Objective:

The following section explains :

- Defining Internal Tables
- Processing Internal Tables
- Accessing Internal Tables
- Initializing Internal Tables

Internal tables are structured data types provided by ABAP/4.
Internal tables cannot be accessed outside the program environment.

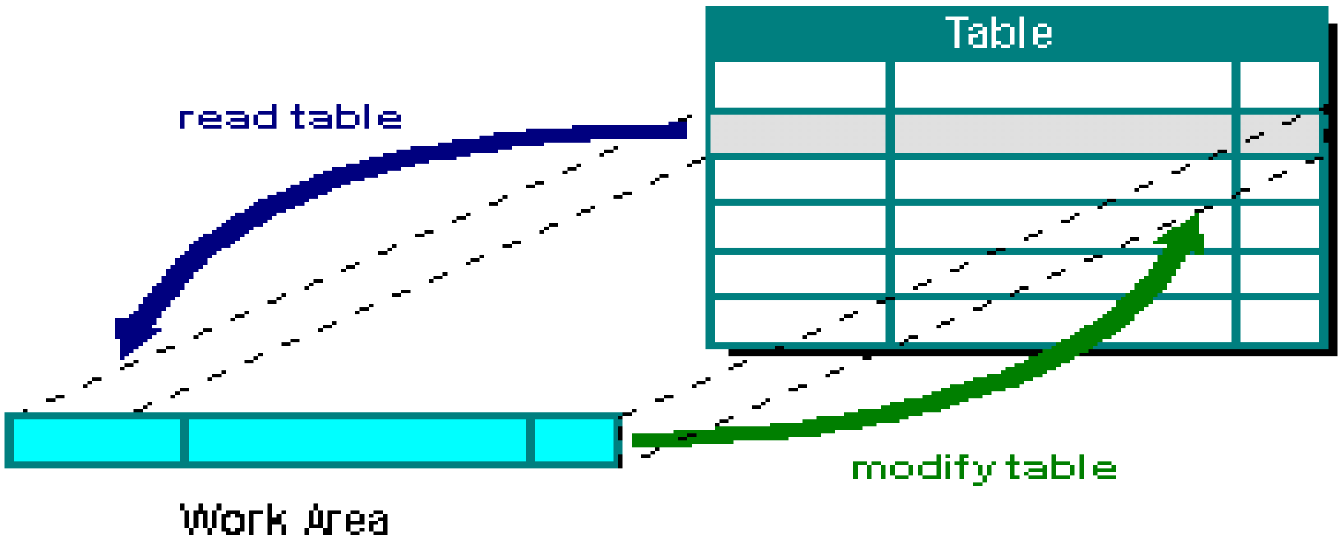
Purpose of internal tables

- Internal tables are used to reorganize the contents of database tables according to the needs of your program
- Internal tables are used to perform calculations on subsets of database tables.
- The number of lines in an internal table is not fixed.
- Internal tables exist only during the run time of a program.

•You access internal tables line by line. You must use a work area as an interface for transferring data to and from the table.

•When you read data from an internal table, the contents of the addressed table line overwrite the contents of the work area..

• When you write data to an internal table, you must first enter the data in the work area from which the system can transfer the data to the internal table.



There are two kinds of internal tables in ABAP/4:

- **Internal tables with header line**

  If you create an internal table with header line, the system automatically creates a work area with the same data type as the lines of the internal table.

Note : Work area has the same name as the internal table.

- **Internal tables without header line**

  Internal tables without a header line do not have a table work area which can be used implicitly you must specify a work area explicitly.

There are two ways by which you can create internal tables

First create an internal table data type using the 'TYPES' statement and then create a data object referring that data type.

Syntax : TYPES <t> <type> OCCURS <n>

Eg:

```
TYPES : BEGIN OF LINE,
          COLUMN1 TYPE I,
          COLUMN2 TYPE I,
          COLUMN3 TYPE I,
          END OF  LINE.

   TYPES ITAB TYPE LINE OCCURS 10.
```

Create an internal table data object by referring to a structure.

Syntax: DATA <f> <type> [WITH HEADER LINE]

Eg:

```
TYPES : BEGIN OF LINE,
          COLUMN1 TYPE I,
          COLUMN2 TYPE I,
          COLUMN3 TYPE I,
          END OF  LINE.
 TYPES ITAB TYPE LINE OCCURS 10.
  DATA ITAB1 TYPE ITAB.
  DATA ITAB2 LIKE ITAB1 WITH HEADER LINE.
```

Creating Internal Tables with header line

In this method an internal table is created with reference to existing structure (another internal table or Dictionary object).

Eg: DATA ITAB LIKE SFLIGHT OCCURS 10 WITH HEADER LINE.

Create an internal table data object directly with the 'DATA' statement.

Eg:
```
DATA : BEGIN OF ITAB OCCURS  0,
          COLUMN1 TYPE I,
          COLUMN2 TYPE I,
          COLUMN3 TYPE I,
          END OF  ITAB.
```

Note : In this method a header line with same name as the internal table is created automatically.

Appending Lines:

To append a line to an internal table, use the APPEND statement as follows:

Syntax

```
APPEND [<wa> TO]  <itab>.
```

Eg:
```
TYPES : BEGIN OF LINE,
          COL1 TYPE C,
          COL2 TYPE N,
          END OF  LINE.
DATA TAB1 LIKE LINE OCCURS 10.
LINE-COL1 = ‘A’. LINE-COL2 = ‘1’.
APPEND LINE TO TAB1.
```

Appending Lines depending on the Standard Key (COLLECT STATEMENT)

To fill an internal table with lines which have unique standard keys.

Syntax

```
COLLECT  [<wa> INTO] <itab>
```

If an entry with the same key already exists(all non-numeric fields) the collect statement does not append a new line but adds the contents of the numeric fields in the work area to the contents of the numeric fields in the existing entry

```
Eg: TYPES : BEGIN OF ITAB1 OCCURS 0,
            COL1 TYPE C,
            COL2 TYPE I,
            END OF  ITAB1.

ITAB1-COL1 = ‘A’. ITAB1-COL2 = ‘1’.
COLLECT TAB1.
ITAB1-COL1 = ‘B’. ITAB1-COL2 = ‘2’.
COLLECT TAB1.
ITAB1-COL1 = ‘A’. ITAB1-COL2 = ‘1’.
COLLECT TAB1.
```

This Produces the output as follows:

```
A           2
B           2
```

Inserting lines :

To insert a new line before a line in an internal table, you use the INSERT statement as follows:

Syntax

```
INSERT [<wa> INTO] <itab> [INDEX <idx>].
```

```
Eg: TYPES : BEGIN OF LINE,
            COL1 TYPE C,
            COL2 TYPE N,
            END OF  LINE.
DATA TAB1 LIKE LINE OCCURS 10.
LINE-COL1 = ‘A’. LINE-COL2 = ‘1’.
INSERT LINE INTO TAB1 INDEX 2.
```

COPYING INTERNAL TABLES :

To copy the entire contents of one internal table into another, use the MOVE statement.

Syntax:

```
 MOVE  <itab1>  to  <itab2>
```

Eg:

```
TYPES : BEGIN OF LINE,
            COL1 TYPE C,
            COL2 TYPE N,
            END OF  LINE.

DATA TAB1 LIKE LINE OCCURS 10.
DATA TAB2 LIKE LINE OCCURS 10.
LINE-COL1 = ‘A’. LINE-COL2 = ‘1’.
APPEND LINE TO TAB1.
MOVE TAB1[ ]  TO TAB2.
```

Filling Internal Table from Database Table:

Example

```
TABLES SPFLI.

DATA ITAB LIKE SPFLI OCCURS 10 WITH HEADER   LINE.

SELECT * FROM SPFLI INTO TABLE ITAB
             WHERE CARRID = 'LH'.

LOOP AT ITAB.
  WRITE: / ITAB-CONNID, ITAB-CARRID.
ENDLOOP.
```

In this example, all lines from the database table SPFLI in which CARRID field contains "LH" are read into the internal table ITAB, where they can be processed further.

To read data component by component into the internal table.

```
TABLES SPFLI.
DATA: BEGIN OF ITAB OCCURS 0,
        NUMBER TYPE I VALUE 1,
         CITYFROM LIKE SPFLI-CITYFROM,
         CITYTO   LIKE SPFLI-CITYTO,
      END OF ITAB.

SELECT * FROM SPFLI  WHERE CARRID = 'LH'.
   MOVE-CORRESPONDING SPFLI TO ITAB.
   APPEND ITAB.
ENDSELECT.
```

In this example, all lines from the database table SPFLI in which CARRID field contains "LH" are read into the internal table ITAB *one by one*, where they can be processed further.

To read the contents of internal tables for further processing, you can use either the LOOP or the READ statement.

Reading Internal Tables Line by Line

You use the the LOOP statement to read internal tables line by line.

```
LOOP AT <itab> [INTO <wa>]  [WHERE <condition>].
   .....
ENDLOOP.
```

Eg:

```
DO 3 TIMES.
LINE-COL1 = SY-INDEX.  LINE-COL2 = SY-INDEX * SY-INDEX.
APPEND LINE TO TAB1.
ENDDO.
LOOP AT TAB1 WHERE COL1 > 2.
WRITE : / TAB1-COL1.
ENDLOOP.
```

You can select a single line by the READ statement:

Syntax:

READ TABLE <itab> [INTO <wa>] WITH KEY<key> [BINARY SEARCH].

Eg:

```
 TYPES : BEGIN OF LINE,
      COL1 TYPE C,
      COL2 TYPE N,
      END OF  LINE.
DATA TAB1 LIKE LINE OCCURS 10.
DO 3 TIMES.
LINE-COL1 = SY-INDEX.  LINE-COL2 = SY-INDEX ** 2.
APPEND LINE TO TAB1.
ENDDO.
READ TABLE TAB1 INTO LINE WITH KEY COL2 =  4.
```

The COMPARING addition, the specified table fields <f $_i$ > of the structured line type are compared with the corresponding fields of the work area before being transported. If the contents of the compared fields are the same, SY-SUBRC is set to 0. If the contents of the compared fields are not the same, it returns the value 2.

Syntax:

READ TABLE <itab> [INTO <wa>] INDEX <idx> COMPARING <fields>.

Eg:

```
      TYPES : BEGIN OF LINE,
          COL1 TYPE C,
          COL2 TYPE N,
          END OF  LINE.
     DATA TAB1 LIKE LINE OCCURS 10.
     DO 3 TIMES.
     LINE-COL1 = SY-INDEX.  LINE-COL2 = SY-INDEX ** 2.
     APPEND LINE TO TAB1.
     ENDDO.
     READ TABLE TAB1 INTO LINE INDEX 2 COMPARING COL1 COL2.
```

You can modify single line using MODIFY statement:

Syntax :

MODIFY itab [FROM wa] [INDEX idx].

Eg:

```
 TYPES : BEGIN OF LINE,
       COL1 TYPE C,
       COL2 TYPE N,
       END OF  LINE.
DATA TAB1 LIKE LINE OCCURS 10.
DO 3 TIMES.
LINE-COL1 = SY-INDEX.  LINE-COL2 = SY-INDEX ** 2.
APPEND LINE TO TAB1.
ENDDO.
LINE-COL1 = ‘A’.
MODIFY TAB1 FROM LINE INDEX 2.
```

Syntax :

MODIFY itab [FROM wa] [TRANSPORTING f1 ... fn [WHERE cond]].

Eg:

```
 TYPES : BEGIN OF LINE,
      COL1 TYPE C,
      COL2 TYPE N,
      END OF  LINE.
DATA TAB1 LIKE LINE OCCURS 10.
DO 3 TIMES.
LINE-COL1 = SY-INDEX.  LINE-COL2 = SY-INDEX ** 2.
APPEND LINE TO TAB1.
ENDDO.
LINE-COL1 = ‘A’.
MODIFY TAB1 FROM LINE TRANSPORTING COL1 WHERE COL2 = 4.
```

To delete lines from an internal table in a loop:

Syntax:

```
  DELETE <itab>.
```

Note: The System can process this statement only within an LOOP..ENDLOOP block.

Eg:

```
DO 3 TIMES.
LINE-COL1 = SY-INDEX.  LINE-COL2 = SY-INDEX * SY-INDEX.
APPEND LINE TO TAB1.
ENDDO.
LOOP AT TAB1.
  IF TAB1-COL1 > 2.
   DELETE TAB1.
  ENDIF.
ENDLOOP.
```

TO delete the lines using Index.

Syntax:

```
DELETE <itab> INDEX <idx>.
```

Eg:

```
 TYPES : BEGIN OF LINE,
         COL1 TYPE C,
         COL2 TYPE N,
         END OF  LINE.
 DATA TAB1 LIKE LINE OCCURS 10.
DO 3 TIMES.
LINE-COL1 = SY-INDEX.  LINE-COL2 = SY-INDEX * SY-INDEX.
APPEND LINE TO TAB1.
ENDDO.
DELETE TAB1 INDEX 2.
```

TO delete the Adjacent Duplicates from the Internal Table.

Syntax:

```
  DELETE ADJACENT DUPLICATE ENTRIES FROM <itab> [ COMPARING
                                                          <comp>]
```

Eg:

```
     TYPES : BEGIN OF LINE,
          COL1 TYPE C,
          COL2 TYPE N,
          END OF  LINE.
     DATA TAB1 LIKE LINE OCCURS 10.
     LINE-COL1 = 'A'. LINE-COL2 = '1'.
     APPEND LINE TO TAB1.
     LINE-COL1 = 'A'. LINE-COL2 = '2'.
     APPEND LINE TO TAB1.
     DELETE ADJACENT DUPLICATE ENTRIES FROM TAB1
                                          COMPARING   COL1.
```

TO delete the Adjacent Duplicates from the Internal Table.

Syntax:

```
DELETE ADJACENT DUPLICATE ENTRIES FROM <itab> [ COMPARING
                                                 <comp>]
```

Eg:

```
   TYPES : BEGIN OF LINE,
          COL1 TYPE C,
          COL2 TYPE N,
          END OF  LINE.
   DATA TAB1 LIKE LINE OCCURS 10.
  LINE-COL1 = ‘A’. LINE-COL2 = ‘2’.
  APPEND LINE TO TAB1.
  LINE-COL1 = ‘A’. LINE-COL2 = ‘2’.
  APPEND LINE TO TAB1.
  DELETE ADJACENT DUPLICATE ENTRIES FROM TAB1
                              COMPARING   ALL FIELDS.
```

TO delete a set of selected lines from the Internal Table.

Syntax:

```
  DELETE  <itab> [FROM <n1>] [TO <n2>] [WHERE <condition>].
```

Eg:

```
     TYPES : BEGIN OF LINE,
          COL1 TYPE C,
          COL2 TYPE N,
          END OF  LINE.
     DATA TAB1 LIKE LINE OCCURS 10.
     LINE-COL1 = ‘A’. LINE-COL2 = ‘1’.
     APPEND LINE TO TAB1.
     LINE-COL1 = ‘A’. LINE-COL2 = ‘2’.
     APPEND LINE TO TAB1.
     DELETE TAB1 WHERE COL2 = 2.
```

TO Sort an Internal Table .

Syntax:

```
SORT  <itab> [<order>] [AS TEXT]  [BY <f1> [<order>] [AS TEXT] . . <fn>
                                                    [<order>] [AS TEXT] ]  .
```

Eg:

```
TYPES : BEGIN OF LINE,
          COL1 TYPE C,
          COL2 TYPE N,
          END OF  LINE.
    DATA TAB1 LIKE LINE OCCURS 10.
   LINE-COL1 = ‘A’. LINE-COL2 = ‘2’.
   APPEND LINE TO TAB1.
   LINE-COL1 = ‘A’. LINE-COL2 = ‘1’.
   APPEND LINE TO TAB1.
   SORT TAB1 BY COL2 ASCENDING .
```

Calculating the totals within loop…endloop.

Syntax: SUM

Eg.

```
TYPES : BEGIN OF LINE,
    COL1 TYPE C,
    COL2 TYPE I,
    END OF  LINE.
DATA TAB1 LIKE LINE OCCURS 10.
DO 3 TIMES.
LINE-COL1 = SY-INDEX.  LINE-COL2 = SY-INDEX ** 2.
APPEND LINE TO TAB1.
ENDDO.
LOOP AT TAB1.
SUM.
ENDLOOP.
```

This topic describes how to use control level statement blocks which process only specific values within loop…endloop.

Syntax:

```
AT <line>
  <statement block>
ENDAT.
```

The line condition <line>, at which the statement block within AT-ENDAT.

<Line>	Meaning
FIRST	First line of the internal table
LAST	Last line of the internal table
NEW <f>	Beginning of a group of line with same contents in the fields <f> & in the fields of <f>.
END OF <f>	End of a group of line with same contents in the fields <f> & in the fields of <f>.

Note: Before working with control breaks, You should sort the internal table in the same order as its columns are defined.

Hierarchy of AT-ENDAT statement.

If the internal table has the columns <col1>,<col2>,… and if it is sorted by the columns as they are defined, the loop is to be programmed as follows:

```
LOOP AT <itab>.
        AT FIRST. …..ENDAT.
                AT NEW <col1>….ENDAT.
                        AT NEW <col2>……ENDAT.
                        …….
                        <single line processing>
                        ……...
                        AT END OF <col2>…..ENDAT.
                AT END OF <col1>……ENDAT.
        AT  LAST……ENDAT.
ENDLOOP.
```

Example:

```
TYPES : BEGIN OF LINE,
         COL1 TYPE C,
         COL2 TYPE I,
         END OF  LINE.
        DATA TAB1 LIKE LINE OCCURS 10.

        LINE-COL1 = ‘A’. LINE-COL2 = ‘2’.
        APPEND LINE TO TAB1.
        LINE-COL1 = ‘B’. LINE-COL2 = ‘1’.
        APPEND LINE TO TAB1.
        LINE-COL1 = ‘A’. LINE-COL2 = ‘3’.
        APPEND LINE TO TAB1.
        LINE-COL1 = ‘C’. LINE-COL2 = ‘4’.
        APPEND LINE TO TAB1.
        SORT TAB1 BY COL1.
```

Cont..

```
LOOP AT TAB1.
          AT FIRST.
                    WRITE:/ ‘HEADING’.
          ENDAT.
          AT NEW COL1.
                    WRITE:/ TAB1-COL1.
          ENDAT.
          AT END OF COL1.
                    SUM.
                     WRITE:/ TAB1-COL1, TAB1-COL2.
          ENDAT.
          AT LAST.
                    SUM.
                     WRITE:/ TAB1-COL1, TAB1-COL2.
          ENDAT.
          ENDLOOP.
```

To initialize an internal table with or without header line.

Syntax :

```
        REFRESH <itab>.
```

This statement resets an internal table.

```
        CLEAR <itab>.
```

If you are working with an internal table with a header line , the clear statement clears only the table work area resetting to initial values.

```
        CLEAR <itab>[ ].
```

The square bracket after the name of the internal table refer to the body of the internal table.This statement also resets an internal table.

```
         FREE <itab>.
```

You can release the memory with the FREE statement once initialized.

Example:

```
TYPES : BEGIN OF LINE,
         COL1 TYPE C,
         COL2 TYPE I,
         END OF  LINE.
        DATA TAB1 LIKE LINE OCCURS 10.

        LINE-COL1 = ‘A’. LINE-COL2 = ‘2’.
        APPEND LINE TO TAB1.
        LINE-COL1 = ‘B’. LINE-COL2 = ‘1’.
       APPEND LINE TO TAB1.
        CLEAR TAB1.
        REFRESH TAB1.
        IF TAB1 IS INITIAL.
        WRITE:/ ‘TAB1 IS EMPTY’.
        FREE TAB1.
        ENDIF.
```