Introduction to ABAP Programming

- ABAP stands for Advanced Business Application Programming.
- ABAP is the programming language of SAP and the runtime environment needed to run a ABAP program is provided by the Application layer.

1. Where we will write the ABAP Code ?

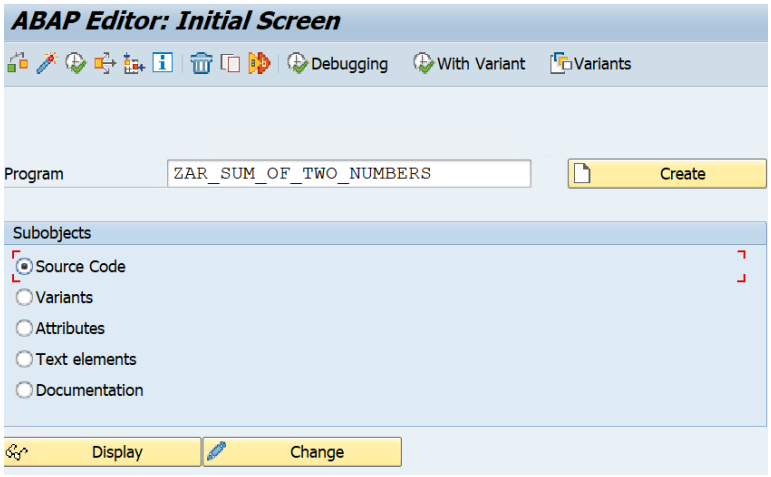
- ABAP Editor is a tool for ABAP coding.
- It is the most important tool of ABAP Workbench (ABAP workbench is collection of ABAP tools like ABAP Dictionary (SE11), ABAP Editor (SE38), Function Builder (SE37), Class Builder(SE24) etc.
- The transaction code for ABAP Editor is SE38.
- Any customized program must start with Z or Y as first alphabet.

---

2. Creating First ABAP Program

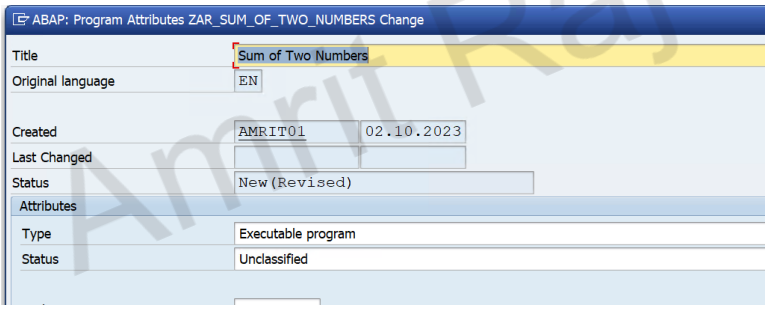
Creating a program for Sum of two Numbers

- Step 1 :- Go to SE38 transaction code and give a name for your program.



  - Click on create button.

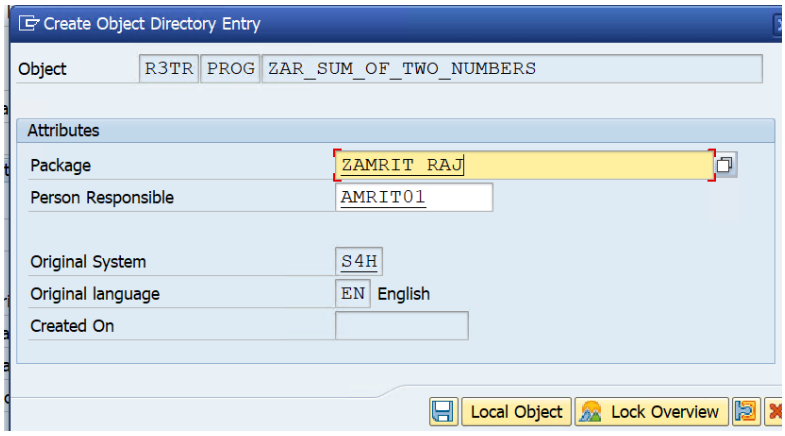
- Step 2 :- Provide the title of the program and in the type section, select executable program.

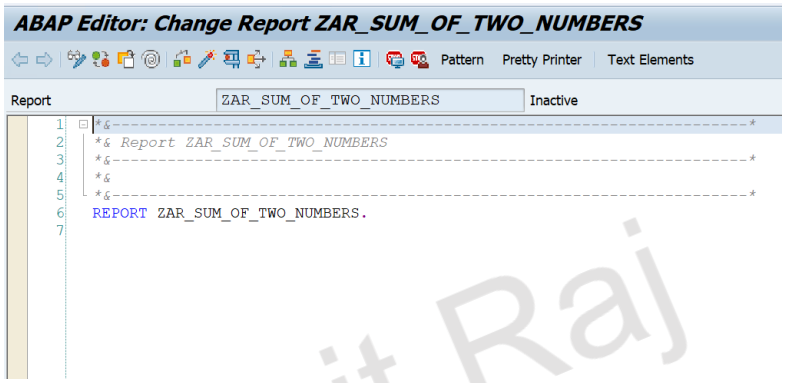


Executable program :-

  - If you want to create a program which can be executed(i.e. you can run the program and get result), then you will select executable program in type section.

- Step 3 :- Click on save button and assign the package and transport request.





- Now your program has been opened in change mode.

1. Important Point Related to ABAP Code :-

- A statement is a sequence of words that ends with a period (dot).

  e.g. REPORT ZAR_SUM_OF_TWO_NUMBERS.

- In the ABAP Editor, the keywords appear in the blue color.

  e.g. we can see the REPORT keyword is appearing in blue color.

Note :-

- If your program name starts with REPORT keyword, it means that your program is a executable program.

2. Three important things we need to remember before executing any program :-

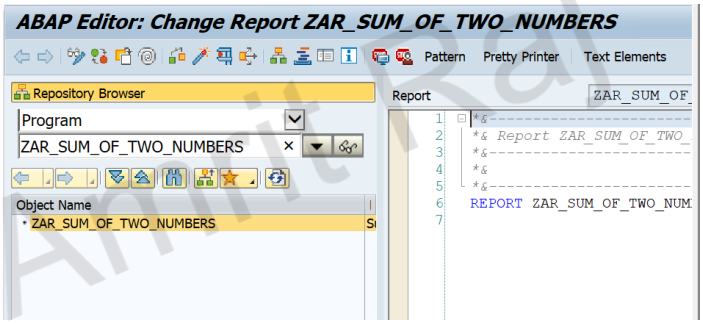
1. To save the program click on save button or use shortcut key Ctrl + S.

2. To check the syntax use the shortcut key Ctrl + F2.
3. To activate the program use the shortcut key Ctrl + F3.

3. Some Important buttons :-

- To change the program in read/write mode, we use the below buttons.

- Display Object List :- We can navigate to SE80 transaction code from our program by clicking on display object list button.



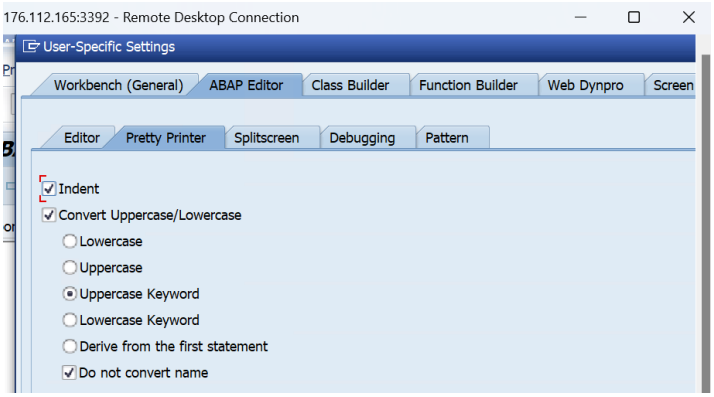
4. Some more Functionalities of ABAP Editor :-

- Execute :- F8
- Display/Change :- Ctrl + F1
- Display Object List :- Ctrl + Shift + F5

5. Pretty Printer

- It is a functionality available in ABAP Editor which is used to format the ABAP code. It increases the readability of the code.
- The short cut for pretty printer is Shift + F1.
- The various functionalities of Pretty Printer are as follows :-

- Indentation :- It helps in proper formatting of the code.
- Convert Uppercase/lowercase :- It helps us to decide whether the keywords and rest statement will be in lowercase or uppercase



- To check Pretty Printer setting, Go to Utilities → Settings → Pretty Printer.

---

3. Comments in ABAP Programming

- A comment is an explanation that is added to the source code of a program to help the person reading the program to understand it.
- Comments are ignored when the program is generated by the ABAP compiler.
- The * character at the start of a program line indicates that the entire line is comment.
- The “character, which can be entered at any position in the line, indicates that the remaining content in the line is a comment.
- Comment our lines - Ctrl + ,
- Uncomment lines - Ctrl +.

```
**********************
*Examples of if statement
if sy-subrc eq 0.
    write : 1, 2.
endif. "End of if statement
```

```
************
```

Note :- Don’t worry, much about the code at this moment, just try to understand how things are going, it will get easier to understand with time.

- As you can see, in the above code, how we have used * and “ to use single line comments.

How to comment and uncomment Multiple lines ?

- For commenting and uncommenting multiple lines at the same, select all the lines and press Ctrl + , to comment and Ctrl + . to uncomment.

---

4. Concept of Data Types

- Data types are just like templates which are used for creating data objects.
- Data types defines the technical attributes (types and length) of data objects.
- Data types do not use any memory space.
- Data types can be defined independently in the ABAP program or in ABAP Dictionary (Go to SE11 → select data type radio button).

5. Data Objects

- A data object is an instance of a data type.
- A data objects holds the content or data.
- It occupies the memory space based upon the data type specified.

```
*******************************
*Example:-
DATA lv_empid(20) type n,
lv_empid = 10.
*******************************
```

- In the above example, DATA = keyword, lv_empid is the name of the data object, TYPE = keyword, n(numeric) = is a data type of that data object.

---

6. Categories of Data Types

- There are three categories of data types.
    1. Elementary Types
    2. Complex Types
    3. Reference Types

1. Elementary Types

- They are predefined data types i.e. they are given by SAP itself.
- They are single data types and are not composed of other data types

Types of Elementary Types

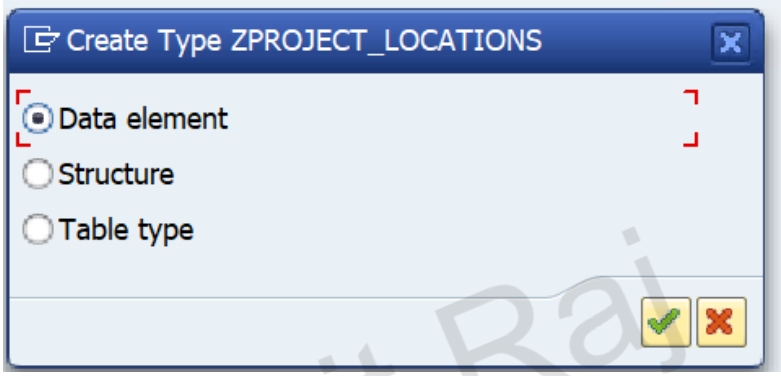
- There are two types of Elementary Data types.
    1. Fixed length data types
        a. C(character)
        b. N(numeric)
        c. I(Integer)
        d. P(packed number)
        e. F(floating point)
        f. D(date)
        g. T(time)
        h. X(hexadecimal)
    2. Variable length data types
        a. String
        b. XString

2. Complex Data Types

- There is no pre-defined complex data type in ABAP.
- They are the combination of Elementary Data Types.

Types of Complex Data Types

1. Structure Type
2. Table Type



3. Reference Data Types

- It is also not a pre-defined data type.
- It describes data objects that contain reference to other objects.

Types of Reference Data Types

1. Data Reference
2. Object Reference

Example

DATA : LO_OBJECT TYPE REF TO ZCLASS.

- In the above syntax
- LO_OBJECT :- Name of data object
- TYPE REF TO :- keyword

- ZCLASS :- Name of already existing class.

Data Objects, Parts of a Program, Write Statement, Chain Operator

1. Data Objects

- Data objects is an instance of a data type which holds the content or data.

1. Types of Data Objects

- There data objects are of two types :-
    1. Literals (Unnamed data objects)
    2. Named Data objects

Literals :-

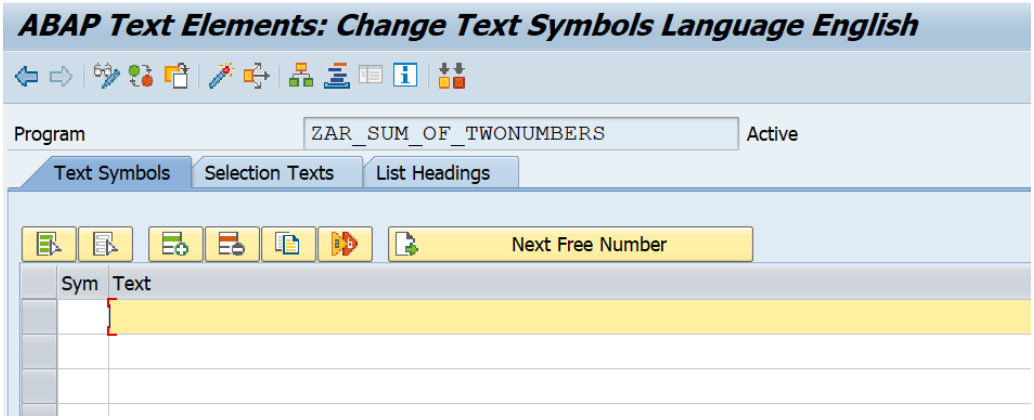
- Literals don’t have any name that’s why they are called as unnamed data objects.
- They are fully defined by their value.
- There are 2 types of Literals.

1. Numeric Literals :-
    - Numeric literals are sequence of numbers. example - 123 , - 4567
    - They can contain both + and minus sign.
2. Character Literals :-
    - Character literals are sequences of alphanumeric characters in single quotation marks.

      Examples :- ‘ABAP 123’, ‘SAP ABAP Development’ etc.

Named Data Objects

- Data Object that have a name are called as named data objects.
- The various types of named data objects are as follows :-
    1. Variables
        - Variable is a named data objects whose contents can be changed.
        - Variables are declared using the DATA, CLASS-DATA, STATICS, PARAMETERS, SELECT-OPTIONS, and RANGES statements.
        - Example :- DATA : LV_EMPLOYEE_ID(4) type n.
            - LV_EMPID = 10
            - LV_EMPID = 20
    2. Constants
        - Constants are data objects whose contents can not be changed.
        - Constants are declared using the CONSTANTS keyword.
        - Example :- CONSTANTS : LCL_PI type P DECIMALS 3 value ‘3.141’.
    3. Text Symbols
        - A text symbol is a named data object that is not declared in the program itself but it is defined as a part of the text element of the program.
            - Just Click on GOTO → TEXT ELEMENTS



---

2. Parts of a Program

- Generally in any programming language, a program have 3 parts :-
  1. Input
     - Used to take input from the user.
     - PARAMETERS and SELECT-OPTIONS are used to take input from the user.
  2. Processing Logic
     - Used to Perform operation over the inputs given by the user.
  3. Output
     - Used to show the final result to the end user.
     - In case of Classical Report, Write statement is used to display the output.

---

3. Writing Sum of two Numbers Program

- We will declare three variable out of which two will be input variable and one will be output variable.

```
*&Creating Input variable.
DATA lv_input1(2) type n. "First Input
DATA lv_input2(2) type n. "Second Input
```

```
*&Creating output variable.
DATA : lv_output(3) type n.
```

- Firstly, we will give default inputs here, later on we will take inputs from the user.

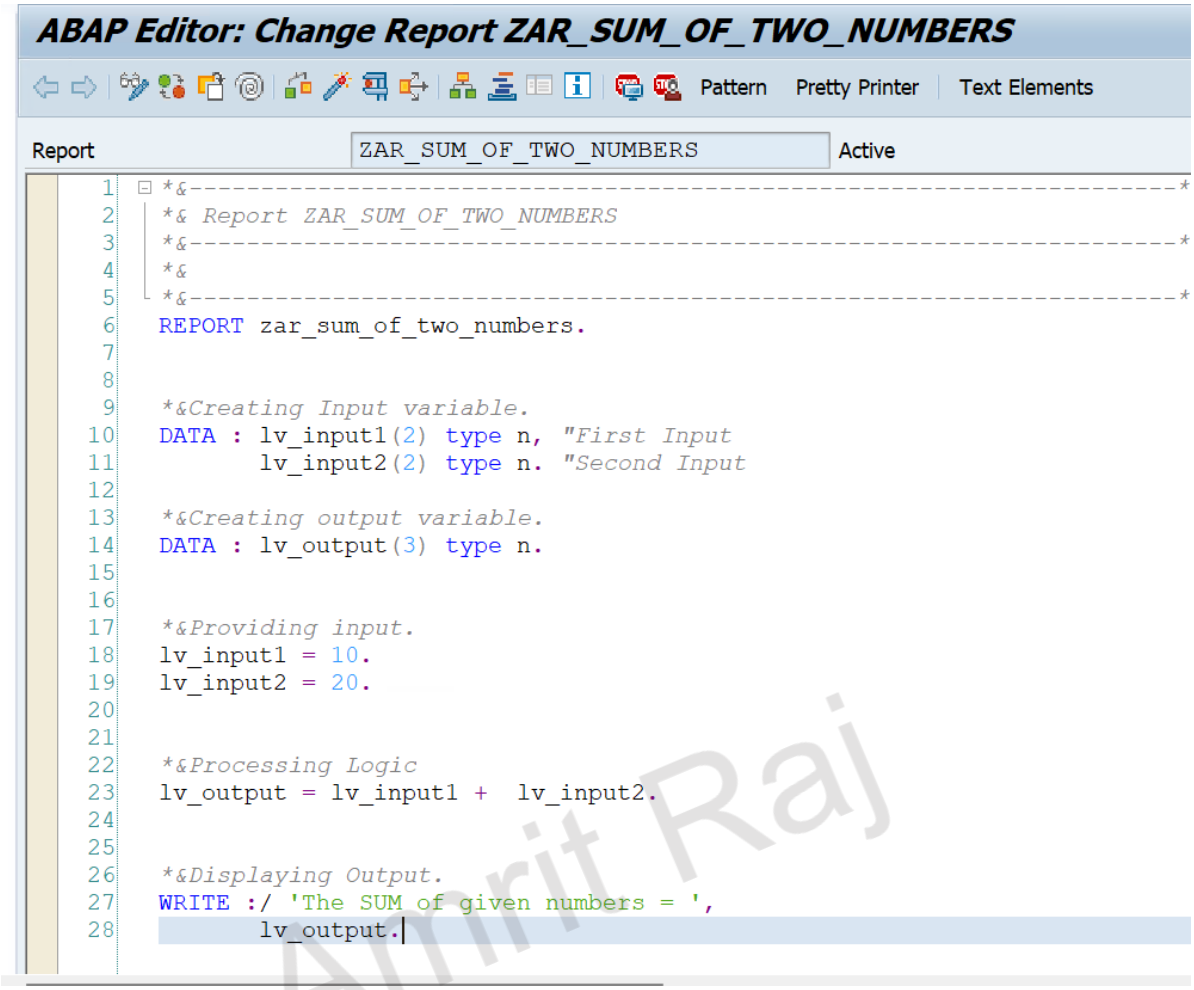
```
*&Providing input.
lv_input1 = 10.
lv_input2 = 20.
```

- Processing logic

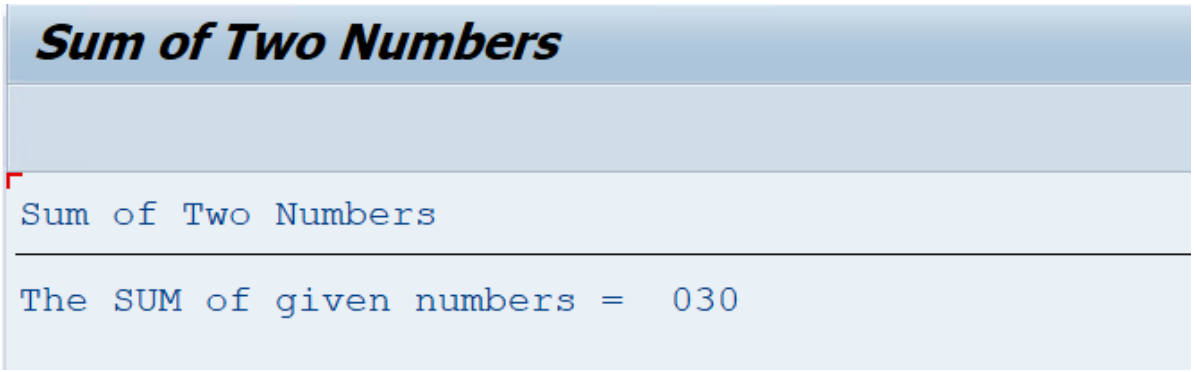
```
*&Processing Logic
lv_output = lv_input1 +  lv_input2.
```

- Displaying the output (We use write statement to display the output).

```
*&Displaying Output.
WRITE :/ 'The SUM of given numbers = ',
       lv_output.
```



- Execute the program.



4. Write Statement

- The basic ABAP statement for displaying data on the screen is WRITE statement.

- We can use ‘/’ in write statement to denotes a new line.

---

5. Chain Operator

- The chain operator is ‘:’ { colon }.
- Chain Operator is used to combine the statements.

Before Using Chain Operator

WRITE var1.

WRITE var2.

WRITE var3.

After Using Chain Operator

WRITE : var1, var2, var3.

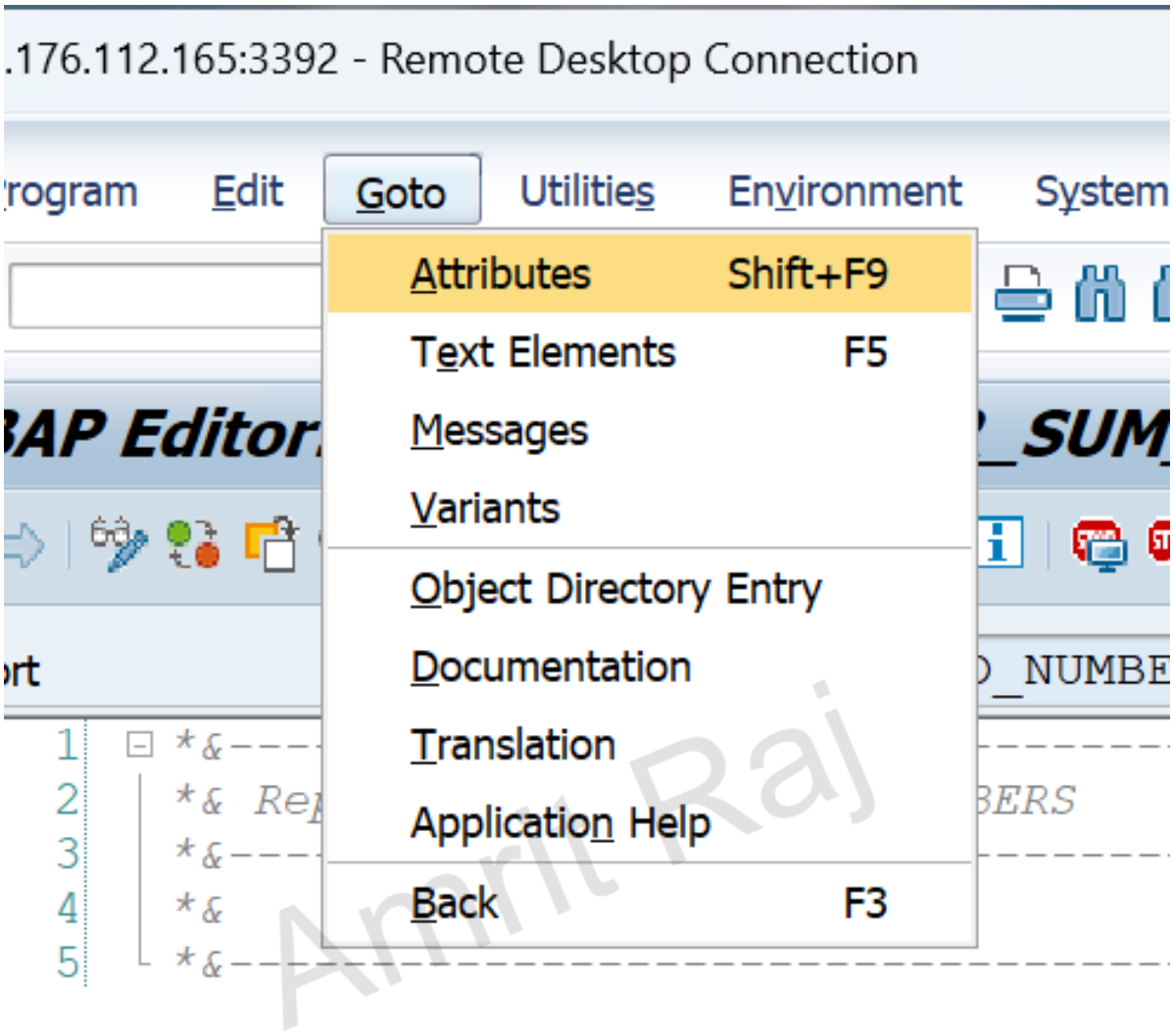
```
8
9    *&Creating Input variable.
10   DATA : lv_input1(2) type n, "First Input
11          lv_input2(2) type n. "Second Input
12
```


- For example, in the above image we have used chain operator to combine multiple inputs.

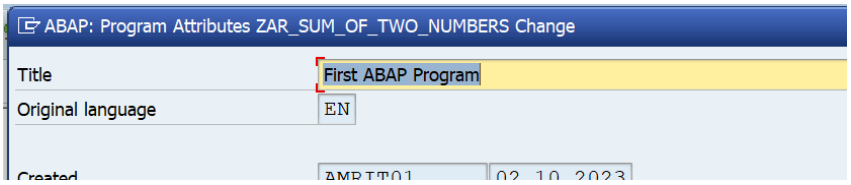
---

6. How to Change the title of the Program ?

- To change the title of the program, click on GOTO → Attributes button.



- now provide the new title



Conditional Statements, Loops, Loops Statements, System Variables

1. Conditional Statements

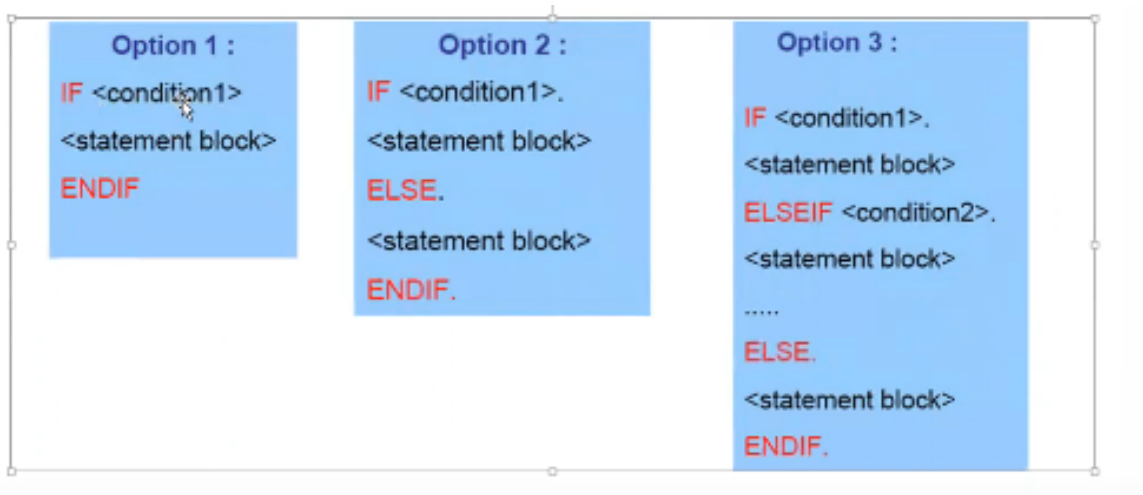
- Conditional Statements allows us to execute a block of code if a certain condition is met.

Types of Conditional Statement :-

1. If Statement
2. CASE Statement

1. IF Statement

- If is a conditional statement.
- We have three versions of IF statements in ABAP.



- Multiple statements blocks are there and depending upon the condition one block executes.
- We provide multiple conditions using elseif.
- If none of the if and else if conditions are satisfied, it goes to else part.

If Else Implementation :-

```
*Declaring input variables
DATA : lv_input(2) TYPE n VALUE 30.


*&Checking If Conditions
IF lv_input = 10.
  WRITE : 'The output is 10'.
ELSEIF lv_input = 20.
  WRITE : 'The output is 20'.
ELSEIF lv_input = 30.
  WRITE : 'The output is 30'.
ELSE.
  WRITE : 'Wrong Answer'.
ENDIF.
```

2. Case Statement

- Case is a conditional statement.

- Multiple statements blocks are there, Depending upon the condition one block executes.
- If none of the conditions are satisfied, it goes to others part.

CASE Statement

```
*Declaring input variables
DATA : lv_input(2) TYPE n VALUE 30.


*Case Statements
CASE lv_input.
  WHEN 10.
    WRITE : 'The output is 10'.
  WHEN 20.
    WRITE : 'The output is 20'.
  WHEN 30.
    WRITE : 'The output is 30'.
  WHEN OTHERS.
    WRITE : 'Wrong Answer'.
ENDCASE.
```


Difference between CASE and IF statement

- IF we have multiple IF conditions, IF checks for all the conditions, until we get a true condition whereas in CASE statement , it directly goes to the true condition.
- The performance of CASE statement is more as compared to IF statement.

---

2. Loop

- Loop allows us to execute a group of statements multiple times.

Types of Loop

1. Do Loop
2. While Loop
3. Loop at <ITAB> where ITAB stands for internal table.

1. Do Loop

- Do Loop is called as unconditional loop i.e. we are not required to pass any condition to it.
- Every Do Loop ends with ENDDO.
- Syntax :

  DO <n> times.

  <statement block>

  ENDDO.

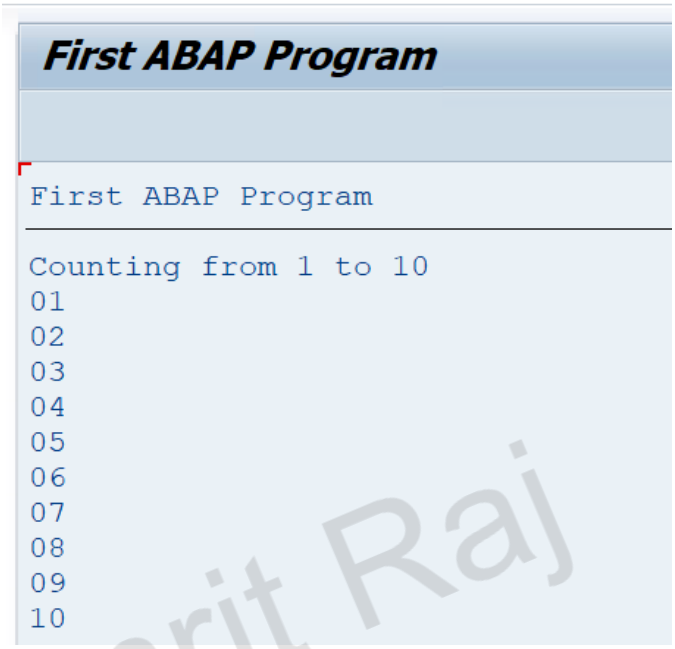
Practical Implementation of Do Loop

- We want to display the data from 1 to 10.

```
**Declaring input variables
DATA : lv_input(2) TYPE n VALUE 1.

WRITE : 'Counting from 1 to 10'.
Do 10 TIMES.
  WRITE :/ lv_input.
```

```
  lv_input = lv_input + 1.
ENDDO.
```

- Execute the Program.



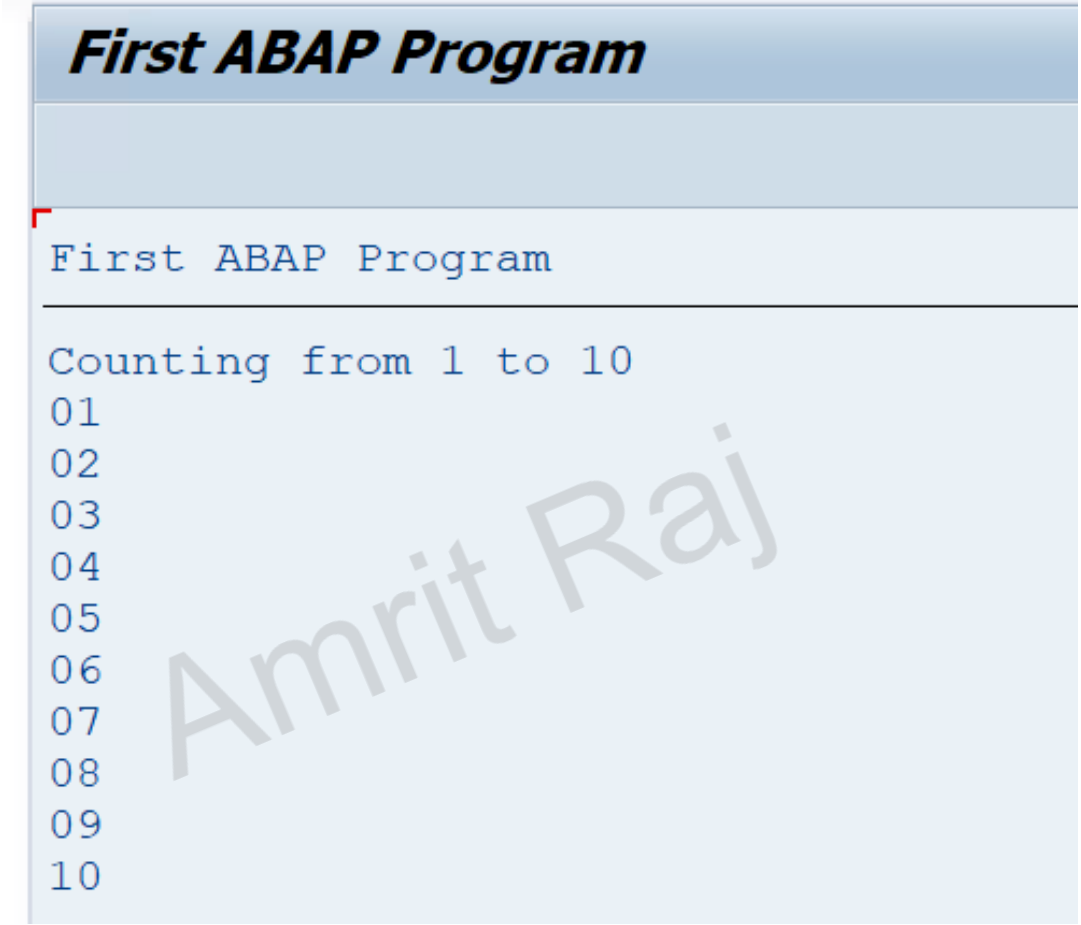
What if we do not specify the number of times, our do loop will execute ?

- If we do not specify the number of times, then it will become a endless loop.
- To overcome this problem, we must specify a condition where loop gets terminated.

```
**Declaring input variables
DATA : lv_input(2) TYPE n VALUE 1.

WRITE : 'Counting from 1 to 10'.
Do .
  WRITE :/ lv_input.
  lv_input = lv_input + 1.
  if lv_input = 10.
      exit.
```

```
    endif.
ENDDO.
```

Output



2. While Loop

- While loop is called as a conditional loop.
- Every while loop ends with ENDWHILE.
- Syntax :-

  WHILE <CONDITION>.

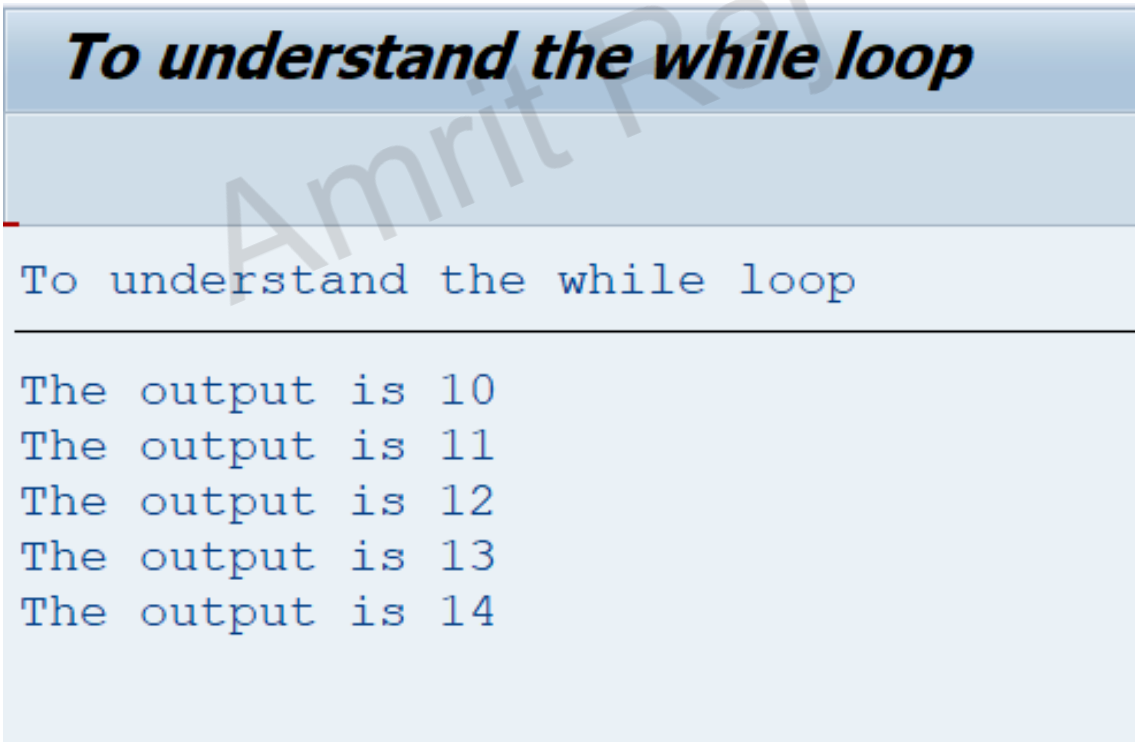
  <STATEMENT BLOCK>

  ENDWHILE.

Practical Implementation of While Loop

```
*******************************
*Implementing While loop
DATA : lv_input(2) TYPE n VALUE 10.

WHILE lv_input < 15. "You can use LT in place of <
  WRITE :/ 'The output is' , lv_input.
  lv_input = lv_input + 1.
ENDWHILE.

*********************************
```

Output



3. Loop Statements :-

- Exit :- Exit is used to exit from the loop.

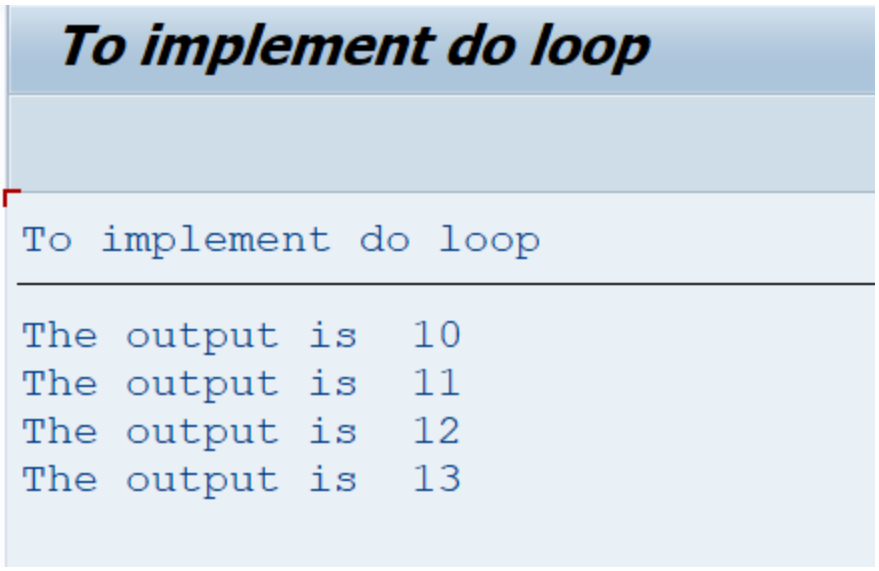
- Continue :- Continue is used to skip the current processing of the record and then process the next record in the loop statements.
- Check :- If the Check condition is not true, loop will skip the current loop pass and move to next loop pass.

---

4. Continue Statement :-

- Whenever continue statement will trigger, it will skip the current iteration and move to next iteration. let’s see the practical implementation.

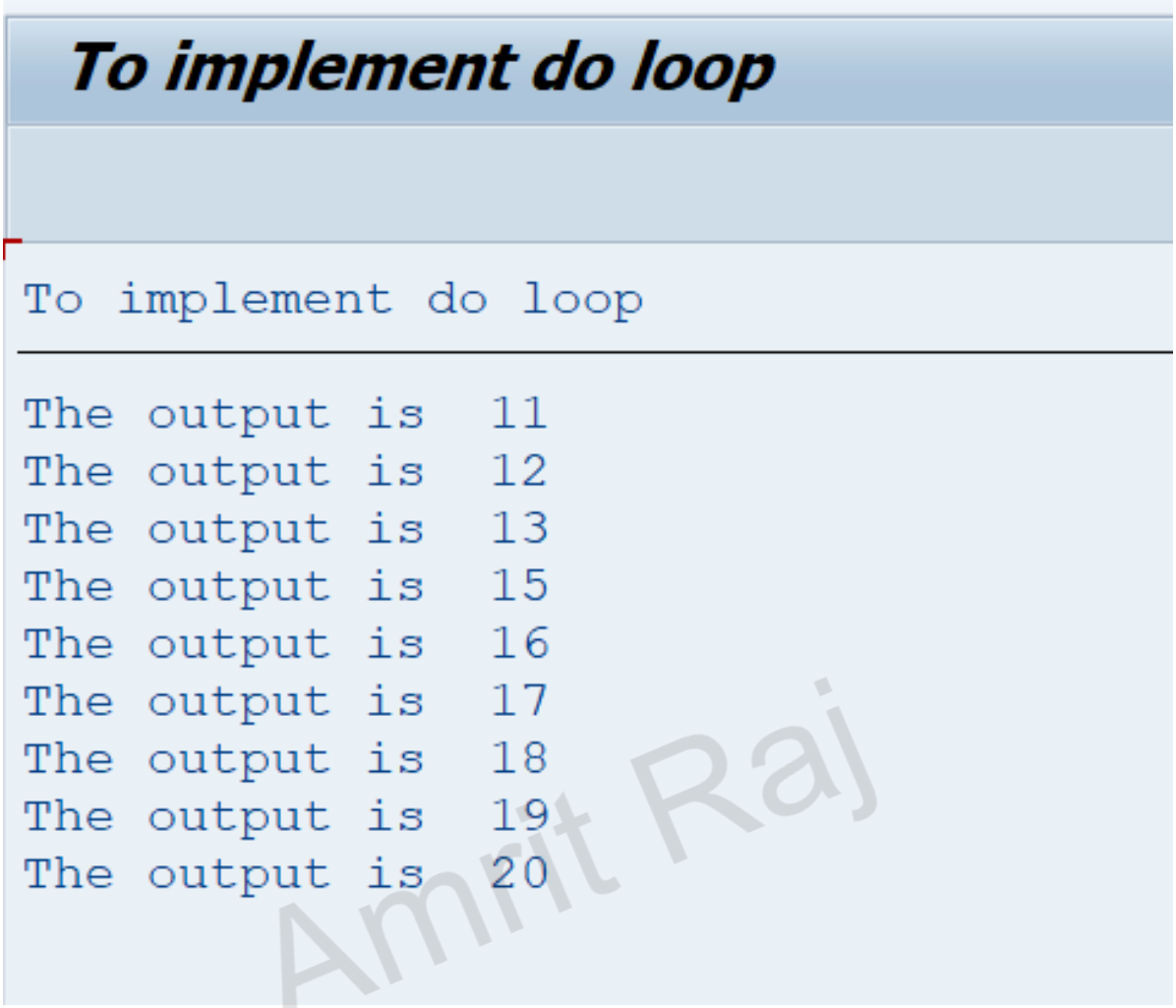
```
************************************
**Using Continue statement in Do loop.
DATA : lv_input(2) TYPE n VALUE 10.

DO 10 TIMES.
  IF lv_input = 14.
    CONTINUE.
  ENDIF.
  WRITE :/ 'The output is ', lv_input.
  lv_input = lv_input + 1.
ENDDO.


************************************
```



Note :- Our loop is executed 10 times, but as soon as it's value becomes 14, since the lv_input is not getting increased, we are getting output only till 13.

Now, see the below example.

```
************************************
**Using Continue statement in Do loop.
DATA : lv_input(2) TYPE n VALUE 10.

DO 10 TIMES.
  lv_input = lv_input + 1.
  IF lv_input = 14.
    CONTINUE.
  ENDIF.
  WRITE :/ 'The output is ', lv_input.
ENDDO.


************************************
```



- In the second example, we initialize our local value with 10, and we are increasing its value right in the beginning of the loop and then we are checking for the if condition and then we are using the write statement.
- So, when the value was 14 it skips the current iteration and rest of the value was printed.

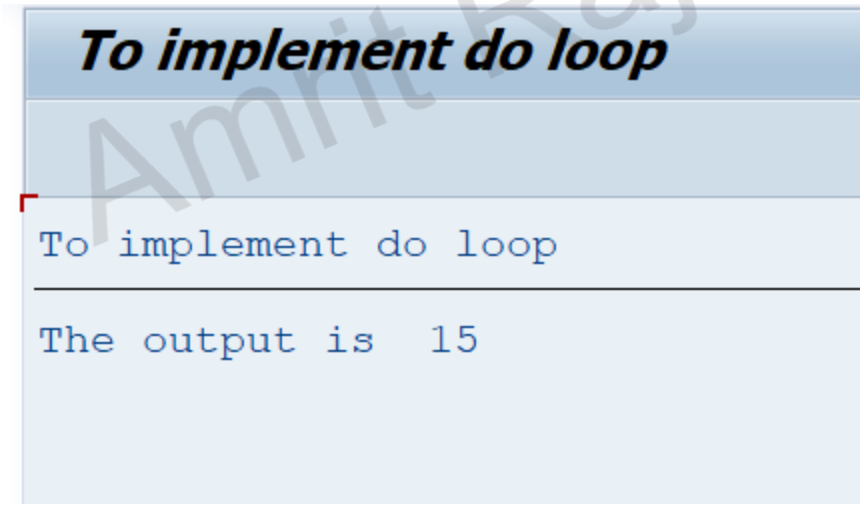
---

5. Check Statement :-

- If the check condition is not true, loop will skip the current loop pass and move to next loop pass.

```
**************************************
***Using Check Statement in Do loop.
DATA : lv_input(2) TYPE n VALUE 10.

DO 10 TIMES.
  lv_input = lv_input + 1.
  CHECK lv_input = 15.
  WRITE :/ 'The output is ', lv_input.
ENDDO.


**************************************
```

- In this program, It will check for each iteration, if value is 15 then and only then, it allows to execute the below statements, otherwise it will skip the current iteration.



  - Check statements are quite similar to if statements which we have discussed earlier.

---

6. System Variables :-

- System Variables are pre-defined variables in SAP.
- SYST is the structure for the system fields.

- All system fields are addressed using SY field name.
- The various system variables are as follows :-
  - SY-SUBRC :- System variables for return code (successful = 0, not successful = other than 0).
  - SY-TABIX :- It returns the current line index inside a loop at internal table.
  - SY-INDEX :- It returns the current line index inside do and while loop.
  - SY-DATUM :- System variables for current date(internal format - yyyymmdd).
  - SY-UNAME :- It returns the logon name of the user.
  - SY-UZEIT :- It returns the current system time(internal format - hhmmss).
  - SY-UCOMM :- System variable for user command.

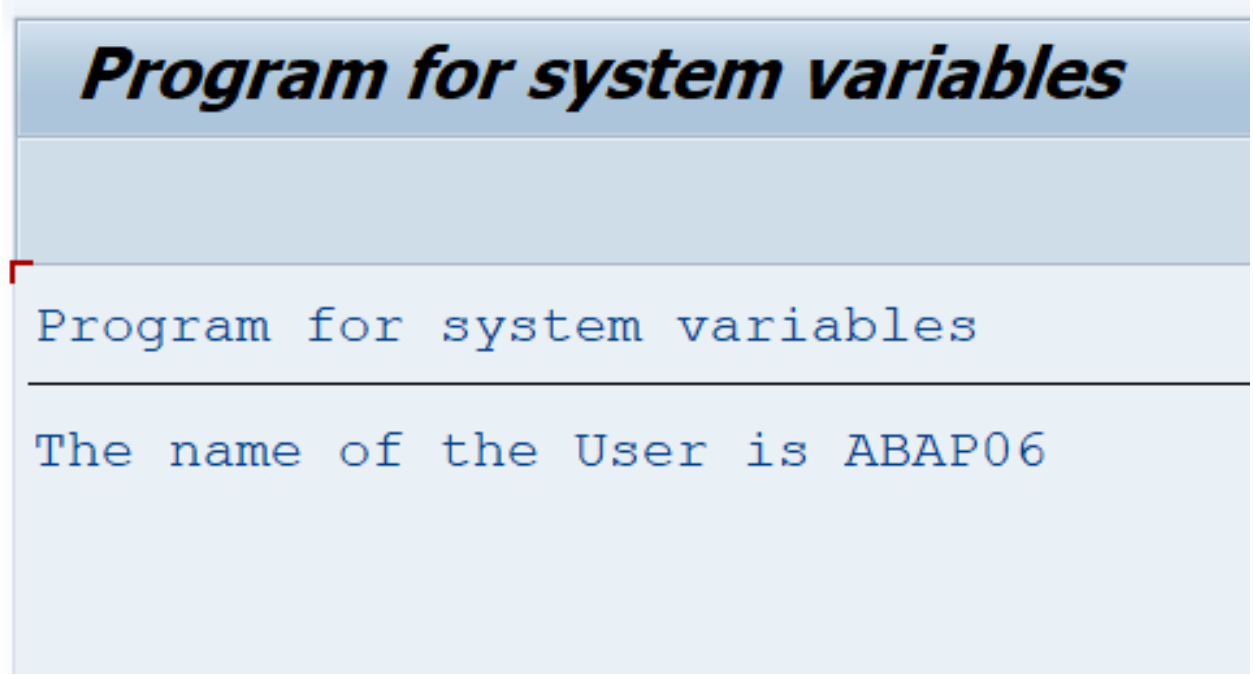
---

let’s see the practical implementation of the above system variables.

For that create a executable program in ABAP Editor.

SY-UNAME :-

```
*****************************************
*Using SY_UNAME system variable
WRITE : 'The name of the User is' , sy-uname.
*****************************************
```



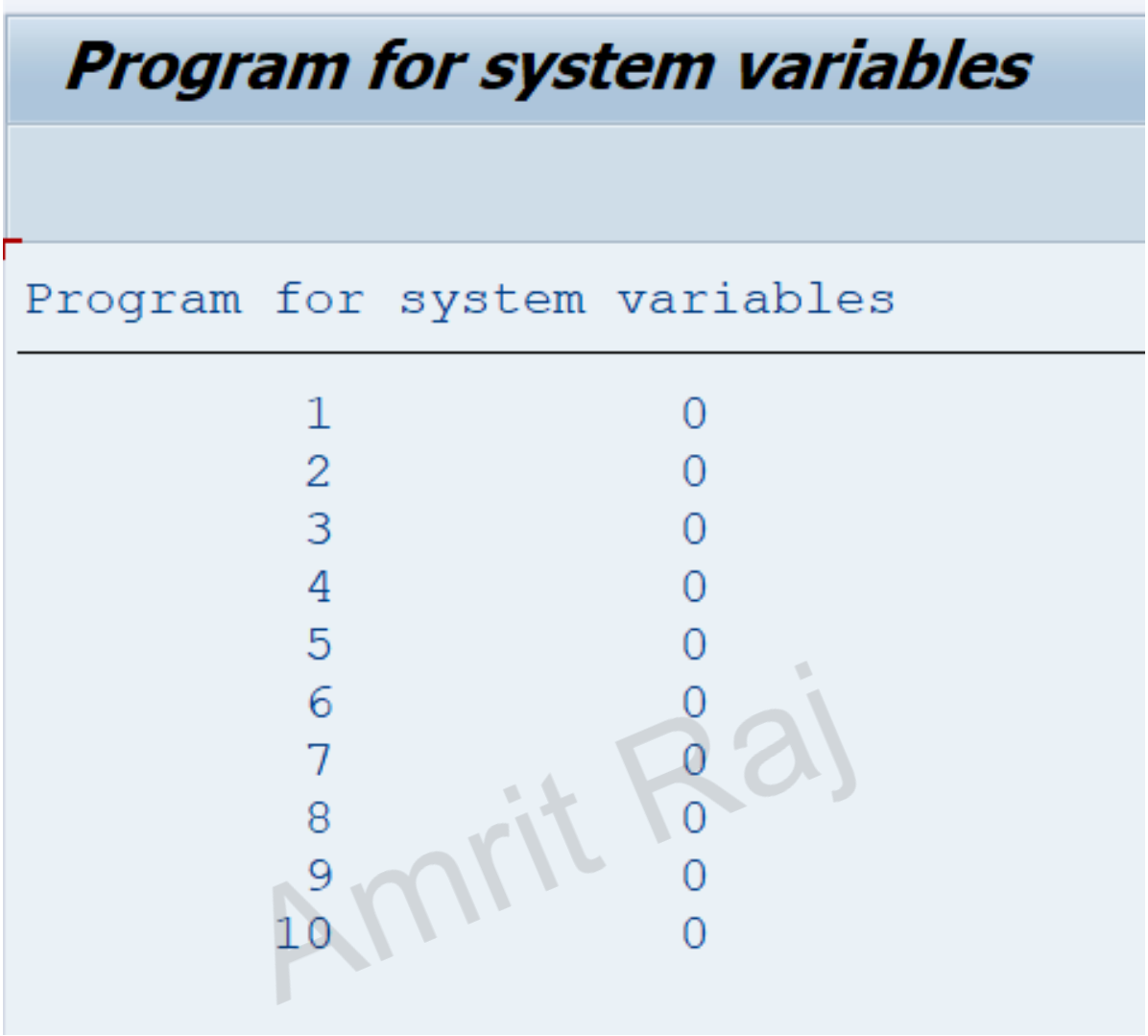
SY-INDEX

SY-INDEX :- We will use this to get the current index inside do and while loop.

Note :- Never ever use the SY-TABIX inside, the do and while loop to get the current index inside do and while loop, it will never return the current index. It is only used in loop at internal table.

```
*****************************************
*SY-INDEX
DO 10 TIMES.
  WRITE : / sy-index, sy-tabix.
ENDDO.
*****************************************
```

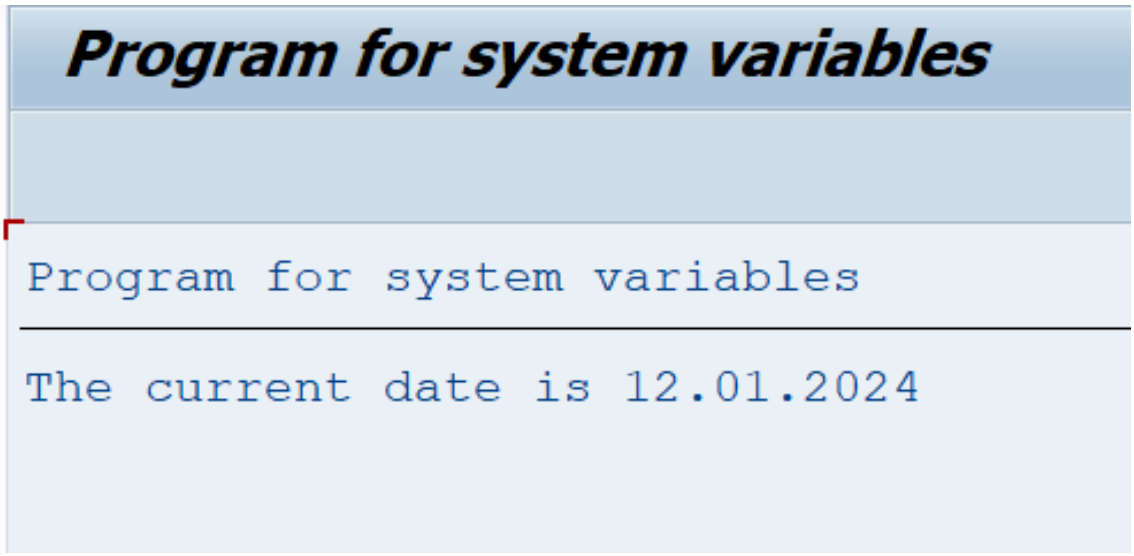
- In the above program we have used the SY-INDEX and SY-TABIX both, but in the output you can clearly see SY-TABIX does not work in do and while loop.



SY-DATUM :-

- Returns the current date.

```
*****************************************
*SY-DATUM
WRITE : 'The current date is', SY-DATUM.
*****************************************
```



Note :- SAP always stores date in YYYYMMDD format and length is 8.

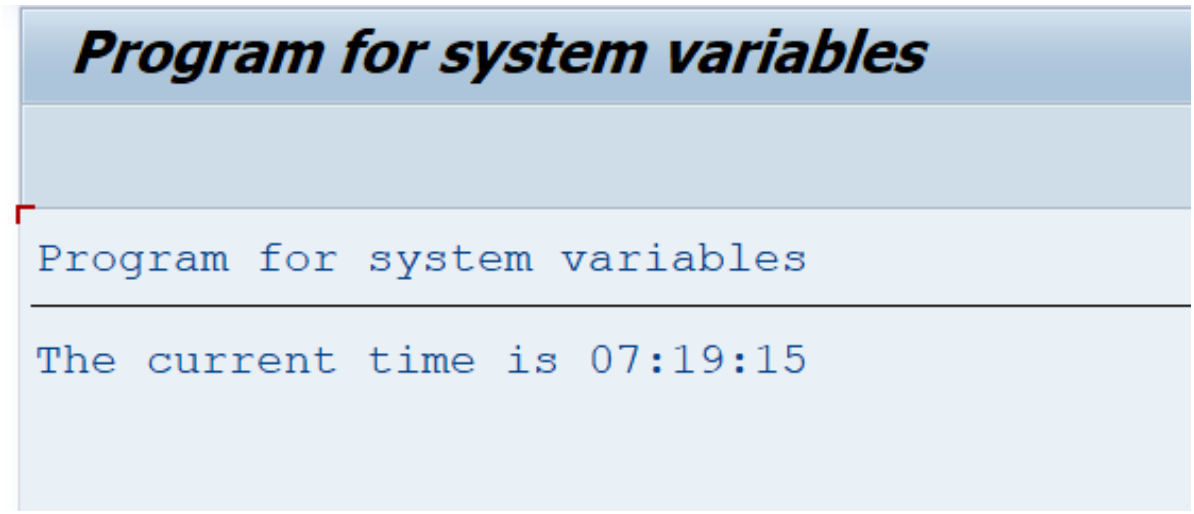
Why does the above output is in DD.MM.YYYY format of length 10 ?

- Since. Date format changes from country to country, So SAP only stores date in YYYYMMDD format and depending upon the user profiles the date is shown to the user.

SY-UZEIT :-

- It returns the current time.

```
*****************************************
*SY-UZEIT
WRITE : 'The current time is', SY-UZEIT.
*****************************************
```



SY-SUBRC :-

- It is one of the most important system variables and base variable of ABAP programming.
- RC stands for return code.
- It returned value is 0, the statement have executed successfully or else the statement has not executed successfully.

String Operations

- A String is a elementary data type of variable length.
- A String is a collection of characters.

1. Concatenate

- Concatenate operation is used to combine multiple strings into one.
- Syntax :

  CONCATENATE <C1> <C2> <C3> …. <CN> into <C> separated by <S>

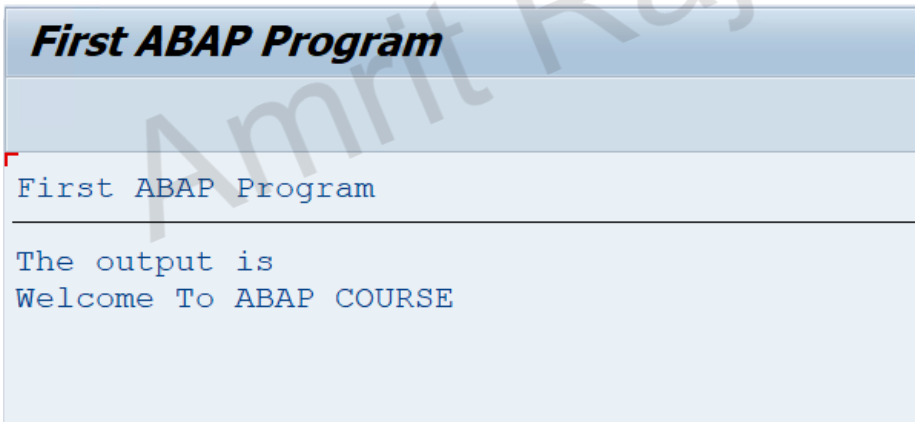
Concatenate Implementation

```
**********************************************************************
*Start of Program
*Declaring variables
DATA : lv_input1(10) type c value 'Welcome',
       lv_input2(10) type c value 'To ABAP',
       lv_input3(10) type c value 'COURSE',
```

```
       lv_output type string. "No need to declare length for
************************************
*Concatenate the string.
CONCATENATE lv_input1 lv_input2 lv_input3
into lv_output SEPARATED BY space.

************************************
*Displaying the output.
Write : 'The output is',
      / lv_output.
*End of Program
************************************************************
```

Note :- separated by used to give a space between variables during concatenation.

Output :



2. SPLIT

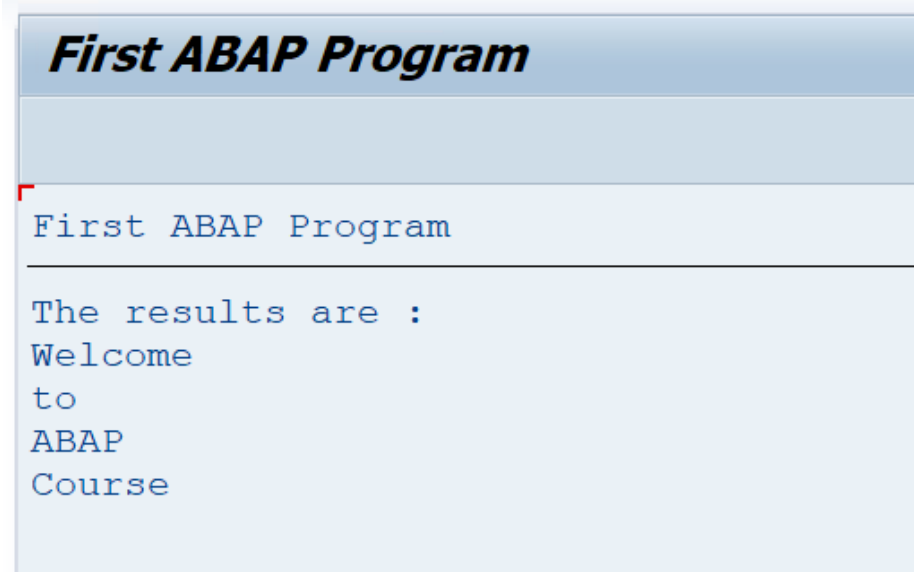
- The purpose of SPLIT is to divide the strings.
- For SPLIT, separator is compulsory.
- Syntax : SPLIT <string> at <separator> into <f1> <f2> <f3>…
  - In the above syntax : split = keyword, <string> = string which we need to split, at = keyword, <separator> = any delimiter, into = keyword, <f1><f2><f3>————- =

individual strings.

SPLIT Implementation

```
DATA : lv_string TYPE string VALUE 'Welcome to ABAP Course'.

DATA : lv_result1 TYPE string,
       lv_result2 TYPE string,
       lv_result3 TYPE string,
       lv_result4 TYPE string.


*&Performing Split
SPLIT lv_string AT space INTO lv_result1 lv_result2 lv_result


WRITE : 'The results are :',
      / lv_result1,
      / lv_result2,
      / lv_result3,
      / lv_result4.
```


Output



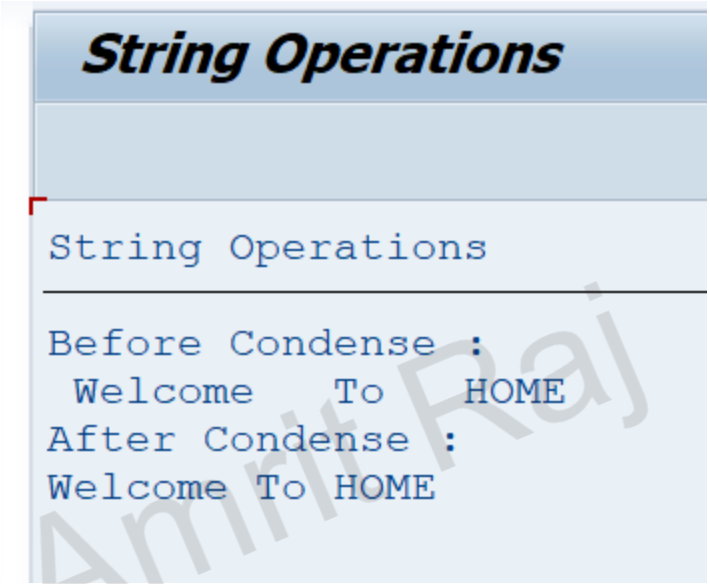
3. CONDENSE

- The purpose of condense is to remove the leading and trailing spaces and convert a sequence of spaces into a single space.
- Syntax : condense<c>.
- In the above syntax : condense = keyword, <c> = string which we want to condense.

```
*********************************************************
*Start of program
*Define a variable with alot of leading and trailing spaces
DATA : lv_input type string value ' Welcome   To   HOME  '.


*Before Aplying Condense.
WRITE : 'Before Condense : ',
      / lv_input.

*Apply Condense.
CONDENSE lv_input.

*After Apply condense
```

```
Write :/ 'After Condense : ',
      / lv_input.

*End of Program.
*********************************************************
```

Output :-



4. CONDENSE NO-GAPS

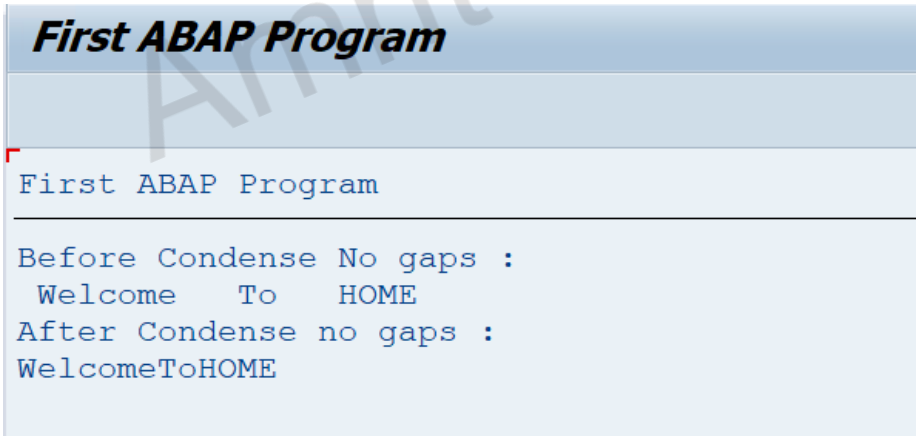
- To remove the entire spaces the addition no-gaps is used with condense.
- Syntax : CONDENSE <C> no-gaps.
- In the above syntax : condense = keyword, <C> = string which we want to condense, no-gaps = keyword.

Implementation

```
*********************************************************
*Start of program
*Define a variable with alot of leading and trailing spaces
DATA : lv_input type string value ' Welcome   To   HOME  '.
```

```
*Before Aplying Condense.
WRITE : 'Before Condense No gaps : ',
      / lv_input.

*Apply Condense.
CONDENSE lv_input NO-GAPS.

*After Apply condense
Write :/ 'After Condense no gaps : ',
      / lv_input.

*End of Program.
*****************************************************************
```

Output



5. STRLEN

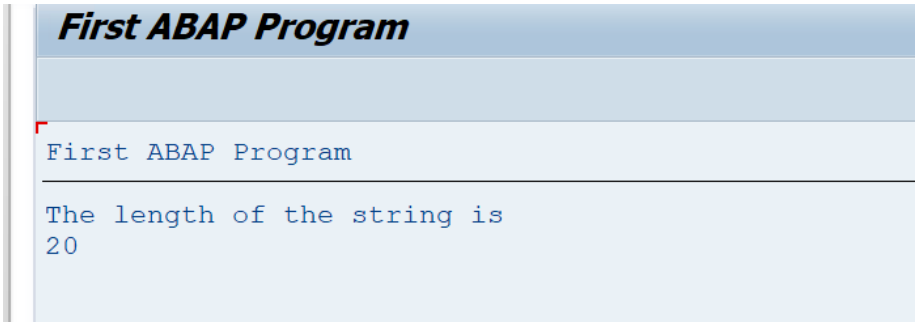
- The purpose of STRLEN is to provide the string length.
- Syntax : LV_LENGTH = STRLEN(string).
  - In the above syntax : LV_LENGTH = variable name which returns the length of the string, STRLEN = pre-defined operation, string = string whose length needs to be

calculated.

Implementation

```
*********************************************************************
*Start of Program
*Defining the variable
DATA : lv_input type string value ' Welcome   To   HOME  '.
DATA : lv_length(2) type n.

*Using strlen
lv_length = STRLEN( lv_input ).

*Displaying the length
WRITE : 'The length of the string is',
      / lv_length.

*End of Program
*********************************************************************
```


Output



6. FIND

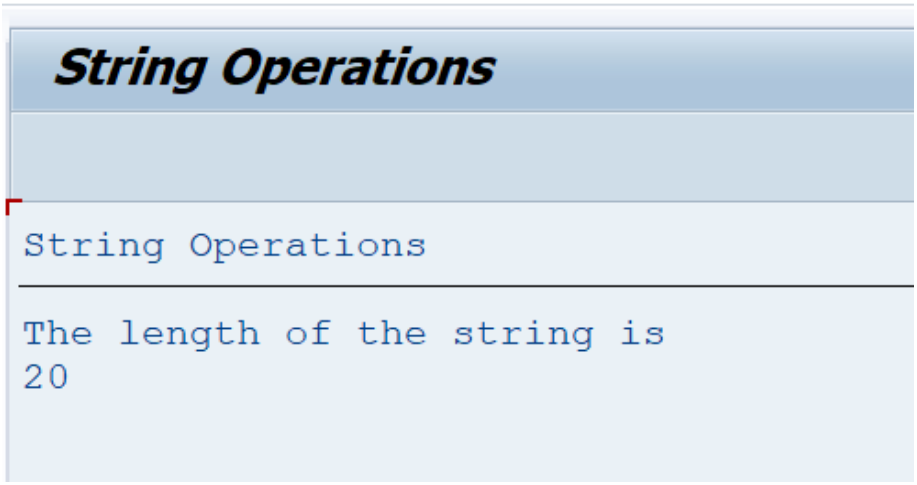
- The purpose of find is to find a particular pattern in a string.
- It is case sensitive as well. (So, to solve this issue, either convert your string lowercase or uppercase).

- Syntax : FIND <pattern> IN <str>.
  - In the above syntax : FIND = keyword, <pattern> = is the sequence of characters we are looking for, IN = keyword, <str> = is the string that is being searched.

Implementation

```
************************************************************
*Start of Program
DATA lv_input(50) type c value 'System Application Product'.

FIND 'System' IN lv_input.
IF sy-subrc eq 0.
  WRITE : 'Successful', sy-subrc.
ELSE.
  WRITE : 'Unsuccessful', sy-subrc.
ENDIF.

*End of Program
************************************************************
```


Output :-



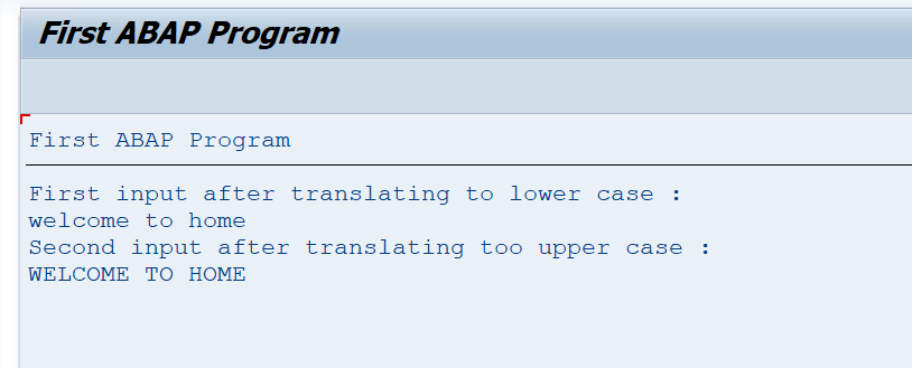
7. TRANSLATE

- The purpose of translate is to convert the string to upper case or lower case.
- Syntax : TRANSLATE <STRING> to upper case/lower case.
    - In the above syntax : TRANSLATE = keyword, <string> = the string which needs to be converted, TO = keyword, UPPER CASE / LOWER CASE = keyword.

Implementation

```
************************************************************
*Start of Program
DATA : lv_input(50) type C value 'Welcome to Home',
       lv_input1(50) type c value 'welcome to home'.

translate lv_input to LOWER CASE.
"Display the first input.
write : 'First input after translating to lower case : ',
      / lv_input.

TRANSLATE lv_input1 TO UPPER CASE.
*Displaying the second input.
write :/ 'Second input after translating too upper case : ',
       / lv_input1.

*End of Program
************************************************************
```


Output



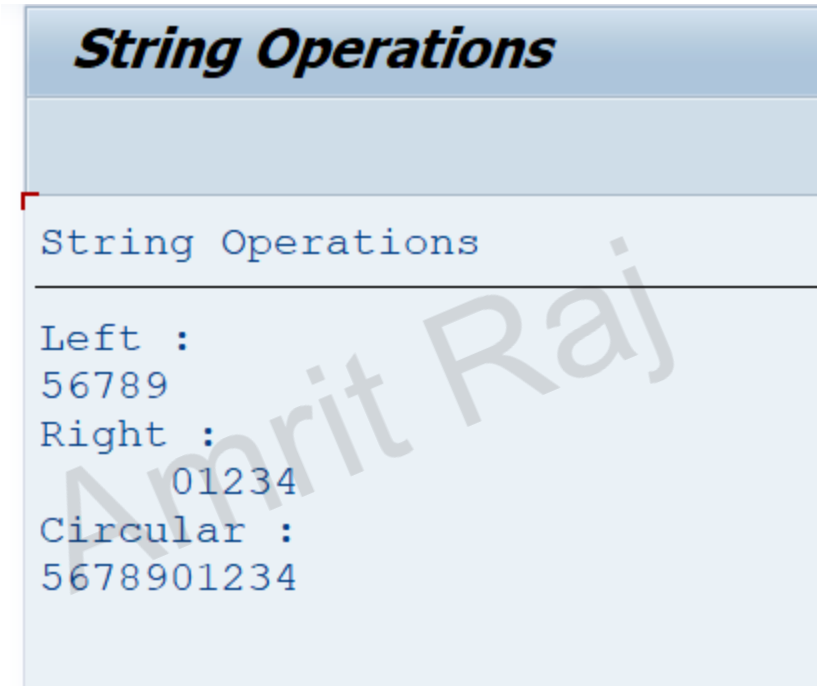
SHIFT

- The purpose of shift is to shift the contents of a string.
- It shifts the string by a number of places.
- Syntax :- shift string by n places <mode>. {by default mode if left}.
- In the above syntax : shift = keyword, string = string which needs to be shifted, by = keyword, n = number, places = keyword, <mode> = left/right/circular.

```
**********************************************************
*Start of Program
DATA : lv_input1(10) TYPE c VALUE '0123456789',
       lv_input2(10) TYPE c VALUE '0123456789',
       lv_input3(10) TYPE c VALUE '0123456789'.

"Applying shift to left.
SHIFT lv_input1 BY 5 PLACES LEFT. " by default left
WRITE : 'Left : ',
      / lv_input1.

*Applying shift to right.
SHIFT lv_input2 BY 5 PLACES RIGHT.
WRITE :/ 'Right :',
       / lv_input2.

*Applying shift to circular
```

```
SHIFT lv_input3 BY 5 PLACES CIRCULAR.
WRITE :/ 'Circular : ',
       / lv_input3.

*End of Program
*****************************************************************
```

Output :-



- Since length is 10, in first case first five elements gets removed and in second case last five elements were removed because of shift operation.

Substring Processing

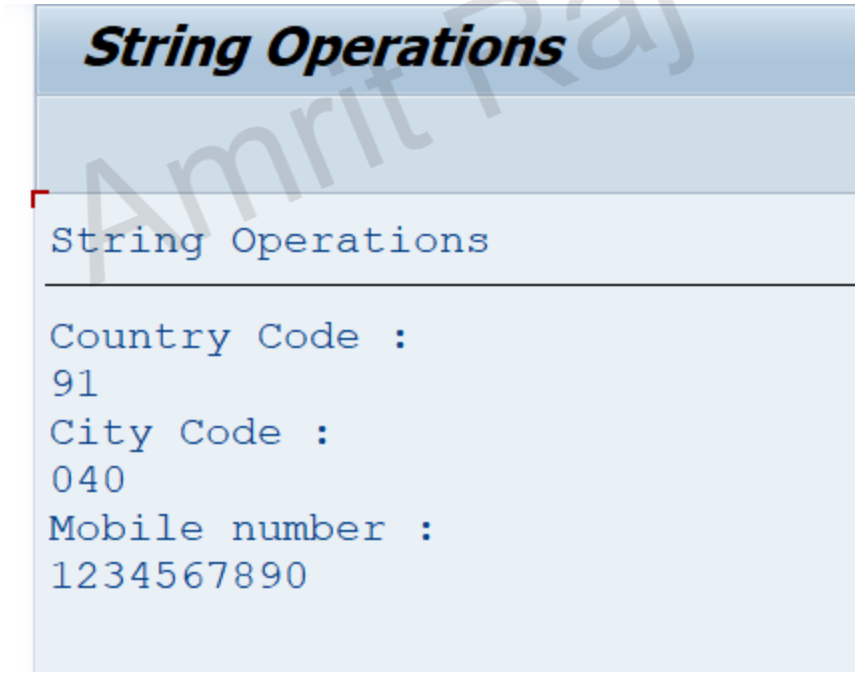
- Substring is a part of the string or small set of characters from the string.
- Depends upon the requirement we need to process the substring.

- Syntax :- Target_variable = Source_variable [+] [starting position of substring](length of the substring).

```
**********************************************************
*Start of Program
DATA : lv_value(50) type c value '91-040-1234567890'.
DATA : lv_country(2) type c.
DATA : lv_city(3) type c.
DATA : lv_number(10) type c.

*Fetching Country Code
lv_country = lv_value+0(2).
write : 'Country Code : ',
      / lv_country.

*Fetching the city Code
lv_city = lv_value+3(3).
write :/ 'City Code : ',
       / lv_city.

*Mob no
lv_number = lv_value+7(10).
write :/ 'Mobile number : ',
       / lv_number.

*End of Program
**********************************************************
```

```
************************************************************
*Start of Program
DATA : lv_value(50) type c value '91-040-1234567890'.
DATA : lv_country(2) type c.
DATA : lv_city(3) type c.
DATA : lv_number(10) type c.

*Fetching Country Code
lv_country = lv_value+0(2).
write : 'Country Code : ',
      / lv_country.

*Fetching the city Code
lv_city = lv_value+3(3).
write :/ 'City Code : ',
       / lv_city.

*Mob no
lv_number = lv_value+7(10).
write :/ 'Mobile number : ',
       / lv_number.

*End of Program
************************************************************
```

Output :-



Internal Tables, Work Area declaration using local and Global Structure, Table Type

1. Internal Table

- Internal table is a temporary storage of data on ABAP Application layer.
- Internal Table can store any number of data at the run time.
- Suppose, you want to perform a operation on the data stored in the database layer, so what we will do is we will bring data from database layer to Application layer and will perform the required operation and then commit the changes in the database layer.

Note :-

- A very important use of internal tables is for storing and formatting data from a database table within a program.

Mob No :- 6261538504

2. Work Area

- Work Area is also a temporary storage of data on application layer.
- However, unlike Internal Table, A Work Are can only store a single record at a time.
- Work Area is used to process the data in an internal table, one line at a time.

---

3. Internal Table and Work Area Declaration

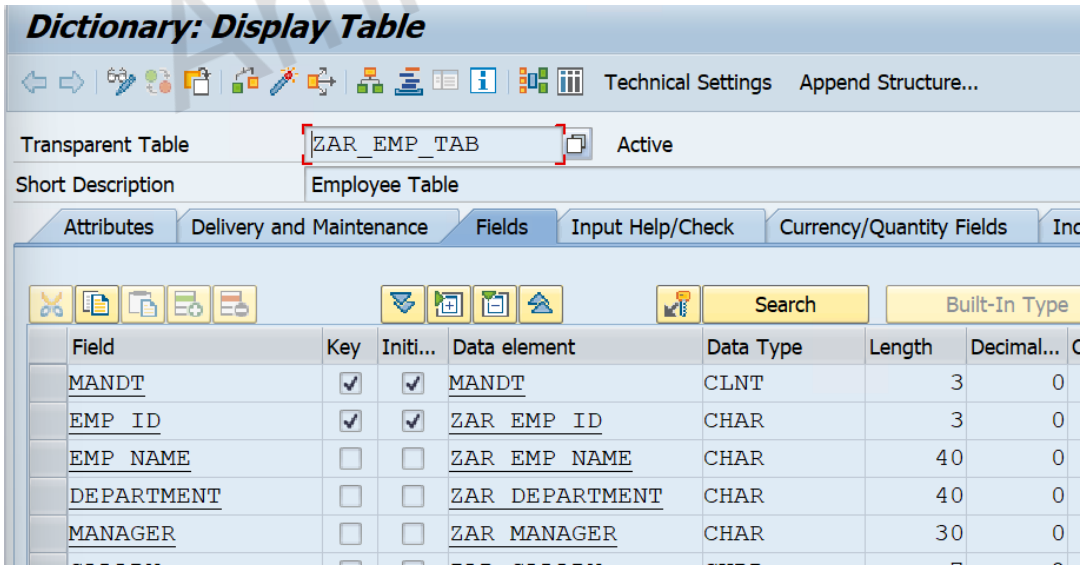
- Before Creation of a internal table, we are require to create a structure.

1. Structure Creation :-

- To create a structure locally, we are require to use the TYPES keyword.
- A Structure is a complex data type which we have discussed in the introduction part.

i) Requirement

- We will create a structure for four columns of our Employee Table.



Implementation

```
TYPES : BEGIN OF ty_employee,
   emp_id TYPE zar_emp_id,
   emp_name TYPE zar_emp_name,
   department TYPE zar_department,
   manager TYPE zar_manager,
  END OF ty_employee.
```

2. Internal Table Declaration

- We use the keyword type table of to declare the internal table.

```
DATA : lt_employee TYPE TABLE OF ty_employee.
```

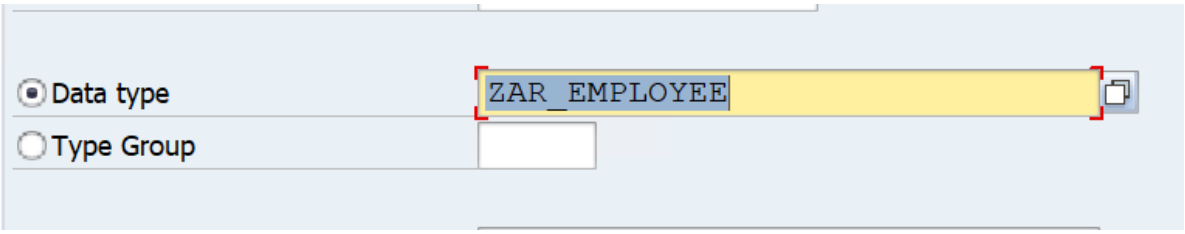
3. Work Area Declaration

- We use the type keyword to declare the work area.

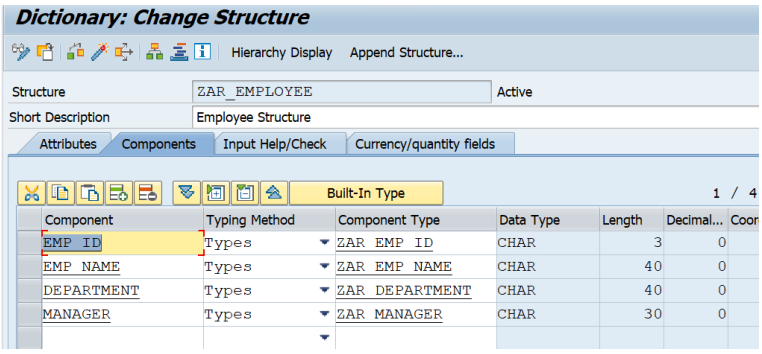
```
DATA : ls_employee TYPE ty_employee.
```


4. Global Structure Creation

- Step 1 :- Go to SE11 transaction code and select the data type radio button and give a name for the structure.



- Step 2 :- Click on create button and select structure radio button and click on Okay button.
- Step 3 :- give the columns and their data types and activate it.

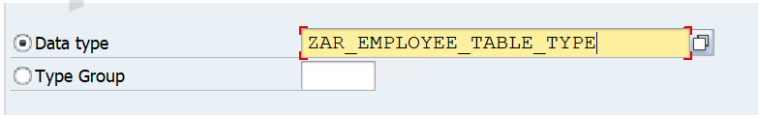


5. Internal Table and Work Area declaration Using Global Structure

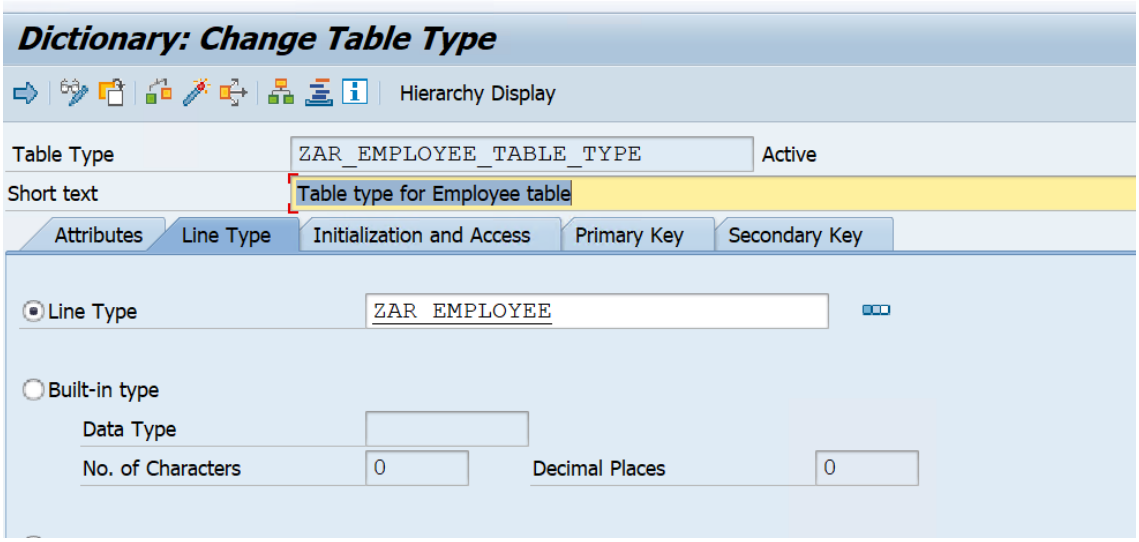
```
DATA : lt_employee type table of ZAR_EMPLOYEE,
       ls_employee type ZAR_EMPLOYEE.
```

6. Table Type Creation

- To create a table type, we will go to SE11 transaction code and select the type radio button and give a name for your table type.

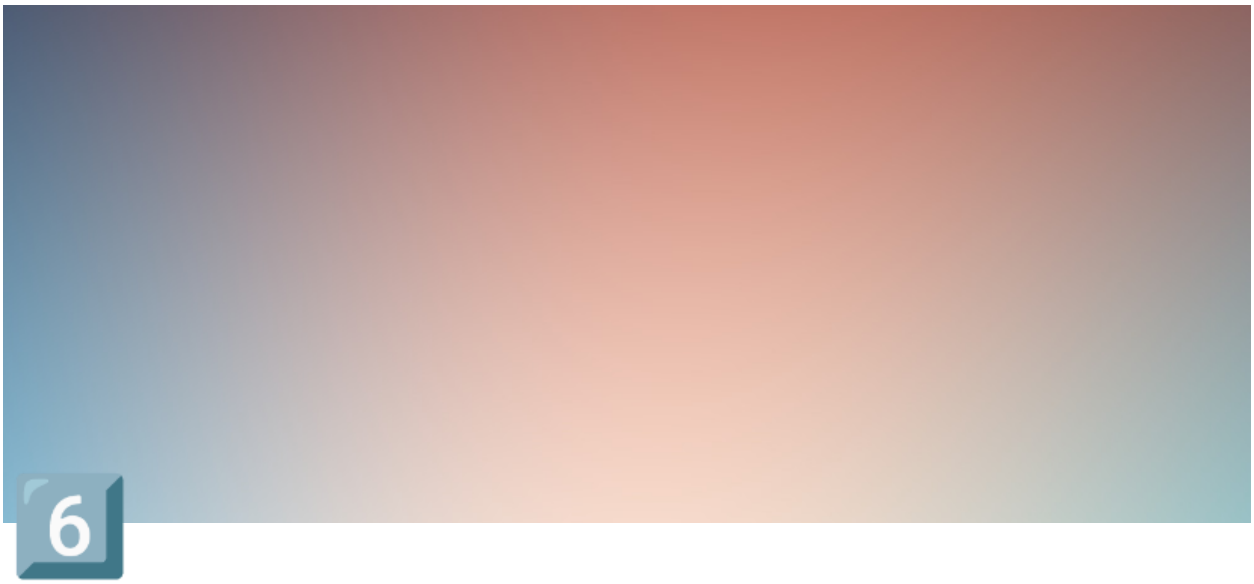


  - Click on create button and select the table type radio button and click on Okay button.
  - In the line type provide the global structure that you have created above.



Declaring internal table using table Type

```
DATA : lt_employee type ZAR_EMPLOYEE_TABLE_TYPE.
```



Internal Table Operations in ABAP

- There are variety of internal table operations using which we can perform various kinds of operations on internal table.
    1. Append Statement :- Append Statement is used to insert data at the last of the internal table.
    2. Delete :- Delete statement is used to delete the records from the internal table.
    3. Modify :- Modify is used to modify the records of the internal table.
    4. Loop :- Loop is used to read the records one by one from the internal table.
    5. Read Table :- Read statement is used to read the first matching record from the internal table.
    6. Clear, Refresh :- It is used to clear the contents of the internal table.
    7. Collect :- Collect statement is used to make sum of amount values based upon unique Character Values.
    8. Sort :- Sort operation is used to sort the internal table.
        - If we are not specifying anything, then by default it sorts in the ascending order. If we want to sort in descending order, then we need to specify the keyword

descending.

9. Describe Table :-
    - Describe Table returns the number of records in the internal table.

      Syntax : DESCRIBE TABLE <ITAB> LINES <LV_LINES>.

---

1. Append Statement

- It is used to insert data at the last of the internal table.
- Values in SAP always passes from right to left.
- It is always very important to clear the work area after transfer of data into internal table.
- Must check how data is getting stored in debugging mode.

Practical Implementation

```
ls_employee-emp_id = '101'.
ls_employee-emp_name = 'Shivam Singh'.
ls_employee-department = 'SAP ABAP'.
ls_employee-manager = 'Amrit Raj'.
APPEND ls_employee TO lt_employee.
CLEAR ls_employee.

ls_employee-emp_id = '102'.
ls_employee-emp_name = 'Abhishek Yadav'.
ls_employee-department = 'SAP ABAP'.
ls_employee-manager = 'Amrit Raj'.
APPEND ls_employee TO lt_employee.
CLEAR ls_employee.

ls_employee-emp_id = '103'.
ls_employee-emp_name = 'Shikhar Srivastav'.
ls_employee-department = 'SAP ABAP'.
ls_employee-manager = 'Amrit Raj'.
```

```
APPEND ls_employee TO lt_employee.
CLEAR ls_employee.
```

---

2. Loop at Internal Table

- Loop is a internal table operation which will read the records from the internal table one by one.
- Loop at <ITAB> where, <ITAB> stands for internal table.
- We will process the records from internal table into work area one by one using Loop at <ITAB>.

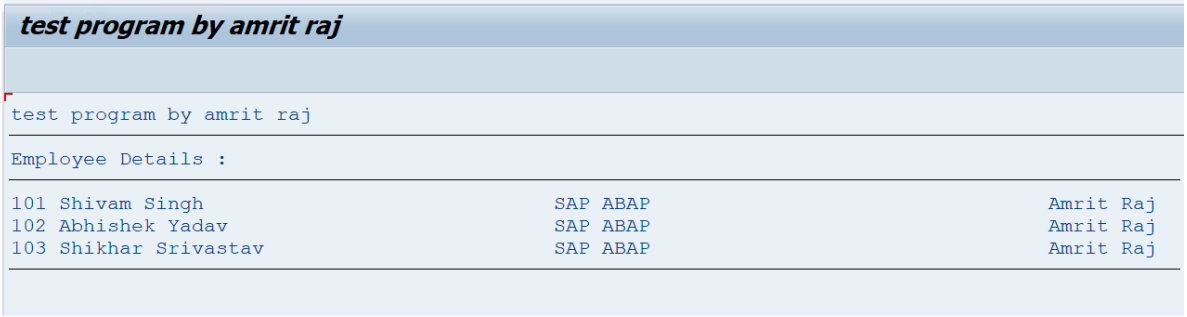
Requirement :-

- We will display the above appended data into the output screen.

Implementation :-

```
WRITE :/ 'Employee Details : '.
WRITE :/ '-------------------------------------------------
LOOP AT lt_employee INTO ls_employee.

  WRITE :/ ls_employee-emp_id,
           ls_employee-emp_name,
           ls_employee-department,
           ls_employee-manager.

ENDLOOP.
WRITE :/ '-------------------------------------------------
```

Output



3. Delete

- Delete operation is used to delete the records from the internal table.
- There are two ways to perform delete operations :-
  1. Delete based on where condition.
  2. Delete based on index.

1. Delete based on where condition

```
DELETE lt_employee WHERE emp_id = '101'.
```

2. Delete based on index

```
DELETE lt_employee INDEX 1.
```

4. Modify

- Modify operation is used to modify the existing records in the internal table.

Note :-

- If a record do not exist it inserts a new record inside the internal table.

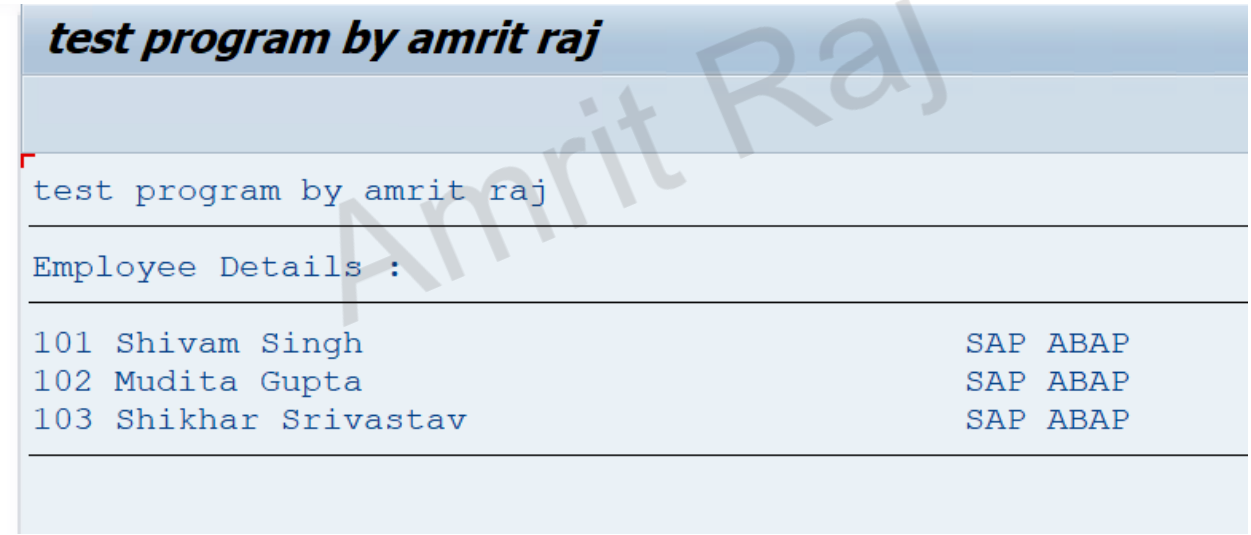
Requirement :-

- We want to change the employee name where Employee Id = ‘102’.

Implementation :-

```
Loop at lt_employee into ls_employee.
  if ls_employee-emp_id = '102'.
    ls_employee-emp_name = 'Mudita Gupta'.
    ls_employee-department = 'Cloud Innovation'.
    modify lt_employee from ls_employee TRANSPORTING emp_name
  endif.
ENDLOOP.
```

Output :-



Note :-

- Only those columns will be changed whose name you will pass into the transporting parameter.

---

5. Read Table

- Read Table is used to read the first matching record from the internal table.

- We should never forget to check the condition SY-SUBRC after the read table.

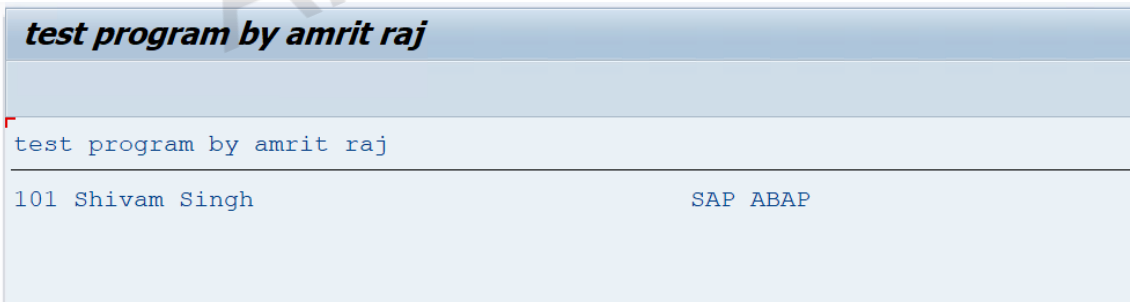
Note :-

- There are two ways to use the Read Table.
  1. Read using key.
  2. Read using index.

1. Read Using Key

```
Read TABLE lt_employee into ls_employee with key emp_id =
if sy-subrc eq 0.
   WRITE :/ ls_employee-emp_id,
            ls_employee-emp_name,
            ls_employee-department,
            ls_employee-manager.
endif.
```


Output

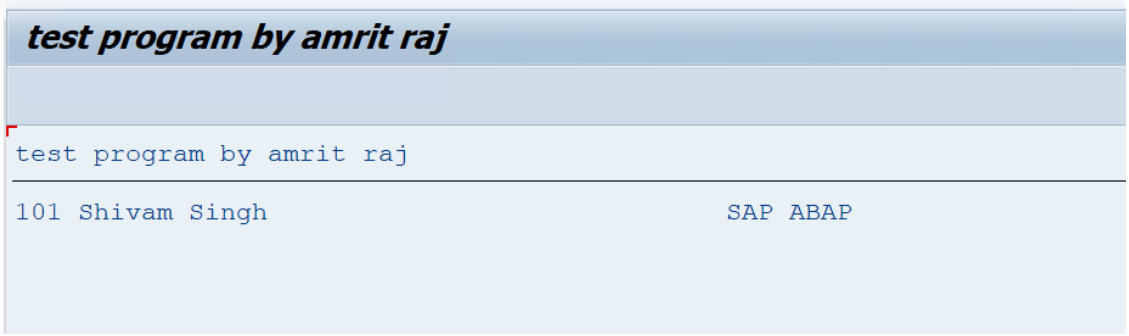


2. Read Using Index

```
Read TABLE lt_employee into ls_employee index 1.
if sy-subrc eq 0.
   WRITE :/ ls_employee-emp_id,
            ls_employee-emp_name,
            ls_employee-department,
```

```
            ls_employee-manager.
endif.
```

Output



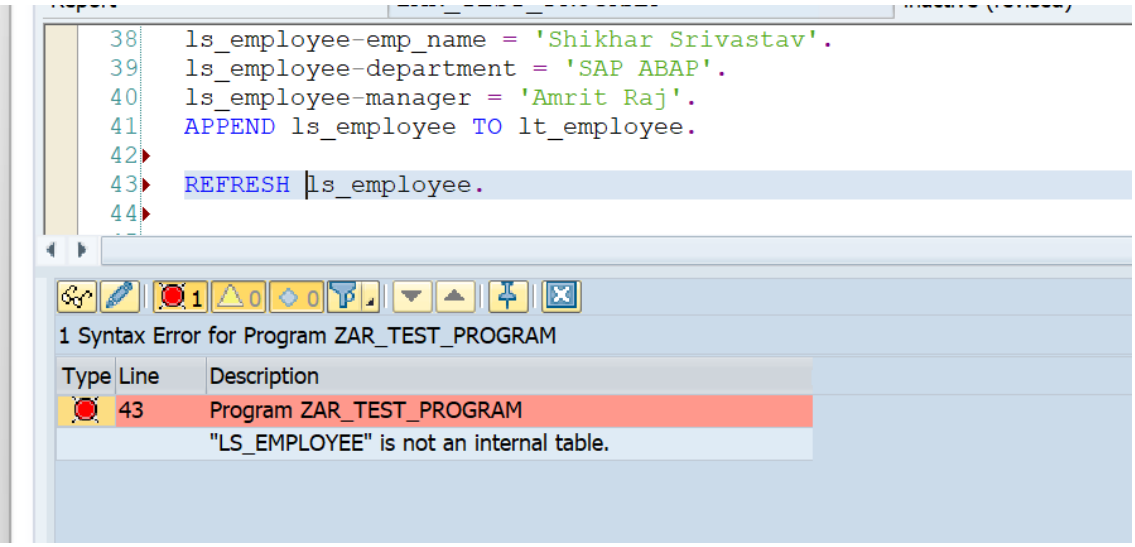
6. Clear, Refresh

- We can clear the records of the internal table using Clear or Refresh statement.

```
CLEAR LT_EMPLOYEE.

REFRESH LT_EMPLOYEE.
```


Note :-

- CLEAR is used to clear the records of work area.
- We cannot use REFRESH to clear the records of work area.



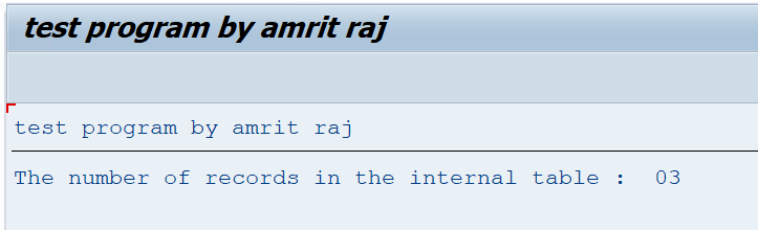
7. Describe

- Describe operation is used to count the number of records stored in the internal table.

Implementation

```
DATA : lv_count(2) TYPE n.
DESCRIBE TABLE lt_employee LINES lv_count.
WRITE :/ 'The number of records in the internal table : ', lv
```

Output



8. Sort

- Sort is used to sort the internal table.

- If we are not specifying anything, then by default it sorts in the ascending order.
- If we want to sort in descending order, then we need to specify the keyword descending.

Implementation

```
*&Sorting in Ascending Order
SORT lt_employee BY emp_id.

*&Sorting in Descending Order
SORT lt_employee BY emp_id DESCENDING.
```

For Multiple Column Sorting

- In case of multiple scenarios, the sorting is sorting + sub sorting.

```
SORT lt_employee BY emp_id ASCENDING emp_name DESCENDING mana
```

Collect Operation in Internal Table

- Collect Operation is used to make sum of amount values based upon unique character(non numeric) field values.

Non Numeric Data Types

- C (Character)
- N(Numeric)
- D(Date)
- T(time)

Numeric Data Types

- I (Integer)
- P (Packed Number)
- F (Floating point)

Requirement

- Suppose, we have got some expanses details of a ABC and ABCD Company for month January and February.

	A	B	C	D	E	F
1	**COMPANY**	**DEPARTMENT**	**Amount**	**Month**		
2	ABC	ABAP	10000	January		
3	ABC	BW	20000	January		
4	ABC	ABAP	50000	February		
5	ABC	MM	10000	February		
6	ABC	BW	20000	February		
7	ABCD	BW	20000	February		
8						

Calculating the Total Expanses :-

- IF we will calculate the total expanses of all the companies.

8			
9		**Total Expanses**	
10	**COMPANY**	**DEPARTMENT**	**Amount**
11	ABC	ABAP	60000
12	ABC	BW	40000
13	ABC	MM	10000
14	ABCD	BW	20000
15			

  - We will get the above results based on unique character values.
  - And you can clearly see how our number of records got reduced because of collect statement.
  - This kind of requirement we handle using collect statement.

Implementing Collect Operation :-

- Step 1 :- Create the type structure for company department and amount followed by the declaration of internal table and work area.

```
TYPES : BEGIN OF ty_data,
          comp_name(4)   TYPE c,
          department(10) TYPE c,
          amount         TYPE zar_salary,
        END OF ty_data.

DATA : lt_data TYPE TABLE OF ty_data,
       ls_data TYPE ty_data.
```

- Step 2 :- Storing data into the internal table.

```
ls_data-comp_name = 'ABC'.
ls_data-department = 'ABAP'.
ls_Data-amount = '10000.00'.
APPEND ls_Data TO lt_data.
CLEAR ls_data.

ls_data-comp_name = 'ABC'.
ls_data-department = 'BW'.
ls_Data-amount = '20000.00'.
APPEND ls_Data TO lt_data.
CLEAR ls_data.

ls_data-comp_name = 'ABC'.
ls_data-department = 'ABAP'.
ls_Data-amount = '50000.00'.
APPEND ls_Data TO lt_data.
CLEAR ls_data.

ls_data-comp_name = 'ABC'.
ls_data-department = 'MM'.
ls_Data-amount = '10000.00'.
```

```
APPEND ls_Data TO lt_data.
CLEAR ls_data.

ls_data-comp_name = 'ABC'.
ls_data-department = 'BW'.
ls_Data-amount = '20000.00'.
APPEND ls_Data TO lt_data.
CLEAR ls_data.

ls_data-comp_name = 'ABCD'.
ls_data-department = 'BW'.
ls_Data-amount = '10000.00'.
APPEND ls_Data TO lt_data.
CLEAR ls_data.
```

- Step 3 :- Create another internal table, and then apply collect statement.

```
DATA  : lt_data1 TYPE TABLE OF ty_data.
LOOP AT lt_data INTO ls_data.
  COLLECT ls_data INTO lt_data1.
  CLEAR ls_data.
ENDLOOP.
```

- Step 4 :- Display the data.

```
loop at lt_data1 into ls_Data.
  WRITE :/ ls_data-comp_name, ls_Data-department, ls_data-amo
ENDLOOP.
```

Code

```
TYPES : BEGIN OF ty_data,
          comp_name(4)   TYPE c,
          department(10) TYPE c,
```

```
          amount         TYPE zar_salary,
        END OF ty_data.

DATA : lt_data TYPE TABLE OF ty_data,
       ls_data TYPE ty_data.
DATA  : lt_data1 TYPE TABLE OF ty_data.


ls_data-comp_name = 'ABC'.
ls_data-department = 'ABAP'.
ls_Data-amount = '10000.00'.
APPEND ls_Data TO lt_data.
CLEAR ls_data.

ls_data-comp_name = 'ABC'.
ls_data-department = 'BW'.
ls_Data-amount = '20000.00'.
APPEND ls_Data TO lt_data.
CLEAR ls_data.

ls_data-comp_name = 'ABC'.
ls_data-department = 'ABAP'.
ls_Data-amount = '50000.00'.
APPEND ls_Data TO lt_data.
CLEAR ls_data.

ls_data-comp_name = 'ABC'.
ls_data-department = 'MM'.
ls_Data-amount = '10000.00'.
APPEND ls_Data TO lt_data.
CLEAR ls_data.

ls_data-comp_name = 'ABC'.
ls_data-department = 'BW'.
ls_Data-amount = '20000.00'.
APPEND ls_Data TO lt_data.
```

```
CLEAR ls_data.

ls_data-comp_name = 'ABCD'.
ls_data-department = 'BW'.
ls_Data-amount = '10000.00'.
APPEND ls_Data TO lt_data.
CLEAR ls_data.


LOOP AT lt_data INTO ls_data.
  COLLECT ls_data INTO lt_data1.
  CLEAR ls_data.
ENDLOOP.

loop at lt_data1 into ls_Data.
  WRITE :/ ls_data-comp_name, ls_Data-department, ls_data-amount
ENDLOOP.
```


Output :-

Collect Statement implementation

Collect Statement implementation

ABC	ABAP	60000,00
ABC	BW	40000,00
ABC	MM	10000,00
ABCD	BW	10000,00

Types of Internal Tables in ABAP Programming

- There are three types of Internal Table in ABAP Programming :-
  1. Standard Internal Table
  2. Sorted Internal Table
  3. Hashed Internal Table

---

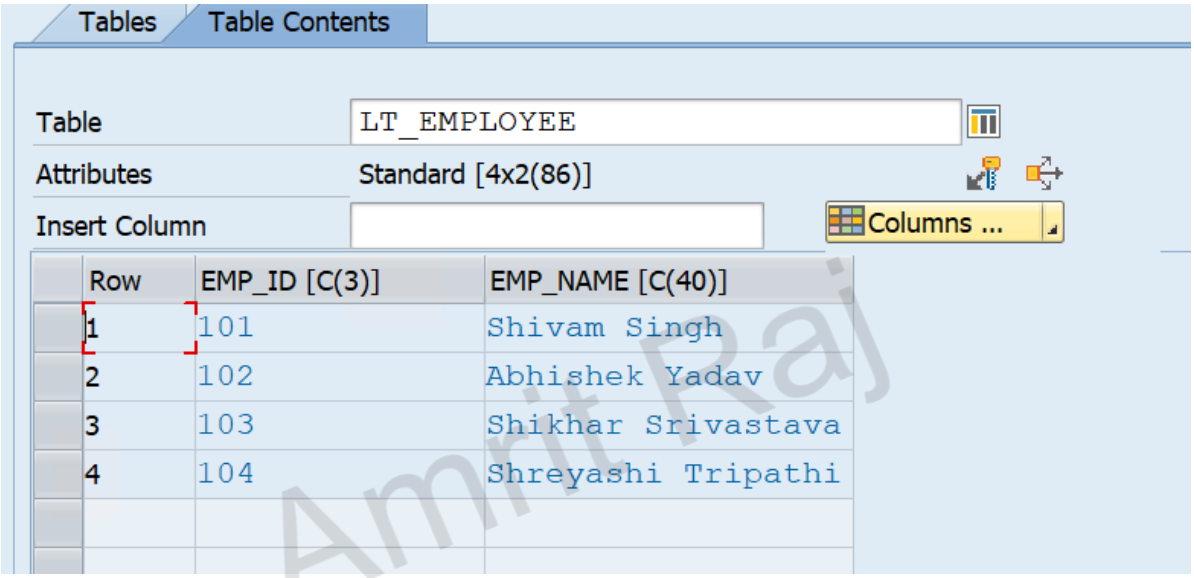
1. Standard Internal Table

1. They are the default internal tables.

```
TYPES : BEGIN OF ty_employee,
          emp_id   TYPE zar_emp_id,
          emp_name TYPE zar_emp_name,
        END OF ty_employee.
```

```
DATA : lt_employee TYPE TABLE OF ty_employee,
       ls_employee TYPE ty_employee.

       OR
DATA : lt_employee type standard table of ty_employee.
```

2. They are the index based internal tables (i.e. in the debugging we can check each and every record has a index).



3. Records can be inserted or appended.

Append :-

- Append statement inserts the record at the last of the internal table.

Insert :-

- Insert statement inserts the records anywhere in the internal table.

```
ls_employee-emp_id = '101'.
ls_employee-emp_name = 'Shivam Singh'.
APPEND ls_employee TO lt_employee.
CLEAR ls_employee.
```

```
*&Inserting at last
ls_employee-emp_id = '102'.
ls_employee-emp_name = 'Abhishek Yadav'.
INSERT ls_employee INTO TABLE lt_employee.
CLEAR ls_employee.

ls_employee-emp_id = '103'.
ls_employee-emp_name = 'Shikhar Srivastava'.
APPEND ls_employee TO lt_employee.
CLEAR ls_employee.

*&Inserting at a particular index
ls_employee-emp_id = '104'.
ls_employee-emp_name = 'Shreyashi Tripathi'.
INSERT ls_employee INTO lt_employee index 2.
CLEAR ls_employee.
```


4. Data is not sorted by default, We can use SORT statement to sort the internal table.

```
SORT lt_employee BY emp_id.
```

5. For standard internal table we can read a record based on key and based on index also.

Based on key

```
READ TABLE lt_employee INTO ls_employee WITH KEY emp_id = '1(
IF sy-subrc EQ 0.
  WRITE :/ ls_employee-emp_id, ls_employee-emp_name.
ENDIF.
```

Based on index

```
READ TABLE lt_employee INTO ls_employee index 1.
IF sy-subrc EQ 0.
```

```
  WRITE :/ ls_employee-emp_id, ls_employee-emp_name.
ENDIF.
```

6. For standard internal tables either we can use the linear search (sequential search) or we can use the binary search to search a record.

Note :-

- You can use binary search if your internal table has been sorted..

```
READ TABLE lt_employee INTO ls_employee with key emp_id =
IF sy-subrc EQ 0.
  WRITE :/ ls_employee-emp_id, ls_employee-emp_name.
ENDIF.
```

- Or else if it is not sorted you can go for linear search.

```
READ TABLE lt_employee INTO ls_employee with key emp_id =
IF sy-subrc EQ 0.
  WRITE :/ ls_employee-emp_id, ls_employee-emp_name.
ENDIF.
```


7. Response time depends upon the number of entries in the internal table.

---

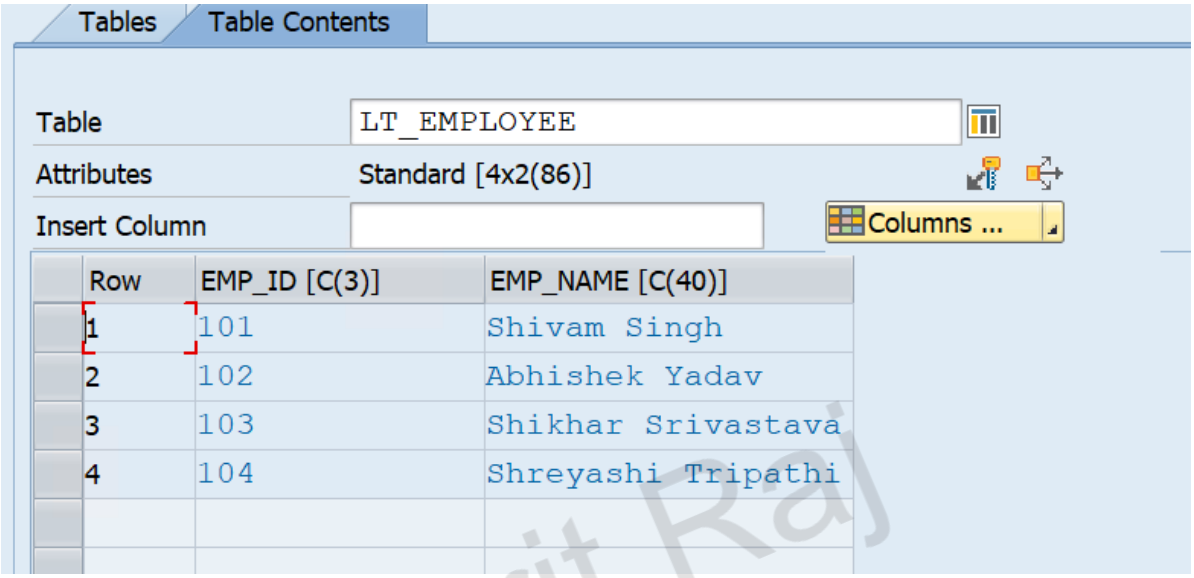
2. Sorted Internal Table

1. Sorted internal tables are a type of internal tables in which data is automatically sorted. We need to specify the key while declaring the sorted internal table.

```
TYPES : BEGIN OF ty_employee,
          emp_id   TYPE zar_emp_id,
          emp_name TYPE zar_emp_name,
        END OF ty_employee.
```

```
DATA : lt_employee TYPE SORTED TABLE OF ty_employee WITH UNI(
       ls_employee TYPE ty_employee.
```

2. They are also indexed based internal table, i.e. data will be stored indexed wise.



3. We should not used append statement for sorted internal table and we should only go for insert statement.

```
ls_employee-emp_id = '101'.
ls_employee-emp_name = 'Shivam Singh'.
insert ls_employee into table lt_employee.
CLEAR ls_employee.

ls_employee-emp_id = '102'.
ls_employee-emp_name = 'Abhishek Yadav'.
insert ls_employee into table lt_employee.
CLEAR ls_employee.

ls_employee-emp_id = '104'.
ls_employee-emp_name = 'Shikhar Srivastava'.
insert ls_employee into table lt_employee.
CLEAR ls_employee.
```

```
ls_employee-emp_id = '103'.
ls_employee-emp_name = 'Shreyashi Tripathi'.
insert ls_employee into table lt_employee.
CLEAR ls_employee.
```

Q. Why we should not use append statement in sorted internal table ?

- Suppose, first three records that we have stored was employee id 101, 102 and 104 and then the fourth record we want to store is 103. So, if you will try to store this as a fourth record and our internal table is sorted by employee so logically 103 should be stored as third row, therefore in this situation it will generate a runtime error.
- And if we will use the insert operation, it will automatically add the record at its suitable position.

4. Data is already sorted, there is no need to use SORT statement.
5. We read a record using key or index.
6. Since, data is already sorted we can use binary search to search a record.
7. Response time of a sorted internal table is very fast as compared to standard internal table.

3. Hashed Internal Table

1. Hashed internal tables are the special type of internal table which works on HASH algorithms.

```
TYPES : BEGIN OF ty_employee,
          emp_id   TYPE zar_emp_id,
          emp_name TYPE zar_emp_name,
        END OF ty_employee.


DATA : lt_employee TYPE HASHED TABLE OF ty_employee WITH UNI(
       ls_employee TYPE ty_employee.
```

Hashing :-

- Hashing is a technique which returns the address of the record based upon the search key without index.

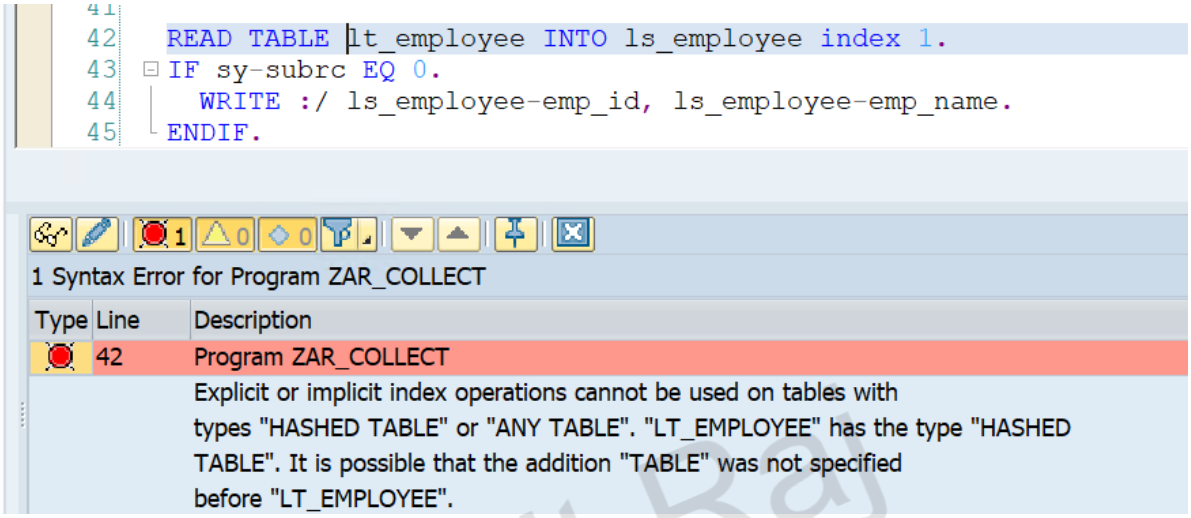
  - Here, if we will go for order number 10, hashing will directly go for address of order number 10.

2. Hashed internal tables do not support the concept of index i.e. why they are not the index based internal table.
3. We use the insert operation, append statement is not supported here, if you will use append statement, it will give a compile time error.

```
ls_employee-emp_id = '101'.
ls_employee-emp_name = 'Shivam Singh'.
insert ls_employee into table lt_employee.
CLEAR ls_employee.

ls_employee-emp_id = '102'.
ls_employee-emp_name = 'Abhishek Yadav'.
insert ls_employee into table lt_employee.
CLEAR ls_employee.

ls_employee-emp_id = '104'.
ls_employee-emp_name = 'Shikhar Srivastava'.
insert ls_employee into table lt_employee.
CLEAR ls_employee.

ls_employee-emp_id = '103'.
ls_employee-emp_name = 'Shreyashi Tripathi'.
```

```
insert ls_employee into table lt_employee.
CLEAR ls_employee.
```

4. There is no impact of SORT, as we read the record based upon the hashed algorithm.
5. We Read a record using key, index is not applicable.



6. Hashed algorithm is used to search a record.
7. Response time is faster as compared to standard and sorted internal table, as response time is independent of number of entries. It is well suited for tables, where a table has huge number of records and we want to search based upon unique key.

---

4. Comparing Standard vs Sorted vs Hashed Internal Tables

1. Index :-

- Standard Internal tables are the index based internal tables.
- Sorted Internal tables are also the index based internal tables.
- Hashed internal tables are not the index based internal tables.

2. Insertion Of Records :-

- Both append and insert operations can be used to insert the records to standard internal tables.
- Only insert operation can be used to insert the records to sorted internal tables.
- Only insert operation can be used to insert the records to hashed internal tables.

3. Sorting :-

- Data is not sorted by default in standard internal table. SORT operation is used sort the data to standard internal table.
- Data is sorted by default. There is no need for SORT operation.
- SORT does not have any impact on the performance of HASHED internal table.

3. Read :-

- The record can be read from standard internal table using KEY or Index.
- They record can be read from sorted internal table using KEY or Index.
- The record can be read from hashed internal table using KEY only.

4. Search :-

- Linear or Binary Search
- Binary Search
- Hashed algorithm

5. Response Time :-

- Response time of standard internal table is less as compared to sorted and hashed internal tables as it uses linear search by default. The response time of standard internal tables depends upon the number of entries. The response time of standard internal table can be improved by using binary search.
- Response time of sorted internal table is fast as compared to standard internal table as it uses binary search.
- Response time of hashed internal table is fast as compared to standard and sorted internal tables. The response time of hashed internal tables does not depend upon the number of

entries. It is well suited for tables, where a table has huge number of records and we want to search based upon unique key.

Some Important Questions :-

1. Which of the internal tables are indexed based internal tables ?

Ans :- Standard, Sorted

2. Is standard internal table only uses linear search ?

Ans :- By default, standard internal table uses linear search, We can use binary search also.

3. Can we use append to insert a record to sorted internal table ?

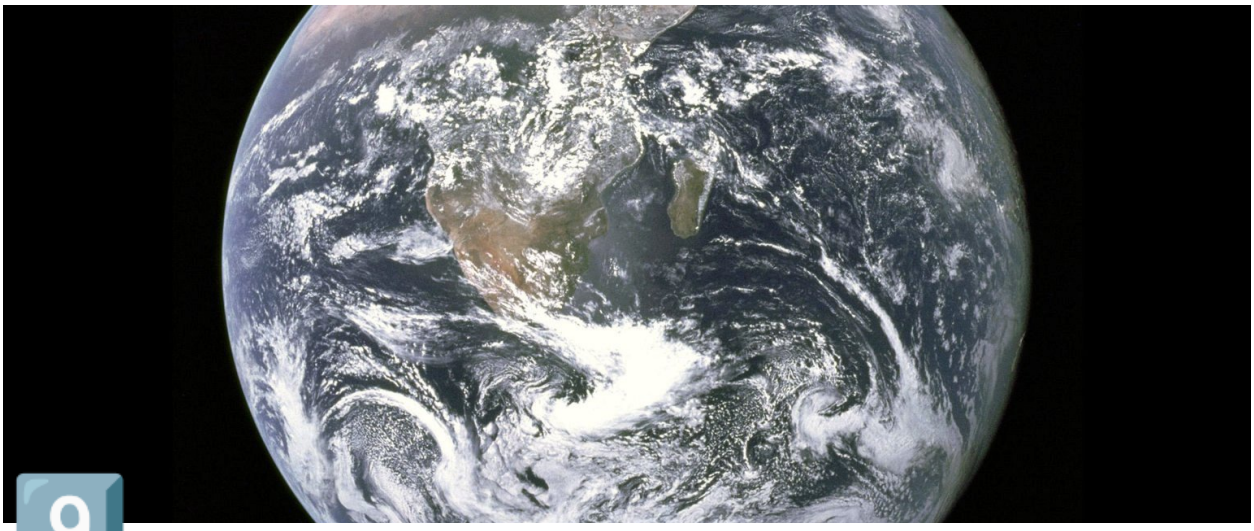
Ans :- No, append is never preferred , as it will result in to runtime error, if we insert a record that is not in sorted order.

4. Which internal table has highest performance when there is large amount of data and search is based upon unique key ?

Ans :- Hash internal Table

5. What is the difference between append and insert ?

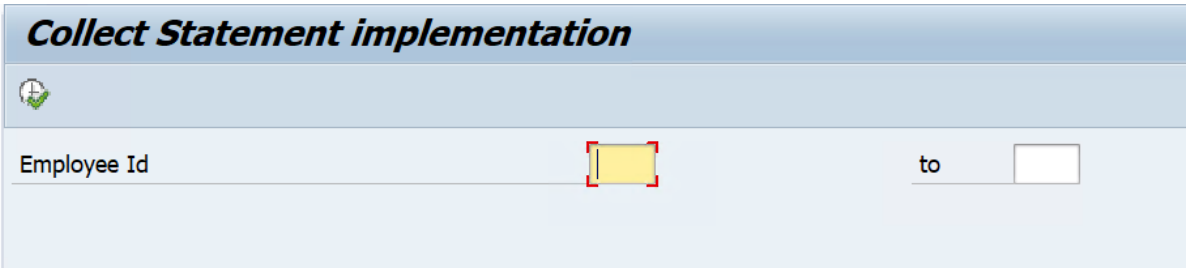
Ans :- Append inserts the record at the last of the internal whereas Insert inserts the record at anywhere in the internal table.



Selection Screen, Parameters, Select-Options

1. Selection Screen

- Selection Screen is also called as input screen i.e. user is going to provide input from selection screen to the program.



- With the help of Selection Screen user provides a input to the program.

What is the Screen Number of Selection Screen ?

→Every Selection Screen or Input Screen always has the number 1000 (It is the standard selection screen).

---

2. Taking input from the User

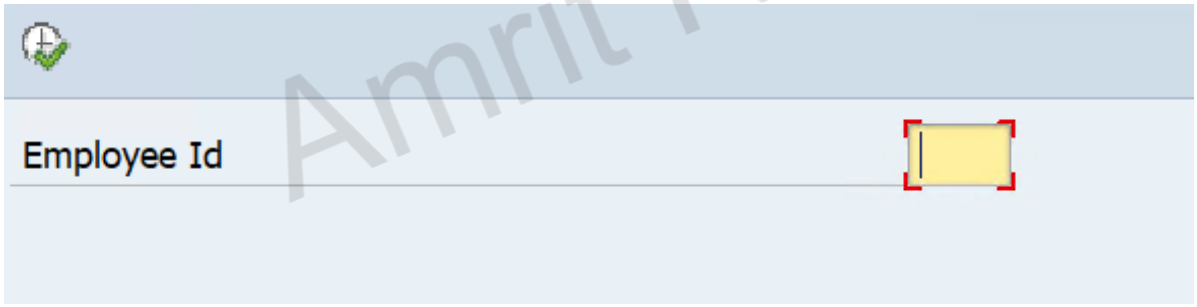
- There are two ways to provide the input to the program :-
  1. Parameters
  2. Select-Options

---

3. Parameters

- Parameters are used to pass the single input to the program.

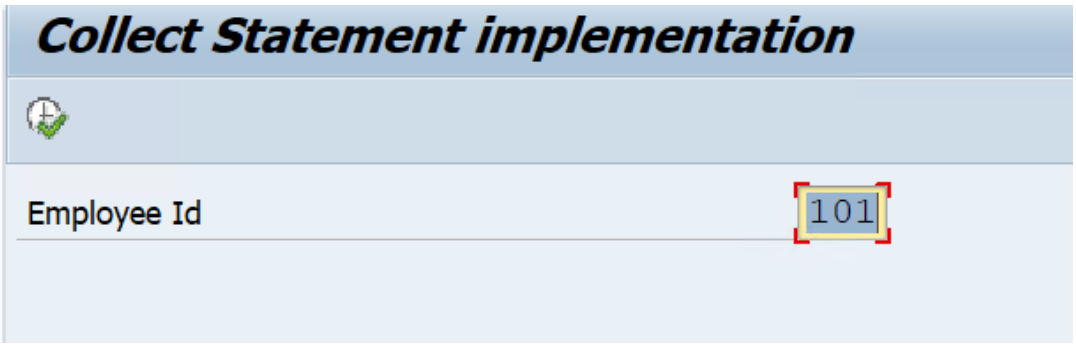
```
PARAMETERS : p_emp_id type ZAR_EMP_ID.
```



Various Parameter Variations :-

1. PARAMETERS <PARAMETER_NAME> …… DEFAULT <VALUE>.
   - It is used to provide a default value to the parameter.

```
PARAMETERS : p_emp_id type ZAR_EMP_ID DEFAULT '101'.
```



2. PARAMETERS <PARAMETER_NAME> …… obligatory.
   - Obligatory is used to make a parameter mandatory, i.e. user must have to provide a input from selection screen.

```
PARAMETERS : p_emp_id type ZAR_EMP_ID OBLIGATORY.
```

3. PARAMETERS <PARAMETER_NAME> ….. RADIOBUTTON GROUP <GROUP_NAME> ..
   - IT is used to create a radio button.

```
PARAMETERS : p_r1 type c RADIOBUTTON GROUP r1,
             p_r2 TYPE c RADIOBUTTON GROUP r1.
```

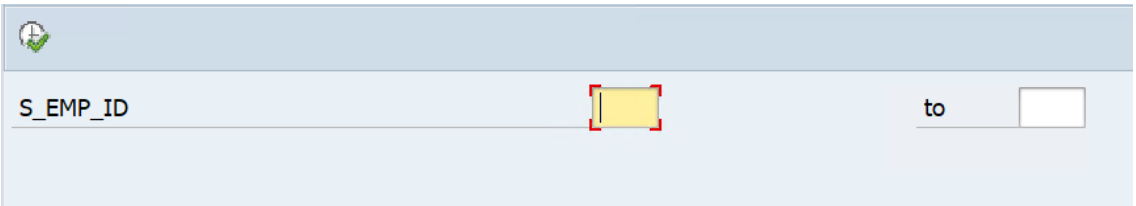
4. PARAMETERS <PARAMETER_NAME> as checkbox…
   - It is used to create a checkbox.

```
PARAMETERS : p_r1 AS CHECKBOX,
             p_r2 as CHECKBOX.
```

4. Select-Options

- Select-Options is used to pass the range of inputs.
  - To create a select option we need to define a local variable for that data element.

```
DATA : lv_emp_id TYPE zar_emp_id.
SELECT-OPTIONS : s_emp_id FOR lv_emp_id.
```



Various Select Option Variations :-

- SELECT-OPTIONS <seltab> FOR <f> …. DEFAULT <g>[TO <h>]
- SELECT-OPTIONS <seltab> FOR <f> …. NO-EXTENSION
  - It is used to remove the extension.

```
DATA : lv_emp_id TYPE zar_emp_id.
SELECT-OPTIONS : s_emp_id FOR lv_emp_id NO-EXTENSION.
```

- SELECT-OPTIONS <seltab> FOR <f> …. NO INTERVALS

```
DATA : lv_emp_id TYPE zar_emp_id.
SELECT-OPTIONS : s_emp_id FOR lv_emp_id NO INTERVALS.
```

- SELECT-OPTIONS <seltab> FOR <f> …. OBLIGATORY

```
DATA : lv_emp_id TYPE zar_emp_id.
SELECT-OPTIONS : s_emp_id FOR lv_emp_id OBLIGATORY.
```

Can we convert a select option into a parameter ?

- Yes we can use no intervals followed by no extension to convert a select option into parameter.

```
DATA : lv_emp_id TYPE zar_emp_id.
SELECT-OPTIONS : s_emp_id FOR lv_emp_id NO INTERVALS no-EX
```

---

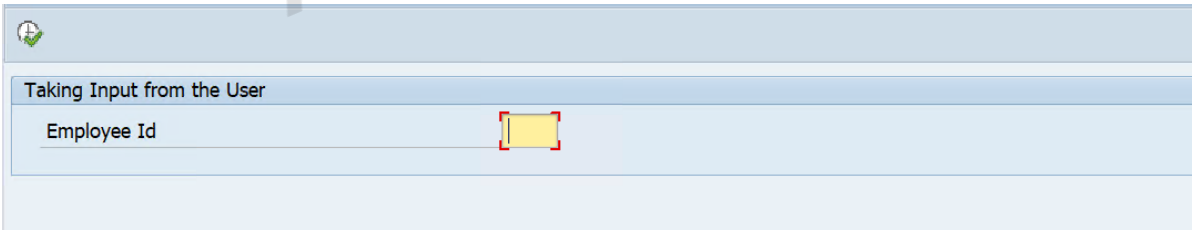
5. Creating Block and Frame on Selection Screen

- On selection screen we can define a block so that we can beautify our selection screen.

Code

```
SELECTION-SCREEN : BEGIN OF BLOCK b1 WITH FRAME TITLE TEXT-0(
  DATA : lv_emp_id TYPE zar_emp_id.
  SELECT-OPTIONS : s_emp_id FOR lv_emp_id NO INTERVALS NO-EX

SELECTION-SCREEN : END OF BLOCK b1.
```


Output



  - We can clearly see, that now our selection screen has become more beautiful and we have create a separate block to take input from the user.

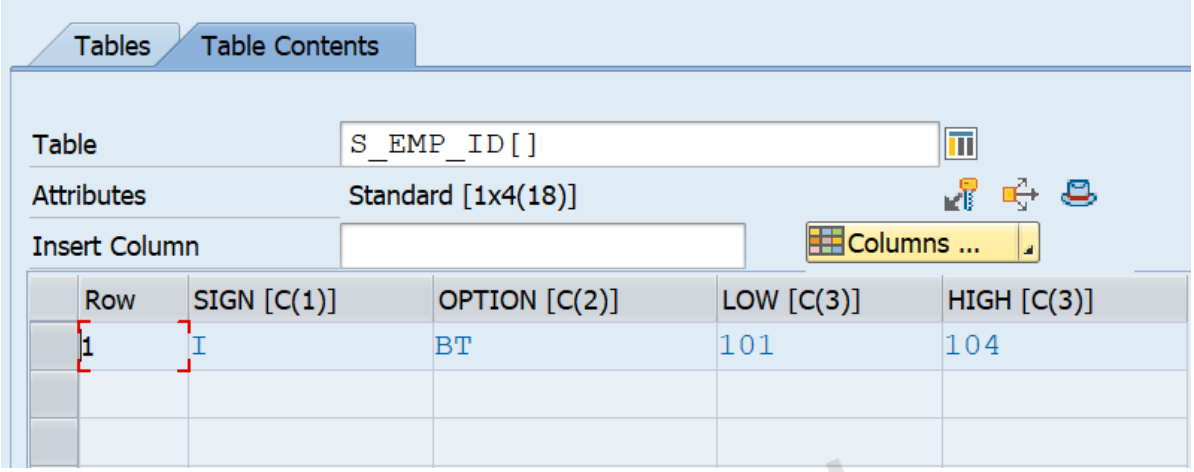
---

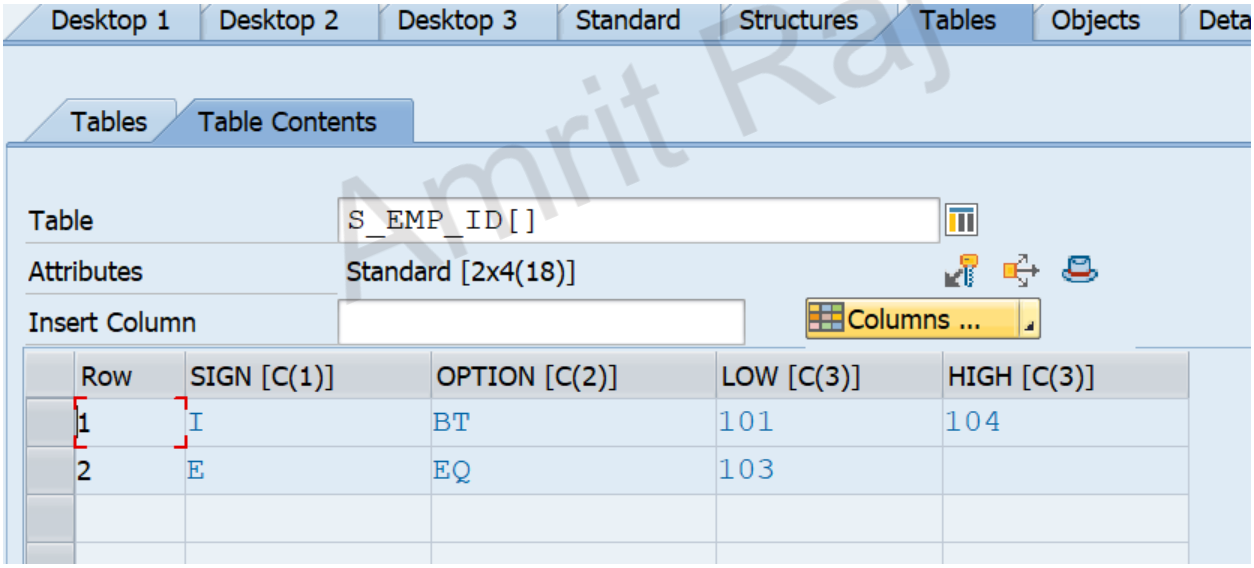
6. Parts of Select-Options

- A select-options has four options.

  1. Sign - I/E (Include/Exclude)

2. Option - Relational Operator (EQ, BT, LT etc.)
3. Low - Low Value
4. High - High Value





Classical Reports, For All Entries

- Reports are programs that read data from the database, processes the data and displays the data in the required format.
- It consists of 2 screens :
  1. Selection Screen
  2. Output Screen
- The Selection Screen is optional.
- SAP ABAP Classical reports are the most basic ABAP reports in which we display the output using WRITE statement.

---

1. Developing First Classical Report

1. Requirement (Taking input from parameter and display output from Employee Table)

Q. How to Write a Select Query

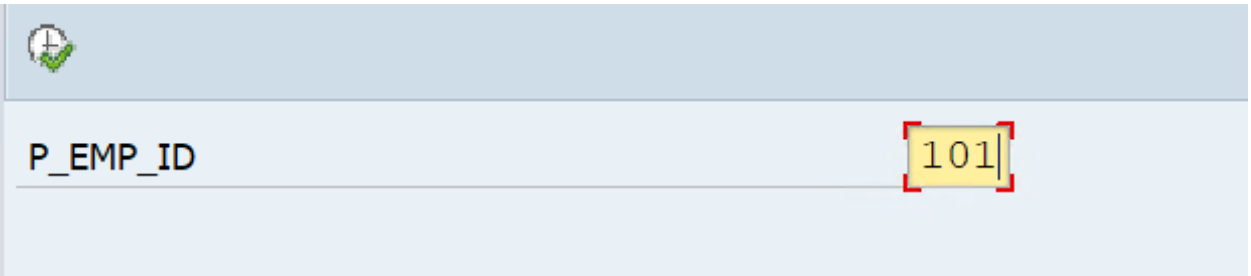
- SQL stands for structured query language
- It is of two types :-
  1. Open SQL :- Open SQL is a database independent.
     - We use open SQL in ABAP Programming
  2. Native SQL :- Native SQL is database dependent.

Implementation :-

```
TYPES : BEGIN OF ty_employee,
          emp_id     TYPE zar_emp_id,
          emp_name   TYPE zar_emp_name,
          department TYPE zar_department,
          manager    TYPE zar_manager,
        END OF ty_employee.

DATA : lt_employee type table of ty_employee,
       ls_employee type ty_employee.



PARAMETERS : p_emp_id type ZAR_EMP_ID.

Select emp_id emp_name department manager
  from ZAR_EMP_TAB
  into table lt_employee
  where emp_id = p_emp_id.



  loop at lt_employee into ls_employee.
    WRITE :/ ls_employee-emp_id, ls_employee-emp_name, ls_employ
             ls_employee-salary, ls_employee-currency.
```

```
  ENDLOOP.
```

Output :-



2. Requirement (Taking input as a select option and displaying the employee details from the employee table)

Implementation :-

```
TYPES : BEGIN OF ty_employee,
          emp_id     TYPE zar_emp_id,
          emp_name   TYPE zar_emp_name,
          department TYPE zar_department,
          manager    TYPE zar_manager,
        END OF ty_employee.

DATA : lt_employee type table of ty_employee,
       ls_employee type ty_employee.
```

```
DATA : lv_emp_id type ZAR_EMP_ID.
SELECT-OPTIONS : s_emp_id for lv_emp_id.

Select emp_id emp_name department manager
  from ZAR_EMP_TAB
  into table lt_employee
  where emp_id in s_emp_id.



  loop at lt_employee into ls_employee.
    WRITE :/ ls_employee-emp_id, ls_employee-emp_name, ls_employ

  ENDLOOP.
```


Output


```
Internal Tables
--------------------------------------------------------------------------------
103 ABHISHEK YADAV                                  SAP ABAP
104 MUDITA GUPTA                                    SAP ABAP
105 SHIKHAR SRIVASTAV                               SAP ABAP
101 SHIVAM SINGH                                    SAP ABAP
102 SHREYASHI TRIPATHI                              SAP ABAP
```

2. For All Entries

- For All Entries is used to fetch out all the records from secondary table based on the data present in primary table.

Requirement :-

1. We will create a select option and we will fetch out records from employee table.
2. now based on records available in employee table, we will fetch out records from our project details table.
3. then we will store all the data into a final table and we will display it on the output screen.
4. Also make the input mandatory.

Solution :-

- Step 1 :- Create a type structure for employee table and project details table and then create a final structure as well, also create a internal table and work area for all.

```
TYPES : BEGIN OF ty_employee,
          emp_id     TYPE zar_emp_id,
          emp_name   TYPE zar_emp_name,
          department TYPE zar_department,
          manager    TYPE zar_manager,
        END OF ty_employee.

TYPES : BEGIN OF ty_project,
          emp_id       TYPE zar_emp_id,
          project_id   TYPE  zar_project_id,
          project_name TYPE zar_project_name,
        END OF ty_project.

TYPES : BEGIN OF ty_final,
          emp_id       TYPE zar_emp_id,
          emp_name     TYPE zar_emp_name,
```

```
          department   TYPE zar_department,
          manager      TYPE zar_manager,
          project_id   TYPE  zar_project_id,
          project_name TYPE zar_project_name,
        END OF ty_final.

DATA : lt_employee TYPE TABLE OF ty_employee,
       ls_employee TYPE ty_employee,
       lt_project  TYPE TABLE OF ty_project,
       ls_project  TYPE ty_project,
       lt_final    TYPE TABLE OF ty_final,
       ls_final    TYPE ty_final.
```

- Step 2 :- Create a Select option.

```
DATA : lv_emp_id TYPE zar_emp_id.
SELECT-OPTIONS : s_emp_id FOR lv_emp_id.
```

- Step 3 :- Write the select query to fetch records from employee table and then use for all entries to fetch out records from the project details table.

```
Select emp_id emp_name department manager
  from ZAR_EMP_TAB
  into table lt_employee
  where emp_id in s_emp_id.
  if lt_employee is NOT INITIAL.
    Select emp_id project_id project_name
      from ZAR_PROJECT_DET
      into table lt_project
      FOR ALL ENTRIES IN lt_employee
      where emp_id = lt_employee-emp_id.

  endif.
```

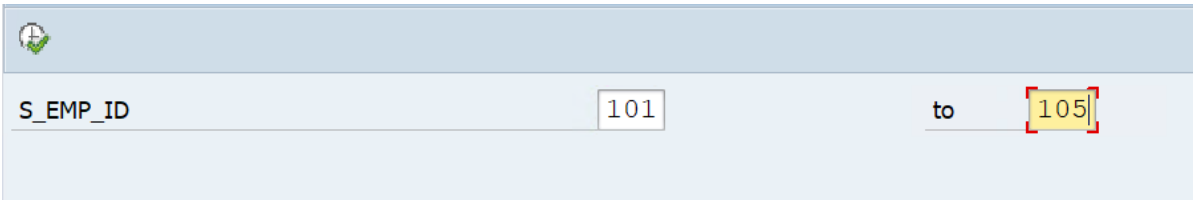
- Step 4 :- Use loop statement to store data into final internal table.

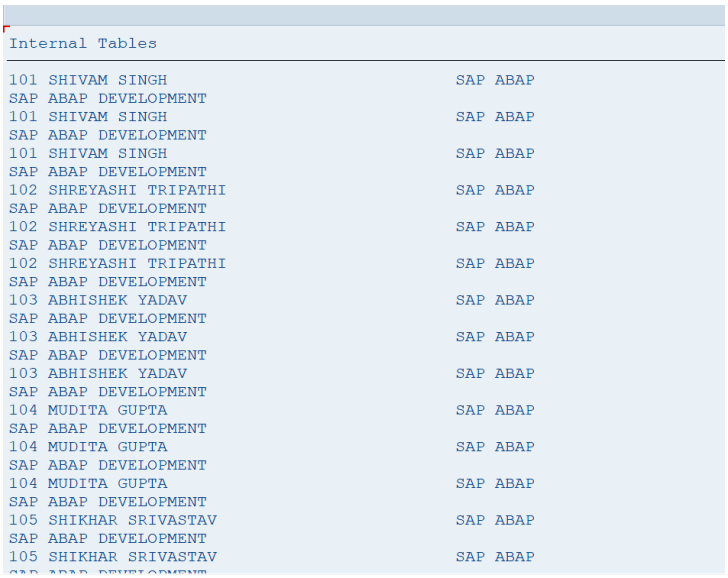
```
LOOP AT lt_project INTO ls_project.
  ls_final-project_id = ls_project-project_id.
  ls_final-project_name = ls_project-project_name.

  READ TABLE lt_employee INTO ls_employee WITH KEY emp_id
  IF sy-subrc EQ 0.
    ls_final-emp_id = ls_employee-emp_id.
    ls_final-emp_name = ls_employee-emp_name.
    ls_final-department = ls_employee-department.
    ls_final-manager = ls_employee-manager.
  ENDIF.

  APPEND ls_final TO lt_final.
  CLEAR ls_final.
ENDLOOP.
```

- Step 5 :- Display the data using write statements.

```
LOOP AT lt_final INTO ls_final.
  WRITE :/ ls_final-emp_id, ls_final-emp_name, ls_final-de
           ls_final-manager, ls_final-project_id, ls_final

ENDLOOP.
```

Output





```
Internal Tables
--------------------------------------------------------------------------------
101 SHIVAM SINGH                            SAP ABAP
SAP ABAP DEVELOPMENT
101 SHIVAM SINGH                            SAP ABAP
SAP ABAP DEVELOPMENT
101 SHIVAM SINGH                            SAP ABAP
SAP ABAP DEVELOPMENT
102 SHREYASHI TRIPATHI                      SAP ABAP
SAP ABAP DEVELOPMENT
102 SHREYASHI TRIPATHI                      SAP ABAP
SAP ABAP DEVELOPMENT
102 SHREYASHI TRIPATHI                      SAP ABAP
SAP ABAP DEVELOPMENT
103 ABHISHEK YADAV                          SAP ABAP
SAP ABAP DEVELOPMENT
103 ABHISHEK YADAV                          SAP ABAP
SAP ABAP DEVELOPMENT
103 ABHISHEK YADAV                          SAP ABAP
SAP ABAP DEVELOPMENT
104 MUDITA GUPTA                            SAP ABAP
SAP ABAP DEVELOPMENT
104 MUDITA GUPTA                            SAP ABAP
SAP ABAP DEVELOPMENT
104 MUDITA GUPTA                            SAP ABAP
SAP ABAP DEVELOPMENT
105 SHIKHAR SRIVASTAV                       SAP ABAP
SAP ABAP DEVELOPMENT
105 SHIKHAR SRIVASTAV                       SAP ABAP
```

Q. When to use For All Entries ?

- Whenever, we will have to fetch data from dependent table we will always go for For All Entries.

1️⃣

Joins in ABAP

1. Introduction to Joins

- Joins are used to fetch/access data from multiple tables using a Single Select Query.

When to use For All Entries In and when to use Joins ?

- When we are working for traditional database (like Oracle, Sy-base) then we will prefer for All Entries.
- When we will be working on HANA database, then we will prefer Joins.

Why we will prefer joins in case of HANA not in case of traditional database ?

- In case of for all entries we write a select query separately for different tables, and load on database is not very high and Since, traditional databases are not very effective we should not load multiple table at once.
- In case of HANA database, HANA is the most effective database and there will be no issue for loading multiple table at once, therefore we can go for Joins.

2. Types of Joins

1. Inner Join :- An inner join finds and returns matching data from the tables.
2. Outer Join :- Outer Join finds and returns matching data and non-matching data from the tables.

---

3. Inner Join

- An inner join finds and returns matching data from tables based upon specified conditions.
- If one or more criteria are not met, no data records are created in the result set.

Version 1

```
tables : zar_emp_tab, zar_project_det.

SELECT ZAR_EMP_TAB~emp_id
       ZAR_EMP_TAB~emp_name
       ZAR_EMP_TAB~department
       ZAR_EMP_TAB~manager
       ZAR_PROJECT_DET~project_id
       ZAR_PROJECT_DET~project_name
  FROM ZAR_EMP_TAB
  JOIN ZAR_PROJECT_DET
    ON ZAR_EMP_TAB~emp_id = ZAR_PROJECT_DET~emp_id
  INTO TABLE lt_final
  WHERE ZAR_EMP_TAB~emp_id IN s_emp_id.
```

Version 2

```
SELECT a~emp_id
       a~emp_name
       a~department
       a~manager
       b~project_id
```

```
      b~project_name
  FROM zar_emp_tab AS a
  JOIN zar_project_det AS b
    ON a~emp_id = b~emp_id
  INTO TABLE lt_final
  WHERE a~emp_id IN s_emp_id.
```

---

4. Outer Join

- Outer Join finds and returns matching data and non-matching data from the tables.
- It is of two types :

1. Left Outer Join

- Left Outer Join takes all the values from the left table and combines them with the values from the right table that meet the criteria.

2. Right Outer Join

- The right outer join takes all the values from the right table and combines them with the values from the left table that meet the criteria.

Outer Join Implementation :-

1. Using Left Outer Join

```
Select a~emp_id
       a~emp_name
       a~department
       a~manager
       b~project_id
       b~project_name
  from ZAR_EMP_TAB as a
  left OUTER join ZAR_PROJECT_DET as b
  on a~emp_id =  b~emp_id
```

```
into table lt_final
where a~emp_id in s_emp_id.
```

2. Using Left Join

```
Select a~emp_id
       a~emp_name
       a~department
       a~manager
       b~project_id
       b~project_name
  from ZAR_EMP_TAB as a
  left join ZAR_PROJECT_DET as b
  on a~emp_id =  b~emp_id
  into table lt_final
  where a~emp_id in s_emp_id.
```

- Similarly we can use right outer join if we have a requirement where we will have to fetch all the records from right tables and only matching tables from left tables.

2️⃣

Message Class in ABAP

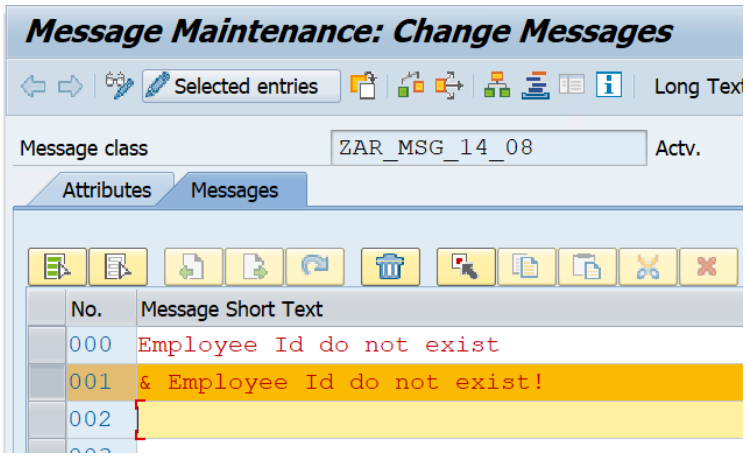
1. Introduction

- Messages play a very crucial role in ABAP Programming.
- A Message is a text that provides information to the user or a programmer about what's happening when code is executed.
- The transaction code to create a message class is SE91.

---

2. Message

- A message number length is 3(000-999).
- One can pass values to a message number using &.



- In a message number we can pass up to 4 &.
- SYNTAX :

  MESSAGE E000(<MSG CLASS>).

  In the above syntax : MESSAGE = keyword, E = error message type, 000 is message number, <MSG CLASS> = name of the message class.

---

3. Types of Messages in ABAP

1. A (Abort)
2. E (Error)
3. I (Information)
4. S (Success/Status)
5. W (Warning)
6. X (Exit)
   - Out of the above 6 messages the most used three messages are Error, Information, Success

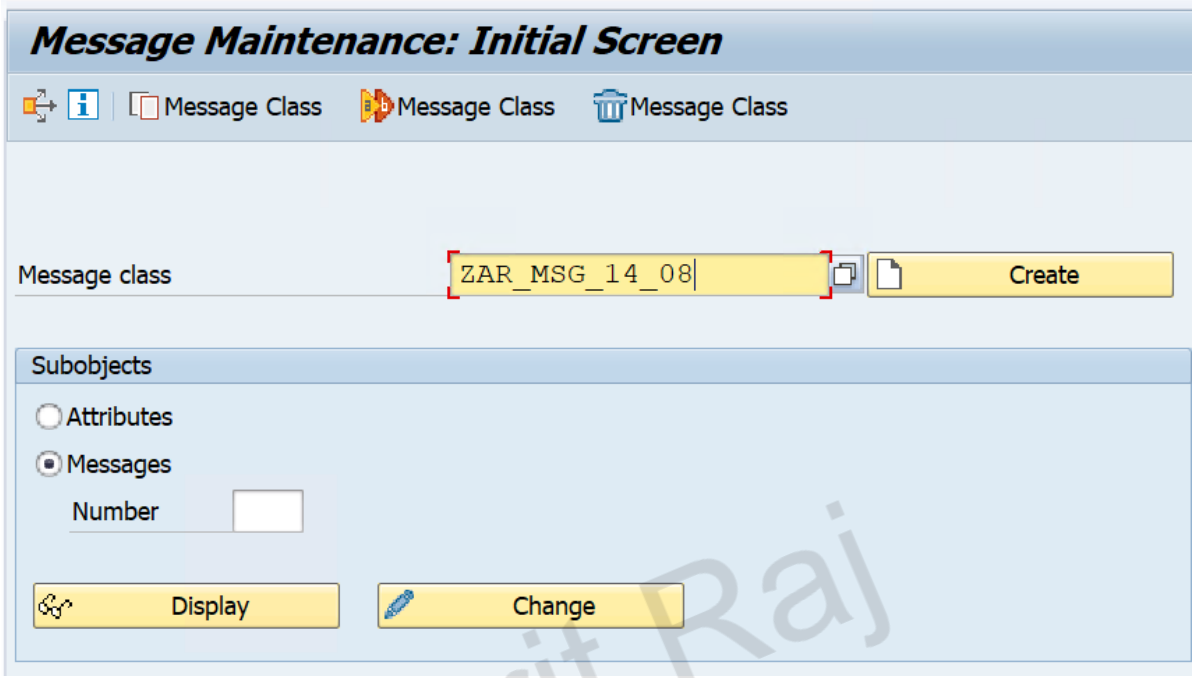
Note :-

- We should always use Message class (SE91) for writing messages, Never hard code messages inside the program.

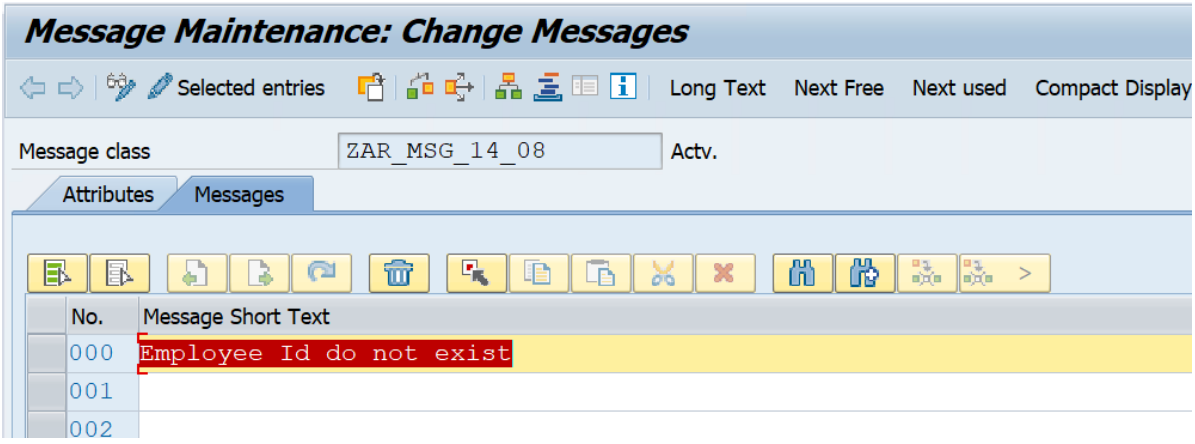
---

4. Message Class Creation

- Step 1 :- Go to SE91 class and give a name for your message class.



- Step 2 :- Click on create button and go to message tab and write the message text which you want to display and click on save button.



5. Requirement :-

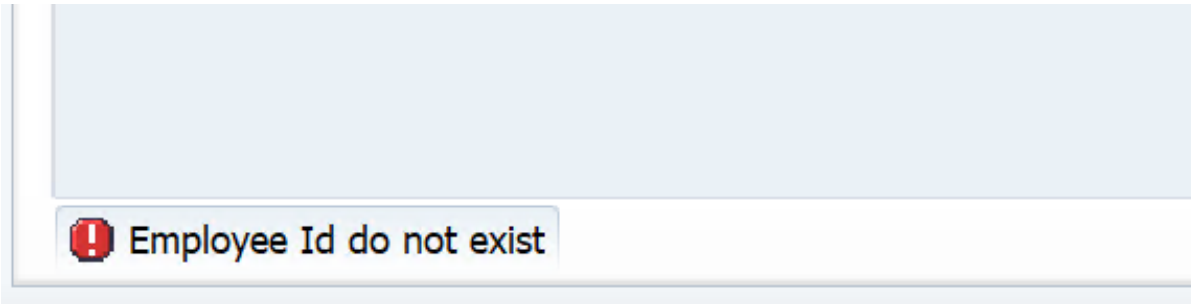
Mob No :- 6261538504

- Create a Program in which take input from parameter and write a select query to fetch records from employee table and if record is not there display a message that Employee Id do not exist.

Implementation

```
TYPES : BEGIN OF ty_employee,
          emp_id     TYPE zar_emp_id,
          emp_name   TYPE zar_emp_name,
          department TYPE zar_department,
          manager    TYPE zar_manager,
        END OF ty_employee.

DATA : lt_employee TYPE TABLE OF ty_employee.


PARAMETERS : p_emp_id TYPE zar_emp_id.

SELECT emp_id emp_name department manager
  FROM zar_emp_tab
  INTO TABLE lt_employee
  WHERE emp_id = p_emp_id.
IF lt_employee IS INITIAL.
  MESSAGE e000(zar_msg_14_08).
ENDIF.
```

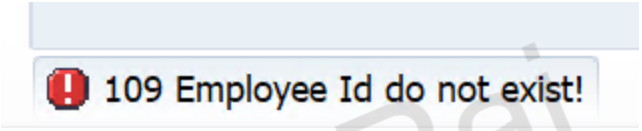
Output



Output 2 :-

- If you want to pass the employee id.

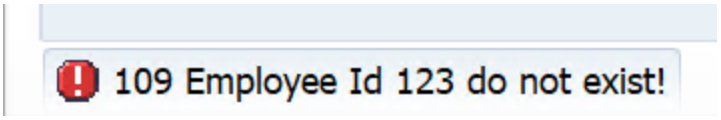
```
MESSAGE e001(zar_msg_14_08) with p_emp_id.
```



Note :-

- If you want to pass multiple numbers, you can use the below syntax.

```
MESSAGE e001(zar_msg_14_08) with p_emp_id '123'.
```

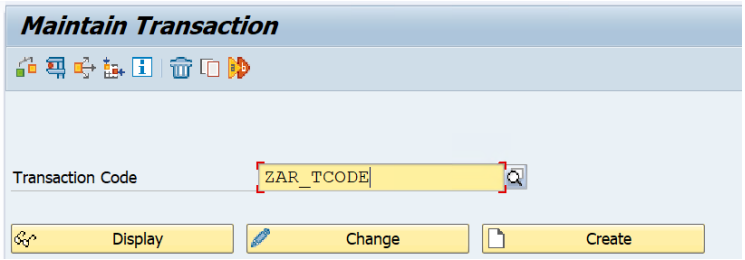


Creating TCODE for ABAP Program

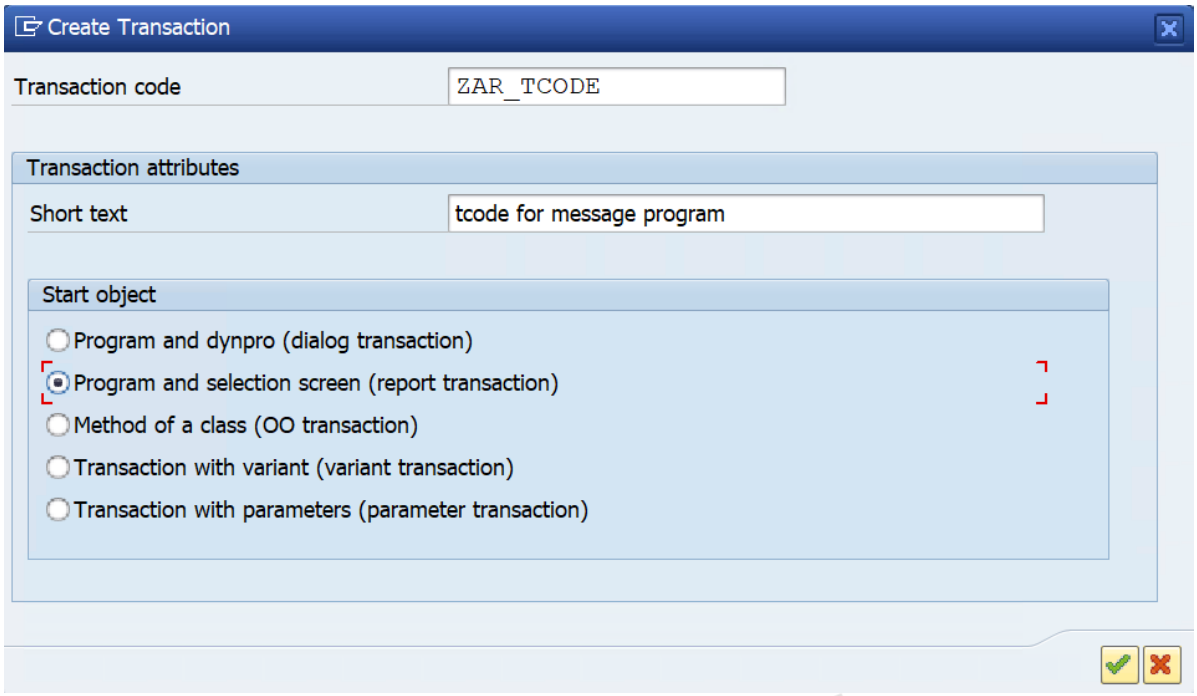
- We can create our own shortcuts/transactions to access the applications.
- The transaction code to create a transaction code is SE93.

1. How to create a transaction code ?

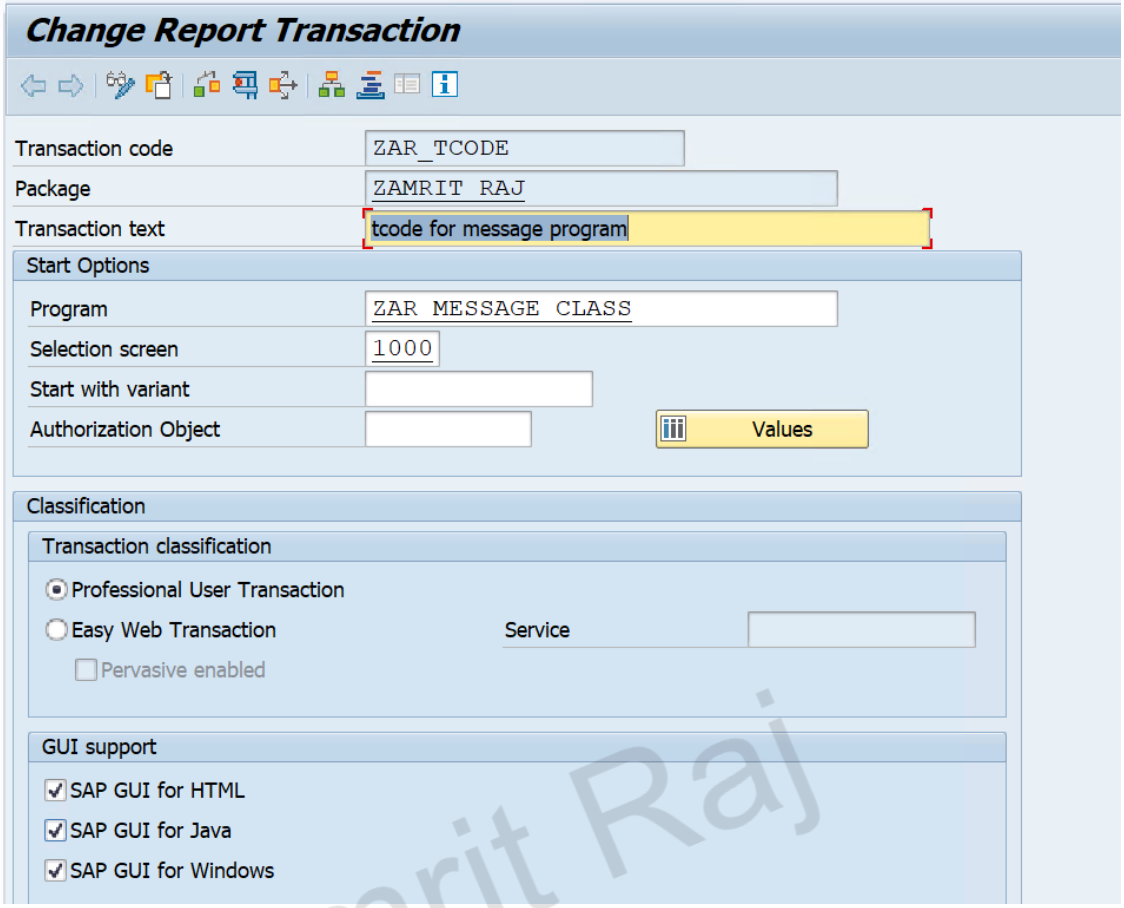
- Step 1 :- Go to SE93 and provide a name for your transaction code.



- Step 2 :- Click on create button and provide a short description and select the 2nd radio button.



- Step 3 :- Click on Okay button.
    - Provide the program name and select all the GUI.



- Click on save button and assign the package and transport request.

2. Executing the TCODE

- It will directly take you to the selection screen.

Best Performance Guidelines in ABAP Programming

1. Never use * in the Select Query, Always fetch the data of only those columns which are required.
2. Column fetching sequence needs to be same as that of data dictionary column sequence.
3. Where condition column sequence needs to be same as that of data dictionary column sequence.
4. Never use corresponding in the query.
5. For traditional databases avoid using JOIN, use For ALL Entries IN.
6. For fetching records from foreign key tables/dependent tables - Always check for SY-SUBRC condition or internal table not initial condition.
7. Use Binary Search in Read Table.
8. Use Parallel Cursor in nested loops.
9. For multiple conditions, Use CASE conditional statement rather than IF conditional statement.

10. Depending upon the requirement, create secondary indexes to improve the performance.

1. Parallel Cursor

- When we have multiple records in case of our header table, then Read operation will not work effectively as we will not be able to fetch out all the records from header as Read statement only returns the first matching records from the internal table.
- So, in this situation we will forced to use loop within a loop (nested loop), so to increases efficiency we will implement the concept of Parallel Cursor.
- Always use parallel cursor when you are using nested loops.

Steps :-

- Step 1 :- Sort the item internal table.

```
SORT lt_project by emp_id.
```

- Step 2 :- Just before starting of inner loop we will use a Read statement to fetch the matching item records using binary search.
  - IF Read statement is true we will store the SY-TABIX value in a variable.

```
SORT lt_project by emp_id.
  DATA : lv_index type i.
  Loop at lt_employee into ls_employee.
    Read table lt_project into ls_project with key emp_id
    if sy-subrc eq 0.
      lv_index = sy-tabix.
    endif.
    loop at lt_project into ls_project where emp_id = ls_e


    ENDLOOP.
  ENDLOOP.
```

- Step 3 :- Now rather going for where condition on inner loop we will go for from condition on inner loop.

```
loop at lt_project into ls_project from lv_index.


    ENDLOOP.
```

- Step 4 :- We will also check that the inner loop should only execute for those primary keys in which data is also present in header.

```
LOOP AT lt_project INTO ls_project FROM lv_index.
    IF ls_employee-emp_id  <> ls_project-emp_id.
      EXIT.
    ELSE.


    ENDIF.
  ENDLOOP.
```


Implementation

```
SORT lt_project BY emp_id.
DATA : lv_index TYPE i.
LOOP AT lt_employee INTO ls_employee.
  READ TABLE lt_project INTO ls_project WITH KEY emp_id = ls_emp
  IF sy-subrc EQ 0.
    lv_index = sy-tabix.
  ENDIF.
  LOOP AT lt_project INTO ls_project FROM lv_index.
    IF ls_employee-emp_id  <> ls_project-emp_id.
      EXIT.
    ELSE.
      ls_final-emp_id = ls_employee-emp_id.
```

```
      ls_final-emp_name = ls_employee-emp_name.
      ls_final-department = ls_employee-department.
      ls_final-manager = ls_employee-manager.
      ls_final-project_id = ls_project-project_id.
      ls_final-project_name = ls_project-project_name.
      APPEND ls_final TO lt_final.
      CLEAR ls_final.
    ENDIF.
  ENDLOOP.
ENDLOOP.
```

Complete Code

```
TYPES : BEGIN OF ty_employee,
          emp_id     TYPE zar_emp_id,
          emp_name   TYPE zar_emp_name,
          department TYPE zar_department,
          manager    TYPE zar_manager,
        END OF ty_employee.

TYPES  : BEGIN OF ty_project,
           emp_id       TYPE zar_emp_id,
           project_id   TYPE zar_project_id,
           project_name TYPE zar_project_name,
         END OF ty_project.

TYPES : BEGIN OF ty_final,
          emp_id       TYPE zar_emp_id,
          emp_name     TYPE zar_emp_name,
          department   TYPE zar_department,
          manager      TYPE zar_manager,
          project_id   TYPE zar_project_id,
          project_name TYPE zar_project_name,
        END OF ty_final.
```

```
DATA : lt_employee TYPE TABLE OF ty_employee,
       ls_employee TYPE ty_employee,
       lt_project  TYPE TABLE OF ty_project,
       ls_project  TYPE ty_project,
       lt_final    TYPE TABLE OF ty_final,
       ls_final    TYPE ty_final.




DATA : lv_emp_id TYPE zar_emp_id.
SELECT-OPTIONS : s_emp_id FOR lv_emp_id.

SELECT emp_id emp_name department manager
  FROM zar_emp_tab
  INTO TABLE lt_employee
  WHERE emp_id IN s_emp_id.
IF lt_employee IS NOT INITIAL.
  SELECT emp_id project_id project_name
    FROM zar_project_det
    INTO TABLE lt_project
    FOR ALL ENTRIES IN lt_employee
    WHERE  emp_id = lt_employee-emp_id.

ENDIF.



SORT lt_project BY emp_id.
DATA : lv_index TYPE i.
LOOP AT lt_employee INTO ls_employee.
  READ TABLE lt_project INTO ls_project WITH KEY emp_id = ls_emp
  IF sy-subrc EQ 0.
    lv_index = sy-tabix.
  ENDIF.
  LOOP AT lt_project INTO ls_project FROM lv_index.
    IF ls_employee-emp_id  <> ls_project-emp_id.
```

```
      EXIT.
    ELSE.
      ls_final-emp_id = ls_employee-emp_id.
      ls_final-emp_name = ls_employee-emp_name.
      ls_final-department = ls_employee-department.
      ls_final-manager = ls_employee-manager.
      ls_final-project_id = ls_project-project_id.
      ls_final-project_name = ls_project-project_name.
      APPEND ls_final TO lt_final.
      CLEAR ls_final.
    ENDIF.
  ENDLOOP.
ENDLOOP.


LOOP AT lt_final INTO ls_final.
  WRITE :/ ls_final-emp_id, ls_final-emp_name, ls_final-departme

ENDLOOP.
```

Classical Report Events

- Classical Reports are the most basic ABAP reports in which we display the output using WRITE statement.
- Event is always triggered by a specific action or a specific occurrence (when a particular time has been reached).
- The same concept of events is applicable to classical Reports. Some events triggered by specific user action and some events triggered at specific occurrence in classical reports.

Various Classical Report Events

1. Initialization
2. At Selection-Screen
3. Start-Of-Selection
4. End-Of-Selection
5. Top-Of-Page
6. End-Of-Page
7. At Selection-Screen Output

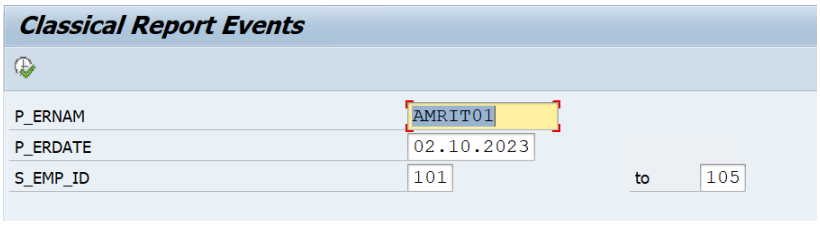
8. At Selection-Screen on Value Request for <Field>
9. At Selection-Screen on Help Request for <Field>

1. Initialization

- Initialization event is triggered before displaying the selection screen or input screen.
- The purpose of this event is to assign the default values to parameters and select options.

```
DATA : lv_emp_id TYPE zar_emp_id.
PARAMETERS : p_ernam  TYPE ernam,
             p_erdate TYPE erdat.
SELECT-OPTIONS : s_emp_id FOR lv_emp_id.


INITIALIZATION.
  p_ernam = sy-uname.
  p_erdate = sy-datum.

  s_emp_id-sign = 'I'.
  s_emp_id-option = 'BT'.
  s_emp_id-low = '101'.
  s_emp_id-high = '105'.
  APPEND s_emp_id.
```


Output



2. At Selection Screen

- At Selection Screen events is triggered when a user performs some actions on selection screen.
    - E.g. enter, any click etc.
- The purpose of this event is to validate the input on Selection Screen.

```
PARAMETERS  : p_emp_id TYPE zar_emp_id.

AT SELECTION-SCREEN.
  IF p_emp_id <> '101'.
    MESSAGE 'Input is Wrong!' TYPE 'I'.
  ELSE.
    MESSAGE 'Input is Correct!' TYPE 'S'.
  ENDIF.
```

Output

3. Start-Of-Selection

- Start of Selection event triggers when user clicks on Execute button (F8) on the selection screen.
- Selection Logic is a part of this event.

```
TYPES : begin of ty_employee,
  emp_id type ZAR_EMP_ID,
  EMP_NAME type  ZAR_EMP_NAME,
  DEPARTMENT type ZAR_DEPARTMENT,
  MANAGER type ZAR_MANAGER,
  end of ty_employee.

DATA : lt_employee type table of ty_employee,
       ls_employee type ty_employee.

DATA : lv_Emp_id type zar_emp_id.
SELECT-OPTIONS : s_emp_id for lv_emp_id.


START-OF-SELECTION.
Select emp_id emp_name department manager
  from ZAR_EMP_TAB
  into table lt_employee
  where emp_id in s_emp_id.

cl_demo_output=>display( lt_employee ).
```


Output

Output

LT_EMPLOYEE

EMP_ID	EMP_NAME	DEPARTMENT	MANAGER
101	SHIVAM SINGH	SAP ABAP	AMRIT RAJ
103	ABHISHEK YADAV	SAP ABAP	AMRIT RAJ
104	MUDITA GUPTA	SAP ABAP	AMRIT RAJ
105	SHIKHAR SRIVASTAV	SAP ABAP	AMRIT RAJ
102	SHREYASHI TRIPATHI	SAP ABAP	AMRIT RAJ
106	ATUL KUMAR	SAP ABAP	AMRIT RAJ
107	BUDDHI VISHWAS	SAP ABAP	AMRIT RAJ

4. End Of Selection

- When our Selection Process ends, SAP automatically call the event End of Selection.
- This event calls when selection process ends.
- This event helps to identify the end of data/records.

```
TYPES : BEGIN OF ty_employee,
          emp_id     TYPE zar_emp_id,
          emp_name   TYPE  zar_emp_name,
          department TYPE zar_department,
          manager    TYPE zar_manager,
        END OF ty_employee.

DATA : lt_employee TYPE TABLE OF ty_employee,
       ls_employee TYPE ty_employee.

DATA : lv_Emp_id TYPE zar_emp_id.
SELECT-OPTIONS : s_emp_id FOR lv_emp_id.

START-OF-SELECTION.
```

```
  SELECT emp_id emp_name department manager
    FROM zar_emp_tab
    INTO TABLE lt_employee
    WHERE emp_id IN s_emp_id.

  LOOP AT lt_employee INTO ls_employee.
    WRITE :/ ls_employee-emp_id, ls_employee-emp_name, ls_emp
  ENDLOOP.



END-OF-SELECTION.
  WRITE :/ 'End of Selection'.
```

Output

5. Top of Page

- TOP-OF-PAGE event calls when the first WRITE statement occurs in the program.
- This event triggers automatically when the situation or time has been reached.
- The purpose of this event is to provide title/header at the beginning of a new page.

```
TYPES : BEGIN OF ty_employee,
          emp_id     TYPE zar_emp_id,
```

```
           emp_name   TYPE  zar_emp_name,
           department TYPE zar_department,
           manager    TYPE zar_manager,
         END OF ty_employee.

DATA : lt_employee TYPE TABLE OF ty_employee,
       ls_employee TYPE ty_employee.

DATA : lv_Emp_id TYPE zar_emp_id.
SELECT-OPTIONS : s_emp_id FOR lv_emp_id.

START-OF-SELECTION.
  SELECT emp_id emp_name department manager
    FROM zar_emp_tab
    INTO TABLE lt_employee
    WHERE emp_id IN s_emp_id.

  LOOP AT lt_employee INTO ls_employee.
    WRITE :/ ls_employee-emp_id, ls_employee-emp_name, ls_emp
  ENDLOOP.



END-OF-SELECTION.
  WRITE :/ 'End of Selection'.


TOP-OF-PAGE.
  WRITE :/ 'Employee Details !'.
```


Output :

6. End of Page

- END-OF-PAGE event calls when the last WRITE statement occurs in a program for a page.
- This event triggers automatically when the situation or time has been reached.
- The purpose of End of Page Event is to provide footer or some information at the end of a new page.
- For End of Page event to trigger we need to provide the line count.

```
REPORT zar_events LINE-COUNT 13(2).
```


Example :-

- LINE COUNT 13(2), It means that total number of lines on a page is 13 and out of that 2 are reserved for end of page.

Code :-

```
REPORT zar_events LINE-COUNT 13(2).

TYPES : BEGIN OF ty_employee,
          emp_id     TYPE zar_emp_id,
          emp_name   TYPE  zar_emp_name,
          department TYPE zar_department,
          manager    TYPE zar_manager,
        END OF ty_employee.
```

```
DATA : lt_employee TYPE TABLE OF ty_employee,
       ls_employee TYPE ty_employee.

DATA : lv_Emp_id TYPE zar_emp_id.
SELECT-OPTIONS : s_emp_id FOR lv_emp_id.

START-OF-SELECTION.
  SELECT emp_id emp_name department manager
    FROM zar_emp_tab
    INTO TABLE lt_employee
    WHERE emp_id IN s_emp_id.

  LOOP AT lt_employee INTO ls_employee.
    WRITE :/ ls_employee-emp_id, ls_employee-emp_name, ls_emp
  ENDLOOP.



END-OF-SELECTION.
  WRITE :/ 'End of Selection'.


TOP-OF-PAGE.
  WRITE :/ 'Employee Details !'.

END-OF-PAGE.
  WRITE:/ 'End of Employee Details !'.
```


Output :-

7. At Selection Screen Output

- At Selection Screen Output Event triggers before displaying the selection screen or input screen.
- The purpose of this event is to modify the selection screen.

Note :-

- We have seen that, Initialization event also triggers just before displaying the selection screen. So, What's the difference between these two events ?

Initialization Vs At Selection Screen Output Comparison :-

- Initialization event calls only first time before displaying the selection screen whereas at selection screen output event calls every time before displaying a selection screen.
- Initialization event is to assign the default values to parameters and select-options whereas at selection-screen output event is used to modify the selection screen.

At Selection Screen and At Selection Screen Output Comparison

- At selection screen event calls when user performs some action(enter, Click etc.) on to the selection screen whereas at selection screen output event calls before displaying the selection screen.

- At selection screen event is to validate the input whereas at selection screen output is used to modify the selection screen.
- At selection screen event is equal to PAI(Process After Input) of module pool whereas at selection-screen output is equal to PBO(Process before Output) of module pool.

At Selection Screen Output Event Implementation :-

Requirement :

1. We will create two radio buttons on Selection Screen first for Employee Details and Second for Project Details.
2. Then we will create 1 parameter for employee id and 1 select option for employee id on selection screen.
3. When we click on Employee Details Radio button, Selection Option will become invisible and Employee data will be displayed and when we will click on Project Details radio button, the parameter radio button will become invisible and Project details data will be displayed.

Implementation :-

```
AT SELECTION-SCREEN OUTPUT.
  IF p_r1 = 'X'.
    LOOP AT SCREEN.
      IF screen-group1 = 'SID'.
        screen-input = 0.
        MODIFY SCREEN.
      ENDIF.
    ENDLOOP.
  ELSEIF p_r2 = 'X'.
    LOOP AT SCREEN.
      IF screen-group1 = 'EID'.
        screen-input = 0.
        MODIFY SCREEN.
      ENDIF.
    ENDLOOP.
```

```
   ENDIF.
```

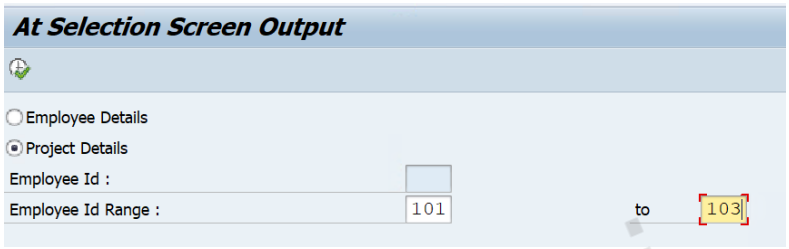
Complete Code :-

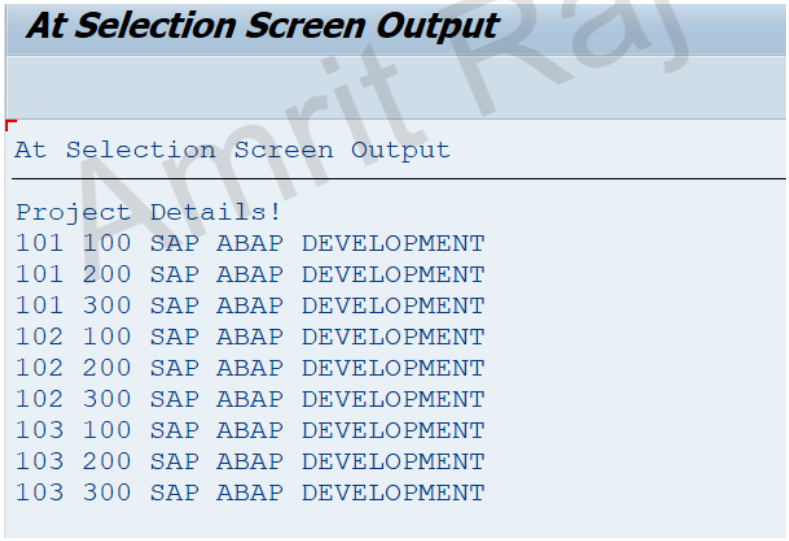
```
TYPES : BEGIN OF ty_employee,
          emp_id     TYPE zar_emp_id,
          emp_name   TYPE zar_emp_name,
          department TYPE zar_department,
          manager    TYPE zar_manager,
        END OF ty_employee.

TYPES : BEGIN OF ty_project,
          emp_id       TYPE zar_emp_id,
          project_id   TYPE ZAR_PROJECT_id,
          project_name TYPE zar_project_name,
        END OF ty_project.

DATA : lt_Employee TYPE TABLE OF ty_employee,
       ls_employee TYPE ty_employee,
       lt_project  TYPE TABLE OF ty_project,
       ls_project  TYPE ty_project.


PARAMETERS : p_r1 TYPE c RADIOBUTTON GROUP r1,
             p_r2 TYPE c RADIOBUTTON GROUP r1.
DATA : lv_Emp_ID TYPE zar_emp_id.
PARAMETERS : p_emp_id TYPE zar_emp_id MODIF ID eid.
SELECT-OPTIONS : s_emp_id FOR lv_emp_id MODIF ID sid.


AT SELECTION-SCREEN OUTPUT.
  IF p_r1 = 'X'.
    LOOP AT SCREEN.
```

```
      IF screen-group1 = 'SID'.
        screen-input = 0.
        MODIFY SCREEN.
      ENDIF.
    ENDLOOP.
  ELSEIF p_r2 = 'X'.
    LOOP AT SCREEN.
      IF screen-group1 = 'EID'.
        screen-input = 0.
        MODIFY SCREEN.
      ENDIF.
    ENDLOOP.


  ENDIF.

START-OF-SELECTION.
  IF p_r1 = 'X'.
    SELECT emp_id emp_name department manager
      FROM zar_emp_tab
      INTO TABLE lt_employee
      WHERE emp_id = p_emp_id.
      WRITE :/ 'Employee Details'.
    LOOP AT lt_employee INTO ls_employee.
      WRITE :/ ls_employee-emp_id, ls_employee-emp_name, ls_e
               ls_employee-manager.


    ENDLOOP.
  ELSEIF p_r2 = 'X'.
    SELECT emp_id project_id project_name
      FROM zar_project_det
      INTO TABLE lt_project
      WHERE emp_id IN s_emp_id.

    WRITE :/ 'Project Details!'.
```

```
    LOOP AT lt_project INTO ls_project.
      WRITE :/ ls_project-emp_id, ls_project-project_id, ls_
    ENDLOOP.
  ENDIF.
```

Output





8. At Selection Screen on Value Request for <Field>

- This event calls when user clicks F4 on a field of Selection Screen.
- The purpose of this event is to provide value help for a input field.

```
AT SELECTION-SCREEN on VALUE-REQUEST FOR p_emp_id.
```

- We will see this implementation in Module Pool Programming chapter.

---

9. At Selection-screen On Help Request For <Field>

- This event calls when user clicks F1 on a field of Selection Screen.
- The purpose of this event is to provide technical information for a field.

```
AT SELECTION-SCREEN on HELP-REQUEST FOR p_emp_id.
```

6️⃣

Control Break Statements

1. Introduction

- Control break statements are also called as control break events.
- Control break statements are used to control loop or we can say they are used to control the data flow in loop.
- Control break statement starts with AT and ends with ENDAT.

Pre-requisites for Control Break Statements :

1. Control Break statements should be applied inside a loop.
2. Internal Table should be in the sorted order.

---

2. Types of Control Break Statements :

- The various control break statements are as follows :
  1. AT FIRST :- It triggers for the first record of the internal table in the loop.
  2. AT LAST :- It triggers for the last record of the internal table in the loop.

Mob No :- 6261538504

3. AT NEW <FIELD_NAME> :- It triggers for the first record of a group having same value for the specified fieldname in loop.
4. AT END OF <FIELD_NAME> :- It triggers for the last record of a group having the same values for the specified fieldname in loop.

---

3. AT First

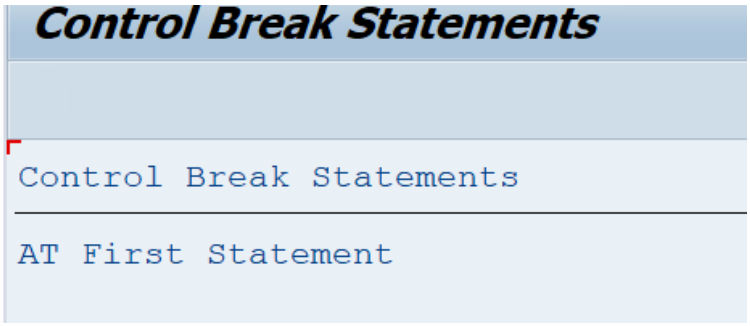
- At first statement triggers for the first record of the internal table.

Implementation on Project Details Table :-

```
TYPES : BEGIN OF ty_project,
          emp_id       TYPE zar_emp_id,
          project_id   TYPE zar_project_id,
          project_name TYPE zar_project_name,
        END OF ty_project.

DATA : lt_project TYPE TABLE OF ty_project,
       ls_project TYPE ty_project.

DATA: lv_emp_id TYPE zar_emp_id.

SELECT-OPTIONS : s_emp_id FOR lv_emp_id.

SELECT emp_id project_id project_name
  FROM zar_project_det
  INTO TABLE lt_project
  WHERE emp_id IN s_emp_id.

SORT lt_project BY emp_id.

LOOP AT lt_project INTO ls_project.
  AT FIRST.
    WRITE :/ 'First record:', ls_project-emp_id, ls_project-p
```

```
  ENDAT.
ENDLOOP.
```

Output



4. At Last Statement

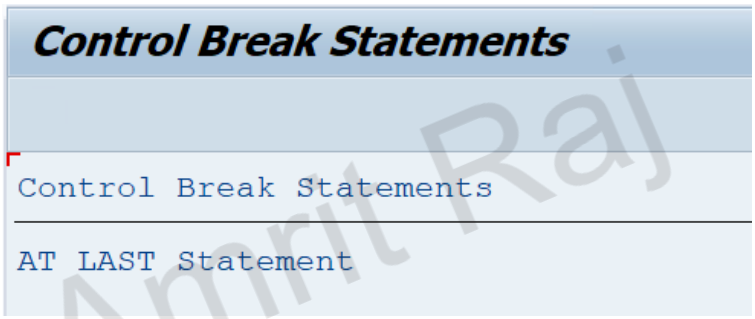
- At Last statement statement triggers for the last record of the internal table.

Implementation

```
TYPES : BEGIN OF ty_project,
          emp_id      TYPE zar_emp_id,
          project_id   TYPE zar_project_id,
          project_name TYPE zar_project_name,
        END OF ty_project.

DATA : lt_project TYPE TABLE OF ty_project,
       ls_project TYPE ty_project,
       lt_prject2 type TABLE of ty_project.

DATA: lv_emp_id TYPE zar_emp_id.

SELECT-OPTIONS : s_emp_id FOR lv_emp_id.

SELECT emp_id project_id project_name
  FROM zar_project_det
```

```
  INTO TABLE lt_project
  WHERE emp_id IN s_emp_id.

SORT lt_project by emp_id.

LOOP AT lt_project INTO ls_project.
  AT LAST.
    WRITE :/ 'AT LAST Statement'.
  ENDAT.
ENDLOOP.
```

Output



5. At New <FIELD_NAME>.

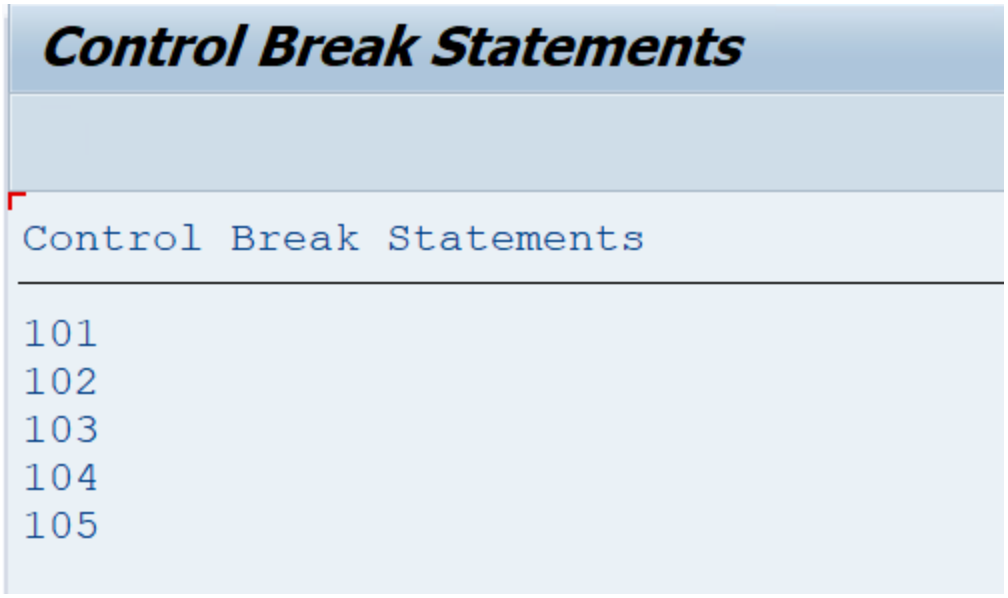
- It triggers for the first record of a group having same value for the specified fieldname in loop.

Implementation

```
TYPES : BEGIN OF ty_project,
          emp_id       TYPE zar_emp_id,
          project_id   TYPE zar_project_id,
          project_name TYPE zar_project_name,
        END OF ty_project.

DATA : lt_project TYPE TABLE OF ty_project,
```

```
       ls_project TYPE ty_project,
       lt_prject2 type TABLE of ty_project.

DATA: lv_emp_id TYPE zar_emp_id.

SELECT-OPTIONS : s_emp_id FOR lv_emp_id.

SELECT emp_id project_id project_name
  FROM zar_project_det
  INTO TABLE lt_project
  WHERE emp_id IN s_emp_id.

SORT lt_project by emp_id.

LOOP AT lt_project INTO ls_project.
  AT NEW emp_id.
    WRITE :/ ls_project-emp_id.
  ENDAT.
ENDLOOP.
```


Output



6. At End of <FIELD_NAME >

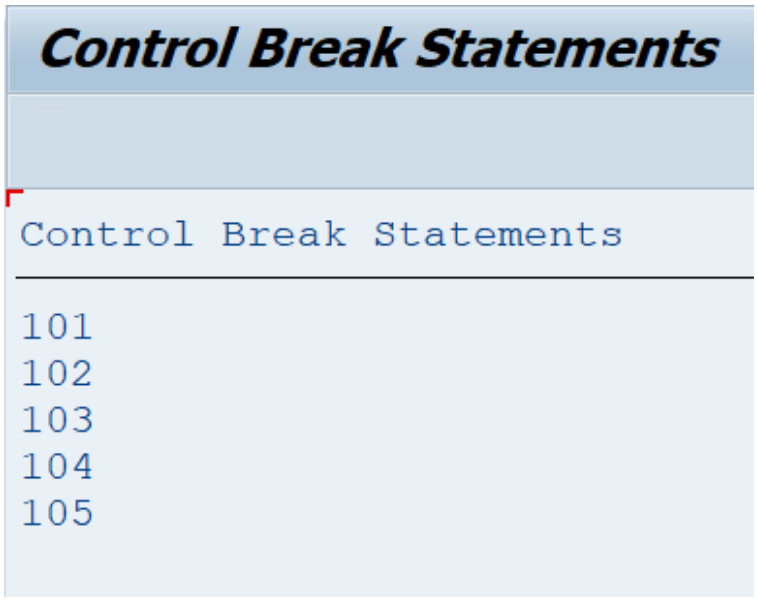
- It triggers for the last record of a group having the same values for the specified fieldname in loop.

Implementation

```
TYPES : BEGIN OF ty_project,
          emp_id       TYPE zar_emp_id,
          project_id   TYPE zar_project_id,
          project_name TYPE zar_project_name,
        END OF ty_project.

DATA : lt_project TYPE TABLE OF ty_project,
       ls_project TYPE ty_project,
       lt_prject2 TYPE TABLE OF ty_project.

DATA: lv_emp_id TYPE zar_emp_id.

SELECT-OPTIONS : s_emp_id FOR lv_emp_id.

SELECT emp_id project_id project_name
  FROM zar_project_det
  INTO TABLE lt_project
  WHERE emp_id IN s_emp_id.

SORT lt_project BY emp_id.

LOOP AT lt_project INTO ls_project.
  AT END OF emp_id.
    WRITE :/ ls_project-emp_id.
  ENDAT.
ENDLOOP.
```


Output

Control Break Statements



7️⃣

Field Symbols in ABAP Programming

1. Introduction

- Field Symbols are placeholders for the data objects (Variable, Work Area etc.).
- Field Symbols do not reserve any physical memory space, but they point to the content of the data objects.
- After declaring a field symbol, we can assign the data object to the field symbol.
- After successful assignment, whenever we address a field symbol, ultimately we are addressing the data object that is assigned to that field symbol.

Syntax : FIELD-SYMBOLS <FS>

  - In the above syntax, FIELD-SYMBOLS = keyword, where fs is the name of the field symbol.
  - The name of the field symbol is always enclosed between <>.

---

2. Types of Field Symbols

- Field symbols are of two types :-
  1. Typed Field Symbol
  2. Generic Field Symbol

---

3. Typed Field Symbol

- When we provide the data element like elementary, complex etc. to the field symbol then it is called as typed field symbol.
- IF we specify a type to the field symbol, the system checks the compatibility of the field symbol and the data object assigned to that field symbol.

Example - FIELD-SYMBOLS : <fs> TYPE i.

- In the above syntax, the field symbol is of elementary data type integer(I).

---

4. Generic Field Symbol

- When we provide the generic data types like ANY and ANY TABLE to the field symbol, then it is called as generic field symbol.
- Generic Field symbols are used for dynamic programming.

Example -

1. FIELD-SYMBOLS : <FS_STR> TYPE ANY.
2. FIELD-SYMBOLS : <FT_TAB> TYPE ANY TABLE.

---

5. Typed Field Symbol Implementation

```
DATA : lv_name(30) TYPE c VALUE 'Shivam Singh'.

*&---------------------------------------------------------------------
*&Defining field symbol
```

```
FIELD-SYMBOLS : <fs_name> TYPE c.

*&---------------------------------------------------------
*&Assigning variable to field symbol.
ASSIGN lv_name TO <fs_name>.
IF <fs_name> IS ASSIGNED.
  <fs_name> = 'Amrit Raj'.
  WRITE :/ lv_name.
ENDIF.
```

Output

```
Amrit Raj
```

- As soon as we will change the value of field symbol, by default the value of data object will change.

---

6. Field Symbol as a Replacement of Work Area

- We can replace work area by field symbol while performing internal table operations.
- This is because work area stores a copy of the internal table row, whereas field symbol directly references the internal table row.
- Hence, processing of internal table with field symbol is faster than the processing of internal table with work area.

Requirement :-

- We will fetch out the records from the employee table and this time we will use field symbols to display those records on the output screen.

Implementation :-

```
TYPES : BEGIN OF ty_employee,
          emp_id     TYPE ZAR_emp_id,
          emp_name   TYPE ZAR_emp_name,
          department TYPE zar_department,
          manager    TYPE zar_manager,
        END OF ty_employee.

DATA : lt_employee TYPE TABLE OF ty_employee,
       ls_employee TYPE ty_employee.

FIELD-SYMBOLS : <fs_employee> TYPE ty_employee.

DATA :  lv_Emp_id TYPE zar_emp_id.
SELECT-OPTIONS : s_emp_id FOR lv_emp_id.


SELECT emp_id emp_name department manager
  FROM zar_emp_tab
  INTO TABLE lt_employee
  WHERE emp_id IN s_emp_id.

LOOP AT lt_employee ASSIGNING <fs_employee>.
  IF <fs_employee> IS ASSIGNED.
    WRITE :/ <fs_employee>-emp_id, <fs_employee>-emp_name, <f
  ENDIF.
ENDLOOP.
```


Output

```
101 SHIVAM SINGH                    SAP ABAP                    AMRIT RAJ
103 ABHISHEK YADAV                  SAP ABAP                    AMRIT RAJ
104 MUDITA GUPTA                    SAP ABAP                    AMRIT RAJ
105 SHIKHAR SRIVASTAV               SAP ABAP                    AMRIT RAJ
102 SHREYASHI TRIPATHI              SAP ABAP                    AMRIT RAJ
106 ATUL KUMAR                      SAP ABAP                    AMRIT RAJ
107 BUDDHI VISHWAS                  SAP ABAP                    AMRIT RAJ
```