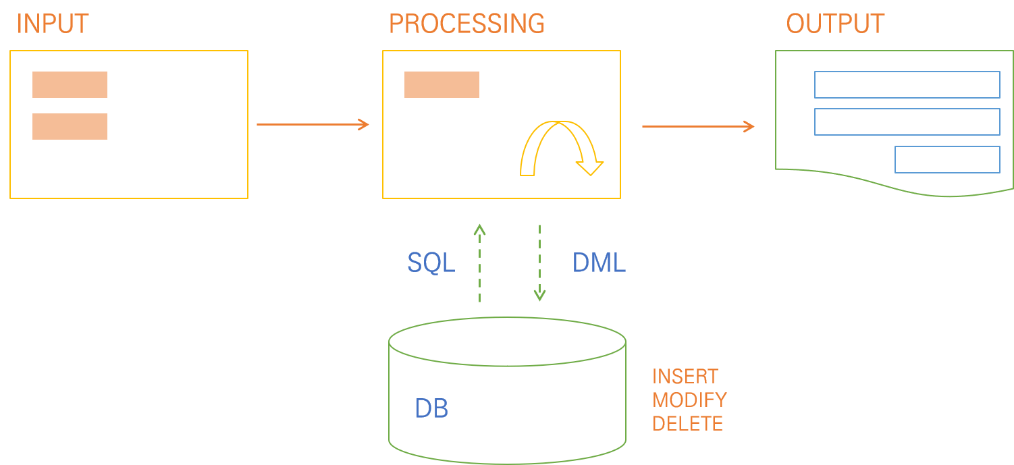
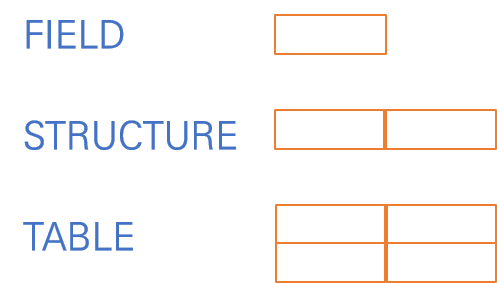
0616 | ABAP

PROGRAM | 처리과정 \*OUTPUT은 LIST로 구성된다.



DATA OBJECT | DATA 값을 참고하여 값을 저장할 수 있는 변수



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 DATA TYPE | COMPLETED PREDEFINED TYPE |

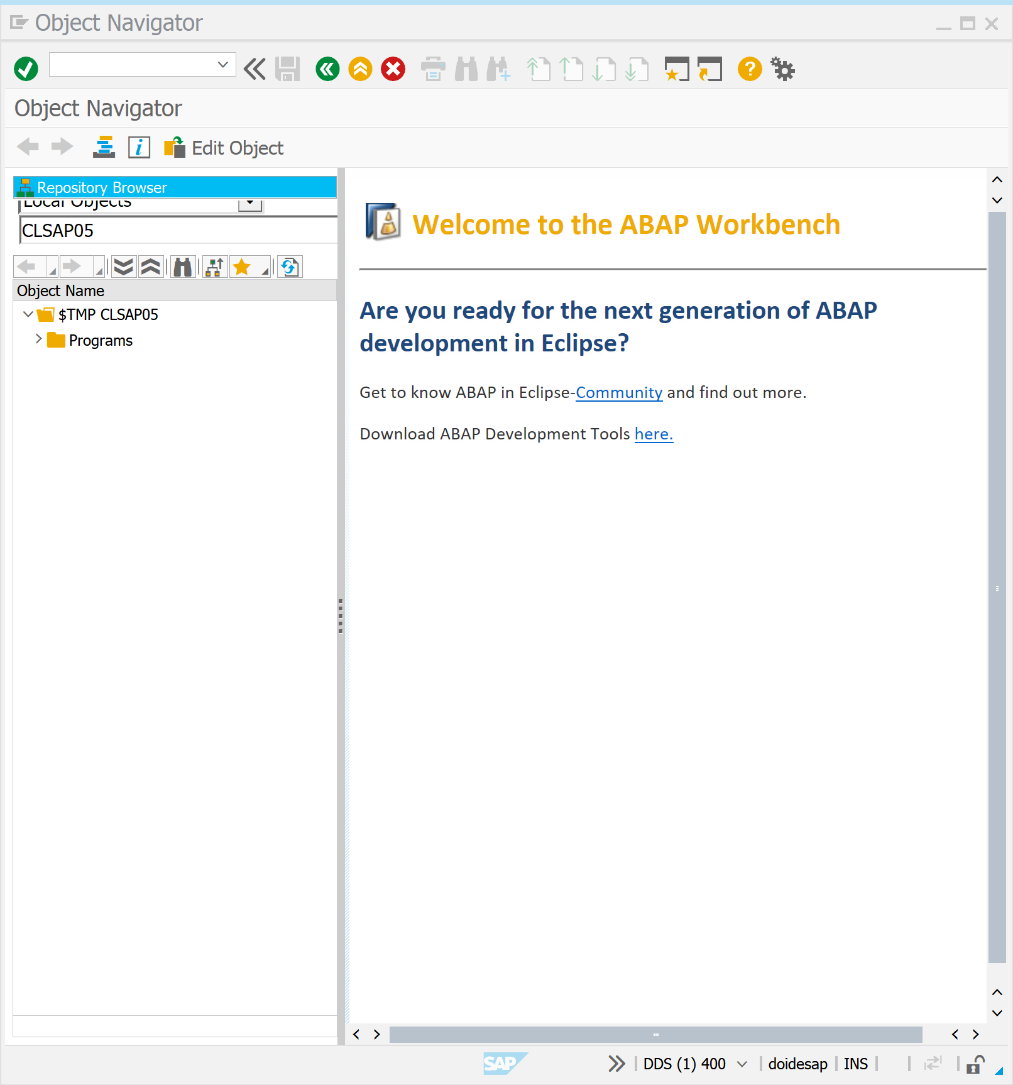
|  |  |
| --- | --- |
| D | DATE| YYYY.MM.DD | 8자리 |
| T | TIME | HHMMSS | 6자리 |
| I\* | INTEGER | 4BYTE |
| F | FLOAT | 8BYTE |
| STRING | 문자열 | DATA 삽입에 따라 가변적 |
| XSTRING | 긴 문자열 | 16진수로 변경 |

\*INTEGER는 정수만 지원한다.

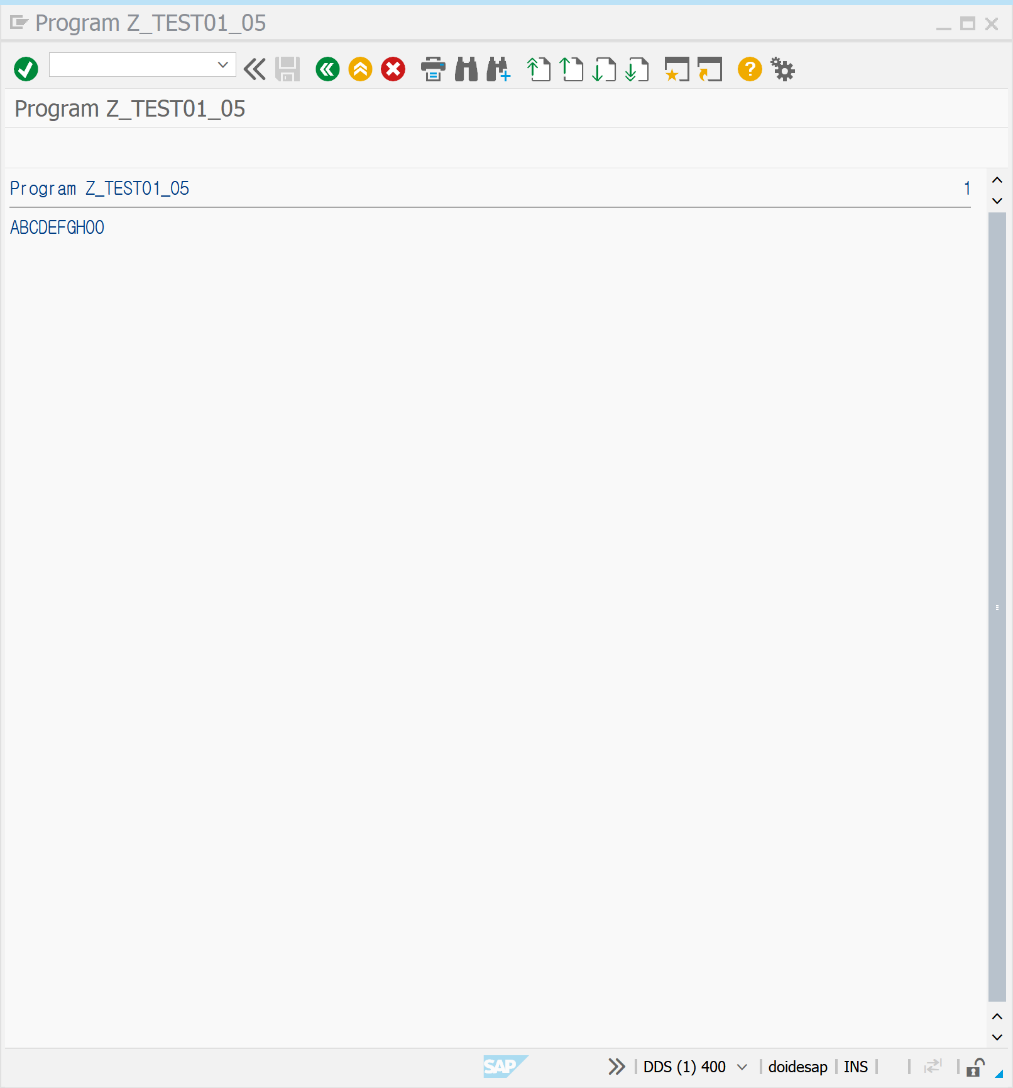
STANDARD | INCOMPLETED PREDEFINED TYPE | 길이를 지정할 수 있다.

|  |  |
| --- | --- |
| C | CHAR | 문자형 |
| L | LENGTH | 정수형 |
| P | PACKED | LENGTH\_정수형 DECIMAL\_소수형 |
| N | NUMERIC | 숫자를 문자처럼 인식한다. |

SE80 | OBJECT NAVIGATOR | ABAP 개발과 관련된 WORKBENCH 통합.



SE80 | TEST01 | FIELD 를 생성해보자  
\*FIELD 를 생성해보자  
\*\*FIELD 는 DATA 데이터 이름 TYPE 데이터 타입 의 형태로 이루어진다.  
  
DATA GV\_VAL TYPE C.  
DATA GV\_VAL2 TYPE C LENGTH 5.  
\*LEGTH 를 통해  DATA의 길이를 지정할 수 있다.  
  
DATA GV\_DATE TYPE D.  
DATA GV\_TIME TYPE T.  
DATA GV\_INT TYPE I.  
DATA GV\_STRING TYPE STRING.  
DATA GV\_XSTRING TYPE XSTRING.  
  
DATA GV\_CH TYPE C LENGTH 10.  
DATA GV\_NUM TYPE N LENGTH 5.  
DATA GV\_PAC TYPE P LENGTH 3 DECIMALS 3.  
\*LENGTH 정수형 길이를 지정한 다음 DECIMALS 소수형 길이를 지정한다.  
  
GV\_CH = 'ABCDEFGH0000'.  
\*작은따옴표 안의 문자는 대/소문자를 구별한다.  
\*\*해당 변수에  값을 지정해준다.  
  
WRITE GV\_CH.  
\*WRITE는 실행 후 해당 DATA를 화면에 출력해준다.



|  |  |
| --- | --- |
| NEW LINE | 개행한다. \*또는 / |
| : | CHAINED STATEMENT 문장을 연결한다. |

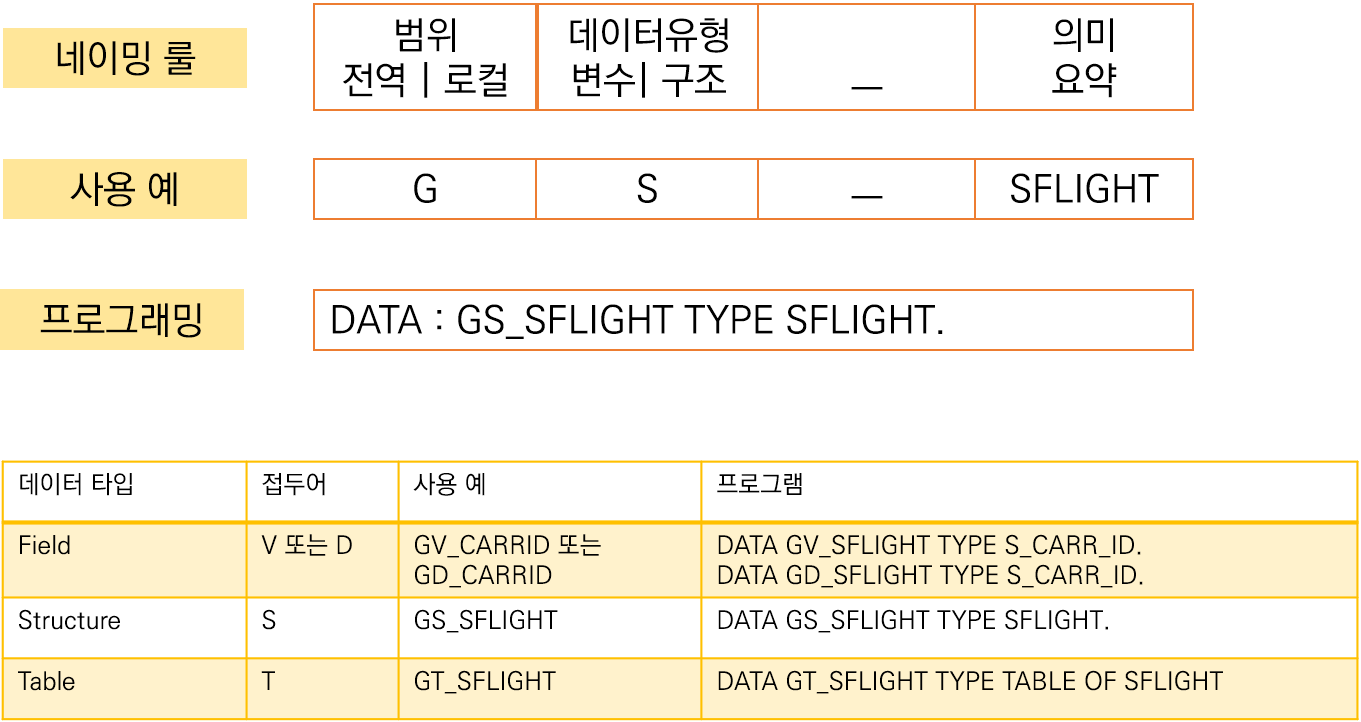
주석 처리 |

|  |  |
| --- | --- |
| CTRL + < | 전체 주석처리 |
| “ OR \* | 해당 줄 주석처리 |

단축키 |

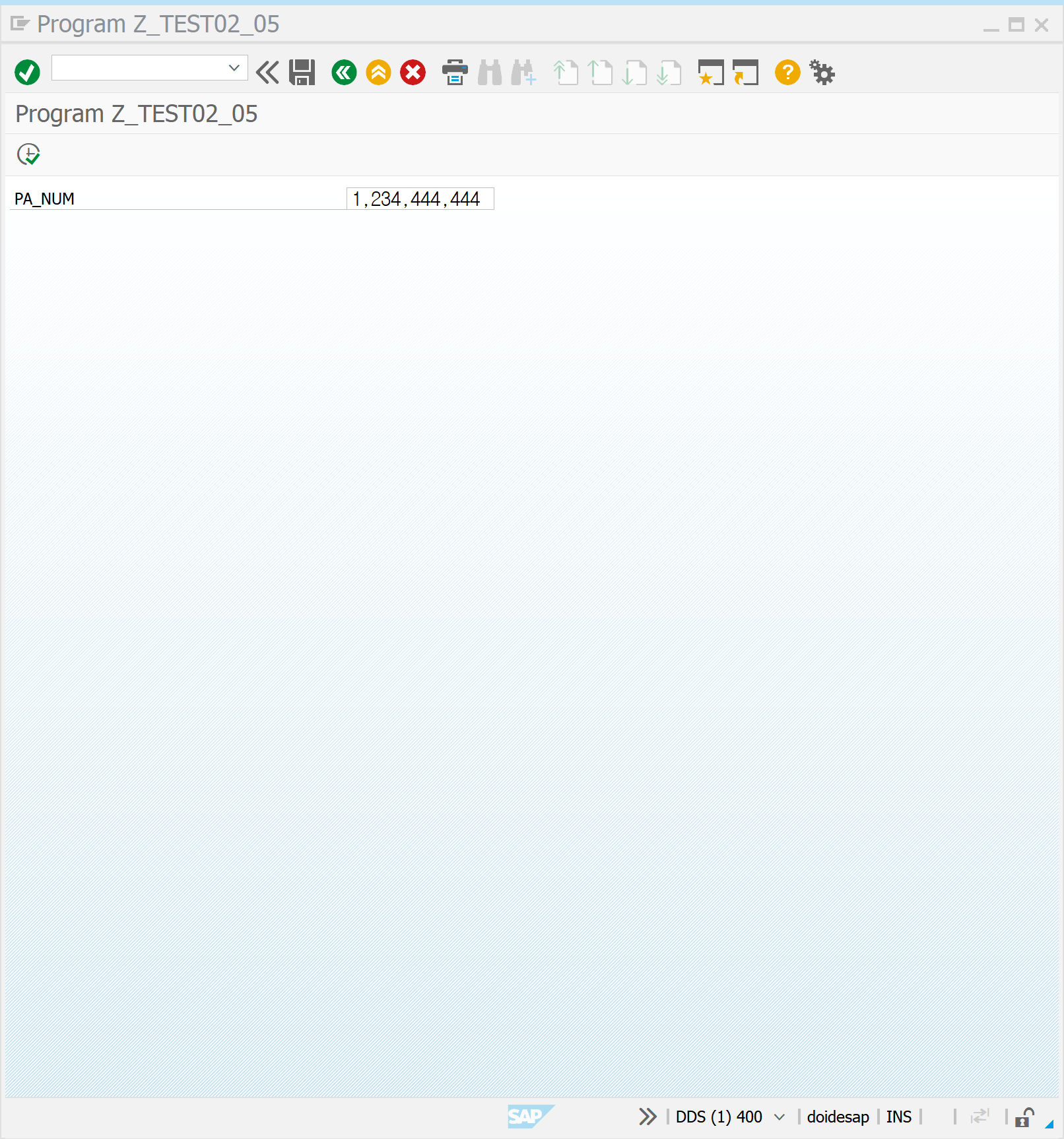
|  |  |
| --- | --- |
| TAB | 자동완성 |
| CTRL + D | 전에 입력한 구문 복사 |
| PRETTY P | PRETTY PRINTER | 소문자를 대문자 전환 \*SHIFT+F1 |

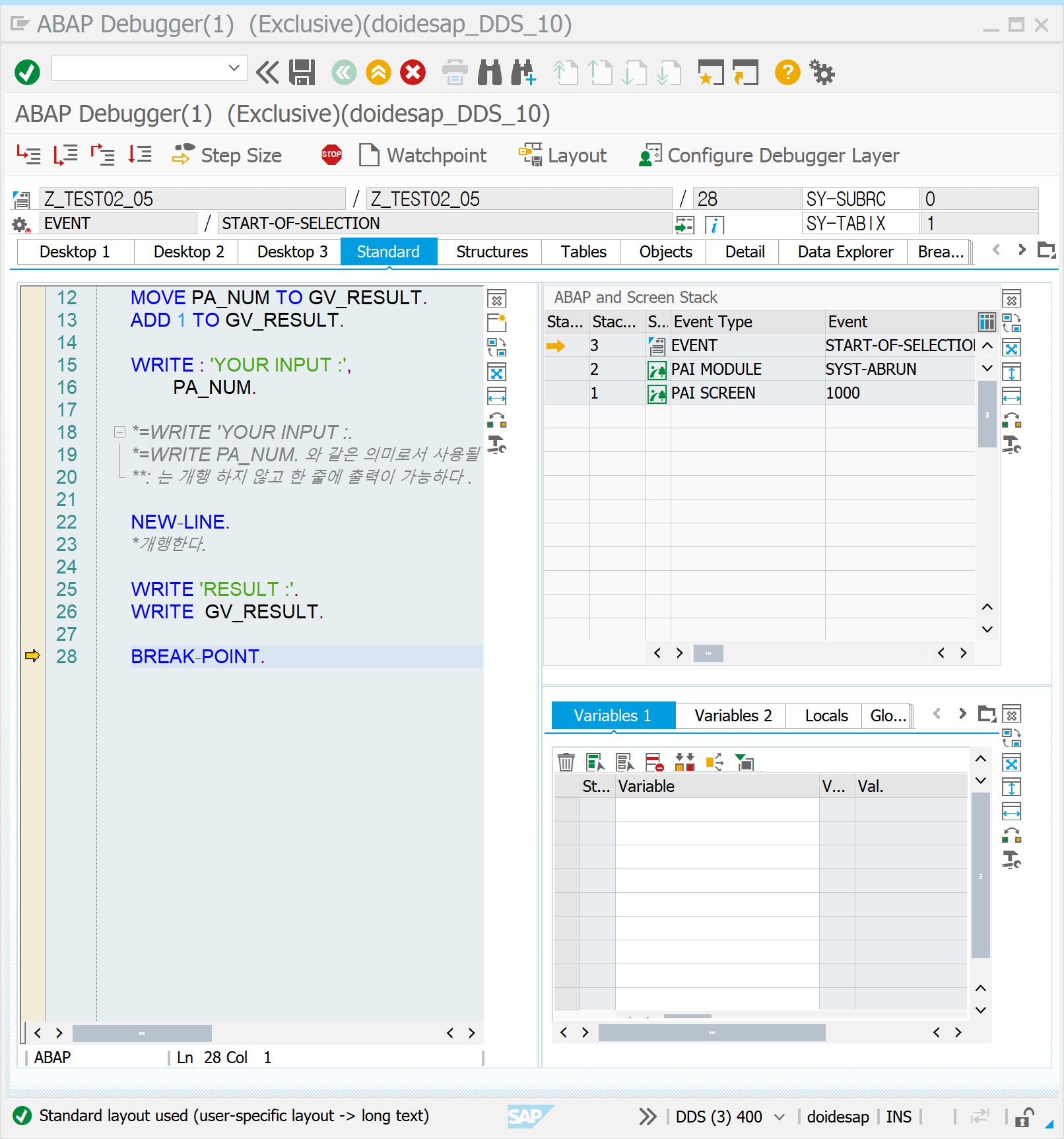
NAMING | 변수 설정

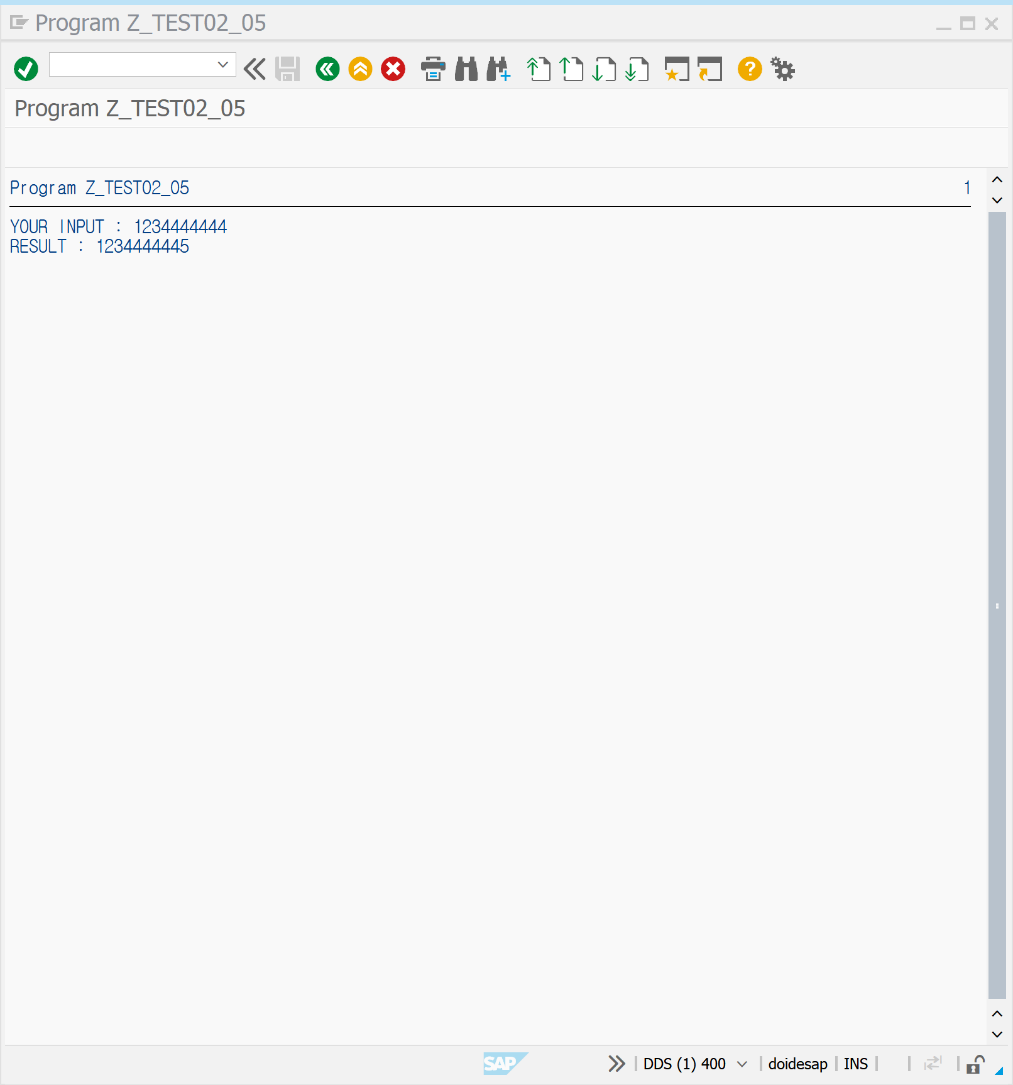


SE80 | TEST02 | PARAMETERS로 DATA를 받아와 지정해보자

PARAMETERS PA\_NUM TYPE I.  
\*PARAMETERS는 입력 받을 수 있는 SCREEN을 출력하며,   
\*사용자에게 입력 받은 값을  변수에 지정한다.   
DATA GV\_RESULT TYPE I.  
MOVE PA\_NUM TO GV\_RESULT.  
ADD 1 TO GV\_RESULT.  
  
WRITE : 'YOUR INPUT :',  
        PA\_NUM.  
  
\*=WRITE 'YOUR INPUT :.  
\*=WRITE PA\_NUM. 와 같은 의미로서 사용될 수 있다.  
\*\*: 는 개행 하지 않고 한 줄에 출력이 가능하다 .  
  
NEW-LINE.  
\*개행한다.  
  
WRITE 'RESULT :'.  
WRITE  GV\_RESULT.  
  
BREAK-POINT.



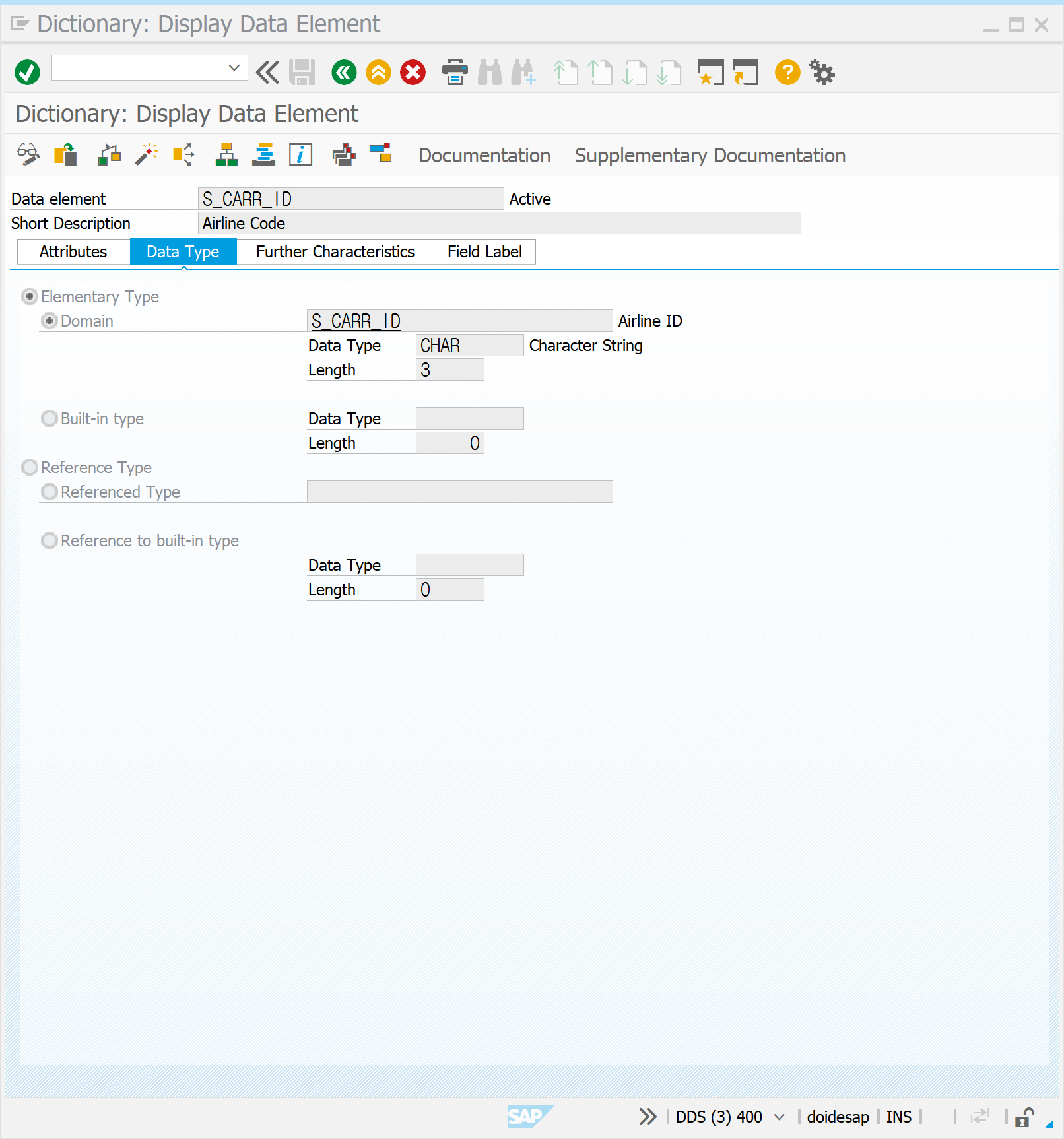




SE80 | TEST02\_01 | LOCAL TYPE과 GLOBAL TYPE의 변수를 지정해보자

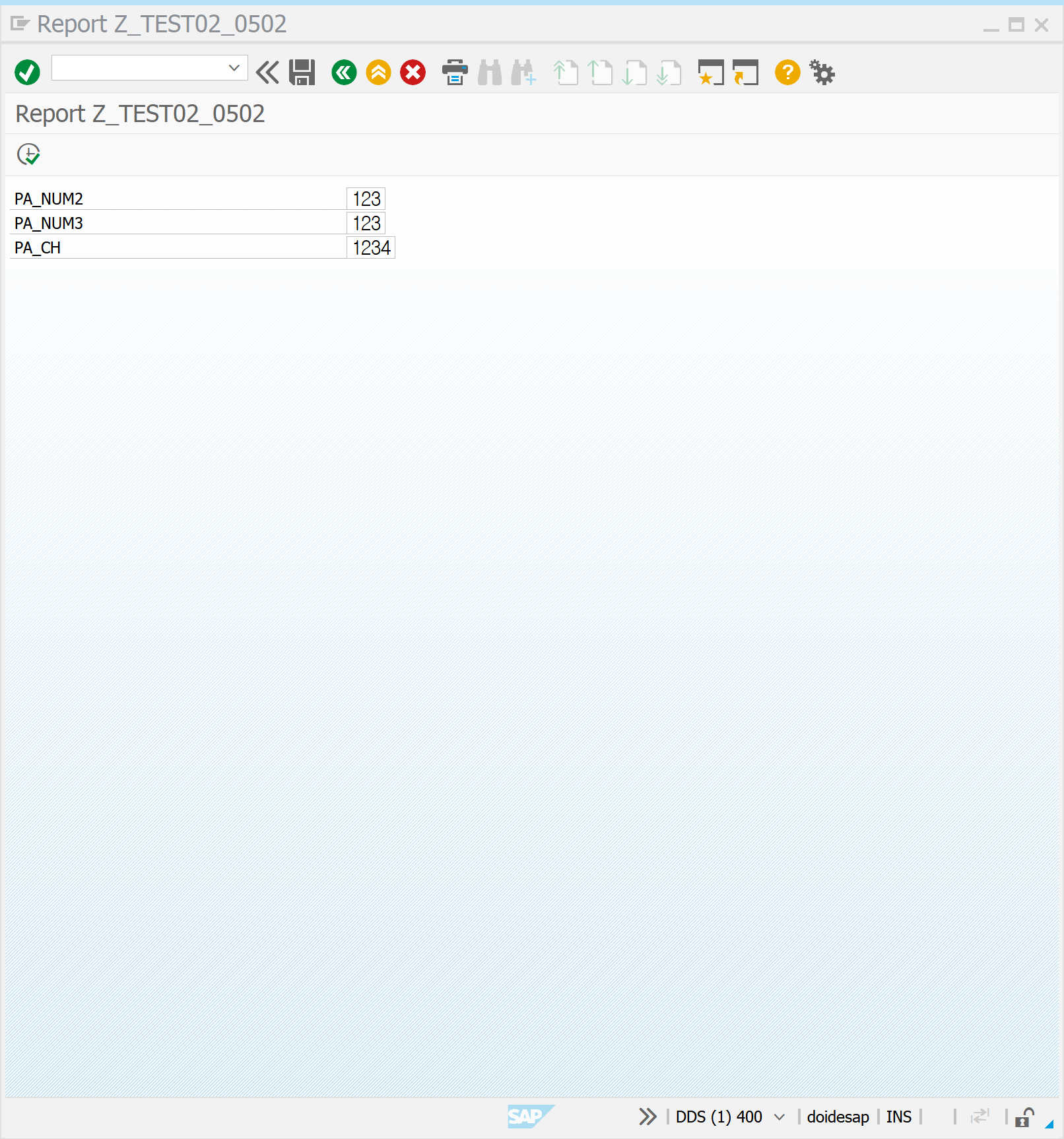
\*DECLARATION OF DATA TYPES  
\*\*LOCAL TYPE의 변수를 지정해 보자.  
TYPES TV\_C\_TYPE TYPE C LENGTH 8.  
TYPES TV\_N\_TYPE TYPE N LENGTH 5.  
TYPES TV\_P\_TYPE TYPE P LENGTH 3 DECIMALS 2.  
  
DATA GV\_CH02 TYPE TV\_C\_TYPE.  
DATA GV\_NUM02 TYPE TV\_N\_TYPE.  
DATA GV\_PAC02 TYPE TV\_P\_TYPE.  
  
\*\*GLOVAL TYPE의 변수를 지정해 보자.  
DATA GV\_CARRID TYPE S\_CARR\_ID.  
  
BREAK-POINT.

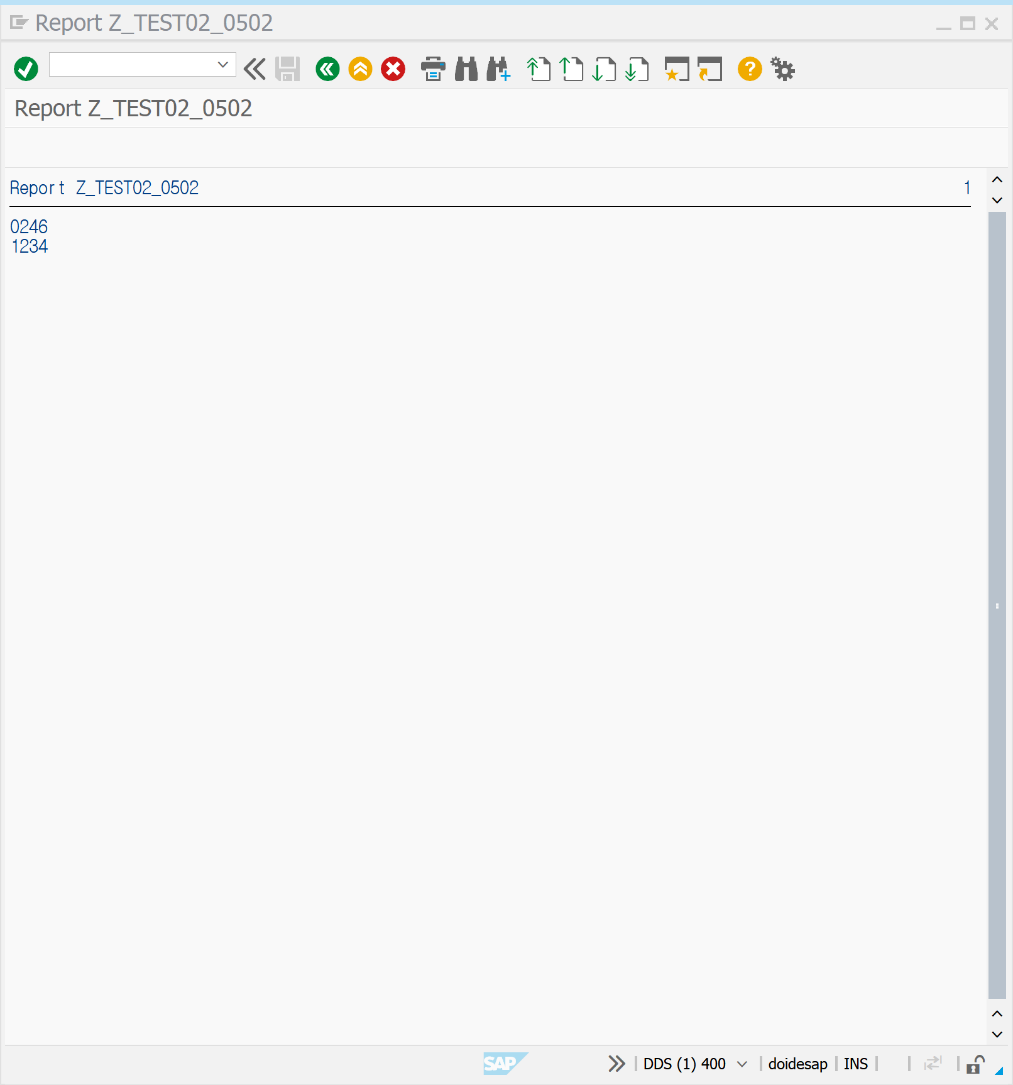
SE11 | TEST02\_01 | DICTIONARY에서 S\_CARR\_ID를 확인해보자



SE80 | TEST02\_02 | DATA TYPE의 특성에 대해 알아보자

DATA GV\_VAL TYPE C LENGTH 3.  
DATA GV\_VAL2 TYPE C.  
DATA GV\_VAL3(3) TYPE C.  
\*=DATA GV\_VAL3 TYPE C LENGTH 3.  
DATA GV\_VAL4.  
\*TYPE을 입력하지 않아도 PROGRAM이 자동으로 TYPE C 를 부여해 준다.  
DATA GV\_VAL5(5).  
  
\*PARAMETERS를 통해 변수의 값을 지정하고, 이를 더해보자.  
PARAMETERS PA\_NUM2 TYPE N LENGTH 3.  
PARAMETERS PA\_NUM3 TYPE N LENGTH 3.  
  
DATA GV\_RESULT TYPE N LENGTH 4.  
\*GV\_RESULT2 라는 변수를 생성하여 위의 PARAMETERS의 합을 지정하자.  
GV\_RESULT = PA\_NUM2 + PA\_NUM3.  
\*NUMERIC TYPE으로 문자이지만 숫자로 계산되는 것을 확인할 수 있다.  
  
PARAMETERS PA\_CH TYPE C LENGTH 4.  
  
WRITE GV\_RESULT.  
NEW-LINE.  
WRITE PA\_CH.  
  
BREAK-POINT.





SE80 | TEST03 | FIELD를 TYPE 형식으로 가져와 선언해보자

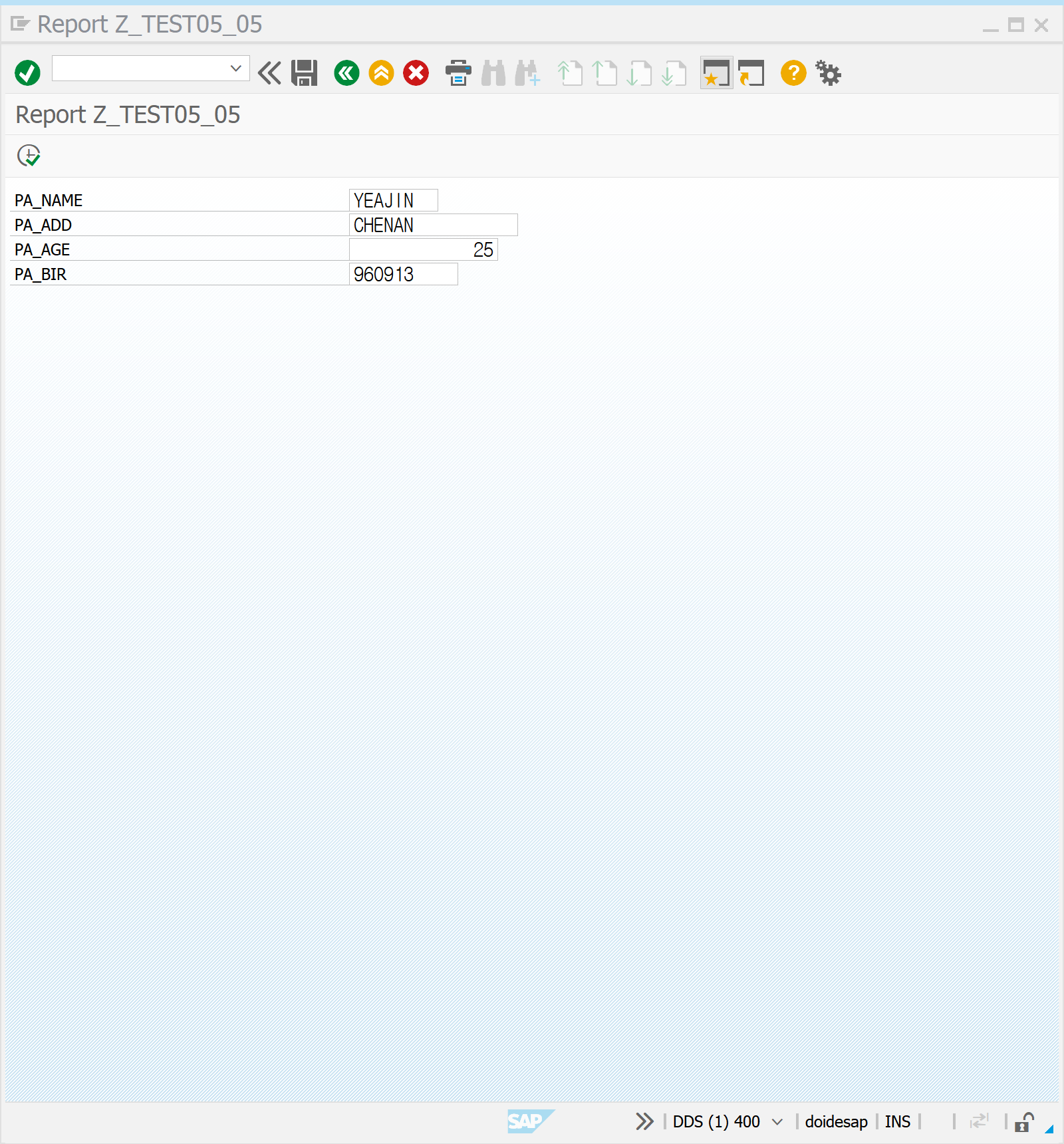
TYPES TV\_PERCENTAGE TYPE P LENGTH 3 DECIMALS 2.  
DATA : GV\_PERCENTAGE TYPE TV\_PERCENTAGE,  
\*LOCAL DATA 타입을 참조하여 GV\_PERCENTAGE 생성  
  
GV\_NUMBER1 TYPE I VALUE 17 ,  
\*표기 값을 17로 설정  
GV\_NUMBER2 LIKE GV\_NUMBER1,  
\*DATA OBJECT를 참조하여 생성한다.  
\*\*LIKE 는  앞에서 생성한 동일한 TYPE의 변수를 선언할 때 사용한다.  
  
GV\_CITY TYPE C LENGTH 15,  
GV\_CARRID TYPE S\_CARR\_ID,  
\*위에서 선언한 것이 아닌 GLOBAL DATA TYPE으로  
\*DICTIONARY에 지정되어 있는 것을 참조하여 생성한다.  
\*해당 DATA TYPE은 더블클릭 하면 확인 할 수 있다.  
  
GV\_CONNID TYPE S\_CONN\_ID.  
\*4자리 NUMERIC 형태로 이루어져 있다.

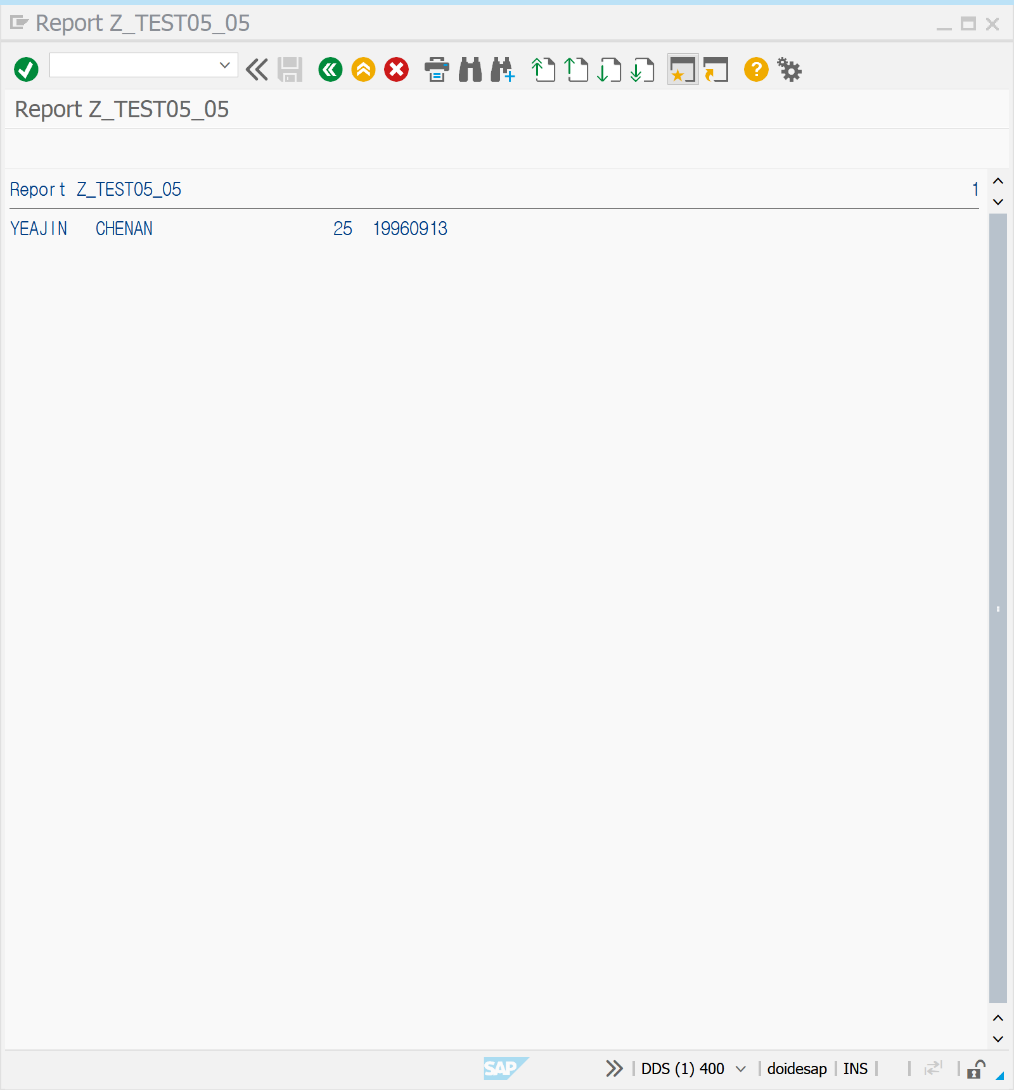
SE80 | TEST04 | STRUCTURE를 선언해보자

\*FIELD 선언  
DATA : GV\_CARRID TYPE S\_CARR\_ID,  
             GV\_CONNID TYPE S\_CONN\_ID.  
  
\*STRUCTURE 선언  
DATA: BEGIN OF GV\_STR,  
\*BEGIN OF STRUCTURE 명 을 선언한다.  
  CARRID TYPE S\_CARR\_ID,  
\*STRUCTURE 내의 CARRID 변수의 TYPE을 지정한다.  
  CONNID TYPE S\_CONN\_ID,  
  END OF GV\_STR.  
  
  
DATA: BEGIN OF GV\_STR1,  
            GV\_CH TYPE C LENGTH 8,  
            GV\_NUM TYPE N LENGTH 4,  
            GV\_INT TYPE I,  
            GV\_DATE TYPE D,  
   END OF GV\_STR1.  
  
  
BREAK-POINT.

SE80 | TEST05 | STRUCTURE를 선언해보자

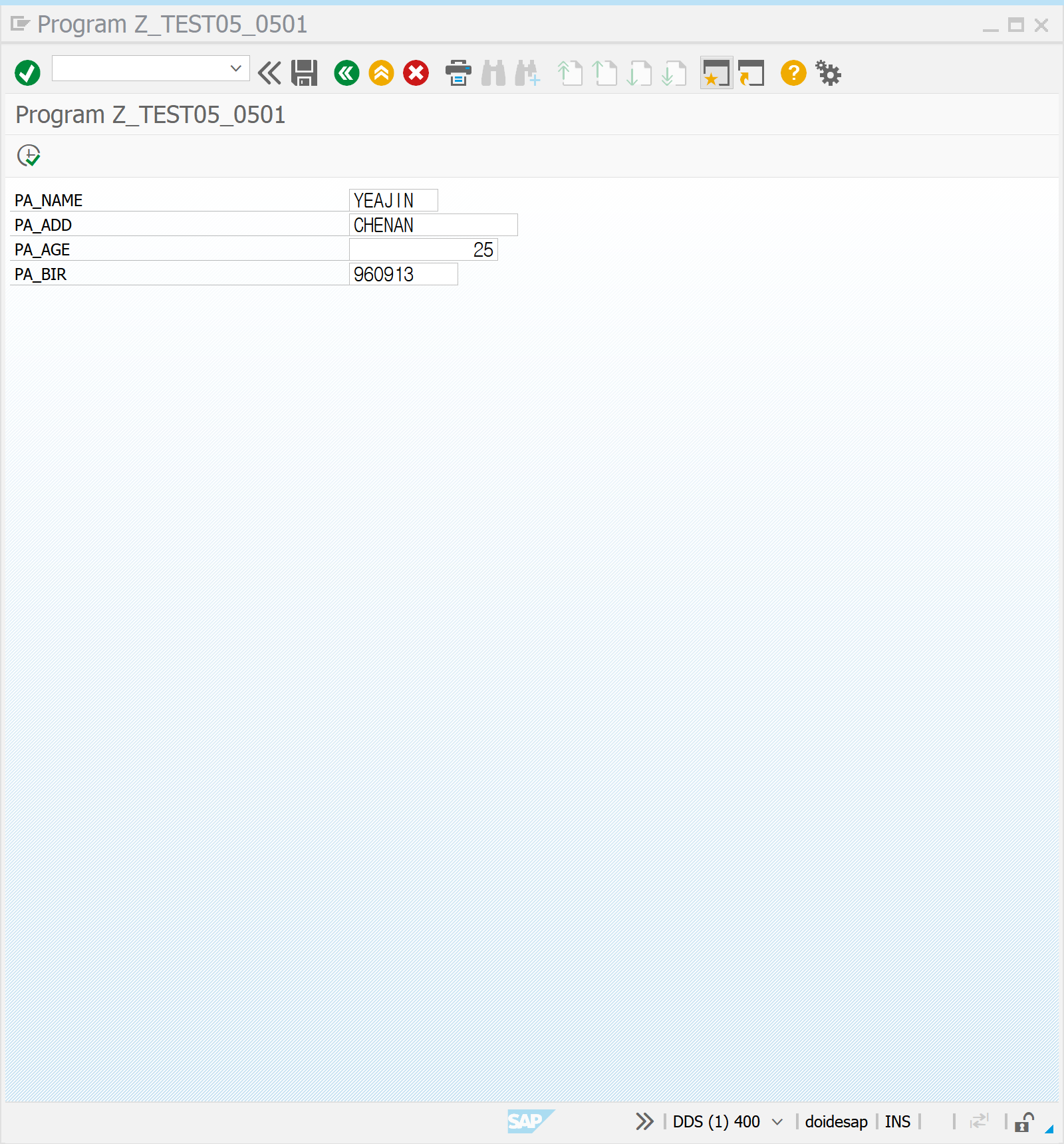
\*STRUCTURE을 선언해보자.   
DATA: BEGIN OF GV\_STR,  
      NAME TYPE C LENGTH 8,  
      ADDRESS TYPE C LENGTH 16,  
      AGE TYPE I,  
      BIRTH TYPE D,  
   END OF GV\_STR.  
     
\*PARAMETERS를 통해 변수의 값을 지정하자.  
 PARAMETERS: PA\_NAME TYPE C LENGTH 8,  
                            PA\_ADD TYPE C LENGTH 16,  
                            PA\_AGE TYPE I,  
                            PA\_BIR TYPE D.  
  
\*CTRL+SPACEBAR를 통해 해당 STRUCTURE에 있는 FIELD를 불러온다.  
\*\*이를 PARAAMETERS를 통해 지정한 변수와 매칭시켜준다.   
  GV\_STR-NAME = PA\_NAME.  
  GV\_STR-ADDRESS = PA\_ADD.  
  GV\_STR-AGE = PA\_AGE.  
  GV\_STR-BIRTH = PA\_BIR.  
  
 WRITE:/ GV\_STR-NAME, GV\_STR-ADDRESS, GV\_STR-AGE, GV\_STR-BIRTH.

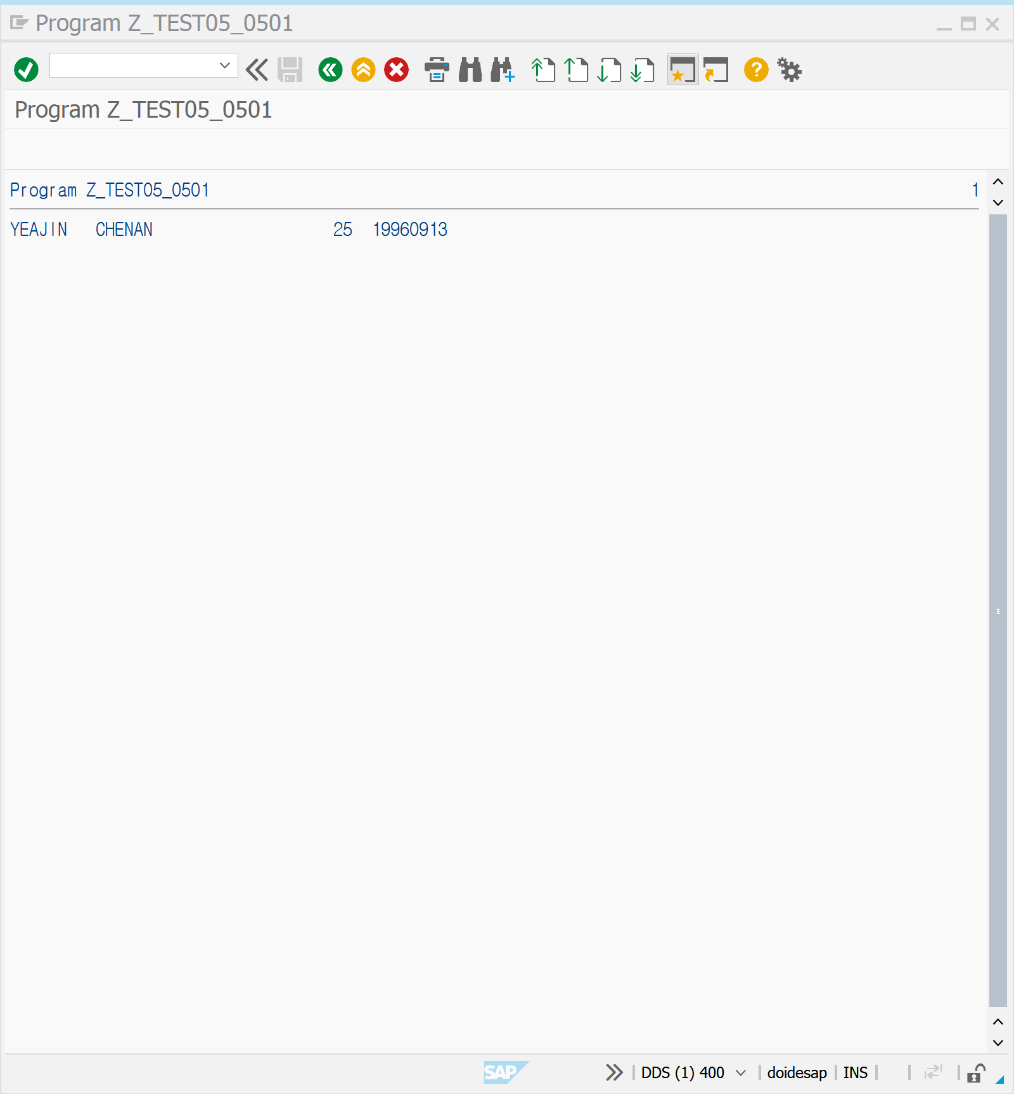




SE80 | TEST0501 | STRUCTURE를 TYPE 형식으로 가져와 선언해보자

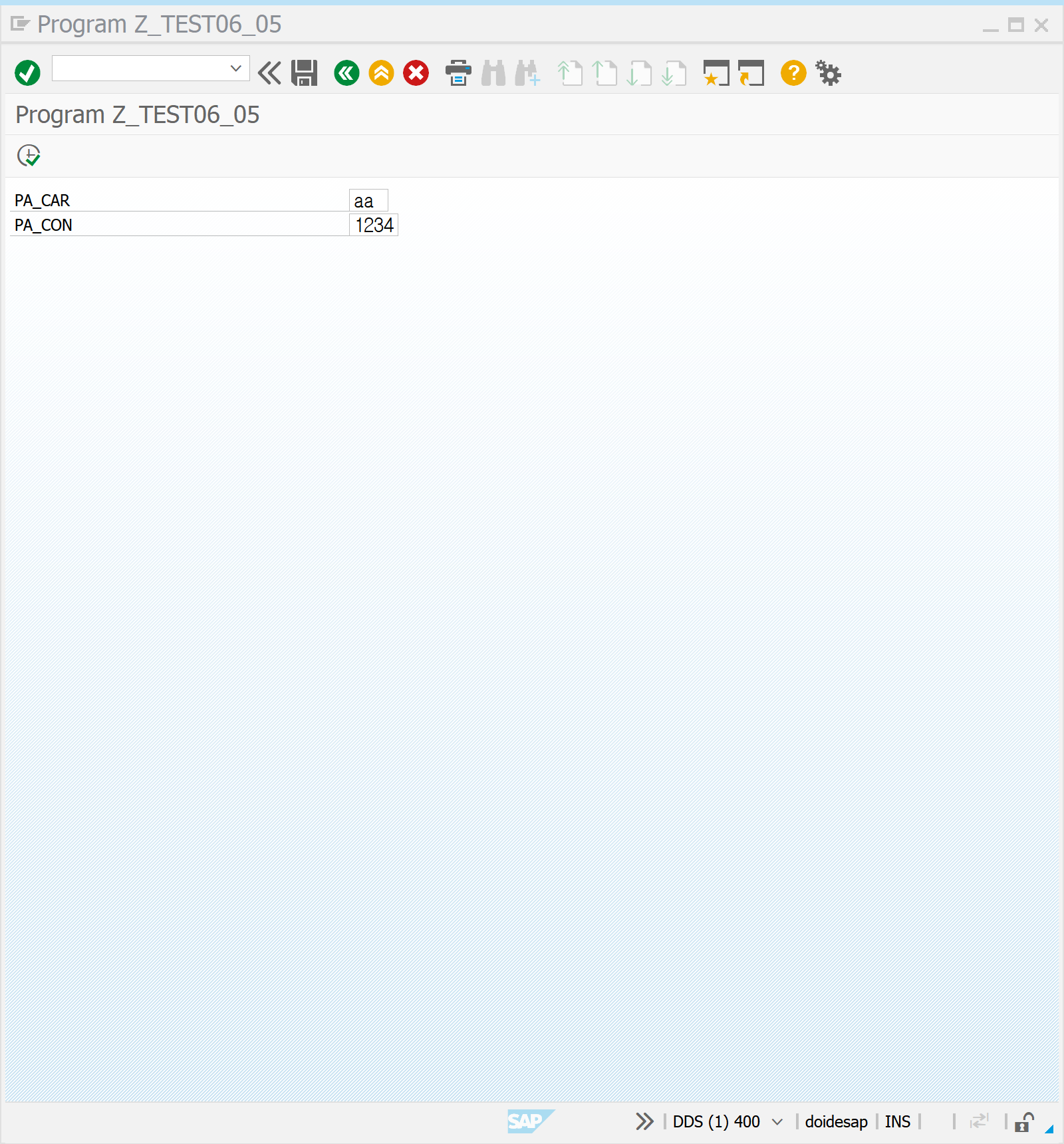
\*STRUCTURE을 TYPE 형태로 가져와서 선언해보자  
TYPES: BEGIN OF TY\_STR,  
      NAME TYPE C LENGTH 8,  
      ADDR TYPE C LENGTH 16,  
      AGE TYPE I,  
      BIRTH TYPE D,  
  END OF TY\_STR.  
  
DATA GV\_STR TYPE TY\_STR.  
\*  
  
\*PARAMETERS를 통해 변수의 값을 지정하자.  
 PARAMETERS: PA\_NAME TYPE C LENGTH 8,  
                            PA\_ADD TYPE C LENGTH 16,  
                            PA\_AGE TYPE I,  
                            PA\_BIR TYPE D.  
  
\*CTRL+SPACEBAR를 통해 해당 STRUCTURE에 있는 FIELD를 불러온다.  
\*\*이를 PARAAMETERS를 통해 지정한 변수와 매칭시켜준다.  
GV\_STR-NAME =  PA\_NAME .  
GV\_STR-ADDR = PA\_ADD.  
GV\_STR-AGE = PA\_AGE.  
GV\_STR-BIRTH = PA\_BIR.  
  
WRITE:/ GV\_STR-NAME, GV\_STR-ADDR, GV\_STR-AGE, GV\_STR-BIRTH.

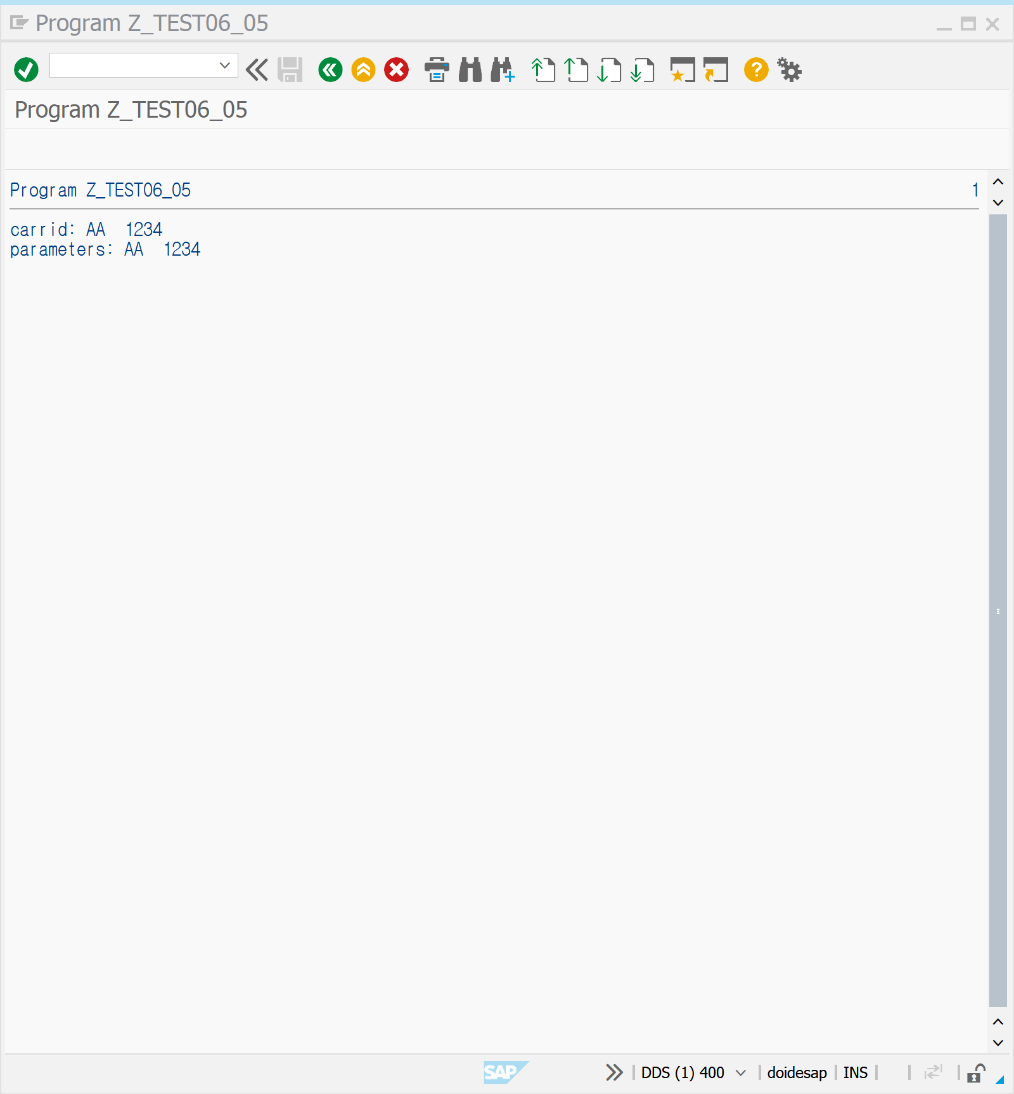




SE80 | TEST06 | 미리 지정된 변수를 사용하는 STRUCTURE를 선언해보자

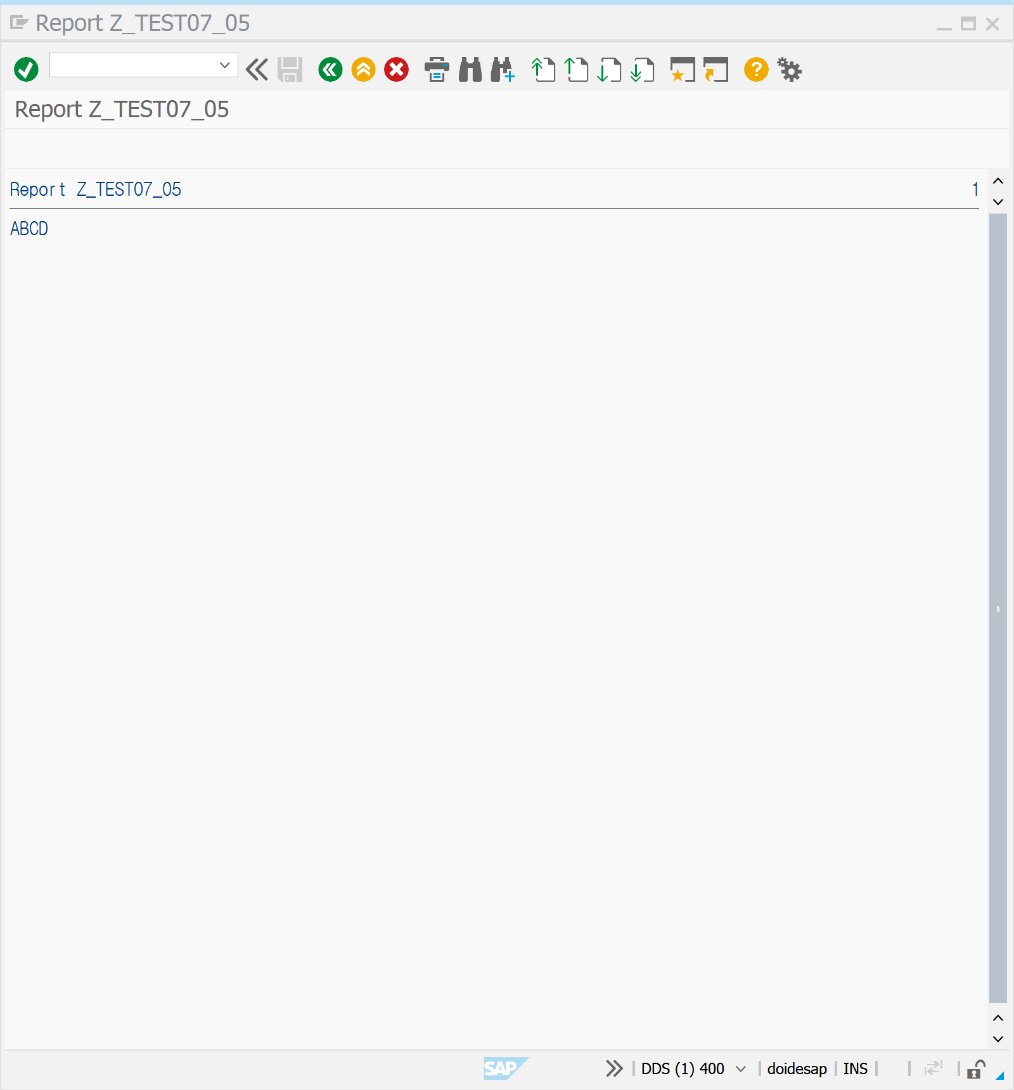
DATA GS\_GLOBAL TYPE SCARR.  
  
DATA GS\_FLIGHT TYPE ZBC400\_S\_FLIGHT.  
PARAMETERS PA\_CAR TYPE S\_CARR\_ID.  
PARAMETERS PA\_CON TYPE S\_CONN\_ID.  
  
GS\_FLIGHT-CARRID = PA\_CAR.  
GS\_FLIGHT-CONNID = PA\_CON.  
  
WRITE: /'carrid:', GS\_FLIGHT-CARRID, GS\_FLIGHT-CONNID.  
WRITE: /'parameters:', PA\_CAR,PA\_CON.  
BREAK-POINT.





SE80 | TEST07 | CONSTANTS 자주 사용하는 값을 상수로 선언해보자

\*CONSTANTS  자주 사용하는 값을 상수로 선언하자   
\*\*상수로 선언하면 프로그램 내에서는 변경할 수 없다.   
CONSTANTS GC\_MYCONST TYPE C VALUE 'c'.  
  
\*GC\_MYCONST ='a'.  
\*값을 수정할 수 없다는 오류 발생  
\*WRITE GC\_MYCONST.  
  
WRITE: TEXT-001.  
\*더블 클릭 후 001에 데이터 값을 입력한다.  
\*이동 화면에서 TRANSLATE를 사용하면 다른 키값의 언어를 작성 할 수 있다.  
  
BREAK-POINT.



SE80 | TEST0701 | DICTIONARY의 TABLE을 참조하여 TABLE 을 생성해보자

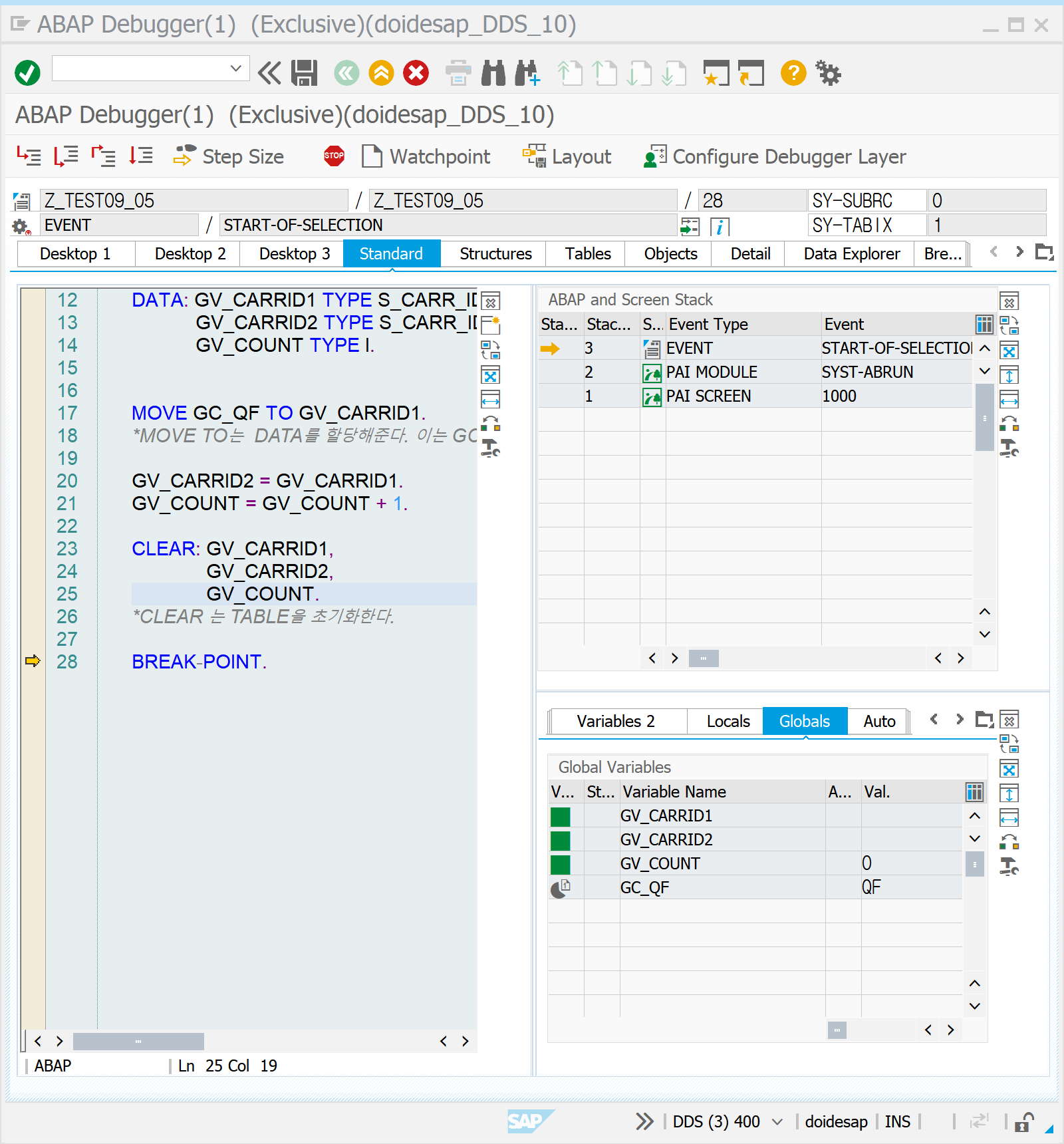
\*TABLE 생성  
DATA: GT\_TABLE TYPE ZBC400\_T\_FLIGHTS.  
\*DICTIONARY에 있는 ZBC400\_T\_FLIGHTS을 참조하여 GT\_TABLE이라는 TYPE를 생성한다.  
DATA: GT\_TABLE2 TYPE TABLE OF ZBC400\_S\_FLIGHT.  
\*DICTIONARY에 있는 ZBC400\_T\_FLIGHTS을 참조하여 GT\_TABLE2이라는 TABLE 생성한다.  
DATA: GS\_STR2 TYPE LINE OF ZBC400\_T\_FLIGHTS.  
\*DICTIONARY에 있는 ZBC400\_T\_FLIGHTS을 참조하여 GS\_STR2이라는 STRUCTURE를 생성한다.

SE80 | TEST08 | DICTIONARY의 TABLE을 참조하여 TABLE 을 생성해보자

\*FIELD 3개로 STRUCTURE 생성 및 STRUCTURE로 TABLE 생성하기  
  
\*FIELD  
DATA: GV\_A TYPE N LENGTH 4,  
            GV\_B TYPE C LENGTH 6,  
            GV\_C TYPE I.  
  
  
\*STR  
DATA: BEGIN OF GS\_STR,  
      GV\_A TYPE N LENGTH 4,  
      GV\_B TYPE C LENGTH 6,  
      GV\_C TYPE I,  
   END OF GS\_STR.  
  
  
\*TABLE  
DATA: GT\_TABLE LIKE TABLE OF GS\_STR.  
\*TYPE으로 정의한 LOCAL TYPE이 아니기 때문에 사용할 수 없다. LIKE 로 변경하자  
\*이는 STRUCTURE 참조하여 TABLE로 구성한다.  
  
\*LOCAL TYPE 지정  
TYPES: BEGIN OF TY\_STR,  
  GV\_A(4) TYPE N,  
  GV\_B(6) TYPE C,  
  GV\_C TYPE I,  
END OF TY\_STR.  
  
DATA GT\_TABLE2 TYPE TABLE OF TY\_STR.  
\*LOCAL 이나 GLOBAL로 지정된 TABLE을 참조해 생성한다.  
DATA GT\_TABLE3 LIKE GT\_TABLE2.  
\*생성한 TABLE을 참조하여 GT\_TABLE3을 생성한다.  
BREAK-POINT.

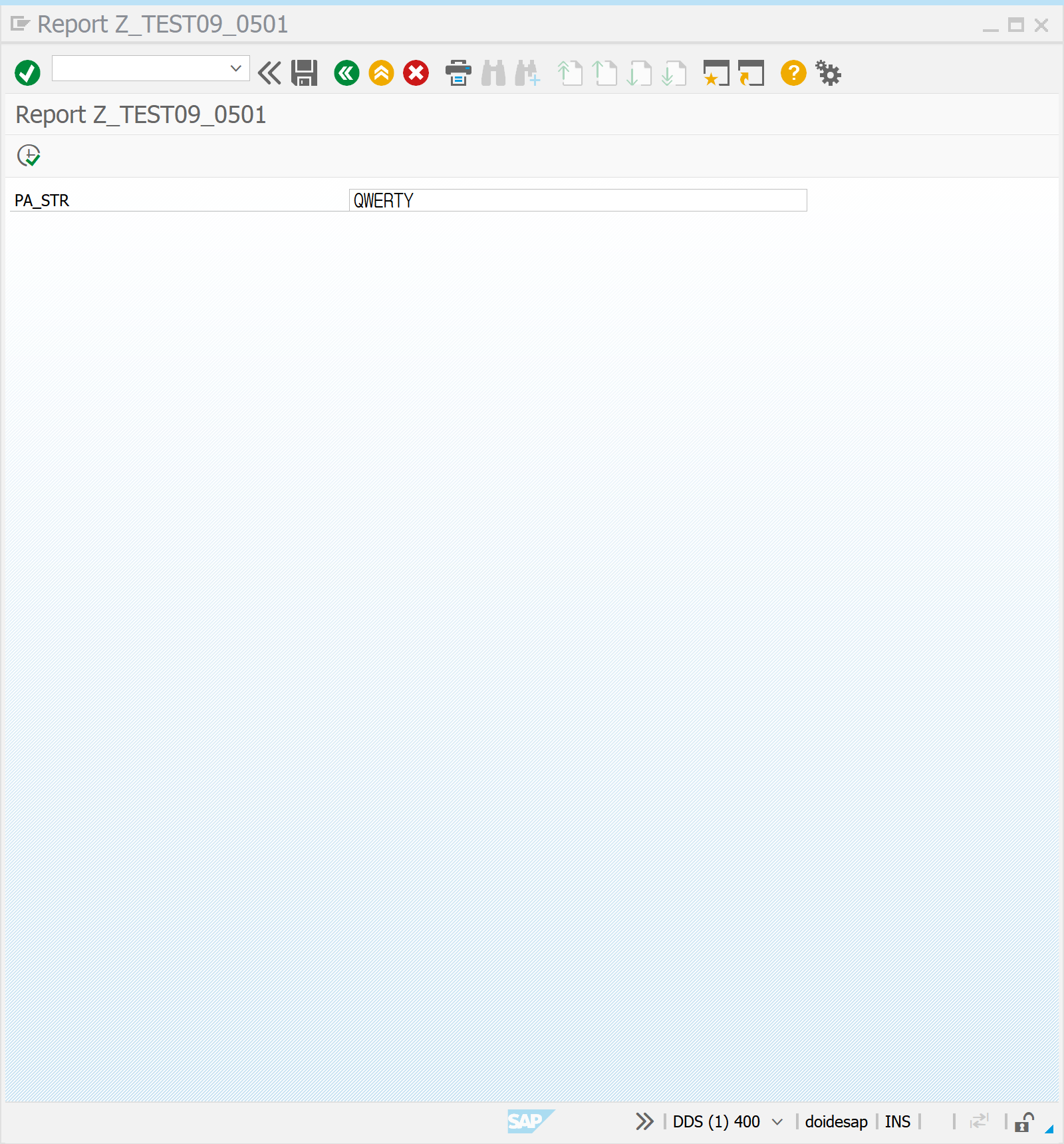
SE80 | TEST09 | MOVE TO 와 CLEAR

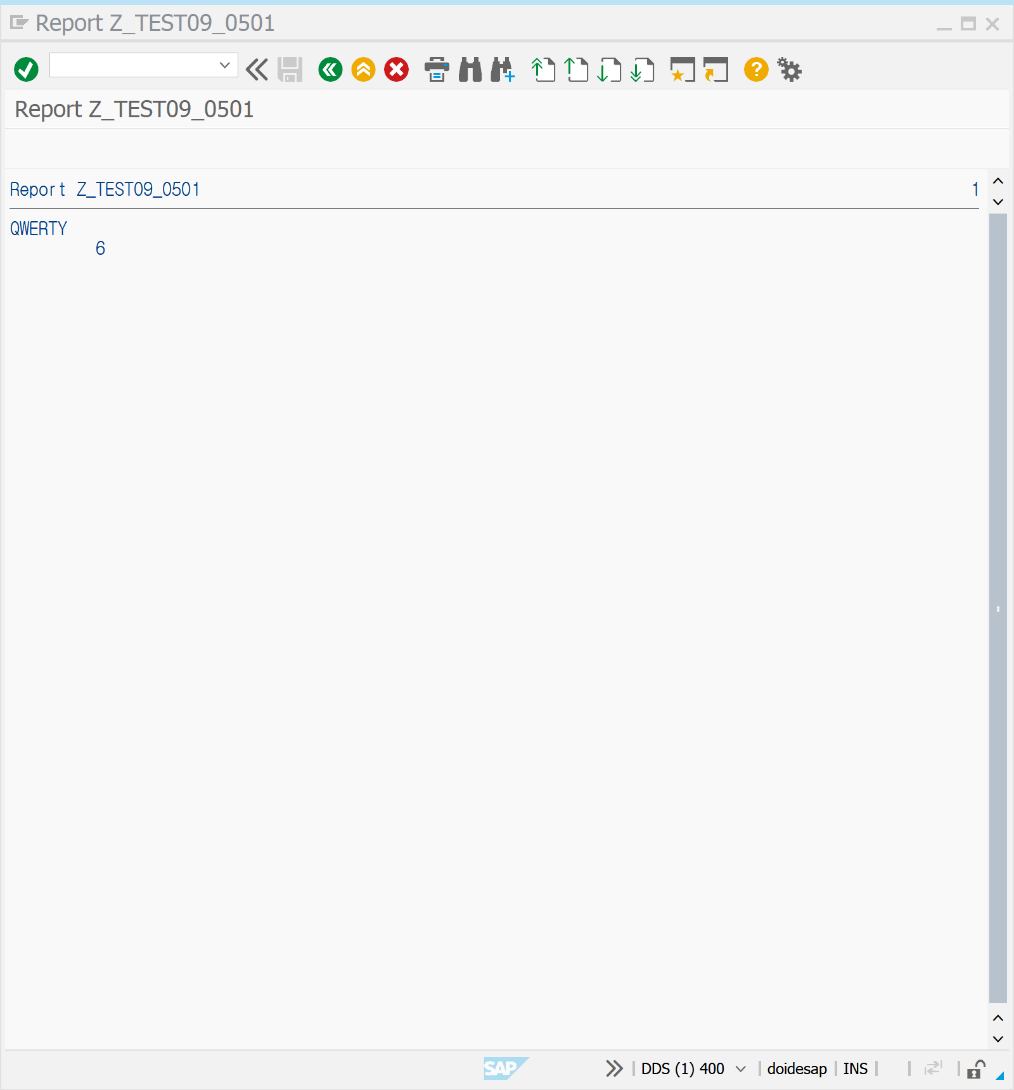
\*CONSTANTS 상수로 한 번 지정하면 변경할 수 없다.  
CONSTANTS GC\_QF TYPE S\_CARR\_ID VALUE 'QF'.  
  
\*FILED  
DATA: GV\_CARRID1 TYPE S\_CARR\_ID,  
            GV\_CARRID2 TYPE S\_CARR\_ID VALUE 'LH',  
            GV\_COUNT TYPE I.  
  
  
MOVE GC\_QF TO GV\_CARRID1.  
\*MOVE TO는  DATA를 할당해준다. 이는 GC\_QF=GV\_CARRID1 를 의미한다.  
  
GV\_CARRID2 = GV\_CARRID1.  
GV\_COUNT = GV\_COUNT + 1.  
  
CLEAR: GV\_CARRID1,  
              GV\_CARRID2,  
              GV\_COUNT.  
\*CLEAR 는 TABLE을 초기화한다.  
  
BREAK-POINT.



SE80 | TEST0901 | PARAMETERS을 활용해 글자 길이를 알아보자

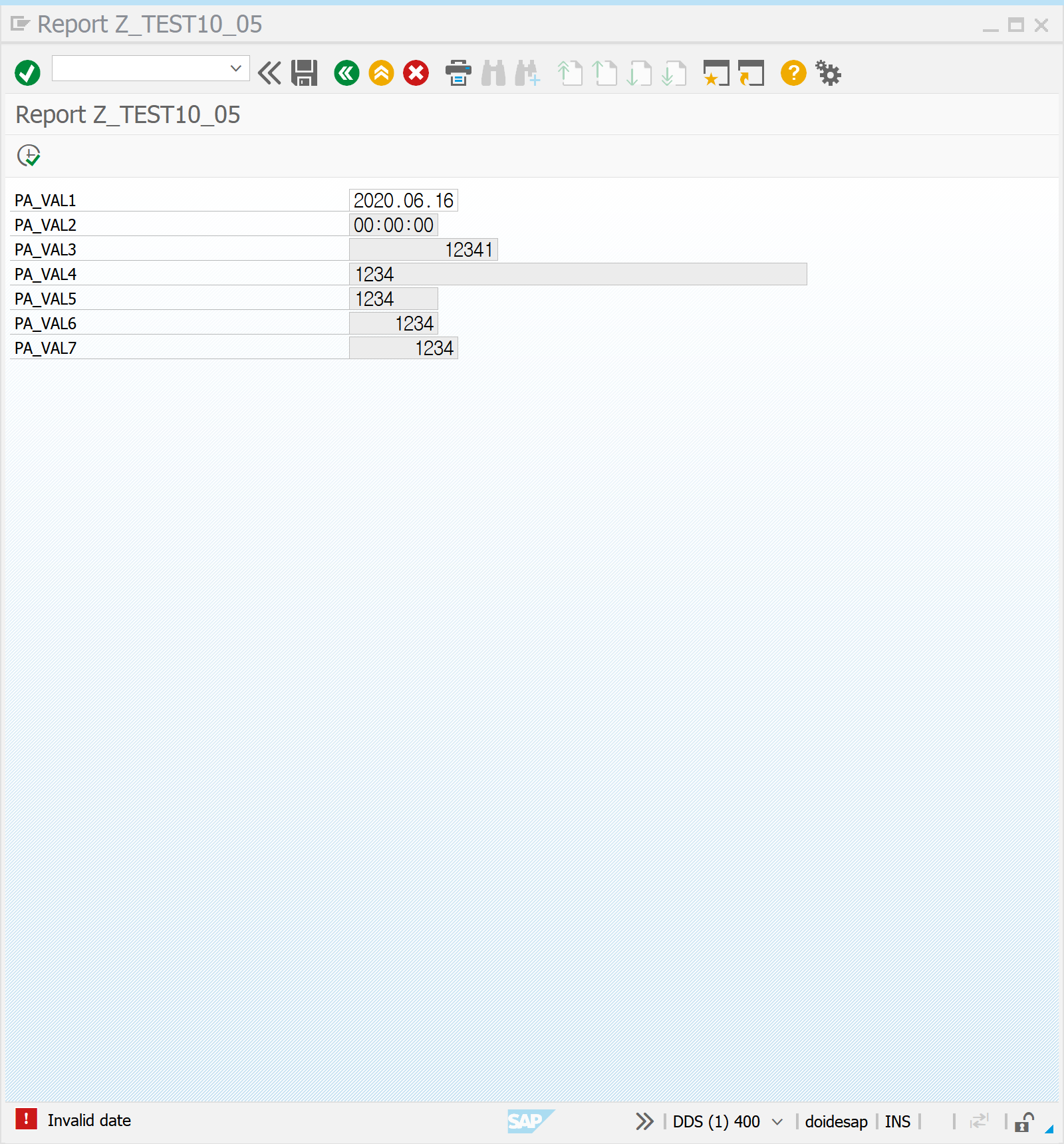
\*PARAMETERS을 이용해  글자의 길이를 알아보자.  
PARAMETERS PA\_STR TYPE STRING.  
  
DATA GV\_STRING TYPE STRING.  
DATA GV\_LENGTH TYPE I.  
\*글자의 길이를 알아볼 LENGTH 변수를 지정한다.  
GV\_LENGTH = STRLEN( PA\_STR ).  
\*사용자가 PARAMETERS를 이용해 입력한 PA\_STR의 길이를 STRLEN을 통해 도출한다.  
  
WRITE:/ PA\_STR.  
WRITE:/ GV\_LENGTH.  
  
BREAK-POINT.





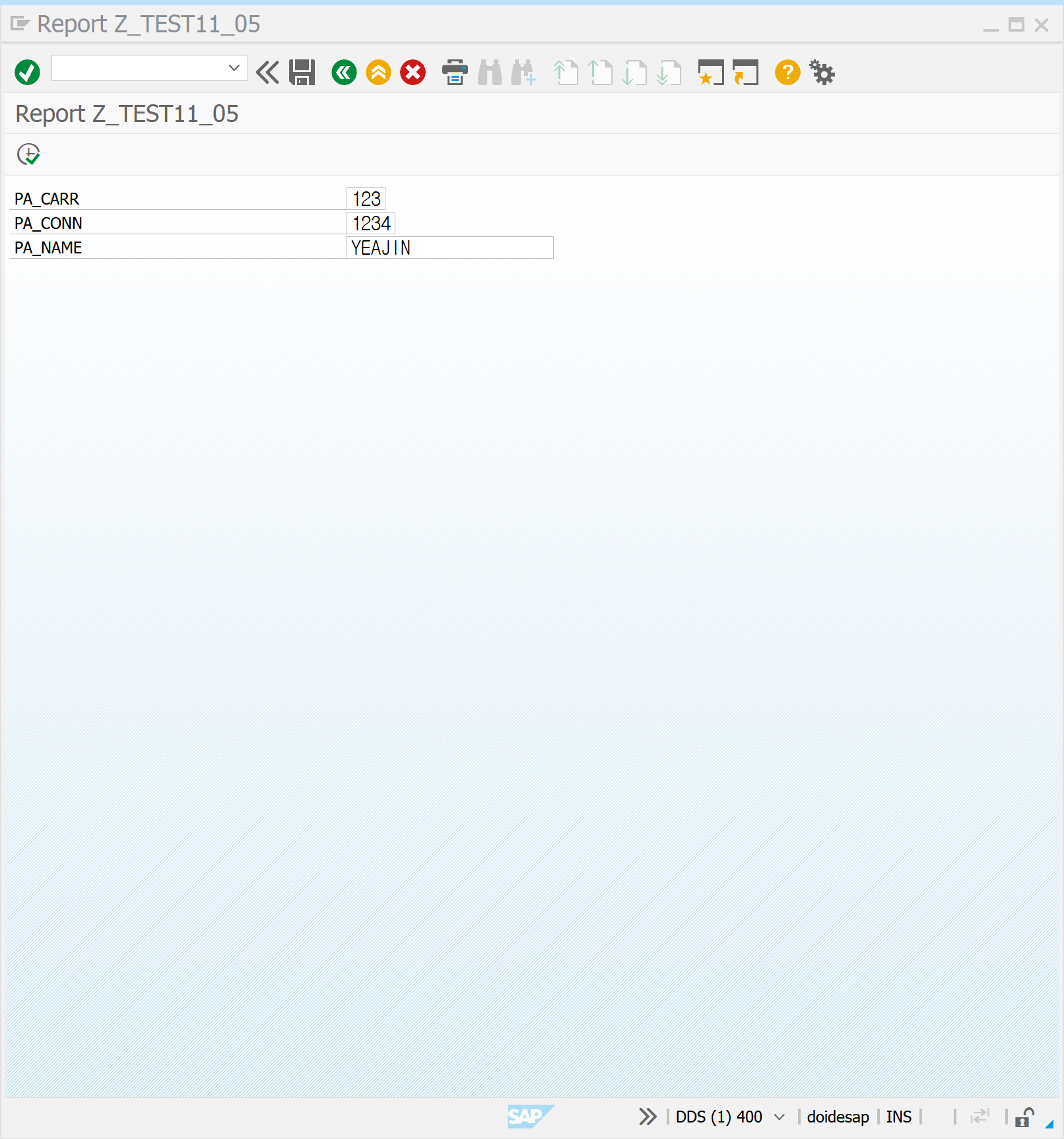
SE80 | TEST10 | PARAMETERS 에 대해 알아보자.

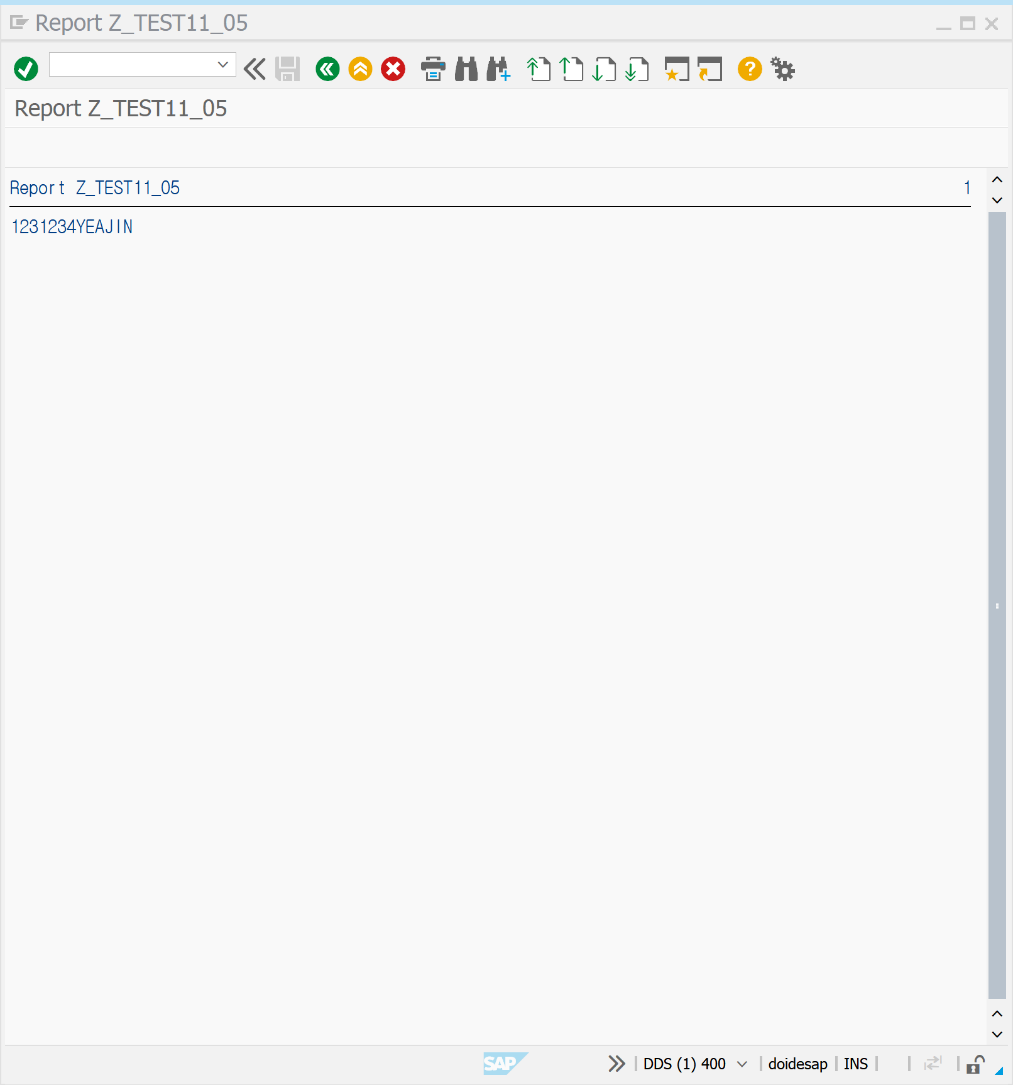
\*PARAMETERS를 통해 사용자 화면에서 입력 값을 받아온다.  
\*각 DATA TYPE에 맞는 값들만 입력할 수 있다.  
\*TYPE N 은 NUMERIC으로 숫자를 문자로 취급하며, 문자는 입력되지 않는다.  
PARAMETERS: PA\_VAL1 TYPE D,  
                           PA\_VAL2 TYPE T,  
                           PA\_VAL3 TYPE I,  
                           PA\_VAL4 TYPE STRING,  
                           PA\_VAL5 TYPE  C LENGTH 8,  
                           PA\_VAL6 TYPE  N LENGTH 8,  
                           PA\_VAL7 TYPE  P LENGTH 4 DECIMALS 2.  
  
  
\*FIELD 지정 후   
\*각 PARAMETERS 값을 매칭시켜주자



SE80 | TEST11 | PARAMETERS 사용해 FIELD와 STRUCTURE을 생성하자

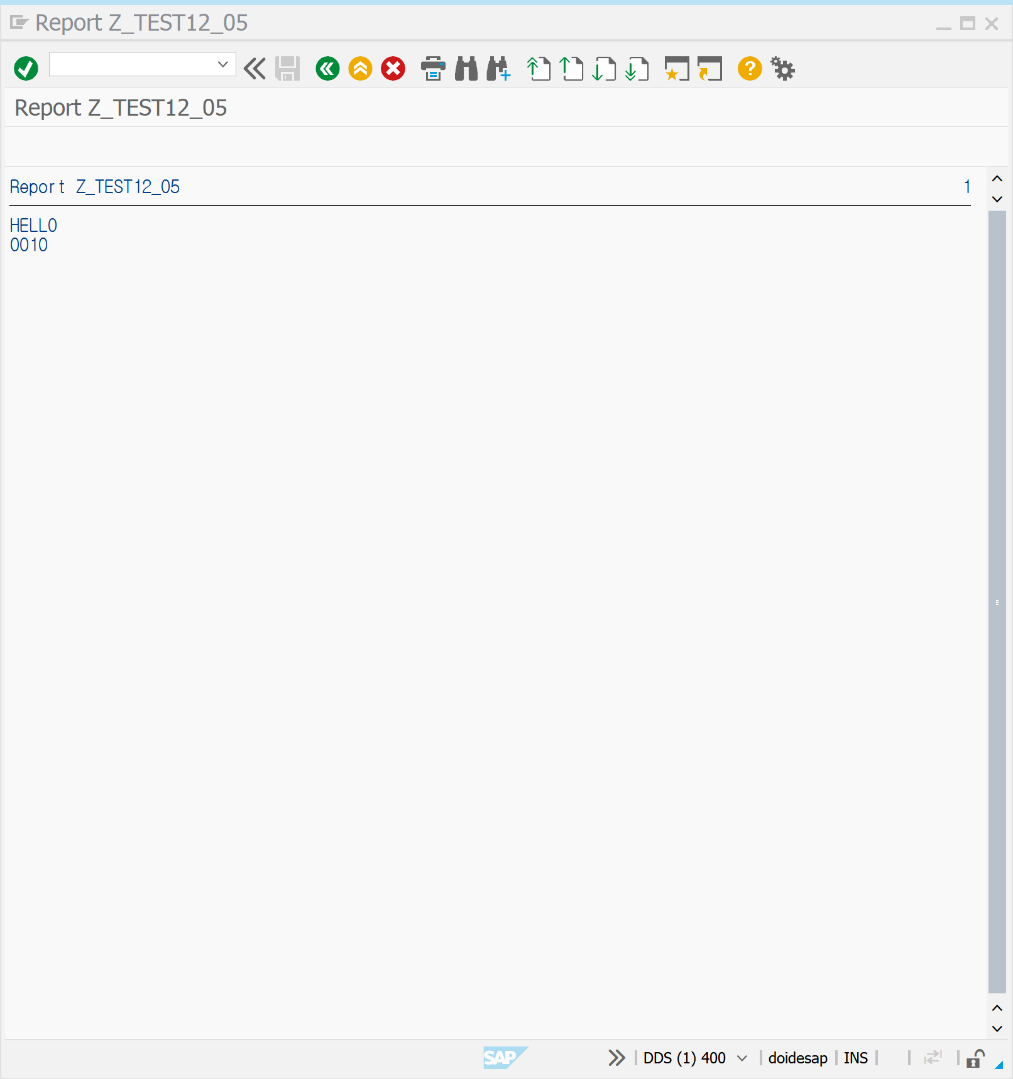
\*PARAMETERS  
\*PARAMETERS를 통해 사용자 화면에서 입력 값을 받아온다.  
\*각 DATA TYPE에 맞는 값들만 입력할 수 있다.  
PARAMETERS: PA\_CARR TYPE C LENGTH 3,  
                          PA\_CONN TYPE C LENGTH 4,  
                          PA\_NAME TYPE C LENGTH 20.  
  
\*FIELD  
DATA GV\_CARRID TYPE C LENGTH 3.  
DATA GV\_CONNID TYPE C LENGTH 4.  
DATA GV\_NAME TYPE C LENGTH 20.  
  
\*STRUCTURE  
\*BEGIN OF - END OF 의 구조를 가진다.  
DATA: BEGIN OF GS\_STR,  
    CARRID TYPE C LENGTH 3,  
    CONNID TYPE C LENGTH 4,  
    NAME TYPE C LENGTH 20,  
END OF GS\_STR.  
  
\*STRUCTURE-FIELD 로 STRUCTURE 내부의 FIELD 값을 가져온다.  
\*해당 STRUCTURE의 FILED 값을 PARAMETER로 받아 온 값과 매칭 시켜준다.  
\*CTRL+SPACEBAR 로 해당 STRUCTURE의 FIELD 값을 가져올 수 있다.  
GS\_STR-CARRID = PA\_CARR.  
GS\_STR-CONNID = PA\_CONN.  
GS\_STR-NAME = PA\_NAME.  
  
WRITE: GS\_STR.





SE80 | TEST12 | 만약 사용자가 내부에서 변수의 값을 지정하고 싶은 경우

\*만약 PARAMETERS을 통해 값을 받아오지 않고, 사용자가 내부에서 지정하고 싶은 경우  
\*FIELD  
DATA: GV\_VAL8 TYPE C LENGTH 8,  
      GV\_VAL9 TYPE C LENGTH 8.  
  
\*DATA 값 을 지정하자.  
GV\_VAL8 ='HELLO'.  
GV\_VAL9 ='0010'.  
  
WRITE: GV\_VAL8, / GV\_VAL9.



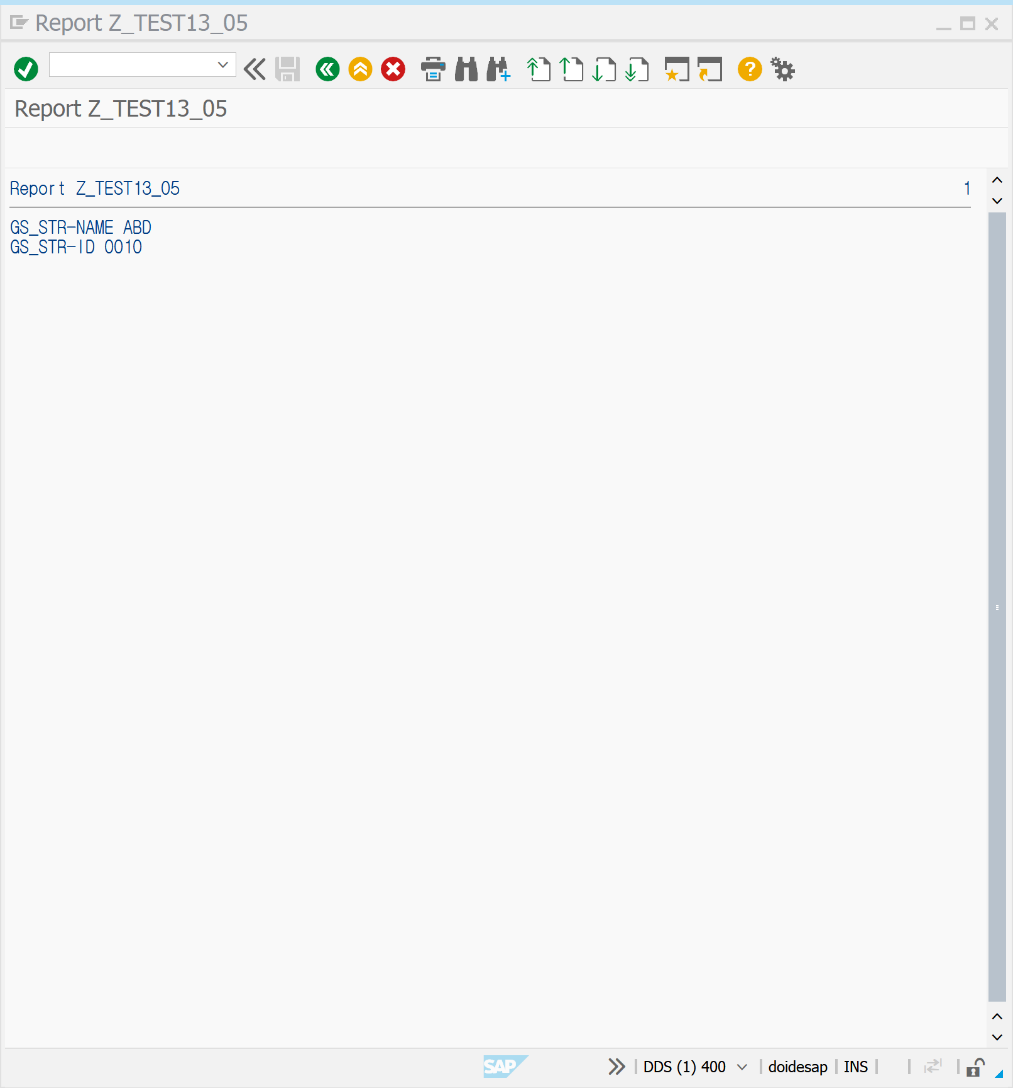
SE80 | TEST13 | 3가지 방법으로 TABLE을 생성해보자.

\*OCCURS 0 은 필드를 2개 가진 TABLE을 생성한다.  
DATA: BEGIN OF GT\_TAB OCCURS 0,  
      NAME TYPE C LENGTH 8,  
      ID TYPE C LENGTH 4,  
      END OF GT\_TAB.

\*TYPES는 TABLE의 형태를 구성한다.  
TYPES: BEGIN OF TY\_STR,  
       NAME TYPE C LENGTH 8,  
       ID TYPE C LENGTH 4,  
       END OF TY\_STR.  
  
\*TYPE 형태를 갖춘 TY\_STR1을 생성한다.  
DATA GS\_STR TYPE TY\_STR.

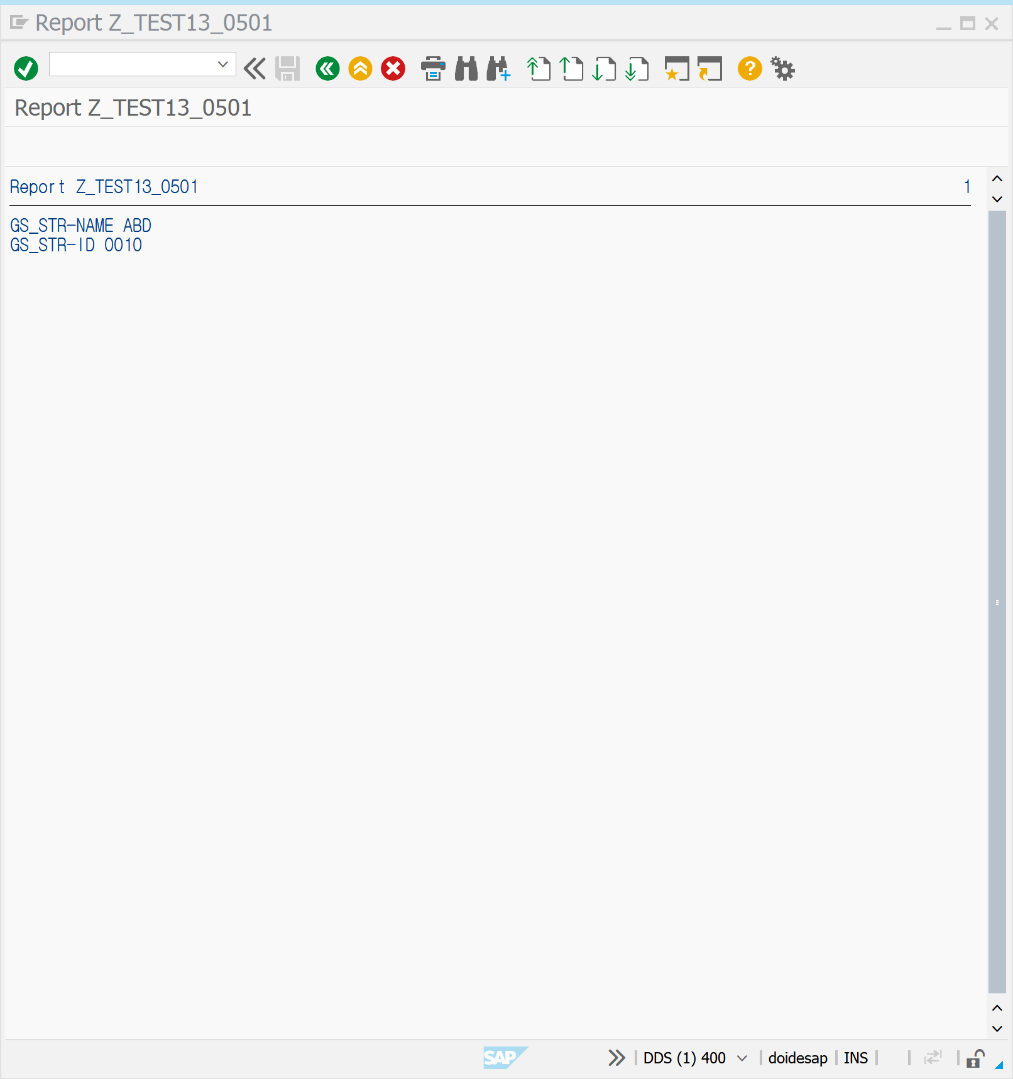
\*TABLE의 DATA를 변경하고 싶을 때 다음과 같이 지정한다.  
GS\_STR-NAME ='ABD'.  
GS\_STR-ID ='0010'.

WRITE:/ 'GS\_STR-NAME', GS\_STR-NAME, / 'GS\_STR-ID', GS\_STR-ID.  
  
DATA GT\_TABLE TYPE TABLE OF TY\_STR.  
\*STRUCTURE 를 참조하여 TABLE을 생성한다.  
  
DATA GT\_TABLE1 LIKE TABLE OF GT\_TABLE.  
\*미리 생성된 DATA OBJECT를 가지고 다른 OBJECT를 생성한다.

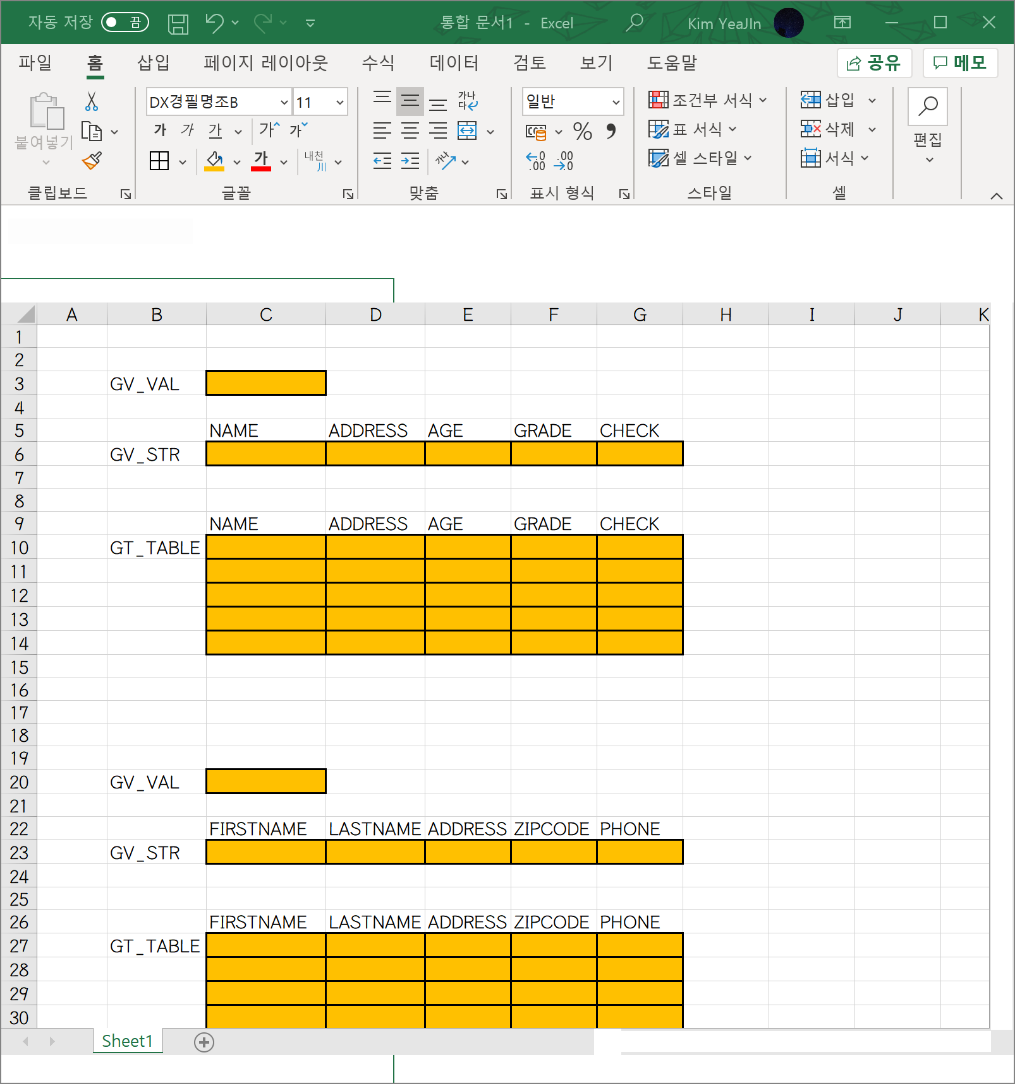


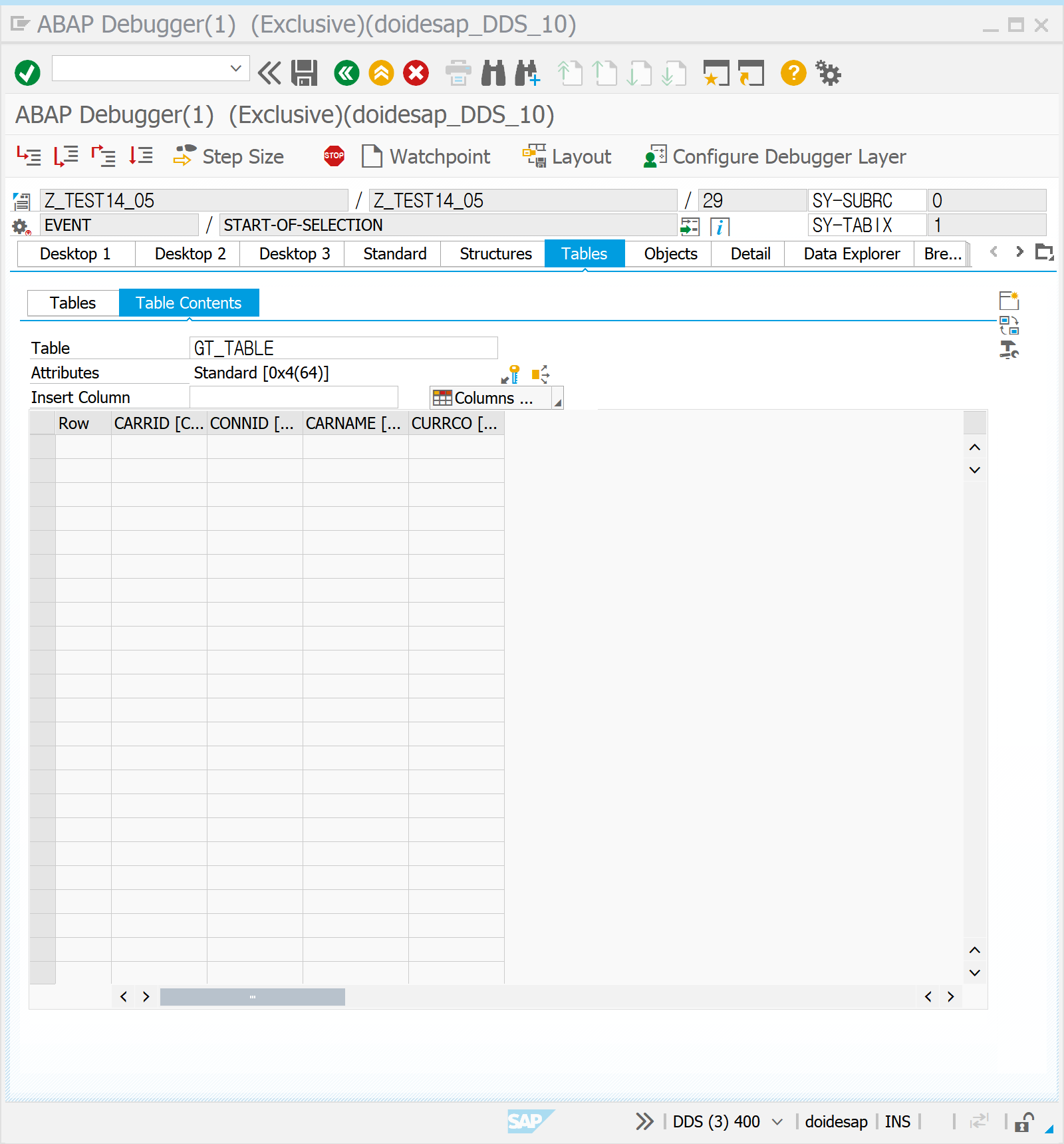
SE80 | TEST1301 | TABLE에 DATA 값을 입력하자.

\*TYPE 형태로 TABLE을 생성하고 값을 입력해보자  
TYPES: BEGIN OF TY\_STR,  
         NAME TYPE C LENGTH 8,  
         ID   TYPE C LENGTH 4,  
       END OF TY\_STR.  
  
DATA GT\_TABLE TYPE TABLE OF TY\_STR.  
DATA GS\_STR TYPE TY\_STR.  
  
GS\_STR-NAME = 'ABD'.  
GS\_STR-ID = '0010'.  
  
APPEND GS\_STR TO GT\_TABLE.  
\*APPEND는 INDEX만 사용할 수 있다.  
  
WRITE:/ 'GS\_STR-NAME', GS\_STR-NAME, / 'GS\_STR-ID', GS\_STR-ID.  
  
BREAK-POINT.



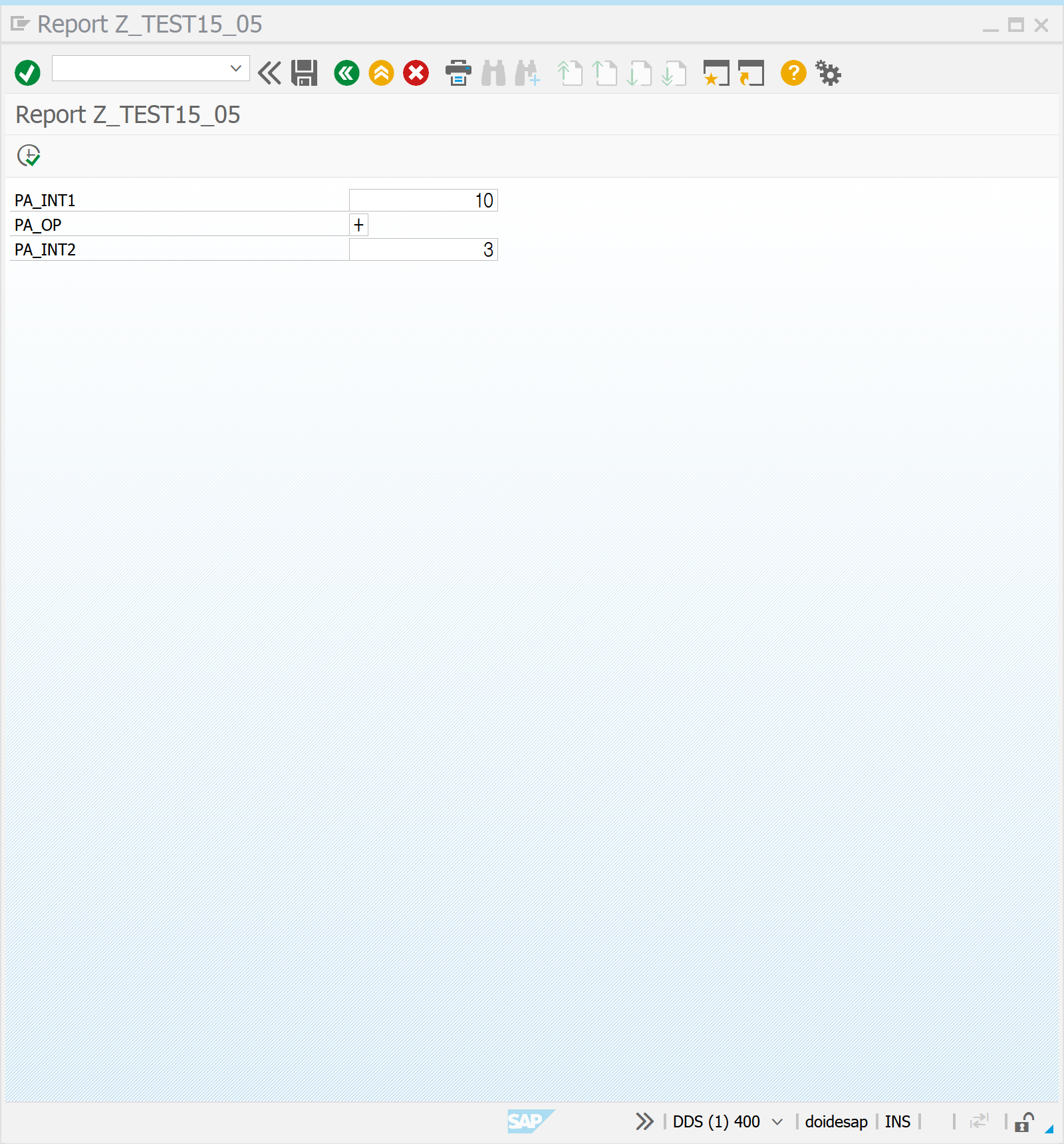
SE80 | TEST14 | FIELD가 5개인 TABLE을 만들어 보자.

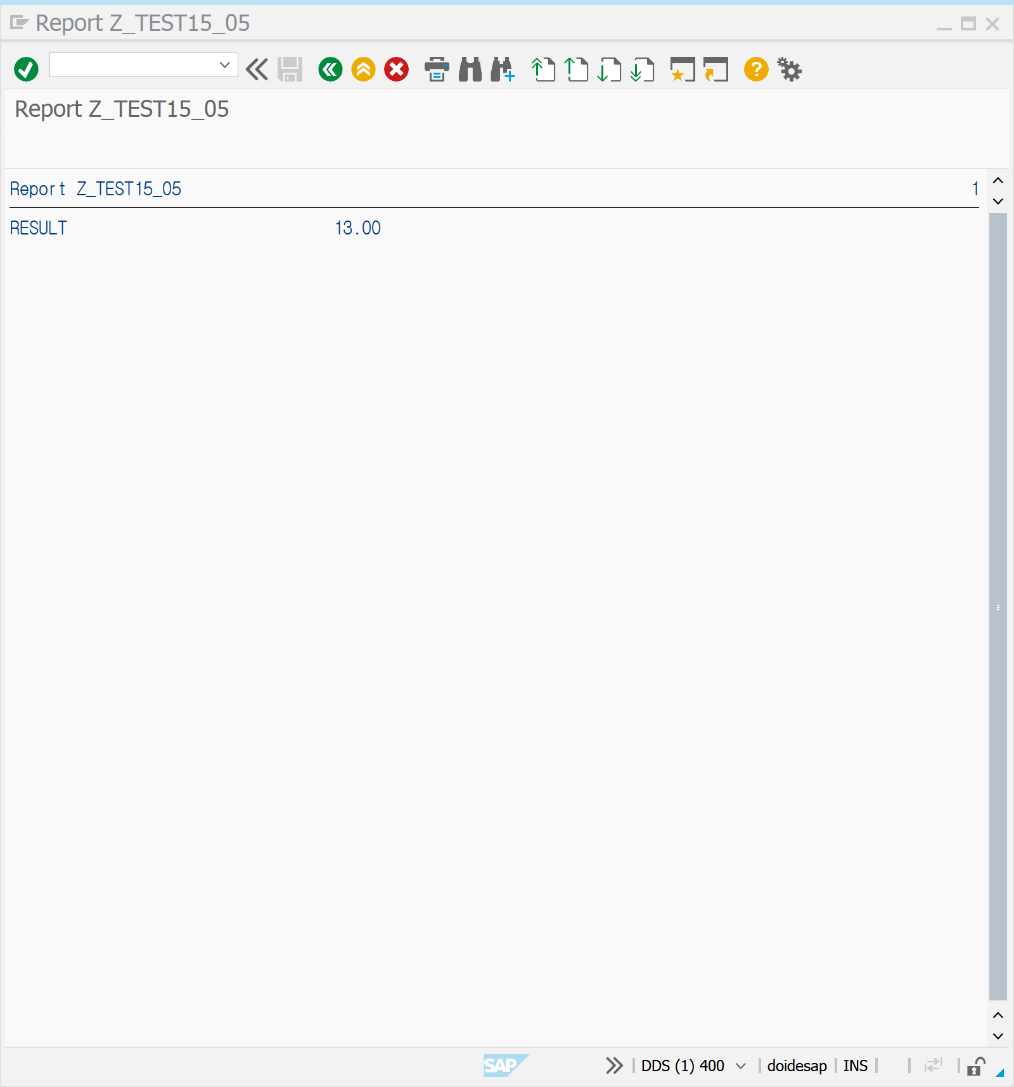
\*FIELD가 5개인 TABLE을 만들어보자.  
\*TYPES는 TABLE의 형태를 구성한다.  
TYPES: BEGIN OF TY\_STR,  
          CARRID TYPE S\_CARR\_ID,  
          CONNID TYPE S\_CONN\_ID,  
          CARNAME TYPE S\_CARRNAME,  
          CURRCO TYPE S\_CURRCODE,  
       END OF TY\_STR.  
  
       DATA GS\_STR TYPE TY\_STR.  
       DATA GT\_TABLE TYPE TABLE OF TY\_STR.  
       DATA GT\_TABLE1 LIKE TABLE OF GS\_STR.  
  
\*STRUCTURE  
DATA: BEGIN OF GS\_STR2,  
      CARRID TYPE S\_CARR\_ID,  
      CONNID TYPE S\_CONN\_ID,  
      CARNAME TYPE S\_CARRNAME,  
      CURRCO TYPE S\_CURRCODE,  
  END OF GS\_STR2.  
  
  BREAK-POINT.



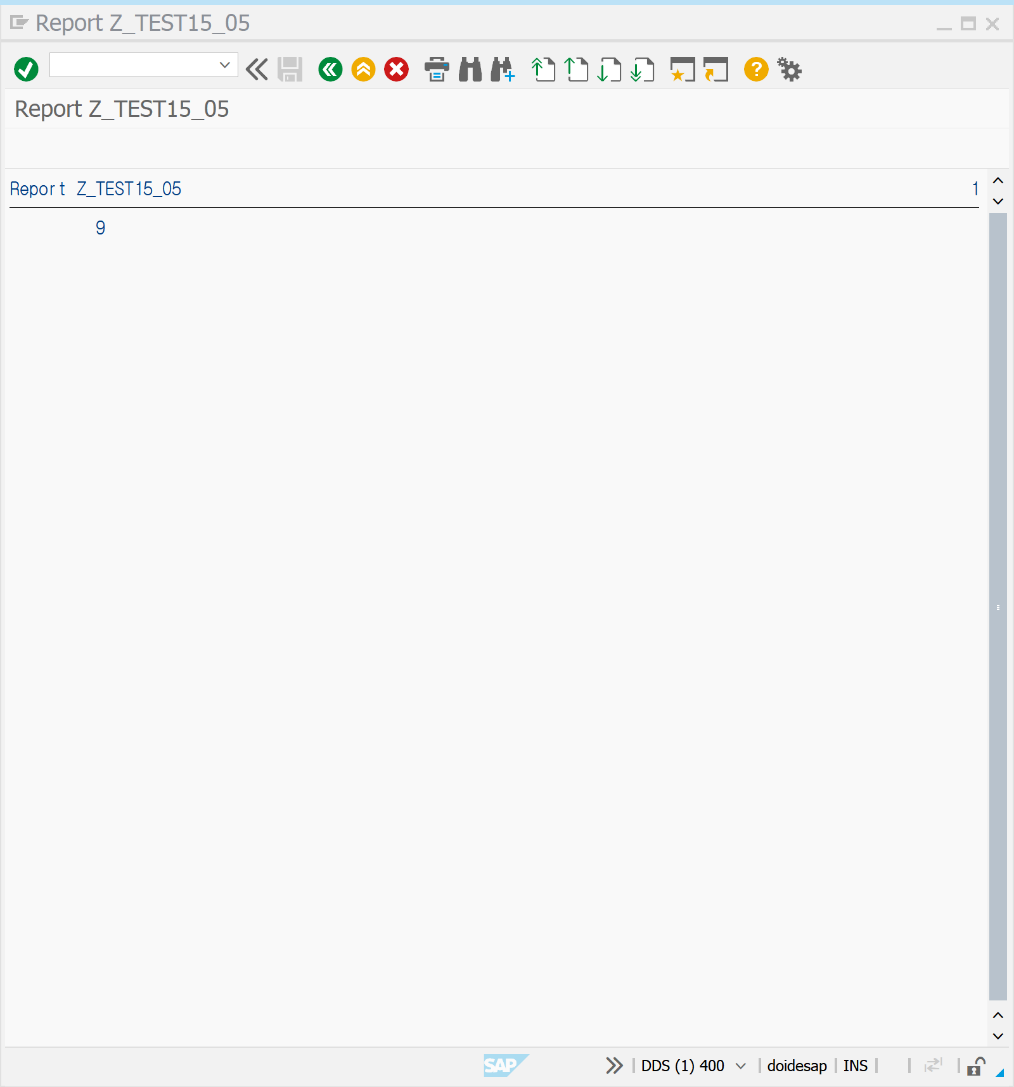
SE80 | TEST15 | 연산 기호를 사용한 계산기 프로그램을 생성해보자.

\*계산기 프로그램을 생성해보자  
PARAMETERS:  
    PA\_INT1 TYPE I,  
    PA\_OP TYPE C LENGTH 1,  
    PA\_INT2 TYPE I.  
  
DATA GV\_RESULT TYPE P LENGTH 16 DECIMALS 2.  
  
IF ( PA\_OP = '+' OR  
      PA\_OP = '-' OR  
      PA\_OP = '\*' OR  
      PA\_OP = '/' AND PA\_INT2 <> 0 ).  
  
 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.  
  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.

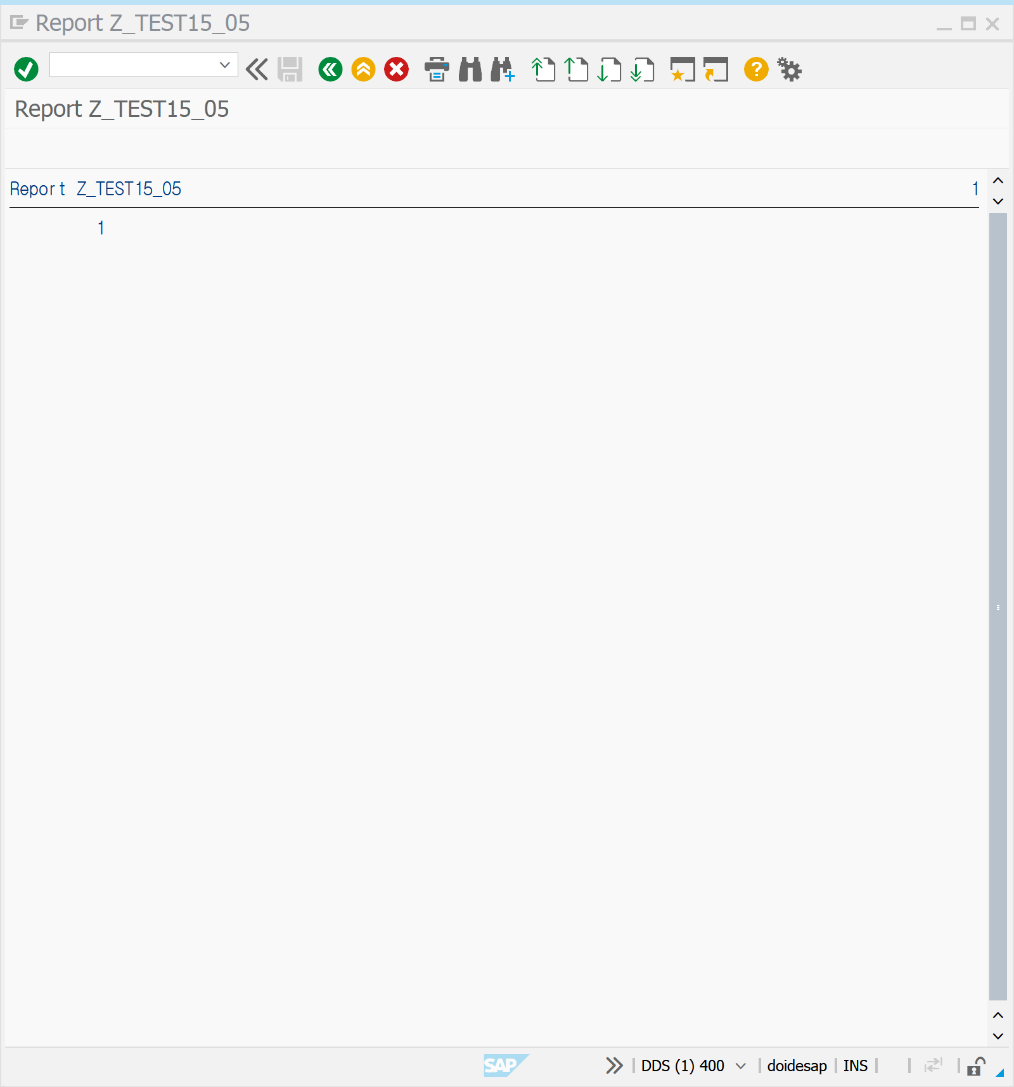




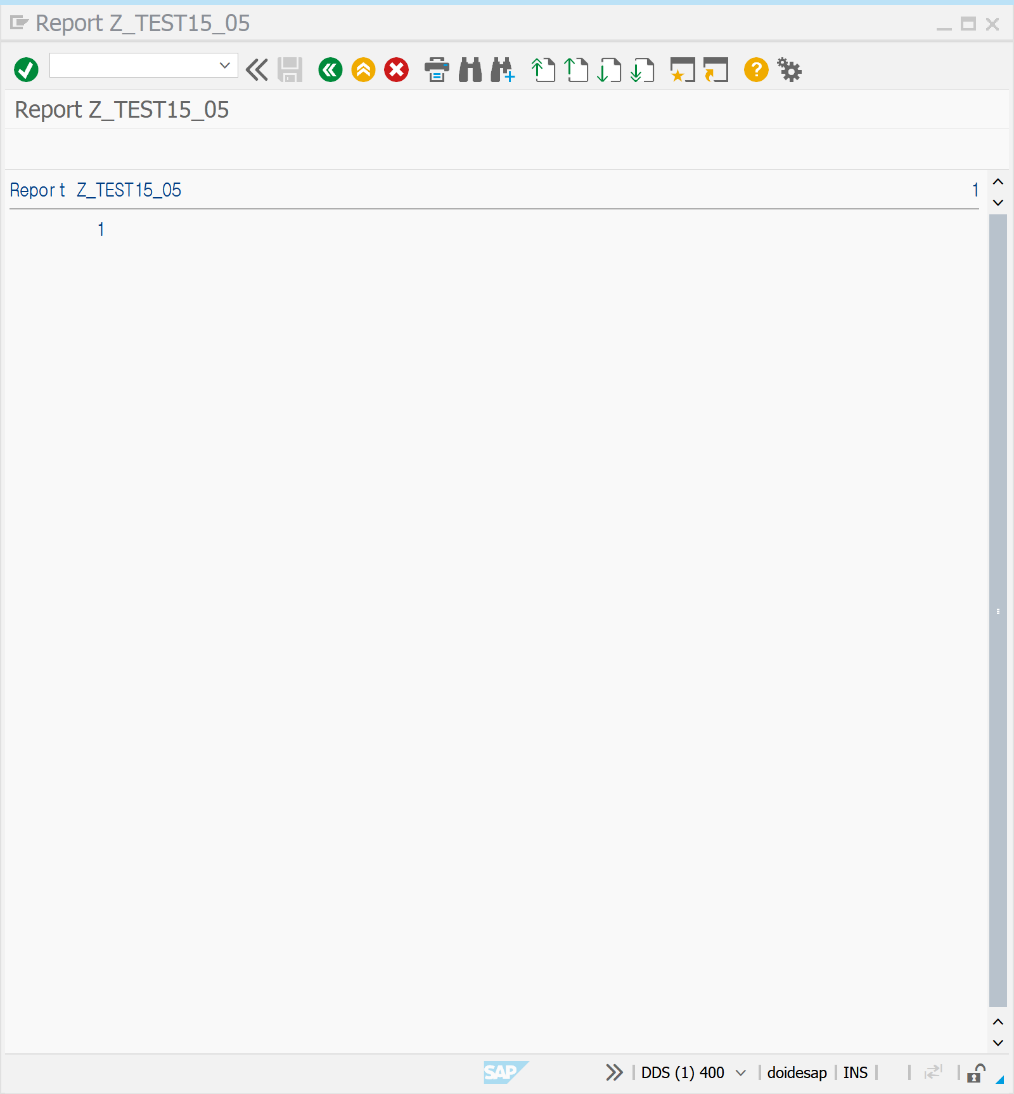
\* \*\*을 사용하면 어떤 결과 값이 나올지 수행해 보자. |제곱을 도출하는  수식  
\*DATA를 선언한다.  
DATA GV\_INT TYPE I.  
DATA GV\_INT2 TYPE I.  
DATA GV\_RESULT1 TYPE I.  
  
GV\_INT = 3.  
GV\_INT2 = 2.  
GV\_RESULT1 = GV\_INT \*\*  GV\_INT2.  
  
WRITE GV\_RESULT1.



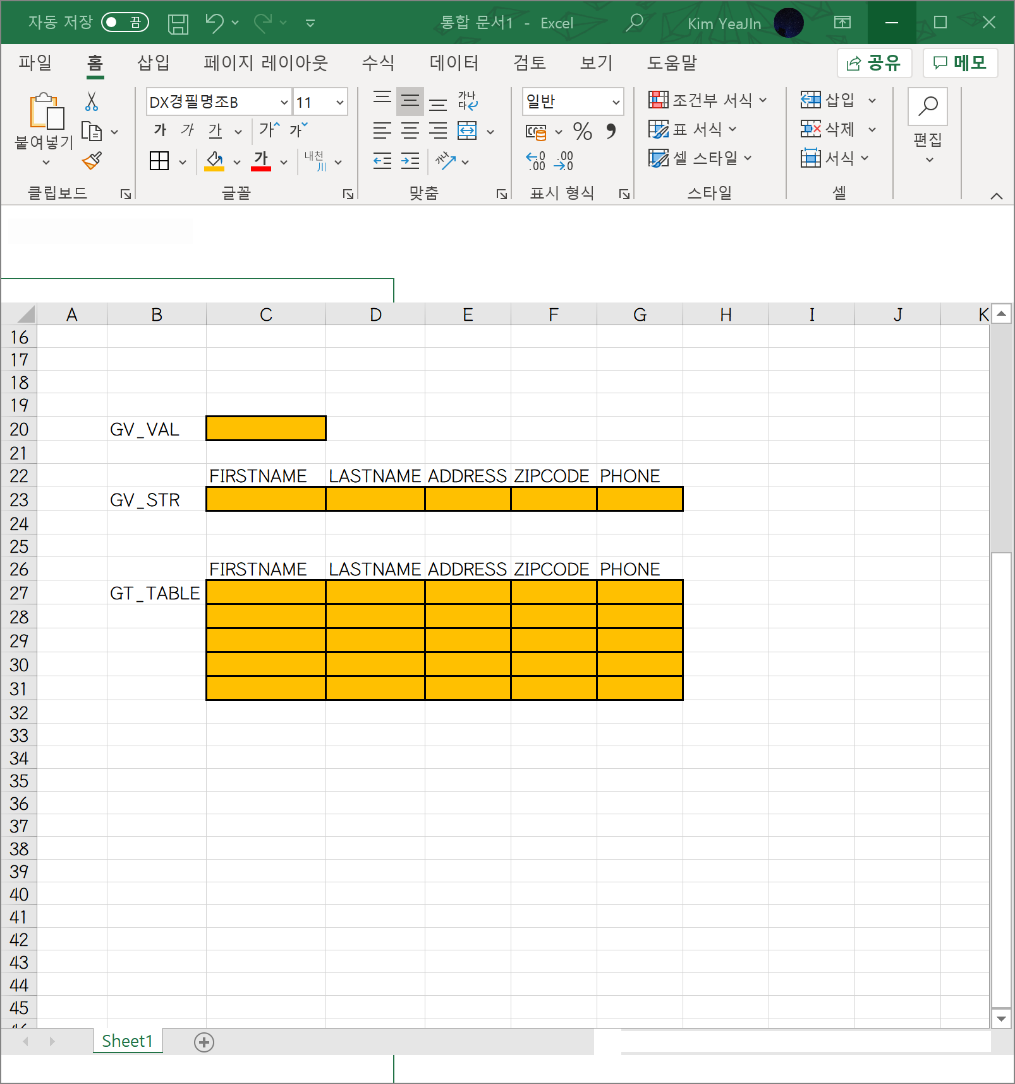
\* DIV을 사용하면 어떤 결과 값이 나올지 수행해 보자. | 몫을 도출해주는 수식  
\*DATA를 선언한다.  
DATA GV\_INT TYPE I.  
DATA GV\_INT2 TYPE I.  
DATA GV\_RESULT2 TYPE I.  
  
GV\_INT = 3.  
GV\_INT2 = 2.  
GV\_RESULT2 = GV\_INT DIV  GV\_INT2.  
  
WRITE GV\_RESULT2.



\* MOD을 사용하면 어떤 결과 값이 나올지 수행해 보자. | 나머지를  도출해주는 수식  
\*DATA를 선언한다.  
DATA GV\_INT TYPE I.  
DATA GV\_INT2 TYPE I.  
DATA GV\_RESULT3 TYPE I.  
  
GV\_INT = 3.  
GV\_INT2 = 2.  
GV\_RESULT3 = GV\_INT MOD  GV\_INT2.  
  
WRITE GV\_RESULT3.



SE80 | TEST15 | FIELD가 5개인 TABLE을 만들어 보자.



*\*\*실습을 해보자 #01*  
*\*PARAMETERS*  
PARAMETERS: FIRSTNA(6) TYPE C,  
                          LASTNA(8) TYPE C,  
                          ADDRESS(14) TYPE C,  
                          ZIPCODE(5) TYPE C,  
                          PHONE(15) TYPE C.  
  
*\*STRUCTURE*  
DATA: BEGIN OF GV\_STR,  
       FIRSTNAME(6) TYPE C,  
       LASTNAME(8) TYPE C,  
       ADDRESS(14) TYPE C,  
       ZIPCODE(5) TYPE C,  
       PHONE(15) TYPE C,  
  END OF GV\_STR.  
  
*\*TABLE*  
DATA: GT\_TABLE LIKE TABLE OF GV\_STR.  
*\*TYPE으로 정의한 LOCAL TYPE이 아니기 때문에 사용할 수 없다. LIKE 로 변경하자*  
*\*이는 STRUCTURE 참조하여 TABLE로 구성한다.*  
  
*\*TABLE의 DATA 값을 지정해 보자.*  
*\*\*DATA 값을 지정해주자.*  
GV\_STR-FIRSTNAME =  FIRSTNA.  
GV\_STR-LASTNAME = LASTNA.  
GV\_STR-ADDRESS = ADDRESS.  
GV\_STR-ZIPCODE = ZIPCODE.  
GV\_STR-PHONE = PHONE.  
  
WRITE GV\_STR.

*\*\*실습을 해보자 #02*  
*\*PARAMETERS*  
TYPES: BEGIN OF TY\_STR,  
            FIRSTNAME(6) TYPE C,  
            LASTNAME(8) TYPE C,  
            ADDRESS(14) TYPE C,  
            ZIPCODE(5) TYPE C,  
            PHONE(15) TYPE C,  
            END OF TY\_STR.  
  
DATA GS\_STR TYPE TY\_STR.  
DATA GT\_TABLE TYPE TABLE OF TY\_STR.  
  
*\*TABLE의 DATA 값을 지정해 보자.*  
GS\_STR-FIRSTNAME ='yeajin'.  
GS\_STR-LASTNAME ='kim'.  
GS\_STR-ADDRESS ='cheonan buldang'.  
GS\_STR-ZIPCODE ='31165'.  
GS\_STR-PHONE ='01074252579'.  
  
*\*\*\*APPEND 명령어를 이용하여 지정한 값을 TABLE에 삽입한다.*  
APPEND GS\_STR TO GT\_TABLE.  
  
BREAK-POINT.

*\*\*실습을 해보자 #03*  
*\*PARAMETERS*  
PARAMETERS: FNAME(6) TYPE C,  
                          LASTNA(8) TYPE C,  
                          ADDRESS(14) TYPE C,  
                          ZIPCODE(5) TYPE C,  
                          PHONE(15) TYPE C.  
  
*\*STRUCTURE*  
DATA: BEGIN OF GT\_TABLE OCCURS 0,  
       FIRSTNAME(6) TYPE C,  
       LASTNAME(8) TYPE C,  
       ADDRESS(14) TYPE C,  
       ZIPCODE(5) TYPE C,  
       PHONE(15) TYPE C,  
  END OF GT\_TABLE.  
  
  
GT\_TABLE-FIRSTNAME =  FNAME.  
GT\_TABLE-LASTNAME = LASTNA.  
GT\_TABLE-ADDRESS = ADDRESS.  
GT\_TABLE-ZIPCODE = ZIPCODE.  
GT\_TABLE-PHONE = PHONE.  
  
BREAK-POINT.