, and special considerations), BASE SAS LANGUAGE Reference Version 6 First Edition, pgs. 217-224.

Creating Indexes with the SQL Procedure (how to create them, types, special considerations, precautions), SAS Guide to the SQL Procedure, Usage and Reference Version 6 First Edition, pgs. 102-104.

KEY= Option on the SET and MODIFY Statements (how it works), SAS Technical Report P-222 Changes and Enhancements to BASE SAS Software, Version 6.07, pgs. 20-21.

Judy Loren, "Using Indexes for Direct Access to SAS Datasets", Proceedings of the 8th Annual NESUG Conference, pgs. 281-282.

SYSRC Autocall Facility Macro, SAS Technical Report P-222 Changes and Enhancements to BASE SAS Software, Version 6.07, pgs. 318-319.

Dana Rafiee, "NO MORE MERGE- Alternative Table Lookup Techniques", Proceedings of the 21st Annual SAS<sup>®</sup> Users Group International Conference, pgs. 747-752.

David Sommer, "Efficient Techniques for Random Access to SAS Datasets", Proceedings of the 20th Annual SAS Users Group International Conference, pgs. 262-267.

The MODIFY Statement (syntax/usage), SAS Technical Report P-222 Changes and Enhancements to BASE SAS Software, Version 6.07, pgs. 36-42.

### **Trademark Notice**

SAS is a registered trademark of the SAS Institute Inc., Cary, NC, USA and other countries.

Any questions or comments regarding the paper may be directed to the author:

Kim L. Kolbe Ritzow  
Systems Seminar Consultants  
Kalamazoo Office  
927 Lakeway Drive  
Kalamazoo, MI 49001  
Phone: (616) 345-6636  
Fax: (616) 345-5793  
E-mail: KRITZOW@AOL.COM

| <b>Indexed Table<br/>(SASUSER.SALES_86)</b> | <b>"Search" Table<br/>(FINDIT)</b>                               | <b>RESULT</b>                                                                                                                                                                                                                                                                     |
|---------------------------------------------|------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| multiple records for each indexed value     | one record for each search value                                 | Only the first record for each indexed value is taken. Ideally, duplicates should be eliminated prior to indexing through a de-duping process, or, when creating the index, the UNIQUE option should be used to prevent the index from being created if duplicates exist.         |
| one record for each indexed value           | multiple records for each search value                           | Only the first record from the search data set is taken. By default, duplicate records in the search data set are ignored unless the UNIQUE option is used on the KEY= option (this is NOT the same as the UNIQUE option used when creating the index– they are very different!). |
| multiple records for each indexed value     | multiple records for each search value                           | Scenario should be avoided at all costs!!! You will get output, but it will not be correct.                                                                                                                                                                                       |
| one record for each indexed value           | contains a search record that does not exist on the lookup table | You will receive correct output, but there will be dump of the PDV in the SAS LOG noting that a record could not be located in the lookup data set                                                                                                                                |

**Table 1**

**Table 2: Summarized Findings Comparing the Various Lookup Techniques**

| Statement Used                   | 10,000                    | 25,000 | 50,000 |
|----------------------------------|---------------------------|--------|--------|
| SET statement<br>and KEY= option | .71 CPU sec. <sup>1</sup> | 1.67   | 3.30   |
| MERGE statement                  | .54                       | 1.23   | 2.58   |
| PROC SQL                         | .46                       | .90    | 2.35   |

<sup>1</sup> .68, 1.64, and 3.27 CPU seconds, respectively, of which were used to create the index, .03 CPU seconds to perform the actual lookup.

Tests were run on an IBM 3290.

10 tests were run for each scenario at different times of the day and CPU seconds were averaged.

In each case 10 records were searched for in a data set containing 10,000, 25,000, and 50,000 records.

Varying where the 10 records were located in the indexed lookup table (that is, if they happened to be located at the top, middle, or near the end of the file) did increase the amount of CPU time required for the MERGE and PROC SQL statements to locate the records, but did not appear to make a difference when using the KEY= option; CPU time needed remained constant in this case.

**Table 3: Summarized Findings Comparing the Various Modifying/Updating Techniques**

| Statement Used                           | 10,000                    | 25,000 | 50,000 |
|------------------------------------------|---------------------------|--------|--------|
| SET/MODIFY statements<br>and KEY= option | .71 CPU sec. <sup>1</sup> | 1.58   | 3.19   |
| MERGE statement                          | .55                       | 1.26   | 2.52   |
| UPDATE statement                         | .55                       | 1.29   | 2.57   |

<sup>1</sup> .68, 1.55, and 3.16 CPU seconds, respectively, of which were used to create the index, .03 CPU seconds to perform the actual lookup.

Tests were run on an IBM 3290.

10 tests were run for each scenario at different times of the day and CPU seconds were averaged.

In each case 10 records were searched for in a data set containing 10,000, 25,000, and 50,000 records.

Varying where the 10 records were located in the indexed lookup table (that is, if they happened to be located at the top, middle, or near the end of the file) did increase the amount of CPU time required for the MERGE and UPDATE statements to locate the records, but did not appear to make a difference when using the KEY= option; CPU time needed remained constant in this case.