- Internal tables is used to hold temporary data.
- It can hold multiple records at a time
- The data is in structured format in the form of rows and colums.
- Internal table is used for calculations during the execution of the program,
- Life span of the internal table is only during the runtime of the pogram

- Field string contains related fields.
- Field string contains a single record
- Internal table can contain more than 1 record.

**Declaring work area using TYPES.**

```
TYPES: BEGIN OF <name>,
       <field1>, ...,
       <fieldn>,
     END OF <name>.
```

```
TYPES: BEGIN OF gty_cust,
       flag  TYPE c ,
       kunnr TYPE kunnr,
       name1 TYPE name1,
       addr1 TYPE addr ,
     END  OF gty_cust.
DATA: wa_cust_dtl TYPE gty_cust.
```

Declaration of a Line type

- A field string is a group of logically related fields. Other words commonly used interchangeably with field string are “record” or “structure.”
- A field string is typically declared with the TYPES statement.
- The start and end of the field string are indicated by BEGIN OF <name> and END OF <name> statements. Between these statements the definition of the fields is included. (These fields are typically referred to as “sub-fields”.)
- The sub-fields of a field string are defined in the same way as individual fields, specifying length, type, and if appropriate, initial value.
- With the LIKE parameter, it is possible to adopt the attributes of internal fields that have already been declared or the attributes of fields defined in the ABAP Dictionary.

## Declaring work area using Inline Data Declaration

```
SELECT SINGLE  f1,
      f2,
      f3,
      f4
      FROM  <table name>
      INTO  @DATA(<workarea _name>).
```

```
SELECT SINGLE carrid,
       connid,
       fldate
       FROM sflight
       INTO @DATA(wa_sflight)
       WHERE carrid = 'AA'.
```

- Using Inline data declaration, the work area can be declared as and when it is used.
- No need to create structures using TYPES.
- No. of fields in work area is decided at run time.
- @DATA (WA) means the work area is declared at run time.

## Declaring internal table using Inline Data declaration

```
SELECT f1,
      f2,
      f3,
      f4
      FROM  <table name>
      INTO TABLE @DATA(<internal_table _name>).
```

```
SELECT carrid,
       connid,
       fldate,
       seatsmax
       FROM sflight
       INTO TABLE @DATA(it_sflight).
```

- Using Inline data declaration, the internal tables can be declared as and when it is used.
- No need to create structures using TYPES.
- No. of fields in the internal table is decided at run time.
- This technique is useful as the developer need not worry about the addition or deletion of any field in the internal table
- Inline data declaration can be used only for Standard internal tables

## Fill work area with data

```
REPORT zrt_demos.

TYPES: BEGIN OF ty_emp,
         empid   TYPE char2,
         empname TYPE char10,
         empcity TYPE char10,
       END OF ty_emp.
DATA(wa_emp) = VALUE ty_emp( empid = '10'
                             empname ='AAA'
                             empcity ='Mumbai' ).

WRITE : wa_emp-empid, wa_emp-empname, wa_emp-empcity.
```

- Work area can be filled with data using the DATA and VALUE keyword as shown
- DATA keyword is used for inline data declaration and VALUE is used to insert the values in the work area.
- All fields in a field string can be referenced by specifying the name of the field string (e.g. empid).
- To reference a field within a field string, specify the name of the field string followed by a hyphen and then the field name (e.g., wa_emp-empcity).

- From the work area, one record at a time can be used to update/insert/modify the internal tables.
- Similarly, it is possible to move 1 record at a time from internal table to work area and vice versa.

## Declaring an Internal Table using TYPES

Define Field String

Define Internal table referring to Field String

```
TYPES : BEGIN OF <line_type> ,
          <field1>   ,
          <field2>   ,
        END OF <line_type>  .

DATA  : <itab> TYPE STANDARD  TABLE OF <line_type> .
```

```
PROGRAM zrep03_trainer .

TYPES: BEGIN OF gty_mat,
       matnr TYPE matnr,
       maktx  TYPE maktx,
       werks  TYPE werks,
     END  OF gty_mat.

DATA: gt_cust_all TYPE STANDARD TABLE OF gty_mat  INITIAL SIZE 0 .
```

- When declaring an internal table, programmer can specify:
- The table name (mandatory)
- The table type: standard, sorted or hashed (mandatory)
- The field string (TYPE) that defines the layout of each table row (mandatory)
- The initial size (in rows) of the table (mandatory; this is used internally by SAP in order to determine the amount of memory to allocate for the table.)
- A work area to be used to facilitate table manipulation.

- An internal table is specified by its:
- Line TYPE - The line TYPE of an internal table refers to the TYPE statement that defines the field string that will act as the template for each row of the internal table.
- Each row in the internal table will contain all fields specified in this TYPE definition.
- Table type - The table type defines how ABAP accesses individual table entries.
- There are three types of internal tables (standard, sorted and hashed), each with a different access method.
- These will be discussed in more detail later in this module.

- An internal table can be populated with data using the keywords APPEND,INSERT,COLLECT,MOVE,SELECT.

## Filling an Internal Table : VALUE

```
REPORT zrt_demos.
*
TYPES: BEGIN OF ty_emp,
         empid   TYPE char2,
         empname TYPE char10,
         empcity TYPE char10,
       END OF ty_emp.
TYPES: tt_emp TYPE TABLE OF ty_emp WITH EMPTY KEY.
DATA(lt_emp) = VALUE tt_emp(
              ( empid = '10' empname = 'Alphy' empcity = 'Bangalore')
              ( empid = '20' empname = 'Mary' empcity = 'Delhi')
              ( empid = '30' empname = 'Rose' empcity = 'Mumbai')
              ( empid = '40' empname = 'Ancy' empcity = 'Pune')
            ).
```

Output

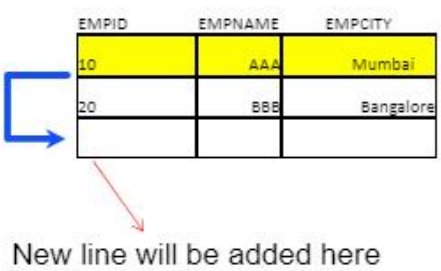
**LT_EMP**

| EMPID | EMPNAME | EMPCITY |
|---|---|---|
| 10 | Alphy | Bangalore |
| 20 | Mary | Delhi |
| 30 | Rose | Mumbai |
| 40 | Ancy | Pune |

- VALUE keyword can be used to insert records directly to an internal table
- No work area is needed
- DATA (internal table ) = VALUE tabletype/# ( (r1f1= value r1f2=value) (r2f1=value r2f2 = value) ) is the generic syntax
- The type of the internal table depends on the tabletype given after the VALUE keyword
- After the above code snippet, contents of the internal table can be displayed using the WRITE keyword.

- When # is used, inline data declaration cannot be used .So we must explicitly declare the internal table before itself.

## Filling an Internal Table : APPEND




```
REPORT zrt_demos.
*
TYPES: BEGIN OF ty_emp,
         empid   TYPE char2,
         empname TYPE char10,
         empcity TYPE char10,
       END OF ty_emp.
Data lt_emp type table of ty_emp.
DATA(wa_emp) = VALUE ty_emp(  empid = '10'
                              empname  = 'AAA'
                              empcity = 'Mumbai' ).
APPEND wa_emp TO lt_emp.

CLEAR wa_emp.
wa_emp = VALUE ty_emp(  empid = '20'
                        empname  = 'BBB'
                        empcity = 'Bangalore' ).

APPEND wa_emp TO lt_emp.
```

- The APPEND statement appends the contents of the work area <wa> at the end of the internal table.
- The maximum number of table entries is determined by the INITIAL SIZE (OCCURS in R/3) parameter

## Filling an Internal Table : INSERT

Used for inserting a new line before current line

```
DATA(wa_mara) = VALUE ty_mara( matnr = '000123'
                               werks ='1001' ).
INSERT  wa_mara INTO lt_mara INDEX 2.
```

Before

| MATNR | WERKS |
|---|---|
| 0000324 | 1002 |
| 0000533 | 5001 |
| 0000293 | 2001 |
| 0000465 | 1006 |

After

| MATNR | WERKS |
|---|---|
| 0000324 | 1002 |
| 0000123 | 1001 |
| 0000533 | 5001 |
| 0000293 | 2001 |
| 0000465 | 1006 |

- If index specified (i.e. row specified ) does not exist in internal table , it will result in abap dump .
- INDEX xx is used to insert the record at a specific row of the internal table

## Filling an Internal Table : COLLECT

COLLECT Statement fills an internal table in such a way that table will contain unique key .

All the non numeric fields ( types: N , C , String ) in an internal table are considered as key for collect statement

```
COLLECT <wa> INTO <itab>.
```

```
DATA(wa_mara) = VALUE ty_mara( matnr = '000123'
                               menge = 60 ).

 COLLECT wa_mara into lt_mara.
```

| | |
|---|---|
| 0000123 | 60 |

Before

| MATNR | MENGE |
|---|---|
| 0000324 | 10 |
| 0000123 | 40 |
| 0000533 | 90 |
| 0000293 | 10 |
| 0000465 | 30 |

After

| MATNR | MENGE |
|---|---|
| 0000324 | 10 |
| 0000123 | 100 |
| 0000533 | 90 |
| 0000293 | 10 |
| 0000465 | 30 |

- COLLECT statement allows the developer to create a unique or summarized dataset.
- The system first tries to find a table entry corresponding to the table key.
- If the system finds an entry, the numeric fields that are not part of the table key are added to the existing entries.
- If it does not find an entry, the system creates a new entry.
- Performance hint: It is better to use APPEND instead of COLLECT because APPEND is considerably faster.
- APPEND, however, always adds a new entry to the table since it does not perform any comparison with existing entries.
- This means that duplicate entries may occur and numeric fields are not added together.
- If data must be collected according to certain key fields, the internal table must be structured appropriately.
- Data declaration part for above code piece is given below:

## Filling an Internal Table: SELECT

```
SELECT  <field1> <field2>  FROM <table> INTO  TABLE <internal table>.
```

```
26
27  SELECT carrid,
28         connid,
29         fldate
30         FROM sflight
31         INTO @DATA(lt_sflight)
32         WHERE carrid = 'AA'.
33
```

- The statement SELECT <field> INTO TABLE <itab> transfers the entire contents of a database table to an internal table in one single statement. Any existing contents of the internal table are overwritten
- The INTO statement has several variants, including:
- INTO CORRESPONDING FIELDS OF TABLE <itab>
- APPENDING TABLE <itab> which will be discussed later
- If at least one record is selected, SY-SUBRC = 0; otherwise, SY-SUBRC = 4.
- @DATA(lt_sflight): Here we need not declare the internal table using TYPES keyword. The structure of the internal table is decided at runtime
- Field list is separated by commas.

## Filling an Internal Table : MOVE

Internal table contents can be copied to another internal table with MOVE and = statements as follows

```
MOVE <itab1> TO <itab2>.
```

```
23
24
25  SELECT lifnr,
26         name1
27         FROM lfa1
28         INTO TABLE @DATA(gt_vend)
29         WHERE land1 = 'US'.
30
31  DATA gt_vend_copy1 LIKE gt_vend.
32
33  MOVE gt_vend TO gt_vend_copy1." Using MOVE
34  MOVE gt_vend TO DATA(gt_vend_copy2)." Incorrect syntax
35  DATA(gt_vend_copy3) = gt_vend. "Using in line data declaration
36
```

- MOVE is used to move contents of one internal to another internal table.
- Here data from gt_vend is moved to the internal table gt_vend_copy.
- While using MOVE, both source and target internal tables must have the same structure.

## Reading Internal Tables

The record pointed by index in the internal table is read into the work area using below syntax

```
DATA(<wa>) = <tab>[ index ].
```

| MATNR | MENGE |
|---|---|
| 0000324 | 10 |
| 0000123 | 40 |
| 0000533 | 90 |
| 0000293 | 10 |
| 0000465 | 30 |

```
DATA(gs_mat) = gt_mat[ 3 ].
```

| | |
|---|---|
| 0000533 | 90 |

```
DATA(gs_mat) = VALUE  #( gt_mat[ 3 ]  OPTIONAL  ).
```

- Using DATA(<wa>) = <tab>[ index ] we can transfer one record from internal table to work area.
- It used to locate a specific record within an internal table and move the contents of that record into work area.
- The index number is mentioned inside the square brackets.
- Here work area  will have the same no. of fields as that of the internal table .
- We need not declare work are explicitly as we are using inline data declaration to declare the work area.
- Note using this approach, the sy-tabix  and sy-subrc are not updated.
- Use the try catch block to handle the errors.
- We can also use the OPTIONAL keyword to avoid runtime error in case the record is not found in the internal table.
- E.g. DATA(gs_mat) = VALUE  #( gt_mat[ 3 ]  OPTIONAL  ) will avoid runtime error even if the record is not found in the internal table

## Reading Internal Tables  with key values

To read a single line from an internal table with a self- defined key, specify the condition inside the braces as shown  below

```
DATA(<tab>) = VALUE  #( <tab>[ <fieldname> = <Value>  ]  OPTIONAL  ).
```

| MATNR | MENGE |
|---|---|
| 0000324 | 10 |
| 0000123 | 40 |
| 0000533 | 90 |
| 0000293 | 10 |
| 0000465 | 30 |

```
DATA(gs_mat) = VALUE  #( gt_mat[ matnr = 0000533  ]  OPTIONAL  ).
```

| | |
|---|---|
| 0000533 | 90 |

- This syntax will read from the internal table based on  the key value specified as condition inside the braces.
- The exact condition to fetch the exact record must be mentioned.
- If there are multiple records with the same partial key value, then first matching record is read .
- The OPTIONAL keyword helps to avoid runtime errors.
- Binary search keyword is not supported. So use  sorted or hashed table types as it will use the binary search or hashed algorithm respectively

## Reading Internal Tables : LOOP AT

LOOP statement is used to read an internal table into work area, line by line (record by record).

```
LOOP AT <itab> INTO DATA(<wa> )
ENDLOOP.
```

```
SELECT lifnr,
        name1
        FROM lfa1
        INTO TABLE @DATA(gt_vend)
        WHERE land1 = 'US'.

IF sy-subrc = 0.

  LOOP AT gt_vend INTO DATA(gs_vend).
    WRITE : / gs_vend-lifnr,
              gs_vend-name1.
  ENDLOOP.

ENDIF.
```


- Ne syntax is LOOP AT <itab> INTO DATA(<wa> ).
- We need not explicitly declare the work area.
- Work area is declared using inline data declaration using the DATA keyword in the LOOP AT statement itself

## Changing Internal Table: MODIFY

MODIFY Statement is used to change contents of an internal table.

```
MODIFY <itab> FROM wa [ index idx ]
```

| | |
|---|---|
| 0000534 | 90 |

```
DATA(wa_mat) = VALUE ty_mara( matnr = 0000534 menge = 30 ).
MODIFY lt_mat FROM wa_mat INDEX 2.
```

| MATNR | MENGE |
|---|---|
| 0000324 | 10 |
| 0000534 | 90 |
| 0000293 | 10 |
| 0000465 | 30 |

| MATNR | MENGE |
|---|---|
| 0000324 | 10 |
| 0000534 | 30 |
| 0000293 | 10 |
| 0000465 | 30 |

- The MODIFY <tab> FROM wa INDEX <i> statement overwrites table line with the contents of the work area.
- The system field SY-SUBRC will be set to zero if the entry is successfully updated or 4 if the entry to be updated does not exist .
- First fill the work area wa_mat with suitable data. Then modify the internal table with the contents of the work area.
- Eg .DATA(wa_mat) = VALUE ty_mara( matnr = 0000534 menge = 30 ).
MODIFY lt_mat FROM wa_mat INDEX 2.

## Changing Internal Table: MODIFY

Code Snippet where MODIFY can be used.

```
MODIFY <itab>  [ index  idx ]
```

```
22
23
24    TYPES: BEGIN OF ty_mara,
25             matnr TYPE matnr,
26             menge TYPE bstmg,
27           END OF ty_mara.
28    TYPES:  tt_mara TYPE TABLE OF ty_mara  WITH KEY matnr.
29
30    DATA(lt_mat) = VALUE tt_mara(
31                         ( matnr = 0000533 menge = 10 )
32                         ( matnr = 0000534 menge = 20 )
33                                  ).
34    DATA(wa_mat) = VALUE ty_mara( matnr = 0000534 menge = 30 ).
35    MODIFY lt_mat FROM wa_mat INDEX 2.
36
```

- Using MODIFY the content of the internal table is updated.
- Suitable index number is mentioned after the keyword INDEX to update a particular record.
- In the above code snippet, the second record is updated as index used is 2
- First work area is filled with suitable data and then from the work area the internal table is updated.

**Changing Internal Table: MODIFY ..... TRANSPORTING**

```
TYPES: BEGIN OF ty_mara,
         matnr TYPE matnr,
         menge TYPE bstmg,
       END OF ty_mara.
TYPES:  tt_mara TYPE TABLE OF ty_mara  WITH KEY matnr.

DATA(lt_mat) = VALUE tt_mara(
                   ( matnr = 0000533 menge = 10 )
                   ( matnr = 0000534 menge = 20 )
                          ).

DATA(wa_mat) = VALUE ty_mara( matnr = 0000534 menge = 30 ).
MODIFY lt_mat FROM wa_mat TRANSPORTING MENGE.
```

| MATNR | MENGE |
|---|---|
| 0000324 | 10 |
| 0000534 | 90 |
| 0000293 | 10 |
| 0000465 | 30 |

| MATNR | MENGE |
|---|---|
| 0000324 | 10 |
| 0000534 | 30 |
| 0000293 | 10 |
| 0000465 | 30 |

- A particular field of the Internal table can be modified using the keyword TRANSPORTING.
- In the example given above, only the MENGE field is updated for the record whose matnr = 0000534.

## Sorting an Internal Table

SORT  statement  sorts the entries in the internal table

```
SORT  <itab> BY  <field1> <field2>  [DESCENDING].
```

Default sorting is always ascending.
SORT without field names sorts the complete record (only non-numeric fields are considered as sort criteria).

- It is possible to sort an internal table with the SORT statement (only for standard internal tables).
- If sort criteria is not specified, the table is sorted by all fields (with the exception of fields with data types P, I, and F) in ascending order in the sequence in which they are declared.
- With the additional specifications BY <field name> and ASCENDING or DESCENDING, the sort process can be restricted to specific fields (or to fields of type P, I, or F) that determines the sorting sequence and hierarchy.
- Whenever possible, limit the number of sort fields with the BY parameter.
- ABAP then requires less storage space in the roll area for the sorting process.
- It is advisable to always specify the fields to be sorted by rather than just SORT.

## Deleting Lines of Internal Tables

DELETE statement is used to delete one or more rows from an internal table. Rows can be deleted by specifying the table key or condition or by duplicate entries.

```
DELETE <itab> [ index idx ]
```

```
DELETE gt_mat index 2.
        OR
DELETE gt_mat where matnr = '0000123' AND menge = 100 .
```

| MATNR | MENGE |
|---|---|
| 0000324 | 10 |
| 0000123 | 100 |
| 0000533 | 90 |
| 0000293 | 10 |
| 0000465 | 30 |

This line gets deleted .

SY-SUBRC set to 0 .
SY-TABIX is set to row number which is deleted.

- Record can be deleted using the DELETE statement
- One record which is mentioned in the wok area is deleted from the internal table.

## Deleting Lines of Internal Tables

DELETE ADJACENT DUPLICATES is a way to delete duplicate lines from an internal table.

It is important to 'SORT' internal table before using this command

```
DELETE ADJACENT DUPLICATES FROM <itab>
 [ COMPARING <field1> <field2> ]
```

**Options:-**

Without the COMPARING option:

- The contents of the standard key fields which are same, get deleted.

With the COMPARING option using selected fields:

- DELETE ADJACENT DUPLICATES COMPARING <f1> <f2> ... ,

With the COMPARING option using all fields:

- DELETE ADJACENT DUPLICATES COMPARING ALL FIELDS ,

- While using DELETE ADJACENT DUPLICATES, the table must be sorted.
- It is a way to delete duplicate lines from an internal table.

## Information About an Internal Table

DESCRIBE statement is used to get information about internal table

DESCRIBE TABLE <itab>... .

## Deleting an Internal Table

For internal table <itab>

CLEAR/REFRESH <itab>
- Deletes all table lines .
- Storage space is not released.

- FREE <itab>
  - Deletes all table lines .
  - Storage space is released.

```
SELECT matnr,
       werks
       INTO TABLE @DATA(gt_mat)
        FROM marc UP TO 50 ROWS.


Describe TABLE gt_mat LINES data(lv_lines)
                      kind data(lv_kind).

WRITE : / 'No. of lines in internal table:',  lv_lines,
        / 'Type ofinternal table:', lv_kind.
```

DESCRIBE statement provides information about an internal table.
The statement DESCRIBE TABLE <itab> LINES <n> KIND <k> returns the number of lines in an internal table and its type .
For details about other parameters of the DESCRIBE TABLE statement, see the online documentation for the keyword DESCRIBE.
NOTE: The optional INITIAL SIZE addition is used to set the initial amount of memory allocated to the table.
Observe '.' in output . It is because of user profile number format which is set as '11.111,00' . So any integer value above 10,000 will be displayed in format 11.111.

## Internal Table Control Break Statements or Events

Control Statements are used within LOOP's for performing calculations/Displaying Summary lists, etc .

Control statements supported by SAP:

| Command | Meaning |
|---|---|
| AT FIRST | At the beginning (first line) of internal table |
| AT LAST | At the end (last line) of internal table |
| AT NEW <f> | Starting of adjacent records having same value in all the fields starting from the beginning (leftmost field) till field <f> |
| AT END Of <f> | End of adjacent records having same value in all the fields starting from the beginning (leftmost field) till field <f> |

- Control break statements is used inside the loop .
- This helps to control the performance and display of the loop
- AT FIRST ,AT LAST AT NEW and AT END of are shown in the next slides

## CONTROL Statements AT FIRST/AT LAST

ZRT_DEMO

```
REPORT zrt_demo.

SELECT matnr,
       werks,
       menge
FROM ekpo
INTO TABLE @DATA(lt_ekpo)
 UP TO 100 ROWS  .

LOOP AT lt_ekpo INTO DATA(wa_ekpo).
  AT FIRST .
    SUM.
    WRITE :/ 'At First executed at line :' , sy-tabix .
    WRITE:/ wa_ekpo-matnr , wa_ekpo-menge .
  ENDAT .

  AT LAST .
    SUM .
    WRITE :/ 'At last executed at line :' , sy-tabix .
    WRITE:/ wa_ekpo-matnr , wa_ekpo-menge .
  ENDAT .
ENDLOOP .
```

- The control break statement AT FIRST is triggered At the beginning (first line) of internal table.
- The control break statement AT LAST is triggered at the At the end(last line) of internal table.
- Here in the demo shown there are 100 records in the internal table, so at last will be triggered at the 100 th iteration

During AT First and AT LAST commands all character, numeric character type fields loose their contents.

## CONTROL Statements: AT NEW

It is important to use SORT command before 'AT NEW' command

ZRT_DEMO

```
SELECT matnr,
       werks,
       menge
FROM ekpo
INTO TABLE @DATA(lt_ekpo) UP TO 8 ROWS  .

DESCRIBE TABLE lt_ekpo LINES  DATA(lv_lines).
WRITE :/ 'No. of lines in internal table :' ,  lv_lines .
SORT  lt_ekpo BY matnr werks .

LOOP AT lt_ekpo INTO DATA(wa_ekpo).
  AT NEW matnr .
    SUM.
    WRITE:/ .
    WRITE :/ 'At NEW matnr  executed at line :' COLOR 3 ,   sy-tabix COLOR 3 .
    WRITE:/ wa_ekpo-matnr , wa_ekpo-werks , wa_ekpo-menge .
  ENDAT .
  AT NEW werks  .
    SUM.
    WRITE:/    .
    WRITE:/'At NEW werks executed at line :' COLOR 5 , sy-tabix COLOR 5 .
    WRITE:/ wa_ekpo-matnr ,wa_ekpo-werks , wa_ekpo-menge .
  ENDAT .
ENDLOOP .
```

- The control statement AT NEW is used at the starting of adjacent records having same value in all the fields starting from the beginning (leftmost field) till field <f>

## CONTROL Statements : AT NEW

| MATNR | WERKS | MENGE |
|---|---|---|
| 0000324 | 1001 | 10 |
| 0000324 | 1001 | 100 |
| 0000324 | 1002 | 90 |
| 0000533 | 1002 | 10 |
| 0000533 | 2001 | 30 |
| 0000533 | 2001 | 40 |
| 0000769 | 2001 | 30 |
| 0000769 | 2002 | 20 |

AT NEW **WERKS** WILL EXECUTE

AT NEW **MATNR** WILL EXECUTE

Observe field WERKS

***Internal Tables***

```
Internal Tables

No. of line in internal table :          8

At NEW matnr  executed at line :         1
0000324            ****            200,000

At NEW werks executed at line :          1
0000324            1001            110,000

At NEW werks executed at line :          3
0000324            1002             90,000

At NEW matnr  executed at line :         4
0000533            ****             80,000

At NEW werks executed at line :          4
0000533            1002             10,000

At NEW werks executed at line :          5
0000533            2001             70,000

At NEW matnr  executed at line :         7
0000769            ****             50,000

At NEW werks executed at line :          7
0000769            2001             30,000

At NEW werks executed at line :          8
0000769            2002             20,000
```

■ During AT NEW command all characters, numeric character type fields, which are on right side of the field used in command, loose their contents.
■Whereas, fields left to field used in command retain their contents.
■ AT NEW command is used for Totals/ Subtotals.
■AT NEW will consider all the fields from the left till the field specified in the statement while comparing the value.

## CONTROL Statements : AT END OF

ZRT_DEMO ▸ loop

```
SELECT matnr,
       werks,
       menge
FROM ekpo
INTO TABLE @DATA(lt_ekpo) UP TO 8 ROWS  .

DESCRIBE TABLE lt_ekpo LINES data(lv_lines).
WRITE :/ 'No. of lines in internal table :' , lv_lines .
SORT  lt_ekpo BY matnr werks .
LOOP AT lt_ekpo INTO data(wa_ekpo) .
  AT END OF matnr .
    SUM.
    WRITE:/ .
    WRITE :/ 'At END OF matnr  executed at line :' COLOR 3 , sy-tabix COLOR 3 .
    WRITE:/ wa_ekpo-matnr , wa_ekpo-werks , wa_ekpo-menge .
  ENDAT .

  AT END OF werks  .
    SUM.
    WRITE:/     .
    WRITE:/'At END OF werks executed at line :' COLOR 5 , sy-tabix COLOR 5 .
    WRITE:/ wa_ekpo-matnr ,wa_ekpo-werks , wa_ekpo-menge .
  ENDAT .
ENDLOOP .
```

- AT END OF is triggered at the end of adjacent records having same value in all the fields starting from the beginning (leftmost field) till field <f>.
- Use SORT command before 'AT END OF' command.

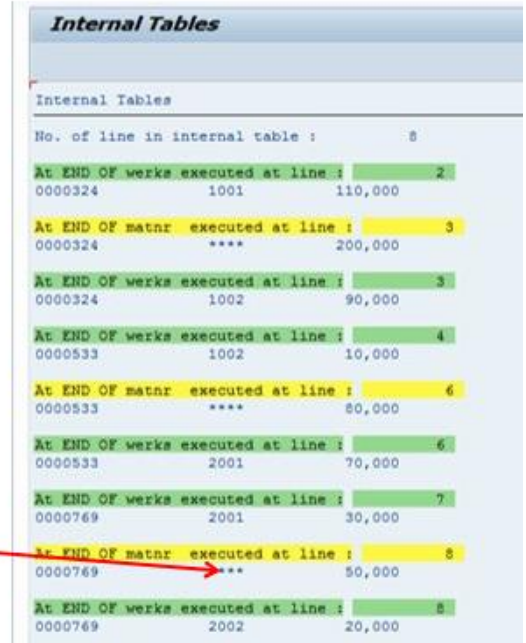
## CONTROL Statements: AT END OF

| MATNR | WERKS | MENGE |
|---|---|---|
| 0000324 | 1001 | 10 |
| 0000324 | 1001 | 100 |
| 0000324 | 1002 | 90 |
| 0000533 | 1002 | 10 |
| 0000533 | 2001 | 30 |
| 0000533 | 2001 | 40 |
| 0000769 | 2001 | 30 |
| 0000769 | 2002 | 20 |

AT END OF **WERKS** WILL EXECUTE

AT END OF **MATNR** WILL EXECUTE

Observe field WERKS




- During AT END OF command all characters, numeric character type fields which are at right side of the field used in command, loose their contents.
- Whereas, fields left to the field used in command retain their contents.
- AT END OF command is used for Totals/ Subtotals.
- AT END will consider all the fields from the left till the field specified in the statement while comparing the value.

- There are three different types of internal tables: standard, sorted and hashed.
- Each of the three table types has different characteristics:
- Standard Table - Has an internal linear index. The system can access records either by using the table index or the key. The response time for key access is proportional to the number of entries in the table. A unique key cannot be specified for a standard table.
- Sorted Table - Table entries are always saved according to the key. For this reason, it takes longer than standard table to insert entries. Sorted tables also have a linear index.
- The system can access records by either the table index or the key.
- The response time for key access is logarithmically proportional to the number of table entries, since the system uses a **binary search**.
- The key of a sorted table can be either unique or non-unique; this must be specified in the table definition.
- **Hashed Table** - Has no linear index. Table entries can only be accessed using a key.
- The response time is independent of the number of table entries and is constant, since the system uses a hashing algorithm (a complex mathematical formula outside the scope of this course).
- The key of a hashed table must be unique; the UNIQUE qualifier must be used when defining a hashed table.
-

## Hashed Table Restrictions

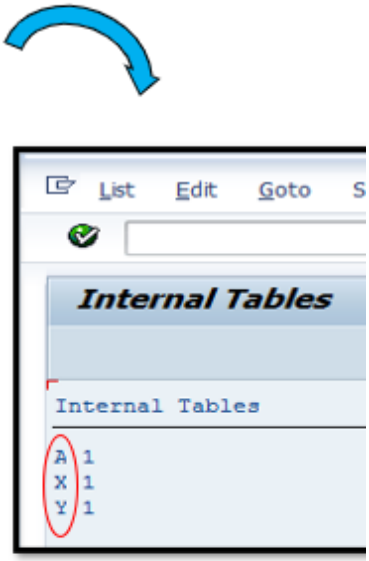
ZRT_DEMOS

```
REPORT zrt_demos.
DATA itab TYPE HASHED TABLE OF scarr WITH UNIQUE KEY carrid.
INSERT VALUE #( carrid   = '100'
                carrname = 'USA Airlines'
                currcode = 'USD') INTO TABLE itab.

INSERT VALUE #( carrid   = '200'
               carrname = 'Indian Airline'
               currcode = 'INR') INTO TABLE itab.
LOOP AT itab INTO DATA(wa).

  WRITE :/ wa-carrid,
           wa-carrname,
           wa-currcode.
ENDLOOP.
```

- Hashed tables have some restrictions on table operations.
- Index operations including APPEND and INSERT…..INDEX cannot be used.
- This table type defines the table which can be managed with an internal hash procedure.
- We can imagine a hashed table as a set, whose elements can be addressed using their unique key.
- Unlike standard and sorted tables, you cannot access hashed tables using an index.
- All entries in the table must have a unique key. Access time using the key is constant, regardless of the number of table entries.
- You can only access a hashed table using the generic key operations or other generic operations ( SORT, LOOP, and so on).
- Explicit or implicit index operations (such as LOOP ... FROM or INSERT itab within a LOOP) are not allowed.

## Sorted Table Restrictions :UNIQUE KEY

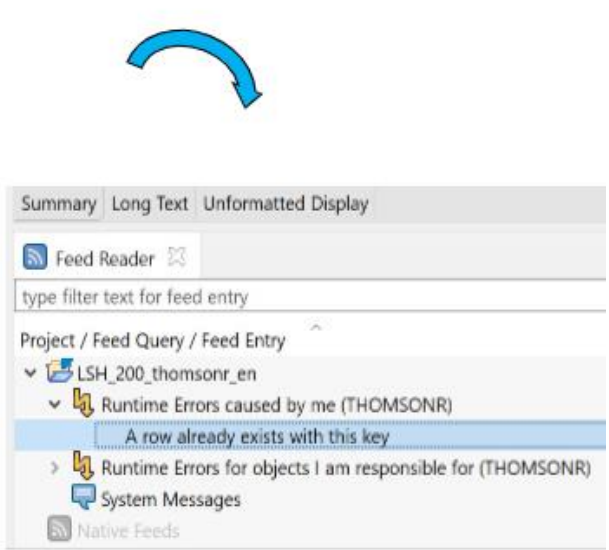
```
REPORT zrt_demos.


TYPES: BEGIN OF ty_demo,
         f1, f2,
       END OF ty_demo.
DATA itab TYPE TABLE OF ty_demo WITH UNIQUE SORTED KEY f1 COMPONENTS table_line.
INSERT VALUE #( f1   = 'A'
                f2   = '1'
                ) INTO TABLE itab.

INSERT VALUE #( f1   = 'X'
                f2   = '1'
                ) INTO TABLE itab.
INSERT VALUE #( f1   = 'Y'
                f2   = '1'
                ) INTO TABLE itab.
LOOP AT itab INTO DATA(wa).

  WRITE :/ wa-f1,
           wa-f2.
ENDLOOP.
```




- Sorted tables cannot be filled using index operations (INSERT wa INTO tab1 INDEX tabix, APPEND tab1 FROM wa) because they can disrupt the sort sequence.
- A sorted table cannot be resorted, or it will cause a syntax check in the ABAP Editor.
- In the demo shown, the addition `WITH UNIQUE SORTED KEY f1` ensures that only usnique values of f1 are added to the table
- Try the same demo code by inserting same value of f1. See the next slide.

## Sorted Table Restrictions

```
1  REPORT zrt_demos.
2
3
4  TYPES: BEGIN OF ty_demo,
5           f1, f2,
6         END OF ty_demo.
7  DATA itab TYPE TABLE OF ty_demo WITH UNIQUE SORTED KEY f1 COMPONENTS table_line.
8  INSERT VALUE #( f1   = 'A'
9                  f2   = '1'
10                 ) INTO TABLE itab.
11
12 INSERT VALUE #( f1   = 'X'
13                 f2   = '1'
14                 ) INTO TABLE itab.
15 INSERT VALUE #( f1   = 'A'
16                 f2   = '1'
17                 ) INTO TABLE itab.
18 LOOP AT itab INTO DATA(wa).
19
20   WRITE :/ wa-f1,
21            wa-f2.
22 ENDLOOP.
23
```




- When f1 = A is entered as the third record, it throws runtime error.
- The error message is shown in the pic.
- This is because of the addition with UNIQUE SORTED key

- The SELECT Statement is the most important Open SQL statement. It can be used to retrieve data from one or more database tables.
- The above slide shows the general syntax of the SELECT statement.- Some simple variations are covered in subsequent sections and
- Use the SELECT statement to read table entries (records) from a table (database).

**Note:** Open SQL is a set of SQL statements that have been incorporated into the ABAP language to aid database access and manipulation.

- The common ANSI SQL equivalents of “SELECT”, “INSERT”, “UPDATE” and “DELETE” are available in Open SQL as well with a few variations.
- The variations, often provide, powerful data-handling capabilities and are integrated completely into the ABAP language.

## SELECT Statement - Basic Form

ZRT_DEMO_VIDEOS

```
1  *&---------------------------------------
2  *&Report zrt_demo_videos
3  *&-------------------s---------------------
4  *&
5  *&---------------------------------------
6  REPORT zrt_demo_videos.
7  SELECT *
8   FROM t001
9   INTO TABLE @DATA(gt_t001).
10 CHECK sy-subrc EQ 0.
11 cl_demo_output=>display( gt_t001 ).
12
```

As the
the rec
fetched

- Use the SELECT statement to read table entries (records) from a table (database).
- Because the SELECT statement can potentially read more than one record from a table, INTO TABLE statement is required to build the table.
- The structure (field names, field attributes) of a table can be displayed via the Help icon (or Utilities -> Help on...).
- The System field SY-SUBRC holds a return code from the SELECT operation. It will have a value 0 if one or more table entries are retrieved or 4 if no table entries are retrieved (table is empty). In the SELECT loop, the SY-SUBRC value should always be 0.
- Selecting all the fields ( * ) is not recommended.

## Restricting Data Selection - WHERE clause

ZRT_DEMO_VIDEOS

```
1  *&---------------------------------------------------------------------
2  *&Report zrt_demo_videos
3  *&------------------s--------------------------------------------------
4  *&
5  *&---------------------------------------------------------------------
6  REPORT zrt_demo_videos.
7  SELECT *  "Will fetch all the fields from DB table
8   FROM t001
9   INTO TABLE @DATA(gt_t001)"Putting data into internal table
10  where bukrs = 1000. "Filtering for Bukrs = 1000.
11 CHECK sy-subrc EQ 0.
12 cl_demo_output=>display( gt_t001 ).
13
14
15
```

WHERE clause will restrict th data selection to the conditio mentioned

Possible comparison operators for WHERE conditions.
EQ =
NE <>
LT <
LE <=
GT >
GE >=

- The WHERE clause restricts the quantity of table entries to be read. In the example shown above, only table entries where the table field T001-BUKRS has the value '1000' will be read.
- The field specified in the WHERE clause to the left of the relational operator must be a table field from the table to be read. It must be referenced here without the table name (BUKRS instead of T001-BUKRS).
- The value of SY-SUBRC should always be checked after any operation against a database table to check whether the operation was successful and then take the appropriate action, for example, write out a suitable a message.
- SY-SUBRC value 0 indicates at least one matching record was found.

## Retrieving a Single Record - SELECT SINGLE

```
3  *&---------------------------------------------------------------------
4  *&
5  *&---------------------------------------------------------------------
6  REPORT zrt_demo_videos.
7
8  SELECT  single *  "Will fetch single record from DB table
9   FROM MARC
10  INTO @DATA(gw_marc)"Putting data into work area
11  where matnr ='0000000000001'
12  AND  werks  = '1000'.
13 CHECK sy-subrc EQ 0.
14 cl_demo_output=>display( gw_marc ).
15
```

INTO TABLE cannot be us fetching SINGLE RECOR

- When the values of all key fields of a table entry are known, use the SELECT SINGLE form of the SELECT statement.
- In this variation of the SELECT statement, all key fields must be specified in the WHERE clause.
- Since this variation of the SELECT statement can retrieve at most only one entry from a table, **no corresponding ENDSELECT statement is required**.
- The SELECT SINGLE allows the user to access a single table entry from the database table.
- The parameters it requires are:
  - The database table from which the record is to be selected
  - The condition in the WHERE clause, using the full PRIMARY key of the table entry
- It then locates the table entry meeting the WHERE condition in the table work area.
- Its return code (SY-SUBRC) is:
  - 0: if finds a table entry
  - 4: if table entry does not exist as per the where condition.

## Retrieving a Single Record ...UP TO <n> ROWS

```
*&---------------------------------------------------------------------
*&
*&---------------------------------------------------------------------
REPORT zrt_demo_videos.

SELECT *  "Will fetch all fields from DB table
 FROM marc
 UP TO 1 ROWS
 INTO @DATA(gw_marc)"Putting data into work area
 WHERE werks  = '1000'.
ENDSELECT.

Cl_demo_output=>display( gw_marc ).
```

1. UP TO 1 ROWS will Fetch a single record
2. SELECT – ENDSELECT Creates a loop
3. Try this select by removing UP TO....

- The UP TO <n> ROWS addition enables the user to determine how many rows matching the selection criteria should be selected.
- ENDSELECT is required even if only one row is accessed.
- Only the first row matching the selection criteria will be chosen.
- PRIMARY KEY in the WHERE Clause is not mandatory

## Retrieving Records in Sequence - ORDER BY

"ORDER BY PRIMARY KEY:

```
*&---------------------------------------------------------------------
REPORT zrt_demo_videos.

SELECT *  "Will fetch all fields from DB table
 FROM marc
 INTO table @DATA(gt_marc)"Putting data into work area
 ORDER BY PRIMARY KEY.

Cl_demo_output=>display( gt_marc ).
```

"ORDER BY f1 f2"

In above Query mention ORDER BY matnr. And observe the difference in output.

- With the addition of ORDER BY, the SELECT statement will retrieve records from a table in a specified sequence.
- With the addition ORDER BY PRIMARY KEY, the records are retrieved in ascending sequence by the primary key fields.
- With the addition ORDER BY f1 f2 ..., the records are retrieved in ascending sequence by the specified fields. It is also possible to retrieve records in descending sequence by specifying the keyword DESCENDING after the relevant field name(s).
- For example: SELECT * FROM SFLIGHT ORDER BY CONNID DESCENDING FLDATE.

## Selecting required Columns

```
*&---------------------------------------------------------------------
REPORT zrt_demo_videos.

SELECT matnr,
       werks  "Will fetch 2 fields matnr and werks from DB table
 FROM marc
 INTO table @DATA(gt_marc)"Putting data into work area
 WHERE  werks = '1000'
 ORDER BY MATNR DESCENDING.

Cl_demo_output=>display( gt_marc ).
```

- Instead of using * which selects all the fields , we can select only required fields by mentioning them in select statement and declaring in Structure.
- The statement SELECT <tab> INTO TABLE <i_tab> transfers the entire contents of a database table to an internal table in one single statement.
- Any existing contents of the internal table are overwritten.
- This SELECT statement is executed as a single statement, not a loop, so no ENDSELECT statement is needed.
- The INTO statement has several variants, including
  - INTO CORRESPONDING FIELDS OF TABLE <i_tab> . " If there is mismatch between fields mentioned in SELECT and Target Structure ( Internal table / Work area ) then this will map the data to corresponding fields . But using this syntax is not advisable due to performance issue
  - If at least one record is selected, SY-SUBRC = 0 otherwise, SY-SUBRC = 4.

## Retrieving Individual Columns

```
*&
*&---------------------------------------------------------------------*
REPORT zrt_demo_videos.

SELECT single matnr,
              werks  "Will fetch 2 fields matnr and werks from DB table
 FROM marc
INTO ( @DATA(lv_matnr),
       @DATA(lv_werks)
     ).
cl_demo_output=>display( |{ lv_matnr } { lv_werks }|
```

Individual variables are used

- To select individual columns from a database table, use the above form of the SELECT statement.
- Immediately following the SELECT keyword, list the fields (columns) to be retrieved.
- After the INTO clause, specify the individual variable names using INLINE Data declaration.
- The fields must be enclosed in parentheses .
- The fields must be separated by a comma (as above).
- Alternatively, it is possible to specify an internal table to be populated. This will be discussed in later modules.
- Retrieving individual columns is better performing than using SELECT *. (Use Transaction SE30, and Utilities -> Tips and Tricks.)

## Aggregate Expressions

```
REPORT zrt_demo_videos.

SELECT DISTINCT werks
 FROM marc
INTO  @DATA(lv_werks).
ENDSELECT.

cl_demo_output=>display( lv_werks ).

```

- To exclude duplicates when retrieving data with the SELECT statement, use the addition DISTINCT.
- In the example above, the statements within the SELECT ... ENDSELECT processing loop will only be executed once for each unique value of the field MARC-WERKS

## Aggregate Expressions Contd..

```
*&---------------------------------------------------------------------*
REPORT zrt_demo_videos.

SELECT COUNT( DISTINCT werks ),
MAX( VKTRW ),
MIN( VKTRW ),
AVG( VKTRW ),
SUM( VKTRW )
 FROM marc
INTO (@DATA(lv_werks),
      @DATA(lv_vktrw_max),
      @DATA(lv_vktrw_min),
      @DATA(lv_vktrw_avg),
      @DATA(lv_vktrw_sum)
      ).
cl_demo_output=>display( |{ lv_werks } { lv_vktrw_max } { lv_vktrw_min } { lv_vktrw_avg } { lv_vktrw_sum }|
```

- A number of aggregate expressions can be specified as part of the SELECT statement.
- The aggregate expressions currently available are:
- AVG( ) - determines the average value of the specified field
- COUNT( DISTINCT <f1> )- determines the number of unique values that exist for the specified field <f1>
- COUNT( * ) - determines the total number of lines retrieved
- MAX( ) - returns the highest value found for the specified field
- MIN( ) - returns the lowest value found for the specified field
- SUM( ) - returns the sum of all values for the specified field
- The previously mentioned additions to the SELECT statement (WHERE, etc.) can be combined with these aggregate expressions.

## SELECT- FOR ALL ENTRIES

- Used to Select data from DB for all the records available in internal table
  - Example : We select material and plant from MARC and now we want description for all these materials from MAKT table .
- Duplicate lines are automatically removed from the resulting set.
- Data from a database table matching the entries in internal table 1 (specified with the FOR ALL ENTIES clause) matching the WHERE clause is fetched to internal table.
- If the internal table specified with FOR ALL ENTRIES is empty, all rows in database table are retrieved to the target internal table, so always check NOT INITIAL on Internal table being used.
- See the demo code snippet in the next slide.

- Adding a FOR ALL ENTRIES statement to a SELECT statement first compares all entries of one internal table to those of a database table.
- Next, in one step, it populates another internal table with the contents of the database table matching the WHERE criteria.
- This type of SELECT does not loop through a database table and does not require an ENDSELECT, due to INTO TABLE notation.
- If the internal table used with FOR ALL ENTRIES is empty, all rows from the database table are retrieved.
- Always check SY-SUBRC to ensure that there is data in the first internal table before executing a FOR ALL ENTRIES statement on it!!

## SELECT- FOR ALL ENTRIES

```
*&---------------------------------------------------------------------*
REPORT zrt_demo_videos.

SELECT carrid, connid
       FROM spfli
       WHERE cityfrom = 'Mumbai'
       INTO TABLE @DATA(entry_tab).

IF entry_tab IS NOT INITIAL.
  SELECT carrid, connid, fldate
         FROM sflight
         FOR ALL ENTRIES IN @entry_tab
         WHERE carrid = @entry_tab-carrid AND
               connid = @entry_tab-connid
         INTO TABLE @DATA(result_tab).

  cl_demo_output=>display( result_tab ).
ENDIF.
```

Fetching from table 1

Fetching matching records from table 2

- Using FOR ALL ENTRIES, matching records can be queried from 2 tables,
- In this demo example, data is first fetched from table SPFLI and put into the internal table entry_tab.
- If internal table entry_tab is empty , then table SFLIGHT is not queried.
- If the internal table entry_tab has data, then matching records are queried from table SFLIGHT .

- In a relational data structure (RDBMS), it is quite normal that related data is split up across several tables to help standardization (i.e. normalization).
- As a result, it is often necessary to retrieve data from several tables to satisfy a query.
- A JOIN is used to link several logically related tables together so that data can be retrieved from those using a single SQL command.
- There are two types of joins in ABAP, INNER and LEFT OUTER JOIN.
- These specifications determine the way the tables are to be linked.
- Joins are more efficient than logical database and nested selects.

## Inner Join

- It joins the columns (specified in the select statement) from the table on the left side with the columns specified from the right table if the join condition is satisfied jointly.

- If the join condition does not match either on left or right-side tables, no record is selected in the resulting set.

- The INNER JOIN provides the capability to produce a set of results from multiple tables via one SELECT statement (as opposed to nested selects). (Views created in the Data Dictionary are inner joins).
- In order to determine the result of a SELECT statement where the FROM clause contains a join, the database system creates a temporary table containing the records that meet the ON condition. The WHERE condition is then applied to the temporary table. It does not matter in an inner join whether the condition is in the ON or the WHERE clause.
- Using a join has the added advantage that only one statement is executed in the database. A disadvantage is that redundant data from the outer table appears in the result if there is a 1:N relationship between the outer and inner tables. For this reason, when a join is used, only necessary fields should be specified.
- If field names are used in the field list that occur in both tables of the inner join, they must be made unique by either prefixing them with an alias name and a tilde sign OR the table name and a tilde sign.
- The performance of a join depends heavily on the performance of the database optimizer that is used, especially if there are more than two tables in the join.

## Left Outer Join

- It creates similar result set as inner join.
- However, at least one record is created in the result set for each line from the left table satisfying the where condition even if no record in the right table satisfies the join condition.

- The columns from the right table in the result set, if the join condition are not satisfied, are filled with NULL.

- None of the columns from the right table should be there in the WHERE condition.

- Unlike an INNER JOIN, the result of a LEFT OUTER JOIN depends on whether the condition is specified in the ON or the WHERE clause. Without a WHERE clause, the resulting table will always contain all the lines of the left table, whereas with a WHERE clause, records from the left table can be taken out.
- Table entries in the join gives result even if there is no corresponding record in the right-hand side table. This is equivalent of the nested SELECT statement in the result set.

## Left Outer Join

```
*&
*&---------------------------------------------
REPORT zrt_demo_videos.

SELECT a~matnr,
       a~werks,
       b~mblnr
 INTO  TABLE @DATA(gt_marc_mseg)
 FROM marc as a LEFT OUTER JOIN
      mseg as b
 ON a~matnr EQ b~matnr
 WHERE a~matnr ='1000'.

cl_demo_output=>display( gt_marc_mseg ).
```

In this case we will find two types of records

1. Where material from left side table exists in right side table also.
2. Where material on left side table doesn't exists in right side table.

Now add mblnr from right side in ON Condition:

- ON a~matnr EQ b~matnr and b~mblnr = '1000000259'

In this case we will find two types of records

1. where material from left side table exists in right side table AND mblnr is '1000000259' .
2. where material on left side table doesn't exists in right side table.

- A left outer join contains the union of rows from one table to another table that meet logical condition(s) defined for the join.
- A left outer join is a quasi union because rows on the left side table, that don't correspond with rows on the right side table, are appended onto the "union" with NULL values where rows from the other table should have been.
- In order to determine the result of a SELECT statement where the FROM clause contains an outer join, the database system creates a temporary internal table containing the rows that meet the ON condition. The remaining rows from the LHS are then added to this internal table and the corresponding fields from the RHS are filled with NULL values. The WHERE condition is then applied to this internal table.

## Syntax: Subquery

- Select statement that is used within where clause of another Select statement

```
*&
*&-------------------------------------------------------------------
REPORT zrt_demo_videos.

SELECT  vbeln,
        posnr,
        matnr
 FROM vbap
 INTO  TABLE @DATA(it_vbap)
 WHERE POSNR IN (
                 SELECT  POSNR
                    FROM vbap
                    WHERE POSNR ='000010'
                 ).

cl_demo_output=>display( it_vbap ).
```

Outer query

Inner query or subquery

- Here a select query is used inside another select query.
- Based on the result returned by the inner subquery , the outer select statement fetches the final output result set .

## Syntax: RANGES

- Used to create internal tables that have the structure of selection tables.
- Use these tables with certain restrictions same as actual selection tables.

ZRT_DEMO_VIDEOS ▸ LWA_WERKS

```
REPORT zrt_demo_videos.

DATA: r_werks   TYPE RANGE OF werks_d, "To declare internal table that have the
                                       "structure of selection tables.
      lwa_werks LIKE LINE OF r_werks.

lwa_werks-low    = '1000'.
lwa_werks-option = 'EQ'.
lwa_werks-sign   = 'I'.
lwa_werks-high   = '1001'.
APPEND lwa_werks TO r_werks .

SELECT  a~matnr,
        a~werks,
        b~mblnr
        INTO TABLE @DATA(gt_marc_mseg) "In line Data declaration
        FROM marc AS a LEFT OUTER JOIN
             mseg AS b
        ON a~matnr EQ b~matnr
        WHERE a~werks IN @r_werks. "Use @ for the host variable


cl_demo_output=>display( gt_marc_mseg ).
```

- The most convenient way of defining an internal table for the IN operator is with the RANGES statement. For this, name and reference fields are required.
- This table definition is similar to a DATA statement for an internal table with the columns SIGN, OPTION, LOW and HIGH.
- The LOW field holds the low value of the range.
- The HIGH field holds the highest value in the range.
- The OPTION field holds the operator for this range. Valid operators are EQ, NE, CP, NP, GE, LT, LE, or GT.
- The SIGN field holds the value I or E. When the sign is I, it indicates the range is inclusive, i.e. comparisons with any value that is within the range is true. When the sign is E, it indicates the range is exclusive, i.e. comparisons with any value within the range is false.

When using operators such as not greater than or not equal, there is no high value to the range, only low value exists.

1. Classical report is a report (type 1) program. It can have a selection screen. I has an output
2. Events of classical report are
   - Initialization
   - At Selection Screen Output
   - At Selection Screen
   - Start of Selection
   - End of Selection
   - Top of Page
   - End of Page
3. AT selection screen
4. SELECT SINGLE should be used when all the keys are specified in the where clause. “SELECT UPTO “ statement should be used when all the keys are not specified in the where clause
5. Select <fields> from <db_table> into <target_internal_table> for all entries in <internal_table1> where <where_condition>. <internal_table1> should not be empty.