

ABAP Part I

Lesson 07: Internal Tables

Lesson Objectives

- After completing this lesson, participants will be able to know:
 - To Define an Internal Table and understand its attributes
 - Types of Internal Tables
 - To Add, Read, Update and Delete Data from an internal Table
 - To Sort the Contents of an Internal Table
 - Control break statements on Internal Table



Internal Tables

- Provides a means of taking data from a fixed structure and storing it in working memory in ABAP
- Internal Tables fulfill the function of Arrays
- A very important use of internal tables is for storing and formatting data from a database table within a program.
- The data type an internal table is fully specified by its
 - line type
 - key
 - table type

Internal Tables

Line Type

- Can be any data type
- The data type of an internal table is usually a Structure
- Each component of a structure is a column in internal table

Key

- Identifies Table rows
- Two kinds of keys
 - Standard Key
 - User-Defined Key

1 Line type			
	CARRID	CONNID	DISTANCE
index			
1	AA	0017	2.572
2	LH	0400	6.162
3	LH	0402	7.273
4	QF	0005	10.000
5	SQ	0866	1.625
6	UA	0007	2.572

2 Key

- Components
- Uniqueness
- Sequence

3 Table type

- Standard
- Sorted
- Hashed

Internal Tables - Key

- Standard Key
 - Tables with Structured and non -structured row type : all Character Type columns
 - Tables with Elementary line-type : entire line is the default key
 - If line type is an internal table : no default key
 - Tables with non-structured row type :
- User-Defined Key
 - Any column of an internal table which is not an internal table by itself
- Key can be UNIQUE or NON-UNIQUE

Internal Tables – Table Type

- Determines how ABAP accesses the individual entries
- Three Types
 - Standard Tables
 - Sorted Tables
 - Hashed Tables

Operations on Individual Lines

■ Key and Index Access

	Standard Table	Sorted Table	Hashed Table
Index Access	✓	✓	X
Key Access	✓	✓	✓
Key Values	Not Unique	Unique/Not Unique	Unique
Preferred Access	Mainly Index	Mainly Key	Key Only

Internal Tables – Standard Table

- Have an internal linear index
- Records are accessed by index or keys
- The response time for key access is proportional to number of entries
- The key is always non-unique

Internal Tables – Sorted Table

- Always saved sorted by the key
- Have an internal index
- Records can be accessed by table index or key
- Uses Binary Search for access
- Key can be Unique or non-Unique

Internal Tables – Hashed Table

- Has no linear index
- Only accessed using its key
- Key must be Unique
- Uses Hash Algorithm for accessing records
- Response time is constant

Creating Internal Tables

- Syntax:

TYPES type TYPE|LIKE tabkind OF linetype [WITH key] [INITIAL SIZE n].
DATA itab TYPE type|LIKE obj.

- Example:

```
DATA : BEGIN OF str_mat,  
        matno(18),  
        matname(30),  
        mattype(4), " Material Type  
        uom(10), "Unit of Measure  
      END OF str_mat.  
DATA IT_MAT like STANDARD TABLE OF str_mat."Internal Table  
with line type str_mat
```

Demo

- Program on Internal table without header line



Internal Tables Objects

- Internal tables are dynamic variable data objects.
- Like all variables, declare them using the DATA statement.

Reference to Declared Internal Table Types

- Declare internal table objects using the LIKE or TYPE.

DATA <itab> TYPE <type>|LIKE <obj> [WITH HEADER LINE].

- Here, the LIKE addition refers to an existing table object in the same program.
- The TYPE addition can refer to an internal type in the program declared using the TYPES statement, or a table type in the ABAP Dictionary.
- The optional addition WITH HEADER LINE declares an extra data object with the same name and line type as the internal table.

Reference to Declared Internal Table Types

- Use it as a work area when working with the internal table.
- When using internal tables with header lines, the header line and the body of the table have the same name.
- If there is an internal table with header line, to address the body of the table, place brackets after the table name (`<itab>[]`).
- If not, ABAP interprets the name as the name of the header line and not of the body of the table.
- It is possible to avoid this potential confusion by using internal tables without header lines.

Defining Internal Tables with Global Types

Table Type

BC400_T_FLIGHTS

Line Type

BC400_S_FLIGHT

Access

Standard Table

Key

Key Type

Non-unique

Key Components

CARRID, CONNID, FLDATE

DATA

gt_flights TYPE ho400_t_flights.

gt_flights

carrid	connid	fldate	seatsmax	seatsocc	percentage

Defining Internal Tables with Local Types

```
TYPES gty_t_flights
      TYPE STANDARD TABLE OF bo400_s_flight
      WITH NON-UNIQUE KEY carrid connid fldate.

DATA gt_flights TYPE gty_t_flights.
```

Local Table Type

Internal Table

gt_flights	carrid	connid	fldate	seatsmax	seatsocc	percentage

Independent Definition of Internal Tables

```
TYPES: BEGIN OF gty_s_type,
        carrid TYPE s_carr_id,
        connid TYPE s_conn_id,
        ... ,
        END OF gty_s_type.
```

Local Structure Type

```
DATA gt_itab TYPE STANDARD TABLE OF gty_s_type
              HASHED
              WITH ... KEY ...
```

Internal Table

Possible Definition of Internal Tables

- 1 `DATA gt_itab TYPE <Table Type> .`
- 2 `DATA gt_itab TYPE STANDARD
SORTED TABLE OF <Structure Type>
HASHED
WITH ... KEY ...`
- 3 `DATA gt_itab TYPE TABLE OF <Structure Type> .`
(Short form for definition of a standard table with
non-unique default key)

Operations on Entire Internal Table

- The entire body of the table is addressed as a single Data Object
- The following operations are done on the entire Internal Table
 - Assigning Internal Tables
 - Initializing Internal Tables
 - Compare Internal Tables
 - Sort Internal Tables
 - Internal Tables as Interface Parameters
 - Determining the Attributes of Internal Tables

Assigning Internal Tables

- Syntax:

- MOVE itab1 TO itab2. or
- itab2 = itab1.

- Example:

```
DATA : it_mat LIKE TABLE OF str_mat,  
      it_mat1 LIKE TABLE OF str_mat.
```

```
str_mat-matno = 'm01'.  
str_mat-matname = 'Notepads'.  
str_mat-matttype = 'Stationery'.  
str_mat-uom = 'pcs'.
```

```
APPEND str_mat TO it_mat .  
MOVE it_mat to it_mat1. "or it_mat1 = it_mat.
```

Initializing Internal Tables

- **CLEAR itab.**
 - Releases the memory space of the internal table, but for the initial memory requirement.
- **REFRESH itab**
 - Applies to the body of the internal table.
 - The initial memory requirement for the table remains reserved.
- **FREE itab**
 - The internal table is initialized and the entire memory space is released.

Demo

- Program on Free, Refresh and Clear



Internal Tables - Processing

- Sorting Internal Tables

- Syntax

`SORT itab [ASCENDING|DESCENDING] [AS text] [STABLE].`

- To get the Attributes of Internal Tables

- Syntax

`DESCRIBE TABLE itab [LINES lin] [OCCURS n] [KIND kind].`

Sort Internal Table

```
SORT gt_flightinfo.  
  
SORT gt_flightinfo BY carrid.  
  
SORT gt_flightinfo BY percentage DESCENDING  
                        carrid ASCENDING .
```

Access Methods to Individual Table Entries

- 2 ways to access the individual rows of an Internal Table

- Accessing the Internal Table Rows Using a Work Area

- The data in the table is not directly accessed but through another Data Object referred as a Work Area
 - Work Area must be compatible with the line type of internal table or must be convertible into line type of the Internal Table
 - When a data is read from the table, the data overwrites the current contents of the Work Area
 - When data is written to the Internal Table, it must be placed in the Work Area and then transferred to the Internal Table
 - If the internal table has a Header Line, the Header Line can act as a Work Area.

Internal Tables with and Without Header Lines

```
DATA gt_itab
  TYPE STANDARD TABLE OF gty
  WITH NON-UNIQUE KEY carrid.
DATA gs LIKE LINE OF gt_itab .
```

Work Area **gs**

Internal Table **gt_itab**

```
INSERT gs INTO gt_itab INDEX n.
READ TABLE gt_itab
  INTO gs INDEX n.
WRITE gs-carrid.
```

Explicit Syntax

```
DATA itab
  TYPE STANDARD TABLE OF gty
  WITH NON-UNIQUE KEY carrid
  WITH HEADER LINE .
```

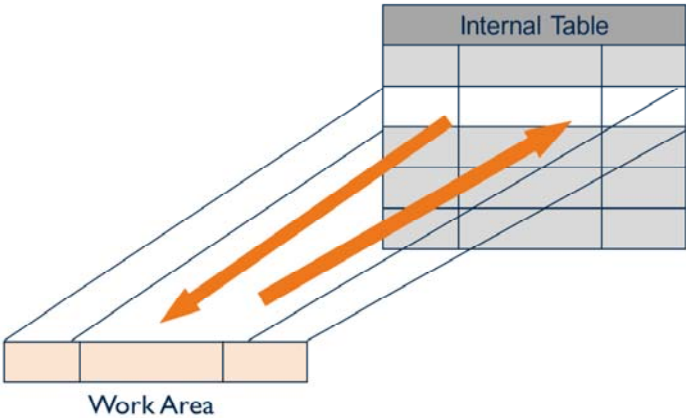
Header **itab**

Internal Table **itab**

```
INSERT itab INDEX n.
READ TABLE itab INDEX n.
WRITE itab-carrid.
```

Implicit Syntax



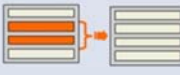
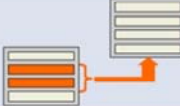
Access Using a Work Area



Accessing Single Records (Overview)



Processing Sets of Records (Overview)

Processing record by record over the entire (or a part) of the internal table		<pre>LOOP AT gt_it INTO gs <condition>. ... ENDLOOP</pre>
Deleting several records		<pre>DELETE gt_it <condition>.</pre>
Inserting several rows from another internal table		<pre>INSERT LINES OF gt_it1 <condition1> INTO gt_it2 <condition2>.</pre>
Appending several rows from another internal table		<pre>APPEND LINES OF gt_it1 <condition> TO gt_it2.</pre>

Filling Internal Tables Line By Line

- To Insert Individual Row at a Specific Position

INSERT wa INTO TABLE itab.

- To Append Rows to the Internal Table

APPEND wa TO itab.

- To Create Unique or Aggregate Internal Tables

COLLECT wa INTO itab.

Filling Internal Tables

- Standard Tables
 - Lines are always Appended to the end of the Internal Table.
- Sorted Tables
 - The lines are inserted into the table according to the table key
 - If the key is non-Unique, the duplicate entry is inserted above the existing entry with the same key.
- Hashed Tables
 - The lines are inserted according to the table key.

Inserting Lines

- Inserting Several Lines

- To insert several lines into the internal table use

```
INSERT LINES OF itab1 [FROM n1] [TO n2] INTO TABLE itab2.
```

- Inserting Lines Using The Index

- Inserting a Single Line

```
INSERT WA INTO itab [INDEX idx].
```

- Inserting Several Lines

```
INSERT LINES OF itab1 [FROM n1] [TO n2] INTO itab2 [INDEX idx].
```

Inserting a Row

```
* define internal table and workarea
```

DATA: gt_flightinfo TYPE bc400_t_flights,

gs_flightinfo LIKE LINE OF gt_flightinfo.

gt_flightinfo

gs_flightinfo

--	--	--	--	--

```
* fill structure with values
```

gs_flightinfo-carriid =
gs_flightinfo-connid =
gs_flightinfo-fldate =
gs_flightinfo-seatsmax =
gs_flightinfo-seatsocc =
gs_flightinfo-percentage =

```
* insert structure into internal table  
INSERT gs_flightinfo INTO TABLE gt_flightinfo.
```

Demo

- Program on Insert a record and Insert Multiple records



Appending Lines

- Appending a Single Line
 - APPEND line TO itab.
- Appending Multiple Lines
 - APPEND LINES of itab1 TO itab2.

Reading Lines of Table

- To Read a Single Line

READ TABLE itab key result.

- The system field SY-SUBRC is set to 0 if there is a matching entry in the internal table and if not, it sets the value to 4.

- To Read a Single line based on the Condition

READ TABLE itab FROM wa result.

READ TABLE itab WITH TABLE KEY k1 = f1 ... kn = fn result.

- To Read a Single Line Using Index (Specifying the row number)

READ TABLE itab INDEX idx result.

Demo

- Program on reading from Internal Tables



Processing Table Entries in Loops

- Syntax:

LOOP AT itab result condition.

Statements...

ENDLOOP.

- Depending on the table types the lines are processed

- Standard and Sorted Table

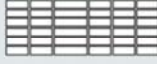

- Processed based on Index
- The system field SY-TABIX stores the index of the current line

- Hashed Table

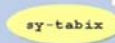
- If the table is not sorted, the lines are accessed in the inserted order
- SY-TABIX = 0 , within the processing block.

Outputting an Internal Table Row-by-Row

```
* define internal table and workarea
DATA: gt_flightinfo TYPE bo400_t_flights,
      gs_flightinfo LIKE LINE OF gt_flightinfo.

gt_flightinfo = 
gs_flightinfo = 

LOOP AT gt_flightinfo INTO gs_flightinfo.
  WRITE: / gs_flightinfo-carrid,
           gs_flightinfo-connid,
           gs_flightinfo-fldate,
           gs_flightinfo-seatsmax,
           gs_flightinfo-seatsocc,
           gs_flightinfo-percentage,
           '%'.
ENDLOOP.
```



Reading Internal table

```
LOOP AT gt_flightinfo INTO gs_flightinfo
    FROM 1 TO 5 .
WRITE: / gs_flightinfo-carrid,
        gs_flightinfo-connid,
        gs_flightinfo-fldate,
        gs_flightinfo-seatsmax,
        gs_flightinfo-seatsocc,
        gs_flightinfo-percentage,
        '%'.
ENDLOOP.
```

sy-tabix

```
READ TABLE gt_flightinfo INTO gs_flightinfo
    INDEX 3

WRITE: / gs_flightinfo-carrid,
        gs_flightinfo-connid,
        gs_flightinfo-fldate,
        gs_flightinfo-seatsmax,
        gs_flightinfo-seatsocc,
        gs_flightinfo-percentage,
        '%'.

```

Reading Internal table

```
LOOP AT gt_flightinfo INTO gs_flightinfo
    WHERE carrid = 'LH' .
    WRITE: / gs_flightinfo-carrid,
             gs_flightinfo-connid,
             gs_flightinfo-fldate,
             gs_flightinfo-seatsmax,
             gs_flightinfo-seatsocc,
             gs_flightinfo-percentage,
             '%'.
ENDLOOP.

READ TABLE gt_flightinfo INTO gs_flightinfo
    WITH TABLE KEY carrid = 'LH'
                   connid = '0400'
                   fldate = sy-datum .
    sy-tabix

IF sy-subrc = 0.
    WRITE: / gs_flightinfo-seatsmax,
             gs_flightinfo-seatsocc,
             gs_flightinfo-percentage,
             '%'.
ENDIF.
```

Changing Lines

- Changing Lines of an Internal Table

MODIFY itab FROM wa [TRANSPORTING f1 f2 ...].

- Changing Internal Table Contents based on Condition

MODIFY itab FROM wa TRANSPORTING f1 f2 ... WHERE cond.

- Changing the contents of an Internal Table Using an Index

MODIFY itab FROM wa [INDEX idx] [TRANSPORTING f1 f2...].

Deleting Lines

DELETE TABLE itab FROM wa.

- Deletion Using Table Key

DELETE TABLE itab WITH TABLE KEY k1 = f1 ... kn = fn.

- Delete the Internal Table Contents Based on a Condition

DELETE itab WHERE cond.

- Deleting Duplicate Records

DELETE ADJACENT DUPLICATE ENTRIES FROM itab [COMPARING f1 f2 ...
|ALL FIELDS].

Deleting Lines – Contd..

- Deleting a line specifying the Index

```
DELETE itab [INDEX idx].
```

- Deleting Several Lines

```
DELETE [FROM n1] [TO n2] [WHERE condition].
```

Demo

- Program on deleting from Internal tables



Determining the Number of Rows in an Internal Table

- To determine the number of rows in an internal table, use `sy-tfill` variable.
- It is set by the `describe table` statement.
- Syntax

The following is the syntax for the *describe table* statement.

`describe table it [lines i] [occurs j].`

- where:
 - `it` is the name of an internal table
 - `i` and `j` are numeric variables

Determining the Number of Rows in an Internal Table

- The describe statement fills the three system variables shown in table below

Variable	Value
<i>Sy-tfill</i>	Number of rows
<i>Sy-rleng</i>	Length of a row in bytes
<i>Sy-toccu</i>	Current value of the occurs clause

- The following points apply:
 - If the lines i addition is specified, the number of rows is placed in both sy-tfill and i
 - If the occurs j addition is specified, the size of the occurs clause is placed in both sy-toccu and j

There is only one instance where sy-toccu will differ from the occurs clause on the table definition.

When $\text{sy-rleng} * \text{sy-toccu} > 8192$, and after one row has been added to the internal table, sy-toccu will be zero.

This indicates that memory is being allocated in 8K chunks for this internal table.

Creating Top 10 Lists Using the append sorted by

- Imagine that you are asked to create a report of the top 10 sales representatives in your company.
- Assuming you have a way of obtaining the total sales for each rep, you could do one of the following:
 - Append all reps and their total sales into an internal table
 - Sort them descending by sales
 - Write out the first 10
- This seems like a logical way to proceed.
- However, using the append sorted by statement often can produce the same result and be twice as efficient

Creating Top 10 Lists Using the append sorted by

- Syntax for the append sorted by Statement:
 - **append** **[wa to] it** **sorted by c.**
- where:
 - it is the name of an internal table
 - wa is a work area having the same structure as row of the internal table
 - c is a component of it

The following points apply:

If wa to is not specified, the row to be appended is taken from the header line

If wa to is specified, the row to be appended is taken from the work area wa

Only one component c can be specified

Creating Top 10 Lists Using the append sorted by

- The append sorted by statement takes a row from the work area and inserts it into the internal table at the point where it belongs in the sort order.
- It has two unusual properties:
 - The number of rows that can be appended is limited by the value on the occurs clause.
 - For example, if the occurs is 10, a maximum of 10 rows can be appended to the internal table.
 - This is the only situation where occurs limits the number of rows that can be added to an internal table
 - It only sorts in descending order
- The net effect is a “top n list,” where n is the number on the occurs clause.

Creating Top 10 Lists Using the append sorted by

- With each append, the system searches the existing table contents to determine where the new record fits.
- The sort order is by c descending.
- If there are fewer rows in the internal table than specified by n on the occurs clause, the row is as per the sort order.
- If there are n rows in the internal table, the row is inserted as per the sort order and the last row is discarded.
- If the value in c already exists in the internal table, the new row is always appended after existing rows that have the same value.
- Therefore, if occurs is 3 and row 3 contains 'X' in c, a new row having 'X' in c will not be appended

Demo

- Program Append Sorted By



Filling an Internal Table Using collect

- Using the collect statement, you can create totals with an internal table as you fill it.
- Syntax for the collect Statement
The following is the syntax for the *collect* statement.
collect [wa into] it.
- where:
 - it is an internal table
 - wa is a work area having the same structure as it



Copyright © Capgemini 2015. All Rights Reserved 54

The following points apply:

If wa into is specified, the row to be collected is taken from the explicit work area wa. In this case, the header line of the internal table is ignored, if it has one. If wa into is not specified, the internal table must have a header line. The row to be collected is taken from the header line for it.


When collect is executed, the system forms a key from the default key fields in the work area. The default key fields are the character fields (types c, n, d, t, and x). Therefore the key is composed of the values from all fields of type c, n, d, t, and x. It doesn't matter if they are beside one another or separated from each other by other fields.


The system then searches the body of the internal table for a row that has the same key as the key in the work area. If it doesn't find one, the row is appended to the end of the table. If it does find one, the numeric fields (types i, p, and f) in the work area are added to the corresponding fields in the found row.

Remarks: If you use collect to add rows to an internal table, all rows should be added using collect. You should not combine collect with append or any other statement that adds data to an internal table. The results will be unpredictable. The only other statement that you can use to modify the contents of an internal table filled via collect is modify...transporting f1 f2..., where f1 and f2 are numeric fields only (non default key fields).

Demo

- Program Collect





Copyright © Capgemini 2015. All Rights Reserved 55

Control Break Processing

- After you fill an internal table with data, you often need to write the data out.
- This output will frequently contain summary information (such as totals) at the top or bottom of the report.
- To do this, you can read the data into the internal table and then, within loop at, use the following statements:
 - at first / endat
 - at last / endat
 - at new / endat
 - at end of / endat
 - sum
 - on change of / endon

The first statement of each of these statement pairs-except for sum-controls when the code that lies between them is executed. This type of control is called a control break. Their purpose is to execute the code between them whenever a specific condition in the data is detected during the processing of the loop

Using the at first and at last Statements

- Use the at first and at last statements to perform processing during the first or last loop pass of an internal table.
- Syntax for the at first and at last Statements

The following is the syntax for the *at first* and *at last* statements.

```

loop at it.
...
    at first.
    ...
    endat.
...
    at last.
    ...
    endat.
endloop

```

■ where:

- it is an internal table
- ... represents any number of lines of code (even zero)



Copyright © Capgemini 2015. All Rights Reserved 57

The following points apply:

These statements can only be used within a loop at; they cannot be used within select at first does not have to come before at last.

The order of these statements can be interchanged

These statements can appear multiple times within the same loop.

For example, you could have two at first and three at last statements within one loop and they can appear in any order

These statements should not be nested inside of another (that is, at last should not be placed inside of at first and endat

The first time through the loop, the lines of code between at first and endat are executed. The last time through the loop, the lines of code between at last and endat are executed. If there are multiple occurrences of at first, they are all executed. at last behaves in a similar fashion.

Using the at first and at last Statements

- Use at first for:
 - Loop initialization processing
 - Writing totals at the top of a report
 - Write headings
- Use at last for:
 - Loop termination processing
 - Writing totals at the bottom of a report
 - Write footings

Between the at first and endat, or between the at last and endat, the component values of the work area will not contain any data. The default key fields are filled with * (asterisks) and the numeric fields are set to zeros. The endat restores the contents to the values they had prior to entering the at. Changes to the work area within at and endat are lost.

Using the at new and at end of Statements

- Use the at new and at end of statements to detect a change in a column from one loop pass to the next.
- Syntax for the at new and at end of Statements

The following is the syntax for the at new and at end of statements.

```
sort by c.
loop at it.
...
    at new c.
    ...
    endat.
...
    at end of c.
    ...
    endat.
endloop
```

■ where:

- it is an internal table
- c is a component of it
- ... represents any number of lines of code (even zero)



The following points apply:

These statements can only be used within a loop at; they cannot be used within select at new does not have to come before at end of. These statements can appear in any order

These statements can appear multiple times within the same loop. For example, you could have two at new and three at end of statements within one loop and they can appear in any order

These statements should not be nested inside of another (that is, at end of should not be placed inside of at new/endat

There are no additions to these statements

Using at New

- Each time the value of c changes, the lines of code between at new and endat are executed.
- This block is also executed during the first loop pass or if any fields to the left of c change.
- Between at and endat, the numeric fields to the right of c are set to zero.
- The non-numeric fields are filled with asterisks (*).
- If there are multiple occurrences of at new, they are all executed.
- at end of behaves in a similar fashion

Using at New

- A control level is the component named on a control break statement; it regulates the control break.
- For example, in the following code snippet, f2 is a control level because it appears on the at new statement.
- loop at it.

```
    at new f2.  
        " (some code here)  
    endat.  
endloop.
```



Copyright © Capgemini 2015. All Rights Reserved 61

It is said that a control break is triggered if the control level changes. This means that when the contents of the control level change, the code between the at and endat is executed.

A control break is also triggered if any of the fields prior to the control level in the structure change. Therefore, you should define the internal table structure to begin with the fields that form your control levels. You must also sort by all fields prior to and including c.

Between at and endat, numeric fields to the right of the control level will be zero and non-numeric fields will be filled with asterisks.

Using at end of

- The lines of code between at end of and endat are executed:
 - If the control level changes
 - If any field prior to the control level changes
 - If this is the last row of the table

Do not use control break statements within a loop at it where... construct; the effects are unpredictable.

Using the sum Statement

- Use the sum statement to calculate totals for the rows of a control level.

- Syntax for the sum Statement

The following is the syntax for the sum statement.

at first / last / new / end of.

...

sum.

...

endat.

- where:

- ... represents any number of lines of code



Copyright © Capgemini 2015. All Rights Reserved 63

sum calculates a total for the current value of the control level field and all fields to the left of it.

This is clearer if you imagine that it does the following:

It finds all rows that have the same values within the control level field and all fields to the left of it

It sums each numeric column to the right of the control level

It places the totals in the corresponding fields of the work area

Summing can result in overflow because the total is placed into a field of the same length. Overflow causes a short dump with the error SUM_OVERFLOW. When using sum, avoid overflow by increasing the length of numeric fields before populating the internal table.

Using the on change of Statement

- Another statement you can use to perform control break processing is on change of. It behaves in a manner similar to at new.

- Syntax for the on change of Statement

The following is the syntax for the *on change of* statement.

```
on change of v1 [ or v2 ... ].
```

```
---
```

```
[else.
```

```
---]
```

```
endon.
```

- where:

- v1 and v2 are variable or field string names
- ... indicates that any number of or conditions might follow
- --- represents any number of lines of code

Obsolete Statements

- Using HEADER LINE as Work Area.
- Example:

```
TYPES: BEGIN OF line,  
       num TYPE i,  
       sqr TYPE i,  
END OF line.  
DATA it_tab TYPE TABLE OF line WITH UNIQUE KEY col1 WITH HEADER LINE.  
DO 5 TIMES.  
    lt_tab-num = sy-index.  
    lt_tab-sqr = sy-index ** 2.  
    INSERT TABLE it_tab.  
ENDDO.  
lt_tab-sqr = 100.  
MODIFY TABLE it_tab.  
lt_tab-num = 4.  
DELETE TABLE it_tab.
```

Demo

- Program on control break processing



Summary

- In this lesson, you have learnt:
 - To Define an Internal Table and understand its attributes
 - Types of Internal Tables
 - To Add, Read, Update and Delete Data from an internal Table
 - To Sort the Contents of an Internal Table
 - Control break statements on Internal Table



Review Question

- Question 1: _____ is a field string with the same structure as a row of the body, but it can hold a single row.
- Question 2: _____ adds a single row to an internal
- Question 3: _____ variable is used to determine the number of rows in an internal table.
- Question 4: _____ statement accumulates the contents of a structure into an internal table.

