

## 0619 | ABAP

### 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.

### 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.

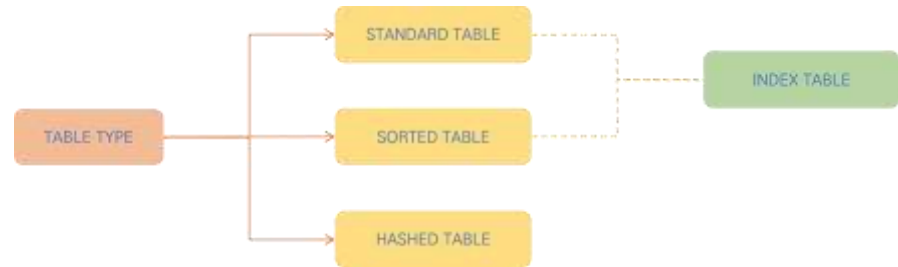
HEADER LINE이 없는 경우	HEADER LINE이 있는 경우
모든 INTERNAL TABLE   STANDARD, SORTED, HASHED TYPE	
INSERT WA INTO TABLE ITAB.	INSERT TABLE ITAB.
COLLECT WA INTO ITAB.	COLLECT ITAB.
READ TABLE ITAB INTO WA.	READ TABLE ITAB...
MODIFY TABLE ITAB FROM WA...	MODIFY TABLE ITAB...
MODIFY ITAB FROM WA... WHERE...	MODIFY 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 편리한 기능으로 인해 실무에서 여전히 활용하고 있다.

## INTERNAL TABLE | 종류 | 개별 ENTRY에 접근하는 방법을 결정한다.



STANDARD TABLE	INDEX 접근
SORTED TABLE	KEY로 정렬된 TABLE, INDEX, KEY 접근
HASHED TABLE	KEY로 접근

## STANDARD TABLE |

순차적인 INDEX를 가지는 TABLE로, TREE 구조를 이루고 있다. INDEX를 이용하여 TABLE ENTRY를 찾을 때 적합하며 READ, MODIFY, DELETE 구문에서도 INDEX를 사용한다. INTERNAL TABLE에서의 INDEX는 데이터가 위치하는 라인의 순번을 의미하며 TABLE의 INDEX와는 개념이 다르다. **STANDARD TABLE KEY는 항상 NON UNIQUE로 선언해야 하며 WITH UNIQUE 구문은 사용할 수 없다.** STANDARD TABLE은 INDEX를 이용하여 검색하기 때문에 TABLE의 라인 수에 비해 탐색 속도가 증가한다. 만약 INDEX가 아닌 칼럼 값을 기준으로 데이터를 읽을 때는 WITH TABLE KEY 나 WITH KEY를 사용한다. WITH TABLE KEY 는 다음과 같이 테이블 내에서 선언 시 정의한 KEY 칼럼을 모두 기술해야 한다. KEY 칼럼 이외의 값을 READ할 경우에는 WITH KEY 구문만 사용하면 된다.

**READ TABLE GT\_ITAB WITH TABLE KEY FIELD1 = 'ENJOY' ~ .**

## SORTED TABLE | \*NON-UNIQUE KEY는 중복을 허용해요 !

SORTED TABLE은 INDEX TABLE로 KEY 값으로 항상 정렬된 INTERNAL TABLE TYPE 이다. 즉 프로그래머가 원하는 KEY 값으로 항상 정렬된 결과로 INTERNAL TABLE에 저장해야 하는 경우 사용한다. INDEX 또는 KEY 값으로 해당 ROW를 찾아가 수 있으며 KEY 값을 사용할 때 WITH UNIQUE를 사용할 수 있다. SORTED TABLE은 내부적으로 BINARY SEARCH 를 이용하기 때문에 TABLE ENTRY의 수와 탐색 속도는 정적 상관관계를 갖는다. SORTED TABLE은 이미 정렬되어 있기 때문에 SORT 명령어를 사용하면 오류가 발생하며, 생성시 UNIQUE/NON-UNIQUE를 반드시 명시하여야 한다.

## HASHED TABLE |

HASHED TABLE은 순차적인 INDEX를 가지고 있지 않으며 HASH 값으로 계산된 KEY 값으로만 탐색할 수 있다. 응답 속도는 INTERNAL TABLE의 ENTRY 수와 상관없이 항상 같으며 HASH 값은 HASH 알고리즘으로 계산된 것으로 메모리에 저장된 주소 값으로 데이터를 바로 읽을 수 있다. HASHED TABLE은 반드시 UNIQUE하게 선언되어야 한다. HASHED TABLE은 INDEX 가 없기 때문에 READ TABLE ~ INDEX 구문을 사용할 수 없으며 READ TABLE ~ WITH TABLE KEY 또는 WITH KEY 구문을 이용해 INTERNAL TABLE DATA에 접근할 수 있다.

## INTERNAL TABLE 명령어 |

### INTERNAL TABLE 값 할당

\*MOVE | MOVE ITAB1 TO ITAB2.

헤더라인이 있는 INTERNAL TABLE은 헤더라인 값만 복사된다.

BODY 까지 복사하려면 다음과 같이 구문을 작성한다.

MOVE ITAB1 [ ] TO ITAB2 [ ].

위 구문은 INTERNAL TABLE TYPE이 동일해야 한다. 만약 TYPE이 다르면 컬럼 순서대로 값을 할당한다. INTERNAL TABLE TYPE이 서로 다른 경우에 프로그램 속성에 UNICODE CHECK ACTIVE 속성이 설정되어 있으면, 문자와 숫자 타입 간에 ALIGNMENT GAP이 생성되어 에러가 발생할 수 있다. LINE TYPE이 다르면 다음 구문을 이용하여 두 오브젝트 간 순서와 관계없이 같은 컬럼 명에만 값을 할당할 수 있다.

MOVE-CORRESPONDING ITAB1 TO ITAB2.

HEADER LINE이 있으면 HEADER LINE과 INTERNAL TABLE 의 이름은 같다. 이를 구분하기 위해 INTERNAL TABLE의 BODY를 [ ] 기호를 이용해 구분한다. 즉, 헤더라인이 있는 INTERNAL TABLE의 이름은 헤더 라인을 의미하고, 헤더 라인이 없는 INTERNAL TABLE은 자기 자신이 된다.

### INTERNAL TABLE 초기화

INTERNAL TABLE을 초기화 하는 구문은 CLEAR, REFRESH, FREE 가 있다.

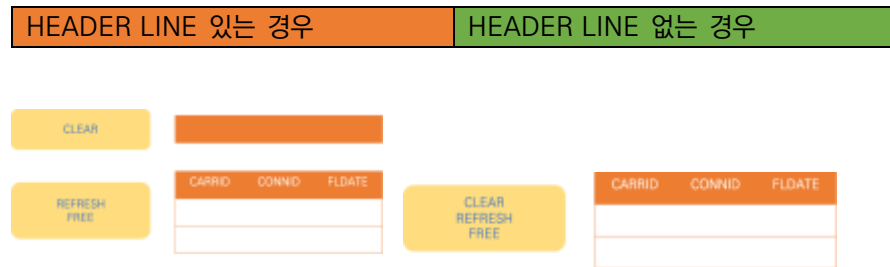
\*CLEAR | CLEAR ITAB. CLEAR ITAB [ ].

메모리 공간을 반환 RELEASE한다. 그러나 처음 메모리의 양을 요구한 정보는 삭제하지 않는다. HEADER LINE을 가지는 INTERNAL TABLE 이라면 헤더 라인만 삭제하며, HEADER LINE이 없는 TABLE은 CLEAR TABLE 만으로도 BODY를 CLEAR 한다. 만약 HEADER LINE이 있는 테이블의 BODY 부분을 삭제하려면 [ ] 를 추가해야 한다.

\*REFRESH | REFRESH ITAB.

INTERNAL TABLE의 DATA 만 지우고, 메모리 공간은 그대로 가지고 있다.

\*FREE | FREE ITAB. 메모리 공간을 반환한다.



즉, CLEAR 구문은 INTERNAL TABLE의 내용을 지우고 할당된 메모리도 반환하나 REFRESH 구문은 테이블의 내용만 삭제한다. REFRESH 구문을 사용하였다면 FREE 구문을 사용해 메모리의 사용을 최소화하는 것이 필요하다.

### INTERNAL TABLE 정렬

\*SORT | SORT ITAB [ ASCENDING | DESCENDING ]

STANDARD 또는 HASHED TYPE의 INTERNAL TABLE을 정렬할 수 있다. INTERNAL TABLE이 가지는 KEY 값으로 SORT 하려면 위 구문을 활용한다. TABLE KEY가 선언되지 않은 경우 문자 타입의 칼럼들을 구성하여 KEY 값으로 만든다. SORT의 기본 정렬은 ASCENDING으로 SORTED TABLE은 테이블 자체에서 정렬된 DATA를 가지고 있기 때문에 SORT 사용시 ERROR가 발생한다.

\*SORT 칼럼 지정 | SORT ITAB [ ASCENDING | DESCENDING ]

BY F1 [ ASCENDING | DESCENDING ]

FN [ ASCENDING | DESCENDING ].

정렬이 필요한 칼럼을 임의로 지정할 때는 위의 구문을 사용한다. 위 구문은 TABLE KEY 이용하지 않고 F1 ~ FN 기준으로 정렬하고 NULL이면 무시한다.

\*STABLE SORT | SORT ITAB ... STABLE.

SORT 명령어를 사용할 때마다 SORT SEQUENCE 가 계속 변한다. STABLE SORT 구문을 활용하면 SORT SEQUENCE 가 보존된다. 그러나 정렬 시간이 더 소요되는 단점이 있다. STABLE 옵션은 같은 데이터라도 처음 위치한 SORT 에 의해 순번이 변경되지 않도록 한다.

### INTERNAL TABLE 속성 알아내기

\*DESCRIBE | DESCRIBE TABLE ITAB [LINES GV\_LINE] [OCCURS GV\_INT] [KIND GV\_KIND].

LINES는 INTERNAL TABLE에 존재하는 현재 라인 수를 반환하며, OCCURS는 INTERNAL TABLE의 초기 라인 수를 반환한다. KIND는 INTERNAL TABLE의 종류를 반환하며, 'T'는 STANDARD TABLE, 'S'는 SORTED TABLE, 그리고 'H'는 HASHED TABLE을 의미한다. INTERNAL TABLE의 속성을 반환하는 옵션 중에서는 LINES가 주로 사용된다.

### INTERNAL TABLE DATA 추가 |

INSERT	APPEND와 동일하나 TABLE TYPE에 따라 수행
APPEND	INTERNAL TABLE 마지막 라인에 DATA 삽입
COLLECT	같은 KEY이면 숫자타입은 SUM, 없으면 DATA 추가

SE80 | 0501 | TABLE에 대해 알아보자.

\*TABLE의 종류는 STANDARD, SORTED, HASED 가 있으며 각 INDEX, INDEX 와 KEY, KEY를 사용하여 구분한다.

\*HEADER가 없는 TABLE은 DATA를 WORKAREA에 올려 확인할 수 있다.

\*TABLE

은 APPEND, INSERT, INSERT, READ TABLE, MODIFY TABLE, LOOP AT ENDLOOP, DELETE, INSERT LINES, APPEND LINES 로 수행할 수 있다.

\*SY-TABIX 를 사용하면 시스템 변수로서 INTERNAL TABLE 의 INDEX 값을 가져온다.

\*READ TABLE은 한 건을 읽어온다.

\*MOVE CORRESPONDING TO 는 같은 필드명을 가지고 있는 곳으로 자료를 이동한다.

\*특정 필드를 조건으로 SORT하고 싶을 때는 SORT BY 와 같은 형태를 사용하며 DESCENDING, ASCENDING를 사용한다.

\*HEADER 가 있는 INTERNAL TABLE의 경우 HEADER 의 이름과 BODY의 이름이 동일하다.

\*CLEAR를 사용하는 경우 HEADER의 DATA를 지우기 때문에 BODY의 내역을 지우기로 싶으면 CLEAR TABLE명 [ ] 와 같은 구문을 사용한다.

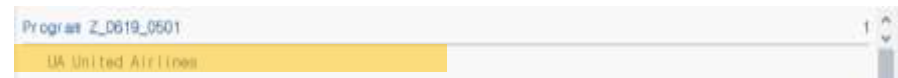
\*DB TABLE은 SCARR을 참조하여 선언하자

DATA GT\_ITAB2 TYPE TABLE OF SCARR.

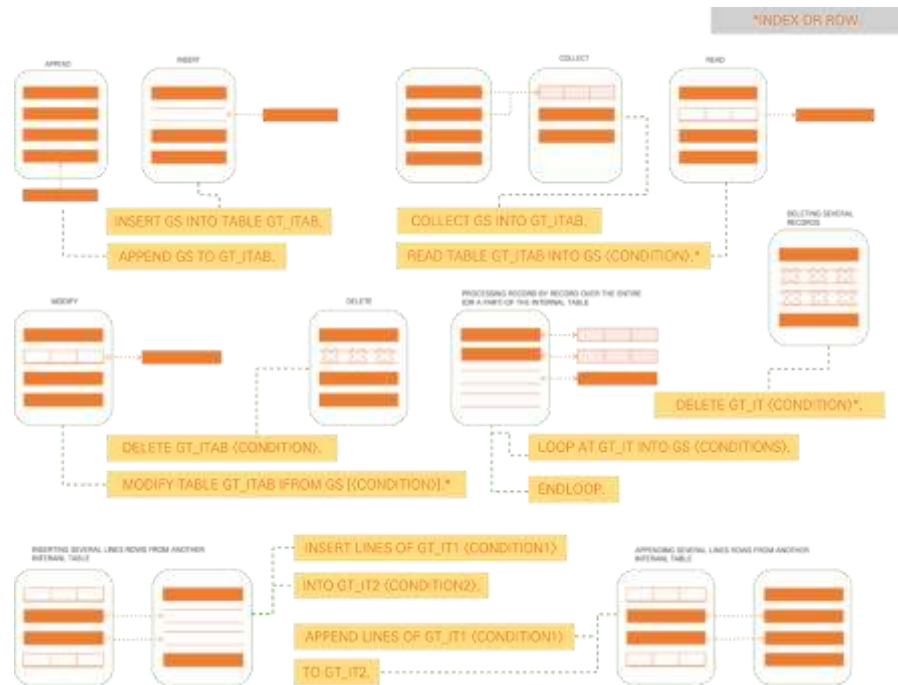
DATA GT-ITAB3 TYPE TABLE OF SCARR WITH HEADER LINE.

```
SELECT CARRID CARRNAME
      FROM SCARR
      INTO CORRESPONDING FIELDS OF GT-ITAB3.
ENDSELECT.
```

WRITE: GT-ITAB3.



INTERNAL TABLE DATA 처리 |



## SUBROUTINE | PERFORM. PERFORM. ~~~ ENDPERFORM.

SUBROUTINE은 FORM으로 시작하여 END FORM으로 종료되는 구문을 의미하며, 스크립트의 모듈화, 재사용, 구조화를 주목적으로 한다. ABAP 프로그램에서는 PERFORM 구문을 이용한 SUBROUTINE으로 유사한 기능을 제공한다. 이 외에 PARAMETER 값을 주고받을 수 있는 FUNCTION MODULE이 존재한다. 모듈화는 의미 있는 기능들을 모아 놓은 프로그램 블록을 의미하며 재사용이 가능한 특성을 가진다. 스크립트가 길면 유지보수 하기 어려우므로, 재사용의 목적이 아니더라도 기능별로 블록화 하여 프로그램을 구조화하는 것이 바람직하다.

SUBROUTINE은 FORM으로 시작하여 END FORM으로 종료되는 구문을 의미한다. FORM은 프로그램의 내부 및 외부에서 호출할 수 있다.

SUBROUTINE	FUNCTION MODULE
LOCAL FUNCTION으로 SUBROUTINE이 포함된 프로그램 내부에서만 사용가능.	SE37  SAP FUNCTION LIBRARY에 저장되어 중앙 집중적으로 관리되는 SUBROUTINE.
예외   EXTERNAL SUBROUTINE 와 같은 외부 프로그램의 SUBROUTINE도 호출할 수 있으나 소속된 프로그램 명시 필수	대표적인 목적은 재사용성

## SUBROUTINE 생성 | 직접 구문을 입력하거나 PERFORM을 실행하기

### PERFORM WRTIE DATA.

\*WRITE DATA를 더블 클릭하면 SUBROUTINE 팝업 창 생성 후 MAIN PROGRAM을 선택한다. 실행하면 화면에 FORM ~ ENDFORM이 생성되며 내부에 스크립트를 작성하면 SUBROUTINE 이 완성된다.

## SUBROUTINE PARAMETER |

PARAMETER는 SUBROUTINE을 호출하는 구문과 호출 받는 사이에 주고받는 값을 의미한다. SUBROUTINE 내에서 PARAMETER는 DATA 구문으로 정의하는 일반적인 LOCAL 변수와 같다. SUBROUTINE을 호출할 때 사용되는 PARAMETER를 ACTUAL PARAMETER라 하고 SUBROUTINE에서 사용되는 PARAMETER를 FORMAL PARAMETER라 정의한다. PERFORM의 USING, CHANGING 구문을 사용하여 PARAMETER를 선언하며, SUBROUTINE과 순서를 동일하게 지정해야 한다. PARAMETER는 ABAP의 모든 기본 DATA TYPE, FIELD SYMBOL, INTERNAL TABLE 등을 사용할 수 있다.

## PARAMETER 전달 방법 | USING, CHANGING 으로 PARAMETER 주고받기

CALL BY VALUE	넘겨주는 ACTUAL PARAMETER와 받는 FORMAL PARAMETER가 물리적으로 다른 메모리 영역을 가짐
CALL BY REFERENCE	물리적으로 같은 메모리 영역을 공유하며 넘겨주는 값을 주소로 가진다.
CALL BY VALUE AND RESULT	변수의 값을 넘겨주고 받는 구문에서 작업을 성공적으로 수행했을 시 변경된 값을 되돌려준다. 물리적으로는 다른 영역을 사용한다.

#### CALL BY VALUE |

FROM SUBR USING ... VALUE (PI) [ TYPE <T> | LIKE <F> ]

\*USING 다음에 PARAMETER를 사용하며 VALUE 구문으로 완성한다. VALUE 구문에서 FORMAL PARAMETER는 자신의 메모리를 가진다. SUBROUTINE을 호출할 시 ACTUAL PARAMETER의 값은 FORMAL PARAMETER에 복사된다. 그러나 FORMAL PARAMETER의 값이 변경되더라도 ACTUAL PARAMETER에는 영향을 미치지 않는다. 만약 FORM 구문에서 USING과 VALUE 키워드를 같이 사용하면 새로운 메모리에 값을 복사하여 전달받는다.

#### CALL BY REFERENCE |

FROM SUBR CHANGING ... PI [ TYPE <T> | LIKE <F> ]

\*CHANGING 다음에 PARAMETER를 사용하면, SUBROUTINE에 전달된 PARAMETER 값이 변경된다. SUBROUTINE의 FORMAL PARAMETER는 자신의 메모리를 가지지 않으며, SUBROUTINE이 호출되는 동안 ACTUAL PARAMETER의 주소 값을 가진다. 즉, SUBROUTINE을 호출한 프로그램 메모리인 동일한 변수 이름에서 작업하게 된다. **만약! VALUE 구문을 사용하지 않으면 USING 과 CHANGING 모두 CALL BY REFERENCE를 이용한다.**

PERFORM 구문에는 USING을 사용하고, FORM에서는 CHANGING 을 사용해도 예러가 발생하지 않는다. 즉, FORM 구문 내에서 VALUE 구문을 사용하지 않으면 USING 과 CHANGING 구문의 기능을 같으나 가독성 차원에서 '사용' 과 '변경' 을 명시적으로 표현하기 위한 구분의 역할을 한다. USING 은 DATA를 전달하고, CHANGING 은 DATA를 전달하고 변경한다는 의미를 가진다. ACTUAL PARAMETER의 값이 SUBROUTINE 내에서 자동으로 변경되는 것을 피하려면 USING 과 VALUE 구문을 함께 사용해야 한다.

#### CALL BY VALUE AND RESULT |

FROM SUBR CHANGING ... VALUE (PI) [ TYPE <T> | LIKE <F> ]

\*CHANGING 다음에 PARAMETER를 사용하고, VALUE 구문으로 완성한다. CALL BY VALUE 와 같이 USING 과 VALUE가 함께 사용되면 SUBROUTINE 내에서 ACTUAL PARAMETER 값을 변경할 수 없으나 CHANGING 구문과 VALUE 구문이 함께 사용되면 SUBROUTINE 이 정상적으로 종료될 경우 ACTUAL PARAMETER 값이 변경된다. 즉 CHANGING 구문은 항상 VALUE 구문과 함께 사용하는 것이 바람직하다.

#### PARAMETER 타입 정의 |

FORM 구문 내의 FORMAL PARAMETER는 TYPE과 LIKE 구문을 이용해 모든 ABAP DATA TYPE을 사용할 수 있다. 타입을 명시적으로 지정하지 않으면, GENERIC TYPE으로 정의되며, ACTUAL PARAMETER 의 기술적 속성을 상속 받게 된다. 이때 TYPE CONVERSION 이 가능한 타입을 사용해야한다. 기본 타입 D, F, I, T를 PERFORM 구문 내에서 사용했다면 FORM 구문의 PARAMETER도 같은 타입을 사용해야 한다. 만약 TYPE D를 파라미터로 FORM 구문에 전달했다면, FORMAL PARAMETER는 다음의 세 가지 방법을 통해 TYPE을 정의할 수 있다.

#### GENERIC TYPE 사용

FORM SUBR CHANGING P\_VAL.

#### ACTUAL PARAMETER와 같은 타입 사용

FORM SUBR CHANGING P\_VAL TYPE D.

#### ACTUAL PARAMETER와 같은 타입 변수 사용

FORM SUBR CHANGING P\_VAL LIKE GV\_VAL.

SE80 | 0504 | PARAMETER에 대해 공부해보자

*\*SUBROUTINE 의 CALL BY VALUE 와 CALL BY RESULT, CALL BY REFERENCE 구문을 사용할 수 있다.*

*\*CALL BY VALUE와 CALL BY RESULT 는 세션이 정상적으로 작동 되었을 때 만 PARAMETER의 값에 적용된다.*

*\*CALL BY REFERENCE는 VALUE 값 없이 사용하며, 수정이나 입력 즉시 PARAMETER 내의 TABLE에도 적용된다.*

*\*FORM 안의 변수는 LOCAL 변수이다.*

TYPES TV\_PERC TYPE P DECIMALS 2.

DATA: GV\_INT1 TYPE I,  
GV\_INT2 TYPE I,  
GV\_RESULT TYPE TV\_PERC.

PERFORM CALC\_PERC  
USING GV\_INT1  
GV\_INT2  
CHANGING GV\_RESULT.

WRITE: GV\_RESULT.

```
*&-----  
-----*  
  
*& Form CALC_PERC
```

```
*&-----  
-----*  
  
*& text  
*&-----  
-----*  
  
*    -->P_GV_INT1 text  
*    -->P_GV_INT2 text  
*    <--P_GV_RESULT text  
*&-----  
-----*
```

FORM CALC\_PERC

USING

PV\_ACT TYPE I

PV\_MAX TYPE I

CHANGING

CV\_PC TYPE TV\_PERC.

DATA IV\_PC TYPE P LENGTH 16 DECIMALS 1.

ENDFORM.



SE80 | 0505 | PARAMETER에 대해 공부해보자

*\*SUBROUTINE 의 CALL BY VALUE 와 CALL BY RESULT, CALL BY REFERENCE 구문을 사용할 수 있다.*

*\*CALL BY VALUE와 CALL BY RESULT 는 세션이 정상적으로 작동 되었을 때 만 PARAMETER의 값에 적용된다.*

*\*CALL BY REFERENCE는 VALUE 값 없이 사용하며, 수정이나 입력 즉시 PARAMETER 내의 TABLE에도 적용된다.*

*\*FORM 안의 변수는 LOCAL 변수이다.*

TYPES TV\_RESULT TYPE P LENGTH 16 DECIMALS 2.

PARAMETERS: PA\_INT1 TYPE I,  
PA\_OP TYPE C LENGTH 1,  
PA\_INT2 TYPE I.

DATA GV\_RESULT TYPE TV\_RESULT.

IF ( PA\_OP = '+' OR PA\_OP = '-' OR  
PA\_OP = '\*' OR PA\_OP = '/' OR  
PA\_INT2 <> 0 OR PA\_OP = '%' ).

CASE PA\_OP.  
WHEN '+'.  
GV\_RESULT = PA\_INT1 + PA\_INT2.

WHEN '-'.  
GV\_RESULT = PA\_INT1 - PA\_INT2.  
WHEN '\*'.  
GV\_RESULT = PA\_INT1 \* PA\_INT2.  
WHEN '/'.  
GV\_RESULT = PA\_INT1 / PA\_INT2.  
WHEN '%'.  
PERFORM CALC\_PERCENTAGE USING PA\_INT1 PA\_INT2  
CHANGING GV\_RESULT.  
ENDCASE.

WRITE: 'RESULT: '(RES), GV\_RESULT.  
ELSEIF PA\_OP = '/' AND PA\_INT2 = 0.  
WRITE: 'NO DIVISION BY ZERO! '(DBZ).  
ELSE.  
WRITE: 'INVALID OPERATOR! '(IOP).  
ENDIF.

\*&-----  
-----\*  
\*& Form CALC\_PERCENTAGE  
\*&-----  
-----\*  
\*& text  
\*&-----

```

-----*
*   -->P_PA_INT1 text
*   -->P_PA_INT2 text
*   <--P_GV_RESULT text
*&-----*

```

```

FORM CALC_PERCENTAGE USING PV_ACT TYPE I
                        PV_MAX TYPE I
                        CHANGING CV_RESULT TYPE TV_RESULT.

```

```

*   IF PV_MAX = 0.
*       CV_RESULT = 0.
*       WRITE 'ERROR IN PERCENTAGE CALCULATION'.
*   ELSE.
*       CV_RESULT = PV_ACT / PV_MAX * 100.
*   ENDIF.

```

ENDFORM.

Report Z\_0619\_0505

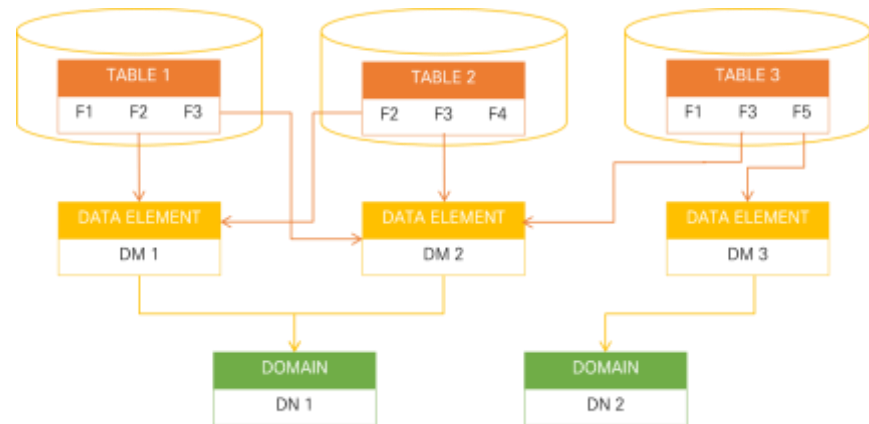
PA_INT1		3.
PA_OP	*	
PA_INT2		4.

Report Z\_0619\_0505

RESULT:		10.00
---------	--	-------

## DOMAIN |

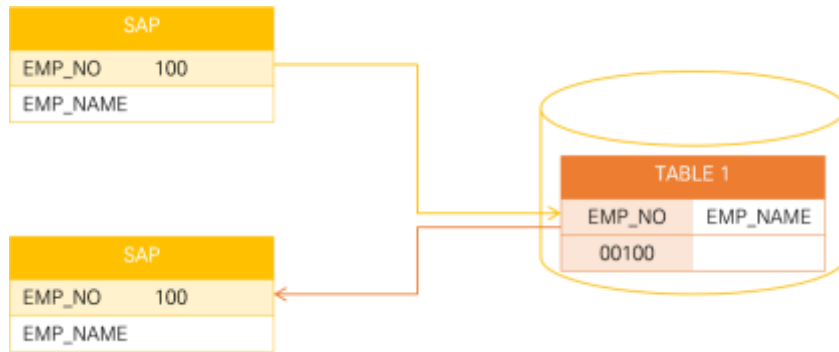


DOMAIN은 필드의 기술적인 속성을 정의하며 DATA ELEMENT에 할당되어 사용한다. TABLE FIELD 는 DATA ELEMENT 를 지정하여 필드의 속성을 정의한다. 이것을 TWO-LEVEL DOMAIN COMCEPT라 하며 DATA ELEMENT는 필드의 내역과 같은 정보를 기술한다. 모든 테이블과 STRUCTURE의 필드는 DOMAIN 이 할당된 DATA ELEMENT를 사용할 수 있다. 필드와 DOMAIN의 관계는 DATA ELEMENT에 의해 정해지며, 같은 DOMAIN이 변경되면 테이블 필드에도 자동으로 반영된다. DOMAIN의 값의 범위는 DATA TYPE과 LENGTH에 의해 정의되나 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 LENGTH 를 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 | CONVERSION ROUTINE



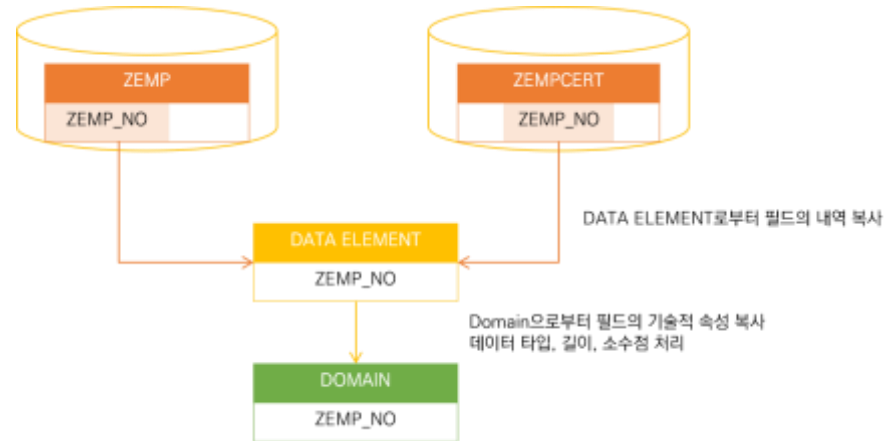
필드의 데이터 타입에 따라 SAP 내부 형태와 화면에 조회되는 값을 변경할 수 있다. 이는 스크린 필드에 조회되는 데이터 포맷과 실제 테이블에 저장되는 포맷이 다를 수 있음을 의미한다. CONVERSION ROUTINE 은 다음 구문의 2 개의 FUNCTION MODULE과 5자리 이름으로 정의된다. INPUT 함수는 조회용 포맷을 내부 포맷으로 변경하고, OUTPUT 함수는 내부 포맷을 조회용 포맷으로 변경한다. CONVERSION ROUTINE 을 포함하는 DOMAIN 을 참조하여 스크린 필드를 사용하면 자동으로 변환 루틴이 수행된다.

## SE11 | ZPERSONO5\_SL 을 생성해보자

Dictionary: Display Structure

Component	Type	Method	Component Type	Data Type	Length	Domain	Short Description
INCLUDE	1	Type	ZPERSONO5_SL		0		ADDRESS
STREET	1	Type	ZPERSONO5_SL	CHAR	30		Street
NO	1	Type	ZPERSONO5_SL	DEC	0		Number
ZIP	1	Type	ZPERSONO5_SL	CHAR	10		Postal Code
CITY	1	Type	ZPERSONO5_SL	CHAR	20		City
NAME	1	Type	ZPERSONO5_SL		0		NAME
PHONE	1	Type	ZPERSONO5_SL		0		PHONE NUMBER

## 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.

## SE80 | 0506 | DOMAIN 생성 후 확인해보자

DATA PERSON TYPE ZPERSONO5\_SL.

DATA WA\_PHONE LIKE LINE OF PERSON-PHONE.

DATA WA\_PHONE2 TYPE ZSTR\_PHONE.

PERSON-NAME-FIRSTNAME = 'YEAJIN'.

PERSON-NAME-LASTNAME = 'KIM'.

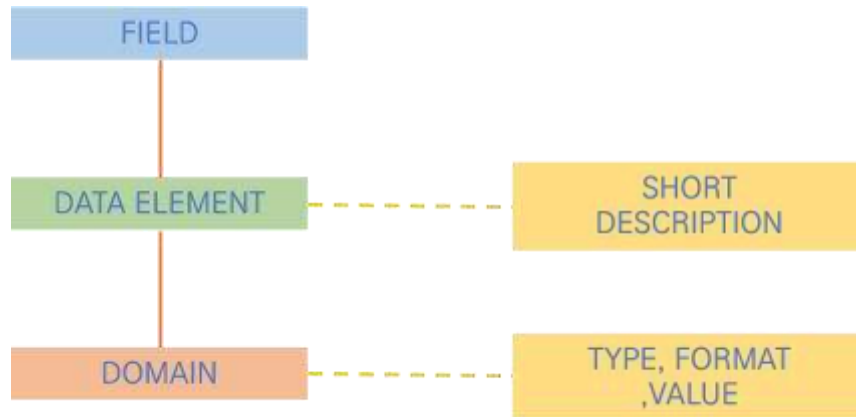
PERSON-STREET = 'CHEONAN'.

BREAK-POINT.

Table: PERSON-NAME-FIRSTNAME, PERSON-NAME-LASTNAME, PERSON-STREET

No.	Field Type	Start	End	Region
1	CHAR	START OF SELECTION	END OF SELECTION	PERSON-NAME
2	CHAR	START OF SELECTION	END OF SELECTION	PERSON-NAME
3	CHAR	START OF SELECTION	END OF SELECTION	PERSON-STREET

DATA TABLE | 데이터를 저장하고 실질적으로 업무처리를 위해 생성한다.



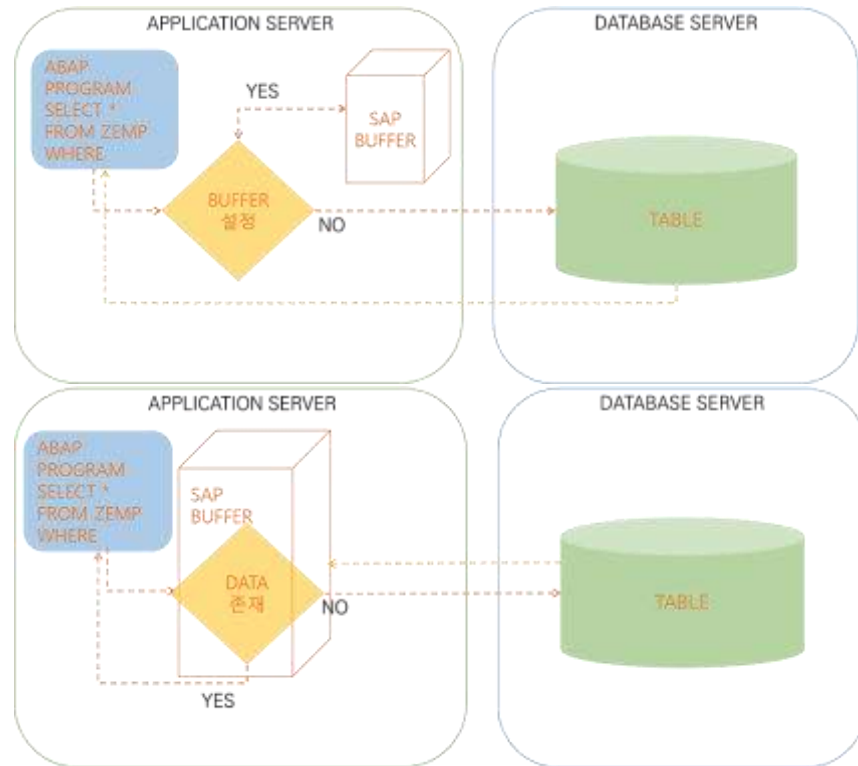
각 FIELD는 DATA ELEMENT를 가지며, DATA ELEMENT는 DOMAIN에 종속.

TABLE TYPE | DATA GT\_ITAB TYPE T\_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 있다.

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

BUFFERING |



BUFFERING 설정 시 DB에서 직접 값을 읽어 오는 것이 아닌 APPLICATION SERVER 영역의 BUFFER에서 DATA를 조회한다. BUFFERING 설정은 특히 CLIENT /SERVER 모델에서 큰 효과를 발휘하며 ACCESS 시간을 효과적으로 절감할 수 있다. 만약 BUFFER에 원하는 값이 존재하지 않아 직접 DB에 ACCESS하여 조회하는 경우 DB에 ACCESS 하고 BUFFER에 DATA를 저장한 후에 반환하기 때문에 DB 조회 시 보다 많은 자원을 소모한다. 따라서 마스터 데이터 성격의 테이블에만 설정하며, 트랜잭션이 자주 발생하는 TABLE은 BUFFERING 설정을 하지 않는 것이 바람직하다.

## BUFFERING |

TRANSPARENT TABLE과 POOLED TABLE만 버퍼 설정이 가능하며, CLUSTER TABLE은 지원되지 않는다. 버퍼링의 모든 키 필드는 CHARACTER DATA TYPE으로 선언되어야 한다. 즉, ABAP TYPES C N D OR T 타입으로 키 필드들이 선언된 테이블만이 버퍼링 설정이 가능하다.

## BUFFERING OPTION |

### BUFFERING NOT PERMITTED

버퍼를 사용하지 않는 경우로, 트랜잭션이 자주 일어나는 테이블은 버퍼를 사용하면 비효율적이다. APPLICATION SERVER 와 DB SERVER 간에 동기화 SYNCHRONIZATION이 진행될 때까지 기다릴 수 없는 최신의 DATA 가 필요한 테이블에 설정한다.

### BUFFERING ALLOWED BUT SWITCHED OFF

고객사에서 버퍼링을 사용할지 결정할 수 있는 BUFFERING 타입으로 BUFFERING TYPE을 설정하지 않아도 됨을 의미한다. 테이블의 크기와 ACCESS PROFILE을 고려하여 BUFFERING을 사용하는 것이 효율적이면 활성화할 수 있다. 이는 SAP에서 개별 고객사의 특성을 다 고려할 수 없기 때문에 상황에 맞게 설정할 수 있도록 해당 타입을 지정한다. 고객사의 판단에 따라 코스트 센터를 자주 변경하지 않는다면 BUFFERING을 활성화할 수 있다.

### BUFFERING ACTIVATED

데이터 변경이 자주 발생하지 않으며 READ 접근이 많은 테이블에 설정한다.

## BUFFERING TYPE |

### SINGLE RECORD

테이블 레코드에 접근한 정보만 버퍼에 저장한다. SINGLE-RECORD 버퍼링은 GENERIC과 FULL 버퍼링보다 스토리지 공간이 적게 필요하나 다른 버퍼링 타입보다 DB에 접근하는 횟수가 증가하게 된다.

### GENERIC AREA BUFFERED

선택된 키 값에 해당하는 테이블의 모든 ENTRY 가 버퍼에 저장된다. GENERIC KEY는 PRIMARY KEY의 일부분이다.

### FULLY BUFFERED

모든 테이블 ROW가 BUFFER에 저장된다. 데이터가 적고 자주 읽히며 데이터가 추가되는 횟수가 적은 테이블에 설정한다.

## BUFFERING SYNCHRONIZATION |

대부분의 SAP는 여러 대의 APPLICATION SERVER가 동작하며 메시지 서버를 통해 각 서버에 접속하고 개별 애플리케이션 서버는 자신의 TABLE BUFFER를 가진다. 즉 우리가 운영서버에 로그인할 때마다 메시지 서버의 교통정리에 의해 같은 서버가 아닌 여러 개의 서버 중 하나에 접속하게 된다.

SYNCHRONIZATION TABLE은 LOCAL BUFFER에서 UPDATE, INSERT, DELETE 문이 발생시 해당 정보를 저장한다. 매분마다 SERVER의 LOCAL BUFFER는 SYNCHRONIZATION TABLE의 데이터를 참고하여 동기화를 수행하며 만약 동기화내 데이터 변경 시 SERVER간 다른 데이터가 존재해 트랜잭션이 자주 발생하는 테이블은 데이터 왜곡이 발생한다. LOCAL BUFFER을 사용하지 않고 DB TABLE에서 DATA를 읽으려면 BYPASSING BUFFER을 사용한다.

SE80 | 0507 | TABLE에 대해 공부해보자

*\*SE11에서 PRIMARY KEY를 변경해서 결과를 확인해보자.*

*\*CARRID 를 PLANETYPE으로 변경해서 확인해보자.*

```
DATA IT_FLIGHT TYPE Z_SFLIGHT_05.  
DATA WA_SFLIGHT TYPE SFLIGHT.  
DATA WA_STR LIKE LINE OF IT_FLIGHT.  
DATA WA_STR2 TYPE LINE OF Z_SFLIGHT_05.
```

```
SELECT * FROM SFLIGHT  
        INTO WA_SFLIGHT  
        WHERE CARRID = 'JL'.
```

```
WRITE: / WA_SFLIGHT-CARRID,  
        WA_SFLIGHT-CONNID,  
        WA_SFLIGHT-FLDATE,  
        WA_SFLIGHT-PRICE,  
        WA_SFLIGHT-CURRENCY,  
        WA_SFLIGHT-PLANETYPE.  
ENDSELECT.
```

ULINE.

*\*내가 원하는 조건에 맞는 DATA를 한번에 TABLE에 담아서 보여주는 역할을 한다.*

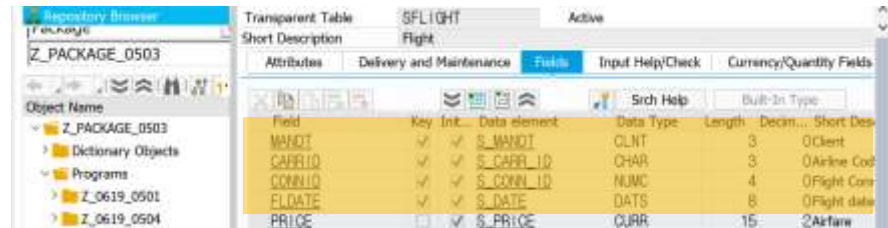
```
SELECT * FROM SFLIGHT  
        INTO WA_SFLIGHT  
        WHERE CARRID = 'JL'.
```

```
LOOP AT IT_FLIGHT INTO WA_SFLIGHT.
```

```
    WRITE: / WA_SFLIGHT-CARRID,  
            WA_SFLIGHT-CONNID,  
            WA_SFLIGHT-FLDATE,  
            WA_SFLIGHT-PRICE,  
            WA_SFLIGHT-CURRENCY,  
            WA_SFLIGHT-PLANETYPE.
```

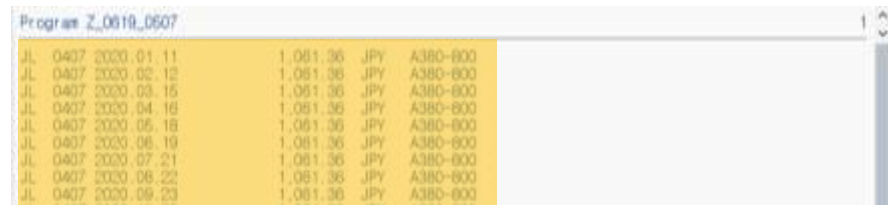
```
ENDLOOP.
```

```
ENDSELECT.
```



The screenshot shows the SAP Repository Browser interface. On the left, the 'Object Name' tree is expanded to 'Z\_PACKAGE\_0503' > 'Programs' > 'Z\_0619\_0501'. The main area displays the 'Transparent Table SFLIGHT' with the 'Fields' tab selected. The table structure is as follows:

Field	Key	Int.	Data element	Data Type	Length	Decim.	Short Des.
MANDT	✓	✓	S_MANDT	CLNT	3		Client
CARRID	✓	✓	S_CARR_ID	CHAR	3		Airline Code
CONNID	✓	✓	S_CONN_ID	NUMC	4		Flight Connection
FLDATE	✓	✓	S_DATE	DATS	8		Flight date
PRICE		✓	S_PRICE	CURR	15		Airfare



The screenshot shows the output of the SAP program Z\_0619\_0507. The output is a table with 8 columns: CARRID, FLDATE, PRICE, CURRENCY, PLANETYPE, and three unlabeled columns. The data is as follows:

CARRID	FLDATE	PRICE	CURRENCY	PLANETYPE			
JL	0407	2020.01.11	1,081.36	JPY	A380-800		
JL	0407	2020.02.12	1,081.36	JPY	A380-800		
JL	0407	2020.03.15	1,081.36	JPY	A380-800		
JL	0407	2020.04.16	1,081.36	JPY	A380-800		
JL	0407	2020.05.18	1,081.36	JPY	A380-800		
JL	0407	2020.06.19	1,081.36	JPY	A380-800		
JL	0407	2020.07.21	1,081.36	JPY	A380-800		
JL	0407	2020.08.22	1,081.36	JPY	A380-800		
JL	0407	2020.09.23	1,081.36	JPY	A380-800		

## SE80 | 0508 | TYPE GROUP에 대해 알아보자

\*TYPE GROUPS는 디렉터리에서 생성할 때 TYPE POOL을 사용하며 사용 시 TYPE POOL 명을 그대로 가져온다.

\*TYPE GROUP은 INCLUDE TECHNIQUE를 기반으로 여러가지 TYPE을 그룹으로 묶어서 선언할 때 사용한다.

\*TYPE GROUP은 GLOBAL TYPE으로 모듈별로 자주 사용하게 되는 타입들을 TYPE GROUP으로 관리하는 것도 좋은 방법이다.

\*GLOBAL TYPE은 모든 프로그램에서 사용 가능하며 DICTIONARY에서 생성한다.

\*LOCAL TYPE은 해당 프로그램에서 사용 가능하며 PROGRAM 내에서 TYPES로 생성한다.

\*NESTED 구조체는 INCLUDE한 구조체가 다른 구조체를 INCLUDE 한다.

TYPE-POOLS : ZMYTP.

DATA 1\_CIRC TYPE F.

DATA 1\_RAD TYPE F VALUE '5.7'.

DATA 1\_PERC TYPE ZMYTP\_PERCENTAGE VALUE '22.5'.

1\_CIRC = 2 \* 1\_RAD \* ZMYTP\_PI.

WRITE: / 1\_CIRC.



## PARAMETER & STRUCTURE |

FORMAL PARAMETER는 모든 ABAP DATA TYPE이 허용되기 때문에 구조체를 사용할 수 있다. 구조체를 PARAMETER로 사용할 때는 TYPE, LIKE 뿐만 아닌 STRUTURE 구문을 이용해 구조체 TYPE을 정의할 수 있다. 만약, 구조체를 전달할 때 타입을 지정하지 않으면 구조체 칼럼을 WRITE 하거나 인식할 때 심볼을 이용한다.

## 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-REQUEST유사, 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

## SELECT-OPTIONS | SELECT-OPTIONS <SELTAB> FOR <F>.

PARAMETERS가 하나의 값만 입력 받을 수 있는 INPUT FIELD인 반면, SELET-OPTIONS는 2개의 INPUT 필드를 통해 다양한 조건 값인 SELECT CRITERIA를 입력 받을 수 있다. RANGE 변수와 같은 구조 (INTERNAL TABLE)를 가지며 항상 FOR 구문과 병행해야 한다. FOR 구문 다음에는 TABLES로 선언된 테이블 필드명이나 DATA로 선언된 변수가 와야 한다.

DELFAULT 'A',	기본 값을 지정
DEFAULT G ~ OPTION OP ~ SIGNS S	OPTION과 SIGN을 지정*
DEFAULT G TO H	구간 값 BETWEEN 지정**
DEFAULT G TO H ~ OPTION OP ~ SIGNS S	앞의 두 구문을 조합한 것으로 OPTION 은 BT와 NB만 가능
MEMORY ID PID	메모리 파라미터를 할당
MATCHCODE OBJECT MOBJ	4.0 DLGN SEARCH HELP 사용***
NO-DISPLAY	화면에 보이지 않음
LOWER CASE	대소문자를 구별함   CASE-SENSITIVE
OBLIGATORY	필수 필드로 지정. 화면 필드에는 ? 표시
NO-EXTENSION	버튼을 제거함
NO-INTERVALS	HIGH 값을 제거함
VISIBLE LENGTH VLEN	필드의 일부 길이까지 화면에 보이게 설정
NO DATABASE SELECTION	LDB 사용옵션   일반 레포트에선 기능 X
VALUE-REQUEST	LDB에서 F4 VALUE HELP를 추가 가능
HELP-REQUEST	VALUR-REQUEST유사, 필드HELP 생성



\*OPTION |

EQ	같다
NE	같지 않다
LT	보다 작다
LE	작거나 같다
GT	보다 크다
GE	크거나 같다
BT	사이 값
NB	사이 값 제외

SIGN |

INCLUSIVE ( I )	EXCLUSIVE ( E )
-----------------	-----------------

\*\*SELECTION-OPTION의 LOW 값에서 HIGH 값을 지정한다.

\*\*\*MOBJ에 SEARCH HELP를 입력하면 POSSIBLE ENTRY가 생성된다.

## SELECTION-SCREEN | 조회 선택 화면

PARAMETER와 SELECTION-OPTION을 사용하면 ABAP 프로그램이 자동으로 필드 내역과 길이를 조절하여 화면인 SELECTION-SCREEN을 생성한다. 만약, 시스템이 생성하는 화면을 커스텀하고 싶으면 SELECTOION-SCREEN 구문을 이용하면 된다.

SELECTION-SCREEN BEGIN OF LINE.
SELECTION-SCREEN END OF LINE.

\*파라미터를 여러 개 묶어 한 라인으로 생성하며 라인에서 SELEC-OPTIONS, SELECT-SCREEN SKIP N 구문을 사용할 수 없다.

## SELECTION-SCREEN SKIP N.

\*빈 라인을 N개 삽입한다.

SELECTION-SCREEN ULINE.	
SELECTION-SCREEN ULINE /1(10).	/ NEW LINE, 위치
SELECTION-SCREEN ULINE POS_LOW(10)	PARAMETER 위치 시작
SELECTION-SCREEN ULINE POS_HIGH(10)	리포트 라인 길이 끝 시작

\*UNDER LINE을 추가한다.

## SELECTION-SCREEN POSITION POS.

\* SELECTION-SCREEN BEGIN OF LINE. 블록 내에서 파라미터의 위치 지정

## SELECTION-SCREEN COMMENT FMT NAME.

\*파라미터에 대한 내역을 지정한다. FMT | /POS(LEN), POS(LEN), (LEN)

## SELECTION-SCREEN PUSHBUTTON FMT NAME USER-COMMAND UCOM.

\*화면에 버튼을 추가하여 클릭하면 AT SELECTION-SCREEN에서 SSCRFIELDS-UCOMM에 저장된다.

SELECTION-SCREEN BEGIN OF BLOCK B1.	
SELECTION-SCREEN END OF BLOCK B1.	
WITH FRAME	프레임을 추가한다.
TITLE	프레임의 TITLE을 추가한다.
NO INTERVALS	블록 내의 SELECT-OPTIONS의 LOW값만 보인다.

\*PARAMETER, SELECT-OPTIONS 등 블록을 형성한다.

## SELECTION-SCREEN FUNCTION KEY N.

\*FUNCTION KEY를 추가한다. TABLES: SSCRFIELDS. 구문이 선언되어야 하면 FUNCTION KEY 내역은 INITIALIZATION에서 MOVE 'FUNCTION KEY1' TO SSCRFIELDS-FUNCTXT\_01. FUNCTION KEY를 클릭하면 AT SELECTION-SCREEN 에서 SSCRFIELDS-UCOMM에 저장된다.

### SE80 | 0509 | TABLE과 SELECT OPTION

\*TABLE을 관리하기 위해서는 KEY FIELD를 반드시 지정해야 한다.

\*KEY FIELD는 중복되지 않은 고유의 값을 가진다.

\*KEY FIELD로 지정되지 않은 FIELD는 FUNCTION FIELD로 정보성 필드이다.

\*DB TABLE은 ELEMENT를 사용하며 이는 DOMAIN에 종속되어 있다.

\*FULL BUFFERING은 DB의 수가 적거나 변경이 없을 때 또는 자주 DATA를 체크할 때 풀 버퍼링 체크를 설정해준다.

\*GENERIC BUFFERING은 키필드의 값이 동일한 것을 버퍼에 올려놓아 데이터가 많을 때 효과적으로 사용할 수 있다.

\*SELECT BUFFERING은 필요한 DATA만 지정하여 버퍼에 올려 원하는 것을 볼 때 사용할 수 있다.

\*삭제 LOG는 SYNCHRONIZATION TABLE에 업로드 한다.

\*FIX VALUE에는 변경을 잘 하지 않거나 엔트리 개수가 적은 것을 지정한다.

\*EVENT 단위로 진행된다.

\*INITIALIZATION 은 초기 셋팅을 하는 부분이며

\*AT SELECTION SCREEN 은 스크린에서 입력이 들어왔을 시 INPUT CHECK

를 하며

\*START OF SELECTION 입력이 들어왔을 때 실행한다.

\*END OF SELECTION ALV GHKAUSDMF 호출하거나 FUNCTION ALV를 통해 INTENAL TABLE에 있는 내용을 확인한다.

\*PBO 화면 보여주기의 이전 수행을 말하며 PAI 는 화면이 나타난 이후에 진행되는 로직을 보여주는 것을 의미한다.

\*SELECT OPTION은 범위의 값 또는 복수의 값을 입력받는다. SCREEN을 지정할 수 있다.

DATA GS\_SFLIGHT TYPE SCARR.

SELECTION-SCREEN BEGIN OF BLOCK CARR WITH FRAME.

SELECT-OPTIONS: SO\_CARR FOR GS\_SFLIGHT-CARRID.

SELECTION-SCREEN END OF BLOCK CARR.

SELECTION-

SCREEN BEGIN OF BLOCK LIMIT WITH FRAME TITLE TEXT-001.

PARAMETERS: PA\_LIM\_1 RADIOBUTTON GROUP LIM DEFAULT

'X',

PA\_LIM\_2 RADIOBUTTON GROUP LIM,

PA\_LIM\_3 RADIOBUTTON GROUP LIM,

PA\_CH AS CHECKBOX DEFAULT 'X'.

SELECTION-SCREEN END OF BLOCK LIMIT.

SELECTION-SCREEN BEGIN OF LINE.

SELECTION-SCREEN COMMENT 1(20) TEXT-S03.

SELECTION-SCREEN COMMENT POS\_LOW(8) TEXT-S04 FOR FIELD PA\_COL.

PARAMETER PA\_COL AS CHECKBOX.

SELECTION-SCREEN COMMENT POS\_HIGH(8) TEXT-S05.

PARAMETERS PA\_ICO AS CHECKBOX.

SELECTION-SCREEN END OF LINE.

The screenshot displays the selection screen for SAP Report Z\_0619\_0509. The interface features a yellow background with the word "BLOCK" centered in red. At the top, there is a header bar with the report name "Report Z\_0619\_0509" and a search icon. Below the header, there are input fields for "SO\_CARR" and "In", along with a magnifying glass icon. On the left side, there is a list of radio buttons for selection: "PA\_LIM\_1" (selected), "PA\_LIM\_2", "PA\_LIM\_3", and "PA\_CH". At the bottom, there is a blue bar with the text "TEST" and two icons for "새 화면" (New Screen) and "화면닫기" (Close Screen).