

## 0618 | ABAP

DATA OBJECT | DATA 값을 참고하여 값을 저장할 수 있는 변수

FIELD	<input type="text"/>				
STRUCTURE	<input type="text"/> <input type="text"/>				
TABLE	<table><tr><td><input type="text"/></td><td><input type="text"/></td></tr><tr><td><input type="text"/></td><td><input type="text"/></td></tr></table>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>				
<input type="text"/>	<input type="text"/>				

\*FIELD + STRUCTURE + TABLE 의 조합으로 이루어진 OBJECT도 있다.

DATA TYPE | 프로그램에서 사용할 수 있는 DATA TYPE을 정의

LOCAL	프로그램 내에서 정의한 DATA TYPE*
GLOBAL	모든 ABAP 프로그램 내에서 사용할 수 있는 DATA TYPE**
STANDARD	PREDEFINED ABAP TYPE   기본 데이터 타입

\*프로그램 내에서 PREDEFINED ABAP TYPE을 이용해 LOCAL TYPE 생성.

\*\*ABAP DICTIONARY DATA TYPE 은 프로그램 내에서 TYPE 구문 사용 가능.

STANDARD |

COMPLETE	길이가 정해져 있음	D T   STRING
INCOMPLETE	길이가 정해져 있지 않음	C N P

INTERNAL TABLE | ABAP 핵심 기술로 단일 프로그램 내에서 정의하여 사용할 수 있는 LOCAL TABLE로 프로그램 개발 및 유지보수를 좀 더 쉽고 편리하게 할 수 있다.

FUNCTION |

FUNCTION은 기능별로 모듈화 하며 재 사용이 가능하도록 지원하는 기능이다. SUBROUTINE과 마찬가지로 기능을 수행하나, FUNCTION은 GLOBAL MODULARIZATION, SUBROUTINE은 LOCAL MODULARIZATION이라고 볼 수 있다. FUNCTION MODULE은 FUNCTION GROUP이라 불리는 POOL에 소속되어야 하며, 예외처리 기능을 제공하여 에러가 발생하면 예외 사항을 호출한 프로그램에 전달할 수 있다. 또한 호출 프로그램에 상관없이 STAND-ALONE 모드에서 테스트할 수 있다. FUNCTION을 호출할 때는 INPUT PARAMETER를 입력하고, FUNCTION 수행결과를 OUTPUT PARAMETER로 받아야한다.

FUNCTION MODULE | INTERFACE

하나의 프로그램에서 같은 기능의 구문을 여러 번 사용하면 스크립트가 길어지고, 변경 사항이 발생할 경우 구문마다 수행해야 하는 비효율적 문제를 해결하기 위해 FUNCTION을 이용한 모듈화를 구현하여 재사용을 제공하고, SCRIPT의 수를 줄일 수 있다. FUNCTION MODULE은 중앙 라이브러리 REPOSITORY에 저장되는 특별한 GLOBAL 서브 루틴으로 GLOBAL MODULARIZATION 이라 부르기도 한다. REPOSITORY 에는 ABAP 프로그램에서 호출하여 사용할 수 있도록 수많은 FUNCTION MODULE 이 존재하며, 사용자가 생성하여 사용할 수도 있다.

IMPORT PARAMETERS	선택   FUNCTION MODULE*에 전달하는 값
EXPORT PARAMETERS	선택   *
CHANGING PARAMETERS	FUNCTION MODULE에 값을 넘기고, 그 값을 변경 가능하다.
TABLES	INTERNAL TABLE**
EXCEPTIONS	ERROR에 대한 정보를 제공한다.

\*FUNCTION MODULE로부터 ABAP 프로그램에 전달받는 값

\*\*FUNCTION MODULE에 전달하고 받을 수 있다.

## FUNCTION GROUP |

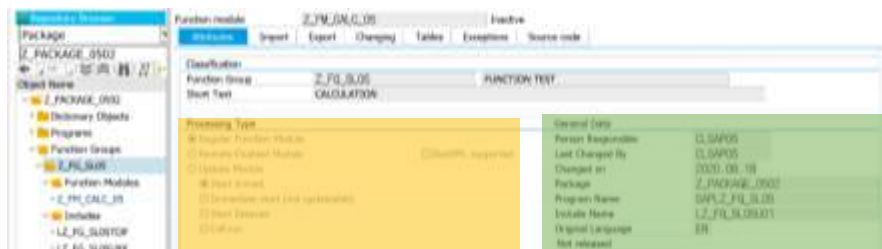
FUNCTION GROUP은 여러 FUNCTION MODULE을 모아 놓은 CONTAINER 이다. FUNCTION MODULE 자체로 실행할 수는 없으나, FUNCTION을 호출 하였을 때 SYSTEM은 호출한 프로그램의 INTERNAL SESSION 내로 FUNCTION GROUP 전체를 LOAD 한다. 이는 FUNCTION GROUP 내에서 DATA를 공유하고, 스크린을 생성하여 호출하며, PERFORM SUBROUTINE 등을 공유할 수 있도록 한다. 주의할 점은 FUNCTION 실행 시 FUNCTION에 소속된 GROUP 내의 모든 FUNCTION이 영향을 받기 때문에 하나의 FUNCTION에 ERROR가 발생할 경우 동일 GROUP 내의 모든 FUNCTION이 실행되지 않아 동일 GROUP 내에서 중요한 FUNCTION은 분리를 검토해보아야 한다.

## FUNCTION GROUP 생성 |

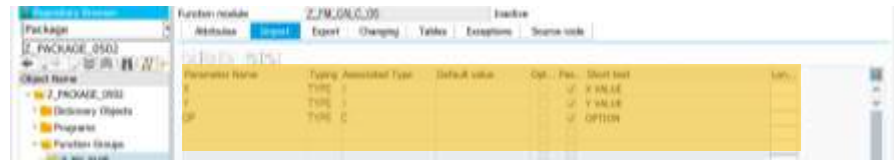
FUNCTION GROUP의 이름은 26자 까지 가능하며, FUNCTION BUILDER를 통해 FUNCTION 과 GROUP을 생성한다. 생성하면 자동으로 **MAIN PROGRAM** 과 **INCLUDE PROGRAM**을 생성하며 MAIN PROGRAM의 이름은 'SAPL' 이 FUNCTION GROUP 앞에 붙어 구성된다.

SE80 | Z\_FG\_SL05 |계산기 FUNCTION MODULE을 생성해보자 !

ATTRIBUTES | 소속GROUP, 작성자, 생성일, 패키지, PROCESSING TYPE 등



## IMPORT PARAMETERS | ABAP 프로그램 내에서 변수 값 전달받는 목적

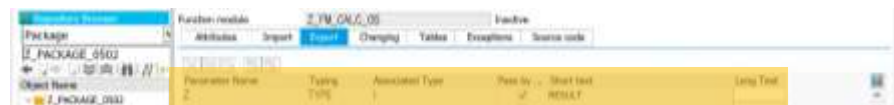


TYPE	PARAMETER의 TYPE 유형 지정*
ASSOCIATED TYPE	DATA TYPE 이나 참조할 특정 TABLE FIELD 지정
DEFAULT VALUE	PARAMETER 의 초기값 설정
OPTIONAL	PARAMETER 를 선택 사항으로 설정
PASSING VALUE	PARAMETER 전달받을 시 새 메모리에 값 복사

\*TYPE, LIKE, TYPE REF TO

\*\*입력 받을 3개의 PARAMETER인 X 와 Y, OP를 정의한다.

## EXPORT PARAMETERS | MODULE에서 ABAP 프로그램으로 값 전달 목적



## EXCEPTIONS | FUNCTION MODULE 실행 시 발생하는 예외 사항 처리



\*0으로 나누는 경우 발생하는 DUMP ERROR를 방지하기 위해 EXCEPTION을 NO\_BY\_ZERO 로 정의한다.

SOURCE CODE | 다음과 같이 계산기 기능을 수행할 코드 작성

```
IF Y = 0.
```

```
    RAISE NO_BY_ZERO. *EXCEPTION을 처리하는 코드  
ENDIF.
```

```
IF OP = '+',
```

```
    Z = X + Y.
```

```
    WRITE : 'IF',Z.
```

```
ELSEIF OP = '-',
```

```
    Z = X - Y.
```

```
    WRITE : 'IF',Z.
```

```
ELSEIF OP = '*',
```

```
    Z = X * Y.
```

```
    WRITE : 'IF',Z.
```

```
ELSEIF OP = '/',
```

```
    Z = X / Y.
```

```
    WRITE : 'IF',Z.
```

```
ELSEIF OP = '**',
```

```
    Z = X ** Y.
```

```
    WRITE : 'IF',Z.
```

```
ELSEIF OP = 'D',
```

```
    Z = X DIV Y.
```

```
    WRITE : 'IF',Z.
```

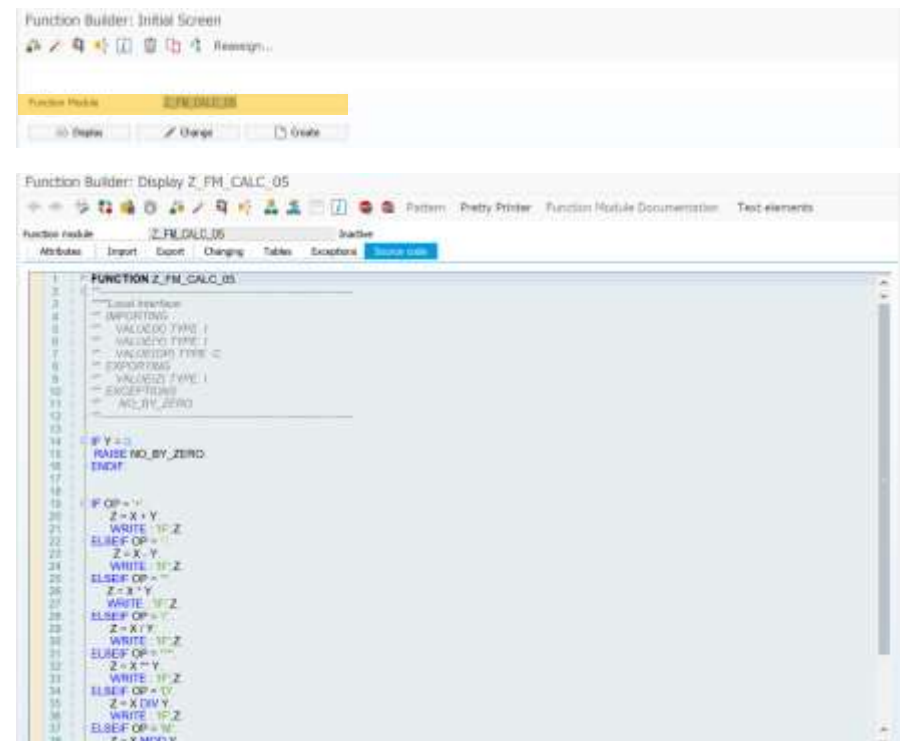
```
ELSEIF OP = 'M',
```

```
    Z = X MOD Y.
```

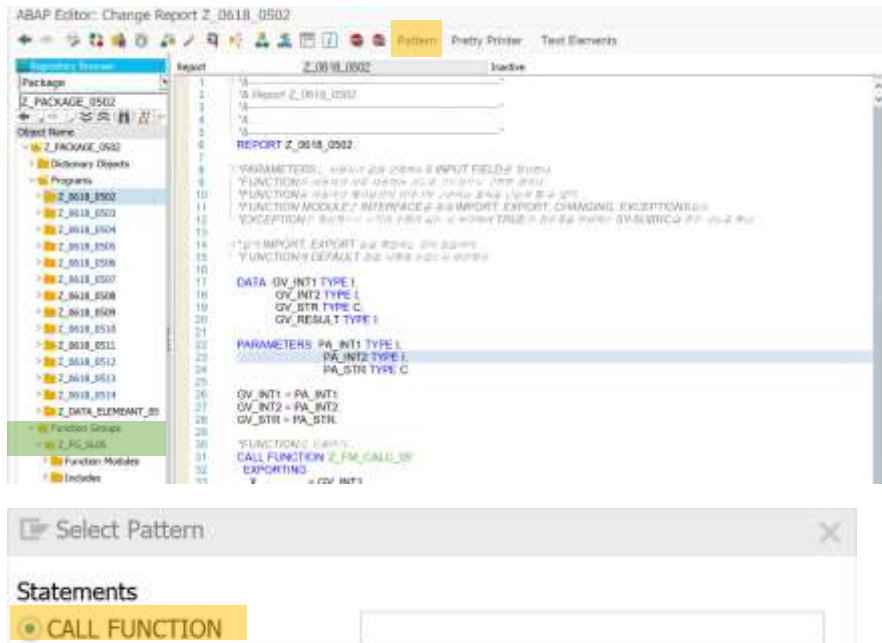
```
    WRITE : 'IF',Z.
```

```
ENDIF.
```

SE37 | 생성한 FUNCTION MODULE을 확인해 보자



## SE80 | 0502 | FUNCTION을 불러와 실행해보자



\*FUNCTION을 호출하는 방법은 생성한 FUNCTION MODULE을 드래그 하거나 상단의 PATTERN 에서 FUNCTION 이름을 입력한다.

- \*PARAMETERS는 사용자가 값을 입력하도록 INPUT FIELD를 정의한다.
- \*FUNCTION은 사용자가 자주 사용하는 기능을 코드상으로 구현한 것이다.
- \*FUNCTION을 사용하면 재사용성이 뛰어나며 구현한 로직을 단순화 할 수 있다.
- \*FUNCTION MODULE은 INTERFACE를 통해 IMPORT, EXPORT, CHANGING, EXCEPTIONS 등을 지정한다.
- \*EXCEPTION은 정상적으로 로직이 수행이 되는 지 확인하며 TRUE인 경우 0 을 반환하는 SY-SUBRC와 같은 기능을 한다.

\*값의 IMPORT, EXPORT 등을 확인하는 것이 중요하다.

\*FUNCTION에 DEFAULT 값을 지정해 놓았는지 확인하자.

DATA: GV\_INT1 TYPE I,  
GV\_INT2 TYPE I,  
GV\_STR TYPE C,  
GV\_RESULT TYPE I.

PARAMETERS: PA\_INT1 TYPE I,  
PA\_INT2 TYPE I,  
PA\_STR TYPE C.

GV\_INT1 = PA\_INT1.

GV\_INT2 = PA\_INT2.

GV\_STR = PA\_STR.

\*FUNCTION을 호출한다.

CALL FUNCTION 'Z\_FM\_CALC\_05'

EXPORTING

X = GV\_INT1  
Y = GV\_INT2  
OP = GV\_STR

IMPORTING

Z = GV\_RESULT

\*SAP에서 초기 설정해준 값,

## EXCEPTIONS

NO\_BY\_ZERO = 1

OTHERS = 2

IF SY-SUBRC = 0.

\* Implement suitable error handling here

WRITE: / GV\_RESULT.

ELSEIF SY-SUBRC = 1.

MESSAGE 'NO BY ZERO' TYPE 'E'.

ENDIF.

Report Z\_0618\_0502

RA_INT1	10
RA_INT2	2
RA_STR	

Report Z\_0618\_0502

IF

EXCEPTIONS | 0으로 나눌 때 ERROR 처리를 확인해보자.

Report Z\_0618\_0502

RA_INT1	10
RA_INT2	0
RA_STR	

All OK ZERO

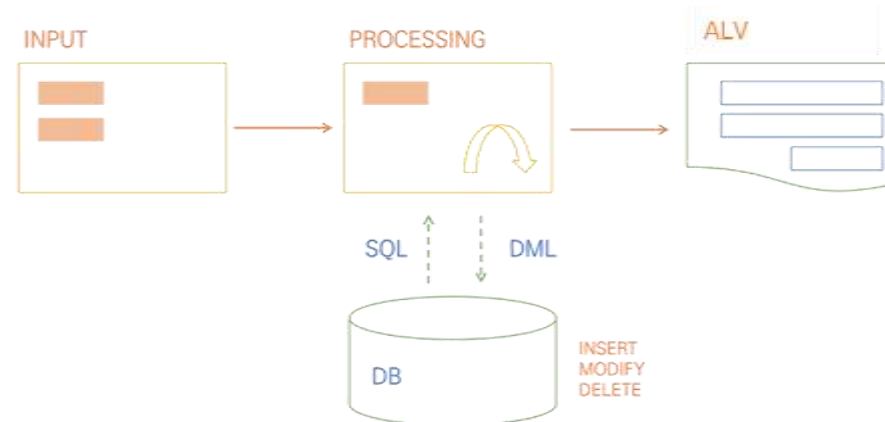
ST22 | RUNTIME ERRORS 확인할 수 있다.

ABAP Runtime Errors - All Clients

Parameters: Standard

Tabkey: 187 Runtime Errors

PROGRAM | 처리과정 \*OUTPUT은 LIST로 구성된다.



SE11 | ABAP DICTIONARY

ABAP Dictionary: Initial Screen

Database table: SCARR

View

Transparent Table: SCARR Active

Short Description: Airline

Field	Key	Init...	Data element	Data Type	Length	Decim...	Short Description
MANDT	✓	✓	S_MANDT	CLNT	3		Client
CARRID	✓	✓	S_CARRID	CHAR	3		Carrier Code
CARRNAME	✓	✓	S_CARRNAME	CHAR	20		Carrier name
CURCODE	✓	✓	S_CURCODE	CURY	5		Currency of airline

Data Browser: Table SCARR Select Entries 19

MANDT	CARRID	CARRNAME	CURCODE	URL	ZCOL1	ZCOL2
400	AA	American Airlines	USD	http://www.aa.com		
400	AB	Air Berlin	EUR	http://www.airberl		
400	AC	Air Canada	CAD	http://www.aircana		

## SELECT | READING DATA

SELECT	데이터베이스 테이블에서 데이터를 조회한다.*
INTO	SELECT에서 읽어온 데이터를 변수에 저장한다.**
FROM	SELECT 할 테이블을 지정해준다.***
WHERE	조회하고자 하는 데이터의 조건을 추가할 수 있다.
GROUP BY****	여러 라인의 결과를 그룹으로 지정해 결과를 얻는다
HAVING	GROUP의 조건을 설정하는 WHERE 구문이다
ORDER BY	조회된 데이터를 정렬 SORT 한다.

\*조회하고자 하는 테이블 필드명을 나열할 수 있으며, 한 건 또는 여러 라인 조회

\*\*해당하는 변수를 ABAP 프로그램에서 사용한다.

\*\*\*위치는 INTO 이전 / 이후 모두 둘 수 있다.

\*\*\*\*\*결과는 하나만 얻을 수 있다.

## SELECT 구문 |

SELECT <LINES> <COLUMNS> 의 형태를 가진다. <LINES>는 하나 또는 여러 라인을 선택할 시 사용되며 **하나의 라인일 때는 SELECT SINGLE** 구문을 사용한다. <COLUMNS>는 테이블 컬럼을 기술한다.

## SELECT 구문 | SINGLE LINE | SELECT SINGLE <COLS> ...

데이터 베이스에서 하나의 라인 값을 읽어올 때 사용한다. 해당 구문을 사용하면 데이터를 한 건만 가져오기 때문에 원하는 데이터의 조건을 정확히 알아야한다. 따라서, WHERE 조건에 **유일한 키 값을** 필요로 하며 **모든 컬럼을 조회하고 싶은 경우 \*** 를 사용한다.

## SELECT | SEVERAL LINES | SELECT [DISTINCT] <|COLS> ENDSELECT

여러 라인을 조회하는 경우 SELECT 결과가 내부 테이블에 저장되며 해당 테이블을 INTERNAL TABLE이라 부른다. 이는 ABAP 메모리에 생성되는 데이터를 저장할 수 있는 가상의 테이블이다. SEVERAL LINE을 조회할 때 SELECT DISTINCT와 같이 **DISTINCT를 사용하면 중복된 값을 제외한다.** 또한 INTO 구문의 결과가 저장되는 곳이 필드나 WORK AREA 일 경우 마지막에 ENDSELECT를 사용해야한다. ENDSELECT는 하나의 값을 읽어 구조에 삽입하고 조건에 해당하는 값을 모두 읽어 올 때까지 LOOP를 수행한다. AS | ALIAS를 사용하여 컬럼 명에 별명을 지정할 수도 있으며 SELECT 구문의 컬럼을 동적으로 선언\*할 수도 있다. 동적인 컬럼에 SELECT 구문을 사용하는 라인의 구조체가 NULL 이면 '\*' 와 같은 구문이 된다.

\*동적인 컬럼은 TABLE을 STRUCTURE로 구성하는 LIKE LINE OF 와 같은 구문을 사용하여 지정한 DATA OBJECT가 해당한다.

## AS : ALIAS | SELECT <COLS> [AS <ALIAS>]...

AS 구문을 사용하여, 컬럼 명에 별명을 지정할 수 있다.

## 동적인 SELECT 구문 | SELECT <LINES> (<ITAB>)

SELECT 구문의 컬럼을 동적으로 선언할 수 있다.

## INTO |

INTO는 SELECT 구문에서 조회한 결과 값을 변수에 저장하는 기능을 수행한다.

## INTO | WORKAREA | SELECT ... INTO [CORRESPONDING FIELDS OF]

구조체 | WORK AREA와 INTERNAL TABLE, SINGLE FIELD이 INTO 구문을 사용한다. 여러 칼럼의 한 라인만 조회하고자 하는 경우 변수나 구조체인 WORK AREA에 값을 할당한다. 애스터리스크인 \*를 사용하면 전체 컬럼의 값을 읽어오며, CORRESPONDING FIELDS OF 를 사용하여 한 번에 WORK AREA의 동일 필드명에 값을 할당한다. \* 의 사용은 개별 필드의 SELECT보다 비효율적이며, 많은 TABLE을 가지고 있는 주요 TABLE에 대해 사용하는 경우 성능에 큰 영향을 미친다. 따라서 SELECT \* 구문의 사용을 삼가는 것이 바람직하다.

## INTO | INTERNAL TABLE | SELECT ...INTO | APPENDING [CORRESPONDING FIELDS OF] TABLE <ITAB> [PACKAGE SIZE <N>]

여러 라인을 조회할 경우 INTERNAL TABLE을 사용한다. APPENDING은 INTERNAL TABLE에 추가로 INSERT하고, INTO는 INTERNAL TABLE의 DATA를 삭제한 다음 INSERT한다. PACKAGE SIZE는 INTERNAL TABLE에 몇 개의 라인을 추가할 것인지 설정하며 만약 PACKAGE SIZE 가 5라면 5개의 값을 여러 번 읽어와 INTERNAL TABLE에 추가하게 되며, END SELECT 구문을 반드시 사용해야 한다.

## INTO | SINGLE FIELD | SELECT ... INTO (F1, F2, F3)

TABLE의 개별 컬럼을 조회하거나 AGGREGATE 함수를 사용하는 경우 다음 구문과 같이 사용한다. INTO 구문 다음에 두 개 이상의 TARGET이 필요한 경우에는 괄호와 변수명을 붙여 쓰며, 공백이 존재하면 SYNTAX ERROR가 발생한다. 만약 SELECT 구문에서 2개의 FIELD가 필요한 경우 `SELECT CARRID CONNID INTO (GV_CARRID, GV_CONNID) FROM SFLIGHT.` 와 같이 사용한다.

## FROM | SELECT ... FROM TABLE OPTION ...

FROM 구문은 데이터를 SELECT할 대상 테이블 또는 뷰를 지정한다. FROM 구문 다음에는 하나의 테이블을 지정하거나 여러 개의 테이블을 JOIN할 수 있다. ALIAS를 사용하여 TABLE 명에 별명을 붙일 수 있으며, TABLE 이름을 동적으로 선언할 수 있다. FROM 구문은 TABLE을 정의하는 부분과 DB 접근을 컨트롤 하는 OPTION으로 나누어진다.

## FROM | OPTION |

CLIENT SPECIFIED	자동 CLIENT 설정을 해제한다
BYPASSING BUFFER	SAP LOCAL BUFFER에서 값을 읽지 않는다*
UP TO N ROWS	SELECT ROW 개수를 제한한다**

\*테이블이 BUFFERING이 설정되어 있더라도 바로 DB TABLE에서 SELECT를 수행한다.

\*\*조회 조건에 날짜를 입력하지 않는 것과 같이 사용자 실수로 대량의 데이터를 요청하는 경우 DB 성능 저하를 예방할 수 있다.

## TABLE 선택 |

정적인 TABLE 선택	하나의 TABLE을 정적으로 선언할 때 사용*
동적인 TABLE 선택	테이블 이름을 동적으로 선언하여 사용가능**

\*ALIAS를 사용할 수 있으나 해당 경우 TABLE명을 SELECT에 사용할 수 없다.

\*\*TABLE의 이름은 반드시 대문자로 지정하며, ABAP DICTIONARY 에 존재하는 이름이어야 한다.

SE80 | 0503 | SELECT 구문을 사용해보자.

*\*SELECT \* 모두 가져와라*

*\*FROM DB TABLE INTO DATA OBJECT WHERE 조건*

*\*SCARR TABLE에서 DATA를 가져와보자.*

*\*DATA를 받아 올 STRUCTURE를 지정하자.*

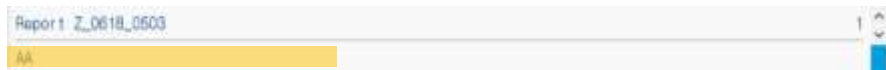
DATA: GS\_SCARR TYPE SCARR.

*\*SELECT SINGLE 은 한 행의 값 만을 선택한다.*

```
SELECT SINGLE * FROM SCARR  
      INTO GS_SCARR.
```

*\*만약 한 칼럼의 값 만을 가져오고 싶다면 - 을 이용해 GS\_SCARR-CARRID.*

WRITE: GS\_SCARR-CARRID.



SE80 | 0504 | SELECT SINGLE 구문을 사용해보자.

*\*SCARR TABLE에서 DATA를 가져와보자.*

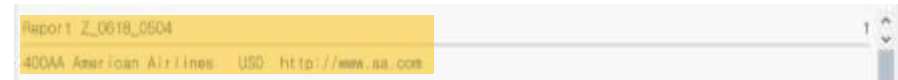
DATA: GS\_SCARR TYPE SCARR.

*\*SELECT SINGLE은 한 개의 ROW만 가져온다.*

```
SELECT SINGLE * FROM SCARR
```

```
      INTO GS_SCARR.
```

WRITE: GS\_SCARR.



WHERE 구문 | SELECT ... WHERE <S> <OPERATOR> <F> ...

WHERE 조건은 SELECT 적중 수를 줄이며, 사용자가 원하는 데이터를 정확하게 선택할 수 있는 조건으로 WHERE 조건에 사용되는 필드가 INDEX 에서 사용 될 때 빠른 성능을 보장하며 UPDATE, DELETE와 같은 명령어에도 사용된다.

WHERE 구문 | 연산자 |

EQ	같다
=	같다
NE	같지 않다
<>	같지 않다
><	같지 않다
LT	보다 작다
<	보다 작다
LE	작거나 같다
<=	작거나 같다
GT	보다 크다
>	보다 크다
GE	크거나 같다
>=	크거나 같다



**WHERE | INTERVAL 조건** | SELECT ... WHERE <S> [NOT] BETWEEN <F1> AND <F2> INTERVAL 조건은 조건에 범위 값을 사용할 때 사용한다.

**WHERE | STRING 비교** | SELECT ~ WHERE COL2 LIKE 'ABC%'

문자열을 비교할 때 LIKE 구문을 사용한다. 만약, ABC로 시작하는 ABCD, ABCE 와 같이 한자리만 비교할 경우 '-' 를 사용하여 WHERE COL2 LIKE 'ABC\_' 와 같이 표기한다.

**WHERE | LIST VALUE** | SELECT ... WHERE [NOT] IN ( <F1> ,,, <FN> )

IN 구문을 사용하여, 여러 조건에 속한 경우의 값을 가져온다.

**WHERE | SELECTION TABLE** | SELECT ... WHERE <S> [NOT] IN <STAB>

IN 구문을 사용하여 SELECTION TABLE, RANGE 변수에 존재하는 값들을 조회할 수 있다. SELECTION TABLE, RANGE 변수는 INTERNAL TABLE과 유사하게 여러 ROW를 저장할 수 있는 변수 이다.

**WHERE | DANAMIC** | SELECT ... WHERE ( <ITAB> ) ...

SELECT 구문의 조건을 설정하는 WHERE 구문을 동적으로 구성할 수 있다.

**WHERE | FOR ALL ENTRIES** | SELECT ... FOR ENTRIES IN <ITAB> WHERE <COND>

FOR ALL ENTRY 구문은 INTERNAL TABLE과 DB TABLE을 JOIN하는 개념과 유사하며, LOOP를 수행하면서 SQL을 수행한다. DB에 반복적으로 접근해 JOIN보다는 비효율적이거나 ABAP에서 유용하게 활용된다. FOR ENTRIES를 사용할 경우 주의할 점은 ITAB의 칼럼과 비교 대상 TABLE의 칼럼 타입은 같아야 하며, LIKE, BETWEEN, IN 과 같은 비교 구문은 사용할 수 없다. ITAB의 중복된 값은 UNIQUE KEY값을 기준으로 하나만 남으며 ITAB이 NULL인 경우 모든 DATA를 읽는다. 만약 ITAB의 수가 3개이면 SELECT ~ END SELECT를 3번 수행하는 것과 동일 함으로 ITAB의 수가 많으면 LOOP의 수가 증가해 속도가 감소한다.

**SE80 | 0505** | SELECT 구문의 WHERE 조건을 사용해보자.

**DATA** GS\_SCARR **TYPE** SCARR.

**PARAMETERS:** PA\_SCARR **TYPE** SCARR.

*\*데이터 가져올 때는 SELECT 구문*

*\*SELECT SINGLE 은 한 행의 값 만을 선택한다.*

*\*중복 값이 있는 경우*

*\*WHERE 조건*

**SELECT SINGLE \***

**FROM** SCARR

**INTO** GS\_SCARR

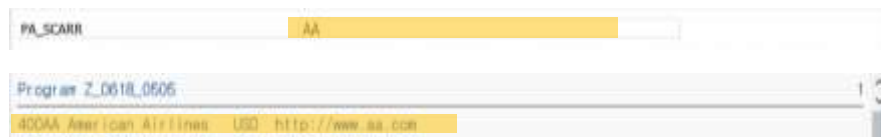
**WHERE** CARRID = PA\_SCARR.

IF SY-SUBRC = 0.

WRITE: GS\_SCARR.

ENDIF.

*\*만약 한 칼럼의 값 만을 가져오고 싶다면 - 을 이용해 GS\_SCARR-CARRID.*



SE80 | 0506 | SELECT \* 구문을 사용해보자.

DATA: GS\_SCARR TYPE SCARR.

DATA: GT\_TABLE TYPE TABLE OF SCARR.

DATA GT\_TABLE2 LIKE TABLE OF SCARR.

*\*데이터 가져올 때는 SELECT 구문*

*\*SELECT SINGLE 은 한 행의 값 만을 선택한다.*

*\*중복 값이 있는 경우*

*\*WHERE 조건*

SELECT \*

FROM SCARR

INTO TABLE GT\_TABLE .

BREAK-POINT.

Row	MANDT	CARRID	CARRNAME	CARRCODE	URL
1	400	AA	American Airlines	USD	http://www.aa.com
2	400	AC	Air Canada	CAD	http://www.aircanada
3	400	AF	Air France	EUR	http://www.airfrance
4	400	AZ	Alitalia	EUR	http://www.alitalia
5	400	BA	British Airways	GBP	http://www.british-airways
6	400	FJ	Air Pacific	USD	http://www.airpacific
7	400	CO	Continental Airlines	USD	http://www.continental
8	400	DL	Delta Airlines	USD	http://www.delta-air
9	400	AB	Air Berlin	EUR	http://www.airberlin
10	400	LH	Lufthansa	EUR	http://www.lufthansa
11	400	NG	Lauda Air	EUR	http://www.laudair
12	400	JL	Japan Airlines	JPY	http://www.jal.co.jp
13	400	NH	Northwest Airlines	USD	http://www.nwa.com
14	400	QF	Qantas Airways	AUD	http://www.qantas.co
15	400	SA	South African Air	ZAR	http://www.saa.co.za
16	400	SQ	Singapore Airlines	SGD	http://www.singapore
17	400	SR	Swire	CHF	http://www.swire.com
18	400	UA	United Airlines	USD	http://www.ual.com
19	400	KO	AIR KOREA	USD	HTTP://WWW.AIRKOREA

SE80 | 0507 | SELECT 조건 구문을 사용해보자.

DATA: GS\_SCARR TYPE SCARR.

DATA: GT\_TABLE TYPE TABLE OF SCARR.

DATA GT\_TABLE2 LIKE TABLE OF GS\_SCARR.

DATA: BEGIN OF GS\_STR,

CARRID TYPE SCARR-CARRID ,

CARRNAME TYPE SCARR-CARRNAME ,

END OF GS\_STR.

\*데이터 가져올 때는 SELECT 구문

\*SELECT SINGLE 은 한 행의 값 만을 선택한다.

\*중복 값이 있는 경우

\*WHERE 조건

```
SELECT CARRID CARRNAME  
FROM SCARR  
INTO TABLE GT_TABLE .
```

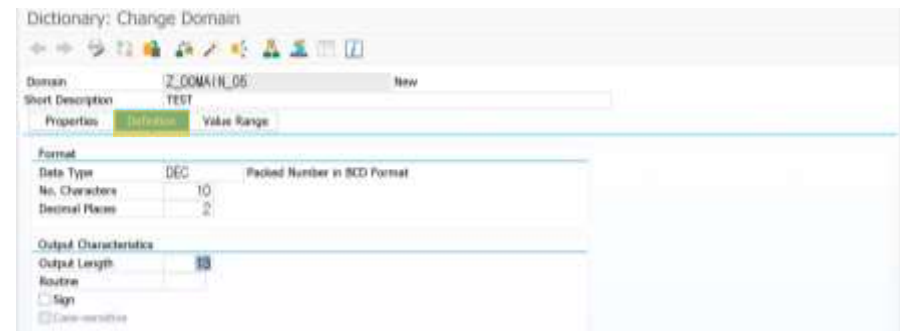
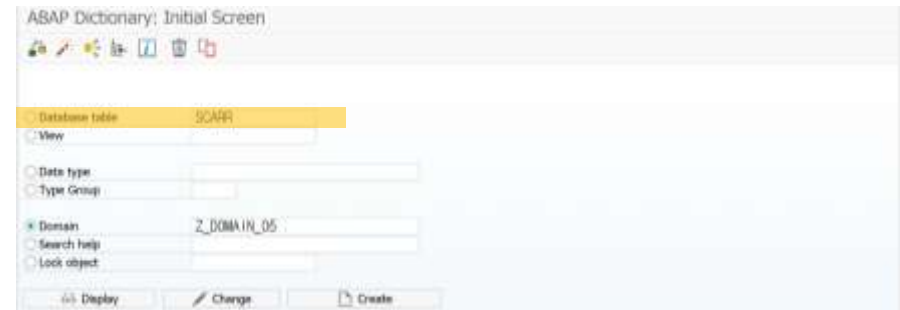
BREAK-POINT.



Row	MANOT [C(3)]	CARRID [C(3)]	CARRNAME [C(20)]	CURRCODE [C(5)]	URL [C(255)]	ZCOL1 [C(1)]
1	AA	Air				
2	AC	Air				
3	AF	Air				
4	AZ	Air				
5	BA	Br				
6	FJ	Air				
7	GO	Com				
8	OL	Del				
9	AB	Air				
10	LH	Luf				
11	NG	Lau				
12	JL	Jap				
13	NW	Nor				
14	GF	Qan				
15	SA	Sou				
16	SD	Sin				
17	SR	Swi				
18	UA	Uni				
19	KD	Air				

SE11 | ABAP DICIONARY에서 NESTED\*\* 구조를 생성할 수 있으며 F1 도움말, F4를 통해 구조화를 확인할 수 있다.

\*\*SELECT ~ ENDSELECT 구문 내에 사용된 중첩된 SELECT를 말한다.



INTERNAL TABLE | 프로그램 내에서 정의하여 사용할 수 있는 LOCAL TABLE

INTERNAL TABLE은 ABAP 프로그램에서 가장 강력한 기능과 편의성을 제공하며 ABAP 프로그램의 꽃이라 한다. 이는 ABAP 핵심 기술로 단일 프로그램 내에서 정의하여 사용할 수 있는 LOCAL TABLE로 프로그램 개발 및 유지보수를 좀 더 쉽고 편리하게 할 수 있다.

## INTERNAL TABLE | DYNAMIC DATA OBJECT

INTERNAL TABLE은 DYNAMIC DATA OBJECT 동적인 구조체 배열로, TYPE 구문을 기반으로 INITIAL SIZE 구문을 사용하여 테이블 크기만을 선언하고 MEMORY에 LOAD 하지 않음을 의미한다. 즉, TABLE의 형태인 크기만을 선언하기 때문에 INSERT 또는 APPEND 를 사용하여 LINE이 추가될 때마다 MEMORY에 LOAD 한다. 이는 INITIAL SIZE RNANSDL 실제로 MEMORY 공간을 할당하는 것이 아닌 RESERVE 예약을 하는 것임을 의미한다. 즉 INTERNAL TABLE은 할당과 추가 APPEND가 이루어져야 한다.

## INTERNAL TABLE | 생성

LOCAL TABLE TYPE	개별 프로그램에만 사용
GLOBAL ABAP DICTIONARY	ABAP DICTIONARY나 구조체 참조

## INTERNAL TABLE | LOCAL TABLE TYPE

LOCAL TABLE TYPE을 이용한 INTERNAL TABLE 생성에는 먼저 TYPES: BEGIN OF T\_STR ~ END OF T\_STR. 구문을 이용하여 구조체 타입을 선언하고, 이를 참조하여 TYPES T\_ITAB TYPE STANDARD TABLE OF T\_STR ~. 과 같은 형태로 구조체 타입을 참조하는 INTERNAL TABLE TYPE을 선언한다. 세 번째로는 이 타입을 참고하여 DATA: GT\_ITAB TYPE T\_ITAB 과 같이 INTERNAL TABLE을 생성한다. TYPE 대신에 LIKE를 사용하기도 한다.

## INTERNAL TABLE | GLOBAL TABLE TYPE

ABAP DICTIONARY 나 구조체를 참조하여 INTERNAL TABLE을 생성한다.

## SE80 | 0508 | LOOP를 사용하여 INTERNAL TABLE 에 값을 할당해보자.

*\*데이터 가져올 때는 SELECT 구문*

*\*SELECT SINGLE 은 한 행의 값 만을 선택한다.*

*\*중복 값이 있는 경우*

*\*WHERE 조건*

*\*LOOP SMS ABAP TABLE 에서 값을 사용하는 방법.*

```
DATA: BEGIN OF GS_STR,  
      CARRID TYPE SCARR-CARRID,  
      CARRNAME TYPE SCARR-CARRNAME,  
END OF GS_STR.
```

```
DATA: GT_TAB LIKE TABLE OF GS_STR.
```

```
SELECT CARRID CARRNAME  
FROM SCARR  
INTO TABLE GT_TAB.
```

*\*이부분이 없으면 디렉터리에 존재하는 정보만 가져옴*

```
GS_STR-CARRID = 'ZZ'.
```

```
GS_STR-CARRNAME = 'ASDFGJL'.
```

```
APPEND GS_STR TO GT_TAB.
```

*\*스트럭처의 내용을 GT\_TAB에 추가하겠다는 의미*

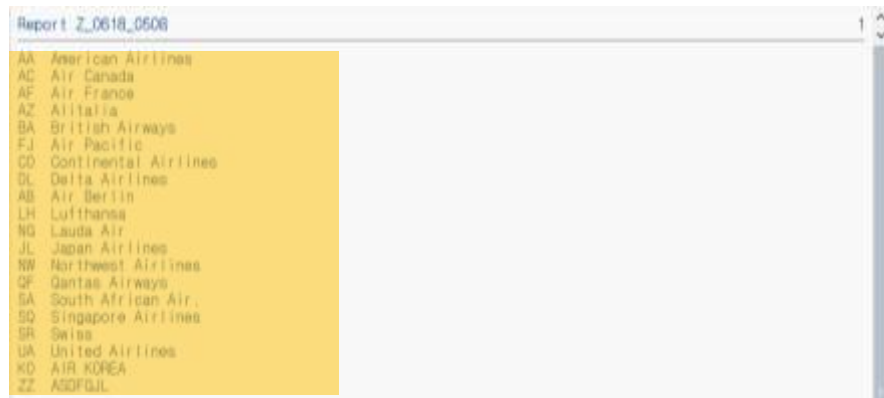
*\*TABLE 값을 WORK AREA로 만드는 과정을 반복.*

*\*LOOP 는 반복문으로 STRUCTURE = WORK AREA 가 해당한다.*

```
LOOP AT GT_TAB INTO GS_STR.
```

```
WRITE:/ GS_STR-CARRID, GS_STR-CARRNAME.
```

```
ENDLOOP.
```



Report Z_0618_0508
AA American Airlines
AC Air Canada
AF Air France
AZ Alitalia
BA British Airways
FJ Air Pacific
CD Continental Airlines
DL Delta Airlines
AB Air Berlin
LH Lufthansa
NG Lunda Air
JL Japan Airlines
NW Northwest Airlines
QF Qantas Airways
SA South African Air
SQ Singapore Airlines
SR Swiss
UA United Airlines
KO AIR KOREA
ZZ ASDFGJL

SE80 | 0509 | INTERNAL TABLE을 생성하여 TABLE에 내역을 추가해 보자.

```
DATA: BEGIN OF GS_STR,
```

```
    CARRID TYPE SCARR-CARRID,
```

```
    CARRNAME TYPE SCARR-CARRNAME,
```

```
END OF GS_STR.
```

```
DATA: GT_TAB LIKE TABLE OF GS_STR.
```

```
SELECT CARRID CARRNAME
```

```
FROM SCARR
```

```
INTO TABLE GT_TAB.
```

```
GS_STR-CARRID = 'ZZ'.
```

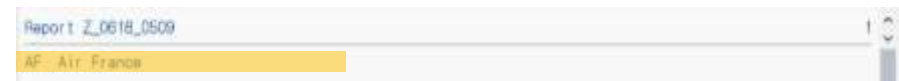
```
GS_STR-CARRNAME = 'ASDFGJL'.
```

```
APPEND GS_STR TO GT_TAB. "스트럭처의 내용을 GT_TAB 마지막 줄에 추가
```

```
CLEAR GS_STR. "CLEAR을 사용하면 내용이 초기화가 됨"
```

```
READ TABLE GT_TAB INTO GS_STR INDEX 3. "3번째 ROW의 값을 읽겠다
```

```
WRITE:/ GS_STR-CARRID, GS_STR-CARRNAME.
```



Report Z_0618_0509
AF Air France

SE80 | 0510 | INTERNAL TABLE을 생성하여 TABLE에 내역을 추가해 보자.

*\*데이터 가져올 때는 SELECT 구문*

*\*SELECT SINGLE 은 한 행의 값 만을 선택한다.*

*\*중복 값이 있는 경우*

*\*WHERE 조건*

*\*LOOP SMS ABAP TABLE 에서 값을 사용하는 방법.*

```
DATA: BEGIN OF GS_STR,
```

```
    CARRID TYPE SFLIGHT-CARRID,
```

```
    FLDATE TYPE SFLIGHT-FLDATE,
```

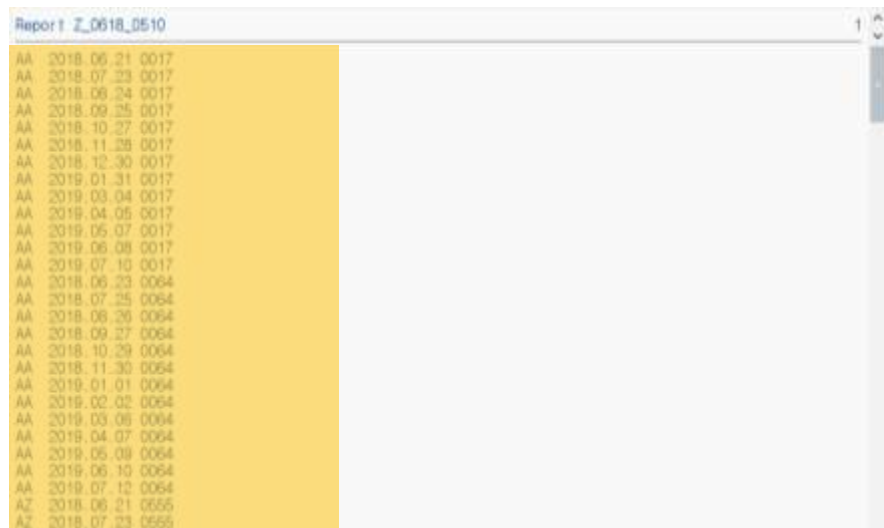
```
CONNID TYPE SFLIGHT-CONNID,  
END OF GS_STR.
```

```
DATA: GT_TAB LIKE TABLE OF GS_STR.
```

```
SELECT CARRID FLDATE CONNID  
FROM SFLIGHT  
INTO TABLE GT_TAB.
```

*\*TABLE 값을 WORK AREA로 만드는 과정을 반복.*

```
LOOP AT GT_TAB INTO GS_STR.  
WRITE:/ GS_STR-CARRID, GS_STR-FLDATE, GS_STR-CONNID.  
ENDLOOP.
```



Report: Z_0018_0510
AA 2018.06.21 0017
AA 2018.07.23 0017
AA 2018.08.24 0017
AA 2018.09.25 0017
AA 2018.10.27 0017
AA 2018.11.28 0017
AA 2018.12.30 0017
AA 2019.01.31 0017
AA 2019.03.04 0017
AA 2019.04.05 0017
AA 2019.05.07 0017
AA 2019.06.08 0017
AA 2019.07.10 0017
AA 2018.06.23 0064
AA 2018.07.25 0064
AA 2018.08.26 0064
AA 2018.09.27 0064
AA 2018.10.29 0064
AA 2018.11.30 0064
AA 2019.01.01 0064
AA 2019.02.02 0064
AA 2019.03.06 0064
AA 2019.04.07 0064
AA 2019.05.08 0064
AA 2019.06.10 0064
AA 2019.07.12 0064
AZ 2018.06.21 0555
AZ 2018.07.23 0555

## INTERNAL TABLE | HEADER LINE

모자 아이콘이 보이는 라인은 헤더 라인이라 하며 WORK AREA로 지칭하기도 한다. 이는 INTERNAL TABLE 생성시 다음과 같이 DATA ITAB TYPE OBJ [ WITH HEADER LINE ]. 구문을 통해 생성하며 INTERNAL TABLE이 LOOP 를 처리하며 개별 LINE으로 이전한다. DATA ITAB TYPE TABLE OF T\_STR WITH HEADER LINE과 같이 HHEADER LINE 이 있는 INTERNAL TABLE을 정의하면 헤더 라인에 담긴 정보를 바로 사용할 수 있다.

## INTERNAL TABLE | HEADER LINE 유무 비교

만약 INTERNAL TABLE의 LOOP 구문에서 HEADER LINE이 없다면 WORK AREA를 선언하고 나서 값을 복사한 다음 사용해야 한다. HEADER LINE 이 있다면 INTERNAL TABLE의 이름은 ABAP PROGRAM 내에서 HEADERT LINE 을 의미하게 된다.

HEADER LINE이 없는 경우	HEADER LINE이 있는 경우
모든 INTERNAL TABLE   STANDARD, SORTED, HASHED TYPE	
INSERT WA INTO TABLE ITAB.	INSERT TABLE ITAB.
COLLECT WA INTO ITAB.	COLLECT ITAB.
READ TABLE ITAB INTO WA.	READ TABLE ITAB...
MODIFY TABLE ITAB FROM WA...	MODIFY TABLE ITAB...
MODIFY ITAB FROM WA... WHERE...	MODIFY ITAB... WHERE...
DELETE TABLE ITAB FROM WA.	DELETE TABLE ITAB.
LOOK AT ITAB INTO WA...	LOOK AT ITAB...
INDEX TABLE   STANDARD, SORTED, HASHED	
APPEND WA INTO ITAB.	APPEND ITAB.
INSERT WA INTO ITAB...	INSERT ITAB...
MODIFY ITAB FROM WA...	MODIFY ITAB...

HEADER LINE 이 있는 INTERNAL TABLE에서 APPEND 구문은 다음과 같이

```
APPEND GT_ITAB = APPEND GT_ITAB TO GT_ITV
```

HEADER LINE 정보를 생략한 것과 동일하다. 즉, INTERNAL TABLE에 HEADER LINE이 존재하지 않으면, WORK AREA를 선언하여 TABLE을 읽거나 변경할 수 있다. ABAP 언어는 OBJECT-ORIENTED 개념이 도입되면서 클래스 내부에서 HEADER LINE이 지원되지 않으며 OCCURS 구문을 포함하여 HEADER LINE이 있는 INTERNAL TABLE을 사용하지 말 것을 권장하고 있다. 그러나 기존 프로그램에서 많이 사용되고, 편리한 기능으로 인해 실무에서 여전히 활용하고 있다.

SE80 | 0511 | HEADER LINE이 없는 TABLE에 대해 알아보자.

```
*LOCAL DATA TYPE
```

```
TYPES: BEGIN OF TS_STR,  
        NO(2) TYPE C,  
        NAME(8) TYPE C,  
        PART(10) TYPE C,  
END OF TS_STR.
```

```
DATA GS_STR TYPE TS_STR.
```

*\*HEADER가 없는 INTERNAL TABLE로 GT\_ITAB이 WORK AREA가 된다.*

*\*HEADER가 없는 TABLE에 DATA를 담을 때에는 WORK AREA에 DATA를 담  
고*

```
DATA GT_ITAB LIKE TABLE OF GS_STR.
```

```
GS_STR-NO = '10'.
```

```
GS_STR-NAME = 'ABAP'.
```

```
GS_STR-PART = 'PART'.
```

```
APPEND GS_STR TO GT_ITAB.
```

```
GS_STR-NO = '20'.
```

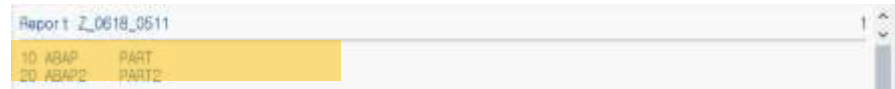
```
GS_STR-NAME = 'ABAP2'.
```

```
GS_STR-PART = 'PART2'.
```

```
APPEND GS_STR TO GT_ITAB.
```

```
LOOP AT GT_ITAB INTO GS_STR.
```

```
    WRITE: / GS_STR-NO, GS_STR-NAME, GS_STR-PART.  
ENDLOOP.
```



10	ABAP	PART
20	ABAP2	PART2

SE80 | 0512 | HEADER LINE이 있는 TABLE에 대해 알아보자.

```
*LOCAL DATA TYPE
```

```
TYPES: BEGIN OF TS_STR,  
        NO(2) TYPE C,  
        NAME(8) TYPE C,  
        PART(10) TYPE C,
```

END OF TS\_STR.

DATA GS\_STR TYPE TS\_STR.

*\*HEADERLINE이 있는 TABLE*

DATA GT\_ITAB2 LIKE TABLE OF GS\_STR WITH HEADER LINE.

GT\_ITAB2-NO = '10'.

GT\_ITAB2-NAME = 'ABAP'.

GT\_ITAB2-PART = 'PART'.

*\*INTO GT\_ITAB2가 생략되어 표현할 수 있다.*

APPEND GT\_ITAB2.

GT\_ITAB2-NO = '20'.

GT\_ITAB2-NAME = 'ABAP2'.

GT\_ITAB2-PART = 'PART2'.

*\*INTO GT\_ITAB2가 생략되어 표현할 수 있다.*

APPEND GT\_ITAB2.

LOOP AT GT\_ITAB2.

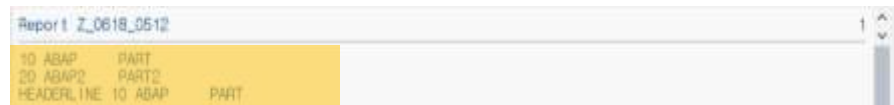
WRITE: / GT\_ITAB2-NO, GT\_ITAB2-NAME, GT\_ITAB2-PART.

ENDLOOP.

*\*INDEX는 지정한 라인만 확인할 수 있다.*

READ TABLE GT\_ITAB2 INDEX 1.

WRITE: / 'HEADERLINE', GT\_ITAB2-NO, GT\_ITAB2-NAME, GT\_ITAB2-PART.



Report: Z_0618_0512
10 ABAP PART
20 ABAP2 PART2
HEADERLINE 10 ABAP PART

INDEX |

INDEX는 DATA의 빠른 접근을 위해 TABLE과 별개의 물리적인 저장 공간에 목록 INDEX를 작성한 것을 말하며, SQL의 성능 향상을 위해 사용한다. INDEX는 SELECT, UPDATE, DELETE 구문에서 사용하며 테이블 데이터에 접근하는 시간을 절약할 수 있도록 도와주는 정렬되어 있는 복사본 테이블이라 정의할 수 있다. SE11에서 TABLE을 활성화하면 PK | PRIMARY KEY 기준으로 0 이라는 INDEX가 자동 생성되며 DB에 INDEX TABLE이 생성된다. TABLE에는 각각 DATA의 위치 번호를 가지는 ROWID 정보가 있으며, INDEX TABLE에서 ROWID를 통해 값을 읽게 된다. INDEX를 이용해 TABLE에 접근하게 되면, INDEX FIELD 기준으로 DATA가 정렬되어 있기 때문에 SELECT 결과 정렬된 DATA 가 출력된다.

INDEX | INDEX를 사용하는 것은 항상 효율적인 것은 아니다.

그 이유는, TABLE에서 추출해야 할 DATA 비율이 높다면 INDEX 목록 TABLE에 접근한 후 TABLE에서 DATA를 가져오는데 중복 시간이 들어가기 때문이다. 이 경우, INDEX TABLE을 이용하지 않고, 바로 사원 TABLE에 접근하여 처음부터 DATA를 읽는 것이 더 효율적일 수 있다.



## INDEX | ABAP DICTIONARY

ABAP DICTIONARY에 추가한 INDEX는 물리적으로 ORACLE과 같은 DB에 INDEX를 생성한다. 이는 SAP 내부에 INDEX라는 OBJECT는 존재하지 않으며, 단순히 ABAP DICTIONARY라는 통로를 통해 DB INDEX를 대체함을 의미한다. ABAP DICTIONARY에는 PRIMARY INDEX와 SECONDARY INDEX 두가지가 존재한다. PRIMARY INDEX는 TABLE의 KEY FIELD로 활성화 시 자동 생성되며 SECONDARY INDEX를 추가로 생성할 수 있다. SECONDARY INDEX를 생성하며 TABLE을 읽는 속도는 증가하나 DATA 추가시에는 속도가 감소하게 되므로 신중하게 선택해야 한다. SECONDARY INDEX는 SELECT 구문의 WHERE 조건에 자주 사용되며, 다른 TABLE과 연관성이 많은 필드를 선택하는 것이 바람직하다. INDEX 이외에 INTERNAL TABLE의 정확한 사용과 JOIN과 같은 프로그램 튜닝을 통해 충분한 성능을 발휘할 수 있다.

SE80 | 0513 | HEADER LINE이 있는 TABLE에 대해 알아보자.

```
DATA: BEGIN OF GS_STR,  
      CARRID TYPE SCARR-CARRID,  
      CARRNAME TYPE SCARR-CARRNAME,  
      CURRCODE TYPE SCARR-CURRCODE,  
END OF GS_STR.
```

```
DATA: GT_TABLE_H LIKE TABLE OF GS_STR WITH HEADER LINE.
```

```
GT_TABLE_H-CARRID = 34.  
GT_TABLE_H-CARRNAME = 'ABAP'.  
GT_TABLE_H-CURRCODE = 'QWERT'.
```

```
APPEND GT_TABLE_H.
```

```
READ TABLE GT_TABLE_H INDEX 1.
```

```
WRITE: / 'HEADERLINE', GT_TABLE_H-CARRID, GT_TABLE_H-  
CARRNAME, GT_TABLE_H-CURRCODE.
```



## METHOD | 선언

METHOD는 CLASS의 구성 요소로 IMPORTING, EXPORTING, CHANGING, RETURNING을 이용하여 PARAMETER INTERFACE를 정의할 수 있다. METHOD는 PARAMETER의 참조주소 REFERENCE와 값 VALUE를 선택하여 사용할 수 있으며 FUNCTION MODULE이 EXPORT/IMPORT/CHANGING 매개변수를 주고받는 것과 같이, CLASS METHOD에서 EXPORT/IMPORT/CHANGING 구문을 사용한다. 값을 매개 변수로 넘겨주려면 VALUE 구문을 선언한다. RETURN VALUE는 항상 값으로 반환되며, 함수 모듈과 같이 예외 처리 시 EXCEPTIONS를 사용할 수 있다.

## METHOD | 구현 | METHOD ~. ENDMETHOD.

CLASS에서 선언된 메소드를 IMPLEMENTATION에서 선언하고 기능을 구현한다. IMPLEMENTATION는 METHOD의 기능을 구현하는 것을 의미한다. METHOD를 구현할 때 CLASS의 METHOD 선언 부분에 정의된 것 이외의 INTERFACE PARAMETER를 정의할 필요는 없으며, METHOD 내 DATA 구문을 이용하여 지역 변수를 정의하면 된다. RAISE EXCEPTION과 MESSAGE RAISING을 통해 에러를 처리할 수 있으며, INSTANCE METHOD가 STATIC과

INHERITANCE 속성에 모두 작동되는 것과 달리 STATIC METHOD를 사용할 때는 CLASS의 STATIC 속성에서만 적용될 수 있으니 주의해야 한다.

**METHOD | 호출 | CALL METHOD ~ EXPORTING ~ IMPORTING ~ CHANGING ~ RECEIVING ~ EXCEPTIONS ~.**

<b>METHOD 내부에서 호출</b>	CALL METHOD ~.
<b>CLASS 외부에서 METHOD 호출</b>	CALL METHOD REF =>~.

REF는 CLASS 의 INSTANCE를 가리키는 값을 가진 객체 참조 변수로, METHOD를 호출할 때 EXPORTING 또는 CHANGING 구문과 같은 필수 엔트리는 반드시 선언해야 한다. IMPORTING 또는 RECEIVING, 예러 처리를 위한 EXCEPTIONS 구문은 반드시 사용해야하는 것은 아니다. 만약 IMPORTING 파라미터 하나로 구성된 METHOD를 호출하려면 다음과 같은 구문을 사용한다.

**CALL METHOD METHOD ( F ).**

여러 개의 IMPORTING PARAMETER로 구성된 경우는 다음과 같이 기술한다.

**CALL METHOD METHOD ( I1 = F1 I2 = F2 )**

파라미터 F 의 값은 각각 변수에 할당되어 사용한다.

<b>IMPORTING PARAMETER</b>	<b>표현식</b>
없음	METH()
한 개	METH( F ) ODER METH ( P = F )
여러 개 (N)	METH( P1 = F1 P2 = F2 )

**SE80 | 0514 | METHOD에 대해 알아보자.**

```
TYPES: BEGIN OF TS_FLIGHT,
        CARRID TYPE S_CARR_ID,
        CARRNAME TYPE S_CARRNAME,
        CONNID TYPE S_CONN_ID,
        FLDATE TYPE S_DATE,
        PERCENTAGE TYPE P LENGTH 3 DECIMALS 2,
END OF TS_FLIGHT.
```

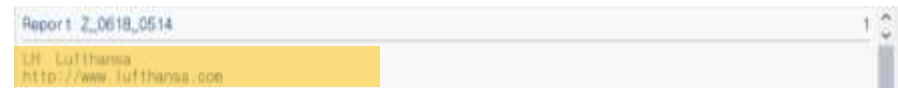
```
DATA GS_FLIGHT TYPE TS_FLIGHT.
```

```
DATA GS_SCARR TYPE ZBC400_S_CARRIER.
```

```
GS_SCARR-CARRID = 'LH'.
```

```
CALL METHOD ZCL_BC400_FLIGHTMODEL=>GET_CARRIER
        EXPORTING IV_CARRID = GS_SCARR-CARRID
        IMPORTING ES_CARRIER = GS_SCARR.
```

```
WRITE: / GS_SCARR-CARRID,
        GS_SCARR-CARRNAME,
        GS_SCARR-URL.
```



## SELECTION-SCREEN | 조회 선택 화면

프로그램의 조회 조건을 입력할 수 있는 SELECTION SCREEN은 REPORT PROGRAM이 실행되면 자동으로 생성된다. SELECTION SCREEN은 상호자와 상호 작용을 하기 위한 INPUT FIELD와 같이 선택 조건을 입력할 수 있는 화면을 제공한다. REPORT PROGRAM에서 SELECTION SCREEN은 'INCLUDE 프로그램명SEL'에 포함하는 것이 좋으며, 프로그래머가 정의하지 않아도 자동으로 SCREEN을 생성하고 FLOW LOGIC을 구현하도록 도와준다.

## SELECTION-SCREEN | PARAMETERS |

사용자가 값을 입력하도록 INPUT FIELD를 정의한다. PARAMETERS는 1개의 값만 입력 받을 수 있다. TYPE을 정의하지 않을 경우 기본 CHAR 1자리로 정의되며 입력된 값은 DATA를 조회하는 SELECT 구문의 조건 등에 사용된다.

DELFAULT 'A',	기본 값을 지정
TYPE CHAR10.	DATA TYPE 정의
LENGTH N,	TYPE C, N, X, P에만 적용, 길이 정의
DECIMALS DEC	소수점 자리 지정
LIKE G	오브젝트와 같은 DATA TYPE 선언
MEMORY ID PID	메모리 파라미터를 할당
MATCHCODE OBJECT MOBJ	4.0 DLGN SEARCH HELP 사용*
MODIF ID MODID	SCREEN-GROUP 지정**
NO-DISPLAY	화면에 보이지 않음
LOWER CASE	대소문자를 구별함   CASE-SENSITIVE
OBLIGATORY	필수 필드로 지정. 화면 필드에는 ? 표시
AS CHECKBOX	CHECK BOX로 표현
RADIOBUTTON GROUP RADI	라디오 버튼으로 표현***
VISIBLE LENGTH VLEN	필드의 일부 길이까지 화면에 보이게 설정
VALUE CHECK	테이블의 필드 속성을 상속받음****

LIKE ( G )	파라미터를 동적으로 선언*****
AS LISTBOX	ABAP DICTIONARY 필드와 연결*****
USER-COMMAND UCOM	체크박스과 라디오 버튼에만 작용*****
AS SEARCH PATTERN	LDB에서 사용*****
VALUE-REQUEST	LDB에서 F4 VALUE HELP를 추가 가능
HELP-REQUEST	VALUR-REQUEST 유사, 필드HELP 생성

\*MOBJ에 SEARCH HELP를 입력하면 POSSIBLE ENTRY가 생성된다.

\*\*그룹별로 화면 속성을 제어한다.

\*\*\*두 개 이상의 필드를 RADIO GROUP으로 선언해야 한다.

\*\*\*\*CHECK TABLE 값을 체크할 수 있다. | 외부 키

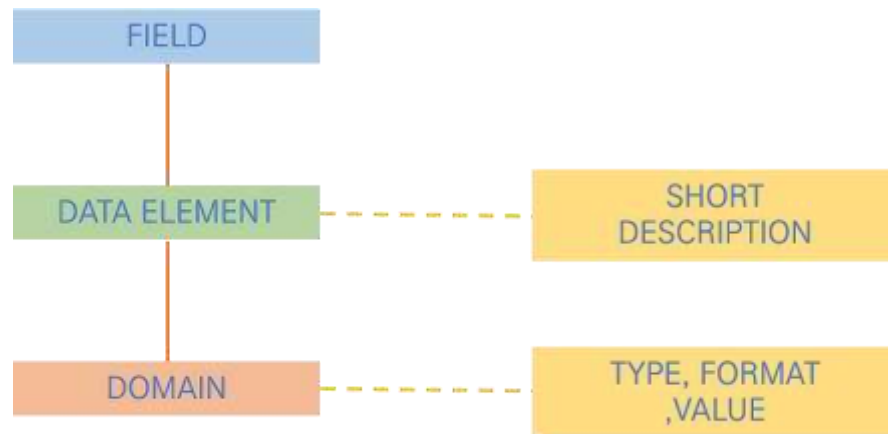
\*\*\*\*\*실행 시 G는 ABAP DICTIONARY에 존재하는 오브젝트 할당 필요

\*\*\*\*\* ABAP DICTIONARY 필드 INPUT HELP와 연결하면 LIST BOX로 보임.

\*\*\*\*\*라디오 버튼 클릭 시 USER COMMAND 수행

\*\*\*\*\* LDB| DBIDBSEL INCLUDE에서 사용하며 SEARCH HELP의 KEY 값으로 INTERNAL TABLE을 구성

DATA TABLE | 데이터를 저장하고 실질적으로 업무처리를 위해 생성한다.



각 FIELD는 DATA ELEMENT를 가지며, DATA ELEMENT는 DOMAIN에 종속되어 있다.

## TABLE TYPE |

ABAP DICTIONARY에 존재하며, INTERNAL TABLE 속성을 정의하는 목적으로 사용된다. ABAP DICTIONARY에서 TABLE TYPE을 생성하게 되면, LINE TYPE, ACCESS TYPE, KEY를 정의해야 한다. LINE TYPE은 ABAP DICTIONARY의 모든 데이터 타입을 사용할 수 있으며 TABLE TYPE 'T\_TYPE'을 참조하려면 DATA GT\_ITAB TYPE T\_TYPE 과 같이 기술한다.

LINE TYPE	인터널 테이블 라인의 데이터 타입 속성과 구조체 정의
ACCESS MODE	인터널 테이블 데이터에 접근하고 관리하기 위한 옵션
KEY	인터널 테이블 KEY   KEY DEFINITION 과 CATEGORY

## TABLE TYPE | SE11 | 생성

## TABLE TYPE | SE11 | LINE TYPE

LINE TYPE	이미 존재하는 타입을 참조*
PREDIFINED TYPE	필드 길이와 타입 직접 지정
REFERENCE TYPE	클래스와 인터페이스에서만 사용가능

\*TABLE TYPE으로 ABAP DICTIONARY에 생성된 DATA ELEMENT, STRUCTURERS, TABLE, TABLE TYPES, VIEW를 선택할 수 있다.

## TABLE TYPE | SE11 | INITIALIZATION AND ACCESS



INITIAL NUMBER OF ROWS | 인터널 테이블의 엔트리 수를 정의

ACCESS MODE |

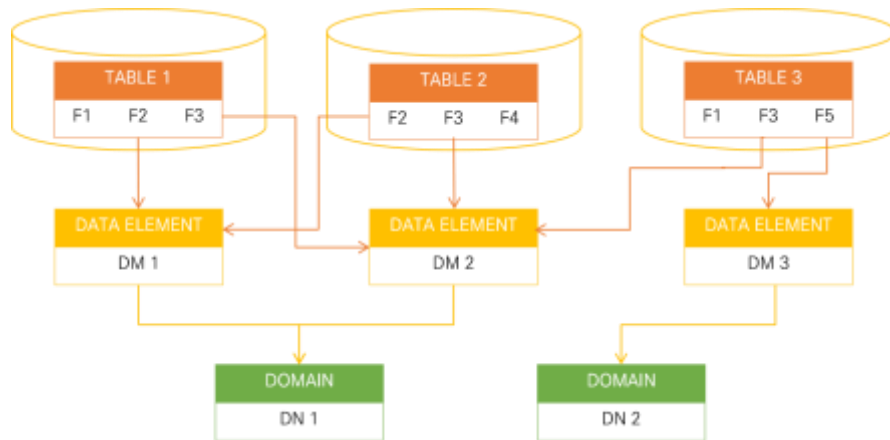
SPECIFIED TABLE TYPE	
STANDARD TABLE	일반적 INDEX 사용, KEY 사용시 순차적 접근 통해 인터널 테이블에 접근 속도   인터널 테이블 라인 수에 따라 비례적 증가
SORTED TABLE	KEY 값으로 내부적 정렬된 TABLE, KEY ACCESS 시 BINARY SEARCH 사용 속도   테이블 수에 지수 함수 비율로 접근 속도 증가
HASH TABLE	HASH 알고리즘에 의해 관리. 모든 인터널 테이블 엔트리는 UNIQUE KEY를 가지며 INDEX를 통해 접근할 수 없다. 속도   모두 같다.
GENERIC TABLE TYPE	
INDEX TABLE	STANDARD TABLE 또는 SORTED TABLE로 INDEX ACCESS가 가능한 INTERNAL TABLE TYPE을 의미한다. FIELD SYMBOL이나 SOUBROUTINE의 FORMAL PARAMETER에서만 사용 가능하다.
NOT SPECIFIED	STANDARD TABLE, SORTED TABLE 또는 HASH TABLE이 될 수 있다. FIELD SYMBOL이나 SUBROUTINE의 FORMAL PARAMETER에서만 사용 가능하다. INDEX 접근은 사용할 수 없으며 각 TABLE TYPE에 맞는 OPERATION을 사용한다.

## TABLE TYPE | SE11 | PRIMARY KEY



ACCESS MODE	KEY CATEGORY
NOT SPECIFIED	NOT SPECIFIED
INDEX TABLE	NOT SPECIFIED
STANDARD TABLE	NON-UNIQUE
SORTED TABLE	UNIQUE, NON-UNIQUE OR NOT SPECIFIED
HASH TABLE	UNIQUE

## DOMAIN |

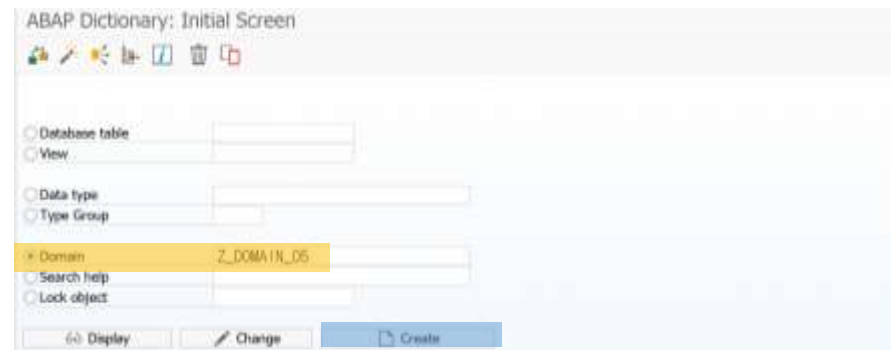


DOMAIN은 필드의 기술적인 속성을 정의하며 DATA ELEMENT에 할당되어 사용한다. TABLE FIELD는 DATA ELEMENT를 지정하여 필드의 속성을 정의한다. 이것을 TWO-LEVEL DOMAIN CONCEPT라 하며 DATA ELEMENT는 필드의 내역과 같은 정보를 기술한다. 모든 테이블과 STRUCTURE의 필드는 DOMAIN이 할당된 DATA ELEMENT를 사용할 수 있다. 필드와 DOMAIN의 관계는 DATA ELEMENT에 의해 정해지며, 같은 DOMAIN이 변경되면 테이블 필드에도 자동으로 반영된다. DOMAIN의 값의 범위는 DATA TYPE과 LENGTH에 의해 정의되거나 고정 값으로 제한할 수 있으며 고정 값은 개별 또는 범위 값으로 직접 입력할 수 있다. DOMAIN은 ABAP DICTIONARY상 참조하는 것이 없는 최소의 단위이다.

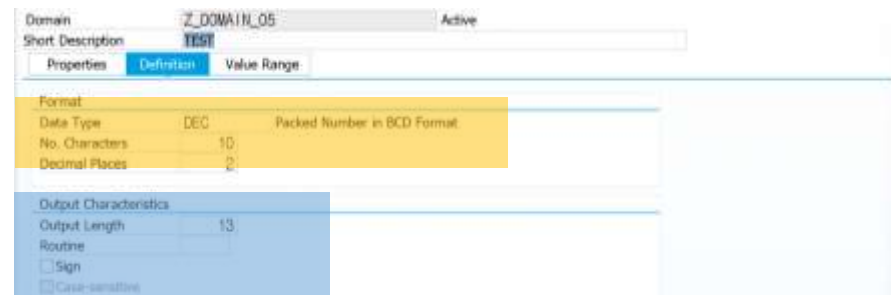
## DOMAIN | 사용하는 이유

DOMAIN은 DATA TYPE과 LENGTH를 결정하며, VALUE를 설정하고 이를 SCREEN에 보여주는 역할을 한다. VALUE RANGE를 통해 직접 또는 VALUE TABLE을 다른 것과 연결하는 등 즉, 재사용성을 위해 DOMAIN을 사용한다.

## DOMAIN | SE11 | 생성 | 주로 DATA ELEMENT에서 생성한다.



## DOMAIN | SE11 | DEFINITION



### FORMAT

\*DATA TYPE과 길이를 지정하며, 숫자타입은 DECIMAL을 지정한다. SIGN OPTION은 +/- 값을 구분하여 저장할 수 있다.

### CHARACTERISTICS

\*문자 타입일 경우 LOWERCASE 체크박스가 활성화되며, 대소문자를 구분한다. CONVERS. ROUTINE를 사용하게 되면 데이터가 테이블에 저장된 값과 조회되는 값을 변경할 수 있다.

## DOMAIN | SE11 | VALUE RANGE

### SINGLE VALUE

\*사용자가 입력할 수 있는 값을 고정하여 해당 값 이외에는 입력하지 못하도록 하여 AUTOMATIC INPUT CHECK가 수행된다. 모든 화면 필드에 자동으로 POSSIBLE ENTRY가 설정되며, 고정 값 이외를 선택하면 에러가 발생한다.

\*\*ABAP 프로그램에서 GET\_DOMAIN\_VALUES 함수를 이용하여 FIXED VALUE에 설정된 데이터를 RANGE 변수에 저장할 수 있다.

### INTERVALS

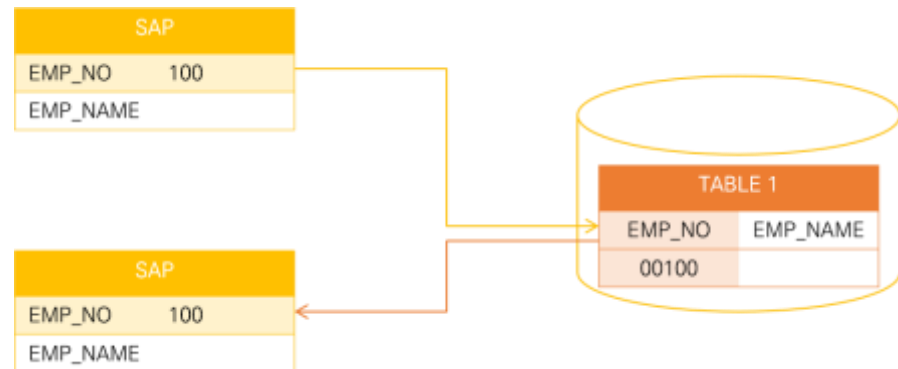
\*정해진 범위 값 내에 존재하는 값만 입력할 수 있다. CHAR, NUMC, DEC, INT 와 같은 유형만 사용할 수 있다.

### VALUE TABLE

\*VALUE TABLE이 존재하는 DOMAIN을 사용하는 필드는 FOREIGN KEY를 정

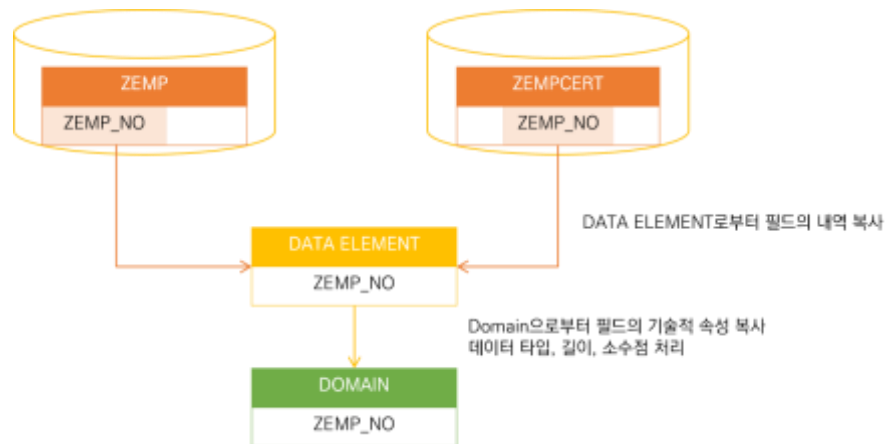
의할 때 시스템이 CHECK TABLE을 자동으로 제안하는 용도로 사용된다. 앞에서 학습한 CHECK TABLE과 VALUE TABLE은 다른 것이므로 구분해서 이해해야 한다. VALUE TABLE은 DOMAIN LEVEL 에서 정의되며 POSSIBLE ENTRY 기능과 INPUT CHECK 의 기능을 없다. VALUE TABLE이 설정된 필드는 FOREIGN KEY가 설정되어야 테이블에서 CHECK TABLE의 역할을 하게 된다. DOMAIN에 VALUE TABLE이 존재하면, 표준 테이블 데이터가 어느 테이블에 저장되는지 쉽게 알 수 있으며, 표준 테이블의 KEY 필드는 대부분 VALUE TABLE이 설정되어 있다.

## DOMAIN | CONVERSION ROUTINE



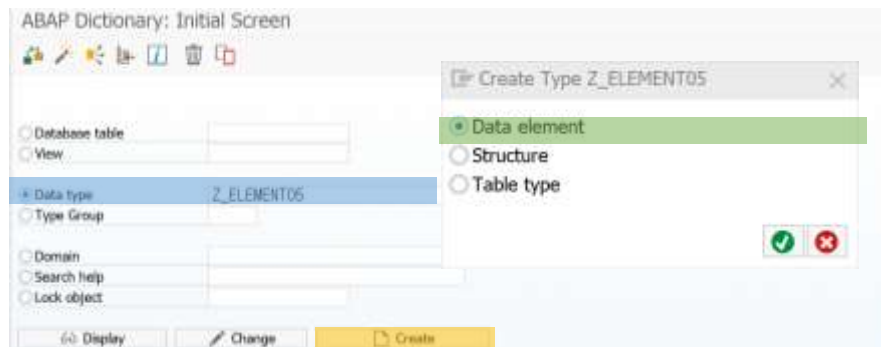
필드의 데이터 타입에 따라 SAP 내부 형태와 화면에 조회되는 값을 변경할 수 있다. 이는 스크린 필드에 조회되는 데이터 포맷과 실제 테이블에 저장되는 포맷 이 다를 수 있음을 의미한다. CONVERSION ROUTINE 은 다음 구문의 2 개의 FUNCTION MODULE과 5자리 이름으로 정의된다. INPUT 함수는 조회용 포맷 을 내부 포맷으로 변경하고, OUTPUT 함수는 내부 포맷을 조회용 포맷으로 변경 한다. CONVERSION ROUTINE 을 포함하는 DOMAIN 을 참조하여 스크린 필 드를 사용하면 자동으로 변환 루틴이 수행된다.

## DATA ELEMENT | 테이블 필드 모든 정보 가진 ABAP DICTIONARY 오브젝트

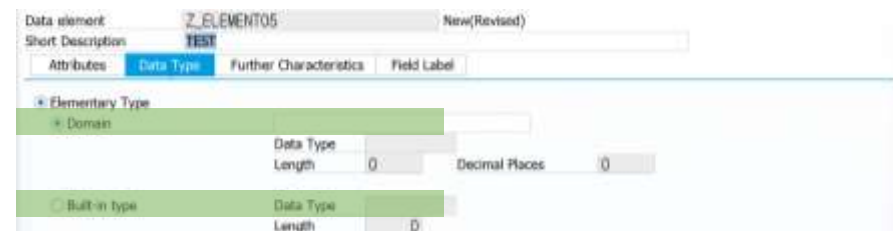


DATA ELEMENT를 생성하면, 모든 테이블의 필드 속성으로 사용할 수 있으며 ABAP 프로그램 내에서 변수를 선언할 때 TYPE 구문의 대상이 될 수 있다.

## DATA ELEMENT | 생성

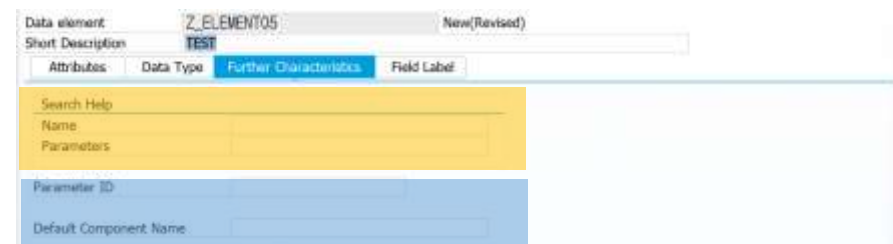


## DATA ELEMENT | DATA TYPE



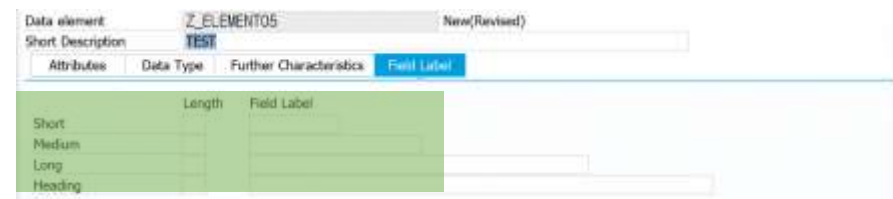
DOMAIN의 기술적 속성을 정의할 수 있으며, PREDIFINED TYPE을 선택하면 DATA TYPE, LENGTH, DECIMAL PLACES를 직접 지정할 수 있다.

## DATA ELEMENT | FURTHER CHARACTERISTICS



SEARCH HELP	PARAMETER는 일반적으로 SEARCH HELP 키 값
PARAMETER IID	SAP 메모리 사용하는 ID로 직접 셋팅
COMPONENT NAME	속성 설명

## DATA ELEMENT | FIELD LABEL





## DATA ELEMENT | ELEMENTARY TYPE

### DOMAIN 사용

\*DOMAIN은 ABAP DICTIONARY에 독립적으로 존재하는 REPOSITORY 오브젝트이며, DATA ELEMENT의 기술적 속성을 정의한다. 하나의 DOMAIN은 여러 개의 DATA ELEMENT에 사용될 수 있다.

### DATA TYPE 직접 사용

\*DATA TYPE은 자주 사용되는 데이터 타입을 ABAP DICTIONARY에서 미리 선언한 TYPE이다. 기본 데이터 타입과 똑같이 프로그램 내에서 선언하여 사용할 수 있다. DATA TYPE을 ELEMENT TYPE이라 부르는 것은 DATA ELEMENT의 사용에 주목적이 있기 때문이다.

ABAP DICTIONARY TYPE	ABAP TYPE
NUMC N	N(N)
DATS	D(8)
CHAR N	C(N)

## DATA ELEMENT | 사용하는 이유

DATA ELEMENT는 재활용의 목적으로 사용한다. DOMAIN과 연결 지어서 DATA TYPE과 LENGTH를 받아오는 역할을 하고, DATA ELEMENT의 SHORT DESCRIPTION을 만든 DATA FIELD에 보여주는 역할을 한다.

## SE80 | 0514 | DOMAIN에 대해 알아보기

DATA: RESULT TYPE Z\_SETS\_05.

PARAMETERS: PA\_FNAME TYPE Z\_FIRSTNAME\_05,  
PA\_LNAME TYPE Z\_LASTNAME\_05,  
PA\_ACTIV TYPE Z\_SETS\_05,  
PA\_LIABS TYPE Z\_ABBILITIES\_05.

START-OF-SELECTION.

NEW-LINE.

WRITE: 'CLIENT:', PA\_FNAME, PA\_LNAME.

RESULT = PA\_ACTIV - PA\_LIABS.

NEW-LINE.

WRITE: 'FIANCE:', PA\_ACTIV, PA\_LIABS, RESULT.

Report Z_0618_0515			
PA_FNAME	YEASIN		
PA_LNAME	KIN		
PA_ACTIV		10	
PA_LIABS		7	

Report Z_0618_0515			
CLIENT: YEASIN		KIN	
FIANCE:	10.00	7.00	3.00