Data Type

20682032 이채희

목차

1) Overview

-Data Type

-Data Variable

-Data 구문

-전역 변수

-지역 변수

2) Predefined ABAP Type

-고정 길이를 가지는 Predefined ABAP Type

-Numeric Data Type

-Character Type

-가변 길이를 가지는 Predefined ABAP Type

-Hexadecimal(16진수) Type

3) 프로그램의 Local Type

-TYPE 선언

4) ABAP Dictionary의 Type

-Table, View를 이용한 TYPE 선언

-Data Type을 이용한 TYPE 선언

-Type Group을 이용한 TYPE 선언

5) Assigning Value(값 할당)

-Assigning value(값 할당)

-offset을 이용한 값 할당

01. Overview

Data Type

: 프로그램에서 사용할 수 있는 데이터 타입을 정의, Data Variable(변수)과 독립적으로 선언

*종류

-Predefined ABAP Type

: c,java의 c,f,I 타입과 같이 이미 정의되어 있는 데이터 type(ABAP은 추가로 p,d,t 타입 존재), 고정길이를 가짐

-Local Data Type in Program

: ABAP 프로그램 내에서 Predefined ABAP Type을 이용해 생성된 데이터 type

-Global Data Type in ABAP Dictionary

: 모든 ABAP 프로그램에서 사용할 수 있는 Global Data Type, ABAP 프로그램 내에서 TYPE 구문 이용 가능, T-code:se11에서 생성

Data Variable

: Data Type을 참고해 값을 저장할 수 있는 변수, 프로그램 실행 시점에 메모리를 차지하는 데이터 변수

*선언

-Predefined ABAP Type 이용

```
DATA : gv_num        TYPE i,
  gv_deci            TYPE f,
  gv_date            TYPE d,
  gv_time            TYPE t,
  gv_text1           TYPE c,
  gv_text2(2)        TYPE c,
  gv_longtext        TYPE string.
```

Predefined ABAP Type

-프로그램의 Local Type 이용

: 프로그램 내에서만 ABAP 기본 데이터 타입처럼 사용 가능

```
TYPES : BEGIN OF t_struct,
  col1  TYPE  c,
  col2  TYPE  i,
  col3  TYPE  d,
  END OF t_struct.
```

Local TYPE 선언

```
DATA : gs_struct    TYPE t_struct,
       gv_var        LIKE gs_struct-col1.
```

-ABAP Dictionary Type 이용

: 모든 프로그램에서 선언해 사용할 수 있음

```
DATA : gv_carrid     TYPE s_carr_id,
  gv_connid          TYPE sflight-carrid,
  gv_matnr           TYPE mara-matnr.
```

Type in ABAP Dictionary

DATA 구문

: 데이터 변수를 선언할 때 사용, 변수명은 (_)기호 포함 30자까지

-TYPE type : 데이터 변수 타입 정의

{ DATA : gv_num TYPE i. }

```
DATA  gv_num1   TYPE I,          DATA : gv_num1   TYPE I,
DATA  gv_num2   TYPE I.    →            gv_num2   TYPE I
```

-LIKE num : 앞에서 생성한 Data Variable과 동일한 타입의 변수를 선언할 때 사용

{ DATA : gv_num2 LIKE gv_num1. }

-VALUE int : 모든 데이터 타입은 Initial Value가 존재하지만 VALUE 옵션을 사용하면 Initial Value 대신 int로 초깃값 설정, 상수 변수 선언시 필수

{ DATA : gv_num TYPE i VALUE 123. }

-LENGTH n : 필드 길이 설정, C,N,P,X 타입에만 사용

{ DATA : gv_num TYPE n LENGTH 2. }

-DECIMALS n : P 타입에만 사용, 1~4 사이의 소수 자리 수 설정

{ DATA : gv_num TYPE P DECIMALS 3. }

***ABAP 기본 문법**

- (:) 기호는 동일한 명령어를 (,)로 구분하여 (.)를 만날 때까지 실행

{ WRITE gv_val. }

- 프로그램 기능을 설명하는 주석은 2가지

(*) 기호는 한 라인 전체를 주석 처리

(") 기호는 뒷부분의 문자열 주석 처리

-문자열은 ' ' 기호로 처리

{ gv_val = 'Easy ABAP'. }

-명령어는 공백을 두고 처리

{ gv_val='Easy ABAP'. (x)}

{ gv_val = 'Easy ABAP'. (o)}

*ABAP Naming Rule

-ABAP Dictionary : 'Z' 또는 'Y'로 시작

-변수명 : (_)기호 포함 30자까지

(_)기호 앞 2자리: 범위 & 데이터 유형

뒷자리: 어떠한 역할을 하는지

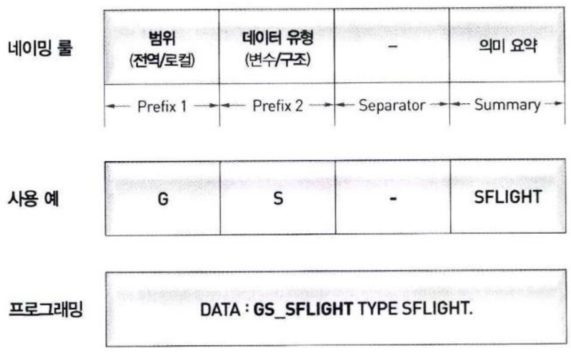


그림 2-2 일반적인 변수 Naming Rule과 사용 예

전역 변수 (Global Variable)

: 하나의 프로그램 또는 함수 그룹 내에서 접근할 수 있는 데이터 유형, 프로그램이 활성화되어 있는 상태에서는 항상 메모리를 점유, G로 시작

표 2-1 전역 변수 정의

데이터 타입	접두어	사용 예	프로그램
필드	V 또는 D	GV_CARRID 또는 GD_CARRID	DATA **GV_SFLGIHT** TYPE S_CARR_ID. 또는 DATA **GD_SFLGIHT** TYPE S_CARR_ID.
구조체	S	GS_SFLIGHT	DATA **GS_SFLIGHT** TYPE SFLIGHT.
상수	C	GC_RATE	CONSTANTS **GC_RATE** TYPE I VALUE '3.14'.
인터널 테이블	T	GT_SFLIGHT	DATA **GT_SFLIGHT** TYPE TABLE OF SFLIGHT.
클래스	O	GO_ALVGRID	DATA **GO_ALVGIRD** TYPE REF TO CL_SALV_TABLE.
RANGE	R	GR_CARRID	RANGES **GR_CARRID** FOR SFLIGHT-CARRID.
데이터 참조 변수 (데이터오브젝트)	F	GF_OBJ	DATA **GF_OBJ** TYPE REF TO OBJECT.

*ABAP 프로그램에만 존재하는 PARAMETER와 SELECT-OPTIONS 변수는 각각 'P','S'와 같이 한가지 접두어만 사용

지역 변수 (Local Variable)

: 프로그램의 모듈(함수나 서브루틴 같이 단위 기능을 구현하기 위해 스크립트를 그룹화한 것) 내에서만 유효

표 2-2 지역 변수 정의

데이터 타입	접두어	사용 예	프로그램
필드	V 또는 D	LV_CARRID 또는 LD_CARRID	DATA **LV_SFLGIHT** TYPE S_CARR_ID. 또는 DATA **LD_SFLGIHT** TYPE S_CARR_ID.
구조체	S	LS_SFLIGHT	DATA **LS_SFLIGHT** TYPE SFLIGHT.
상수	C	LC_RATE	CONSTANTS **LC_RATE** TYPE I VALUE '3.14'.
인터널 테이블	T	LT_SFLIGHT	DATA **LT_SFLIGHT** TYPE TABLE OF SFLIGHT.
클래스	O	LO_ALVGRID	DATA **LO_ALVGIRD** TYPE REF TO CL_SALV_TABLE.
RANGE	R	LR_CARRID	RANGES **LR_CARRID** FOR SFLIGHT-CARRID.
데이터 참조 변수 (데이터 오브젝트)	F	LF_OBJ	DATA **LF_OBJ** TYPE REF TO OBJECT.

02. Predefined ABAP Type

고정 길이를 가지는 Predefined ABAP Type

- 8가지 기본 Data Type이 있음

- D,F,I는 이미 기술적 속성이 정의되어 있어 필드 길이를 정의하지 않고 바로 사용할 수 있다

-C,N,X를 사용할 때는 필드 길이 정의 필수

-P는 DECIMALS를 명시하지 않으면 소수 자리 인식 못함

{ DATA : gv_num TYPE p DECIMALS }

	초기 길이	가능 길이	초깃값	의미
Numeric Types				
I	4	4	0	Integer (whole number)
F	8	8	0	Floating point number
P	8	1 – 16	0	Packed number
Character types				
C	1	1 – 65535	' … '	Text Field (alphanumeric characters)
D	8	8	'00000000'	Date field (Format: YYYYMMDD)
N	1	1 – 65535	'0 … 0'	Numeric text field (numeric characters)
T	6	6	'000000'	(format: HHMMSS)
Hexadecimal type				
X	1	1 – 65535	X'0 … 0'	Hexadecimal field

Numeric Data Type

-I (Integers)

: 정수 (정수가 아닌 것은 반올림), Counter, Item 수, Index 등에 쓰임

-P (Packed numbers)

: 소수 자리 허용, 사용 Size 1~16byte, 소수 최대 14자리 가능, 정확한 계산을 위한 비즈니스 계산을 목적으로 사용

-F (Floating point number)

: 지수형태로 표현되어 Function Module을 이용해 다른 데이터 타입으로 변환되어 출력, 높은 정확도를 요구한다면 P, 매우 작거나 큰 수나 Value 범위가 넓은 경우나 반올림 오류가 중요하지 않을 때는 F 사용

Character Type

: 문자 길이나 Data Type을 선언하지 않으면 기본적으로 Character 1자리로 정의,

• C	문자, 숫자, 특수문자에 사용.
• N	숫자를 C 타입으로 표현. Integer 형태를 문자 타입으로 보여줌.
• D	날짜 타입을 표현.
• T	시간 타입을 표현.

-C

: 변수 뒤에 길이를 입력하거나 LENGHTH 옵션으로 문자 길이를 명시적으로 선언해야 함

```
DATA  gv_fd.
DATA  gv_fd   TYPE c.
DATA  gt_fd(1) TYPE c.
DATA  gv_fd   TYPE c LENGTH 1.
```

Ref)

STRLEN 명령어는 문자의 길이를 반환

(/)기호는 새로운 라인을 의미

-N

: 공백은 0으로 표현되어 문자 길이만큼 LIST에 조회

-D

: sy-datum 시스템의 오늘 날짜

-T

: sy-uzeit 시스템의 현재 시간

가변 길이를 가지는 Predefined ABAP Type

- S (String)

: Data Type C와 유사하며 아주 긴 문자열을 표현할 경우 사용

Hexadecimal(16진수) Type

: 1byte가 2개의 digits로 표현되는 TYPE, ABAP에서 16진수는 프린트 언어와 그래픽 관련 영역에서 많이 쓰임

03. 프로그램의 Local Type

TYPE 선언

: 프로그램 내에서 선언해 로컬 프로그램에서만 특별한 타입으로 사용하기 위해 구문 이용

{ TYPES t_char10(10) TYPE c. } -> t_chr10이라는 새로운 타입 선언

예제 | 2-11

```
REPORT  Z02_11.

* 구조체 TYPE 선언
TYPES: BEGIN OF t_ren,
  name     TYPE c LENGTH 20,
  country  TYPE c LENGTH 15,
  city     TYPE c LENGTH 10,
END OF t_ren.

* nested 구조체 type 선언
TYPES: BEGIN OF t_info.
  INCLUDE TYPE t_ren as ren. → t_ren 구조체를 포함한다
  TYPES : phone TYPE c LENGTH 10,
END OF t_info.

* 구조체 Object 선언
DATA gs_people TYPE t_info.

gs_people-ren-name = 'KIM SUNGJOON'.
gs_people-ren-country = 'KOREA'.
gs_people-ren-city = 'SEOUL'.
gs_people-phone = '123456789'.

WRITE :/ gs_people-ren-name, gs_people-ren-country,
         gs_people-ren-city, gs_people-phone.
```

구조체 타입이 아닌 DATA BEGIN OF ~ END OF 구문을 이용할 수도 있음

마찬가지로 INCLUDE STRUCTURE-> Nested Structure

예제 | 2-13

```
REPORT  Z02_13.

DATA: BEGIN OF gs_ren,
      name     TYPE c LENGTH 20,
      country  TYPE c LENGTH 15,
      city     TYPE c LENGTH 10,
 END OF gs_ren.

 DATA : BEGIN OF gs_people.
          INCLUDE STRUCTURE gs_ren AS ren.
 DATA : phone TYPE c LENGTH 10,
 END OF gs_people.

 gs_people-ren-name = 'KIM SUNGJOON'.
 gs_people-ren-country = 'KOREA'.
 gs_people-ren-city = 'SEOUL'.
 gs_people-phone = '123456789'.

 WRITE :/ gs_people-ren-name, gs_people-ren-country,
          gs_people-ren-city, gs_people-phone.
```

04. ABAP Dictionary의 Type

Table, View를 이용한 TYPE 선언

Database Table과 View를 참고하여 구조체 및 인터널 테이블을 선언할 수 있고 Table은 개별 필드를 참고하여 변수를 선언할 수 있다.

-전체 참고

{ DATA : gs_sflight TYPE sflight. }

-개별 필드 참고

{ DATA : gv_carrid TYPE sflight-carrid. }

Data Type을 이용한 TYPE 선언

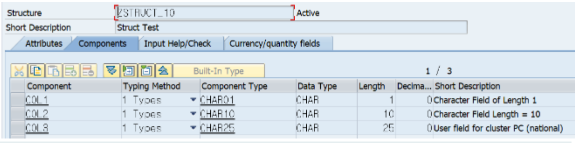
-Data Element

: 개별 필드를 정의, Data Element는 Domain을 참고, 같은 Domain을 여러 Data Element에서 사용할 수 있고 같은 Data Element를 여러 개의 테이블 필드에서 사용할 수 있음

{ DATA gs_carrid TYPE s_carr_id. }

-Sturcture

: 여러 개의 Table에 존재하는 필드들을 조합할 때 자주 사용




```
REPORT zdatatype_test_10_p18.

DATA gs_str TYPE zstruct_10.

gs_str-col1 = '1'.
gs_str-col2 = 'structure'.
gs_str-col3 = 'ABAP Dictionary Test'.

WRITE :/ gs_str-col1, gs_str-col2, gs_str-col3.
```

Type Group을 이용한 TYPE 선언

: 여러 가지 TYPE을 그룹으로 묶어서 선언할 때 사용

05. Assigning Value

Assigning value (값 할당)

```
<f2> = <f1>.
MOVE <f1> TO <f2>.
MOVE-CORRESPONDING <STRING1> TO <STRING2>.
WRITE <f1> TO <f2> [<option>].
```

MOVE TO : (=)기호와 동일하게 데이터를 할당

MOVE-CORRESPONDING : 동일한 필드명에 데이터를 순서대로 할당

WRITE : 리스트 출력 및 변수 할당

예제 | 2-20

```
REPORT  Z02_20.

DATA :  BEGIN OF gs_ren,
          name(20)        VALUE 'KIM SUNGJOON',
          country(10)     VALUE 'KOREA',
          city(20)        VALUE 'SEOUL',
        END OF gs_ren.

DATA :  BEGIN OF gs_info,
          name(20),
          city(20),
          phone(10) VALUE '1521001234',
        END OF gs_info.

MOVE-CORRESPONDING gs_ren TO gs_info.
WRITE : / gs_info-name, gs_info-city, gs_info-phone.
```

->MOVE-CORRESPONDING 예제

offset을 이용한 값 할당

: 필드의 일부 값을 정의할 수 있음

```
DATA : f1(8) VALUE 'ABCDEFGH',
       f2(8).
MOVE    f1+2(3)   TO   f2.
```

write f2의 결과 : CDE