

**Chapter
1**

**TABLE
HANDLING**

*Get on the
Fast Track!*



TM

**SYS-ED/
Computer
Education
Techniques, Inc.**

Objectives

You will learn:

- Why and when it is appropriate to use a table.
- Table handling capabilities of the COBOL programming language.
- Data Division considerations for tables.
- Subscripting.
- Indexing.
- SET statement.
- Performing a binary search of a table.
- SEARCH statement.

1 Tables - Purpose and Facilities

A table is a collection of data items with common attributes.

A table consists of a table name and subordinate items referred to as table elements. A table is the COBOL equivalent of an array.

The OCCURS clause in the DATA DIVISION is used for defining a table.

The advantages associated with this practice include:

- The code presents the unity of the table elements.
- Subscripts and indexes can be used for referring to the table elements.
- It is easy to repeat data items.

In order to code a table, designate a group name for the table and define the subordinate item to be repeated n times.

Example:

```
01 TABLES-SKILLS.  
   05 TBL-SKILLS          OCCURS 30 TIMES.  
       10 TB-SOCSEC      PIC X(9).  
       10 TB-NAME        PIC X(25).
```

The TABLES-SKILLS is the name of an alphanumeric group item. The table element definition, which includes the OCCURS clause, is subordinate to the group item that contains the table. The OCCURS clause cannot appear in a level-01 description.

In order to create tables of two to seven dimensions, use nested OCCURS clauses.

In order to create a variable-length table, code the DEPENDING ON phrase of the OCCURS clause.

2 Accessing an Item in a Table

A table element has a collective name; the individual items within it do not have unique data-names.

There are two techniques for referring to an item.

1. Use the data-name of the table element, along with its occurrence number in parentheses; the occurrence number is referred to as a subscript.

This technique is subscripting.

2. Use the data-name of the table element, along with an index that is added to the address of the table to locate an item. The index added to this address is the displacement from the beginning of the table.

This technique is indexing; it also is referred to as subscripting using index-names.

3 Subscripting

The lowest possible subscript value is 1, which references the first occurrence of a table element.

In a one-dimensional table, the subscript corresponds to the row number. A literal or a data-name can be used as a subscript.

When a data-name is used as a variable subscript, the data-name must be defined as an elementary numeric integer.

The most efficient format is COMPUTATIONAL (COMP) with a PICTURE size that is smaller than five digits.

A literal or variable subscript can be incremented or decremented by a specified integer amount.

Example:

```
TABLE-COLUMN (SUB1 - 1, SUB2 + 3)
```

Instead of changing the entire element, it is possible to change part of a table element. This is done by referring to the character position and length of the substring to be changed.

Example:

```
01 MY-TABLE.  
   05 TABLE-ELEMENT      PIC X(9)OCCURS 3 TIMES.  
   . . .  
  
   MOVE '00' TO TABLE-ELEMENT (1) (3 : 2).
```

This MOVE will set the first row starting at the third character for a length of two to the value '00'.

4 Indexing

An index is created by using the INDEXED BY phrase of the OCCURS clause to identify an index-name.

Example:

The INX-A in this code is an index-name:

```
05 TABLE-ITEM PIC X(8)
   OCCURS 10 INDEXED BY INX-A.
```

The compiler calculates the value contained in the index as the occurrence number (subscript) minus 1, multiplied by the length of the table element.

The fifth occurrence of TABLE-ITEM is the binary value contained in INX-A is $(5 - 1) * 8$, or 32.

An index-name can be used for referencing another table only if both table descriptions have the same number of table elements and the table elements are of the same length.

The USAGE IS INDEX clause can be used for creating an index data item and an index data item with any table.

Example:

INX-B is an index data item in this code:

```
01 INX-B  USAGE IS INDEX.
. . .
   SET INX-A TO 10
   SET INX-B TO INX-A.
   PERFORM VARYING INX-A FROM 1 BY 1 UNTIL INX-A > INX-B
       DISPLAY TABLE-ITEM (INX-A)
       . . .
   END-PERFORM.
```

5 Loading a Table Dynamically

If the initial values of a table are different with each execution of a program, then the table can be defined without initial values. The changed values can be read into the table dynamically before the program refers to the table.

In order to load a table, use the PERFORM statement and either subscripting or indexing.

When reading data to load a table, ensure and test that the data does not exceed the space allocated for the table.

Use a named value rather than a literal for the maximum item count. In this way, if the table is made larger, it will be necessary to change only one value instead of all references to a literal.

A table can be loaded by coding one or more INITIALIZE statements.

6 Sample Program

```
DATA DIVISION.
FILE SECTION.
FD STUDENTFILE.
01 STUDENTDETAILS.
   02 STUDENTID          PIC 9(7).
   02 STUDENTNAME.
     03 SURNAME          PIC X(8).
     03 INITIALS        PIC XX.
   02 DATEOFBIRTH.
     03 YOBIRTH         PIC 9(4).
     03 MOBIRTH         PIC 9(2).
     03 DOBIRTH         PIC 9(2).
   02 COURSECODE        PIC X(4).
   02 GENDER            PIC X.

WORKING-STORAGE SECTION.
01 MONTHTABLE.
   02 TABLEVALUES.
     03 FILLER          PIC X(18) VALUE 'JANUARY FEBRUARY'.
     03 FILLER          PIC X(18) VALUE 'MARCH APRIL'.
     03 FILLER          PIC X(18) VALUE 'MAY JUNE'.
     03 FILLER          PIC X(18) VALUE 'JULY AUGUST'.
     03 FILLER          PIC X(18) VALUE 'SEPTEMBER OCTOBER'.
     03 FILLER          PIC X(18) VALUE 'NOVEMBER DECEMBER'.
   02 FILLERH REDEFINES TABLEVALUES.
     03 MONTH OCCURS 12 TIMES PIC X(9).

01 MONTHCOUNT OCCURS 12 TIMES PIC 999 VALUE ZEROS.

01 MONTHIDX            PIC 999.

01 HEADINGLINE        PIC X(19) VALUE ' MONTH STUDCOUNT'.

01 DISPLAYLINE.
   02 PRNMONTH         PIC X(9).
   02 FILLER           PIC X(4) VALUE SPACES.
   02 PRNSTUDENTCOUNT PIC ZZ9.

01 EOF-FLAG           PIC X VALUE LOW-VALUES.
88 ENDOFSTUDENTFILE  VALUE HIGH-VALUES.
```

```
PROCEDURE DIVISION.  
BEGIN.  
  OPEN INPUT STUDENTFILE  
  READ STUDENTFILE  
    AT END SET ENDOFSTUDENTFILE TO TRUE  
  END-READ  
  
  PERFORM UNTIL ENDOFSTUDENTFILE  
    ADD 1 TO MONTHCOUNT(MOBIRTH)  
    READ STUDENTFILE  
      AT END SET ENDOFSTUDENTFILE TO TRUE  
    END-READ  
  END-PERFORM  
  
  DISPLAY HEADINGLINE  
  
  PERFORM VARYING MONTHIDX FROM 1 BY 1 UNTIL MONTHIDX > 12  
    MOVE MONTH(MONTHIDX) TO PRNMONTH  
    MOVE MONTHCOUNT(MONTHIDX) TO PRNSTUDENTCOUNT  
    DISPLAY DISPLAYLINE  
  END-PERFORM.  
  
  CLOSE STUDENTFILE  
  STOP RUN.
```

7 Sample Program

```
DATA DIVISION.
FILE SECTION.
FD  LOCATION-FILE
   RECORDING MODE F
   BLOCK 0 RECORDS
   RECORD 80 CHARACTERS
   LABEL RECORD STANDARD.
01  LOCATION-RECORD.
   05  LOC-CODE                PIC XX.
   05  LOC-DESCRIPTION         PIC X(20).
   05  FILLER                  PIC X(58).
. . .
WORKING-STORAGE SECTION.
01  FLAGS.
   05  LOCATION-EOF-FLAG      PIC X(5) VALUE SPACE.
       88  LOCATION-EOF      VALUE 'YES'.
01  MISC-VALUES.
   05  LOCATION-TABLE-LENGTH  PIC 9(3) VALUE ZERO.
   05  LOCATION-TABLE-MAX     PIC 9(3) VALUE 100.

01  LOCATION-TABLE.
   05  LOCATION-CODE OCCURS 1 TO 100 TIMES
       DEPENDING ON LOCATION-TABLE-LENGTH  PIC X(80).
. . .
PROCEDURE DIVISION.
. . .
PERFORM TEST AFTER
   VARYING LOCATION-TABLE-LENGTH FROM 1 BY 1
   UNTIL LOCATION-EOF
   OR LOCATION-TABLE-LENGTH = LOCATION-TABLE-MAX
MOVE LOCATION-RECORD TO
   LOCATION-CODE (LOCATION-TABLE-LENGTH)
READ LOCATION-FILE
   AT END SET LOCATION-EOF TO TRUE
END-READ
END-PERFORM
```

8 Variable-length Tables

If before run time it is not known how many times a table element occurs, then a variable-length table needs to be defined.

The OCCURS DEPENDING ON (ODO) clause can be used to define the variable-length table.

Example:

```
X OCCURS 1 TO 10 TIMES DEPENDING ON Y
```

In this code, X is the ODO subject, and Y is the ODO object.

Example:

This code demonstrates a group item (REC-1) that contains both the subject and object of the OCCURS DEPENDING ON clause. The way the length of the group item is determined depends on whether it is sending or receiving data.

```
WORKING-STORAGE SECTION.  
01 MAIN-AREA.  
    03 REC-1.  
        05 FIELD-1 PIC 9.  
        05 FIELD-2 OCCURS 1 TO 5 TIMES  
            DEPENDING ON FIELD-1 PIC X(05).  
01 REC-2.  
    03 REC-2-DATA PIC X(50).
```

If there is a requirement to move REC-1, which is the sending item, to REC-2, the length of REC-1 is determined immediately before the move, using the current value in FIELD-1.

If the content of FIELD-1 conforms to its PICTURE clause and contains a zoned decimal item, the move can proceed based on the actual length of REC-1. Otherwise, the result will be unpredictable.

Before initiating the move, it will be necessary to ensure that the ODO object has the correct value.

When performing a move to the receiving item REC-1, the length of REC-1 is determined using the maximum number of occurrences.

Example:

In this program, five occurrences of FIELD-2, plus FIELD-1, yield a length of 26 bytes. In this case, it will not be necessary to set the FIELD-1 ODO object before referencing REC-1 as a receiving item.

The sending field's ODO object will need to be set to a valid numeric value between 1 and 5 for the ODO object of the receiving field to have been validly set by the move.

```
DATA DIVISION.
FILE SECTION.
FD  LOCATION-FILE
   RECORDING MODE F
   BLOCK 0 RECORDS
   RECORD 80 CHARACTERS
   LABEL RECORD STANDARD.
01  LOCATION-RECORD.
   05  LOC-CODE                PIC XX.
   05  LOC-DESCRIPTION        PIC X(20).
   05  FILLER                 PIC X(58).
   . . .
WORKING-STORAGE SECTION.
01  FLAGS.
   05  LOCATION-EOF-FLAG      PIC X(5) VALUE SPACE.
   88  LOCATION-EOF          VALUE 'YES'.
01  MISC-VALUES.
   05  LOCATION-TABLE-LENGTH  PIC 9(3) VALUE ZERO.
   05  LOCATION-TABLE-MAX     PIC 9(3) VALUE 100.

01  LOCATION-TABLE.
   05  LOCATION-CODE OCCURS 1 TO 100 TIMES
       DEPENDING ON LOCATION-TABLE-LENGTH  PIC X(80).
   . . .
PROCEDURE DIVISION.
   . . .
   PERFORM TEST AFTER
       VARYING LOCATION-TABLE-LENGTH FROM 1 BY 1
       UNTIL LOCATION-EOF
       OR LOCATION-TABLE-LENGTH = LOCATION-TABLE-MAX
   MOVE LOCATION-RECORD TO
       LOCATION-CODE (LOCATION-TABLE-LENGTH)
   READ LOCATION-FILE
       AT END SET LOCATION-EOF TO TRUE
   END-READ
END-PERFORM
```

9 SEARCH - Doing a Serial Search

The SEARCH statement can be used for performing a serial search beginning at the current index setting. This is a sequential search.

The SET statement is used for modifying the index setting.

The conditions in the WHEN phrase are evaluated in the order in which they appear:

- If none of the conditions is satisfied, the index is increased to correspond with the next table element, and the WHEN conditions are evaluated again.
- If one of the WHEN conditions is satisfied, the search ends.
 - The index remains pointing to the table element that satisfied the condition.
- If the entire table has been searched and no conditions have been met, the AT END imperative statement will be executed.
 - If an AT END has not been coded, control passes to the next statement in the program.

Only one level of a table can be referenced with each SEARCH statement. Nested SEARCH statements can be used for searching multiple levels of a table. Each nested SEARCH statement is delimited with END-SEARCH.

10 Searching a Table

COBOL provides two search techniques for tables:

serial	binary
--------	--------

Serial Search

SEARCH and indexing is used for performing serial searches. PERFORM with subscripting or indexing is used for variable-length tables.

For a serial search, the number of comparisons is of the order of n , the number of entries in the table.

Binary Search

SEARCH ALL and indexing is used for performing a binary search.

A binary search can be considerably more efficient than a serial search. For a binary search, the number of comparisons is to the order of the logarithm (base 2) of n .

A binary search requires that the table items already have been sorted. When the values in a table are not ordered, the only way to find an item is to search through the table sequentially, element by element.

With the COBOL language, the SEARCH verb is used to search a table sequentially.

```
SEARCH Table-Name [VARYING identifier]
    AT END
        Statements
    WHEN conditions
        Statements
END-SEARCH
```

Table-Name must identify a data-item in the table hierarchy with both OCCURS and INDEXED BY clauses. The index specified in the INDEXED BY clause of Table-Name is the controlling index of the SEARCH.

The SEARCH can only be used if the table to be searched has an index item associated with it. An index item is associated with a table by using the INDEXED BY phrase in the table declaration. The index item is known as the table index. The table index is the subscript which the SEARCH uses to access the table.

The SEARCH searches a table sequentially starting at the element pointed to by the table index.

The starting value of the table index is determined by the programmer. It will be necessary to ensure that when the SEARCH executes, the table index points to some element in the table. The table index cannot have a value of 0 or be greater than the size of the table.

The VARYING phrase is required when the data-item is to mirror the values of the table index. When the VARYING phrase is used, and the associated data-item is not the table index, then the data-item is varied along with the index.

The AT END phrase allows the programmer to specify an action to be taken if the searched for item is not found in the table. When the AT END is specified and the index is incremented beyond the highest legal occurrence for the table, then the statements following the AT END will be executed and the SEARCH will terminate.

The conditions attached to the SEARCH are evaluated in sequence and once one statement is true, the statements following the WHEN phrase are executed and the SEARCH ends.

11 SET Statement

This is the syntax of the SET statement.

```
SET {index|identifier} TO {index|identifier|integer}
SET index {UP|DOWN} BY {identifier|integer}
```

Example: Sample Program

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01  LETTERTABLE.
    02  LETTERVALUES.
        03  FILLER PIC X(13)
            VALUE 'ABCDEFGHIJKLM'.
        03  FILLER PIC X(13)
            VALUE 'NOPQRSTUVWXYZ'.
    02  FILLER REDEFINES LETTERVALUES.
        03  LETTER PIC X OCCURS 26 TIMES
            INDEXED BY LETTERIDX.

01  LETTERIN      PIC X.
01  LETTERPOS     PIC 99.
01  PRNPOS        PIC Z9.

PROCEDURE DIVISION.
BEGIN.
    MOVE 'D' TO LETTERIN

    SET LETTERIDX LETTERPOS TO 1

    SEARCH LETTER VARYING LETTERPOS

        AT END DISPLAY 'LETTER ` LETTERIN ` NOT FOUND!'

        WHEN LETTER(LETTERIDX) = LETTERIN
            MOVE LETTERPOS TO PRNPOS
            DISPLAY LETTERIN, ' IS IN POSITION ', PRNPOS

    END-SEARCH

    STOP RUN.
```

12 Binary Search - SEARCH ALL

When the SEARCH ALL is used for performing a binary search, it will not be necessary to set the index prior to starting the search. The index is always associated with the first index-name in the OCCURS clause. The index will vary during execution to maximize the search efficiency.

In order to use the SEARCH ALL statement for searching a table, the table must specify the ASCENDING or DESCENDING KEY phrases of the OCCURS clause, or both, and must already be ordered on the key or keys that are specified in the ASCENDING and DESCENDING KEY phrases.

In the WHEN phrase of the SEARCH ALL statement, any key that is named in the ASCENDING or DESCENDING KEY phrases for the table can be tested. However, all preceding keys must be tested. The test must be an equal-to condition and the WHEN phrase must specify either a key subscripted by the first index-name associated with the table or a condition-name that is associated with the key.

The WHEN condition can be a compound condition that is formed from simple conditions that use AND as the only logical connective. Each key and its object of comparison must be compatible according to the rules for comparison of data items.

13 SEARCH ALL Statement

The SEARCH ALL is used when a binary search is required; the SEARCH ALL will only work on an ordered table. The table must be ordered on some key field.

The key field may be the element itself or, where the element is a group item, a field within the element.

The key field must be identified by using the KEY IS phrase in the table declaration.

This is the syntax of the SEARCH ALL statement:

```
SEARCH ALL Tablename
      [AT END
        Statements]
      WHEN condition...
        Statements
END-SEARCH
```

The OCCURS clause of the table to be searched must have a KEY IS clause in addition to an INDEXED BY clause. The KEY IS phrase identifies the data-item upon which the table is ordered.

When the SEARCH ALL is used, it will not be necessary to set the table index to a starting value; the SEARCH ALL will control it automatically.

- If the table to be searched is only partially populated, the SEARCH ALL may not work correctly.
- If the table is ordered in ascending sequence, then to get the SEARCH ALL to function correctly the unpopulated elements must be filled with HIGH-VALUES.
- If the table is ordered in descending sequence, the unpopulated elements should be filled with LOW-VALUES.

14 Sample Program

DATA DIVISION.

WORKING-STORAGE SECTION.

```
01  LETTERTABLE.
    02  LETTERVALUES.
        03  FILLER PIC X(13)
            VALUE 'ABCDEFGHIJKLM'.
        03  FILLER PIC X(13)
            VALUE 'NOPQRSTUVWXYZ'.
    02  FILLER REDEFINES LETTERVALUES.
        03  LETTER PIC X OCCURS 26 TIMES
            ASCENDING KEY IS LETTER
            INDEXED BY LETTERIDX.
```

```
01  SEARCHLETTER PIC X.
01  LETTERPOS    PIC 99.
```

PROCEDURE DIVISION.

BEGIN.

MOVE 'N' TO ACCEPT SEARCHLETTER

SET LETTERIDX LETTERPOS TO 1

SEARCH ALL LETTER

AT END DISPLAY 'LETTER ` SEARCHLETTER ` WAS NOT FOUND!'

WHEN LETTER(LETTERIDX) = SEARCHLETTER

SET LETTERPOS TO LETTERIDX

DISPLAY SEARCHLETTER ` IS IN POSITION ` LETTERPOS

END-SEARCH

STOP RUN.