

## 0623 | 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 OBJECT과 생성

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.

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. 세번째로는 이 타입을 참고하여 DATA: GT\_ITAB TYPE T\_ITAB 과 같이 INTERNAL TABLE을 생성한다. TYPE 대신에 LIKE를 사용하기도 한다.

INTERNAL TABLE | GLOBAL TABLE TYPE

DATA: ITAB TYPE ~ WITH [UKEY|NUKEY] KEY <KEY> WITH HEADERLINE.

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 LOOP를 처리하며 개별 LINE으로 이전한다. DATA ITAB TYPE TABLE OF T\_STR WITH HEADER LINE과 같이 HEADER LINE 이 있는 INTERNAL TABLE을 정의하면 헤더 라인에 담긴 정보를 바로 사용할 수 있다.

## INTERNAL TABLE | HEADER LINE 유무 비교 | 실무에서는 활용 중!

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 LINE을 의미하게 된다. 즉, HEADER LINE의 명령어는 WORK AREA가 생략된 것이다.

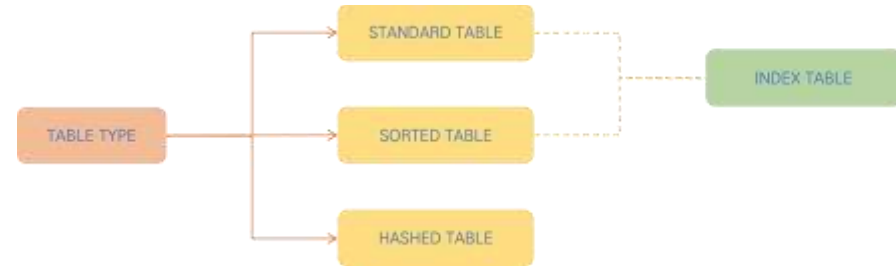
HEADER LINE이 없는 경우	HEADER LINE이 있는 경우
<b>INTERNAL TABLE LOOP 구문</b>	
LOOP AT ITAB INTO WORKAREA, WRITE: WORKAREA-CARRID. ENDLOOP.	LOOP AT ITAB. WRITE: ITAB-CARRID. ENDLOOP.
<b>모든 INTERNAL TABLE   STANDARD, SORTED, HASHED TYPE</b>	
INSERT WA INTO TABLE ITAB.	INSERT TABLE ITAB.
COLLECT WA INTO ITAB.	COLLECT 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...
<b>INDEX TABLE   STANDARD, SORTED, HASHED</b>	
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.

## 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 | READ TABLE ITAB WITH TABLE KEY FIELD1 = 'EN'.

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.** STANDRD 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 KEY는 다음과 같이 테이블 내에서 선언 시 정의한 KEY 칼럼을 모두 기술해야 한다. KEY 칼럼 이외의 값을 READ할 경우에는 WITH KEY 구문만 사용하면 된다.

## 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 찾아갈 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 접근가능 하다.

## 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 칼럼 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 칼럼 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 삭제해HEADER 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.

## HEADER LINE 있는 경우      HEADER LINE 없는 경우

CLEAR

CARRID	CONNO	FLDATE

CLEAR  
REFRESH  
FREE

CARRID	CONNO	FLDATE

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 SORT에 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 | SFLIGHT TABLE에서 DATA를 가져오고 METHOD를 호출하자

*\*2권 399P 실습을 해보자*

*\*SFLIGHT TABLE을 확인하자*

*\*파라미터로 사용자로 부터 값을 받아보자*

```
PARAMETERS: PA_CAR TYPE ZBC400_S_FLIGHT-CARRID,  
            PA_CON TYPE ZBC400_S_FLIGHT-CONNID,  
            PA_DATE TYPE ZBC400_S_FLIGHT-FLDATE.
```

*\*DATA를 선언해 보자*

```
DATA: GS_CARRIER TYPE ZBC400_S_CARRIER,  
      GS_FLIGHT TYPE ZBC400_S_FLIGHT.
```

*\*STRUCTURE TYPE을 선언하자*

```
TYPES: BEGIN OF TS_CARRIERFLIGHT,  
      CARRID TYPE ZBC400_S_FLIGHT-CARRID,  
      CONNID TYPE ZBC400_S_FLIGHT-CONNID,  
      FLDATE TYPE ZBC400_S_FLIGHT-FLDATE,  
      SEATMAX TYPE ZBC400_S_FLIGHT-SEATSMAX,  
      SEATSOCC TYPE ZBC400_S_FLIGHT-SEATSOCC,  
      PERCENTAGE TYPE ZBC400_S_FLIGHT-PERCENTAGE,  
      CARRNAME TYPE ZBC400_S_CARRIER-CARRNAME,  
      CURRCODE TYPE ZBC400_S_CARRIER-CURRCODE,  
      URL TYPE ZBC400_S_CARRIER-URL,  
END OF TS_CARRIERFLIGHT.
```

*\*INTERNAL TABLE을 선언하자*

```
DATA: GS_CARRIERFLIGHT TYPE TS_CARRIERFLIGHT.
```

*\*TRY CATCH문을 통해 EXCEPTION 발생을 잡아보자!*

*TRY. "METHOD를 불러온다"*

```
CALL METHOD ZCL_BC400_FLIGHTMODEL=>GET_FLIGHT  
EXPORTING  
    IV_CARRID = PA_CAR  
    IV_CONNID = PA_CON  
    IV_FLDATE = PA_DATE  
IMPORTING  
    ES_FLIGHT = GS_FLIGHT.
```

\*

```
CALL METHOD ZCL_BC400_FLIGHTMODEL=>GET_CARRIER  
EXPORTING  
    IV_CARRID = PA_CAR  
IMPORTING  
    ES_CARRIER = GS_CARRIER.
```

```
CATCH ZCX_BC400_NO_DATA . "만약 NO DATA EXCEPTION 발생 시"  
    WRITE: 'NO DATA FOUND'(NDF).  
CATCH ZCX_BC400_NO_AUTH .  
    WRITE: 'NO AUTHORITY FOR THIS CARRIER'(NAU).  
ENDTRY.
```

*\*LINE TYPE이 다른 경우 두 OBJECT 간 순서와 관계 없이 같은 칼럼 명에만 값을 할당한다.*

MOVE-CORRESPONDING GS\_CARRIER TO GS\_CARRIERFLIGHT.  
 MOVE-CORRESPONDING GS\_FLIGHT TO GS\_CARRIERFLIGHT.

\*IS NOT INITIAL 초기값인지 확인한 후 아닐 경우 출력  
 \*할당된 값을 사용하자.

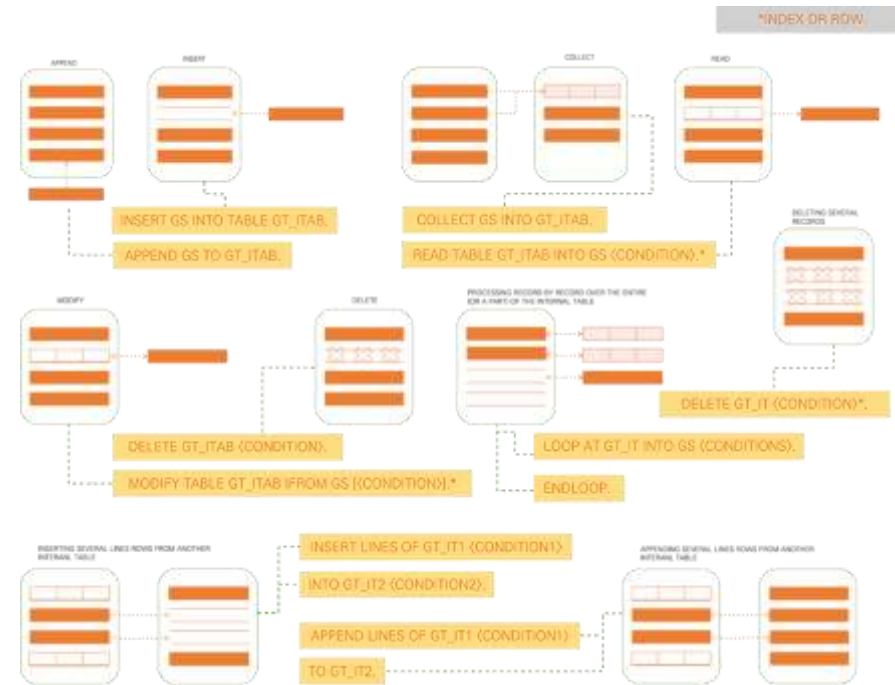
IF GS\_CARRIERFLIGHT IS NOT INITIAL.

WRITE: GS\_CARRIERFLIGHT-CARRID,  
 GS\_CARRIERFLIGHT-CONNID,  
 GS\_CARRIERFLIGHT-FLDATE,  
 GS\_CARRIERFLIGHT-CARRNAME,  
 GS\_CARRIERFLIGHT-CURRCODE,  
 GS\_CARRIERFLIGHT-SEATMAX,  
 GS\_CARRIERFLIGHT-SEATSOCC,  
 GS\_CARRIERFLIGHT-PERCENTAGE, '%',  
 GS\_CARRIERFLIGHT-URL.

ENDIF.



## INTERNAL TABLE DATA 처리 |



## INTERNAL TABLE DATA 추가 |

### INSERT

\*TABLE KEY를 이용해 한 라인 추가 | INSERT LINE INTO TABLE ITAB

한 라인을 삽입하려면 위 구문을 사용하며, KEY 값을 이용하여 INTERNAL TABLE에 라인을 추가한다. INSERT 성공 시 로직이 정상적으로 수행되었는지에 대한 여부를 SY-SUBRC 라는 변수에 0을 저장한다. INTERNAL TABLE이 UNIQUE KEY 값을 가지는 경우라면 중복된 KEY가 존재할 경우 SY-SUBRC에 4를 반환하고 DUMP ERROR는 발생시키지 않는다.

\*TABLE KEY를 이용해 여러 라인 추가 |

INSERT LINES OF ITAB1 [FROM N1] [TO N2] INTO TABLE ITAB2.

INSERT 구문을 이용하여 여러 라인을 삽입할 수 있다. 위의 구문은 ITAB의 인덱스 N1, N2를 ITAB2로 삽입한다 라는 의미를 가지고 있다.

\*INDEX를 이용해 여러 라인 추가 | INSERT LINE INTO ITAB [INDEX IDX].

INDEX 구문을 이용하면 INDEX 값 위치에 라인을 삽입할 수 있다. 이때 HASHED TYPE의 INTERNAL TABLE에는 사용할 수 없다. 성공하면 SY-SUBRC 변수는 0을 반환하며 INTERNAL TABLE의 INDEX를 저장하는 변수인 SY-TABIX는 INDEX 값을 반환한다. INDEX의 위치를 이용하여 INSERT LINES OF ITAB1 INTO ITAB2 [INDEX IDX]. 와 같이 여러 라인을 추가할 수 있다.

INTERNAL TABLE TYPE에 따른 INSERT 효과	
STANDARD	INTERNAL TABLE 마지막에 DATA 추가 APPEND 구문과 동일한 효과
SORTED	INTERNAL TABLE 순서에 따라 DATA 추가 NON-UNIQUE KEY이면 중복된 라인은 동일 KEY 위에 추가
HASHED	TABLE KEY의 HASH INDEX 순서에 따라 DATA 추가

## APPEND

APPEND 구문은 INDEX만 이용하여 INTERNAL TABLE에 DATA를 추가할 수 있다. 즉, APPEND는 HASHED TYPE TABLE에서는 사용할 수 없다.

\*한 라인 추가 | INSERT LINE INTO TABLE ITAB

한 라인을 삽입하려면 위의 구문을 사용하며 APPEND를 한 후에는 SY-TABIX에 INTERNAL TABLE에 추가된 라인의 INDEX 번호를 저장한다.

\*여러 라인 추가 | APPEND LINES OF ITAB TO ITAB2.

INSERT 구문과 동일하게 INTERNAL TABLE을 한 번에 다른 INTERNAL TABLE로 추가할 수 있다. 만약 INTERNAL TABLE ITAB1의 인덱스 N1 N2의 사이의 값을 ITAB2에 추가하고 싶다면 다음의 구문을 이용하면 된다. APPEND LINES OF ITAB [FROM N1] [TO N2] TO ITAB2.

\*APPEND INITIAL LINE |

APPEND INITIAL LINE TO ITAB. ~ APPEND WA TO ITAB.

INTERNAL TABLE을 빈 공간에서 미리 생성한 후, 라인을 추가할 수 있다. SORTED BY 구문을 사용하면, 칼럼 F를 기준으로 DESCENDING 정렬을 수행하여 추가한다. APPEND WA TO ITAB SORTED BY F. 이때 STANDARD TYPE의 INTERNAL TABLE만 효력이 있으며 INITIAL SIZE로 크기를 지정해야 한다. INTERNAL TABLE 선언 시 INITIAL SIZE로 크기를 지정하고 SORTED BY 구문과 함께 사용하면 필요한 라인 수만큼 사용할 수 있다.

INTERNAL TABLE TYPE에 따른 APPEND 효과	
STANDARD	INTERNAL TABLE 마지막에 DATA 추가 SORTED BY 옵션을 이용해 KEY 값을 기준으로 DESCENDING 정렬을 하며 추가할 수 있음
SORTED	DATA가 정렬된 상태로 INTERNAL TABLE에 추가되도록 로직을 구성하지 않으면 DUMP ERROR 발생
HASHED	APPEND 구문을 사용할 수 없음

## COLLECT

\*COLLECT | COLLECT WA INTO ITAB.

COLLECT 는 INTERNAL TABLE의 숫자 타입 칼럼을 합산하는 기능을 수행한다. 따라서 KEY 값을 제외한 칼럼들은 NUMERIC TYPE인 F I P로 선언되어야 한다.

COLLECT 구문을 수행하면, 같은 KEY 값이 있을 때는 숫자 타입의 칼럼을 합산하고 없을 시에는 APPEND 기능을 수행한다. KEY 값이 없는 TABLE은 CHAR 타입 칼럼들을 기준으로 같은 작업을 수행한다.

SE80 | 0502 | STANDARD INTERNAL TABLE에 따른 APPEND 효과

*\*STANDARD는 INDEX와 KEY로 접근할 수 있다.*

*\*UNIQUE는 중복 값을 허용하지 않는다.*

*\*KEY로만 ACCESS가 가능하며 UNIQUE한 값을 가질 수 있는 HASHED TABLE*

*\*LOCAL STRUCTURE TYPE 선언*

```
TYPES: BEGIN OF TS_TYPE,  
        CARRID TYPE S_CARR_ID,  
        CONNID TYPE S_CONN_ID,  
        SEATMAX TYPE S_SEATSMAX,  
END OF TS_TYPE.
```

*\*INTERNAL TABLE-STANDARD 선언*

```
DATA GT_ITAB TYPE STANDARD TABLE OF TS_TYPE.
```

```
DATA: GT_STR LIKE LINE OF GT_ITAB.
```

*\*값을 할당하자*

```
GT_STR-CARRID = 'AA'.
```

```
GT_STR-CONNID = 17.
```

```
GT_STR-SEATMAX = 100.
```

```
APPEND GT_STR TO GT_ITAB. "APPEND를 통해 STR를 TABLE에 추가하자"
```

```
CLEAR GT_STR. "WORK AREA로 사용하는 STR을 비워주자"
```

```
GT_STR-CARRID = 'LH'.
```

```
GT_STR-CONNID = 20.
```

```
GT_STR-SEATMAX = 150.
```

```
APPEND GT_STR TO GT_ITAB.
```

```
GT_STR-CARRID = 'BB'.
```

```
GT_STR-CONNID = 15.
```

```
GT_STR-SEATMAX = 90.
```

```
APPEND GT_STR TO GT_ITAB.
```

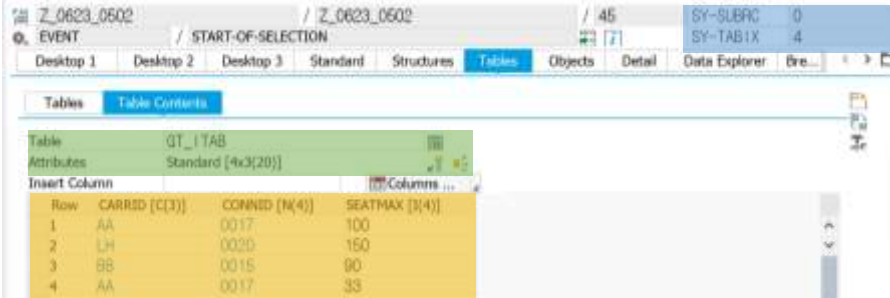
```
GT_STR-CARRID = 'AA'.
```

```
GT_STR-CONNID = 17.
```

```
GT_STR-SEATMAX = 33.
```

```
APPEND GT_STR TO GT_ITAB.
```

BREAK-POINT.



Row	CARRID (C(3))	CONNID (N(4))	SEATMAX (I(4))
1	AA	0017	100
2	LH	0020	150
3	BB	0015	90
4	AA	0017	33



SE80 | 0503 | HEADER가 없는 STANDARD TABLE의 APPEND 효과 출력

*\*STANDARD는 INDEX와 KEY로 접근할 수 있다.*

*\*UNIQUE는 중복 값을 허용하지 않는다.*

*\*KEY로만 ACCESS가 가능하며 UNIQUE한 값을 가질 수 있는 HASHED TABLE*

*\*HEADER LINE이 없는 INTERNAL TABLE*

*\*LOCAL STRUCTURE TYPE*

```
TYPES: BEGIN OF TS_TYPE,  
        CARRID TYPE S_CARR_ID,  
        CONNID TYPE S_CONN_ID,  
        SEATMAX TYPE S_SEATSMAX,  
END OF TS_TYPE.
```

*\*INTERNAL TABLE-STANDARD*

```
DATA GT_ITAB TYPE STANDARD TABLE OF TS_TYPE.
```

```
DATA: GT_STR LIKE LINE OF GT_ITAB.
```

```
GT_STR-CARRID = 'AA'.
```

```
GT_STR-CONNID = 17.
```

```
GT_STR-SEATMAX = 100.
```

```
APPEND GT_STR TO GT_ITAB. "APPEND를 통해 STR를 TABLE에 추가하자"
```

```
CLEAR GT_STR. "WORK AREA로 사용하는 STR을 비워주자"
```

```
GT_STR-CARRID = 'LH'.
```

```
GT_STR-CONNID = 20.
```

```
GT_STR-SEATMAX = 150.
```

```
APPEND GT_STR TO GT_ITAB.
```

```
CLEAR GT_STR.
```

```
GT_STR-CARRID = 'BB'.
```

```
GT_STR-CONNID = 15.
```

```
GT_STR-SEATMAX = 90.
```

```
APPEND GT_STR TO GT_ITAB.
```

```
CLEAR GT_STR.
```

```
GT_STR-CARRID = 'AA'.
```

```
GT_STR-CONNID = 17.
```

```
GT_STR-SEATMAX = 33.
```

```
APPEND GT_STR TO GT_ITAB.
```

```
CLEAR GT_STR.
```

*\*TABLE에 있는 DATA를 읽어보자*

*\*\*SY-INDEX는 LOOP 횟수, TABIX는 INTERNAL TABLE의 INDEX*

```
LOOP AT GT_ITAB INTO GT_STR.
```

```
    WRITE: / 'INDEX', SY-INDEX, 'TABIX:', SY-TABIX.
```

```
    WRITE: GT_STR-CARRID,  
           GT_STR-CONNID,  
           GT_STR-SEATMAX.
```

```
ENDLOOP.
```

```
DO 4 TIMES.
```

```
    WRITE: / 'INDEX', SY-INDEX, 'TABIX:', SY-TABIX.  
ENDDO.
```

Report Z\_0603\_0603

INDEX	0	TABIX:	1	AA	0017	100
INDEX	0	TABIX:	2	LH	0020	150
INDEX	0	TABIX:	3	BB	0015	90
INDEX	0	TABIX:	4	AA	0017	33
INDEX	1	TABIX:	4			
INDEX	2	TABIX:	4			
INDEX	3	TABIX:	4			
INDEX	4	TABIX:	4			

SE80 | 0504 | HEADER가 있는 STANDARD TABLE의 APPEND 효과 출력

*\*HEADER LINE이 있는 INTERNAL TABLE*

```
TYPES: BEGIN OF TS_TYPE,
        CARRID TYPE S_CARR_ID,
        CONNID TYPE S_CONN_ID,
        SEATMAX TYPE S_SEATSMAX,
END OF TS_TYPE.
```

```
DATA: GT_STR TYPE TS_TYPE.
```

```
DATA: GT_ITAB LIKE TABLE OF GT_STR WITH HEADER LINE.
```

*\*CLEAR를 사용하면 HEADERLINE의 DATA가 삭제된다.*

*\*CLEAR GT\_ITAB[ ]. 을 하면 BODY도 삭제 가능하다.*

```
GT_ITAB-CARRID = 'AA'.
GT_ITAB-CONNID = 17.
GT_ITAB-SEATMAX = 100.
APPEND GT_ITAB.
CLEAR GT_ITAB.
```

```
GT_ITAB-CARRID = 'LH'.
GT_ITAB-CONNID = 20.
GT_ITAB-SEATMAX = 150.
```

```
APPEND GT_ITAB.
```

```
CLEAR GT_ITAB.
```

```
GT_ITAB-CARRID = 'BB'.
```

```
GT_ITAB-CONNID = 15.
```

```
GT_ITAB-SEATMAX = 90.
```

```
APPEND GT_ITAB.
```

```
CLEAR GT_ITAB.
```

```
GT_ITAB-CARRID = 'AA'.
```

```
GT_ITAB-CONNID = 17.
```

```
GT_ITAB-SEATMAX = 33.
```

```
APPEND GT_ITAB.
```

```
CLEAR GT_ITAB.
```

*\*TABLE에 있는 DATA를 읽어보자*

*\*\*SY-INDEX 는 LOOP 횟수 , TABIX 는 INTERNAL TABLE의 INDEX*

```
LOOP AT GT_ITAB.
```

```
WRITE: / 'INDEX', SY-INDEX, 'TABIX:', SY-TABIX.
```

```
WRITE: GT_ITAB-CARRID,
        GT_ITAB-CONNID,
        GT_ITAB-SEATMAX.
```

```
ENDLOOP.
```

Report Z\_0603\_0604

INDEX	0	TABIX:	1	AA	0017	100
INDEX	0	TABIX:	2	LH	0020	150
INDEX	0	TABIX:	3	BB	0015	90
INDEX	0	TABIX:	4	AA	0017	33

SE80 | 0505 | STANDARD TABLE의 LOOP 조건 | FROM TO 으로 출력

*\*LOCAL STRUCTURE TYPE*

```
TYPES: BEGIN OF TS_TYPE,  
        CARRID TYPE S_CARR_ID,  
        CONNID TYPE S_CONN_ID,  
        SEATMAX TYPE S_SEATSMAX,  
END OF TS_TYPE.
```

*\*INTERNAL TABLE-STANDARD*

```
DATA GT_ITAB TYPE STANDARD TABLE OF TS_TYPE.
```

```
DATA: GT_STR LIKE LINE OF GT_ITAB.
```

```
GT_STR-CARRID = 'AA'.  
GT_STR-CONNID = 17.  
GT_STR-SEATMAX = 100.  
APPEND GT_STR TO GT_ITAB.  
CLEAR GT_STR.
```

```
GT_STR-CARRID = 'LH'.  
GT_STR-CONNID = 20.  
GT_STR-SEATMAX = 150.  
APPEND GT_STR TO GT_ITAB.  
CLEAR GT_STR.
```

```
GT_STR-CARRID = 'BB'.  
GT_STR-CONNID = 15.
```

```
GT_STR-SEATMAX = 90.
```

```
APPEND GT_STR TO GT_ITAB.  
CLEAR GT_STR.
```

```
GT_STR-CARRID = 'AA'.  
GT_STR-CONNID = 17.  
GT_STR-SEATMAX = 33.  
APPEND GT_STR TO GT_ITAB.  
CLEAR GT_STR.
```

*\*TABLE에 있는 DATA를 읽어보자*

*\*\*SY-INDEX 는 LOOP 횟수 , TABIX 는 INTERNAL TABLE의 INDEX*

```
LOOP AT GT_ITAB INTO GT_STR FROM 1 TO 2.  
    WRITE: / 'INDEX', SY-INDEX, 'TABIX:', SY-TABIX.  
    WRITE:   GT_STR-CARRID,  
            GT_STR-CONNID,  
            GT_STR-SEATMAX.  
ENDLOOP.
```

Report Z\_0603\_0605

INDEX	0	TABIX:	1	AA	0017	100
INDEX	0	TABIX:	2	LH	0020	150

SE80 | 0508 | HEADER가 있는 경우 STANDARD TABLE | FROM TO 출력

*\*HEADER LINE이 있는 INTERNAL TABLE*

```
TYPES: BEGIN OF TS_TYPE,  
        CARRID TYPE S_CARR_ID,  
        CONNID TYPE S_CONN_ID,  
        SEATMAX TYPE S_SEATSMAX,  
END OF TS_TYPE.
```

*\*INTERNAL TABLE-STANDARD*

```
DATA: GT_STR TYPE TS_TYPE.
```

```
DATA: GT_ITAB LIKE TABLE OF GT_STR WITH HEADER LINE.
```

*\*CLEAR를 사용하면 HEADERLINE의 DATA가 삭제된다.*

*\*CLEAR GT\_ITAB[ ]. 을 하면 BODY도 삭제 가능하다.*

```
GT_ITAB-CARRID = 'AA'.  
GT_ITAB-CONNID = 17.  
GT_ITAB-SEATMAX = 100.  
APPEND GT_ITAB.  
CLEAR GT_ITAB.
```

```
GT_ITAB-CARRID = 'LH'.  
GT_ITAB-CONNID = 20.  
GT_ITAB-SEATMAX = 150.  
APPEND GT_ITAB.  
CLEAR GT_ITAB.
```

```
GT_ITAB-CARRID = 'BB'.
```

```
GT_ITAB-CONNID = 15.
```

```
GT_ITAB-SEATMAX = 90.
```

```
APPEND GT_ITAB.
```

```
CLEAR GT_ITAB.
```

```
GT_ITAB-CARRID = 'AA'.
```

```
GT_ITAB-CONNID = 17.
```

```
GT_ITAB-SEATMAX = 33.
```

```
APPEND GT_ITAB.
```

```
CLEAR GT_ITAB.
```

*\*TABLE에 있는 DATA를 읽어보자*

*\*\*SY-INDEX 는 LOOP 횟수 , TABIX 는 INTERNAL TABLE의 INDEX*

```
LOOP AT GT_ITAB FROM 1 TO 2.
```

```
WRITE: / 'INDEX', SY-INDEX, 'TABIX:', SY-TABIX.
```

```
WRITE: GT_ITAB-CARRID,  
        GT_ITAB-CONNID,  
        GT_ITAB-SEATMAX.
```

```
ENDLOOP.
```

Report: Z_0603_0508					
INDEX	0	TABIX:	1	AA	0017
INDEX	0	TABIX:	2	LH	0020

SE80 | 0506 | STANDARD TABLE의 LOOP 조건 | KEY 값으로 출력

*\*LOCAL STRUCTURE TYPE*

```
TYPES: BEGIN OF TS_TYPE,  
        CARRID TYPE S_CARR_ID,  
        CONNID TYPE S_CONN_ID,  
        SEATMAX TYPE S_SEATSMAX,  
END OF TS_TYPE.
```

*\*INTERNAL TABLE-STANDARD*

```
DATA GT_ITAB TYPE STANDARD TABLE OF TS_TYPE.  
DATA: GT_STR LIKE LINE OF GT_ITAB.
```

```
GT_STR-CARRID = 'AA'.  
GT_STR-CONNID = 17.  
GT_STR-SEATMAX = 100.  
APPEND GT_STR TO GT_ITAB.  
CLEAR GT_STR.
```

```
GT_STR-CARRID = 'LH'.  
GT_STR-CONNID = 20.  
GT_STR-SEATMAX = 150.  
APPEND GT_STR TO GT_ITAB.  
CLEAR GT_STR.
```

```
GT_STR-CARRID = 'BB'.  
GT_STR-CONNID = 15.
```

```
GT_STR-SEATMAX = 90.  
APPEND GT_STR TO GT_ITAB.  
CLEAR GT_STR.
```

```
GT_STR-CARRID = 'AA'.  
GT_STR-CONNID = 17.  
GT_STR-SEATMAX = 33.  
  
*APPEND GT_STR TO GT_ITAB.  
COLLECT GT_STR INTO GT_ITAB.  
CLEAR GT_STR.
```

*\*DELETE GT\_ITAB WHERE CARRID = 'AA'.*

*\*TABLE에 있는 DATA를 읽어보자*

*\*\*SY-INDEX 는 LOOP 횟수 , TABIX 는 INTERNAL TABLE의 INDEX*

```
LOOP AT GT_ITAB INTO GT_STR WHERE CARRID = 'AA'.  
    WRITE: / 'INDEX', SY-INDEX, 'TABIX:', SY-TABIX.  
    WRITE:    GT_STR-CARRID,  
              GT_STR-CONNID,  
              GT_STR-SEATMAX.
```

```
ENDLOOP.
```

Program Z_0623_0506			
INDEX	0	TABIX:	1 AA 0017 133

SE80 | 0509 | HEADER가 있는 경우 STANDARD TABLE | 조건 KEY 출력

*\*HEADER LINE이 있는 INTERNAL TABLE*

```
TYPES: BEGIN OF TS_TYPE,  
        CARRID TYPE S_CARR_ID,  
        CONNID TYPE S_CONN_ID,  
        SEATMAX TYPE S_SEATSMAX,  
END OF TS_TYPE.
```

*\*INTERNAL TABLE-STANDARD*

```
DATA: GT_STR TYPE TS_TYPE.
```

```
DATA: GT_ITAB LIKE TABLE OF GT_STR WITH HEADER LINE.
```

*\*CLEAR를 사용하면 HEADERLINE의 DATA가 삭제된다.*

*\*CLEAR GT\_ITAB[ ]. 을 하면 BODY도 삭제 가능하다.*

```
GT_ITAB-CARRID = 'AA'.  
GT_ITAB-CONNID = 17.  
GT_ITAB-SEATMAX = 100.  
APPEND GT_ITAB.  
CLEAR GT_ITAB.
```

```
GT_ITAB-CARRID = 'LH'.  
GT_ITAB-CONNID = 20.  
GT_ITAB-SEATMAX = 150.  
APPEND GT_ITAB.  
CLEAR GT_ITAB.
```

```
GT_ITAB-CARRID = 'BB'.
```

```
GT_ITAB-CONNID = 15.
```

```
GT_ITAB-SEATMAX = 90.
```

```
APPEND GT_ITAB.
```

```
CLEAR GT_ITAB.
```

```
GT_ITAB-CARRID = 'AA'.
```

```
GT_ITAB-CONNID = 17.
```

```
GT_ITAB-SEATMAX = 33.
```

```
APPEND GT_ITAB.
```

```
CLEAR GT_ITAB.
```

*\*TABLE에 있는 DATA를 읽어보자*

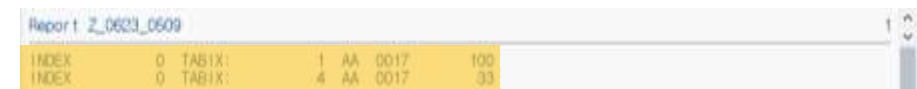
*\*\*SY-INDEX 는 LOOP 횟수 , TABIX 는 INTERNAL TABLE의 INDEX*

```
LOOP AT GT_ITAB WHERE CARRID = 'AA'.
```

```
    WRITE: / 'INDEX', SY-INDEX, 'TABIX:', SY-TABIX.
```

```
    WRITE: GT_ITAB-CARRID,  
            GT_ITAB-CONNID,  
            GT_ITAB-SEATMAX.
```

```
ENDLOOP.
```



INDEX	0	TABIX:	1	AA	0017	100
INDEX	0	TABIX:	4	AA	0017	33

SE80 | 0507 | STANDARD TABLE의 READ 조건 | INDEX 값

*\*LOCAL STRUCTURE TYPE*

```
TYPES: BEGIN OF TS_TYPE,  
        CARRID TYPE S_CARR_ID,  
        CONNID TYPE S_CONN_ID,  
        SEATMAX TYPE S_SEATSMAX,  
END OF TS_TYPE.
```

*\*INTERNAL TABLE-STANDARD*

```
DATA GT_ITAB TYPE STANDARD TABLE OF TS_TYPE.  
DATA: GT_STR LIKE LINE OF GT_ITAB.
```

```
GT_STR-CARRID = 'AA'.  
GT_STR-CONNID = 17.  
GT_STR-SEATMAX = 100.  
APPEND GT_STR TO GT_ITAB.  
CLEAR GT_STR.
```

```
GT_STR-CARRID = 'LH'.  
GT_STR-CONNID = 20.  
GT_STR-SEATMAX = 150.  
APPEND GT_STR TO GT_ITAB.  
CLEAR GT_STR.
```

```
GT_STR-CARRID = 'BB'.  
GT_STR-CONNID = 15.
```

```
GT_STR-SEATMAX = 90.  
APPEND GT_STR TO GT_ITAB.  
CLEAR GT_STR.
```

```
GT_STR-CARRID = 'AA'.  
GT_STR-CONNID = 17.  
GT_STR-SEATMAX = 33.  
APPEND GT_STR TO GT_ITAB.  
CLEAR GT_STR.
```

*\*TABLE에 있는 DATA를 읽어보자*

*\*\*SY-INDEX 는 LOOP 횟수 , TABIX 는 INTERNAL TABLE의 INDEX*

```
READ TABLE GT_ITAB INTO GT_STR INDEX 3.  
WRITE: / 'INDEX', SY-INDEX, 'TABIX:', SY-TABIX.  
WRITE: GT_STR-CARRID,  
        GT_STR-CONNID,  
        GT_STR-SEATMAX.
```



Report Z\_0603\_0507

INDEX	0	TABIX:	3	BB	0015	90
-------	---	--------	---	----	------	----

SE80 | 0510 | HEADER가 있는 경우 STANDARD TABLE | 조건 INDEX READ

*\*HEADER LINE이 있는 INTERNAL TABLE*

```
TYPES: BEGIN OF TS_TYPE,  
        CARRID TYPE S_CARR_ID,  
        CONNID TYPE S_CONN_ID,  
        SEATMAX TYPE S_SEATSMAX,  
END OF TS_TYPE.
```

*\*INTERNAL TABLE-STANDARD*

```
DATA: GT_STR TYPE TS_TYPE.
```

```
DATA: GT_ITAB LIKE TABLE OF GT_STR WITH HEADER LINE.
```

*\*CLEAR를 사용하면 HEADERLINE의 DATA가 삭제된다.*

*\*CLEAR GT\_ITAB[ ]. 을 하면 BODY도 삭제 가능하다.*

```
GT_ITAB-CARRID = 'AA'.  
GT_ITAB-CONNID = 17.  
GT_ITAB-SEATMAX = 100.  
APPEND GT_ITAB.  
CLEAR GT_ITAB.
```

```
GT_ITAB-CARRID = 'LH'.  
GT_ITAB-CONNID = 20.  
GT_ITAB-SEATMAX = 150.  
APPEND GT_ITAB.  
CLEAR GT_ITAB.
```

```
GT_ITAB-CARRID = 'BB'.
```

```
GT_ITAB-CONNID = 15.
```

```
GT_ITAB-SEATMAX = 90.
```

```
APPEND GT_ITAB.
```

```
CLEAR GT_ITAB.
```

```
GT_ITAB-CARRID = 'AA'.
```

```
GT_ITAB-CONNID = 17.
```

```
GT_ITAB-SEATMAX = 33.
```

```
APPEND GT_ITAB.
```

```
CLEAR GT_ITAB.
```

*\*TABLE에 있는 DATA를 읽어보자*

*\*\*SY-INDEX 는 LOOP 횟수 , TABIX 는 INTERNAL TABLE의 INDEX*

```
READ TABLE GT_ITAB INDEX 3.
```

```
WRITE: / 'INDEX', SY-INDEX, 'TABIX:', SY-TABIX.
```

```
WRITE: GT_ITAB-CARRID,  
        GT_ITAB-CONNID,  
        GT_ITAB-SEATMAX.
```

Report: Z_0623_0510					
INDEX	0	TABIX:	3	BB	0015 90



### INTERNAL TABLE DATA 변경 |

INTERNAL TABLE의 한 라인을 변경하려면 MODIFY 구문을 사용한다. 해당 라인을 KEY, INDEX 조건으로 찾아 변경할 수 있다.

#### TABLE KEY를 이용하여 한 라인 변경

**\*MODIFY TABLE ITAB FROM WA [TRANSPORTING F1 F2 ... ].**

위의 구문을 사용하여 KEY 값을 기준으로 INTERNAL TABLE의 LINE을 변경한다. INTERNAL TABLE이 NON-UNIQUE KEY 이며 중복된 값이 존재할 때 MODIFY 구문을 수행할 시 첫 번째 라인이 변경된다. HEADER LINE이 있는 경우 FROM WA 구문을 생략할 수 있다. TRANSPORTING을 통해 해당 칼럼만 변경할 수 있다. 한 라인 전체를 변경하는 것보다 일부 칼럼만 변경하는 것이 성능이 좋긴 하나 영향이 크지 않다.

#### WHERE 조건을 이용하여 여러 라인 변경

**\*MODIFY ITAB FROM WA TRANSPORTING F1 F2 ... WHERE CONDITION.**

위의 구문을 사용하여 하나 이상의 라인을 변경할 수 있다. WHERE 조건을 이용하여 여러 칼럼을 조건에 추가할 수 있다.

INTERNAL TABLE을 LOOP 처리할 시 사용할 수 있는 구문	
AT FIRST.	INTERNAL TABLE의 첫 번째 값이 실행될 때 수행
AT NEW F1.	칼럼 F1에 새로운 값이 들어올 때 수행
AT END OF F1.	칼럼 F1의 값이 마지막일 때 수행
AT LAST.	INTERNAL TABLE의 마지막 값이 실행될 시 수행

#### INDEX를 이용하여 한 라인 변경

**\*MODIFY ITAB FROM WA [INDEX IDX] [TRANSPORTING F1 F2 ... ].**

위의 구문을 사용하여 INDEX를 이용한 해당 라인의 값을 변경할 수 있다. INDEX

를 이용해 값을 변경하기 때문에 STANDARD, SORTED TYPE의 INTERNAL TABLE에서만 사용할 수 있다. LOOP 구문 내에서는 INDEX 옵션은 생략할 수 있으며 해당 경우 현재 INTERNAL TABLE의 LINE INDEX 값을 변경하게 된다.

### INTERNAL TABLE DATA 삭제 |

INTERNAL TABLE의 한 라인을 삭제하려면 DELETE 구문을 사용한다. 해당 라인을 KEY와 INDEX 조건으로 찾아 삭제할 수 있다.

#### TABLE KEY를 이용하여 한 라인 삭제

**\*DELETE TABLE ITAB [FROM WA].**

**\*DELETE TABLE ITAB WITH TABLE KEY K1 = F1 ... KN = FN.**

위의 구문을 사용하여 KEY값 기준으로 INTERNAL TABLE의 라인을 삭제한다. NON-UNIQUE KEY로 설정된 STANDARD TYPE의 경우 WITH TABLE KEY 구문은 중복된 KEY DATA 중에서 한 건만 삭제한다.

#### WHERE 조건을 이용하여 여러 라인 삭제

**\*DELETE ITAB WHERE COND.**

하나 이상의 라인을 삭제하려면 WHERE 구문을 사용한다. WHERE 조건은 논리 연산으로 구성된 여러 칼럼의 조건들이 추가될 수 있다.

## INDEX를 이용하여 삭제

**\*DELETE ITAB [INDEX IDX].**

위의 구문을 이용하여 해당 라인의 값을 삭제할 수 있다. HASHED TYPE의 INTERNAL TABLE에서는 사용할 수 없으며 INDEX를 이용해 여러 라인을 한 번에 삭제할 수 있다. 밑에 구문은 INDEX N1 N2 사이의 LINE을 삭제한다. FROM N1 구문만 사용하면, N1 번째 INDEX 이후의 모든 DATA를 삭제한다. 이와 반대로 TO N2 구문만 사용하면 처음부터 N2 번째에 존재하는 DATA가 삭제된다.

**\*DELETE ITAB FROM N1 TO N2.**

**DELETE ITAB FROM N1.**

**DELETE ITAB TO N2.**

## ADJACENT DUPLICATE 구문을 이용해 중복 라인 삭제

**\*DELETE ADJACENT DUPLICATE ENTRIES FROM ITAB [COMPARING F1 F2 ... | ALL FIELDS] .**

ADJACENT DUPLICATE 구문을 이용하여 중복된 라인을 삭제할 수 있다. 구문을 수행하기 이전에 SORT 구문으로 INTERNAL TABLE을 정렬해야 원하는 결과를 얻을 수 있다. COMPARING 구문을 사용하지 않으면, TABLE KEY 값이 중복된 DATA를 삭제한다. KEY 값을 선언하지 않은 경우는 문자열의 선행 필드들이 DEFAULT KEY로 구성된다.

## INTERNAL TABLE DATA 읽기 |

INTERNAL TABLE에서 원하는 DATA를 읽으려면 READ 구문을 사용한다. 헤더 라인이 있으면 해당 데이터가 헤더 라인으로 복사되고, 그렇지 않으면 WORK AREA에 복사한다.

## TABLE KEY를 이용

**\*READ TABLE ITAB FROM WA INTO RESULT.**

**READ TABLE ITAB WITH TABLE KEY K1 = F1. KN = FN INTO RESULT**

KEY 값을 이용하여 값을 찾을 수 있다. RESULT는 READ 결과를 저장하게 되는 WORK AREA이다. HEADER LINE이 존재하는 INTERNAL TABLE은 INTO 이하를 생략하고 INTERNAL TABLE 이름 자체를 WORK AREA 자체로 사용할 수 있다. 성공시 SY-SUBRC 변수에 0을 반환하고, 실패하면 4를 반환한다. SY-TABIX 변수는 LINE의 INDEX를 반환한다. 두 번째 구문을 이용하면 같은 칼럼을 여러 번 사용할 수 없다.

## WORK AREA 할당

**\*READ TABLE ITAB WITH KEY K1 ... INTO WA [COMPARING F1 F2 ... | ALL FIELDS] [TRANSPORTING F1 F2 ... | ALL FIELDS | NO FIELDS] .**

READ 구문 수행 결과를 WORK AREA로 할당하는 구문이다. COMPARING 구문은 READ 구문의 결과 값에 비교 조건을 추가한다. 즉 COMPARING 구문 다음에 기술된 FIELD 들이 WORK AREA 의 값과 INTERNAL TABLE에 존재하는 값이 같으면 SY-SUBRC = 0을 반환하고, 같지 않으면 SY-SUBRC = 2 를 반환한다. TRANSPORTING은 READ한 결과를 해당 칼럼만 TARGET에 저장하는 기능을 수행한다.

## INDEX 를 이용해 READ 구문 수행

```
*READ TABLE ITAB INDEX IDX INTO RESULT.
```

INDEX를 이용하여 해당 라인의 값을 얻을 수 있다. INDEX를 사용하기 때문에 HASHED TYPE 의 INTERNAL TYPE의 INTERNAL TABLE에서는 사용할 수 없다. 성공 시에는 SY-SUBRC 변수에 0을 반환하고 실패시에는 4를 반환한다. 그리고 SY-TABIX 에는 INTERNAL TABLE의 INDEX 순번이 저장된다.

## READ BINARY SEARCH

```
*READ TABLE ITAB WITH KEY KI = F1 ... KN = FN INTO RESULT BINARY SEARCH.
```

STANDARD TYPE의 INTERNAL TABLE을 READ할 때 BINARY SEARCH를 이용할 수 있다. BINARY SEARCH 대상 칼럼을 기준으로 SORT 한 후 사용하며, READ 속도가 일반 READ 속도보다 훨씬 빠르다.

## DUMP ERROR

ABAP 프로그램에서 예기치 않은 에러 때문에 프로그램이 종료되는 것을 DUMP ERROR또는 RUNTIME ERROR라고 한다. RUNTIME ERROR 유형은 다양한 원인에서 발생하므로 에러 처리 후의 덤프 화면에 대해서 분석할 수 있는 안목을 길러야 한다. LONG TEXT 화면에서 ABAP RUNTIME ERROR에 대한 상세한 정보를 확인할 수 있으며 DEBUGGER 버튼을 선택하여 디버깅 화면으로 이동한 후, 에러가 발생한 시점에 변수들이 어떠한 값들을 저장하고 있는지 추적해야 한다. 백그라운드에서 실행된 것을 포함하여 ABAP 프로그램에서 발생한 모든 RUNTIME ERROR는 [ST22 | ABAP RUNTIME ERROR](#) 에서 조회할 수 있다.

[SE80 | 0511](#) | METHOD 호출 후 SORT하여 TABLE에 있는 DATA를 가져오자!

```
DATA: GT_CONNECTIONS TYPE ZBC400_T_CONNECTIONS,  
      GS_CONNECTION TYPE ZBC400_S_CONNECTION.
```

```
TRY.
```

```
CALL METHOD ZCL_BC400_FLIGHTMODEL=>GET_CONNECTIONS  
IMPORTING
```

```
ET_CONNECTIONS = GT_CONNECTIONS .
```

```
.  
CATCH ZCX_BC400_NO_DATA .
```

```
WRITE: / 'NO DATA FOUND !'(NDF).
```

```
ENDTRY.
```

*\*DEPTIME을 오름차순 기준으로 정렬하라*

```
SORT GT_CONNECTIONS ASCENDING BY DEPTIME.
```

```
LOOP AT GT_CONNECTIONS INTO GS_CONNECTION.
```

```
WRITE:/ GS_CONNECTION-CARRID,
```

```
GS_CONNECTION-CONNID,
```

```
GS_CONNECTION-CITYFROM,
```

```
GS_CONNECTION-AIRPFROM,
```

```
GS_CONNECTION-CITYTO,
```

```
GS_CONNECTION-AIRPTO,
```

```
GS_CONNECTION-FLTIME,
```

```
GS_CONNECTION-DEPTIME,
```

```
GS_CONNECTION-ARRTIME.
```

```
ENDLOOP.
```

Report Z\_0823\_0511

LH	2407	BERLIN	TXL	FRANKFURT	FRA	1:05	07:10:00	08:15:00
AA	0064	SAN FRANCISCO	SFO	NEW YORK	JFK	5:21	09:00:00	17:21:00
DL	1984	SAN FRANCISCO	SFO	NEW YORK	JFK	5:25	10:00:00	18:25:00
LH	0400	FRANKFURT	FRA	NEW YORK	JFK	7:24	10:10:00	11:34:00
LH	2402	FRANKFURT	FRA	BERLIN	SXF	1:05	10:30:00	11:35:00
AZ	0790	ROME	FCO	OSAKA	KIX	13:35	10:35:00	08:10:00
UA	3517	FRANKFURT	FRA	NEW YORK	JFK	8:15	10:40:00	12:55:00
AA	0017	NEW YORK	JFK	SAN FRANCISCO	SFO	6:01	11:00:00	14:01:00
AZ	0789	TOKYO	TYO	ROME	FCO	15:40	11:45:00	19:25:00
AZ	0788	ROME	FCO	TOKYO	TYO	12:55	12:00:00	08:55:00
JL	0407	TOKYO	NRT	FRANKFURT	FRA	12:05	13:30:00	17:35:00
LH	0402	FRANKFURT	FRA	NEW YORK	JFK	7:35	13:30:00	15:05:00
UA	0941	FRANKFURT	FRA	SAN FRANCISCO	SFO	11:36	14:30:00	17:06:00
UA	3504	SAN FRANCISCO	SFO	FRANKFURT	FRA	10:30	15:00:00	10:30:00
SQ	0158	SINGAPORE	SIN	JAKARTA	JKT	1:35	15:25:00	16:00:00
SQ	0015	SAN FRANCISCO	SFO	SINGAPORE	SIN	18:45	16:00:00	02:45:00
UA	3516	NEW YORK	JFK	FRANKFURT	FRA	7:25	16:20:00	05:45:00
SQ	0988	SINGAPORE	SIN	TOKYO	TYO	6:40	16:35:00	00:15:00
SQ	0002	SINGAPORE	SIN	SAN FRANCISCO	SFO	18:25	17:00:00	19:25:00
DL	1699	NEW YORK	JFK	SAN FRANCISCO	SFO	6:22	17:15:00	20:37:00
LH	0401	NEW YORK	JFK	FRANKFURT	FRA	7:15	18:30:00	07:45:00
AZ	0555	ROME	FCO	FRANKFURT	FRA	2:05	19:00:00	21:05:00
DL	0106	NEW YORK	JFK	FRANKFURT	FRA	7:55	19:35:00	09:30:00
JL	0406	FRANKFURT	FRA	TOKYO	NRT	11:15	20:25:00	15:40:00
QF	0006	FRANKFURT	FRA	SINGAPORE	SIN	11:10	20:55:00	15:05:00
QF	0005	SINGAPORE	SIN	FRANKFURT	FRA	13:45	22:50:00	05:35:00

SE80 | 0512 | 0511의 HEADER LINE이 있는 경우를 수행해보자

DATA: ITAB TYPE ZBC400\_T\_CONNECTIONS WITH HEADER LINE.

TRY.

CALL METHOD ZCL\_BC400\_FLIGHTMODEL=>GET\_CONNECTIONS

IMPORTING

ET\_CONNECTIONS = ITAB[] .

CATCH ZCX\_BC400\_NO\_DATA .

WRITE: / 'NO DATA FOUND !'(NDF).

ENDTRY.

SORT ITAB ASCENDING BY DEPTIME.

LOOP AT ITAB.

WRITE:/ ITAB-CARRID,

ITAB-CONNID,

ITAB-CITYFROM,

ITAB-AIRPFROM,

ITAB-CITYTO,

ITAB-AIRPTO,

ITAB-FLTIME,

ITAB-DEPTIME,

ITAB-ARRTIME.

ENDLOOP.

Report Z\_0823\_0512

LH	2407	BERLIN	TXL	FRANKFURT	FRA	1:05	07:10:00	08:15:00
AA	0064	SAN FRANCISCO	SFO	NEW YORK	JFK	5:21	09:00:00	17:21:00
DL	1984	SAN FRANCISCO	SFO	NEW YORK	JFK	5:25	10:00:00	18:25:00
LH	0400	FRANKFURT	FRA	NEW YORK	JFK	7:24	10:10:00	11:34:00
LH	2402	FRANKFURT	FRA	BERLIN	SXF	1:05	10:30:00	11:35:00
AZ	0790	ROME	FCO	OSAKA	KIX	13:35	10:35:00	08:10:00
UA	3517	FRANKFURT	FRA	NEW YORK	JFK	8:15	10:40:00	12:55:00
AA	0017	NEW YORK	JFK	SAN FRANCISCO	SFO	6:01	11:00:00	14:01:00
AZ	0789	TOKYO	TYO	ROME	FCO	15:40	11:45:00	19:25:00
AZ	0788	ROME	FCO	TOKYO	TYO	12:55	12:00:00	08:55:00
JL	0407	TOKYO	NRT	FRANKFURT	FRA	12:05	13:30:00	17:35:00
LH	0402	FRANKFURT	FRA	NEW YORK	JFK	7:35	13:30:00	15:05:00
UA	0941	FRANKFURT	FRA	SAN FRANCISCO	SFO	11:36	14:30:00	17:06:00
UA	3504	SAN FRANCISCO	SFO	FRANKFURT	FRA	10:30	15:00:00	10:30:00
SQ	0158	SINGAPORE	SIN	JAKARTA	JKT	1:35	15:25:00	16:00:00
SQ	0015	SAN FRANCISCO	SFO	SINGAPORE	SIN	18:45	16:00:00	02:45:00
UA	3516	NEW YORK	JFK	FRANKFURT	FRA	7:25	16:20:00	05:45:00
SQ	0988	SINGAPORE	SIN	TOKYO	TYO	6:40	16:35:00	00:15:00
SQ	0002	SINGAPORE	SIN	SAN FRANCISCO	SFO	18:25	17:00:00	19:25:00
DL	1699	NEW YORK	JFK	SAN FRANCISCO	SFO	6:22	17:15:00	20:37:00
LH	0401	NEW YORK	JFK	FRANKFURT	FRA	7:15	18:30:00	07:45:00
AZ	0555	ROME	FCO	FRANKFURT	FRA	2:05	19:00:00	21:05:00
DL	0106	NEW YORK	JFK	FRANKFURT	FRA	7:55	19:35:00	09:30:00
JL	0406	FRANKFURT	FRA	TOKYO	NRT	11:15	20:25:00	15:40:00
QF	0006	FRANKFURT	FRA	SINGAPORE	SIN	11:10	20:55:00	15:05:00
QF	0005	SINGAPORE	SIN	FRANKFURT	FRA	13:45	22:50:00	05:35:00

SE80 | 0513 | 0511의 HEADER LINE이 있는 경우를 수행해보자

*\*강사님 코드*

```
DATA: GT_CONNECTIONS TYPE ZBC400_T_CONNECTIONS,  
      GS_CONNECTION TYPE ZBC400_S_CONNECTION.
```

```
DATA: ITAB LIKE GT_CONNECTIONS WITH HEADER LINE.
```

*\*GET DATA*

TRY.

```
CALL METHOD ZCL_BC400_FLIGHTMODEL=>GET_CONNECTIONS
```

*\* EXPORTING*

*\* IV\_CARRID = SPACE*

IMPORTING

```
ET_CONNECTIONS = GT_CONNECTIONS.
```

```
CATCH ZCX_BC400_NO_DATA .
```

```
WRITE:/ 'No Data Found!'(ndf).
```

*\* CATCH ZCX\_BC400\_NO\_AUTH .*

ENDTRY.

```
SORT GT_CONNECTIONS ASCENDING BY DEPTIME.
```

```
LOOP AT GT_CONNECTIONS INTO GS_CONNECTION.
```

```
MOVE-CORRESPONDING GS_CONNECTION TO ITAB.
```

```
APPEND ITAB.
```

```
WRITE :/ ITAB-CARRID,
```

```
ITAB-CONNID,
```

```
ITAB-CITYFROM,
```

```
ITAB-AIRPFROM,
```

```
ITAB-CITYTO,
```

```
ITAB-AIRPTO,
```

```
ITAB-FLTIME,
```

```
ITAB-DEPTIME,
```

```
ITAB-ARRTIME.
```

```
CLEAR ITAB.
```

```
ENDLOOP.
```

Report: Z\_0623\_0513

LH	2407	BERLIN	TXL	FRANKFURT	FRA	1:05	07:10:00	08:15:00
AA	0064	SAN FRANCISCO	SFO	NEW YORK	JFK	5:21	09:00:00	17:21:00
DL	1984	SAN FRANCISCO	SFO	NEW YORK	JFK	5:25	10:00:00	18:25:00
LH	0400	FRANKFURT	FRA	NEW YORK	JFK	7:24	10:10:00	11:34:00
LH	2402	FRANKFURT	FRA	BERLIN	SWF	1:05	10:30:00	11:35:00
AZ	0790	ROME	FCO	OSAKA	KIX	13:35	10:35:00	08:10:00
UA	3517	FRANKFURT	FRA	NEW YORK	JFK	8:15	10:40:00	12:55:00
AA	0017	NEW YORK	JFK	SAN FRANCISCO	SFO	6:01	11:00:00	14:01:00
AZ	0789	TOKYO	TYO	ROME	FCO	15:40	11:45:00	19:25:00
AZ	0788	ROME	FCO	TOKYO	TYO	12:55	12:00:00	08:55:00
JL	0407	TOKYO	NRT	FRANKFURT	FRA	12:05	13:30:00	17:35:00
LH	0402	FRANKFURT	FRA	NEW YORK	JFK	7:35	13:30:00	15:05:00
UA	0941	FRANKFURT	FRA	SAN FRANCISCO	SFO	11:36	14:30:00	17:06:00
UA	3504	SAN FRANCISCO	SFO	FRANKFURT	FRA	10:30	15:00:00	10:30:00
SQ	0158	SINGAPORE	SIN	JAKARTA	JKT	1:35	15:25:00	16:00:00
SQ	0015	SAN FRANCISCO	SFO	SINGAPORE	SIN	18:45	16:00:00	02:45:00
UA	3516	NEW YORK	JFK	FRANKFURT	FRA	7:25	16:20:00	05:45:00
SQ	0988	SINGAPORE	SIN	TOKYO	TYO	6:40	16:35:00	00:15:00
SQ	0002	SINGAPORE	SIN	SAN FRANCISCO	SFO	18:25	17:00:00	19:25:00
DL	1999	NEW YORK	JFK	SAN FRANCISCO	SFO	6:22	17:15:00	20:37:00
LH	0401	NEW YORK	JFK	FRANKFURT	FRA	7:15	18:30:00	07:45:00
AZ	0555	ROME	FCO	FRANKFURT	FRA	2:05	19:00:00	21:05:00
DL	0106	NEW YORK	JFK	FRANKFURT	FRA	7:55	19:35:00	09:30:00
JL	0408	FRANKFURT	FRA	TOKYO	NRT	11:15	20:25:00	15:40:00
QF	0006	FRANKFURT	FRA	SINGAPORE	SIN	11:10	20:55:00	15:05:00
QF	0005	SINGAPORE	SIN	FRANKFURT	FRA	13:45	22:50:00	05:35:00

SE80 | 0514 | 실습해보자

*\*GT\_ITAB에 SFLIGHT-CARRID와 동일한 값을 가진 SCARR-CARRID  
의 CARRNAME DATA를 추가해보자*

```
DATA: BEGIN OF GS_FLIGHT,  
    CARRID TYPE SFLIGHT-CARRID,  
    CONNID TYPE SFLIGHT-CONNID,  
    FLDATE TYPE SFLIGHT-FLDATE,  
    SEATSMAX TYPE SFLIGHT-SEATSMAX,  
    SEATSOCC TYPE SFLIGHT-SEATSOCC,  
END OF GS_FLIGHT.
```

```
DATA: GT_SFLIGHT LIKE TABLE OF GS_FLIGHT.
```

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

```
DATA: GT_SCARR LIKE TABLE OF GS_SCARR.
```

```
TYPES: BEGIN OF GT_TYPE,  
    CARRID TYPE SFLIGHT-CARRID,  
    CARRNAME TYPE SCARR-CARRNAME,  
    CONNID TYPE SFLIGHT-CONNID,  
    FLDATE TYPE SFLIGHT-FLDATE,
```

```
    SEATSMAX TYPE SFLIGHT-SEATSMAX,  
    SEATSOCC TYPE SFLIGHT-SEATSOCC,  
    PERCENTAGE TYPE P LENGTH 3 DECIMALS 2,  
END OF GT_TYPE.
```

```
DATA GS_ITAB TYPE GT_TYPE.  
DATA GT_ITAB TYPE TABLE OF GT_TYPE.
```

*\*SFLIGHT에서 DATA SELECT / CARRID 가 AA 이거나 LH 인것만 가져오자*

```
SELECT CARRID CONNID FLDATE SEATSMAX SEATSOCC  
    FROM SFLIGHT  
    INTO TABLE GT_SFLIGHT  
    WHERE CARRID = 'AA'  
    OR CARRID = 'LH'.
```

*\*SCARR에서 CARRID CARRNAME을 가져오자*

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

\*

```
LOOP AT GT_SFLIGHT INTO GS_FLIGHT.  
    MOVE-CORRESPONDING GS_FLIGHT TO GS_ITAB.  
    READ TABLE GT_SCARR INTO GS_SCARR  
        WITH KEY CARRID = GS_FLIGHT-CARRID.  
    GS_ITAB-CARRNAME = GS_SCARR-CARRNAME.  
    GS_ITAB-PERCENTAGE = GS_ITAB-SEATSOCC / GS_ITAB-
```

SEATSMAX \* 100.

APPEND GS\_ITAB TO GT\_ITAB.

CLEAR: GS\_ITAB, GS\_SCARR, GS\_FLIGHT.

ENDLOOP.

BREAK-POINT.

2_0023_0514	/ 2_0023_0514	/ 64	DY-GURIC	0							
EVENT	/ START-OF-SELECTION	77	DY-TAB1X	1							
Desktop 1	Desktop 2	Desktop 3	Standard	Structures	<b>Tables</b>	Objects	Detail	Data Explorer	Break/Watchpoints	Diff	Script
<div> <div>Tables</div> <div>Table Contents</div> </div>											
<div> <div>Table</div> <div>DY_TAB</div> </div>											
<div> <div>Attributes</div> <div>Standard (103x704)</div> </div>											
<div> <div>Insert Columns</div> <div>Columns</div> </div>											
Row	CARRID [C3]	CARRIER [C20]	CONNID [N4]	FLDATE [D8]	SEATSMAX [X4]	SEATSOCC [X4]	PERCENTAGE [P]				
1	AA	American Airlines 0017	20000109	385	385	100.00					
2	AA	American Airlines 0017	20000210	385	385	100.00					
3	AA	American Airlines 0017	20000313	385	383	99.48					
4	AA	American Airlines 0017	20000414	385	377	97.92					
5	AA	American Airlines 0017	20000516	385	385	100.00					
6	AA	American Airlines 0017	20000617	385	385	100.00					
7	AA	American Airlines 0017	20000719	385	380	98.70					
8	AA	American Airlines 0017	20000820	385	385	100.00					
9	AA	American Airlines 0017	20000921	385	72	18.70					
10	AA	American Airlines 0017	20001023	385	73	18.96					
11	AA	American Airlines 0017	20001124	385	88	17.66					
12	AA	American Airlines 0017	20001226	385	35	9.09					
13	AA	American Airlines 0017	20010127	385	20	5.19					
14	AA	American Airlines 0064	20000111	330	330	100.00					
15	AA	American Airlines 0064	20000212	330	330	100.00					
16	AA	American Airlines 0064	20000315	330	330	100.00					
17	AA	American Airlines 0064	20000416	330	330	100.00					
18	AA	American Airlines 0064	20000518	330	330	100.00					
19	AA	American Airlines 0064	20000619	330	324	98.18					
20	AA	American Airlines 0064	20000721	330	321	97.27					
21	AA	American Airlines 0064	20000822	330	218	66.06					
22	AA	American Airlines 0064	20000923	330	212	64.24					
23	AA	American Airlines 0064	20001025	330	138	41.82					

SE80 | 0515 | MODIFY 수정해보자

TYPES : BEGIN OF TS\_TYPE1,

CARRID TYPE S\_CARR\_ID,

CONNID TYPE S\_CONN\_ID,

SEATESMAX TYPE S\_SEATESMAX,

NO TYPE I,

END OF TS\_TYPE1.

TYPES : BEGIN OF TS\_TYPE2,

CARRID TYPE S\_CARR\_ID,

CONNID TYPE S\_CONN\_ID,

SEATESMAX TYPE S\_SEATESMAX,

END OF TS\_TYPE2.

DATA : GS\_STR TYPE TS\_TYPE1.

DATA : GT\_ITAB LIKE TABLE OF GS\_STR.

DATA : GS\_WA TYPE TS\_TYPE2.

DATA : GT\_ITAB2 LIKE TABLE OF GS\_WA.

DATA GV\_TABIX TYPE SY-TABIX.

GS\_STR-CARRID = 'AA'.

GS\_STR-CONNID = '17'.

GS\_STR-SEATESMAX = 100.

GS\_STR-NO = 1.

APPEND GS\_STR TO GT\_ITAB.

CLEAR GS\_STR.



```
GS_STR-CARRID = 'LH'.
GS_STR-CONNID = '20'.
GS_STR-SEATESMAX = 150.
GS_STR-NO = 2.
APPEND GS_STR TO GT_ITAB.
CLEAR GS_STR.
```

```
GS_STR-CARRID = 'BB'.
GS_STR-CONNID = '15'.
GS_STR-SEATESMAX = 90.
GS_STR-NO = 3.
APPEND GS_STR TO GT_ITAB.
CLEAR GS_STR.
```

```
GS_STR-CARRID = 'AA'.
GS_STR-CONNID = '17'.
GS_STR-SEATESMAX = 33.
GS_STR-NO = 4.
APPEND GS_STR TO GT_ITAB.
CLEAR GS_STR.
```

```
GS_WA-CARRID = 'AA'.
GS_WA-CONNID = '17'.
GS_WA-SEATESMAX = 5.
APPEND GS_WA TO GT_ITAB2.
```

```
CLEAR GS_WA.
```

```
GS_WA-CARRID = 'LH'.
GS_WA-CONNID = '20'.
GS_WA-SEATESMAX = 10.
APPEND GS_WA TO GT_ITAB2.
CLEAR GS_WA.
```

```
GS_WA-CARRID = 'BB'.
GS_WA-CONNID = '15'.
GS_WA-SEATESMAX = 2.
APPEND GS_WA TO GT_ITAB2.
CLEAR GS_WA.
```

```
GS_WA-CARRID = 'AA'.
GS_WA-CONNID = '17'.
GS_WA-SEATESMAX = 4.
APPEND GS_WA TO GT_ITAB2.
CLEAR GS_WA.
```

*\*GT\_ITAB의 DATA를 GS\_STR로 이전 : 왜? 헤더라인이 없으니까!*

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

```
    GV_TABIX = SY-TABIX.
```

*"GS\_WA의 DATA를 GT\_ITAB2 로 이전하여 수정 : 왜? 헤더라인이 없으니까!"*

```
    READ TABLE GT_ITAB2 INTO GS_WA
        WITH KEY CARRID = GS_STR-CARRID
```



CONNID = GS\_STR-CONNID.

GS\_STR-SEATESMAX = GS\_WA-SEATESMAX.

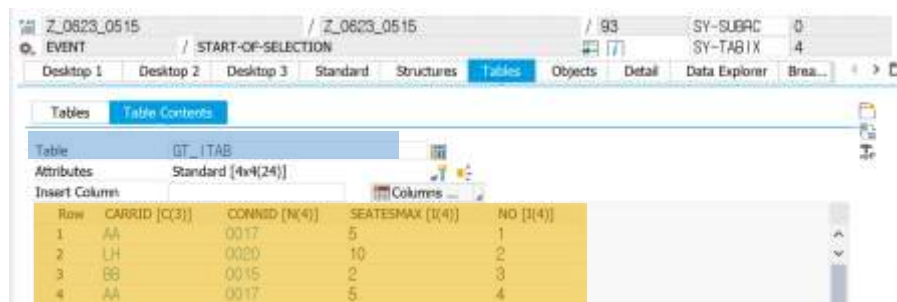
MODIFY GT\_ITAB FROM GS\_STR INDEX GV\_TABIX.

CLEAR GS\_STR.

ENDLOOP.

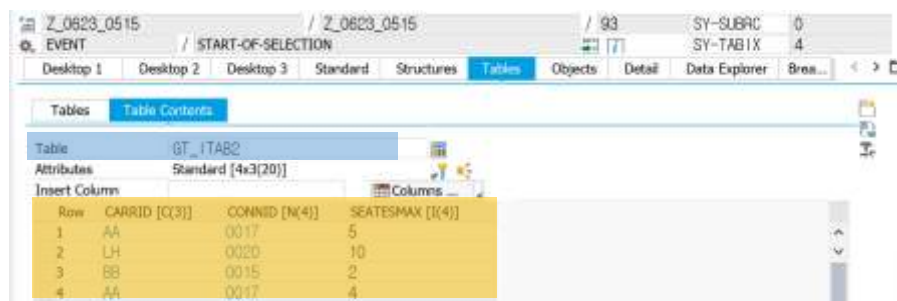
*\*즉 수정된 내역은 GT\_ITAB2에서 확인해야지~!*

BREAK-POINT.



The screenshot shows the SAP Table GT\_ITAB in the 'Table Contents' view. The table has 4 columns: CARRID [C(3)], CONNID [N(4)], SEATESMAX [I(4)], and NO [I(4)]. The data is as follows:

Row	CARRID [C(3)]	CONNID [N(4)]	SEATESMAX [I(4)]	NO [I(4)]
1	AA	0017	5	1
2	LH	0020	10	2
3	BB	0015	2	3
4	AA	0017	5	4



The screenshot shows the SAP Table GT\_ITAB2 in the 'Table Contents' view. The table has 4 columns: CARRID [C(3)], CONNID [N(4)], and SEATESMAX [I(4)]. The data is as follows:

Row	CARRID [C(3)]	CONNID [N(4)]	SEATESMAX [I(4)]
1	AA	0017	5
2	LH	0020	10
3	BB	0015	2
4	AA	0017	4