CSCI 465 **7. COBOL Tables**

COBOL tables are defined in the DATA DIVISION using the OCCURS n [TIMES] clause. If the table level is to be indexed, both the OCCURS n [TIMES] and the INDEXED BY clauses must be used together in the table's definition.

If using a subscript to reference a table entry, simply define it in WORKING-STORAGE as an integer large enough to hold the maximum number of entries. Always define a subscript as BINARY SYNC, COMP SYNC or COMP-4 SYNC. For example, if a table has an OCCURS 500, the subscript could be defined as:

```
01  SUBSCRIPTS.
    05  TBL-SUB          PIC S9(3)   BINARY SYNC.
```

or

```
01  TBL-SUB              PIC S9(3)   BINARY SYNC.
```

Unlike subscripts, the INDEXED BY clause simply gives a name to an index and the named index requires no further definition. In other words, and index will have **NO** PIC clause anywhere in the program.

If using an index, it cannot be manipulated by the standard arithmetic verbs (ADD, SUBTRACT,...) or the MOVE verb. Instead, the SET verb must be used:

FORMAT 1: SET var_name_1/index_1 TO var_name_2/index_2/integer

FORMAT 2: SET index_1 index_2 ... UP/DOWN BY var_name/integer

Like subscripts, index values can also be altered by PERFORM VARYING, SEARCH, or SEARCH ALL verbs.

Indexes are stored as binary values that represent the displacement from the beginning of the table to the specific entry.

Unlike third, fourth and fifth-generation languages and arrays, the valid range for both subscripts and indexes is 1 to n with n being the maximum number of entries. Of course, because indexes are stored as byte displacements, an index value of 1 corresponds to a byte displacement of 0.

A multi-dimensional table is created by nesting OCCURS clauses.

```
01  ANNUAL-SALES-DATA.
    05  YEARLY-SALES           OCCURS 5
                                INDEXED BY YR-NDX.
        10  MONTHLY-SALES      OCCURS 12
                                INDEXED BY M-NDX.
            15  SALE-AMOUNT    PIC  9(7)V99.
```

To access an entry in a multi-dimensional table, the order of the indexes will correspond to the nesting order of the OCCURS clauses.

SALE-AMOUNT(1, 4) references year 1 and month 4 (April).
SALE-AMOUNT(4, 1) references year 4 and month 1 (January).

To access an entry in a multi-dimensional table, every field needs as many indexes as there are `OCCURS` clauses that are a part of it or above it. In other words:

`SALE-AMOUNT(4)` would be invalid and will not compile.

Example of Multi-Dimensional Table Processing

The following is defined `WORKING-STORAGE`:

```
01  ANNUAL-SALES-DATA.
    05  YEARLY-SALES           OCCURS 5
                               INDEXED BY YR-NDX.
        10  MONTHLY-SALES      OCCURS 12
                               INDEXED BY M-NDX.
            15  SALE-AMOUNT    PIC  9(7)V99.
```

The following are statements in the `PROCEDURE-DIVISION`:

```
0000-MAIN.
    .
    .     other statements can precede the following
    .
    PERFORM VARYING YR-NDX FROM 1 BY 1 UNTIL YR-NDX > 5
      PERFORM VARYING M-NDX FROM 1 BY 1 UNTIL M-NDX > 12
        ADD SALE-AMOUNT(YR-NDX, M-NDX) TO TOTAL-AMOUNT
      END-PERFORM
    END-PERFORM.
    .
    .
    GOBACK.

0000-EXIT. EXIT.
```

or

```
0000-MAIN.
    .
    .     other statements can precede the following
    .
    PERFORM 1000-ADD VARYING YR-NDX FROM 1 BY 1 UNTIL YR-NDX > 5
      AFTER M-NDX FROM 1 BY 1 UNTIL M-NDX > 12.
    .
    .
    GOBACK.

0000-EXIT. EXIT.

1000-ADD.

    ADD SALE-AMOUNT(YR-NDX, M-NDX) TO TOTAL-AMOUNT.

1000-EXIT. EXIT.
```

Both of the above examples would accomplish the same thing.

COBOL Subscripts vs. Indexes

Differences Between Subscripts and Indexes:

Subscripts	***Indexes***
1. Needs to be declared with PIC clause.	1. Does NOT need to be declared with a PIC clause
2. Holds a binary value of 1 to n.	2. Holds a binary value representing a displacement in bytes from the beginning of the table.
3. Can be displayed using DISPLAY.	3. Cannot be displayed using DISPLAY.
4. Can be altered using the arithmetic verbs.	4. Can only be altered using the SET verb.
5. Can be altered using the MOVE verb.	5. Cannot be altered using the MOVE verb.
6. Can refer to any entry at any level of any tree.	6. Can only refer to an entry at a specific level of a specific tree.
7. Cannot be used with the SEARCH or SEARCH ALL verbs.	7. Required to be used with the SEARCH or SEARCH ALL verbs.

Similarities Between Subscripts and Indexes:

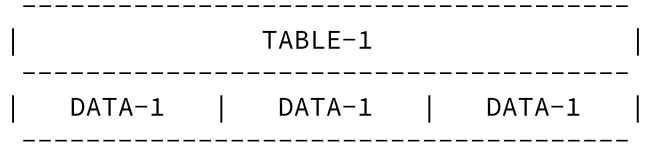
- Both reference a single entry at a single level of a COBOL table.
- Can be altered, or used, with the PERFORM VARYING with no special considerations.

Note: Subscripts should always be defined as BINARY SYNC or COMP SYNC and, for good form, a subscript name should end with the suffix -SUB and an index with the suffix -NDX.

Examples of COBOL Table Storage

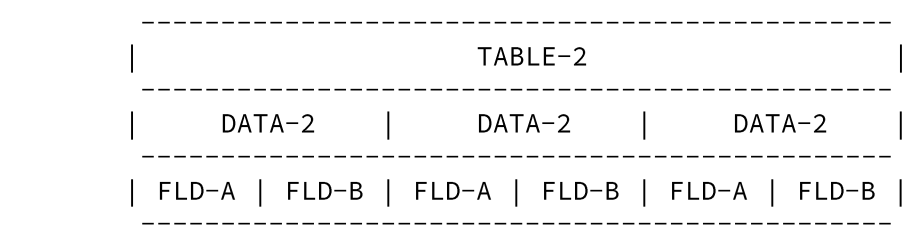
One- or Single-Dimensional Table:

```
01  TABLE-1.
    05  DATA-1      OCCURS 3    PIC X(6).
```



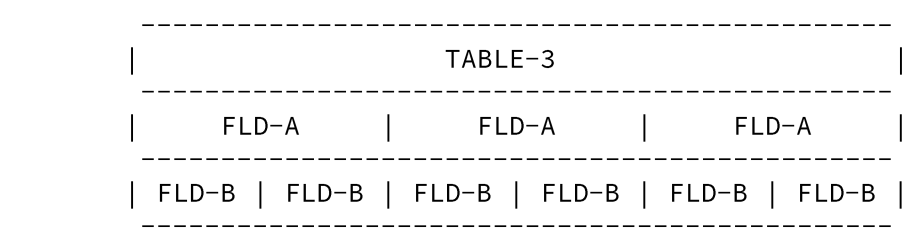
Another One- or Single-Dimensional Table:

```
01  TABLE-2.
    05  DATA-2        OCCURS 3.
        10  FLD-A     PIC X(4).
        10  FLD-B     PIC X(4).
```



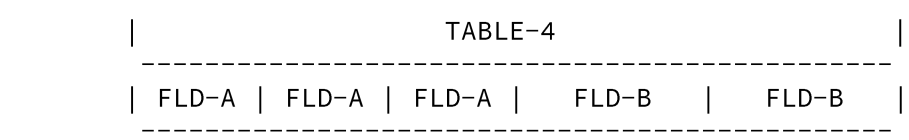
Two-Dimensional Table:

```
01  TABLE-3.
    05  FLD-A          OCCURS 3.
        10  FLD-B      OCCURS 2
                       PIC X(4).
```



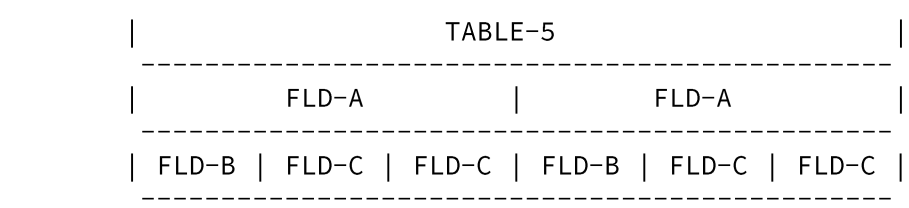
Two One- or Single-Dimensional Tables Under the Same 01-Level:

```
01  TABLE-4.
    05  FLD-A          OCCURS 3  PIC X(3).
    05  FLD-B          OCCURS 2  PIC X(4).
```



Two-Dimensional Table:

```
01  TABLE-5.
    05  FLD-A          OCCURS 2.
        10  FLD-B      PIC X(4).
        10  FLD-C      OCCURS 2
                       PIC X(4).
```



Note that any of the above-defined table examples can be indexed along with the `OCCURS` clause.

Also note that indexed table entries can be manipulated using a subscript as long as the subscript is large enough to hold the maximum number of entries for that table level. Indexes cannot be used to manipulate other levels of other tables.

COBOL SEARCH and SEARCH ALL Verbs

COBOL provides a means by which a table level can be searched for a matching value in a field without necessarily implementing a `PERFORM VARYING`. `SEARCH` is used to implement a sequential search through a table or a portion of a table and `SEARCH ALL` is used to implement a binary search.

Requirements of `SEARCH`

- Table levels on which a `SEARCH` will be implemented must be indexed.
- `SEARCH` can be implemented to look for a match in any field on the indexed level as long as the field is not itself part of a lower level table.
- The index value must be set before the `SEARCH` is implemented.
- The table level need not be in any sort order based on the field being searched.

Requirements of `SEARCH ALL`

- As with `SEARCH`, table levels on which a `SEARCH ALL` will be implemented must be indexed.
- The table level being searched with `SEARCH ALL` must be declared using the `ASCENDING KEY` or `DESCENDING KEY` field.
- For it to work, the table level must actually be in order based on the `ASCENDING` or `DESCENDING KEY` field.
- Unlike with `SEARCH`, the index value need not be set before the `SEARCH ALL` is implemented. In fact, if it is set, `SEARCH ALL` immediately alters in to begin the search in a binary fashion.

Examples

Given the following table definition:

```
01  FUND-TBL.
    05  FUND-TBL-ENTRY          OCCURS 300
                                ASCENDING KEY FUND-NBR
                                INDEXED BY FUND-NDX.
        10  FUND-NBR            PIC 9(2).
        10  FUND-SHR-PRC        PIC S9(3)V99 DECIMAL.
```

(to be continued...)