Introduction to SAP ABAP

ABAP is first developed in 1980's it is one of the main application-specific fourth-generation language.

ABAP used to be an abbreviation of ***A****llgemeiner* ***B****erichts****A****ufbereitungs****P****rozessor*, German for "generic report preparation processor", but was later renamed to the English ***A****dvanced* ***B****usiness* ***A****pplication* ***P****rogramming*. The ABAP language was originally used by developers to develop the SAP R/3 platform. In 1999, SAP released an object-oriented extension to ABAP called ABAP Objects, along with R/3 release 4.6. SAP's current development platform NetWeaver supports both ABAP and Java.

SAP R/3 : Three-Tier Architecture:

1. Presentation Layer.
2. Application Layer.
3. Data Base Layer.

Presentation Layer: This layer takes care of the presentation logic. It provides the user interface to the user where user can see the screen interacts with the screen and performs operations that he wanted to and sees the result. The formation of screens and user interfaces are taken care by this layer.

Application Layer: In this layer business logic is executed. The application layer can be installed on one machine, or it can be distributed among more than one system.

Database Layer: The database layer holds the data. SAP supports any relational database. Data that are created in the organization are stored permanently in the database this layer provides the data to the application layer.

Transaction: A transaction in SAP words is the execution of program. The ABAP program is executed by entering the transaction code in the SAP system. Transaction can be called via system-defined or user-specific menus. They can also be executed by entering the transaction code directly in the command field. Transactions can also be worked programmatically by ABAP statements CALL TRANSACTION and LEAVE TO TRANSCTION. Transaction is also called as a Logical Unit Of Work (LUW) in SAP terminology. The short form of transaction code is T-Code.

Types Of ABAP Programs:

ABAP program is either an executable unit or a library, which provides reusable code to other programs and is not independently executable.

ABAP distinguishes two types of executable programs:

- Reports
- Module pools

Reports follow a relatively simple programming model whereby a user optionally enters a set of parameters (e.g., a selection over a sub SET of data) and the program then uses the input parameters to produce a report in the form of an interactive list. The term "report" can be somewhat misleading in that reports can also be designed to *modify* data; the reason why these programs are called reports is the "list-oriented" nature of the output they produce.

Module pools define more complex patterns of user interaction using a collection of screens. The term "screen" refers to the actual, physical image that the user sees. Each screen also has a "flow logic", which refers to the ABAP code implicitly invoked by the screens, which is divided into a "PBO" (Process Before

Output) and "PAI" (Process After Input) section. In SAP documentation the term “dynpro” (dynamic program) refers to the combination of the screen and its flow logic.

The non-executable program types are:

- INCLUDE modules
- Subroutine pools
- Function groups
- Object classes
- Interfaces
- Type pools

ABAP Workbench

The ABAP **Workbench** is part of the ABAP system and is accessed via SAP GUI. It contains different tools for editing programs. The most important of these are (Transaction codes):

- (SE38) *ABAP Editor* for writing and editing reports, module pools, includes and subroutine pools
- (SE11) *ABAP Dictionary* for processing database table definitions and retrieving global types
- (SE41) *Menu Painter* for designing the user interface (menu bar, standard toolbar, application toolbar, function key assignment)
- (SE51) *Screen Painter* for designing screens and flow logic
- (SE37) *Function Builder* for function modules
- (SE24) *Class Builder* for ABAP Objects classes and interfaces

(Transaction SE80) The *Object Navigator* provides a single integrated interface into these various tools.

Therefore an ideal report program should start with:

```
Report <report name> no standard page heading
line-size <size>
line-count <n(n1)>
message-id <message class>.
```

Comments

ABAP has 2 ways of defining text as a comment:

- An asterisk (*) in the leftmost column of a line makes the entire line a comment
- A double quotation mark (") anywhere on a line makes the rest of that line a comment

Example:

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

** Program: GTP_Academy              **

** Author:  Madhavi  14-Dec-2007     **

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

REPORT GTP_Academy.

* Read flight bookings from the database

SELECT * **FROM** FLIGHTINFO

WHERE CLASS = 'Y' "Y = economy

OR CLASS = 'C'. "C = business

Chained statements (:): Consecutive statements with an identical first (leftmost) part can be combined into a "chained" statement using the chain operator :. The common part of the statements is written to the left of the colon, the differing parts are written to the right of the colon and separated by commas. it is a chaining operator by this we can define more than one data declaration by separating each field with comma(,).

WRITE FLIGHTINFO-CITYFROM.

WRITE FLIGHTINFO-CITYTO.

WRITE FLIGHTINFO-AIRPTO.

Chaining the statements results in a more readable and more intuitive form:

WRITE: FLIGHTINFO-CITYFROM, FLIGHTINFO-CITYTO, FLIGHTINFO-AIRPTO.

The entire common part of the consecutive statements can be placed before the colon. Example:

REPLACE 'A' **WITH** 'B' **INTO** LASTNAME.

REPLACE 'A' **WITH** 'B' **INTO** FIRSTNAME.

REPLACE 'A' **WITH** 'B' **INTO** CITYNAME.

could be rewritten in chained form as:

REPLACE 'A' **WITH** 'B' **INTO:**LASTNAME, FIRSTNAME, CITYNAME.

Selection Screen

"Selection screen" is the input screen based on that the program should run.
The selection screen is normally generated from the

1. Parameters
2. Select-Options

Syntax

```
Selection-screen begin of screen <screen #>
selection-screen begin of block <#>  with frame title <text>
.....
.....
selection-screen end of block <#>
selection-screen end of screen <screen #>
```

Parameters

Parameters help one to do dynamic selection. They can accommodate only one value for one cycle of execution of the program.

Syntax:

Defining parameters as a data type

```
Parameters p_id(30) type c.
Parameters: p_name(10) thpe c,
            P_age(3) type i.
```

Defining parameters like a table field.

```
Parameter p_id like <table name>-<field name>.
```

Parameters can be Checkboxes as well as Radiobuttons.

```
Parameters p_id as checkbox.Parameters p_id1 radiobutton group <group name>.
Parameters p_id2  radiobutton group <group name>.
```

Parameters can be listbox.

```
Parameter p_id like <table name>-<field name> as listbox
```

Select Options

A Select-Option is used to input a range of values or a set of values to a program

Syntax

Select-options: s_vbeln for vbak-vbeln.

You can also define a select option like a variable

Select-options: s_vbeln for vbak-vbeln no intervals.

Select-options: s_vbeln for vbak-vbeln no-extension.

```
select-options:  s_vbeln for vbak-vbeln no intervals no-extension
```

Formatting the report

ABAP allows the reports to be formatted as the user wants it to be. For example, "Alternate Lines" must appear in different colors and the "Totals" line should appear in Yellow.

Syntax

```
Format Color n
Format Color n Intensified On
```

n may correspond to various numbers

Please note that there are other additions along with format as well

```
FORMAT COLOR OFF INTENSIFIED OFF INVERSE OFF HOTSPOT OFF INPUT OFF
```

Message Class: To display any information or error message we add a message class to the program using the addition:

Syntax:

Message-id <message class name>. Message classes are maintained in SE91.

Internal Table Basics:

Internal table is a data object in ABAP that exists only at run time of a program. It means when the program execution is complete then the internal table will be lost. We use internal table to store database table data after fetching it by a select query. The ABAP run-time system dynamically manages the internal table's memory. It means we developer do not need to work on memory management of internal table.

Internal table has three parts – **rows, columns & work area**.

- Rows are the line type of internal table. It is a structure which contains several fields. Those fields are of data elements. We need to declare the structure locally or globally to declare the internal table.
- Columns are the fields of internal table. Those fields are of different data elements declared by locally or globally.
- The most important part of an internal table is its work area. Work area is basically the line type of an internal table. It means it has the same structure of the rows of internal table. Work area contains the same fields of same type of the rows. It is of two types – implicit & explicit work area.

1. When we declare an internal table with header line then a work area is automatically created with the same name of the table. This work area is called implicit work area which is actually the header line. There is no need to declare work area separately. This work area / header line contains the same table as of the internal table.

   Example:-

```
TYPES: BEGIN OF ty_mara,
  matnr TYPE mara-matnr,
  werks TYPE marc-werks,
  lgort TYPE mard-lgort,
```

END OF ty_mara.
DATA: it_mara TYPE STANDARD TABLE OF ty_mara WITH HEADER LINE.

Header line concept:

MATNR	WERKS	LGORT

The name of this work area / header line is IT_MARA.

When we create the internal table then it is like following:

MATNR	WERKS	LGORT

It also contains the same name IT_MARA but it is mentioned IT_MARA[ ] in the program.

2. If we declare an internal table without header line then we need to declare its work area seperately. Since we are declaring the work area explicitly it is called explicit work area. This work area contains the different name from the internal table.

Example :–

```
TYPES: BEGIN OF ty_mara,
         matnr TYPE mara-matnr,
         werks TYPE marc-werks,
         lgort TYPE mard-lgort,
       END OF ty_mara.

DATA: it_mara TYPE STANDARD TABLE OF ty_mara,
      wa_mara TYPE ty_mara. OR  wa_mara like line of it_mara.
```

Work area concept:

MATNR	WERKS	LGORT

The name of this work area is WA_MARA.

When we create the internal table then it is like following:

MATNR	WERKS	LGORT

The table contains the name IT_MAT.

Types Of Internal Tables:

1. **Standard table** is an index table which has non-unique key. It can be accessed by index or key also. If we want to access by key then the key must be defined otherwise default key would be considered. The declaration is as follows:

   DATA: it_mara TYPE STANDARD TABLE OF ty_mat WITH NON-UNIQUE KEY matnr.
   OR
   DATA: it_mat TYPE TABLE OF ty_mat WITH NON-UNIQUE KEY matnr.

   If we don't mention "Standard table of" clause then by default the system takes it as a standard internal table. We can enter data into a standard internal table by using the APPEND statement. Append always enters data at the last row of the table.

   APPEND wa_mara TO it_mara.

2. **Sorted table** is another kind of index table which has unique / non unique key. It also can be accessed via index or key. For sorted table the key must be specified. The declaration is as follows:

   DATA: it_mara TYPE SORTED TABLE OF ty_mara WITH UNIQUE KEY matnr,
   it_mara TYPE SORTED TABLE OF ty_mara WITH NON-UNIQUE KEY matnr.

   Unique key means the MATNR (material no) will must be unique. If same material number is inserted then a run time error will happen. However we can declare the sorted table with non unique key also. In this case same material number can be entered but it will be sorted after entering the number. Here the sorted table behaves similar to sorted standard table. We use INSERT statement to enter any records to the sorted table.

   INSERT wa_mara INTO it_mara.

3. **Hashed table** is not an index table. It follows the hash algorithm. Here the declaration of key is must and also the key must be unique. Hence no duplicate entry will be in the hashed table. We can access records only by the key.

   DATA: it_mara TYPE HASHED TABLE OF ty_mara WITH UNIQUE KEY matnr.

   Similar to sorted tables data can be inserted here by INSERT statement. Hashed tables are used when the internal table contains huge volume of data.

   INSERT wa_mara INTO TABLE it_mara.

Table kind	**Index Tables**		**Hashed Tables**
	Standard Table	**Sorted Table**	
Index Access	Yes	Yes	No
Key Access	Yes	Yes	Yes
Key Uniqueness	Non unique	Unique/Non unique	Unique
Usage	Index access	Key access	Only key access