***ABAP Syntax***

***2020.08.01***

Contents

1 New Keyword in ABAP 5

DATA 5

FIELD-SYMBOL 6

NEW 6

REF 6

VALUE 7

BASE 8

FOR 9

LET 12

CONV 13

SWITCH 13

COND 14

CORRESPONDING 14

MOVE-CORRESPONDING（DEEP） 18

REDUCE 21

GROUP BY （FOR LOOP） 24

FILTER 26

EXACT 28

2 Open SQL 30

CONSTANT 30

Host Variable / Expression 31

Aggregate Expressions 32

Built-In Functions（Num） 33

Built-In Functions（String） 34

Built-In Functions（Date/Time） 36

Built-In Functions（Time Stamp） 37

Built-In Functions（Time Zone） 37

Built-In Functions（Date/Time Conversion） 38

CASE 39

NULL Value 39

CAST 41

CDS with Parameters 42

Internal Table 43

JOIN Expression 44

WHERE Condition 44

OFFSET 45

Strict Mode 46

3 Data Processing 47

String Template 47

Format Option 47

CURRENCY 48

COUNTRY 48

ALPHA 48

NUMBER/DATE/TIME 48

TIMESTAMP 49

TIMEZONE 49

WIDTH 50

ALIGN 50

PAD 50

CASE 50

SIGN 51

EXPONENT 51

DECIMALS 52

ZERO 52

XSD 52

STYLE 53

String Functions 54

STRLEN 54

DISTANCE 55

FIND 55

FIND\_END 56

FIND\_ANY\_OF 56

FIND\_ANY\_NOT\_OF 56

COUNT 57

COUNT\_ANY\_OF 57

COUNT\_ANY\_NOT\_OF 57

CONTAINS 57

CONTAINS\_ANY\_OF 57

CONTAINS\_ANY\_NOT\_OF 57

XSDBOOL 58

REPLACE 58

INSERT 59

CMAX/CMIN 59

CONDENSE 60

CONCAT\_LINES\_OF 60

ESCAPE 61

MATCH 61

REPEAT 61

REVERSE 62

TRANSLATE 62

TO\_MIXED 62

FORM\_MIXED 62

TO\_UPPER/TO\_LOWER 62

SHIFT\_LEFT/SHIFT\_RIGHT 63

SUBSTRING 64

SUBSTRING\_FROM 64

SUBSTRING\_AFTER 64

SUBSTRING\_BEFORE 64

SUBSTRING\_TO 64

SEGMENT 65

Numeric Functions 66

Internal Table Expressions 67

OPTIONAL 68

DEFAULT 68

Internal Table Functions 68

LINES 69

LINE\_EXISTS 69

LINE\_INDEX 69

4 Special Usages 70

TRY… CATCH… 70

CATCH … INTO … 70

RETRY 70

RAISE EXCEPTION 71

CLEANUP 71

CATCH BEFORE UNWIND 72

RESUME 73

Call Method 74

Demo Output 74

Demo Input 75

Fixed point arithmetic 76

# *1 New Keyword in ABAP*

## DATA

新语法允许在语句内部动态声明变量，该作用域与普通定义的变量范围一致

* ABAP、ICON 这些类型池语句不再需要通过TYPE-POOL声明

DATA(lv\_true\_false) = abap\_false.

DATA(lv\_icon) = icon\_material.

* 通过赋值的方式来动态声明变量，系统会根据所赋的值来确定变量类型

其中，字符类型会参照数据长度来指定

DATA(lv\_char) = 'This is a sentence'.

DATA(lv\_integer) = 15.

* 动态声明内表/结构，字段类型与 SELECT LIST 对应的字段一致，在包含常量字段时，需要赋别名，以指定内表/结构中的字段名

SELECT matnr, maktx FROM makt INTO TABLE @DATA(lt\_data).

SELECT SINGLE matnr, 'Constant' AS field  
  FROM makt  
  INTO @DATA(lwa\_data).

* 在 LOOP 内表时动态声明结构，可以避免使用带表头的内表

LOOP AT gt\_data INTO DATA(lwa\_data).

...

ENDLOOP.

* 在常见的操作语句中动态声明变量，多数情况下可以在 INTO 语句后使用，如下为连接/拆分字符串示例

CONCATENATE 'Text' '-' '01' INTO DATA(lv\_concat).

SPLIT lv\_concat AT '-' INTO DATA(lv\_part1) DATA(lv\_part2).  
SPLIT lv\_concat AT '-' INTO TABLE DATA(lt\_data).

* 在调用方法时动态声明变量来接收传出参数，可以避免因类型不一致而导致的dump

go\_splitter\_row->get\_container( EXPORTING row = 1  
                                          column = 1  
                                RECEIVING container = DATA(go\_cont) ).

## FIELD-SYMBOL

与DATA关键字类似，FIELD-SYMBOL允许在语句内部动态声明字段符号

* 在 LOOP 时动态声明字段符号，需要区分的是，普通定义使用的是 FIELD-SYMBOLS

LOOP AT gt\_data ASSIGNING FIELD-SYMBOL(<fs\_data>).

...  
ENDLOOP.

* 在 ASSIGN 时动态声明字段符号并分配

ASSIGN (lv\_fieldname) TO FIELD-SYMBOL(<fs\_field>).

## NEW

使用 NEW 创建（实例化）引用对象，用来代替CREATE OBJECT

* 使用 NEW 关键字时，如果等号左侧的对象还没有确定类型，则必须在 NEW 关键字后指定类型，如 GO\_GRID\_NEW；

如果是已经预定义的对象，则可以用 # 代替，如 GO\_GRID

例：

DATA(go\_grid\_new) = NEW cl\_gui\_alv\_grid( i\_parent = go\_con ).  
  
DATA: go\_grid TYPE REF TO cl\_gui\_alv\_grid.  
go\_grid = NEW #( i\_parent = go\_con ).

## REF

使用 REF 定义引用变量，用来代替 CREATE DATA

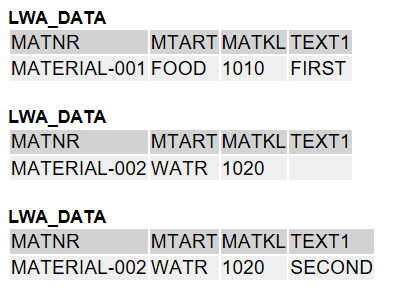
* 在使用 REF时，不需要提前声明变量，也不用指定类型，类型默认会与被指向的变量保持一致

例：

TYPES:BEGIN OF lty\_data,  
        matnr TYPE mara-matnr,  
        mtart TYPE mara-mtart,  
        matkl TYPE mara-matkl,  
        text1 TYPE char50,  
      END OF lty\_data.  
  
DATA(lwa\_data) = VALUE lty\_data( matnr = 'MATERIAL-001'  
                                 mtart = 'FOOD'  
                                 matkl = '1010'  
                                 text1 = 'FIRST' ).  
  
DATA(lv\_ref) = REF #( lwa\_data ).  
  
lv\_ref->\* = VALUE #( matnr = 'MATERIAL-002'  
                     mtart = 'WATR'  
                     matkl = '1020’ ).

lv\_ref->text1 = 'SECOND'.

测试结果：



## VALUE

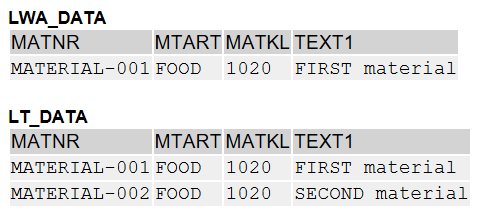
新语法中，可以使用 VALUE 作为赋值语句，主要用来为内表、结构、变量等对象赋值

* 参数类型引用同NEW关键字，在VALUE子句中，字段可以分开赋值，也可以使用结构整体赋值，为内表赋值时，需要用小括号将一行的数据包在一起

例：

TYPES:BEGIN OF lty\_data,  
        matnr TYPE mara-matnr,  
        mtart TYPE mara-mtart,  
        matkl TYPE mara-matkl,  
        text1 TYPE char50,  
      END OF lty\_data.  
        
DATA: lt\_data TYPE TABLE OF lty\_data.  
  
DATA(lwa\_data) = VALUE lty\_data( matnr = 'MATERIAL-001'  
                                 mtart = 'FOOD'  
                                 matkl = '1020'  
                                 text1 = 'FIRST material' ).  
  
lt\_data = VALUE #( ( lwa\_data )  
                   ( matnr = 'MATERIAL-002'  
                     mtart = 'FOOD'  
                     matkl = '1020'  
                     text1 = 'SECOND material' ) ).

测试结果：

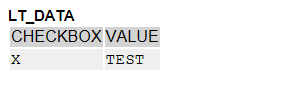


* 此外，VALUE语句作为结构时，可以直接在特定语句中使用

APPEND VALUE #( value = 'TEST' ) TO lt\_data.

MODIFY lt\_data FROM VALUE #( checkbox = 'X' ) TRANSPORTING checkbox WHERE checkbox IS INITIAL.

测试结果：



## BASE

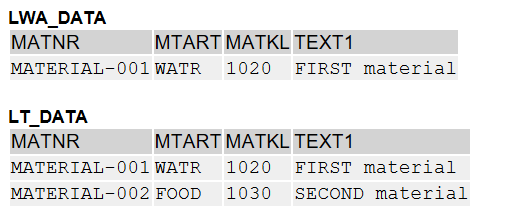
在使用 VALUE 作为赋值语句时，默认会覆盖原有的数据，通过BASE子句基于原有数据进行赋值

* 在结构赋值语句中使用 BASE 时，原有字段的数据会被保留，但是当在VALUE语句中对同一字段再次赋值时，该字段数据会被覆盖

例：

DATA(lwa\_data) = VALUE gty\_data( matnr = 'MATERIAL-001'  
                                 mtart = 'FOOD' ).  
  
lwa\_data = VALUE #( BASE lwa\_data  
                    mtart = 'WATR'  
                    matkl = '1020'  
                    text1 = 'FIRST material' ).

测试结果：

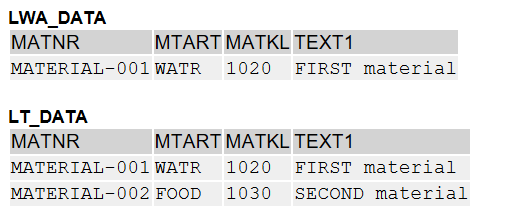


* 在内表赋值语句中使用 BASE 时，内表原有的数据会被保留，新增条目会被追加到内表中，效果同 Append Line

例：

lt\_data = VALUE #( ( lwa\_data ) ).  
  
lt\_data = VALUE #( BASE lt\_data  
                   ( matnr = 'MATERIAL-002'  
                     mtart = 'FOOD'  
                     matkl = '1030'  
                     text1 = 'SECOND material' ) ).

测试结果：



* 使用 BASE 语句时，尽量保持前后结构一致，在使用不同的结构时，可能不报错但数据会错位

## FOR

在内表赋值语句中，可以使用FOR语句从其他内表中批量引入数据并处理

* 使用FOR语句时，需要为内表定义临时工作区，如LWA\_DATA，仅允许在当前语句中使用，赋值过程中会使用到该工作区，但在WHERE条件里，只能直接使用内表的字段名，需要注意的是，WHERE后面接的条件语句必须使用小括号包起来，INDEX INTO 定义的临时变量可用来记录当前操作行的序列，作用与LOOP语句中的系统变量 SY-TABIX 类似

例：

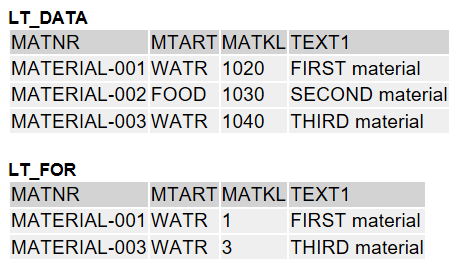
TYPES: lty\_table TYPE TABLE OF lty\_data WITH DEFAULT KEY.

lt\_data = VALUE #( ( matnr = 'MATERIAL-001'  
                     mtart = 'WATR'  
                     matkl = '1020'  
                     text1 = 'FIRST material' )  
                   ( matnr = 'MATERIAL-002'  
                     mtart = 'FOOD'  
                     matkl = '1030'  
                     text1 = 'SECOND material' )  
                   ( matnr = 'MATERIAL-003'  
                     mtart = 'WATR'  
                     matkl = '1040'  
                     text1 = 'THIRD material' ) ).  
  
DATA(lt\_for) = VALUE lty\_table( FOR lwa\_data IN lt\_data

INDEX INTO lv\_index

WHERE ( mtart = 'WATR' )  
                                ( matnr = lwa\_data-matnr  
         mtart = lwa\_data-mtart  
         matkl = CONV #( lv\_index )  
         text1 = lwa\_data-text1 ) ).

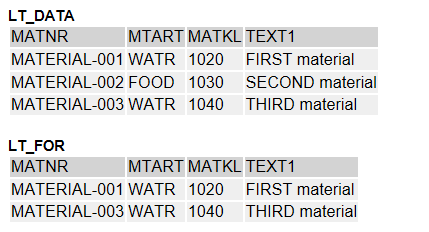
测试结果：



* 在FOR语句中允许将结构作为整体直接进行赋值，但是结构必须与表行兼容，可以用于从内表中获取特定条件的数据

DATA(lt\_for) = VALUE lty\_table( FOR lwa\_data IN lt\_data   
                                WHERE ( mtart = 'WATR' )  
                                ( lwa\_data ) ).

测试结果：

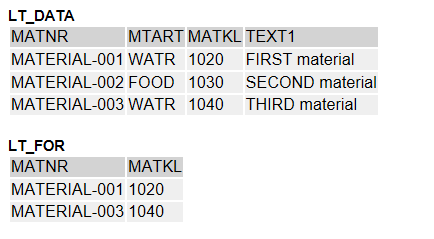


* 如果结构与表行不兼容，可以嵌套使用CORRESPONDING语句进行赋值，如下例（ CORRESPONDING具体用法可参考后续示例 ）

TYPES:BEGIN OF lty\_data\_new,  
        matnr TYPE mara-matnr,  
        matkl TYPE mara-matkl,  
      END OF lty\_data\_new.  
  
TYPES: lty\_table\_new TYPE TABLE OF lty\_data\_new WITH DEFAULT KEY.

DATA(lt\_for) = VALUE lty\_table\_new( FOR lwa\_data IN lt\_data  
                                    WHERE ( mtart = 'WATR' )  
                                    ( CORRESPONDING #( lwa\_data ) ) ).

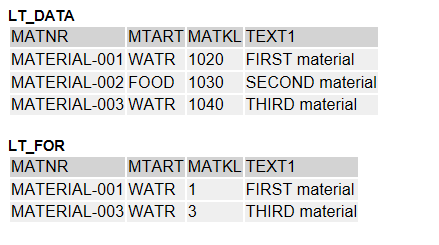
测试结果：



* 结构字段过多时，一般建议使用整体赋值，如果少部分字段存在特殊的赋值逻辑，可嵌套使用VALUE以及BASE语句进行处理，将例1的逻辑转换如下：

DATA(lt\_for) = VALUE lty\_table( FOR lwa\_data IN lt\_data  
                                INDEX INTO lv\_index  
                                WHERE ( mtart = 'WATR' )  
                                ( VALUE #( BASE lwa\_data  
                                           matkl = CONV #( lv\_index ) ) ) ).

测试结果：



## LET

使用 LET 引入短生命周期变量，可以用来简化部分冗余代码

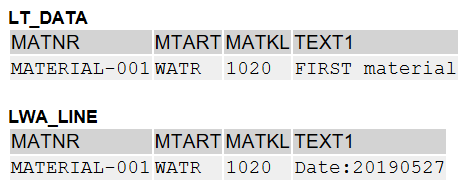
* LET关键字可以使用在VALUE，SWITCH，COND等语句中；

与 FOR 语句类似，LET 语句中定义的临时变量同样只能在当前语句中使用，在其他语句中使用时会检查出语法错误

例：

lwa\_line = VALUE #( LET  
       lwa\_data = VALUE #( lt\_data[ matnr = 'MATERIAL-001' ] OPTIONAL )  
        lv\_string = 'Date:' && sy-datum   
        IN  
        matnr = lwa\_data-matnr  
        mtart = lwa\_data-mtart  
        matkl = lwa\_data-matkl  
       text1 = lv\_string ).

测试结果：



## CONV

数据类型的转换可以用 CONV 实现，部分类型不再需要通过中间变量来转换

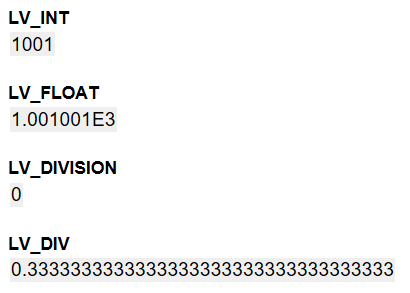
* 合理的使用 CONV 可以避免因为类型不一致而导致的 dump 问题，例如下例的LV\_RESULT，即在调用方法时做参数的类型转换
* 另外，在接口中处理传入参数时，一般情况下也会对数据的类型做对应的转换，但不是所有类型都可以互相转换，例如将含有非数字的 CHAR 类型数据强制转换成 INT 类型时，会抛出异常CX\_SY\_CONVERSION\_ERROR

例：

DATA(lv\_string) = '001001.001'.  
DATA(lv\_int) = CONV i( lv\_string ).  
DATA(lv\_float) = CONV f( lv\_string ).

DATA(lv\_division) = 1 / 3.  
DATA(lv\_div) = CONV decfloat34( 1 / 3 ).  
lv\_result = me->get\_char20( CONV #( me->get\_string( ) ) ).

测试结果：



## SWITCH

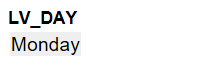
动态赋值语句，通常根据同一变量的不同数据来动态处理，用法类似于 CASE 语句

* SWITCH语句的判断条件相对单一，WHEN关键字后只能使用常量，THEN/ELSE后面可以使用表达式进行赋值

例：

DATA(lv\_indicator) = 1.  
DATA(lv\_day) = SWITCH char10( lv\_indicator  
                              WHEN 1 THEN 'Monday'  
                              WHEN 2 THEN 'Tuesday'  
                              WHEN 3 THEN 'Wednesday'  
                              WHEN 4 THEN 'Thursday'  
                              WHEN 5 THEN 'Friday'  
                              WHEN 6 THEN 'Saturday'  
                              WHEN 7 THEN 'Sunday'  
                              ELSE '404' && '-Error' ).

测试结果：



## COND

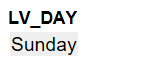
动态赋值语句，可以根据不同条件来动态处理，用法类似于CASE/IF语句

* COND语句中允许使用较为复杂的判断条件，因此VALUE语句中动态赋值通常会使用COND

例：

DATA(lv\_indicator) = 7.  
DATA(lv\_day) = COND char10( WHEN lv\_indicator = 1 THEN 'Monday'  
                            WHEN lv\_indicator = 2 THEN 'Tuesday'  
                            WHEN lv\_indicator = 3 THEN 'Wednesday'  
                            WHEN lv\_indicator = 4 THEN 'Thursday'  
                            WHEN lv\_indicator = 5 THEN 'Friday'  
                            WHEN lv\_indicator = 6 THEN 'Saturday'  
                            WHEN lv\_indicator = 7 AND sy-langu EQ 'E' THEN 'Sunday'  
                            WHEN lv\_indicator = 7 AND sy-langu EQ 'F' THEN 'Dimanche'  
                            ELSE '404' && '-Error' ).

测试结果：



## CORRESPONDING

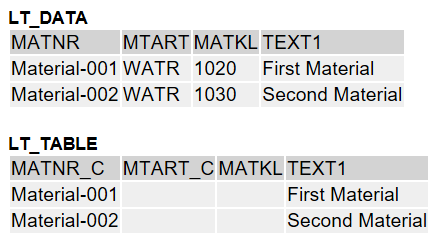
结构赋值语句，CORRESPONDING语句允许控制组件映射关系

* 在ABAP 7.40之前，主要通过 MOVE-CORRESPONDING 来传递结构化数据，但需要保持结构内部组件名称一致，否则数据将不会被传递，而使用 CORRESPONDING 后，该语句在保持同名组件自动进行数据传递的基础上，MAPPING 可以允许我们将不同名称的组件映射到一起，EXCEPT 可以规避掉我们不需要传值的一些字段

例：

lt\_data = VALUE #( ( matnr = 'Material-001'  
                     mtart = 'WATR'  
                     matkl = '1020'  
                     text1 = 'First Material' )  
                   ( matnr = 'Material-002'  
                     mtart = 'WATR'  
                     matkl = '1030'  
                     text1 = 'Second Material' ) ).  
  
DATA(lt\_table) = CORRESPONDING lty\_table( lt\_data  
                                          MAPPING matnr\_c = matnr  
                                          EXCEPT  matkl ).

测试结果：

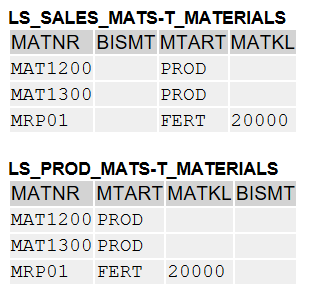


* 在 MAPPING 语句中，需要注意两边的字段类型，以免类型不兼容而导致程序 dump
* 使用DEEP处理深层结构数据，相当于MOVE CORRESPONDING [ EXPANDING NESTED TABLES ]

例：

TYPES: BEGIN OF lty\_prod\_mat,  
         matnr TYPE matnr,  
         mtart TYPE mtart,  
         matkl TYPE matkl,  
         bismt TYPE bismt,  
       END OF lty\_prod\_mat.  
  
DATA: BEGIN OF ls\_prod\_mats,  
        section     TYPE char10,  
        t\_materials TYPE TABLE OF lty\_prod\_mat,  
      END OF ls\_prod\_mats.  
  
TYPES: BEGIN OF lty\_sales\_mat,  
         matnr TYPE matnr,  
         bismt TYPE bismt,  
         mtart TYPE mtart,  
         matkl TYPE matkl,  
       END OF lty\_sales\_mat.  
  
DATA: BEGIN OF ls\_sales\_mats,  
        t\_materials TYPE TABLE OF lty\_sales\_mat,  
        section     TYPE char10,  
      END OF ls\_sales\_mats.  
  
SELECT matnr mtart matkl bismt FROM mara  
  INTO CORRESPONDING FIELDS OF TABLE ls\_sales\_mats-t\_materials  
 UP TO 3 ROWS.  
  
ls\_prod\_mats = CORRESPONDING #( DEEP ls\_sales\_mats ).

测试结果：

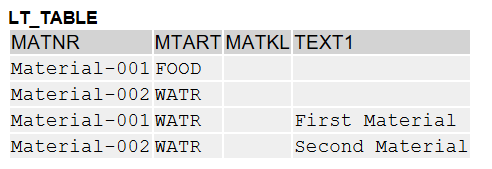


* 使用BASE保留初始数据，为内表赋值时相当于MOVE CORRESPONDING [ KEEPING TARGET LINES ]，为结构赋值时类似于VALUE [ BASE ]

例：

lt\_data = VALUE #( ( matnr = 'Material-001'  
                     mtart = 'WATR'  
                     matkl = '1020'  
                     text1 = 'First Material' )  
                   ( matnr = 'Material-002'  
                     mtart = 'WATR'  
                     matkl = '1030'  
                     text1 = 'Second Material' ) ).  
  
lt\_table = VALUE #( ( matnr = 'Material-001'  
                      mtart = 'FOOD' )  
                    ( matnr = 'Material-002'  
                      mtart = 'WATR') ).  
  
lt\_table = CORRESPONDING #( BASE ( lt\_table ) lt\_data EXCEPT matkl ).

测试结果：



* CORRESPONDING中可以从两个内表中引入数据，FORM子句后的内表结构需要定义为排序表或哈希表

该语法不能与DEEP/BASE关键字同时使用，但是可以使用MAPPING/EXCEPT

如下例所示，以表1为基表，根据USING定义的关联条件去表2查找数据，如果查找到，则将表2的该条数据复写到表1对应的记录上并返回，否则直接返回表1的数据

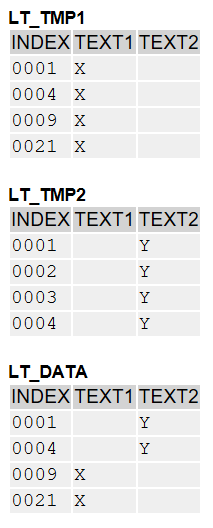
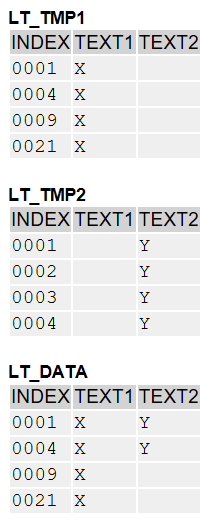
MAPPING/EXCEPT作用于表2数据复写到表1对应记录的过程中，特定条件下可以用来实现读取数据并更新内表的操作，但是需要定义非标准表，有一定的局限性

例：

TYPES:BEGIN OF lty\_data,  
        index TYPE char4,  
        text1 TYPE char10,  
        text2 TYPE char10,  
      END OF lty\_data.  
  
DATA: lt\_tmp1 TYPE SORTED TABLE OF lty\_data WITH UNIQUE KEY index,  
      lt\_tmp2 TYPE SORTED TABLE OF lty\_data WITH UNIQUE KEY index,  
      lt\_data TYPE TABLE OF lty\_data.  
  
lt\_tmp1 = VALUE #( ( index = '0001' text1 = 'X' )  
                   ( index = '0004' text1 = 'X' )  
                   ( index = '0009' text1 = 'X' )  
                   ( index = '0021' text1 = 'X' ) ).  
  
lt\_tmp2 = VALUE #( ( index = '0001' text2 = 'Y' )  
                   ( index = '0002' text2 = 'Y' )  
                   ( index = '0003' text2 = 'Y' )  
                   ( index = '0004' text2 = 'Y' ) ).  
  
lt\_data = CORRESPONDING #( lt\_tmp1 FROM lt\_tmp2 USING index = index ).

lt\_data = CORRESPONDING #( lt\_tmp1 FROM lt\_tmp2 USING index = index EXCEPT text1 ).

测试结果：

## MOVE-CORRESPONDING（DEEP）

在 ABAP 7.40 后，MOVE-CORRESPONDING语句针对深层结构赋值进行了扩展，并且允许保留结构在被赋值前的数据

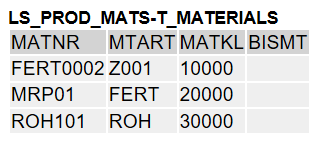
* 在深层结构中使用 MOVE-CORRESPONDING传递数据时，需要保持嵌套的深层次结构一致，包括字段名以及顺序等，否则数据会错位

例：

TYPES: BEGIN OF lty\_prod\_mat,  
         matnr TYPE matnr,  
         mtart TYPE mtart,  
         matkl TYPE matkl,  
         bismt TYPE bismt,  
       END OF lty\_prod\_mat.  
  
DATA: BEGIN OF ls\_prod\_mats,  
        section     TYPE char10,  
        t\_materials TYPE TABLE OF lty\_prod\_mat,  
      END OF ls\_prod\_mats.  
  
DATA: BEGIN OF ls\_sales\_mats,  
        section     TYPE char10,  
        t\_materials TYPE TABLE OF lty\_prod\_mat,  
      END OF ls\_sales\_mats.  
  
SELECT matnr mtart matkl bismt FROM mara  
  INTO CORRESPONDING FIELDS OF TABLE ls\_sales\_mats-t\_materials

UP TO 3 ROWS.  
  
MOVE-CORRESPONDING ls\_sales\_mats TO ls\_prod\_mats.

测试结果：



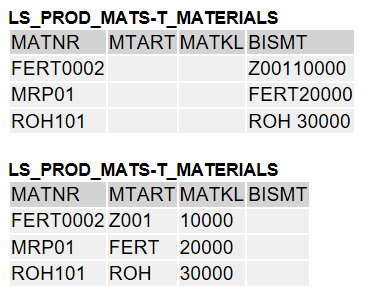
* 使用 EXPANDING NESTED TABLES 避免因结构不一致而导致的数据异常，对于深层结构，只有相同名称的字段被复制，且不需要它们顺序相同，但是该语句只考虑第一级的深层次结构，不适用于更深层次的数据

例：

TYPES: BEGIN OF lty\_sales\_mat,  
         matnr TYPE matnr,  
         bismt TYPE bismt,   
         mtart TYPE mtart,  
         matkl TYPE matkl,  
       END OF lty\_sales\_mat.   
  
DATA: BEGIN OF ls\_sales\_mats,  
        t\_materials TYPE TABLE OF lty\_sales\_mat,  
        section     TYPE char10,  
      END OF ls\_sales\_mats.

MOVE-CORRESPONDING ls\_sales\_mats TO ls\_prod\_mats.  
MOVE-CORRESPONDING ls\_sales\_mats TO ls\_prod\_mats EXPANDING NESTED TABLES.

测试结果：



* 在使用 MOVE-CORRESPONDING 时，默认都会先清空目标结构的数据，再将数据填充到结构里面，当我们需要保留目标结构原有的数据时，可以使用 KEEPING TARGET LINES 语句，该语句适用于表结构赋值，且不会修改原有的记录，仅添加需要填充的数据

例：

SELECT matnr mtart matkl bismt

  FROM mara

  INTO CORRESPONDING FIELDS OF TABLE ls\_sales\_mats-t\_materials

  UP TO 4 ROWS.

SELECT matnr mtart matkl bismt

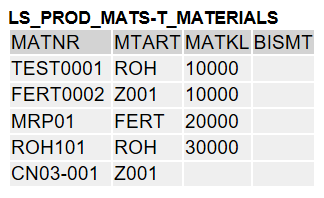
  FROM mara

  INTO CORRESPONDING FIELDS OF TABLE ls\_prod\_mats-t\_materials

  UP TO 1 ROWS.

MOVE-CORRESPONDING ls\_sales\_mats-t\_materials TO ls\_prod\_mats-t\_materials KEEPING TARGET LINES.

测试结果：



## REDUCE

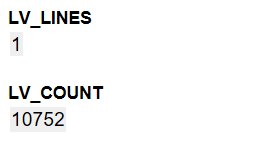
REDUCE赋值语句一般用于需要循环处理的数据

* 一般情况下，我们可以使用 REDUCE 来统计内表中特定条件的记录条数，或是汇总/拼接内表的部分字段

例：

lt\_data = VALUE #( ( matnr = 'Material-001'  
                     mtart = 'WATR'  
                     matkl = '1020'  
                     int   = 10086 )  
                   ( matnr = 'Material-002'  
                     mtart = 'WATR'  
                     matkl = '1030'  
                     int   = 666 ) ).  
  
lv\_lines = REDUCE #( INIT lv\_result = 0  
                     FOR ls\_data IN lt\_data WHERE ( matkl = '1020' )   
                     NEXT lv\_result = lv\_result + 1 ).

测试结果：

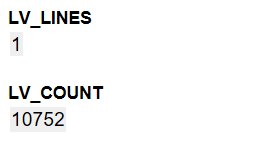


* REDUCE语句中可以使用UNTIL关键字，在使用时需要注意 NEXT 语句至少会被执行一次，作用类似于DO循环，限制条件可以自行定义

例：

lv\_count = REDUCE #( INIT lv\_compt = 0  
                     FOR i = 1 UNTIL i > lines( lt\_data )  
                     NEXT lv\_compt = lv\_compt + VALUE #( lt\_data[ i ]-int OPTIONAL ) ).

测试结果：



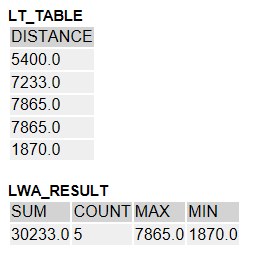
* REDUCE语句返回值为结构时，常用来汇总字段或作其他处理（例如取最大值/最小值/平均值等）

例：

TYPES:BEGIN OF lty\_result,  
        sum   TYPE s\_distance,

count TYPE i,  
        max   TYPE s\_distance,  
        min   TYPE s\_distance,  
      END OF lty\_result.  
  
TYPES: lty\_table TYPE TABLE OF lty\_result WITH DEFAULT KEY.  
SELECT distance FROM spfli INTO TABLE @DATA(lt\_table) UP TO 5 ROWS WHERE distance > 0.  
  
DATA(lwa\_result) = REDUCE #( INIT lwa\_tmp = VALUE lty\_result( )  
                             FOR lwa\_table IN lt\_table  
                             INDEX INTO lv\_index  
                             NEXT lwa\_tmp = VALUE #( BASE lwa\_tmp  
                                                     sum  = lwa\_tmp-sum + lwa\_table-distance  
                                                     count = lwa\_tmp-count + 1  
                                                     max  = nmax( val1 = lwa\_tmp-max val2 = lwa\_table-distance )  
                                                     min  = COND #( WHEN lv\_index = 1 THEN lwa\_table-distance  
                                                                    ELSE nmin( val1 = lwa\_tmp-min val2 = lwa\_table-distance ) ) ) ).

测试结果：



* REDUCE语句返回值为内表时，常用来汇总内表记录，可适用于深层嵌套的内表

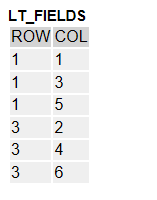
例：

TYPES:BEGIN OF lty\_field,  
        row TYPE i,  
        col TYPE i,  
      END OF lty\_field.  
  
TYPES:lty\_field\_tab TYPE TABLE OF lty\_field WITH DEFAULT KEY.  
  
TYPES:BEGIN OF lty\_data,  
        check TYPE char1,  
        field TYPE lty\_field\_tab,  
      END OF lty\_data.

DATA: lt\_data TYPE TABLE OF lty\_data.

lt\_data[] = VALUE #( ( check = abap\_true  
                       field = VALUE #( ( row = 1 col = 1 )  
                                        ( row = 1 col = 3 )  
                                        ( row = 1 col = 5 ) ) )  
                     ( check = abap\_false  
                       field = VALUE #( ( row = 2 col = 1 )  
                                        ( row = 2 col = 2 )  
                                        ( row = 2 col = 3 ) ) )  
                     ( check = abap\_true  
                       field = VALUE #( ( row = 3 col = 2 )  
                                        ( row = 3 col = 4 )  
                                        ( row = 3 col = 6 ) ) ) ).  
  
DATA(lt\_fields) = REDUCE #( INIT lt\_temp = VALUE lty\_field\_tab( )  
                            FOR ls\_data IN lt\_data  
                            WHERE ( check = abap\_true )  
                            NEXT lt\_temp = VALUE #( BASE lt\_temp  
                                                    FOR ls\_field IN ls\_data-field  
                                                    ( ls\_field ) ) ).

测试结果：



## GROUP BY （FOR LOOP）

在 LOOP 语句中使用 GROUP BY 实现分组处理数据

* 在 LOOP 中使用 GROUP BY 后，LWA\_DATA 中不会存储相应的数据，同样，如果使用 FIELD-SYMBOL，也不会被分配，如果需要修改内表数据，只能通过每个组进行修改，对内表数据进行分组时，可通过 ASCENDING / DESCENDING 按组排序，否则按前后的顺序依次输出；

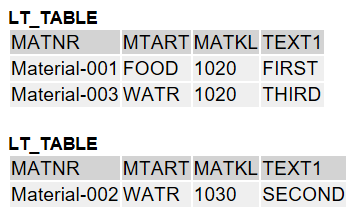
GROUP BY 在需要使用多个字段进行分组时：GROUP BY ( KEY1 = field1 KEY2 = field2 … )

例：

lt\_data = VALUE #( ( matnr = 'Material-001'  
                     mtart = 'FOOD'  
                     matkl = '1020'  
                     text1 = 'FIRST' )  
                   ( matnr = 'Material-002'  
                     mtart = 'WATR'  
                     matkl = '1030'  
                     text1 = 'SECOND' )  
                   ( matnr = 'Material-003'  
                     mtart = 'WATR'  
                     matkl = '1020'  
                     text1 = 'THIRD' ) ).

LOOP AT lt\_data INTO DATA(lwa\_data) GROUP BY lwa\_data-matkl INTO DATA(g1).  
  lt\_table = VALUE #( FOR lwa\_table IN GROUP g1 ( lwa\_table ) ).  
ENDLOOP.

测试结果：

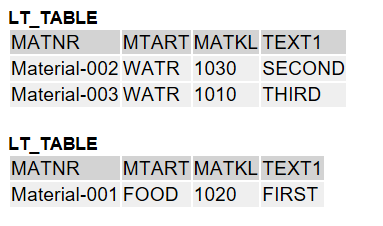


* 如果需要根据自定义条件进行分组，可以使用 COND 语句将特定条件转换成字符或数字再进行分组

例：

lt\_data = VALUE #( ( matnr = 'Material-001'  
                     mtart = 'FOOD'  
                     matkl = '1020'  
                     text1 = 'FIRST' )  
                   ( matnr = 'Material-002'  
                     mtart = 'WATR'  
                     matkl = '1030'  
                     text1 = 'SECOND' )  
                   ( matnr = 'Material-003'  
                     mtart = 'WATR'  
                     matkl = '1010'  
                     text1 = 'THIRD' ) ).   
  
LOOP AT lt\_data INTO DATA(lwa\_data)  
                GROUP BY COND string( WHEN lwa\_data-matkl = '1020' THEN 'A'  
                                      ELSE 'B' ) DESCENDING INTO DATA(g1).  
  lt\_table = VALUE #( FOR lwa\_table IN GROUP g1 ( lwa\_table ) ).  
ENDLOOP.

测试结果：



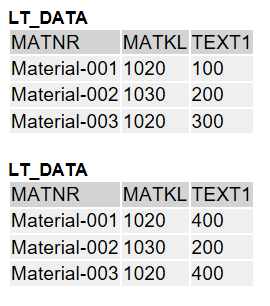
* 修改内表数据示例如下，第一层 LOOP 遍历的是每个组，第二层遍历的是对应组里的数据，我们需要在第二层做变更

例：

lt\_data = VALUE #( ( matnr = 'Material-001'  
                     matkl = '1020'  
                     text1 = 100 )  
                   ( matnr = 'Material-002'  
                     matkl = '1030'  
                     text1 = 200 )  
                   ( matnr = 'Material-003'  
                     matkl = '1020'  
                     text1 = 300 ) ).   
  
LOOP AT lt\_data INTO DATA(lwa\_data) GROUP BY lwa\_data-matkl ASCENDING INTO DATA(g1).  
  DATA(lv\_count) = REDUCE #( INIT lv\_index = 0

FOR lwa\_group IN GROUP g1  
                             NEXT lv\_index = lv\_index + lwa\_group-text1 ).  
  LOOP AT GROUP g1 ASSIGNING FIELD-SYMBOL(<fs\_line>).  
    <fs\_line>-text1 = lv\_count.  
  ENDLOOP.  
ENDLOOP.

测试结果：



## FILTER

使用 FILTER 根据条件来过滤内表数据

* 使用 FILTER 时，待过滤的内表结构至少需要有一个用于访问的 SORTED KEY 或 HASHED KEY，否则不能通过语法检查，另外，在 WHERE 条件中运算符两边的字段类型需要完全兼容，否则也不能通过语法检查；

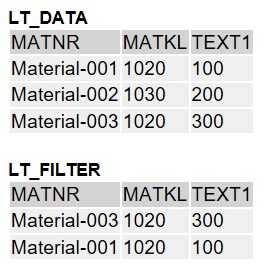
根据条件进行过滤的功能可以使用 VALUE 嵌套 FOR 语句实现，而且不用考虑内表的键值问题

例：

DATA:lt\_data TYPE TABLE OF lty\_data WITH KEY matnr WITH NON-UNIQUE SORTED KEY matkl COMPONENTS matkl.

lt\_data = VALUE #( ( matnr = 'Material-001'  
                     matkl = '1020'  
                     text1 = 100 )  
                   ( matnr = 'Material-002'  
                     matkl = '1030'  
                     text1 = 200 )  
                   ( matnr = 'Material-003'  
                     matkl = '1020'  
                     text1 = 300 ) ).  
  
DATA(lt\_filter) = FILTER #( lt\_data USING KEY matkl WHERE matkl = CONV matkl( '1020' ) ).

测试结果：



* 参照内表来过滤数据时，被参照的内表仍然需要有 SORTED KEY 或 HASHED KEY，对于过滤数据的内表没有键值要求；

在 FILTER 语句中可以通过 EXCEPT 关键字来指定是需要过滤数据还是保留数据；

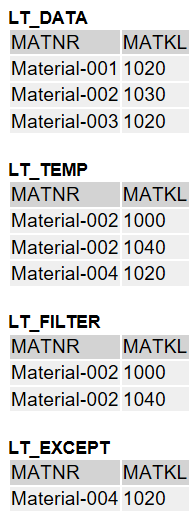
该语法可以实现 FOR ALL ENTRIES IN，但是需要将数据全部取出，影响性能，不建议使用，且在 ABAP 7.52 后，允许将内表作为数据源，可以用来代替 FOR ALL ENTRIES IN 使用

例：

DATA: lt\_data TYPE SORTED TABLE OF lty\_data WITH UNIQUE KEY matnr,  
      lt\_temp TYPE TABLE OF lty\_data.   
  
lt\_data = VALUE #( ( matnr = 'Material-001' matkl = '1020' )  
                   ( matnr = 'Material-002' matkl = '1030' )  
                   ( matnr = 'Material-003' matkl = '1020' ) ).  
  
lt\_temp = VALUE #( ( matnr = 'Material-002' matkl = '1000' )  
                   ( matnr = 'Material-002' matkl = ’1040' )  
                   ( matnr = 'Material-004' matkl = '1020' ) ).  
  
DATA(lt\_filter) = FILTER #( lt\_temp IN lt\_data WHERE matnr = matnr ).

DATA(lt\_except) = FILTER #( lt\_temp EXCEPT IN lt\_data WHERE matnr = matnr ).

测试结果：



## EXACT

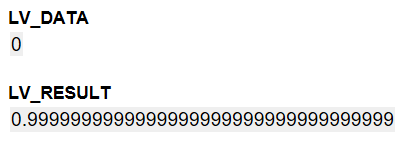
关键字 EXACT 可以用来检查操作语句返回值是否存在丢失，如果存在丢失则会抛出异常

* 抛出异常的范围比CONV更广，例如将CHAR10的数据赋值到CHAR1时，因此在使用时需要注意异常的捕获，如果没有特殊的处理或属性需求，可以直接使用父类异常 CX\_SY\_CONVERSION\_ERROR 进行捕获
* 判断数值语句是否被精确计算，如下实际抛出的异常是CX\_SY\_CONVERSION\_ROUNDING，获取到该异常类中的属性字段VALUE的数据

例：

TRY .  
    DATA(lv\_data) = EXACT #( 3 \* ( 1 / 3 ) ).  
  CATCH cx\_sy\_conversion\_rounding INTO DATA(lo\_except).  
    DATA(lv\_result) = lo\_except->value.  
ENDTRY.

测试结果：

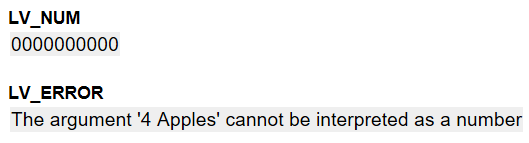


* 字段类型不兼容时，如下实际抛出的异常是 CX\_SY\_CONVERSION\_NO\_NUMBER，这里获取的是父类CX\_SY\_CONVERSION\_ERROR中的返回信息

例：

TYPES lty\_num TYPE n LENGTH 10.  
TRY .  
  DATA(lv\_num) = EXACT lty\_num( '4 Apples' ).  
CATCH cx\_sy\_conversion\_error INTO DATA(lo\_convert).  
  DATA(lv\_error) = lo\_convert->get\_text( ).  
ENDTRY.

测试结果：



# *2 Open SQL*

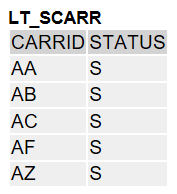
## CONSTANT

* 常量字段可以用来为内表中的部分字段赋初始值

例：

SELECT carrid, 'S' AS status  
  FROM scarr  
    UP TO 5 ROWS  
  INTO TABLE @DATA(lt\_scarr).

测试结果：

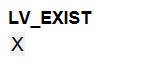


* 当只需要判断数据库表中是否存在特定的记录并且不用取表数据时，可以用常量字段代替

例：

DATA(lv\_carrier) = CONV s\_carr\_id( 'AA' ).  
  
SELECT SINGLE @abap\_true  
  FROM scarr  
 WHERE carrid = @lv\_carrier  
  INTO @DATA(lv\_exist).  
  
IF lv\_exist = abap\_true.  
  *"...*  
ENDIF.

测试结果：

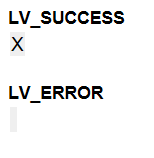


* 上例也可以用COUNT( \* ) / COUNT(\*) 代替，使用COUNT( \* ) 来查找特定记录时，可以不指定INTO语句，即不需要声明变量来存储数据，直接判断SUBRC的返回值

例：

DATA(lv\_carrier) = CONV s\_carr\_id( 'AA' ).  
  
SELECT COUNT( \* )  
  FROM scarr  
 WHERE carrid = @lv\_carrier.  
IF sy-subrc IS INITIAL.  
  DATA(lv\_success) = 'X'.  
ELSE.  
  DATA(lv\_error) = 'X'.  
ENDIF.

测试结果：



## Host Variable / Expression

在查询语句中使用 @ 作为转义符

* 通常在查询语句中，程序声明（非数据库层级）的变量前需要使用转义符 @ 进行标识，这些 Host Variable 通常被用作 Open SQL 语句中的操作数

在查询语句内部声明结构/内表时，应该在 DATA 前使用转义符

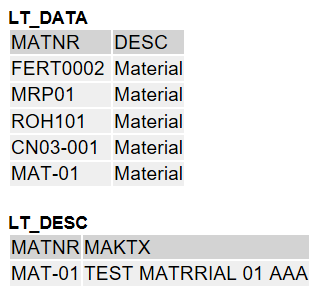
此外，在使用 Host Expression（在 Open SQL 中作为操作数使用的一些表达式） 时，也需要添加转义字符，如下例所示

（注：表达式内部的变量不需要再使用转义符，且不能使用表达式外部的数据库对象）

例：

SELECT matnr,  
       @( COND #( WHEN sy-langu = 'E' THEN 'Material'  
                  WHEN sy-langu = '1' THEN '物料' ) ) AS desc  
  FROM mara  
  INTO TABLE @DATA(lt\_data)  
    UP TO @( CONV #( '005' ) ) ROWS.   
  
SELECT matnr, maktx  
  FROM makt  
 WHERE spras = @sy-langu  
   AND matnr = @( VALUE #( lt\_data[ 5 ]-matnr OPTIONAL ) )  
  INTO TABLE @DATA(lt\_desc).

测试结果：



## Aggregate Expressions

聚合表达式用于对一组值执行计算并返回单一的值，可以使用在SELECT或HAVING子句中，不能用在WHERE子句

* WHERE与HAVING的区别：

WHERE 子句的搜索条件在进行分组操作之前应用；而 HAVING 的搜索条件则在进行分组操作之后应用

* 常见的聚合表达式如下，表达式内部可选用DISTINCT对数据去重后再进行处理：

AVG：返回结果集的平均值，返回类型默认为浮点型，可通过AS语句返回指定类型，如DEC，CURR，QUAN或FLTP

MAX：返回结果集的最大值

MIN：返回结果集的最小值

SUM：返回结果集的汇总值

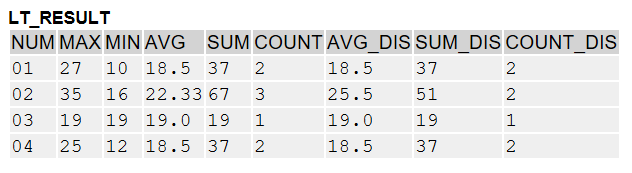
COUNT：返回结果集的条目数，通常情况下使用COUNT( \* ) / COUNT(\*)，需要使用DISTINCT时则要指定字段名

例：

TYPES:BEGIN OF lty\_data,  
        num   TYPE char2,  
        value TYPE i,  
      END OF lty\_data.  
  
DATA: lt\_data TYPE TABLE OF lty\_data.  
  
lt\_data = VALUE #( ( num = '01' value = 10 )  
                   ( num = '01' value = 27 )  
                   ( num = '02' value = 16 )  
                   ( num = '02' value = 35 )  
                   ( num = '02' value = 16 )  
                   ( num = '03' value = 19 )  
                   ( num = '04' value = 12 )  
                   ( num = '04' value = 25 ) ).

SELECT num,  
       MAX( value ) AS max,  
       MIN( value ) AS min,  
       AVG( value AS DEC( 12,2 ) ) AS avg,  
       SUM( value ) AS sum,  
       COUNT(\*) AS count,  
       AVG( DISTINCT value AS DEC( 12,2 ) ) AS avg\_dis,  
       SUM( DISTINCT value ) AS sum\_dis,  
       COUNT( DISTINCT value ) AS count\_dis  
  FROM @lt\_data AS a  
 GROUP BY num  
 ORDER BY num  
  INTO TABLE @DATA(lt\_result).

测试结果：



## Built-In Functions（Num）

在 SELECT LIST 使用内嵌表达式来处理数值，使用时需注意传入参数的类型

* 常见的数值表达式如下：

ABS：获取绝对值

CEIL：向上取整

FLOOR：向下取整

DIV：除法计算，取整数位

DIVISION：除法计算，保留 N 位小数

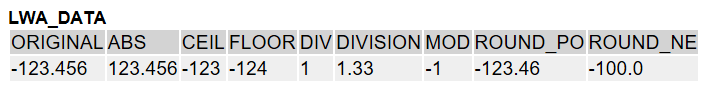
MOD：除法计算，取余数

ROUND：计算舍入值

例：

DATA(lv\_dec) = CONV zdec\_3\_demo( '-123.456' ).  
  
SELECT SINGLE  
       @lv\_dec AS original,  
       abs( @lv\_dec ) AS abs,  
       ceil( @lv\_dec ) AS ceil,  
       floor( @lv\_dec ) AS floor,  
       div( -4 , -3 ) AS div,  
       division( -4 , -3 , 2 ) AS division,  
       mod( -4 , -3 ) AS mod,  
       round( @lv\_dec , 2 ) AS round\_po,  
       round( @lv\_dec , -2 ) AS round\_ne  
  FROM sflight  
  INTO @DATA(lwa\_data).

测试结果：



## Built-In Functions（String）

在 SELECT LIST 使用内嵌表达式来处理字符串, 通常情况下字符串返回结果不能超过255个字符，如果字符长度异常，语法检查时会有错误提示

* CONCAT：连接字符串，参数固定为2个，各个表达式之间可以嵌套使用，CONCAT内部也可以使用 &&
* &&：连接字符串，参数没有个数限制，但不能将其他内嵌表达式当作参数使用，仅作为操作符使用，在非SELECT语句中也可以被使用
* CONCAT\_WITH\_SPACE：连接字符串，并用 N 个空格分隔，该表达式结果不能超过1333个字符
* INSTR：遍历字符串，查找指定字符 s1 并返回第一次出现的位置，没有查到则返回0
* LEFT/RIGHT：从字符串左/右侧开始取出 N 位字符，忽略前导/尾部的空格
* LENGTH：计算字符串长度

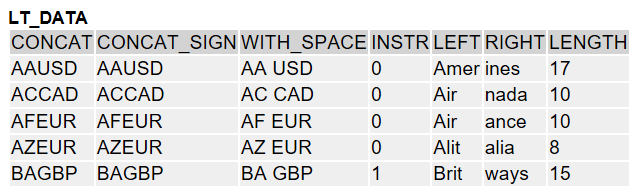
例：

SELECT concat( carrid , currcode ) AS concat,

       carrid && currcode AS concat\_sign,  
       concat\_with\_space( carrid , currcode , 1 ) AS with\_space,  
       instr( carrid , 'BA' ) AS instr,  
       left( carrname , 4 ) AS left,

       right( carrname , 4 ) AS right,  
       length( carrname ) AS length  
  FROM scarr  
  INTO TABLE @DATA(lt\_data)  
    UP TO 5 ROWS.

测试结果：

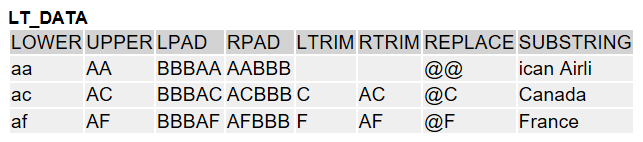


* LOWER/UPPER：将字符串转换成小写/大写
* LPAD/RPAD：在字符串左侧/右侧填充指定字符 s1，直到字符串长度为 N，如果初始值长度>=N，则不会填充，但是超过 N 位的字符会被截断
* LTRIM/RTRIM：从字符串左侧/右侧开始逐个删除指定字符 s1，直到出现其他字符为止，默认会删除尾部的空格
* REPLACE：将字符串中所有的指定字符 s1 用其他字符 s2 代替
* SUBSTRING：从字符串第 N 位开始截取长度为 M 的字符，系统会默认检查截取范围是否超出该字段最大长度，以避免造成DUMP

例：

SELECT lower( carrid ) AS lower,  
       upper( carrid ) AS upper,  
       lpad( carrid , 5 , 'B' ) AS lpad,  
       rpad( carrid , 5 , 'B' ) AS rpad,  
       ltrim( carrid , 'A' ) AS ltrim,  
       rtrim( carrid , 'A' ) AS rtrim,  
       replace( carrid , 'A' , '@' ) AS replace,  
       substring( carrname , 5 , 10 ) AS substring   
  FROM scarr  
  INTO TABLE @DATA(lt\_data)  
    UP TO 3 ROWS.

测试结果：



## Built-In Functions（Date/Time）

在 SELECT LIST 使用内嵌表达式来处理日期/时间

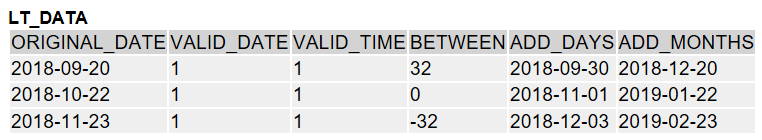
* DATS\_IS\_VALID/TIMS\_IS\_VALID：校验日期/时间有效性，有效时返回 1，否则返回 0
* DATS\_DAYS\_BETWEEN：计算日期d1和d2相隔的天数
* DATS\_ADD\_DAYS：为指定日期加上N天
* DATS\_ADD\_MONTHS：为指定日期加上N月

例：

DATA(lv\_date) = CONV datum( '20181022' ).

SELECT fldate AS original\_date,  
       dats\_is\_valid( fldate ) AS valid\_date,  
       tims\_is\_valid( @sy-uzeit ) AS valid\_time,   
       dats\_days\_between( fldate , @lv\_date ) AS between,  
       dats\_add\_days( fldate , 10 ) AS add\_days,  
       dats\_add\_months( fldate , 3 ) AS add\_months  
  FROM sflight  
  INTO TABLE @DATA(lt\_data)  
    UP TO 3 ROWS.

测试结果：



## Built-In Functions（Time Stamp）

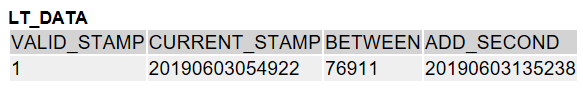
在 SELECT LIST 使用内嵌表达式来处理时间戳

* TSTMP\_IS\_VALID：校验时间戳有效性，有效时返回 1，否则返回 0
* TSTMP\_CURRENT\_UTCTIMESTAMP：返回当前时间戳
* TSTMP\_SECONDS\_BETWEEN：计算时间戳 t1 和 t2 相隔的秒数，需要用赋值语句进行传参，可以添加相应的错误处理
* TSTMP\_ADD\_SECONDS：为指定时间戳加上 N 秒，N 必须为 timestamp 类型

例：

DATA(lv\_stamp\_now) = CONV timestamp( '20190603133559' ).  
DATA(lv\_stamp\_past) = CONV timestamp( '20190602161408' ).  
  
SELECT tstmp\_is\_valid( @lv\_stamp\_now ) AS valid\_stamp,  
       tstmp\_current\_utctimestamp( ) AS current\_stamp,  
       tstmp\_seconds\_between( tstmp1 = @lv\_stamp\_past,  
                              tstmp2 = @lv\_stamp\_now,  
                              on\_error = @sql\_tstmp\_seconds\_between=>set\_to\_null ) AS between,  
       tstmp\_add\_seconds( tstmp = @lv\_stamp\_now,  
                          seconds = @( CONV timestamp( 999 ) ),  
                          on\_error = @sql\_tstmp\_add\_seconds=>set\_to\_null ) AS add\_second  
  FROM sflight  
  INTO TABLE @DATA(lt\_data)  
    UP TO 1 ROWS.

测试结果：



## Built-In Functions（Time Zone）

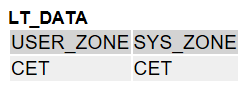
在 SELECT LIST 使用内嵌表达式来处理时区

* ABAP\_USER\_TIMEZONE：获取用户时区，不传参时默认获取当前用户当前 Client 的时区
* ABAP\_SYSTEM\_TIMEZONE：获取系统时区，不传参时默认获取当前 Client 的时区

例：

SELECT abap\_user\_timezone( user = @( CONV uname( 'JIANGRE' ) ),  
                           client = '130',  
                           on\_error = @sql\_abap\_user\_timezone=>set\_to\_null ) AS user\_zone,  
       abap\_system\_timezone( client = '130',  
                             on\_error = @sql\_abap\_system\_timezone=>set\_to\_null ) AS sys\_zone  
  FROM sflight  
  INTO TABLE @DATA(lt\_data)  
    UP TO 1 ROWS.

测试结果：



## Built-In Functions（Date/Time Conversion）

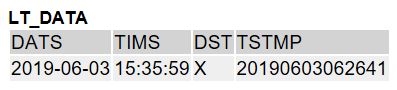
在 SELECT LIST 使用内嵌表达式来转换日期/时间/时间戳

* TSTMP\_TO\_DATS：将时间戳转换成对应时区的日期
* TSTMP\_TO\_TIMS：将时间戳转换成对应时区的时间
* TSTMP\_TO\_DST：根据时间戳获取对应时区的夏令时标识
* DATS\_TIMS\_TO\_TSTMP：将日期和时间根据时区转换成时间戳

例：

DATA(lv\_stamp) = CONV timestamp( '20190603133559' ).  
  
SELECT tstmp\_to\_dats( tstmp = @lv\_stamp,  
                      tzone = @( CONV tznzone( 'CET' ) ) ) AS dats,  
       tstmp\_to\_tims( tstmp = @lv\_stamp,  
                      tzone = @( CONV tznzone( 'CET' ) ) ) AS tims,  
       tstmp\_to\_dst( tstmp = @lv\_stamp,  
                     tzone = @( CONV tznzone( 'CET' ) ) ) AS dst,  
       dats\_tims\_to\_tstmp( date = @sy-datum,  
                           time = @sy-uzeit,  
                           tzone = @( CONV tznzone( 'CET' ) ) ) AS tstmp  
  FROM sflight  
  INTO TABLE @DATA(lt\_data)  
    UP TO 1 ROWS.

测试结果：



## CASE

在 SELECT 语句中使用 CASE 作为条件语句，与一般条件判断使用的 CASE 类似，但有所区别

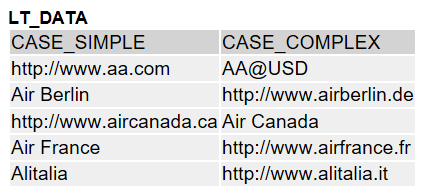
* 该语句不仅可以用于单值判断，也可以根据复杂条件进行判断；

此外，WHEN OTHERS 不再适用，需要使用 ELSE 代替，语句结束时使用 END，而不是 ENDCASE，且需要定义别名

例：

SELECT CASE currcode  
       WHEN 'EUR' THEN carrname  
       ELSE url  
       END AS case\_simple,  
  
       CASE  
       WHEN currcode = 'EUR' THEN url  
       WHEN carrname <> ' '  THEN carrname  
       ELSE carrid && '@' && currcode  
       END AS case\_complex  
  FROM scarr  
  INTO TABLE @DATA(lt\_data)  
    UP TO 5 ROWS.

测试结果：



## NULL Value

使用条件语句判断并处理 NULL 值

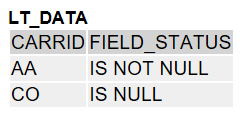
* 在使用 LEFT / RIGHT OUTER JOIN 关联外表时，如果主表中存在记录，但在外表中没有关联到数据，则外表的这部分字段的值在取数过程中始终为 NULL，在取数完成后传入数据对象时，NULL 会再转换成系统兼容的值，通常为初始值；

NULL 值用于数值计算或是字符串处理时返回结果仍为NULL值，可以在条件语句中用 IS [ NOT ] NULL 判断以及处理

例：

DATA: lr\_carrid TYPE RANGE OF s\_carr\_id.  
  
lr\_carrid = VALUE #( sign = 'I' option = 'EQ' ( low = 'AA' )  
                                              ( low = 'CO' ) ).  
SELECT DISTINCT  
       r~carrid,  
       CASE  
       WHEN t~seatsocc IS NULL THEN 'IS NULL'  
       WHEN t~seatsocc IS NOT NULL THEN 'IS NOT NULL'  
       END AS field\_status  
  FROM scarr AS r  
  LEFT OUTER JOIN sflight AS t ON t~carrid = r~carrid  
  INTO TABLE @DATA(lt\_data)  
 WHERE r~carrid IN @lr\_carrid.

测试结果：



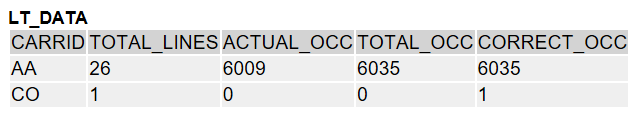
* 使用内嵌表达式 COALESCE( arg1, arg2, arg3 ... argn )处理NULL值

该表达式用来返回第一个非 NULL 字段，参数至少有2个，至多255个，如果参数都为 NULL，则返回 NULL

例：

DATA: lr\_carrid TYPE RANGE OF s\_carr\_id.  
  
lr\_carrid = VALUE #( sign = 'I' option = 'EQ' ( low = 'AA' )  
                                              ( low = 'CO' ) ).  
SELECT r~carrid,  
       SUM( 1 ) AS total\_lines,  
       SUM( t~seatsocc ) AS actual\_occ,  
       SUM( t~seatsocc + 1 ) AS total\_occ,  
       SUM( coalesce( t~seatsocc + 1 , 1 ) ) AS correct\_occ   
  FROM scarr AS r  
  LEFT OUTER JOIN sflight AS t ON t~carrid = r~carrid  
  INTO TABLE @DATA(lt\_data)  
 WHERE r~carrid IN @lr\_carrid  
 GROUP BY r~carrid.

测试结果：



## CAST

在查询语句中使用 CAST 实现字段的类型转换

* 以获取指定日期的汇率为例，汇率表中存储的日期不能直接使用，例如日期20180831 对应存储的数据为 79819168，所以我们在使用时需要转换类型

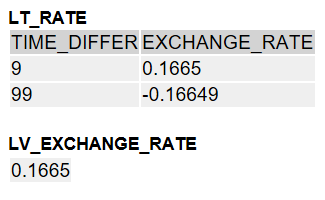
在程序中，我们可以调用 FM 来获取汇率，这里只用来测试 CAST 的使用，在CDS View中可能会被使用

例：

DATA(lv\_date) = CONV datum( '20180830' ).  
  
SELECT MIN( CAST( CAST( @lv\_date AS NUMC ) AS INT4 ) -  
            ( 99999999 - CAST( CAST( gdatu AS NUMC ) AS INT4 ) ) ) AS time\_differ,  
       ukurs AS exchange\_rate  
  FROM tcurr  
  INTO TABLE @DATA(lt\_data)  
 WHERE kurst = 'M'  
   AND fcurr = 'CNY'  
   AND tcurr = 'EUR'  
 GROUP BY gdatu, ukurs  
HAVING MIN( CAST( CAST( @lv\_date AS NUMC ) AS INT4 ) -  
            ( 99999999 - CAST( CAST( gdatu AS NUMC ) AS INT4 ) ) ) >= 0  
 ORDER BY time\_differ DESCENDING.

DATA(lv\_exchange\_rate) = VALUE #( lt\_rate[ 1 ]-exchange\_rate OPTIONAL ).

测试结果：



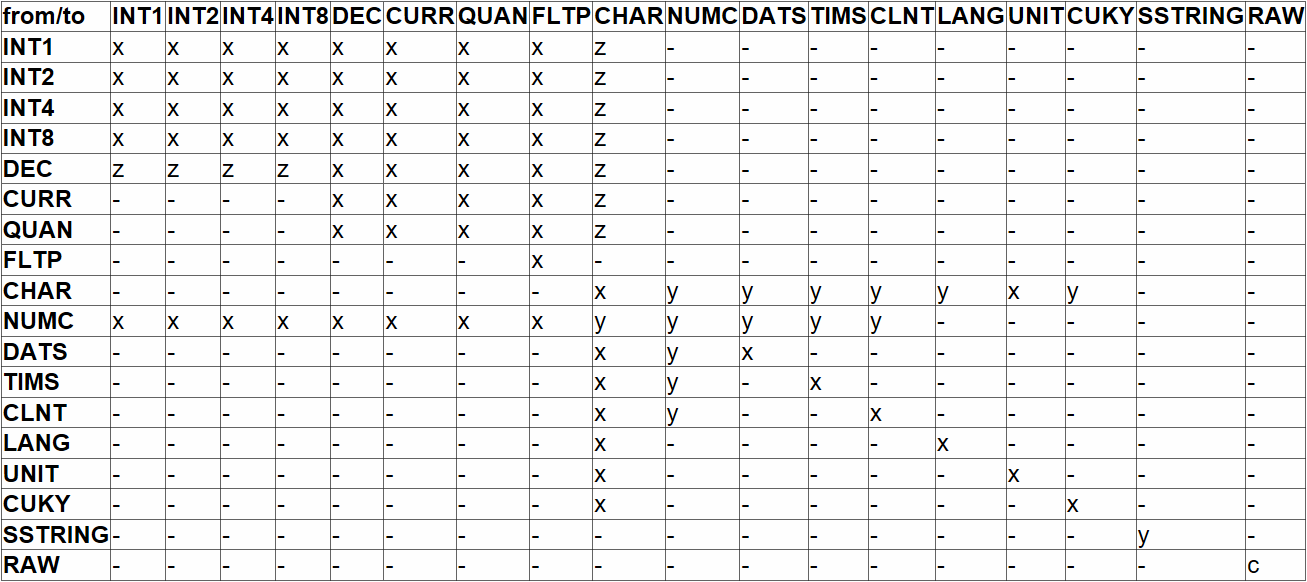
* 转换规则（ ABAP 7.53 ）如下图：

X：没有特殊限制

Y：初始长度不能小于目标长度

Z：目标长度必须足够显示源数据类型的值

C：初始长度必须等于目标长度

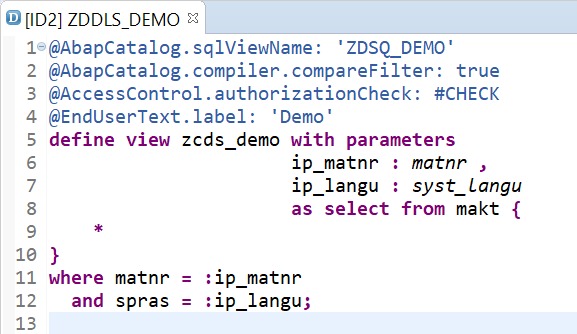


## CDS with Parameters

现支持 SELECT 传入 Parameter 从 CDS View 中获取数据

* 在查询语句中，参数必须紧接在对应的数据源后边，作为一个整体来使用，否则不能通过语法检查

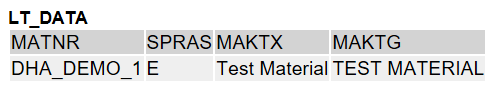
测试用CDS View如下图所示，该视图需要传入两个参数 ip\_matnr 以及 ip\_langu



例：

SELECT \*  
  FROM zcds\_demo( ip\_matnr = 'DHA\_DEMO\_1',  
                  ip\_langu = @sy-langu ) AS a   
  INTO TABLE @DATA(lt\_data).

测试结果：



## Internal Table

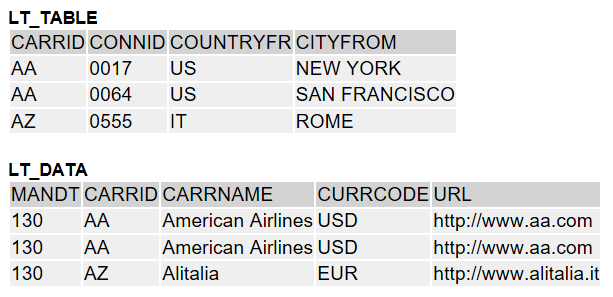
在 ABAP 7.52 后，支持将内表作为数据源使用

* 内表作为数据源使用时，需要定义别名并使用转义符@，该用法可以用来代替 FOR ALL ENTRIES IN，但FROM 语句中最多使用一个内表

例：

SELECT carrid, connid, countryfr, cityfrom  
  FROM spfli INTO TABLE @DATA(lt\_table) UP TO 3 ROWS.  
  
SELECT s~\*  
  FROM scarr AS s  
 INNER JOIN @lt\_table AS l ON l~carrid = s~carrid  
  INTO TABLE @DATA(lt\_data).

测试结果：



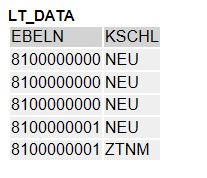
## JOIN Expression

* 在特定的应用场景中，需要使用字符长度不一致的两个字段进行关联时，可以使用相应的表达式处理，但要注意表达式的位置，一般需要放在等式左边，如下例

例：（NAST-OBJKY类型为CHAR30，EKKO-EBELN类型为CHAR10）

SELECT k~ebeln,  
       t~kschl  
  FROM nast AS t  
 INNER JOIN ekko AS k ON left( t~objky, 10 ) = k~ebeln  
  INTO TABLE @DATA(lt\_data).

测试结果：



## WHERE Condition

* 常用的条件语句，整理如下：

[ NOT ] IN：除了 SELECT-OPTION，也支持多个自定义或者通过子查询获取的值

ANY/SOME/ALL：允许将子查询获取到的结果集作为限制条件使用

[ NOT ] BETWEEN ... AND ...：使用范围条件

[ NOT ] LIKE ... [ ESCAPE ]：使用模糊查询

IS [ NOT ] NULL：判断是否有关联到相应的记录

IS [ NOT ] INITIAL：判断是否为空值

[ NOT ] EXISTS：根据指定条件到数据库表查询数据，判断查询结果是否存在

例：

SELECT i~\*  
  FROM scarr AS r  
  LEFT OUTER JOIN spfli AS i ON i~carrid = r~carrid  
 WHERE i~carrid IN ( 'AA' , 'CO' )

   AND i~carrid IN ( SELECT DISTINCT carrid FROM sflight )   
   AND i~carrid = ANY ( SELECT DISTINCT carrid FROM sflight )  
   AND i~carrid NOT BETWEEN 'BA' AND 'CA'  
   AND ( i~carrid LIKE 'A%' OR i~carrid LIKE 'C%' )  
   AND r~carrid IS NOT NULL  
   AND i~carrid IS NOT INITIAL  
   AND EXISTS ( SELECT carrid FROM sflight WHERE carrid = i~carrid )  
  INTO TABLE @DATA(lt\_data).

## OFFSET

从ABAP 7.51开始，Open SQL中引入了关键字OFFSET，可以指定查询的开始位置

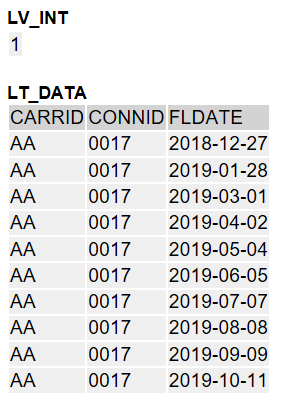
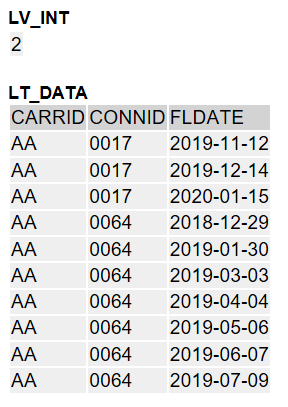
* 指定OFFSET时，结果集必须使用ORDER BY进行排序，与UP TO一起使用可以实现分页查询

例：

DATA: lv\_int TYPE i VALUE 1.  
cl\_demo\_input=>request( CHANGING field = lv\_int ).

SELECT carrid, connid, fldate  
  FROM sflight  
 ORDER BY carrid, connid, fldate  
  INTO TABLE @DATA(lt\_data)  
 UP TO 10 ROWS  
OFFSET @( ( lv\_int - 1 ) \* 10 ).

测试结果：

## Strict Mode

使用了部分新语法后，系统会以严格模式进行语法检查

* 每个版本有不同的 Strict Mode，但都会包含旧版本的内容，在语法检查时一般都会有提示性的错误信息，我们只需要按提示做修改即可；

但是有些错误信息可能看不出来是哪些语句的问题，这个时候我们就要自己去检查一下新语法的使用规则

* 部分检查示例：

SLECT LIST 中的字段需要用 ，分隔

本地对象需要使用转义符@

INTO 语句需要放在 SELECT 语句中的最后一部分

# *3 Data Processing*

## String Template

新语法引入了字符串模板，用于处理字符串连接以及格式转换

* 字符串模板在 | ... | 之间定义，主要分为两部分，固定文本和变量

其中，变量只能在 { ... } 内使用，大括号之外的所有字符均作为固定文本使用，空格始终不会被忽略，见例1

在使用变量时，可以通过控制语句来指定数据的显示格式，如例2，将日期用系统格式输出

在固定文本中，如果出现 | ，{ } 或 \ 等特殊字符时，需要使用转义符 \

例1：

DATA(lv\_vbeln) = CONV vbeln( '123456' ).  
  
lv\_vbeln = |{ lv\_vbeln ALPHA = IN }|.  
  
lv\_vbeln = | { lv\_vbeln ALPHA = IN } |.

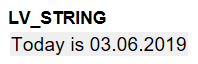
测试结果：

 注：在第二次赋值时，因为空格已经占据了第一位字符，导致最后一位字符被截断

例2：

DATA(lv\_string) = |Today is { sy-datum DATE = ENVIRONMENT }|.

测试结果：



## Format Option

在字符串模板中可以使用多种格式选项

### CURRENCY

根据指定货币 cur 调整数值的小数位，参考表 TCURX 【 CURRENCY = cur 】

### COUNTRY ★

根据指定国家 cty 格式化数据（数值/日期/时间），参考表 T005X 【 COUNTRY = cty 】

### ALPHA ★

添加/移除前导零，返回值与字段类型一致，可使用CONV转换成其他的类型进行处理。默认不做变更（RAW）【 ALPHA = [ IN | OUT | RAW ] 】

### NUMBER/DATE/TIME ★

设置数值/日期/时间显示格式

RAW：默认值，【数值】使用 . 作为小数点，且没有千分符 【日期/时间】直接输出内容

USER：使用用户设置

ENVIRONMENT：使用系统环境设置

ISO：仅用于设置【日期/时间】，使用 ISO 标准格式，yyyy-mm-dd / hh:mm:ss

例：

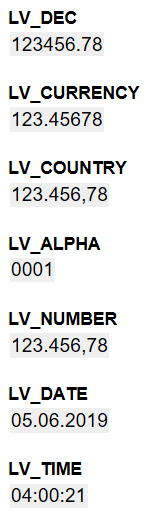
DATA(lv\_dec) = CONV zdec\_2\_demo( '123456.78' ).

DATA(lv\_currency) = |{ lv\_dec CURRENCY = 'AU5' }|.

DATA(lv\_country) = |{ lv\_dec COUNTRY = 'CN ' }|.

DATA(lv\_alpha) = |{ CONV char4( '1' ) ALPHA = IN }|.  
DATA(lv\_number) = |{ lv\_dec NUMBER = ENVIRONMENT }|.  
DATA(lv\_date) = |{ sy-datum DATE = USER }|.  
DATA(lv\_time) = |{ sy-uzeit TIME = ISO }|.

测试结果：



### TIMESTAMP

设置时间戳格式

SPACE：使用空格连接 ISO 格式下的日期和时间，yyyy-mm-dd hh:mm:ss

ISO：使用 ISO 标准格式，yyyy-mm-ddThh:mm:ss

RAW：无格式，直接输出

USER：使用用户设置

ENVIRONMENT：使用系统环境设置

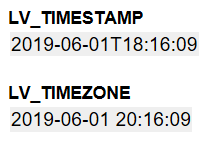
### TIMEZONE

将时间戳转换成指定时区的时间戳，参考表 TTZZ 【 TIMEZONE = tz 】

例：

DATA(lv\_stamp) = CONV timestamp( '20190601181609' ).  
DATA(lv\_timestamp) = |{ lv\_stamp TIMESTAMP = ISO }|.  
DATA(lv\_timezone) = |{ lv\_stamp TIMEZONE = 'CET' }|.

测试结果：



以下格式选项可以用于所有类型

### WIDTH

预定义字符串长度，当实际长度超出 len 时，按实际长度取值，否则按长度 len 在右侧填充空格 【 WIDTH = len 】

### ALIGN

定义字符串对齐方式，适用于使用了 WIDTH 选项且预定义长度超出实际长度的情况【 ALIGN = [ LEFT | CENTER | RIGHT ] 】

### PAD

使用指定字符串中的第一位字符填充剩余的位置，适用于使用了 WIDTH 选项且预定义长度超出实际长度的情况，默认会使用空格填充 【 PAD = c 】

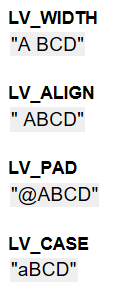
### CASE ★

将字符串进行大小写转换，默认为 RAW，该选项不会更改大小写格式【 CASE = [ RAW | LOWER | UPPER ] 】

例：

DATA(lv\_width) = |"{ 'A' WIDTH = 2 }BCD"|.  
DATA(lv\_align) = |"{ 'A' WIDTH = 2 ALIGN = RIGHT }BCD"|.  
DATA(lv\_pad) = |"{ 'A' WIDTH = 2 ALIGN = RIGHT PAD = '@' }BCD"|.  
DATA(lv\_case) = |"{ 'A' CASE = LOWER }BCD"|.

测试结果：



### SIGN

用来调整数值类型符号的显示

LEFT/RIGHT：不显示正号，将符号放置在数值左侧/右侧

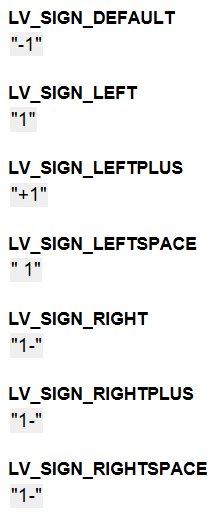
LEFTPLUS/RIGHTPLUS：显示正号，并将符号放置在数值左侧/右侧

LEFTSPACE/RIGHTSPACE：用空格代替正号显示，并将符号放置在数值左侧/右侧

例：

DATA(lv\_sign\_default) = |"{ -1 }"|.  
DATA(lv\_sign\_left) = |"{ 1 SIGN = LEFT }"|.  
DATA(lv\_sign\_leftplus) = |"{ 1 SIGN = LEFTPLUS }"|.  
DATA(lv\_sign\_leftspace) = |"{ 1 SIGN = LEFTSPACE }"|.  
DATA(lv\_sign\_right) = |"{ -1 SIGN = RIGHT }"|.  
DATA(lv\_sign\_rightplus) = |"{ -1 SIGN = RIGHTPLUS }"|.  
DATA(lv\_sign\_rightspace) = |"{ -1 SIGN = RIGHTSPACE }"|.

测试结果：



### EXPONENT

用于浮点型，设置指数【 EXPONENT = exp 】

### DECIMALS

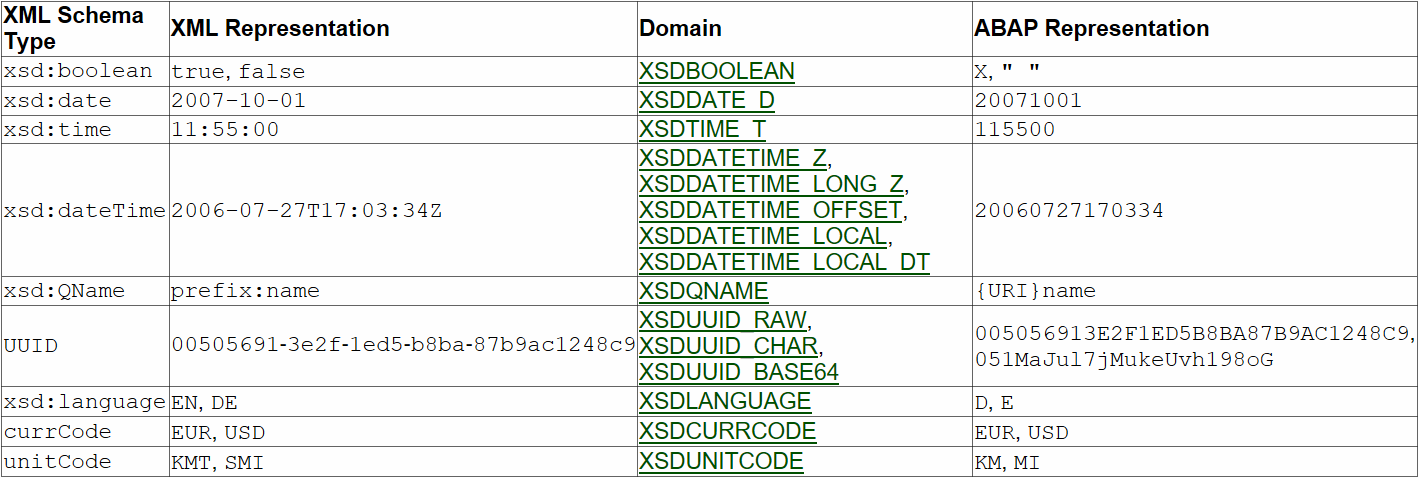
用于数值类型，设置小数位数【 DECIMALS = dec 】

### ZERO ★

用于数值类型，设置显示/隐藏零【 ZERO = [ YES | NO ] 】

### XSD

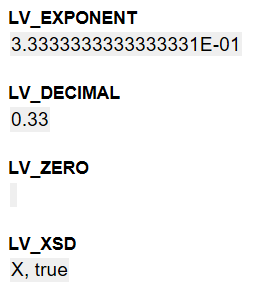
当数据参照以下 domain 时，会根据规则转换成相应的 XML 数据【 XSD = [ YES | NO ] 】



例：

DATA(lv\_exponent) = |{ CONV f( 1 / 3 ) EXPONENT = -1 }|.  
DATA(lv\_decimal) = |{ CONV f( 1 / 3 ) DECIMALS = 2 }|.  
DATA(lv\_zero) = |{ 0 ZERO = NO }|.  
DATA(lv\_xsd) = |{ CONV xsdboolean( abap\_true ) }, { CONV xsdboolean( abap\_true ) XSD = YES }|.

测试结果：



### STYLE

该选项用于设置浮点型格式

SIMPLE：默认选项，使用使用预定义格式

SIGN\_AS\_POSTFIX：在数值右侧添加负号/空格，并截断小数位尾部的 0

SCALE\_PRESERVING：范围保护，小数位尾部的 0 不会被截断

SCIENTIFIC：科学计数法，结果至少有 2 位带符号的指数，如果不指定 EXPONENT 选项，则结果的整数位始终只有一位

SCIENTIFIC\_WITH\_LEADING\_ZERO：整数位只能是 0 ，小数第一位不为 0

SCALE\_PRESERVING\_SCIENTIFIC：范围保护，类型为 decfloat16 时，指数位数为 3，类型为 decfloat34 时，指数位数为 4

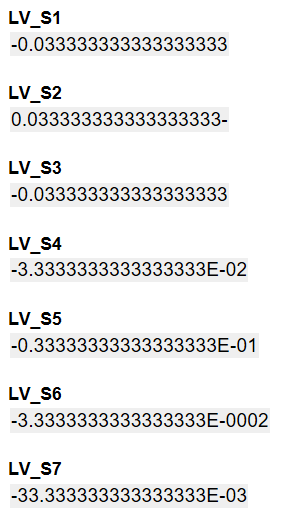
ENGINEERING：整数位范围为 1 - 999 ，指数必须是 3 的整数倍

例：

DATA(lv\_f) = CONV f( -1 / 30 ).

DATA(lv\_s1) = |{ lv\_f STYLE = SIMPLE }|.  
DATA(lv\_s2) = |{ lv\_f STYLE = SIGN\_AS\_POSTFIX }|.  
DATA(lv\_s3) = |{ lv\_f STYLE = SCALE\_PRESERVING }|.  
DATA(lv\_s4) = |{ lv\_f STYLE = SCIENTIFIC }|.  
DATA(lv\_s5) = |{ lv\_f STYLE = SCIENTIFIC\_WITH\_LEADING\_ZERO }|.  
DATA(lv\_s6) = |{ lv\_f STYLE = SCALE\_PRESERVING\_SCIENTIFIC }|.  
DATA(lv\_s7) = |{ lv\_f STYLE = ENGINEERING }|.

测试结果：



## String Functions

使用表达式处理字符串数据

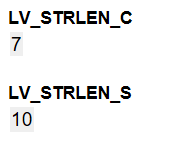
### STRLEN ★

获取字符串长度，当字符串类型为 CHAR 时，尾部空格会被忽略，当字符串类型为 STRING 时，尾部空格不会被忽略，仍会按字符被计入长度内

例：

DATA(lv\_strlen\_c) = strlen( CONV char10( |1234567    | ) ).  
DATA(lv\_strlen\_s) = strlen( CONV string( |ACDEFGH    | ) ).

测试结果：



### DISTANCE

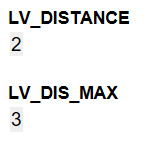
计算将 text1 改动成 text2 的最少操作次数，每次操作仅允许删除/添加/更改 1 个字符

MAX 参数用来限制最大操作次数，当最少次数大于 max 时，返回 max

例：

DATA(lv\_distance) = distance( val1 = '123ABC'  val2 = '124AC' ).  
DATA(lv\_dis\_max) = distance( val1 = '123ABC'  val2 = '124ACDE’ max = 3 ).

测试结果：



### FIND ★

搜索指定字符串并计算偏移量，没有遍历到时返回 -1

可以使用 SUB ( 固定文本 ) 或者 REGEX ( 正则表达式 ) 作为指定条件进行搜索

CASE = [ abap\_true | abap\_false ]：大小写检查，默认为 abap\_true，即区分大小写

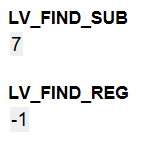
OCC = N：指定字符串在第 N 次出现，当 N 是负数时，从字符串右边开始遍历

OFF = N LEN = M：指定搜索区域，从第 N+1 为字符开始长度为 M 的范围

例：

DATA(lv\_find\_sub) = find( val = 'ABA123CAD' sub = 'a' case = ' ' occ = 3 ).  
DATA(lv\_find\_reg) = find( val = 'ABA123CAD' regex = '\d' off = 0 len = 3 ).

测试结果：



参考：

  正则表达式教程：<https://www.runoob.com/regexp/regexp-syntax.html>

  在线测试工具：<http://tool.oschina.net/regex/?optionGlobl=global>

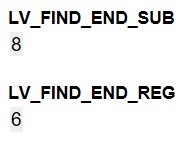
### FIND\_END

与 FIND 用法一致，但是偏移量会计算本身的长度

例：

DATA(lv\_find\_end\_sub) = find\_end( val = 'ABA123CAD' sub = 'A' occ = 3 ).  
DATA(lv\_find\_end\_reg) = find\_end( val = 'ABA123CAD' regex = '\d+' ).

测试结果：



以下两种表达式只能使用 SUB 指定文本，且始终区分大小写

### FIND\_ANY\_OF

搜索指定字符串中的任一字符并返回最小偏移量

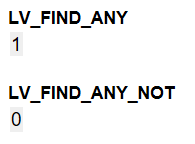
### FIND\_ANY\_NOT\_OF

搜索非指定字符串中的任意字符并返回最小偏移量

例：

DATA(lv\_find\_any) = find\_any\_of( val = 'ABA123CAD' sub = '1B' ).  
DATA(lv\_find\_any\_not) = find\_any\_not\_of( val = 'ABA123CAD' sub = '1B' ).

测试结果：



### COUNT ★

用法与 FIND 类似，但是返回值是指定字符串出现的次数，因此不能指定 OCC 参数

### COUNT\_ANY\_OF

计算指定字符串中的任一字符出现的总次数

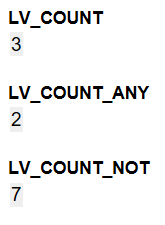
### COUNT\_ANY\_NOT\_OF

计算非指定字符串中任意字符出现的总次数

例：

DATA(lv\_count) = count( val = 'ABA123CAD' sub = 'a' case = ' ' ).  
DATA(lv\_count\_any) = count\_any\_of( val = 'ABA123CAD' sub = '1B' ).  
DATA(lv\_count\_not) = count\_any\_not\_of( val = 'ABA123CAD' sub = '1B' ).

测试结果：



### CONTAINS

用法与 FIND 类似，用于判断包含关系，返回值为布尔型数据

当指定 START/END 时，直接从左侧/右侧比较对应长度的字符，使用正则表达式时不可指定

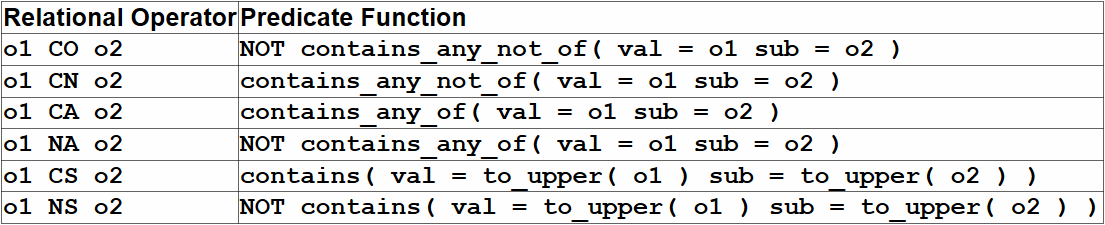
### CONTAINS\_ANY\_OF

判断是否包含指定字符串中的任一字符

### CONTAINS\_ANY\_NOT\_OF

判断是否包含非指定字符串中的任意字符

可参照部分与操作符等效的表达式用法，如下图：

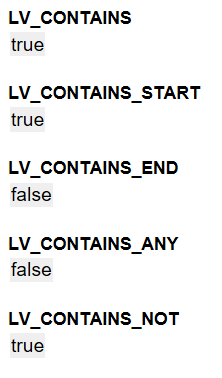


例：

DATA(lv\_contains) = xsdbool( contains( val = 'ABA123CAD' sub = 'a' case = ' ' ) ).

DATA(lv\_contains\_start) = xsdbool( contains( val = 'ABA123CAD' start = 'AB' ) ).  
DATA(lv\_contains\_end) = xsdbool( contains( val = 'ABA123CAD' end = 'BD' ) ).   
DATA(lv\_contains\_any) = xsdbool( contains\_any\_of( val = 'ABA123CAD' sub = 'E' ) ).  
DATA(lv\_contains\_not) = xsdbool( contains\_any\_not\_of( val = 'ABA123CAD' sub = '1B' ) ).

测试结果：



### XSDBOOL ★

将布尔型数据 true 转换为 abap\_true，false 转换为 abap\_false，返回值类型为 char1

### REPLACE ★

替换字符串，可以指定位置进行替换，也可以查找指定字符串并替换

WITH = new 指定用于替换的字符串

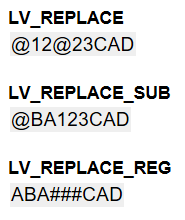
OCC = N 指定字符串第 N 次出现时进行替换，N 为 0 时表示需要全部替换

其他参数可参照 FIND 表达式

例：

DATA(lv\_replace) = replace( val = 'ABA123CAD' off = 0 len = 4 with = '@12@' ).  
DATA(lv\_replace\_sub) = replace( val = 'ABA123CAD' sub = 'a' with = '@' case = ' ' ).  
DATA(lv\_replace\_reg) = replace( val = 'ABA123CAD' regex = '\d' with = '#' occ = 0 ).

测试结果：



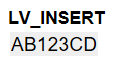
### INSERT ★

插入字符串，可以使用 OFF 指定插入的位置，默认为 0

例：

DATA(lv\_insert) = insert( val = 'ABCD' sub = '123' off = 2 ).

测试结果：



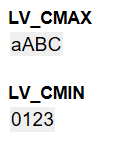
### CMAX/CMIN

返回数个字符串中的最大值/最小值，最多可以有 9 个参数，比较规则：按 0 - 9 ，A - Z，a - z 的顺序从小到大

例：

DATA(lv\_cmax) = cmax( val1 = 'aABC' val2 = 'ZABC' val3 = '0123' ).  
DATA(lv\_cmin) = cmin( val1 = 'aABC' val2 = 'ZABC' val3 = '0123' ).

测试结果：



### CONDENSE ★

压缩字符串，默认会移除头部/尾部的空格，其他部分的空格都会被压缩至 1 位

DEL = del 指定需要删除的字符，指定后，从字符串两侧开始遍历并删除字符，直到出现非指定字符

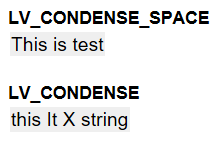
FROM = from TO = to 处理完 DEL 后，再遍历字符串，将 from 中出现的字符，替换成 to 的第一位字符

在遍历过程中，当同一个字符连续出现时，会被当成一个整体进行替换，所有字符均区分大小写

例：

DATA(lv\_condense\_space) = condense( |  This  is   test | ).  
  
DATA(lv\_condense) = condense( val  = '  XXThis ISSS X sTringXX'  
                              del  = |X |  
                              from = 'TS'   
                              to   = 'to' ).

测试结果：



### CONCAT\_LINES\_OF ★

将内表中所有的记录连接起来，通过 sep 指定分隔符

例：

DATA: lt\_data TYPE TABLE OF char10.  
lt\_data = VALUE #( ( 'ABC' ) ( '123' ) ( 'DEF' ) ).  
DATA(lv\_concat\_lines) = concat\_lines\_of( table = lt\_data sep = '@' ).

测试结果：



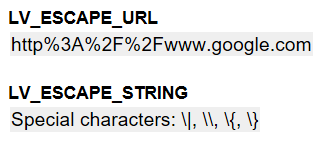
### ESCAPE

基于规则转义特定字符，FORMAT 用于指定转换规则

例：

DATA(lv\_escape\_url) = escape( val    = 'http://www.google.com'  
                              format = cl\_abap\_format=>e\_url\_full ).  
  
DATA(lv\_escape\_string) = escape( val    = 'Special characters: |, \, {, }'  
                                 format = cl\_abap\_format=>e\_string\_tpl ).

测试结果：



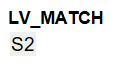
### MATCH ★

根据正则表达式匹配字符

例：

DATA(lv\_match) = match( val = 'S1S2H3H4' regex = 'S.' occ = 2 ).

测试结果：



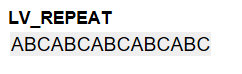
### REPEAT

循环字符串 N 次

例：

DATA(lv\_repeat) = repeat( val = 'ABC' occ = 5 ).

测试结果：



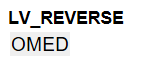
### REVERSE ★

字符串反转

例：

DATA(lv\_reverse) = reverse( 'DEMO' ).

测试结果：



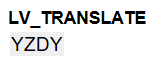
### TRANSLATE

按照指定规则替换字符，from 和 to 中的字符一一对应，没有对应关系的字符会被删除

例：

DATA(lv\_translate) = translate( val  = 'ABCDAB'  
                                from = 'ACB'  
                                to   = 'YZ' ).

测试结果：



### TO\_MIXED

处理大小写格式，首个字符大小写与 CASE 参数中的第一个字符一致，从第二个字符开始转换成小写

sep 符号（默认为 \_ ）后面的字符都会被转换成大写，min 可以指定前 N 位中的 sep 符号不起作用

### FORM\_MIXED

处理大小写格式，大小写与 CASE 参数中的第一个字符一致（默认大写），在转换前字符是大写的，会在该字符之前添加 sep 符号

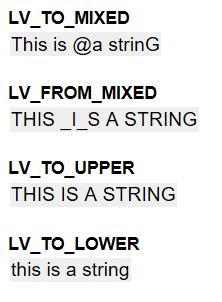
### TO\_UPPER/TO\_LOWER ★

将字符串转换成大写/小写

例：

DATA(lv\_to\_mixed) = to\_mixed( val = 'THIS is @A STRIN@G' sep = '@' case = 'X’ min = 10 ).  
DATA(lv\_from\_mixed) = from\_mixed( val = 'This IS a string' ).  
DATA(lv\_to\_upper) = to\_upper( val = 'this IS a string' ).  
DATA(lv\_to\_lower) = to\_lower( val = 'THIS IS A STRING' ).

测试结果：



### SHIFT\_LEFT/SHIFT\_RIGHT

将字符串左移/右移 N 位

指定 CIRCULAR 参数时，每次移除的字符需要被添加到另一侧

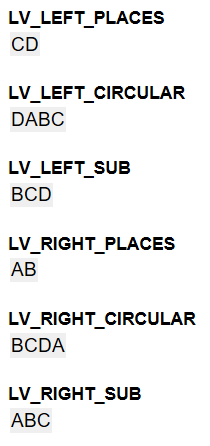
指定 SUB 参数时，如果 SUB 与字符串左侧/右侧部分字符完全匹配，则移除这些字符

例：

DATA(lv\_left\_places) = shift\_left( val = 'ABCD’ places = 2 ).  
DATA(lv\_left\_circular) = shift\_left( val  = 'ABCD' circular = 3 ).  
DATA(lv\_left\_sub) = shift\_left( val = 'ABCD' sub = 'A' ).

DATA(lv\_right\_places) = shift\_right( val = 'ABCD' places = 2 ).  
DATA(lv\_right\_circular) = shift\_right( val  = 'ABCD' circular = 3 ).  
DATA(lv\_right\_sub) = shift\_right( val = 'ABCD' sub = 'D' ).

测试结果：



以下表达式可以用来截取字符串

### SUBSTRING ★

从第 off + 1 位开始取长度为 len 的字符串，如果截取范围超出原有字符长度，会抛出异常CX\_SY\_RANGE\_OUT\_OF\_BOUNDS

### SUBSTRING\_FROM

从指定文本 sub 或是正则表达式 regex 匹配到的字符串（包含本身）开始截取，默认截至最后一位

len 指定长度，occ 指定出现次数，case 指定大小写检查

### SUBSTRING\_AFTER

从指定文本后一位开始截取，不包含本身

### SUBSTRING\_BEFORE

从第一位字符开始，截取到指定文本前一位，不包含本身

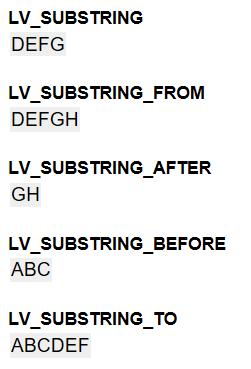
### SUBSTRING\_TO

从第一位字符开始，截取到指定文本结束，包含本身

例：

DATA(lv\_substring) = substring( val = 'ABCDEFGH' off = 3 len = 4 ).  
DATA(lv\_substring\_from) = substring\_from( val = 'ABCDEFGH' sub = 'DEF' ).  
DATA(lv\_substring\_after) = substring\_after( val = 'ABCDEFGH' sub = 'DEF' ).  
DATA(lv\_substring\_before) = substring\_before( val = 'ABCDEFGH' sub = 'DEF' ).  
DATA(lv\_substring\_to) = substring\_to( val = 'ABCDEFGH' sub = 'DEF' ).

测试结果：



### SEGMENT ★

根据分隔符获取指定位置的字符串，可以用来拆分字符串，INDEX 用来指定位置，指定位置不存在时，会抛出异常 CX\_SY\_STRG\_PAR\_VAL

通过 SEP 指定的分隔符会被当做一个整体进行操作，当分隔符连续出现时，该位置会返回空字符串；

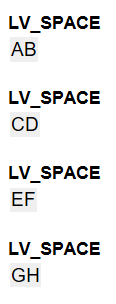
而通过 SPACE 指定的分隔符中，每个字符都会被视作单独的分隔符，且在分隔符连续出现时也不会单独返回空串

例：

DO.  
  TRY.  
      lv\_sep = segment( val   = 'AB;CD ;EF ; ;GH'  
                        index = sy-index  
                        sep = ' ;' ).  
    CATCH cx\_sy\_strg\_par\_val.  
      EXIT.  
  ENDTRY.  
ENDDO.

DO.  
  TRY.  
      lv\_space = segment( val   = 'AB  CD - EF\_GH'  
                         index = sy-index  
                         space = ' -\_' ).  
    CATCH cx\_sy\_strg\_par\_val.  
      EXIT.  
  ENDTRY.  
ENDDO.

测试结果：

## Numeric Functions

* 常见的数值表达式，整理如下：

ABS：取绝对值

SIGN( N )：N>0时返回 1；N<0时返回 -1；N=0时返回 0

CEIL：向上取整

FLOOR：向下取整

TRUNC：取整数位

FRAC：取小数位

IPOW：计算幂值，可以用来代替 \*\* 使用，避免部分数据丢失精度

NMAX/NMIN：返回参数中的最大值/最小值，参数最多传入 9 个

ROUND：计算舍入值，DEC 指定舍入位置，可以使用 MODE 指定舍入规则

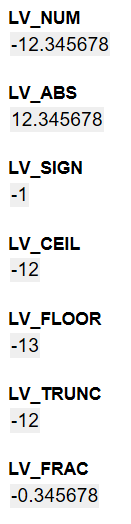
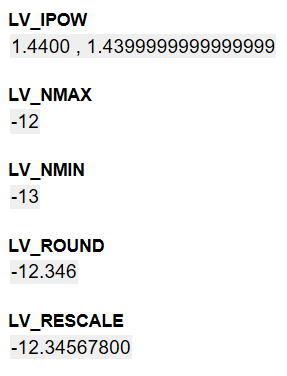
RESCALE：与 ROUND 用法一致，但是当需要保留的位数大于实际位数时，RESCALE 会在尾部填充 0，而 ROUND 不会

例：

DATA(lv\_abs) = abs( lv\_num ).  
DATA(lv\_sign) = sign( lv\_num ).  
DATA(lv\_ceil) = ceil( lv\_num ).  
DATA(lv\_floor) = floor( lv\_num ).  
DATA(lv\_trunc) = trunc( lv\_num ).  
DATA(lv\_frac) = frac( lv\_num ).

DATA(lv\_ipow) = |{ ipow( base = '1.2’ exp = 2 ) } , { ( '1.2' \*\* 2 ) }|.  
DATA(lv\_nmax) = nmax( val1 = lv\_ceil val2 = lv\_floor ).  
DATA(lv\_nmin) = nmin( val1 = lv\_ceil val2 = lv\_floor ).  
DATA(lv\_round) = round( val = lv\_num dec = 3 ).  
DATA(lv\_rescale) = rescale( val = lv\_num dec = 8 ).

测试结果：

## Internal Table Expressions

新语法中内表的读取方式

* 内表读取不再需要使用 READ TABLE，直接使用类似于数组的方式去读取

与READ TABLE读表方式类似，可以通过 INDEX 去读取指定位置的行，也可以根据条件去获取行，但无法指定BINARY SEARCH

默认情况下如果没有读到记录，会抛出异常 CX\_SY\_ITAB\_LINE\_NOT\_FOUND

### OPTIONAL ★

使用 OPTIONAL 语句时，没有读到记录也不会抛异常，而是返回空的结构

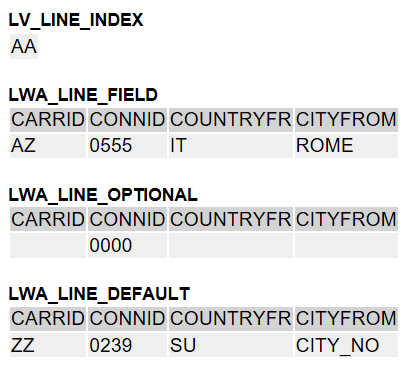
### DEFAULT ★

使用 DEFAULT 语句，在没有读到记录时，返回一个默认值，如果系统不支持这两种，则需要使用 TRY 语句来捕获异常

例：

SELECT carrid, connid, countryfr, cityfrom  
  FROM spfli INTO TABLE @DATA(lt\_table) UP TO 3 ROWS.  
  
DATA(lv\_line\_index) = lt\_table[ 1 ]-carrid.  
  
DATA(lwa\_line\_field) = lt\_table[ carrid = 'AZ'  
                                 connid = '0555' ].  
  
DATA(lwa\_line\_optional) = VALUE #( lt\_table[ 4 ] OPTIONAL ).  
  
DATA(lwa\_line\_default) = VALUE #( lt\_table[ 4 ] DEFAULT VALUE #( carrid = 'ZZ'  
                                                                 connid = '0239'  
                                                                 countryfr = 'SU'  
                                                                 cityfrom = 'CITY\_NO' ) ).

测试结果：



## Internal Table Functions

适用于内表的表达式

### LINES ★

计算内表总行数

### LINE\_EXISTS ★

判断根据特定条件能否在内表中读取到记录，返回值为布尔型数据

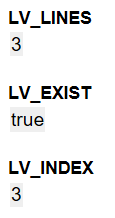
### LINE\_INDEX ★

获取内表中满足特定条件的记录所在的行数( INDEX )

例：

SELECT \* FROM spfli INTO TABLE @DATA(lt\_table) UP TO 3 ROWS.  
  
DATA(lv\_lines) = lines( lt\_table ).  
DATA(lv\_exist) = xsdbool( line\_exists( lt\_table[ carrid = 'AZ' ] ) ).  
DATA(lv\_index) = line\_index( lt\_table[ carrid = 'AZ' ] ).

测试结果：



# *4 Special Usages*

## TRY… CATCH…

在异常处理过程中使用的一些语句

### CATCH … INTO …

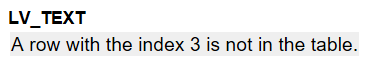
在INTO 语句中可以直接声明对象，用于查看具体的异常信息，可用的参数/方法可以在对应的异常类中查看

CATCH 语句可以用于捕获多个异常类型，可以直接在 CATCH 后添加多个异常类，也可以添加多个 CATCH 语句

例：

SELECT \* FROM spfli INTO TABLE @DATA(lt\_data) UP TO 2 ROWS.  
  
TRY .  
    DATA(lwa\_data) = lt\_data[ 3 ].  
  CATCH cx\_sy\_itab\_line\_not\_found INTO DATA(lo\_error).  
    DATA(lv\_text) = lo\_error->get\_text( ).  
ENDTRY.

测试结果：



### RETRY

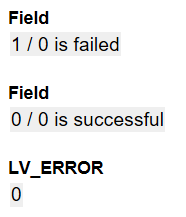
捕获到异常时，在CATCH 语句中对异常进行处理后，使用 RETRY 再次执行 TRY 语句

使用 RETRY 处理异常时，尽量多的添加限制条件，避免导致死循环

例：

DATA(lv\_num) = 1.  
TRY .  
    DATA(lv\_error) = lv\_num / 0.  
  CATCH cx\_sy\_zerodivide.  
    IF lv\_num IS NOT INITIAL.  
      CLEAR lv\_num.  
      RETRY.  
    ENDIF.  
ENDTRY.

测试结果：



### RAISE EXCEPTION

在 TRY 语句中，使用 RAISE EXCEPTION 手动抛出异常【使用 RAISE RESUMABLE EXCEPTION 抛出可恢复异常】

例：

DATA(lv\_num) = 101.

TRY.  
    IF abs( lv\_num ) > 100.  
      RAISE EXCEPTION TYPE cx\_demo\_abs\_too\_large.  
    ENDIF.

  CATCH cx\_demo\_abs\_too\_large INTO DATA(lo\_large).  
    DATA(lv\_text) = 'CATCH cx\_demo\_abs\_too\_large : ' && lo\_large->get\_text( ).   
ENDTRY.

测试结果：



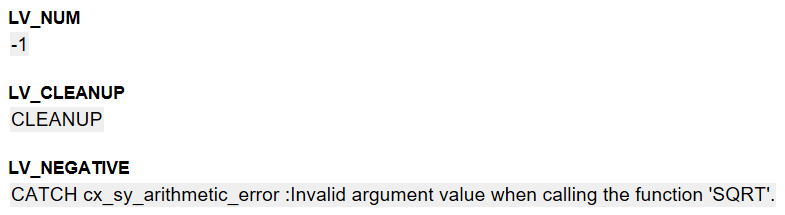
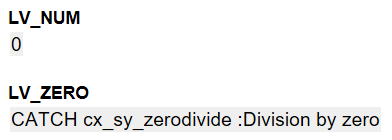
### CLEANUP

CLEANUP 语句仅在当前 TRY 语句有异常抛出且无法被当前 CATCH 语句捕获时执行，执行后再将异常传递至上层处理语句

例：

TRY.  
    TRY.  
        DATA(lv\_div) = 1 / lv\_num.  
        DATA(lv\_sqrt) = sqrt( lv\_num ).  
      CATCH cx\_sy\_zerodivide INTO DATA(lo\_zero).  
        DATA(lv\_zero) = 'CATCH cx\_sy\_zerodivide : ' && lo\_zero->get\_text( ).  
      CLEANUP.  
        DATA(lv\_cleanup) = 'CLEANUP'.  
    ENDTRY.  
  CATCH cx\_sy\_arithmetic\_error INTO DATA(lo\_negative).  
    DATA(lv\_negative) = 'CATCH cx\_sy\_arithmetic\_error : ' && lo\_negative->get\_text( ).  
ENDTRY.

测试结果：

### CATCH BEFORE UNWIND

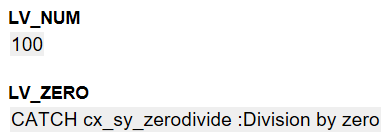
CATCH BEFORE UNWIND 用法与 CATCH 一致，但在使用时，该异常抛出部分的 CLEANUP 代码块将被推迟执行，在 CATCH BEFORE UNWIND 语句执行结束后，才会回到 CLEANUP 语句中继续执行

例：

DATA(lv\_num) = 0.  
TRY.  
    TRY.  
        DATA(lv\_div) = 1 / lv\_num.  
      CLEANUP.  
        lv\_num = lv\_num + 100.  
    ENDTRY.

  CATCH BEFORE UNWIND cx\_sy\_zerodivide INTO DATA(lo\_zero).  
    CLEAR lv\_num.  
    DATA(lv\_zero) = 'CATCH cx\_sy\_zerodivide : ' && lo\_zero->get\_text( ).  
ENDTRY.

测试结果：



### RESUME

使用 RESUME 处理可恢复异常，使该异常所在的 TRY 语句继续执行

该语句只能在 CATCH BEFORE UNWIND 下使用， 不可恢复的异常使用 RESUME 时会抛运行时错误

检查异常是否可以被恢复：IS\_RESUMABLE，该成员变量的类型为 ABAP\_BOOL

例：

TRY.  
    cl\_demo\_output=>write\_data( 1 ).  
    RAISE RESUMABLE EXCEPTION TYPE cx\_sy\_conversion\_rounding.  
    cl\_demo\_output=>write\_data( 2 ).

  CATCH BEFORE UNWIND cx\_sy\_conversion\_rounding INTO DATA(lo\_convert).   
    cl\_demo\_output=>write\_data( 3 ).  
    IF lo\_convert->is\_resumable = abap\_true.  
      RESUME.  
      cl\_demo\_output=>write\_data( 4 ).  
    ENDIF.   
ENDTRY.  
cl\_demo\_output=>display( ).

测试结果：



## Call Method

* 新语法中，可以不使用 CALL METHOD 关键字，直接用内嵌式的方法传参并获取返回值

调用方法时，可以在内部嵌套调用其他的方法，但是该方法的返回值类型与参数需要保持一致

例：

DATA(go\_grid) = NEW cl\_gui\_alv\_grid( i\_parent = go\_splitter->get\_container( row = 1

column = 1 ) ).

## Demo Output

使用 CL\_DEMO\_OUTPUT 内置的方法来展示数据，常用于测试

* 可以通过 WRITE 方法将需要显示的对象加载到 OUTPUT 界面上，再通过 DISPLAY 方法展示该界面

不同类型的对象可以选择使用对应的 WRITE 方法，如下：

WRITE / WRITE\_DATA / WRITE\_TEXT / WRITE\_HTML / WRITE\_XML / WRITE\_JSON

在 DISPLAY 的方法中也可以传入相应的对象，实际是将 WRITE\_XXX 以及 DISPLAY 方法进行了封装，如下：

DISPLAY / DISPLAY\_DATA / DISPLAY\_TEXT / DISPLAY\_HTML / DISPLAY\_XML / DISPLAY\_JSON



* 在展示数据时，可以选择添加一些辅助性的内容：
* 使用 BEGIN\_SECTION / NEXT\_SECTION / END\_SECTION 添加标题，划分章节
* 使用 LINE 添加分割线

例：

SELECT \* FROM makt INTO TABLE @DATA(lt\_table) UP TO 2 ROWS.  
DATA(lwa\_data) = lt\_table[ 1 ].  
DATA(lv\_maktx) = lt\_table[ 1 ]-maktx.  
  
cl\_demo\_output=>begin\_section( '---Display Table---' ).  
cl\_demo\_output=>write\_data( lt\_table ).  
cl\_demo\_output=>line( ).  
  
cl\_demo\_output=>next\_section( '---Display Work Area---' ).  
cl\_demo\_output=>write\_data( lwa\_data ).  
cl\_demo\_output=>line( ).  
  
cl\_demo\_output=>next\_section( '---Display Field---' ).  
cl\_demo\_output=>write\_data( lv\_maktx ).  
cl\_demo\_output=>end\_section( ).  
  
cl\_demo\_output=>display( ).

测试结果：



## Demo Input

使用 CL\_DEMO\_INPUT 内置的方法来展示数据，常用于测试

* 与OUTPUT类似，INPUT使用ADD\_FIELD加载字段到界面，并使用REQUEST调出该界面

ADD\_FIELD：默认将字段名作为文本显示，text可以指定字段文本，as\_checkbox用来设置字段以复选框格式显示

REQUEST：可以直接加载字段并显示，参数与ADD\_FIELD一致，如下右图

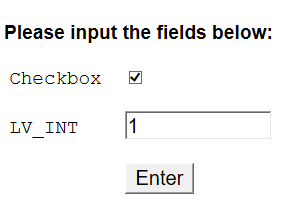
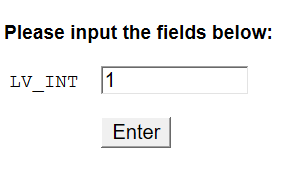
ADD\_TEXT：添加文本内容

ADD\_LINE：添加空行

例：

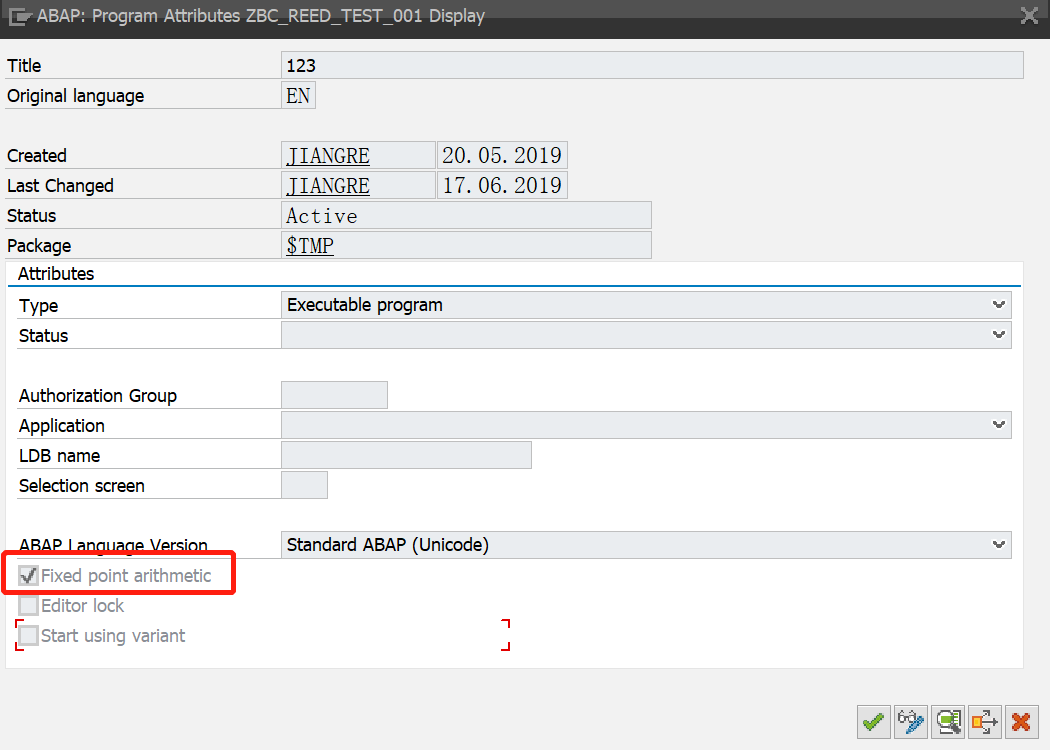
DATA: lv\_int   TYPE i VALUE 1,  
      lv\_char1 TYPE char1 VALUE 'X'.  
  
cl\_demo\_input=>add\_text( 'Please input the fields below:' ).  
cl\_demo\_input=>add\_field( EXPORTING text = 'Checkbox'  
                                    as\_checkbox = 'X'  
                           CHANGING field = lv\_char1 ).  
cl\_demo\_input=>add\_line( ).  
cl\_demo\_input=>add\_field( CHANGING field = lv\_int ).  
cl\_demo\_input=>request( ).  
  
cl\_demo\_input=>add\_text( 'Please input the fields below:' ).  
cl\_demo\_input=>request( CHANGING field = lv\_int ).

测试结果：

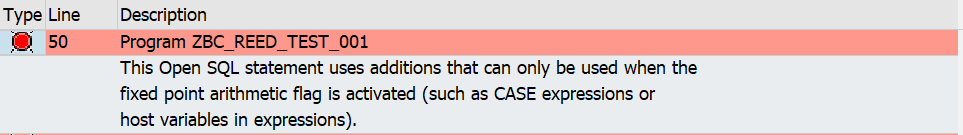
## Fixed point arithmetic

在创建程序时，需要勾选 Fixed point arithmetic，否则会影响部分新语法的使用



* 正常情况下，该选项默认为选中状态，作用是设置该程序中所有的计算都使用定点计算

在不勾选该选项时，部分新语法，例如下图中 OPEN SQL 的一些使用方式将不会被支持（尤其是在部分增强出口里）

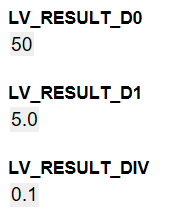
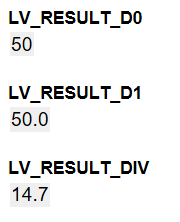


* 使用非定点运算时，无论定义了多少位小数，在赋值、比较和计算中使用压缩型数据 (ABAP/4类型P、字典类型CURR、DEC或QUAN)时，它们都将被视为整数，算术计算的中间结果也将四舍五入到整数，只有在输出该数据时，才会考虑该小数位个数

例：

DATA:lv\_result\_d0  TYPE p DECIMALS 0,  
     lv\_div        TYPE p DECIMALS 1 VALUE '6.8',  
     lv\_result\_d1  TYPE p DECIMALS 1,  
     lv\_result\_div TYPE p DECIMALS 1.  
  
lv\_result\_d0 = 100 / 2.  
lv\_result\_d1 = 100 / 2.  
lv\_result\_div = 100 / lv\_div.

测试结果：

左图是没有勾选 Fixed point arithmetic 选项时的执行结果，而右图为勾选状态下的结果

上例中，LV\_RESULT\_D0 没有小数位，所以结果为50；

LV\_RESULT\_D1有一位小数，因此结果变为 5.0；

LV\_DIV 会被视为整数 68 并用于计算，100 / 68 结果为 1.47，四舍五入后得到 1，LV\_RESULT\_DIV有一位小数，则结果为 0.1