

■ Antes (ABAP clásico) ■ Ahora (7.40 / 7.50 y posteriores)

1. Declaraciones en línea

El tipo sale de lo que se asigna: se declara donde se usa.

```
DATA lv_total TYPE netwr.
DATA ls_vbak TYPE vbak.
lv_total = ls_vbak-netwr.

DATA(lv_total) = ls_vbak-netwr.
LOOP AT lt_vbak INTO DATA(ls_vbak).
SELECT ... INTO TABLE @DATA(lt_vbak).
```

2. Tablas internas sin cabecera

```
DATA: BEGIN OF it_ped OCCURS 0,
      vbeln LIKE vbak-vbeln,
    END OF it_ped.
DESCRIBE TABLE it_ped LINES n.
REFRESH it_ped.

TYPES: BEGIN OF ty_ped,
      vbeln TYPE vbak-vbeln,
    END OF ty_ped.
DATA lt_ped TYPE SORTED TABLE OF ty_ped
      WITH UNIQUE KEY vbeln.
DATA(n) = lines( lt_ped ).
CLEAR lt_ped.
```

STANDARD: acceso por índice · SORTED: ordenada por clave · HASHED: acceso por clave muy rápido.

3. Leer una línea

```
READ TABLE lt_ped INTO ls_ped
  WITH KEY vbeln = lv_vbeln.
IF sy-subrc = 0.
  lv_x = ls_ped-netwr.
ENDIF.

lv_x = VALUE #( lt_ped[ vbeln = lv_vbeln ]-netwr
  OPTIONAL ).
IF line_exists( lt_ped[ vbeln = lv_vbeln ] ).
DATA(i) = line_index( lt_ped[ vbeln = lv_vbeln ] ).
```

Sin OPTIONAL ni DEFAULT, si no existe salta CX_SY_ITAB_LINE_NOT_FOUND.

4. Recorrer y modificar

```
LOOP AT it_ped.
  it_ped-netwr = it_ped-netwr * 2.
  MODIFY it_ped.
ENDLOOP.

LOOP AT lt_ped ASSIGNING FIELD-SYMBOL(<p>).
  <p>-netwr = <p>-netwr * 2.
ENDLOOP.
```

5. Constructores VALUE, CORRESPONDING, CONV, NEW

```
ls_lin-campo = 'MATNR'.
APPEND ls_lin TO lt_lin.
MOVE-CORRESPONDING ls_a TO ls_b.
lv_txt = lv_num.
CREATE OBJECT lo_alv.

lt_lin = VALUE #( BASE lt_lin
  ( campo = 'MATNR' ) ( campo = 'WERKS' ) ).
ls_b = CORRESPONDING #( ls_a ).
lv_txt = CONV string( lv_num ).
DATA(lo_alv) = NEW zcl_alv( ).

" Un rango
DATA(lr_bukrs) = VALUE rsdsselo_t(
  ( sign = 'I' option = 'EQ' low = '1000' ) ).
```

6. COND y SWITCH

```
CASE lv_est.
  WHEN 'A'. lv_t = 'Abierto'.
  WHEN OTHERS. lv_t = '?'.
ENDCASE.

DATA(lv_t) = SWITCH string( lv_est
  WHEN 'A' THEN `Abierto` ELSE `?` ).
DATA(lv_tramo) = COND string(
  WHEN lv_imp > 1000 THEN `Alto`
  ELSE `Bajo` ).
```

7. Plantillas de texto

```
CONCATENATE 'Pedido' lv_vbeln
  INTO lv_txt SEPARATED BY space.
WRITE lv_fecha TO lv_txt_f.

lv_txt = |Pedido { lv_vbeln ALPHA = OUT }|.
lv_txt = |{ lv_fecha DATE = USER }|.
lv_txt = |{ lv_imp NUMBER = USER }|.
lv_txt = lv_a && lv_b.
```

Formatos: ALPHA = IN/OUT, DATE = ISO/USER, TIME, NUMBER, CURRENCY = waers, WIDTH + ALIGN, SIGN = LEFT.

8. Funciones de texto

Antes	Ahora
TRANSLATE x TO UPPER CASE	x = to_upper(x)
SEARCH x FOR 'AB'	find(val = x sub = 'AB') (-1 si no está)
CONDENSE x NO-GAPS x+2(3)	condense(val = x del = ` `) substring(val = x off = 2 len = 3)
REPLACE ALL OCCURRENCES ...	replace(val = x sub = 'a' with = 'b' occ = 0)
Longitud con DESCRIBE FIELD	strlen(x), numofchar(x)

9. FOR y REDUCE

```
LOOP AT lt_vbap INTO ls_vbap.
  APPEND ls_vbap-matnr TO lt_mat.
  lv_total = lv_total + ls_vbap-netwr.
ENDLOOP.

lt_mat = VALUE #( FOR p IN lt_vbap
                  ( p-matnr ) ).
DATA(lv_total) = REDUCE netwr(
  INIT s = 0
  FOR p IN lt_vbap NEXT s = s + p-netwr ).

DATA(lt_vivas) = VALUE ty_t_vbap( FOR p IN lt_vbap
  WHERE ( abgru IS INITIAL ) ( p ) ).
" FILTER #( ... ) hace lo mismo con una clave SORTED o HASHED
```

10. Agrupar: LOOP AT ... GROUP BY

```
SORT lt_v BY kunnr.
LOOP AT lt_v INTO ls_v.
  AT END OF kunnr.
    SUM.
    WRITE: / ls_v-kunnr, ls_v-netwr.
  ENDAT.
ENDLOOP.

LOOP AT lt_v INTO DATA(ls_v)
  GROUP BY ls_v-kunnr INTO DATA(lv_kunnr).
  DATA(lv_suma) = REDUCE netwr( INIT s = 0
  FOR m IN GROUP lv_kunnr
  NEXT s = s + m-netwr ).
ENDLOOP.
```

No hace falta ordenar antes ni que el campo esté a la izquierda.

11. ABAP SQL

```
SELECT vbeln kunnr netwr
FROM vbak
INTO TABLE lt_vbak
WHERE erdat >= lv_desde.

SELECT FROM vbak
FIELDS vbeln, kunnr, netwr
WHERE erdat >= @lv_desde
INTO TABLE @DATA(lt_vbak).

SELECT FROM vbak AS k
INNER JOIN vbap AS p ON p~vbeln = k~vbeln
FIELDS k~vbeln, p~posnr, p~netwr * 2 AS doble,
CASE k~auart WHEN 'TA' THEN 'Normal'
ELSE 'Otro' END AS tipo,
COALESCE( p~abgru, ' ' ) AS rechazo
WHERE k~erdat >= @( sy-datum - 30 )
INTO TABLE @DATA(lt_pos)
UP TO 100 ROWS.
```

Variables con @, campos separados por comas, INTO al final. @(expresión) para cálculos. Escribe también con @: INSERT ztab FROM @ls.

12. Clases y excepciones

```
PERFORM calcular USING a
  CHANGING b.
CALL FUNCTION 'Z_LEER'
  EXCEPTIONS not_found = 1.
IF sy-subrc <> 0. ... ENDIF.

b = lcl_calc=>calcular( a ).
TRY.
  DATA(ls) = lo_repo->leer( lv_id ).
  CATCH zcx_no_encontrado INTO DATA(lx).
  MESSAGE lx TYPE 'E'.
ENDTRY.

RAISE EXCEPTION NEW zcx_no_encontrado( textid = ... ).
CALL METHOD lo->m EXPORTING x = 1. → lo->m( x = 1 ).
```

Las clases se pueden probar con ABAP Unit (FOR TESTING) y existen en ABAP Cloud; FORM no.

13. Obsoleto → qué usar

Antes	Ahora
MOVE a TO b, COMPUTE	b = a
ADD 1 TO n	n += 1 (7.54)
OCCURS, WITH HEADER LINE	Tabla sin cabecera + área de trabajo
REFRESH itab	CLEAR itab
LIKE	TYPE
TABLES vbak	No hace falta con INTO
WS_UPLOAD, WS_DOWNLOAD	GUI_UPLOAD, GUI_DOWNLOAD
REUSE_ALV_GRID_DISPLAY	CL_SALV_TABLE
Macros, FORM	Métodos
Bases de datos lógicas	JOIN o vistas CDS
SE30	SAT
Editor SE38 / SE80	ADT en Eclipse o VS Code

14. En ABAP Cloud

- Solo objetos **liberados** (contrato C1): se lee con vistas CDS como I_Product o I_SalesOrder, no con las tablas.
- No hay WRITE, dynpros, SUBMIT ni ficheros: la interfaz es Fiori y los servicios son OData (RAP).
- Para probar código, una clase de consola:

```
CLASS zcl_prueba DEFINITION PUBLIC FINAL CREATE PUBLIC.
  PUBLIC SECTION. INTERFACES if_oo_adt_classrun.
ENDCLASS.

CLASS zcl_prueba IMPLEMENTATION.
  METHOD if_oo_adt_classrun~main.
    " Se ejecuta con F9 en ADT
    out->write( |Hola desde ABAP Cloud| ).
  ENDMETHOD.
ENDCLASS.
```