SAP BW ABAP

Author : Sanjay Chaurasia

Date: 19/06/2018

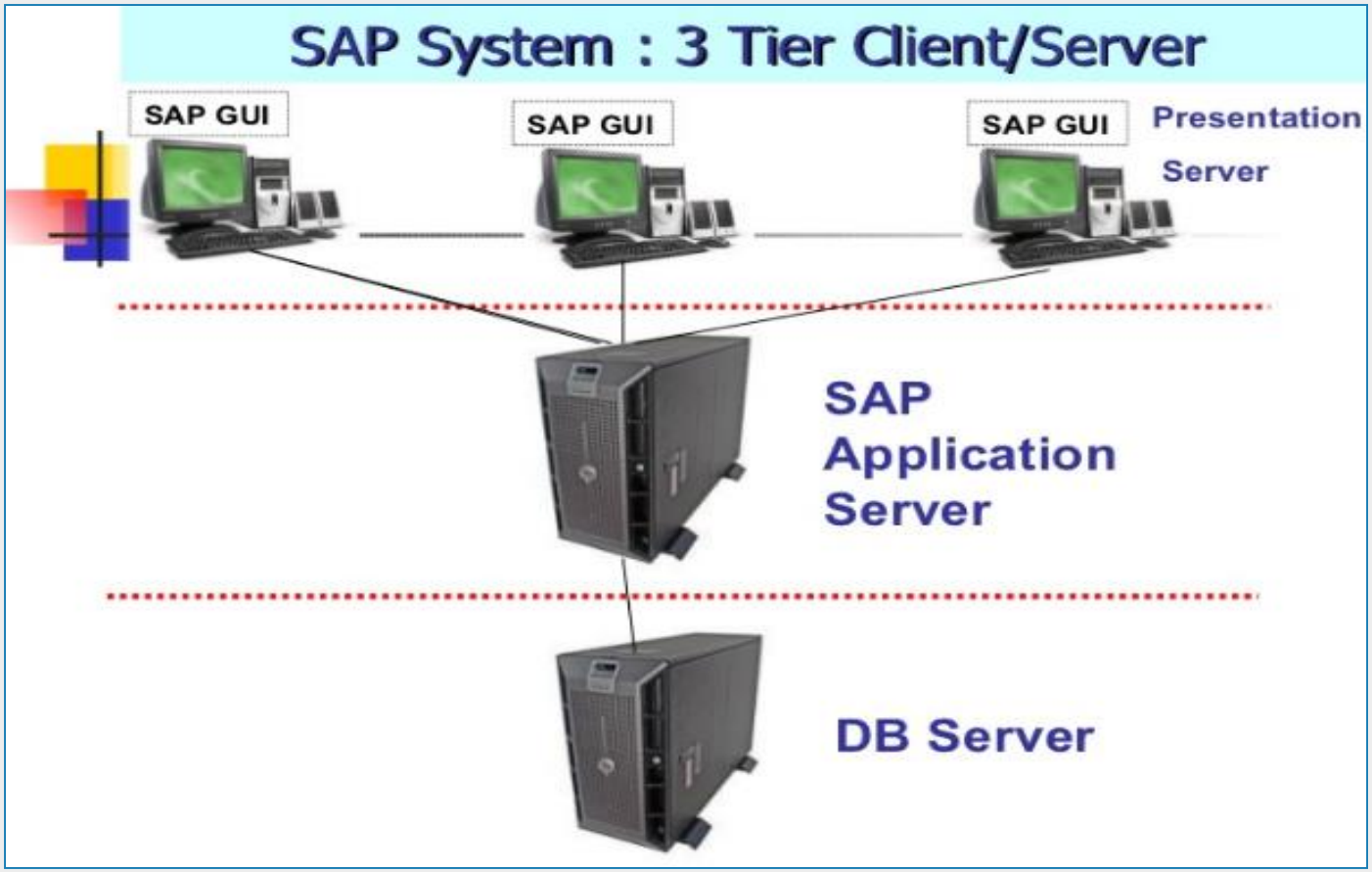
Capgemini

ABAP Fundamentals/Overview

ABAP stands for Advanced Business Application Programming, a 4GL (4th generation) language.
Currently it is positioned, along with Java, as the main language for SAP application server programming.

Below is the example of 3 Tier Client/Server Architecture

- Presentation Layer - The Presentation layer consists of any input device that can be used to control SAP system.
- Application Server - All the central processing takes place in Application server.
- DB server - This is the layer where data actually gets stored.



Introduction to ABAP Programming & Core ABAP Topics

Data Types

ABAP offers the programmer a rich assortment of fixed length as well as variable length data types. Following table lists down ABAP elementary data types −

Type	Keyword
Byte field	X
Text field	C
Integer	I
Floating point	F
Packed number	P
Text string	STRING

Some of the fields and numbers can be modified using one or more names as the following −

- byte
- numeric
- character-like

Data Types

The following table shows the data type, how much memory it takes to store the value in memory, and the minimum and maximum value that could be stored in such type of variables

Type	Typical Length	Typical Range
X	1 byte	Any byte values (00 to FF)
C	1 character	1 to 65535
N (numeric text filed)	1 character	1 to 65535
D (character-like date)	8 characters	8 characters
T (character-like time)	6 characters	6 characters
I	4 bytes	-2147483648 to 2147483647
F	8 bytes	2.2250738585072014E-308 to 1.7976931348623157E+308 positive or negative
P	8 bytes	[-10^(2len -1) +1] to [+10^(2len -1) 1] (where len = fixed length)
STRING	Variable	Any alphanumeric characters
XSTRING (byte string)	Variable	Any byte values (00 to FF)

Data Types

Example:

```
REPORT YR_SEP_12.

DATA text_line TYPE C LENGTH 40.
text_line = 'A Chapter on Data Types'.
Write text_line.

DATA text_string TYPE STRING.
text_string = 'A Program in ABAP'.
Write / text_string.

DATA d_date TYPE D.
d_date = SY-DATUM.
Write / d_date.
```

In this example, we have a character string of type C with a predefined length 40. STRING is a data type that can be used for any character string of variable length (text strings). Type STRING data objects should generally be used for character-like content where fixed length is not important.

Constants

Constants :
Constants are used to store a value under a name. We must specify the value when we declare a constant and the value cannot be changed later in the program.

Use **CONSTANTS** keyword to declare a constant.

```
CONSTANTS: pi TYPE p DECIMALS 2 VALUE '3.14',
                    yes TYPE c VALUE 'X'.
```

Variables & it’s declaration

Variables : ABAP Variables are instances of data types. Variables are created during program execution and destroyed after program execution.

Use keyword **DATA** to declare a variable.

```
DATA: firstname(10) TYPE c,
          index TYPE i,
         student_id(5) TYPE n.
```

```
DATA: firstname(10) TYPE c,
          lastname LIKE firstname. " Observe LIKE keyword
```

Variables & It’s Categories

•ABAP has predefined variables which are called system variables and are accessible from all ABAP programs.
•These fields are actually filled by the run-time environment.
•The values in these fields indicate the state of the system at any given point of time.
•You can find the complete list of system variables in the SYST table in SAP.
•Individual fields of the SYST structure can be accessed by using either “SYST-” or “SY-”.

Example:

```
REPORT Z_Test123_01.

WRITE:/'SY-ABCDE', SY-ABCDE,
/'SY-DATUM', SY-DATUM,
/'SY-DBSYS', SY-DBSYS,
/'SY-HOST ', SY-HOST,
/'SY-LANGU', SY-LANGU,
/'SY-MANDT', SY-MANDT,
/'SY-OPSYS', SY-OPSYS,
/'SY-SAPRL', SY-SAPRL,
/'SY-SYSID', SY-SYSID,
/'SY-TCODE', SY-TCODE,
/'SY-UNAME', SY-UNAME,
/'SY-UZEIT', SY-UZEIT.
```

Strings

Strings, which are widely used in ABAP programming, are a sequence of characters.

We use data type C variables for holding alphanumeric characters, with a minimum of 1 character and a maximum of 65,535 characters. By default, these are aligned to the left.

Creating Strings :
The following declaration and initialization creates a string consisting of the word 'Hello'. The size of the string is exactly the number of Characters in the word 'Hello‘.

Following program is an example of creating strings.

```
REPORT YT_SEP_15.
DATA my_Char (5)VALUE 'Hello'.
Write my_Char.
```

ABAP supports a wide range of statements that manipulate strings

S.No.	Statement & Purpose
1	**CONCATENATE** Two strings are joined to form a third string.
2	**CONDENSE** This statement deletes the space characters.
3	**STRLEN** Used to find the length of a field.
4	**REPLACE** Used to make replacements in characters.
5	**SEARCH** To run searches in character strings.
6	**SHIFT** Used to move the contents of a string left or right.
7	**SPLIT** Used to split the contents of a field into two or more fields.

Operators

ABAP provides a rich set of operators to manipulate variables. All ABAP operators are classified into four categories −

- Arithmetic Operators
- Comparison Operators
- Bitwise Operators
- Character String Operators

Arithmetic Operators

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra.
The following list describes arithmetic operators. Assume integer variable A holds 20 and variable B holds 40.

S.No.	Arithmetic Operator & Description
1	**+ (Addition)**
	Adds values on either side of the operator. Example: A + B will give 60.
2	**− (Subtraction)**
	Subtracts right hand operand from left hand operand. Example: A − B will give -20.
3	*** (Multiplication)**
	Multiplies values on either side of the operator. Example: A * B will give 800.
4	**/ (Division)**
	Divides left hand operand by right hand operand. Example: B / A will give 2.
5	**MOD (Modulus)**
	Divides left hand operand by right hand operand and returns the remainder. Example: B MOD A will give 0.

Example:

```
REPORT YS_SEP_08.

DATA: A TYPE I VALUE 150,
      B TYPE I VALUE 50,
      Result TYPE I.

Result = A / B.

WRITE / Result.
```

Operators

Comparison Operators : Let's discuss the various types of comparison operators for different operands.

S.No.	Comparison Operator & Description
1	**= (equality test). Alternate form is EQ.** Checks if the values of two operands are equal or not, if yes then condition becomes true. Example (A = B) is true.
2	**<> (Inequality test). Alternate form is NE.** Checks if the values of two operands are equal or not. If the values are not equal then the condition becomes true. Example (A <> B) is true.
3	**> (Greater than test). Alternate form is GT.** Checks if the value of left operand is greater than the value of right operand. If yes then condition becomes true. Example (A > B) is not true.
4	**< (Less than test). Alternate form is LT.** Checks if the value of left operand is less than the value of right operand. If yes, then condition becomes true. Example (A < B) is true.
5	**>= (Greater than or equals) Alternate form is GE.** Checks if the value of left operand is greater than or equal to the value of right Operand. If yes, then condition becomes true. Example (A >= B) is not true.
6	**<= (Less than or equals test). Alternate form is LE.** Checks if the value of left operand is less than or equal to the value of right operand. If yes, then condition becomes true. Example (A <= B) is true.
7	**a1 BETWEEN a2 AND a3 (Interval test)** Checks whether a1 lies in between a2 and a3 (inclusive). If yes, then the condition becomes true. Example (A BETWEEN B AND C) is true.
8	**IS INITIAL** The condition becomes true if the contents of the variable have not changed and it has been automatically assigned its initial value. Example (A IS INITIAL) is not true
9	**IS NOT INITIAL** The condition becomes true if the contents of the variable have changed. Example (A IS NOT INITIAL) is true.

Operators

Comparison Operators :

Note : If the data type or length of the variables does not match then automatic conversion is performed. Automatic type adjustment is performed for either one or both of the values while comparing two values of different data types. The conversion type is decided by the data type and the preference order of the data type.

Following is the order of preference −

•If one field is of type I, then the other is converted to type I.
•If one field is of type P, then the other is converted to type P.
•If one field is of type D, then the other is converted to type D. But C and N types are not converted and they are compared directly. Similar is the case with type T.
•If one field is of type N and the other is of type C or X, both the fields are converted to type P.
•If one field is of type C and the other is of type X, the X type is converted to type C.

Example 1:

```
REPORT YS_SEP_08.

DATA: A TYPE I VALUE 115,
          B TYPE I VALUE 119.

IF A LT B.
    WRITE: / 'A is less than B'.
ENDIF
```

Example 2:

```
REPORT YS_SEP_08.

DATA: A TYPE I.

IF A IS INITIAL.
   WRITE: / 'A is assigned'.
ENDIF.
```

Operators

Bitwise Operators : ABAP also provides a series of bitwise logical operators that can be used to build Boolean algebraic expressions. The bitwise operators can be combined in complex expressions using parentheses and so on.

S.No.	Bitwise Operator & Description
1	**BIT-NOT** Unary operator that flips all the bits in a hexadecimal number to the opposite value. For instance, applying this operator to a hexadecimal number having the bit level value 10101010 (e.g. 'AA') would give 01010101.
2	**BIT-AND** This binary operator compares each field bit by bit using the Boolean AND operator.
3	**BIT-XOR** Binary operator that compares each field bit by bit using the Boolean XOR (exclusive OR) operator.
4	**BIT-OR** Binary operator that compares each field bit by bit using the Boolean OR operator.

For example, following is the truth table that shows the values generated when applying the Boolean AND, OR, or XOR operators against the two bit values contained in field A and field B.

Field A	Field B	AND	OR	XOR
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Operators

Character String Operators: Following is a list of character string operators −

S.No.	Character String Operator & Description
1	**CO (Contains Only)** Checks whether A is solely composed of the characters in B.
2	**CN (Not Contains ONLY)** Checks whether A contains characters that are not in B.
3	**CA (Contains ANY)** Checks whether A contains at least one character of B.
4	**NA (NOT Contains Any)** Checks whether A does not contain any character of B.
5	**CS (Contains a String)** Checks whether A contains the character string B.
6	**NS (NOT Contains a String)** Checks whether A does not contain the character string B.
7	**CP (Contains a Pattern)** It checks whether A contains the pattern in B.
8	**NP (NOT Contains a Pattern)** It checks whether A does not contain the pattern in B.

Example:

```
REPORT YS_SEP_08.

DATA: P(10) TYPE C VALUE 'APPLE',
      Q(10) TYPE C VALUE 'CHAIR'.

IF P CA Q.
   WRITE: / 'P contains at least one
character of Q'.
ENDIF.
```

Control Statements (Decision)

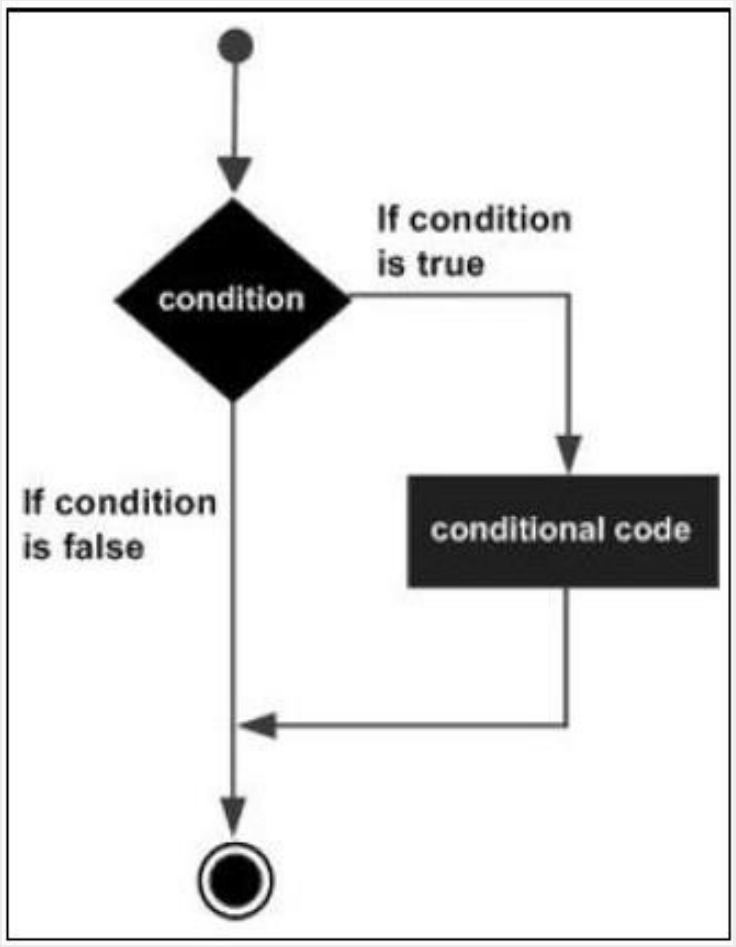
Control Statements or Decision making structures have one or more conditions to be evaluated or tested by the program, along with a statement or statements that are to be executed, if the condition is determined to be true, and optionally, other statements to be executed, if the condition is determined to be false

ABAP programming language provides the following types of decision-making statements.

S.No.	Statement & Description
1	IF Statement An IF statement consists of a logical expression followed by one or more statements.
2	IF.. Else Statement An IF statement can be followed by an optional ELSE statement that executes when the expression is false.
3	Nested IF Statement You may use one IF or ELSEIF statement inside another IF or ELSEIF statement.
4	CASE Control Statement CASE statement is used when we need to compare two or more fields or variables.

If Statement

‘IF’ is a control statement used to specify one or more conditions.
If the expression evaluates to true, then the IF block of code will be executed.

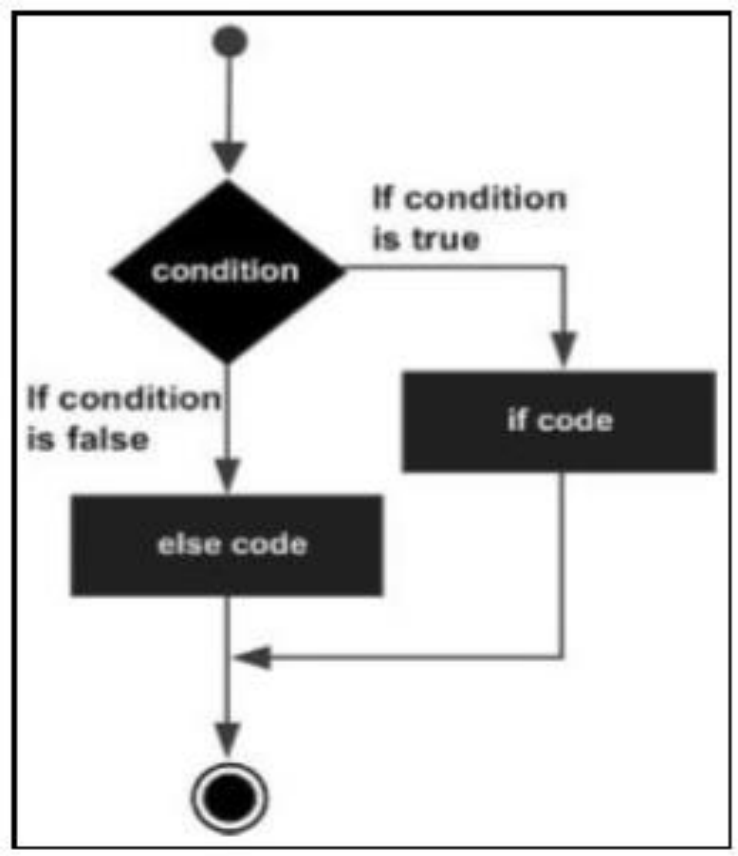


Example :

```
Report YH_SEP_15.

 Data Title_1(20) TYPE C.

Title_1 = 'Tutorials'.

 IF Title_1 = 'Tutorials'.
    write 'This is IF statement'.
ENDIF.
```

If .. Else Statement

In case of IF….ELSE statements, if the expression evaluates to true then the IF block of code will be executed. Otherwise, ELSE block of code will be executed.



Example :

```
Report YH_SEP_15.

Data Title_1(20) TYPE C.
       Title_1 = 'Tutorials'.

IF Title_1 = 'Tutorial'.
    write 'This is IF Statement'.
ELSE.
    write 'This is ELSE Statement'.
ENDIF.
```

Nested If Statement

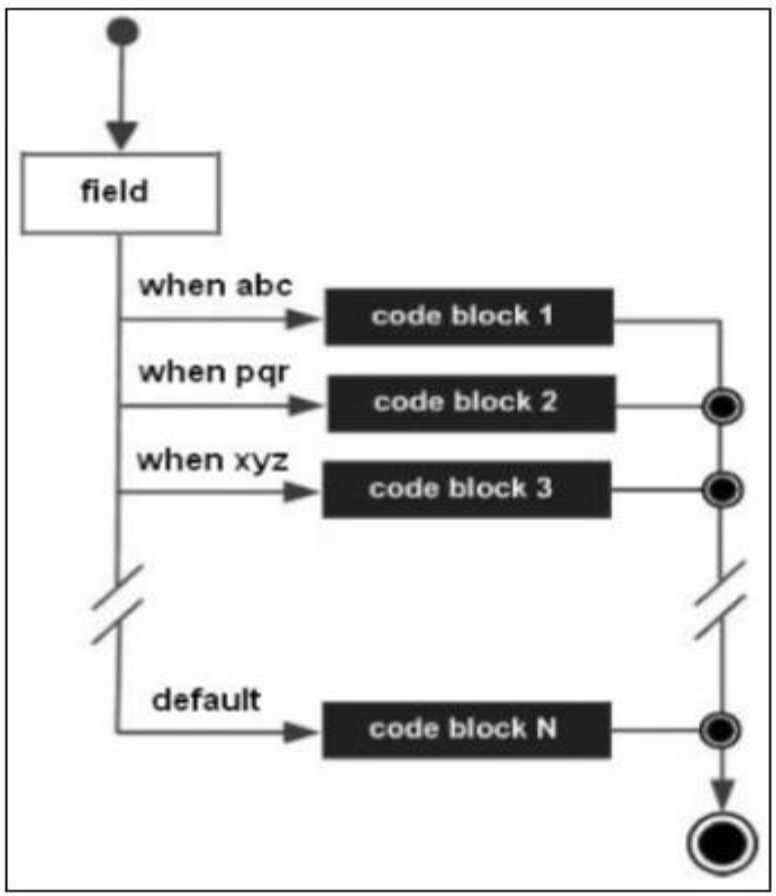
It is always legal to nest IF....ELSE statements, which means you can use one IF or ELSEIF statement inside another IF or ELSEIF statement.

Example

```
Report YH_SEP_15.

Data: Title_1(10) TYPE C,
Title_2(15) TYPE C,
Title_3(10) TYPE C.
Title_1 = 'ABAP'.
Title_2 = 'Programming'.
Title_3 = 'Tutorial'.

IF Title_1 = 'ABAP'.

   IF Title_2 = 'Programming'.

      IF Title_3 = 'Tutorial'.
         Write 'Yes, It's Correct'.
      ELSE.
         Write 'Sorry, It's Wrong'.
     ENDIF.

  ENDIF.

ENDIF.
```

Case Control Statement

The CASE control statement is used when you need to compare two or more fields.



Example :

```
Report YH_SEP_15.

Data: Title_1(10) TYPE C,
      Title_2(15) TYPE C.

Title_1 = 'ABAP'.
Title_2 = 'Programming'.

CASE Title_2.

WHEN 'ABAP'.
   Write 'This is not the title'.

WHEN 'Tutorials'.
   Write 'This is not the title'.

WHEN 'Limited'.
   Write 'This is not the title'.

WHEN 'Programming'.
  Write 'Yes, this is the title'.

WHEN OTHERS.
 Write 'Sorry, Mismatch'.

ENDCASE.
```

Loop Control

Loop Control: A **loop statement** allows us to execute a statement or group of statements multiple times.

ABAP programming language provides the following types of loop to handle looping requirements.

S.No.	Loop Type & Description
1	WHILE loop Repeats a statement or group of statements when a given condition is true. It tests the condition before executing the loop body.
2	Do loop The DO statement is useful for repeating particular task a specific number of times.
3	Nested loop You may use one or more loops inside any another WHILE or DO loop.

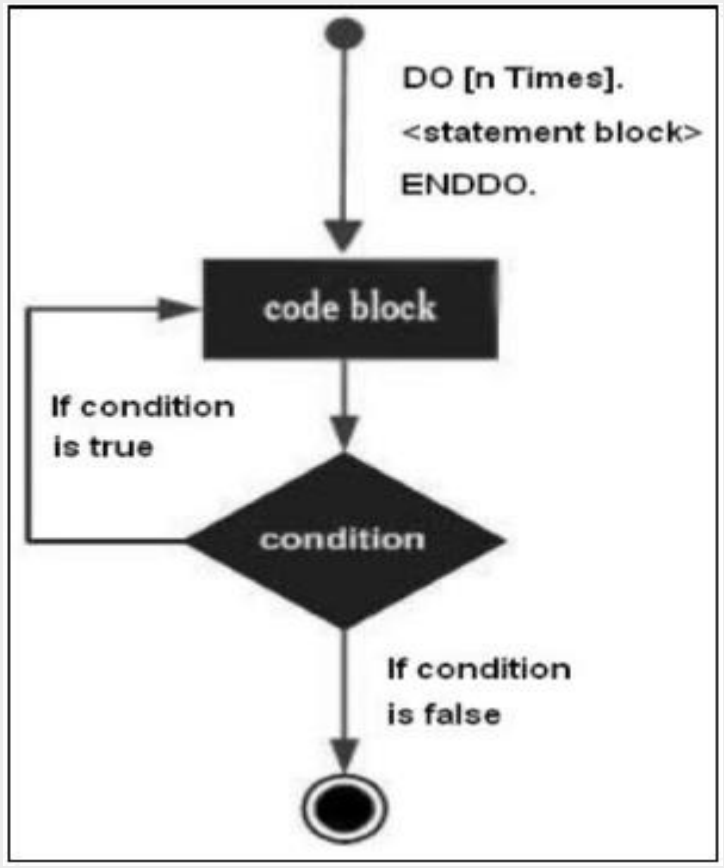
Loop Control Statements : Loop control statements change execution from its normal sequence. ABAP includes control statements that allow loops to be ended prematurely. It supports the following control statements.

S.No.	Control Statement & Description
1	CONTINUE Causes the loop to skip the remainder of its body and starts the next loop pass.
2	CHECK If the condition is false, then the remaining statements after the CHECK are just ignored and the system starts the next loop pass.
3	EXIT Terminates the loop entirely and transfers execution to the statement immediately following the loop.

Do Loops

The **DO** statement implements unconditional loops by executing a set of statement blocks several times unconditionally.

'Times' in below diagram imposes a restriction on the number of loop passes, which is represented by 'n'. The value of 'n' should not be negative or zero. If it is zero or negative, the statements in the loop are not executed.



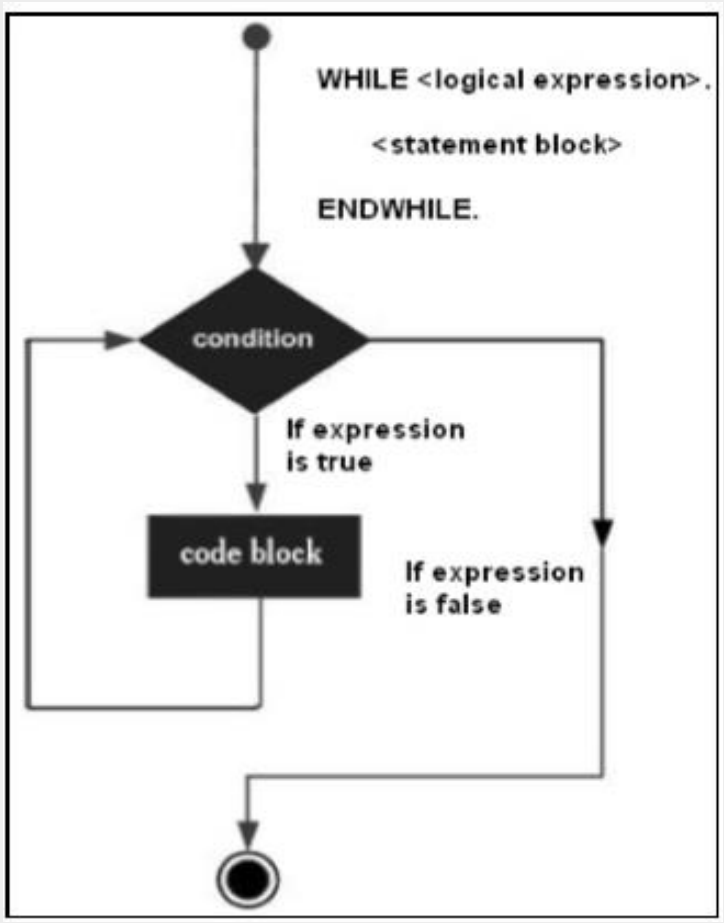
Example:

```
Report YH_SEP_15.

Do 15 TIMES.
   Write: / 'Hello'.
ENDDO.
```

While loop

The **WHILE loop** executes the statements enclosed by the WHILE and ENDWHILE commands until the logical expression becomes false.

The WHILE command is preferable while considering the performance of programs. The loop continues until the logical statement is found to be untrue and exits the loop if a false statement is found, and the first statement after the WHILE loop is executed.



Example :

```
REPORT YS_SEP_15
.
DATA: a type i.
a = 0.

WHILE a <> 8.
    Write: / 'This is the line:', a.
     a = a + 1.
ENDWHILE.
```

Nested Loop

The DO and WHILE statements can be nested as well as combined with other loop forms. Each nested loop will have it’s own SY-INDEX created and monitored by the system.

Example

```
REPORT YS_SEP_15.

Data: a1 type I,
        b1 type I.

a1 = 0.
b1 = 0.

Do 2 times.
    a1 = a1 + 1.
   Write: /'Outer', a1.

   Do 10 times.
        b1 = b1 + 1.
        Write: /'Inner', b1.
   ENDDo.

ENDDo.
```

Date Operations

ABAP implicitly references the Gregorian calendar, valid across most of the world. We can convert the output to country specific calendars. A date is a time specified to a precise day, week or month with respect to a calendar. A time is specified to a precise second or minute with respect to a day. ABAP always saves time in 24-hour format. The output can have a country specific format. Dates and time are usually interpreted as local dates that are valid in the current time zone.

ABAP provides two built-in types to work with dates and time −

D data type
T data type
Following is the basic format

```
DATA: date TYPE D,
      time TYPE T.

DATA: year TYPE I,
month TYPE I,
day TYPE I,
hour TYPE I,
minute TYPE I,
second TYPE I.
```

The following code snippets retrieve the current system date and time.

```
REPORT YR_SEP_15.
DATA: date_1 TYPE D.

date_1 = SY-DATUM.
Write: / 'Present Date is:', date_1
DD/MM/YYYY.

date_1 = date_1 + 06.
Write: / 'Date after 6 Days is:', date_1
DD/MM/YYYY.
```

Internal Table concept and Usage

Internal tables provide a means of taking data from a fixed structure and storing it in working memory in ABAP.
The data is stored line by line in memory, and each line has the same structure.
In ABAP, internal tables fulfill the function of arrays.

Internal tables are used to obtain data from a fixed structure for dynamic use in ABAP.
Each line in the internal table has the same field structure.
The main use for internal tables is for storing and formatting data from a database table within a program.

Database tables concept

Tables can be defined independent of the database in ABAP Dictionary. When a table is activated in ABAP Dictionary, similar copy of its fields is created in the database as well. The tables defined in ABAP Dictionary are translated automatically into the format that is compatible with the database because the definition of the table depends on the database used by the SAP system.

A table can contain one or more fields, each defined with its data type and length. The large amount of data stored in a table is distributed among the several fields defined in the table

A table consists of many fields, and each field contains many elements. The following table lists the different elements of table fields −

S.No.	Elements & Description
1	**Field name** This is the name given to a field that can contain a maximum of 16 characters. The field name may be composed of digits, letters, and underscores. It must begin with a letter.
2	**Key flag** Determines whether or not a field belongs to a key field.
3	**Field type** Assigns a data type to a field.
4	**Field length** The number of characters that can be entered in a field.
5	**Decimal places** Defines the number of digits permissible after the decimal point. This element is used only for numeric data types.
6	**Short text** Describes the meaning of the corresponding field.

Viewing of database Tables and selection functions

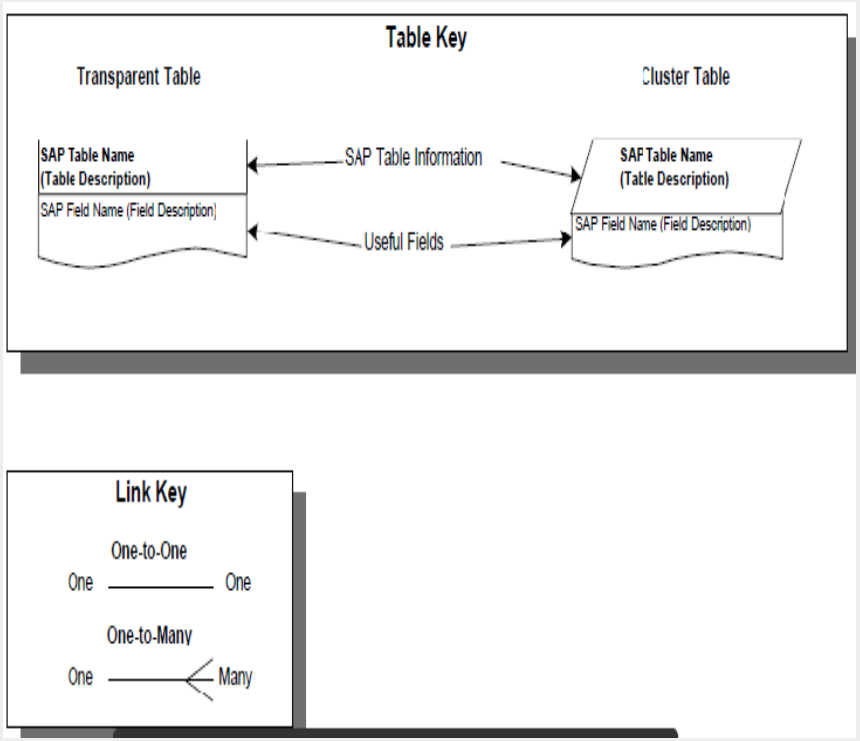
Database tables can be viewed by tcodes SE11 and SE16.

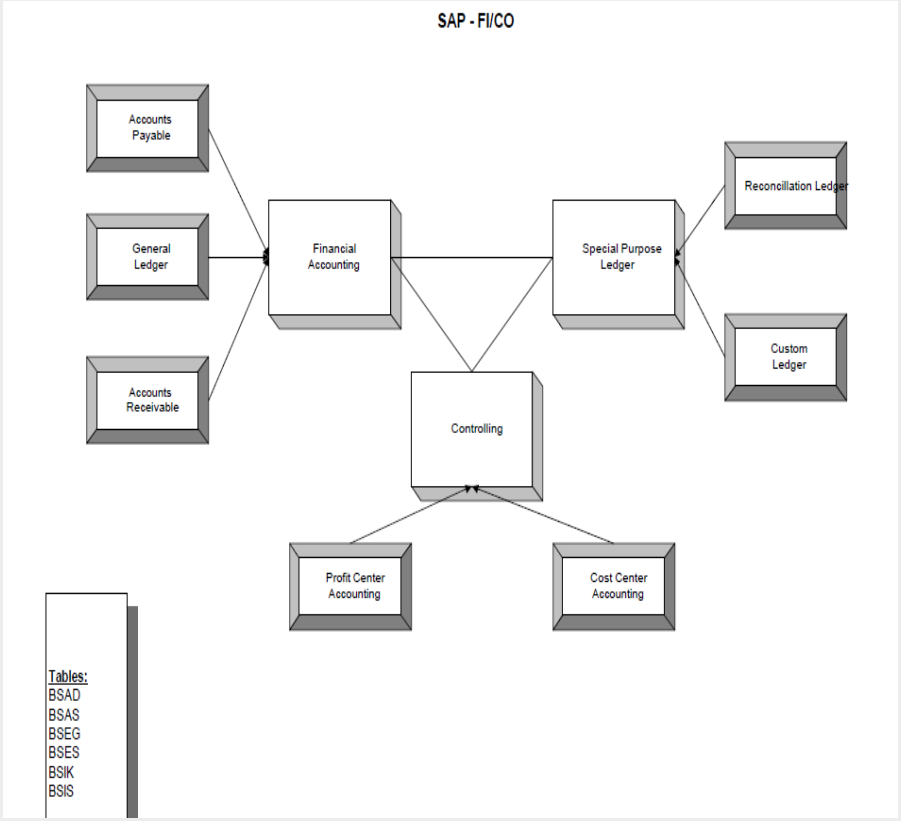
SE11 – is ABAP dictionary where table can be maintained and fields can be added/deleted from the table as well as other operations can be performed with the table. It also helps programmers/user to view data.

SE16 - it is like a data browser to see the data of the table.

ECC Database Table & its Relationship

In ECC which is generally source system to BW systems, different modules such as FI/CO, MM, PP, SD etc. are linked by means of database tables via primary and foreign key relationships.





Structures

ABAP structures can be identified and created in two ways.

The first way is to create from SE11 transaction

Second way is simply define a structure directly in-line in your program using data/type: begin of or to create a data dictionary structure globally visible to all other ABAP reports.

```
Types: Begin of T_VBAP,
            vbeln type vbeln,
            posnr type posnr,
            matnr type matnr,
            netwr type netwr,
       end of T_VBAP.
```

In this training we will look into second way of structure creation as we would need them in internal tables topic

Internal Table

An **Internal table** is a temporary table gets created in the memory of application server during program execution and gets destroyed once the program ends. It is used to hold data temporarily or manipulate the data. It contains one or more rows with same structure.

An internal table can be defined using the keyword TABLE OF in the DATA statement. Internal table can be defined by the following ways.

```
Types: Begin of T_VBAP,
            vbeln type vbeln,
            posnr type posnr,
            matnr type matnr,
            netwr type netwr,
       end of T_VBAP.

data : IT_VBAP_S type STANDARD TABLE OF T_VBAP,
       wa_vbap_S type T_VBAP.
```

Types of Internal Table

There are three types of internal tables in SAP ABAP programming.

Standard Internal Tables
Sorted Internal Tables
Hashed Internal Tables

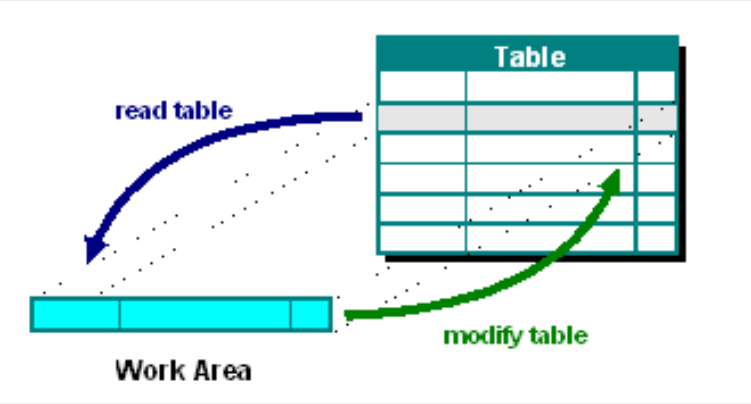
1.Standard Internal Tables: These tables have a linear index and can be accessed using the index or the key. The response time is in linear relationship with number of table entries. These tables are useful when user wants to address individual table entries using the index.

2.Sorted Internal Tables: These tables also have an index and the key. But, the response time is in logarithmic relationship with number of table entries, since it uses binary search algorithm instead of linear search. These tables are useful when user wants the table to be sorted while additional entries have to be added.

3.Hashed Internal Tables: These tables have no index, but have the key. The response time is constant irrespective of number of table entries, since it uses a Hash algorithm. These tables are useful when user wants to access the entries with key only.

Work Area

Work area is a variable declared with the TYPE of an internal table or a database table. It can store only one record at a time. It is like a structure declaration in C.



Select Statement

SELECT is the Open SQL statement for reading data from one or more database tables into data objects.

The select statement reads a result set (whose structure is determined in **result**) from the database tables specified in **source**, and assigns the data from the result set to the data objects specified in **target**. You can restrict the result set using the **WHERE** addition.

Below are different types of select statements used in SAP ABAP programming to read data from database table.

Step1 : Using Select * in SAP ABAP
Step2 : Using Select Single in SAP ABAP
Step3 : Using Select Max in SAP ABAP
Step4 : Using Select Min in SAP ABAP
Step5 : Using Select UP TO in SAP ABAP
Step6 : Using Select Distinct in SAP ABAP
Step6 : Using Select FOR ALL ENTRIES in SAP ABAP

```
Example.

Data : vbeln type vbap-vbeln,
         posnr type vbap-posnr,
        matnr type vbap-matnr.

write :/ '**********Output using variables**********'.

SELECT vbeln posnr matnr
      into (vbeln, posnr, matnr)
      from vbap
      up to 10 rows.

     write :/ vbeln, posnr, matnr.

ENDSELECT.
```

Read Statement

READ statement is used to read the lines of internal table
We can read the lines of a table by using the following syntax of the READ TABLE statement −

```
READ TABLE <internal_table> FROM <work_area_itab>.
```

In this syntax, the <work_area_itab> expression represents a work area that is compatible with the line type of the <internal_table> table. We can specify a search key, but not a table key, within the READ statement by using the WITH KEY clause, as shown in the following syntax −

```
READ TABLE <internal_table> WITH KEY = <internal_tab_field>.
```

Here the entire line of the internal table is used as a **search key**. The content of the entire line of the table is compared with the content of the <internal_tab_field> field.

```
READ TABLE <internal_table> <key> INTO <work_area_itab> [COMPARING <F1> <F2>...<Fn>].
```

When the COMPARING clause is used, the specified table fields <F1>, <F2>....<Fn> of the structured line type are compared with the corresponding fields of the work area before being transported.

```
REPORT ZREAD_DEMO.
 */Creating an internal table
DATA: BEGIN OF Record1,
              ColP TYPE I,
              ColQ TYPE I,
         END OF Record1.
DATA mytable LIKE HASHED TABLE OF Record1 WITH UNIQUE KEY ColP.
DO 6 Times.
         Record1-ColP = SY-INDEX.
        Record1-ColQ = SY-INDEX + 5.
        INSERT Record1 INTO TABLE mytable.
ENDDO.
       Record1-ColP = 4.
       Record1-ColQ = 12.
READ TABLE mytable FROM Record1 INTO Record1 COMPARING ColQ.
       WRITE: 'SY-SUBRC =', SY-SUBRC.
       SKIP.
      WRITE: / Record1-ColP, Record1-ColQ.
```

Other Important ABAP Statements/Keywords

INSERT → Adds a new record to the table. If the row (key) exists, issues an error.

UPDATE → Updates an existing record to the table. If the row (key) does not exist, issues an error.

MODIFY → If the key exists, modifies the record. If the key does not exist, adds the record to the table.

MODIFY….TRANSPORTING → TRANSPORTING Statement tells ABAP that only specific fields will be modified which are given after the TRANSPORTING clause.

APPEND → This adds a new record to the internal table in the last position.

COLLECT → The table is first checked for an entry of the key-fields in the table with the comparison of the coming entry. If the key-fields' entry is present, this adds up all the numeric fields of the record with the existing record. If the record is totally new, then it is appended to the table.

DELETE → is used to delete one or more records from an internal table. The records of an internal table are deleted either by specifying a table key or condition or by finding duplicate entries.

DELETE ADJACENT DUPLICATE → This statement works logically on sorted standard table and sorted table with non-unique key. It compares the adjacent rows. If similar records are found based on the comparing fields then it deletes from the second records onward. The system keeps only the first record.

MOVE-CORRESPONDING → The statement MOVE-CORRESPONDING is used to assign components with the same name in structured data objects to each other.

CLEAR → Clears the contents of the variable.
Variables may be internal table , work area , variables declared using elementary data types etc.

REFRESH → Clears the content of internal table only.

FREE → Clears the contents and releases the memory of internal table. This keyword is applicable only in case of internal table

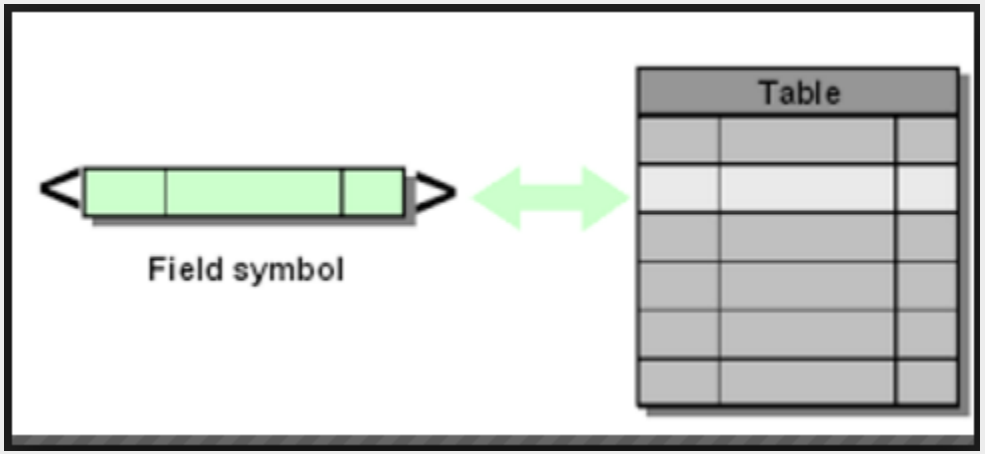
Field Symbols

Field symbols are similar to pointers in C language, field symbols does not have any memory instead they will be pointing to a memory location. The concept of field symbols is very important in order to increase the performance of SAP applications, but unethical use of field-symbols leads to application issues.

Field symbol name should always be within <>.

Example:.

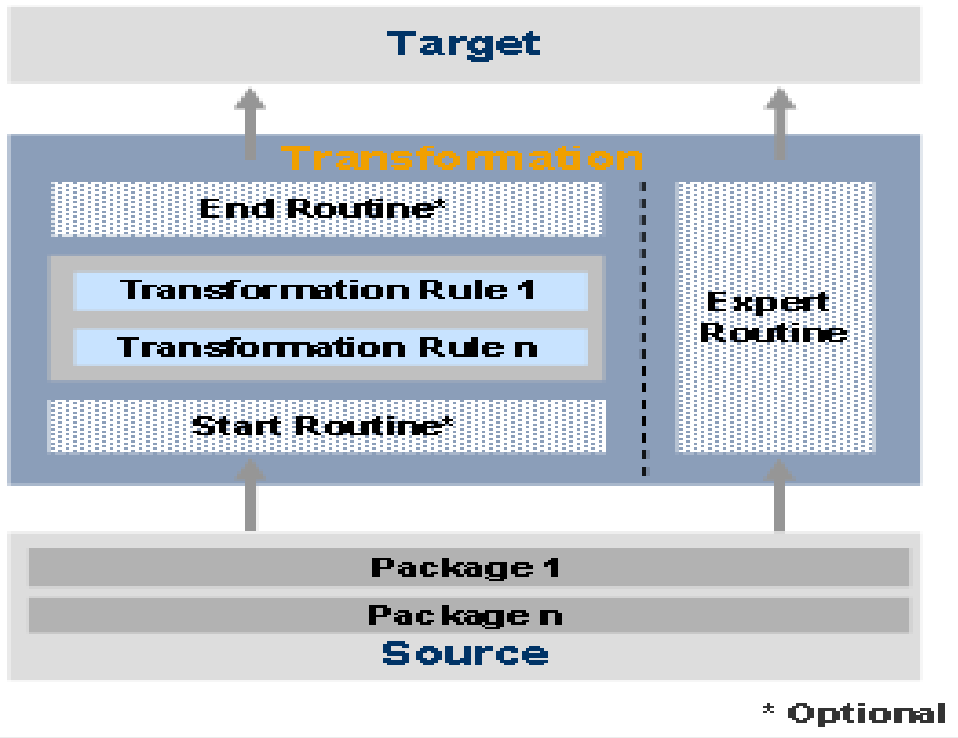
```
FIELD-SYMBOLS : <FIELD_SYMBOL> TYPE MARA-MATNR. "here MARA-MATNR is a variable type
FIELD-SYMBOLS : <FIELD_SYMBOL> TYPE MARA. "here MARA is a structure
FIELD-SYMBOLS : <FIELD_SYMBOL> TYPE REF TO DATA . "here DATA is a reference type
```



BW transformation routines (Start, End, Expert, Field Routines)

Routines − Routines are used to perform complex transformations. Routines are defined as local ABAP classes and it consists of predefined definition and implementation area.
A Routine is created in implementation area and inbound and outbound parameters are defined in definition area.

The following graphic shows the position of these routines in the data flow:



BW transformation routines (Start, End, Expert, Field Routines)

START ROUTINE
The start routine is run at the start of the transformation. The start routine has a table in the format of the source structure as input and output parameters. It is used to perform preliminary calculations and store these in a global data structure or in a table. This structure or table can be accessed from other routines. You can modify or delete data in the source_package

END ROUTINE
An end routine is a routine with a table in the target structure format as input and output parameters. You can use an end routine to postprocess data after transformation on a package-by-package basis. Data is stored in result_package.

End Routine is processed after start routine, mappings, field routines and finally before the values is transferred to the output. End routine has the structure of the target and result_package contains the entire data which finally is the output .

FIELD ROUTINE
It operates on a single record for a single characteristic or key figure. The value gets modified in the routine based on one or more source fields before it is transferred to the target

EXPERT ROUTINE
An Expert routine is a routine with contains both the source and target structure. we can use Expert routine if there are not sufficient functions to perform transformation.

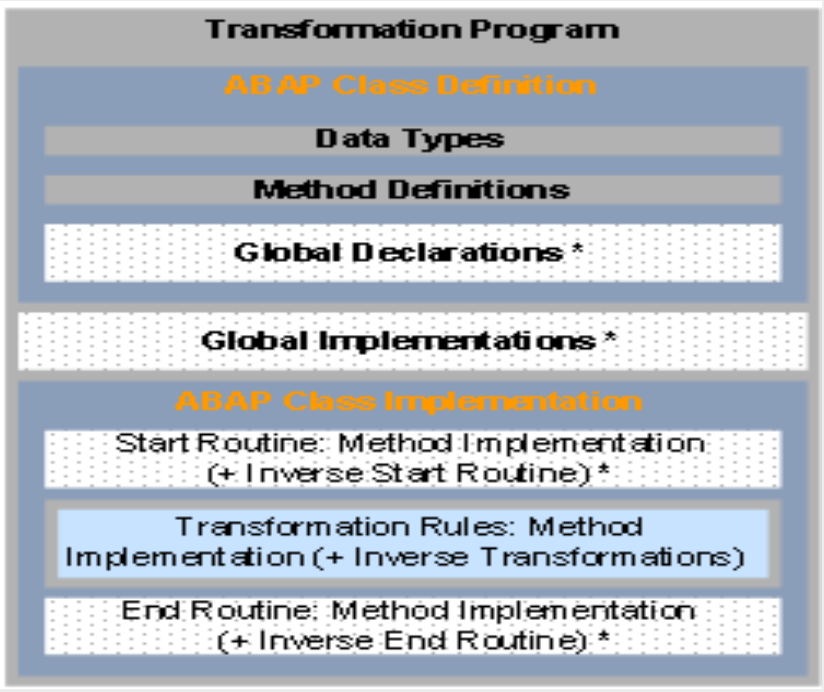
For Expert Routine every thing needs to be written using coding. In simple word an expert routine performs all the actions of Start Routine, Mappings, Field and End Routines.

BW transformation routines (Start, End, Expert, Field Routines)

The routine has a global part and a local part. In the global part you define global data declarations 'CLASS DATA'. These are available in all routines.

You can create function modules, methods or external subprograms in the ABAP Workbench if you want to reuse source code in routines. You can call these in the local part of the routine. If you want to transport a routine that includes calls of this type, the routine and the object called should be included in the same transport request.

The following figure shows the structure of the transformation program with transformation rules, start routine and end routine



The following figure shows the structure of the transformation program with expert routine

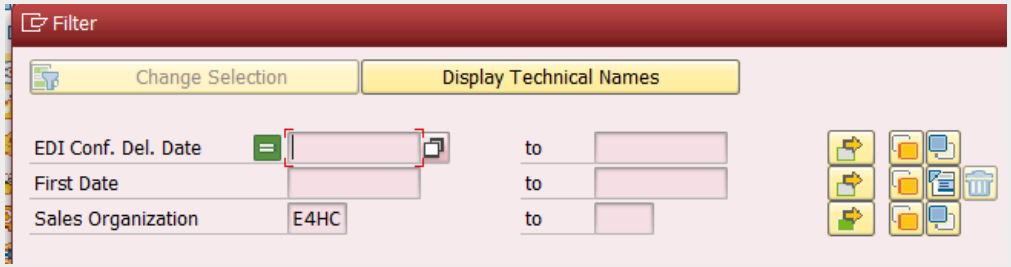
Routines in the Data transfer Process

Routines in the Data Transfer Process help us to populate value of a particular field dynamically during the DTP execution.

Example:
We have a scenario where we have two fields Valid from and Valid to in the DTP filter.
These two fields are to be populated at DTP execution dynamically with the date- last day of previous month.

This functionality can be achieved using Routines in the Data Transfer Process.

The below screenshot shows that DTP Routine is written for the field First Date to calculate current system date – 120 days.



Filter

Pattern | Pretty Printer | Routines Info.

```
  type-pools: rsarc, rsarr, rssm.
  tables: rssdlrange.
* Global code used by conversion rules
*$*$ begin of global - insert your declaration only below this line  *-*
* TABLES: ...
* DATA:   ...
*$*$ end of global - insert your declaration only before this line   *-*
* ---------------------------------------------------------------------
*     Fieldname        = /BIC/LOOTFDT
*     data type        = DATS
*     length           = 000008
* ---------------------------------------------------------------------
form c_/BIC/LOOTFDT
   tables l_t_range structure rssdlrange
   using i_r_request type ref to IF_RSBK_REQUEST_ADMINTAB_VIEW
         i_fieldnm type RSFIELDNM
  changing p_subrc like sy-subrc.
*       Insert source code to current selection field
*$*$ begin of routine - insert your code only below this line        *-*
DATA: lv_datum TYPE sy-datum.
lv_datum = sy-datum - 120.
l_t_range-sign = 'I'.
          l_t_range-option = 'GE'.
         l_t_range-iobjnm = 'LOOTFDT'.
          l_t_range-fieldname = '/BIC/LOOTFDT'.
          l_t_range-low = lv_datum.
          l_t_range-high = sy-datum.
 append l_t_range.

           p_subrc = 0.

*$*$ end of routine - insert your code only before this line         *-*
endform.
```

BADI (Business Ad-Inns): Advantages, Types and Implementation

BADI (Business Add-In) is a SAP Object Oriented enhancement technique which is used to add our own business functionality to the existing SAP standard functionality.
BADI's are available in SAP R/3 from the system release 4.6c

Advantages of BADI:
In contrast to the earlier enhancement techniques, BADI follows Object Oriented approach to make them reusable.
A BADI can be used any number of times where as standard enhancement techniques can be used only once.
For example if we assign an enhancement to one custom project, then that enhancement cannot be assigned to any other custom projects.
To overcome this drawback SAP has provided a new enhancement technique called BADI.

Transaction code for BADI Definition:
SE18
When you create a BADI definition, a class interface will be automatically created and you can define your methods in the interface. The implementation of the methods can be done in SE19 transaction.

When a BADI is created following are automatically generated:
An interface with **'IF_EX_'** inserted between the first and second characters of the BADI name
An adapter class with **'CL_EX_'** inserted between the first and second characters of the BADI name

Transaction code to Implement BADI:
SE19

BADI (Business Ad-Inns): Advantages, Types and Implementation

Types of BADI's:
There are two types of BADI's.

1) Multi use BADI:

With this option, any number of active implementations can be assigned to the same definition BADI. By default this option is checked.

If you have multiple-use BADI definitions, the sequence must not play any role.
The drawback in **Multiple use BADI** is, it is not possible to know which BADI is active especially in country specific version.

2) Filter dependent BADI:

Using this option we can define the BADI's according to the filter values to control the add-in implementation on specific criteria.
Ex: Specific country value.

BADI (Business Ad-Inns) in the BW Data source Extraction Process

Datasource Enhancement using BADI:

Extraction enhancement can be defined as supplementing additional fields to an extract structure. One part is to extend (enhance) the extract structure with one or more fields. The other part is the program logic to fill these additional fields. It can be applied in any SAP source system. Although technically possible, you will rarely or never use it in the SAP BW system itself.

Generally SAP user exit RSAP0001 is used to add any additional fields to the Standard BW Extractors which includes both Master Data and Transactions Data extractors
But as per SAP's recommendation BADI's are more efficient way to enhance the standard extractors than using the exit RSAP0001.
Enhancing a Standard DataSource using BADI has the following advantage over using the exit RSAP0001:
BADI can have multiple implementations so several developers can work simultaneously on different implementations and different logics.

RSU5_SAPI_BADI is used to enhance the standard BI DataSources.
The following methods of interface **IF_EX_RSU5_SAPI_BADI** can be implemented:

- **DATA_TRANSFORM** – Business Add-Ins Method for General Data Transfer
- **HIER_TRANSFORM** – Business Add-Ins Method for Hierarchy Data Transfer

Scenario

In the below example the Standard MM Master DataSource 2LIS_06_INV is appended with the field ZZDUE_DATE – Due Date. The due date is calculated by using function module by passing the value to the due date field.

Procedure

Enhancing the Extract Structure.
First go to transaction SBIW and expand business content DataSources and click Transfer Business Content Datasources
After identifying the datasource **2LIS_06_INV** and its extract structure **MC06M_0ITM** we need to add the new field to the structure.

Now click on "Append Structure" and create append structure and then add the required field **ZZDUE_DATE**(if there is an append structure already available, you can use the same append structure or can create a new append structure.) to the append structure and activate the Extract Structure "**MC06M_0ITM'.**

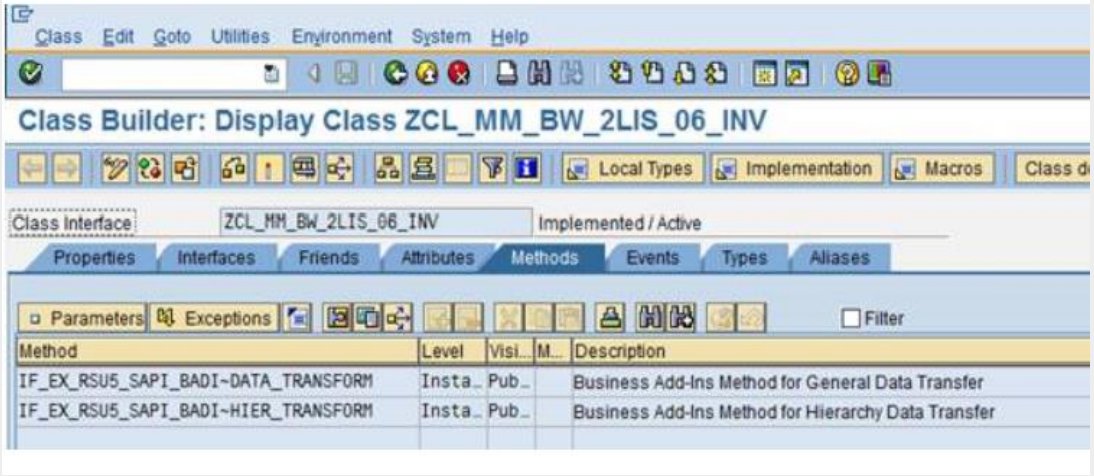
BADI (Business Ad-Inns) in the BW Data source Extraction Process

Activate the datasource 2LIS_06_INV. Verify the custom fields are added in the Extraction structure. Make sure the hide field and field only unchecked**.**

Populating Data to the enhanced field:

Go to SE24 and create a class "ZCL_MM_BW_2LIS_06_INV" (Note: Use appropriate name that way you can easily identify the datasource) and in the Interfaces tab of the class enter "IF_EX_RSU5_SAPI_BADI

Now 2 methods "DATA_TRANSFORM" and "HIER_TRANSFORM" will be available in the Method tab.

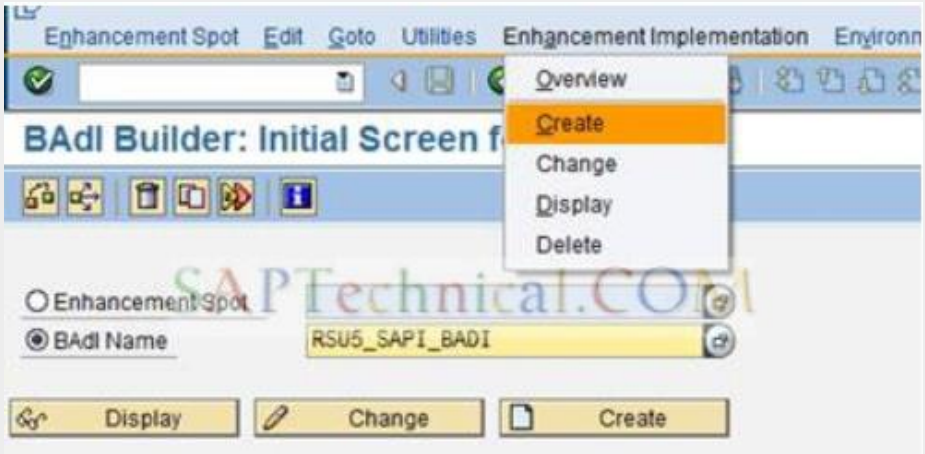


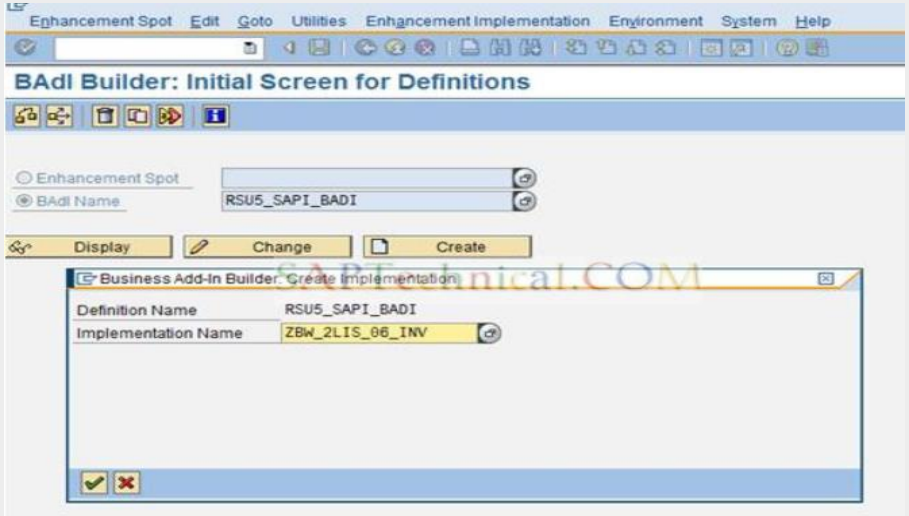
BADI (Business Ad-Inns) in the BW Data source Extraction Process

Write the code in the method “IF_EX_RSU5_SAPI_BADI~DATA_TRANSFORM” to populate data in the Enhanced field and activate the class.

Implementation of created BADI

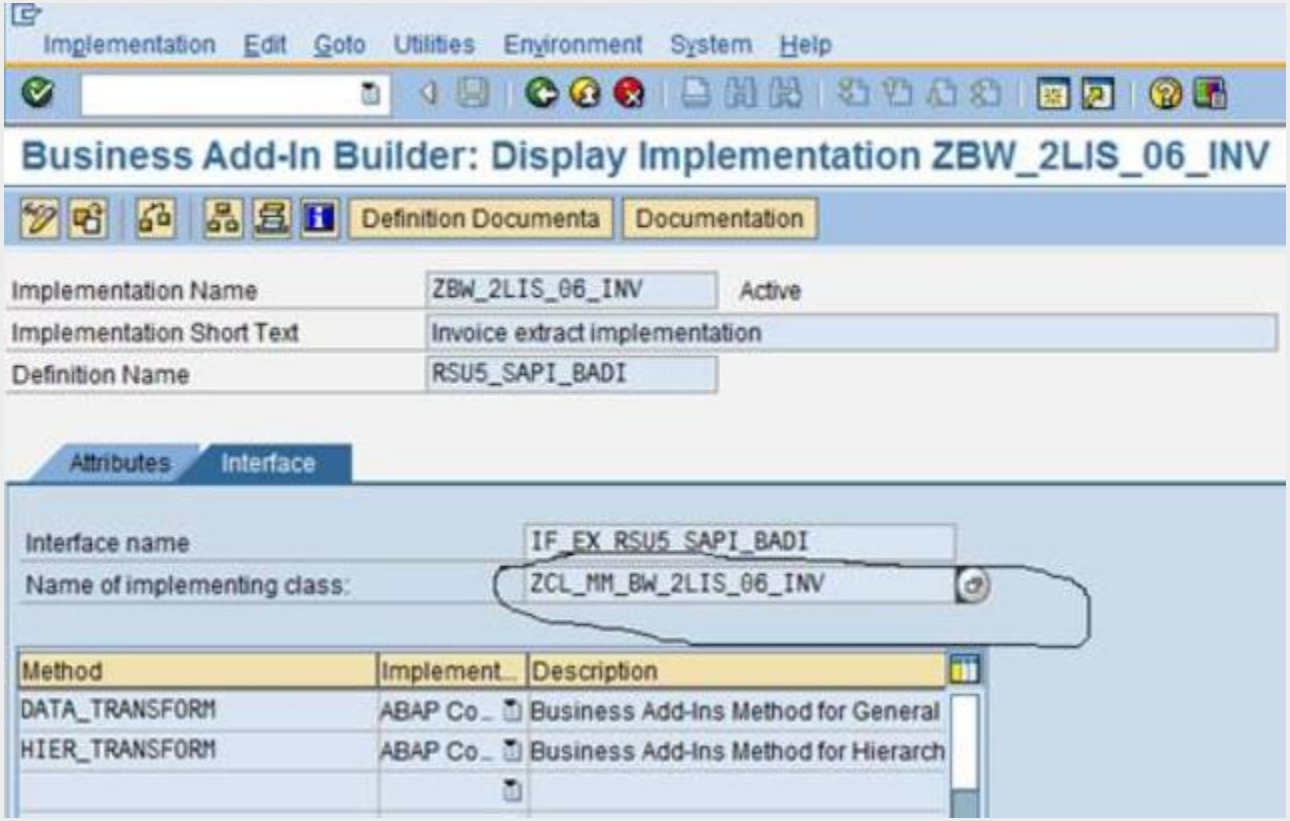
Go to BADI Definition “**RSU5_SAPI_BADI”** using T-code **SE18** and create implementation “**ZBW_2LIS_06_INV**” or you can use T-code **SE19** and create the implementation.





BADI (Business Ad-Inns) in the BW Data source Extraction Process

Go to the Interface tab and enter the created class “ZCL_MM_BW_2LIS_06_INV” in the Implementing class name field and activate



Testing through RSA3

Go to RSA3 and execute the DataSource “2LIS_06_INV” and check the extracted data to see the data to the enhanced field is filled.

CMOD Datasource Enhancement (Normal Datasource)

Steps :

FOR enhancement of datasources other than LO COCKPIT Datasources

1. Goto transaction RSA6 which is used for post-processing Datasource after they have been activated from the Business Content using transaction RSA5. Choose the Datasource and double click on it.
2. Open the extract structure and then click on append structure. If the enhancement is being done for the first time then you'll have to create a new append structure by specifying a technical name.
3. Add the new fields with a prefix of ZZ and specify there component type. Check, save and activate the append structure and this will appear in the extract structure of the Datasource.
4. Goto transaction RSA6 and edit the Datasource. Unhide the newly enhanced fields which will be present at the end.
5. Write an enhancement code for populating these new fields from their respective table. Goto transaction CMOD and choose the project for Enhancements ZBW220 (in present case), if it's not present then you have to create a project using the enhancement RSAP0001. SAP provides enhancement RSAP0001 that can be used to populate the extract structure. It has four components that are specific to each of the four types of R/3DataSources:

Transaction data - EXIT_SAPLRSAP_001
Master data attributes - EXIT_SAPLRSAP_002
Master data texts - EXIT_SAPLRSAP_003
Master data hierarchies - EXIT_SAPLRSAP_004

6. Go to enhancement of Transaction data attributes Datasources and click on include ZXRSAU01. Code will be written inside this include. Once the code is written then activate the include program.
7. Create a new executable program in transaction SE38 for the enhanced Datasource. Check and activate the program

8. Check the data using transaction RSA3. And with this we finish with the enhancement of the data-source in the source system.
9. Logon to the BI system, replicate the data-source and activate it for the changes to appear in BI.

CMOD Datasource Enhancement (LO Cockpit Datasource)

Steps :
FOR enhancement of LO COCKPIT Datasources

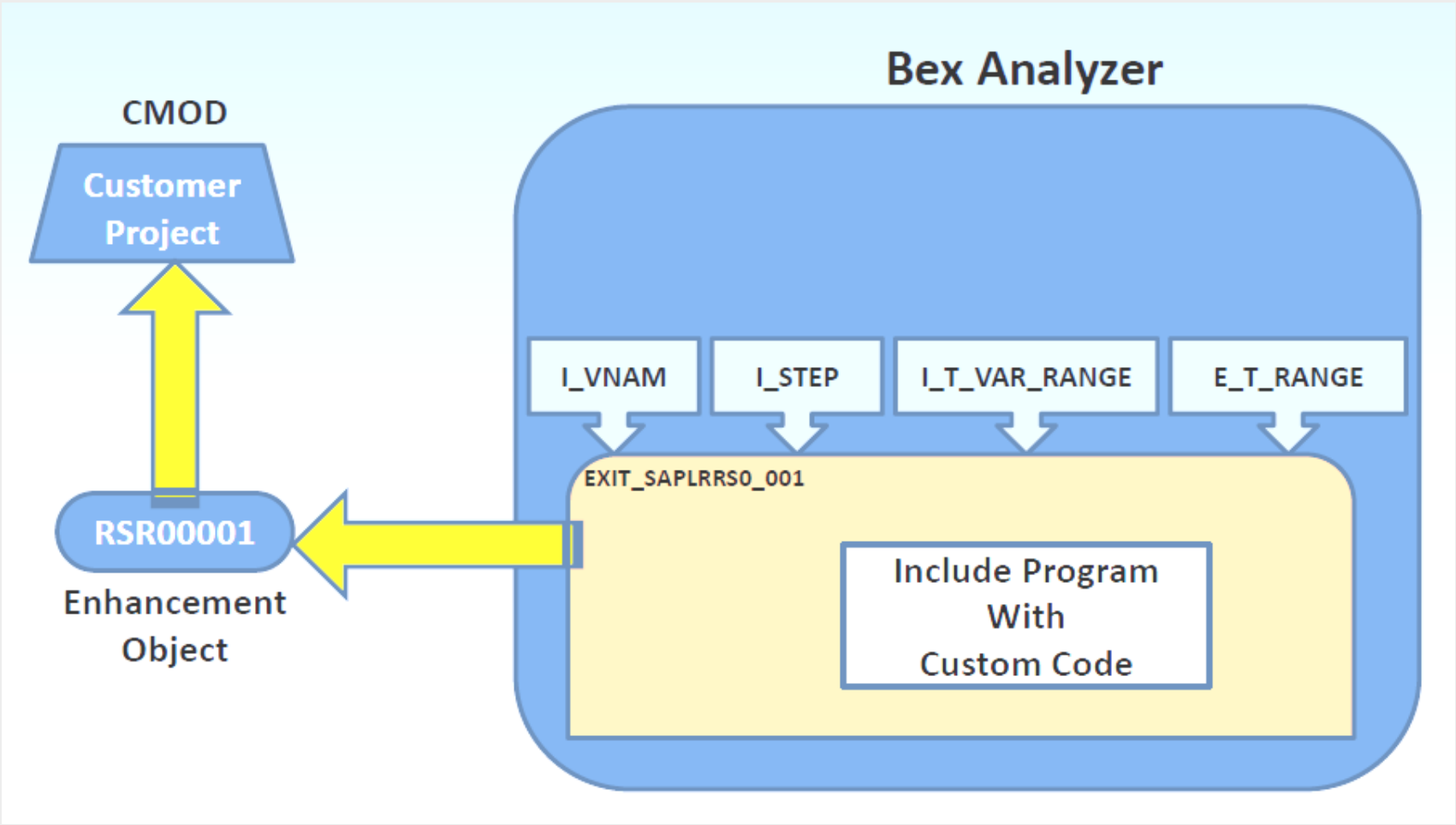
1. Goto transaction LBWG to delete the setup tables for the application specific to this Datasource.
2. Check for any Outbound Queue for the specific application component in transaction LBWQ and if some entries are present for that then we have to delete them.
3. Transaction RSA7 for our particular data-source and if some records have been posted then we should also delete them.
4. After completing these steps Goto transaction LBWE which is for customizing the data-sources like enhancements or Choosing a different Update Mode depending on the requirement.

5. Goto transaction RSA6 which is used for post-processing Datasource after they have been activated from the Business Content using transaction RSA5. Choose the Datasource and double click on it.
6. Open the extract structure and then click on append structure. If the enhancement is being done for the first time then you'll have to create a new append structure by specifying a technical name.
7. Add the new fields with a prefix of ZZ and specify there component type. Check, save and activate the append structure and this will appear in the extract structure of the Datasource.
8. Goto transaction RSA6 and edit the Datasource. Unhide the newly enhanced fields which will be present at the end.
9. Write an enhancement code for populating these new fields from their respective table. Goto transaction CMOD and choose the project for Enhancements ZBW220 (in present case), if it's not present then you have to create a project using the enhancement RSAP0001. SAP provides enhancement RSAP0001 that can be used to populate the extract structure.
It has four components that are specific to each of the four types of R/3DataSources:
Transaction data - EXIT_SAPLRSAP_001
Master data attributes - EXIT_SAPLRSAP_002
Master data texts - EXIT_SAPLRSAP_003
Master data hierarchies - EXIT_SAPLRSAP_004
10. Go to enhancement of Transaction data attributes Datasources and click on include ZXRSAU01. Code will be written inside this include. Once the code is written then activate the include program.
11. Create a new executable program in transaction SE38 for the enhanced Datasource. Check and activate the program

12. Fill the setup table using tcode OLI*BW.
13. Check the data using transaction RSA3. And with this we finish with the enhancement of the data-source in the source system.
14. Logon to the BI system, replicate the data-source and activate it for the changes to appear in BI.

Customer Exit Variable in Bex Reporting

Customer Exit Variable: In BI project, consultant often come across the situation when they need to use a processing type other than manual entry / default value, replacement path, SAP exit, or authorization to fulfill the requirement of the customer, then a customer exit gives you the option of setting up a processing type for variables, tailor-made to your specific needs.



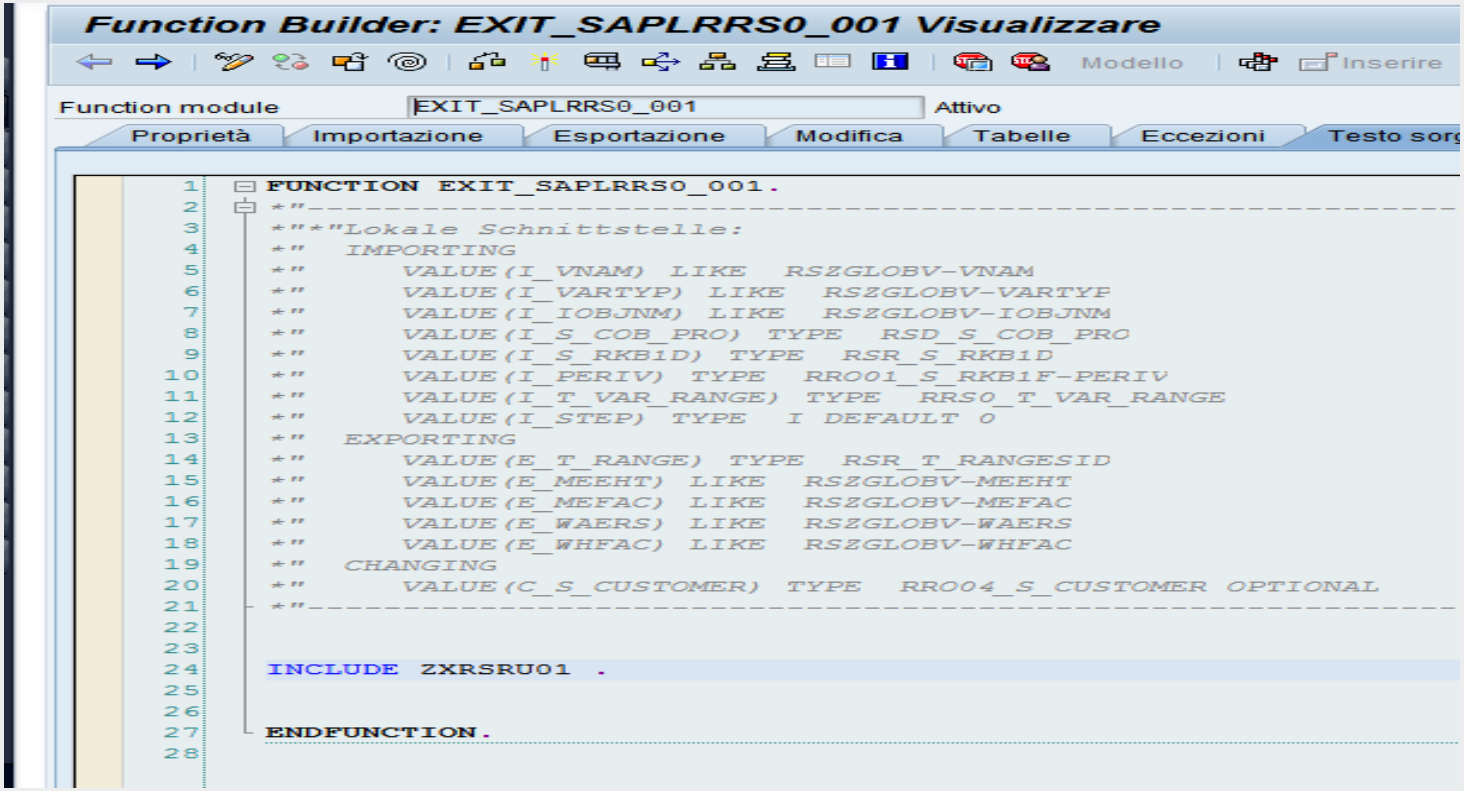
Customer Exit Variable in Bex Reporting

With CMOD you can activate Enhancement RSR00001 (**BW: Enhancements for global variables in reporting)**.
In Function Module, EXIT_SAPLRRS0_001 you have to write code within Include ZXRSRU01

In the function module (exit) shown below the import parameter **I_VNAM holds the variable (i.e name of the variable)**

The import parameter **I_T_VAR_RANGE holds all the variables and its values that were created in the BEx report**

The export parameter **E_T_RANGE** is the one to which the calculated value needs to be passed



Customer Exit Variable in Bex Reporting

The import parameter **I_STEP can hold values 0 or 1 or 2 or 3. This is a very important parameter, as this identifies, the step in which the required variable is to be processed,**

I_STEP = 0 The enhancement is not called from the BEx Query variable screen. The call can come from the authorization check or from the Monitor.

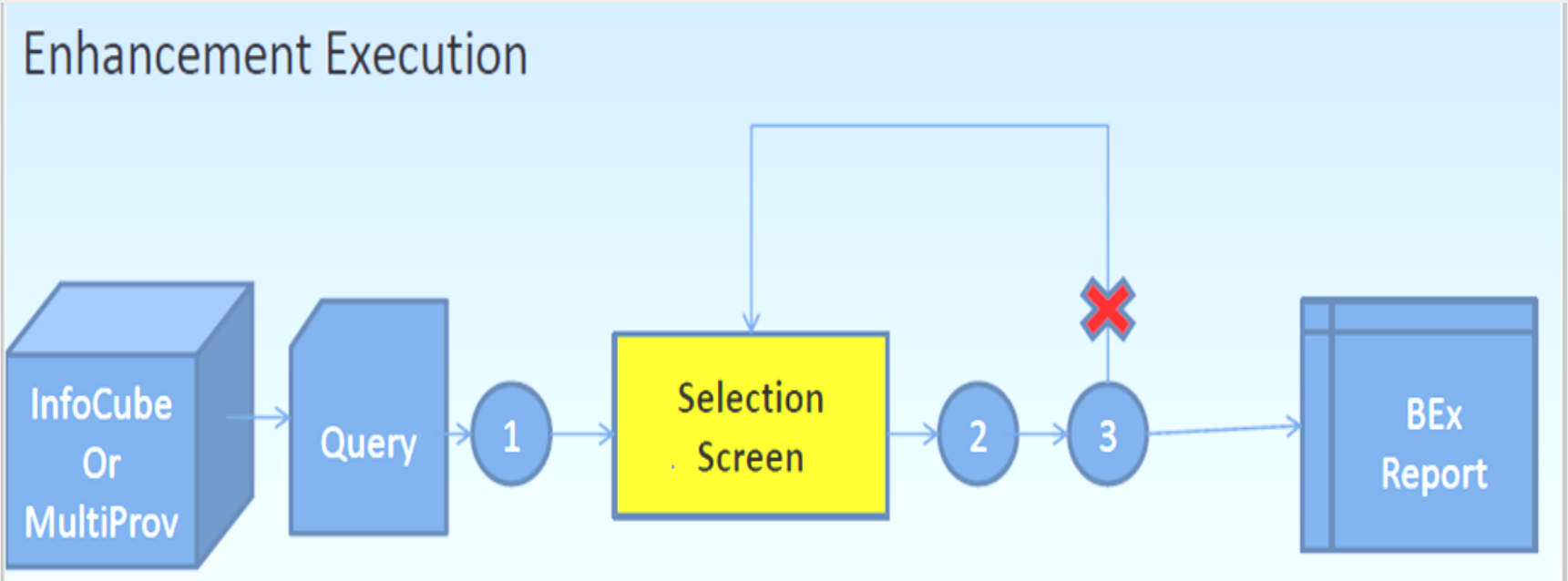
I_STEP = 1 The first step is before the processing of the variable pop-up and gets called for every variable of the processing type customer exit. You can use this step to fill your variable with default values.

I_STEP = 2 The second step is called after the processing of the variable pop-up. This step is called only for those variables that are not marked as ready for input and are set to mandatory variable entry

I_STEP = 3 The third step (I_STEP = 3) is called after all variable processing and gets called only once and not per variable. Here you can validate the user entries.

```
CASE i_STEP.
    WHEN '0'.
        ...
    WHEN '1'.
        ...

    WHEN '2'.
        CASE i_vnam.
            WHEN 'VARIABLE1'.
                LOOP AT i_t_var_range INTO loc_var_range WHERE vnam = 'XYZ'.
                    ...
                ENDLOOP.
            WHEN 'VARIABLE2'.
                ...
            WHEN 'VARIABLE3'.
                ...
        ENDCASE.
    WHEN '3'.
        ...
    ENDCASE.
```

Customer Exit Variable in Bex Reporting



Stage 1	Stage 2	Stage 3
• Programatically Identified by **I_STEP = 1** • Used to Provide Default value for a Customer Exit Variable • Variable value is maintained as a record in **E_T_RANGE**	• Programatically Identified by **I_STEP = 2** • Used to Calculate value of a Customer Exit Variable based on User Input • Variable value is maintained as a record in **E_T_RANGE** • User Input is Obtained from **I_T_VAR_RANGE**	• Programatically Identified by **I_STEP = 3** • Used to Validate User Input • Occurrence of Error can be Identified by **RAISE**ing an Exception • Error messages can be generated by the Function Module **RRMS_MESSAGE_HANDLING**

Customer Exit Variable in Bex Reporting

Structure of Internal Tables

I_T_VAR_RANGE

VNAM	SIGN	OPT	LOW	HIGH

LOW — Holds Lower Limit / Single Value

HIGH — Holds Upper Limit Value

E_T_RANGE

SIGN	OPT	LOW	HIGH

VNAM — Name of Current Variable

SIGN — 'I' for Included Values

'E' for Excluded Values

OPT — 'EQ' for Single Values

'BT' for Range of values

BW ABAP Best Practices

In a normal BW environment we use ABAP to write a code for doing a look-up in transformation or to code function modules to extract data from source systems. This process would generally involve declarations, selection of data, read the selected data to manipulate it or to perform calculations.
Below mentioned are a few best practices that a BW consultant should follow while writing ABAP code:

1. Aesthetics of programming

ALWAYS comment your code. Use comment blocks to explain a complete code flow and single line comments for individual statements wherever necessary. Your comments should explain the question of “why” and not the “what”. What is happening can be understood from the code. Maintain a change log at the start of the program with date, user ID and a short description of the change.

2. Variable Declaration

Use only required length when declaring a variable using ‘Elementary Data Types‘
E.g. : lv_xxx TYPE i; lv_yyy TYPE c length 10

Always follow a convention for declaring variable names. This will make it easier to understand the code and a lot easier when you debug the code.
E.g.: var1 TYPE c length 4 does NOT say much about the purpose of the variable, but a declaration like “lv_user_date TYPE sy-datum” would imply that the variable is a local variable and it is used to store user date in the system date format of YYYYMMDD
lv_<var> – Local Variable
gv_<var> – gloal variable
l_s_<structure> – structure definition
l_t_<table> – internal table
wa_<area> – work area

While declaring an internal table, make sure you define a structure first with the fields that would be needed. It is not recommended to define the internal table on the structure of the database table unless you are going to consume all the fields in the table
Doing this saves on space allocated for the temporary table and also improves the speed of SELECT’s and READ’s

3. SELECT Statements

Avoid using multiple SELECT statements to the same database table. Instead SELECT all that would be needed, fields to an internal table and then use a READ with necessary restriction.
Avoid using:
SELECT… ENDSELECT
SELECT Statements inside LOOPs
Non-Key fields as search conditions in SELECT statements
In the context of choosing records from a database table based on records in different database table, use a JOIN in your SELECT statement, instead of executing two separate SELECTs and looping through two internal tables

Make sure you check that the source or result package is not empty before doing a ‘FOR ALL ENTRIES’ based SELECT.
Also, try to eliminate duplicate entries from the internal table based on which records are selected using FOR ALL ENTRIES.
Make sure you check that the source or result package is not empty before doing a ‘FOR ALL ENTRIES’ based SELECT.
Using INTO CORRESPONDING FIELDS OF statement in your SELECT does not affect performance, but make sure the target internal table contains only the needed fields in the right order.
Use as many ‘WHERE’ conditions as possible in the ‘SELECT’ statements instead of having a open select and then deleting the selected entries from the internal table.

4. General:

Avoid using nested LOOP statements wherever possible.
Observe the following for READ statement

- Always SORT an internal table before you do a READ
- It is always recommended to do a BINARY SEARCH in a READ statement as long as the fields used for Sorting are same as the WITH KEY fields used when Reading
- A BINARY SEARCH in A READ statement may pick no records if the SORT and READ statements use different fields
- READ statements must always be followed by a ‘IF sy-subrc = 0’ check
- READ with TRANSPORTING NO FIELDS can be used if the fields are not used for further processing

Never hard code a break point in a program. There is always a chance that the program might get transport with the breakpoint.

In the context of transformations, when there are only one-to-one mapping between the source and target, you might use a simple expert routine. The advantage is that,

- Source package can be directly moved to result package without having to go through individual field routines for each field mapping
- LOOP AT SOURCE_PACKAGE INTO wa_source_package.
  MOVE-CORRESPONDING wa_source_package TO wa_result_package.
  APPEND wa_result_package TO RESULT_PACKAGE.

ENDLOOP.

Key performance improvement areas

1. Infocube Indexes:

Transaction code-> RSA1, Manage (of info-cubes)- Performance tab
Indexes are data structure sorted values containing pointer to records in table. Indexes are used to improve data reading performance / query performance improvement but decreases data loading/writing performance .We delete/drop them during the data loading to data target and create again after loading finished. Its recommended to include them while designing the process chain. In process chain, before loading the data to cube use the delete index process and load the cube and create index.

Use transaction **RSRV (and RSRVALT)** on a regular basis to check Infocubes. Most importantly tab 'Database', option 'Database indices of an infocube and its aggregate' to check the health of the cube

Using the Check Indexes button, you can check whether indexes already exist and whether these existing indexes are of the correct type (bitmap indexes).
Yellow status display: There are indexes of the wrong type
Red status display: No indexes exist, or one or more indexes are faulty

2. Delete PSA Data:

Transaction Code –> RSA15
• Determine a retention period for data in the PSA tables. This will depend on the type of data involved and data uploading strategy. If PSA data is not deleted on a regular basis, the PSA tables go unrestricted. Very large tables increase the cost of data storage, the downtime for maintenance tasks and performance of the data load.
You can also delete PSA data by adding the PSA Deletion process in the process chains by which the data from the PSA will be deleted periodically.

3. Delete Change log data:

For change logs, the deletion can be done from DSO->Manage->Environment->Delete change log data.
Please note that only already updated change log requests can be deleted and after deletion a reconstruction of requests for subsequent data targets using the DSO change log will not be possible.

4. Compression:

Transaction Code –> RSA11
• Info cubes should be compressed regularly (See notes 375132,407260,590370 for more details). Uncompressed cubes increase data volume and have negative effect on query and aggregate build performance. If too many uncompressed requests are allowed to build up in an infocube, this can eventually cause unpredictable and severe performance problems.

Key performance improvement areas

5. Archiving:

Transaction Code –> SARA, ABAP 'RSEXARCA
Without archiving, unused data is stored in the database and the DSO's and Infocubes can grow unrestricted. This can lead to deterioration of general performance.
The benefits of BW archiving include:
• Enables you to archive data from InfoCubes and ODS objects and delete the archived data from the BW database. This reduces the data volume and, thus, improves upload and query performance.
• Reduction of online disk storage.
• Improvements in BW query performance.
• Increased data availability as rollup, change runs and backup times will be shorter.
• Reduced hardware consumption during loading and queries.

6. **Data loads: Info packages, DTP's and process chains:**

Transaction Codes –> RSMO, RSMON, RSPC, RSPCM
SAP delivered ABAP program: /SSA/BWT
• Make sure that all the infopackages finished successfully with Tx.RSMO
• Make sure all process chains have finished successfully, rerun them if necessary.
• Verify if transaction data loads with 0 records are valid
• Analyze PSA error requests that occur. Determine if the error records are valid.

7. Aggregates: Status/Usage:

Transaction Code –> RSA1, RSMON
Aggregates should boost query performance on an Infocube, but may affect the load performance while filling them. An aggregate should not be created lightly; there should always be a good reason for its existence. If the query runtime is mainly spent on the database, suitable aggregates should be created. The ratios between the rows selected and rows transferred indicates potential performance improvements with aggregates

Introduction to ABAP Object Oriented Programming

Object-oriented programming (OOP) is a programming language model organized around objects rather than "actions" and data rather than “logic”. Historically, a program has been viewed as a logical procedure that takes input data, processes it, and produces output data.
The object-oriented approach, however, focuses on objects that represent abstract or concrete things of the real world. These objects are first defined by their character and their properties, which are represented by their internal structure and their attributes (data). The behavior of these objects is described by methods (functionality).

Comparison between Procedural and Object Oriented Programming

Features	Procedure Oriented approach	Object Oriented approach
Emphasis	Emphasis on tasks	Emphasis on things that does those tasks.
Modularization	Programs are divided into smaller programs known as functions	Programs are organized into classes and objects and the functionalities are embedded into methods of a class.
Data security	Most of the functions share global data	Data can be hidden and cannot be accessed by external sources.
Extensibility	Relatively more time consuming to modify for extending existing functionality.	New data and functions can be easily added whenever necessary

Object Oriented Programming Concept

The key features of Object Oriented Approach are:

1. Better Programming structure
2. Real world entity can be modeled very well
3. Stress on data security and access
4. Reduction in code redundancy
5. Data encapsulation and abstraction

Attributes of Object Oriented Programming:

1. Inheritance
2. Abstraction
3. Encapsulation
4. Polymorphism

Inheritance:

One of the most important concepts in object oriented programming is that of inheritance. Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application. This also provides an opportunity to reuse the code functionality and fast implementation time.
When creating a class, instead of writing completely new data members and methods, the programmer can designate that the new class should inherit the members of an existing class. This existing class is called the base class or super class, and the new class is referred to as the derived class or sub class.

An object of one class can acquire the properties of another class.
Derived class inherits the data and methods of a super class. However, they can overwrite methods and also add new methods.
The main advantage of inheritance is reusability.
The inheritance relationship is specified using the '**INHERITING FROM**' addition to the class definition statement.

Following is the syntax −
CLASS <subclass> DEFINITION INHERITING FROM <superclass>.

A derived class can access all the non-private members of its base class. Thus super class members that should not be accessible to the member functions of sub classes should be declared private in the super class. We can summarize the different access types according to who can access them in the following way –

Object Oriented Programming Concept

Access	Public	Protected	Private
Same calss	Yes	Yes	Yes
Derived class	Yes	Yes	No
Outside class	Yes	No	No

When deriving a class from a super class, it can be inherited through public, protected or private inheritance. The type of inheritance is specified by the access specifier as explained above. We hardly use protected or private inheritance, but public inheritance is commonly used. The following rules are applied while using different types of inheritance.

Public Inheritance − When deriving a class from a public super class, public members of the super class become public members of the sub class and protected members of the super class become protected members of the sub class. Super class's private members are never accessible directly from a sub class, but can be accessed through calls to the public and protected members of the super class.

Protected Inheritance − When deriving from a protected super class, public and protected members of the super class become protected members of the sub class.

Private Inheritance − When deriving from a private super class, public and protected members of the super class become private members of the sub class.

Redefining Methods in Sub Class

The methods of the super class can be re-implemented in the sub class. Few rules of redefining methods −

The redefinition statement for the inherited method must be in the same section as the definition of the original method.

If you redefine a method, you do not need to enter its interface again in the subclass, but only the name of the method. Within the redefined method, you can access components of the direct super class using the super reference. The pseudo reference super can only be used in redefined methods.

Object Oriented Programming Concept

Polymorphism

Polymorphism is a characteristic of being able to assign a different behavior or value in a subclass, to something that was declared in a parent class. For example, a method can be declared in a parent class, but each subclass can have a different implementation of that method. This allows each subclass to differ, without the parent class being explicitly aware that a difference exists.

1. Allows one interface to be used for a general class of actions.
2. When objects from different classes react differently to the same procedural call.
3. User can work with different classes in a similar way, regardless of their implementation.
4. Allows improved code organization and readability as well as creation of “extensible” programs.
5. Although the form of address is always the same, the implementation of the method is specific to a particular class.

Encapsulation:

The wrapping up of data and methods into a single unit (called class) is known as Encapsulation. The data is not accessible to the outside world only those methods, which are wrapped in the class, can access it.

Encapsulation is an Object Oriented Programming (OOP) concept that binds together data and functions that manipulate the data, and keeps both safe from outside interference and misuse. Data encapsulation led to the important OOP concept of data hiding.

Encapsulation is a mechanism of bundling the data and the functions that use them, and data abstraction is a mechanism of exposing only the interfaces and hiding the implementation details from the user.

ABAP supports the properties of encapsulation and data hiding through the creation of user-defined types called classes. As discussed earlier, a class can contain private, protected and public members. By default, all items defined in a class are private.

Encapsulation by Interface

Encapsulation actually means one attribute and method could be modified in different classes. Hence data and method can have different form and logic that can be hidden to separate class.

Let's consider encapsulation by interface. Interface is used when we need to create one method with different functionality in different classes. Here the name of the method need not be changed. The same method will have to be implemented in different class implementations.

Example - Think of the capsules that doctor recommend you , which is nothing but the packaging of different beneficial drugs , leaves etc. into one reusable module.

Abstraction:

Abstraction is a feature through which the essential parts of the object is represented hiding the background details.

•Data is abstracted by high level language constructs such as numbers ,letters , high level data concepts etc.

•Functionality is abstracted by presenting interfaces that hide complex logics such as methods or functions.

Example - When we are driving car we simply apply the brakes without knowing what is happening in the background that stop the car i.e unnecessary details are abstracted.

Objects – Classes and their Types

Objects:
An object is a section of source code that contains data and provides services. The data forms the attributes of the object. The services are known as methods (also known as operations or functions). They form a capsule which combines the character to the respective behavior. Objects should enable programmers to map a real problem and its proposed software solution on a one-to-one basis.

Classes:
Classes describe objects. From a technical point of view, objects are runtime instances of a class. In theory, you can create any number of objects based on a single class. Each instance (object) of a class has a unique identity and its own set of values for its attributes.
With all of the robust data object types available in the ABAP programming language, you might wonder why we even need classes in the first place. After all, isn't a class just a fancy way of defining a structure or function group? This limiting view has caused many developers to think twice about bothering with OO development. As you will see, there are certain similarities between classes and structured data types, function groups, etc. However, the primary difference with classes centers around the quality of abstraction. Classes group data and related behavior(s) together in a convenient package that is intuitive and easy to use. This intuitiveness comes from the fact that classes are modeled based on real-world phenomena. Thus, you can define solutions to a problem in terms of that problem's domain – more on this in a moment.

Classes and Objects: Defined
You can think of a class as a type of blueprint for modeling some concept that you are simulating in a problem domain. Inside this blueprint, you specify two things: attributes and behaviors. An attribute describes certain characteristics of the concept modeled by the class. For instance, if you were creating a "Car" class, that class might have attributes such as "make", "model", "color", etc. Technically, attributes are implemented using various data objects (e.g. strings, integers, structures, other objects, etc.). The behavior of the class is defined using methods. The "Car" class described earlier might define methods such as "drive( )", "turn( )", and "stop( )" to pattern actions that can be performed on a car.

Classes and their Types

Local and Global Classes
As mentioned earlier a class is an abstract description of an object. Classes in ABAP Objects can be declared either globally or locally.

Global Class: Global classes and interfaces are defined in the Class Builder (Transaction SE24) in the ABAP Workbench. They are stored centrally in class pools in the class library in the R/3 Repository. All of the ABAP programs in an R/3 System can access the global classes

Local Class: Local classes are defined in an ABAP program (Transaction SE38) and can only be used in the program in which they are defined.

	Global Class	Local Class
Accessed By	Any program	Only the program where it is defined.
Stored In	In the Class Repository	Only in the program where it is defined.
Created By	Created using transaction SE24	Created using SE38
Namespace	Must begin with Y or Z	Can begin with any character

Every class will have two sections.
(1) Definition. (2) Implementation

Definition: This section is used to declare the components of the classes such as attributes, methods, events. They are enclosed in the **ABAP statements CLASS ... ENDCLASS.**
CLASS < class> DEFINITION.
...
ENDCLASS.

Implementation: This section of a class contains the implementation of all methods of the class. The implementation part of a local class is a processing block where actually the business logic/functionality is written.
CLASS < class> IMPLEMENTATION.
...
ENDCLASS.

Thank You

Capgemini