# Manuale personale SAP/ABAP – Omar (v17 – versione completa come richiesto)

PER **YCS\_TESTPP**, **Z\_FM\_REPORTORDER4**, **YCS\_REPORT\_ORDER4**, più **errori comuni dettagliati** e **snippet spiegati**.

# INDICE

1. Fondamenta (SAP/ABAP, strumenti, struttura di un report, WRITE/FORM/variabili)
2. Accesso al Database (SELECT, JOIN, FOR ALL ENTRIES, SORT+DELETE DUPL, BINARY SEARCH)
3. Output e Reportistica (WRITE, ALV Grid, field catalog, layout, eventi)
4. Modularizzazione avanzata (INCLUDE, FORM, Function Module, Enhancement, RFC/BAPI, Smart Forms)
5. Tabelle Standard e Glossario (VBAK, VBAP, KNA1, LIPS, MAKT, AUFK, AFKO, AFVC, RESB; SD/MM/PP)
6. **Esercizi reali – consegna → soluzione → codice commentato**  
   6.1 **YCS\_TESTPP** (PP)  
   6.2 **Z\_FM\_REPORTORDER4** (FM)  
   6.3 **YCS\_REPORT\_ORDER4** (Report + ALV)
7. **Errori Comuni (molto dettagliato) + fix**
8. **Snippet Riutilizzabili (spiegati) + nuovi**
9. (Opzionale) Modulepool – guida completa (PBO/PAI, Table Control, riuso FM)
10. Extra (LUW, SAP LUW)

## 1) Fondamenta

### Cos’è SAP e ABAP

* **SAP**: ERP che gestisce vendite, acquisti, logistica, produzione, contabilità.
* **ABAP**: linguaggio per leggere/scrivere dati nelle tabelle SAP e creare logiche (report, interfacce, stampe).

### Standard vs Custom

* **Standard** = fornito da SAP (transazioni/report “di fabbrica”).
* **Custom** = sviluppi nostri (prefisso **Z** o **Y**). Esempio: Z\_FM\_REPORTORDER4, YCS\_TESTPP.

### Strumenti base

* **SE11** (Data Dictionary), **SE38** (report), **SE80** (Object Navigator), **SE93** (transaction code), **Debugger**.

### Struttura tipica di un report

REPORT z\_demo.  
INCLUDE z\_demo\_top. " tipi / variabili globali  
INCLUDE z\_demo\_sel. " parametri di selezione  
INCLUDE z\_demo\_form. " logica (FORM)

### WRITE & FORM

REPORT z\_demo\_write.  
WRITE 'Ciao Mondo'.  
WRITE / sy-datum.  
  
FORM somma USING a b CHANGING res.  
 res = a + b.  
ENDFORM.

## 2) Accesso al Database (Open SQL) – pillole

**SELECT SINGLE**:  
Legge **una sola riga** dal DB, quella che matcha la condizione WHERE.  
 Uso tipico: lettura dell’header di un ordine, di un cliente, ecc.  
 Attenzione: senza ORDER BY il sistema prende **la prima riga trovata dall’indice DB** → non è garantito che sia quella “giusta” se ci sono più record. **SELECT … INTO TABLE**:  
Legge **più righe** in un colpo solo e le mette in una tabella interna → più performante che fare SELECT … ENDSELECT.

**JOIN**:  
Query fatta dal DB → risultato già “incrociato”.  
Vantaggi: più veloce, tutto dal DB.  
Svantaggi: query più complessa, rischio duplicati se non si usa bene la chiave.

**FOR ALL ENTRIES (FAE)**:  
Fai più SELECT in cascata: prima prendi chiavi dalla tabella interna, poi chiedi al DB tutte le righe con quelle chiavi.  
 È una specie di **WHERE IN ( … )**, ma con tabella interna.  
 Regole:

* La tabella interna **non deve essere vuota** (altrimenti scarica *tutto* il DB!).
* Prima fai sempre un SORT + DELETE ADJACENT DUPLICATES.

Regola d’oro:

* Pochi record → meglio JOIN.
* Tanti record con molti campi → meglio FAE (meno duplicati, più leggibile).

**SORT + DELETE ADJACENT DUPLICATES**

* Quando hai una tabella interna con potenziali duplicati (es. dopo un JOIN o un FAE), fai **SORT** e poi **DELETE ADJACENT DUPLICATES**. Così eviti di elaborare più volte gli stessi record.  
  Necessario anche prima di usare FOR ALL ENTRIES

**READ TABLE … BINARY SEARCH**

* READ TABLE cerca una riga in tabella interna.
* Se la tabella è **ordinata per la chiave di ricerca**, puoi usare BINARY SEARCH → ricerca binaria (molto più veloce di lineare). Se non fai SORT coerente con la chiave di ricerca, il BINARY SEARCH ti dà risultati sbagliati.

## 3) Output e Reportistica – pillole

**WRITE (per prove veloci)**

* È il comando base per **stampare a video**.
* Utile solo in report di test, debug, o prove rapide.
* Nei programmi reali → quasi sempre si usa **ALV**.

Varianti comuni:

* / = a capo
* NO-ZERO = non scrive lo zero
* CURRENCY / UNIT = stampa con formato valuta/unità
* DECIMALS n = forza i decimali

**ALV Grid con REUSE\_ALV\_GRID\_DISPLAY**

L’**ALV (ABAP List Viewer)** è lo standard SAP per visualizzare i report:

* Tabella interattiva con filtri, ordinamenti, salvataggi layout.
* Serve un **field catalog** (descrizione delle colonne).
* Puoi passare un **layout** per colori, subtotal, ecc. (opzionale).

**Layout (opzionale)**

Puoi aggiungere un layout per impostare colore, zebra, key fields.

## 4) Modularizzazione avanzata – pillole

**INCLUDE (TOP / SEL / FORM)**

In ABAP i programmi grandi vengono spezzati in più file con gli **INCLUDE**.  
 Servono per separare meglio codice e logica, senza duplicare.

* TOP = dichiarazioni globali (tipi, variabili, tabelle interne)
* SEL = schermata di selezione (PARAMETERS, SELECT-OPTIONS)
* FORM = subroutine con la logica principale

**FORM (subroutine)**

* Le **FORM** sono blocchi di codice riutilizzabili *solo all’interno dello stesso programma*.
* Parametri passati USING o CHANGING.

Utile per modularizzare i report, ma poco flessibile rispetto a **Function Module**.

**Function Module (FM)**

* Moduli funzione definiti in **SE37**, raccolti in un **Function Group**.
* Hanno interfacce **tipizzate** (IMPORTING, EXPORTING, CHANGING, TABLES).
* Possono essere **riutilizzati da più programmi**.
* Possono essere **RFC-enabled** → chiamabili da altri sistemi.

Vantaggi:

* Tipizzazione forte
* Gestione errori tramite **EXCEPTIONS**
* Riutilizzabili da altri programmi (a differenza delle FORM)

**Panoramica strumenti avanzati**

**Enhancement**

Permettono di aggiungere logica a programmi standard SAP senza modificarli direttamente:

* **Customer Exit** → vecchia tecnica, basata su include
* **BAdI** → basata su classi e interfacce (più moderna, OOP)
* **Enhancement Framework** → aggiunte implicite/esplicite al codice

Serve per “personalizzare” SAP senza rompere lo standard.

**RFC (Remote Function Call)**

* FM che può essere chiamato da altri sistemi SAP o esterni.
* Tipi:
  + **sincro** (DESTINATION …)
  + **tRFC/qRFC** = transactional/queued (gestione asincrona con commit)
  + **aRFC** = asincrono parallelo

**BAPI (Business API)**

* Sono FM **standard SAP**, già RFC-enabled.
* Espongono funzioni di business (es. creare ordine, leggere anagrafica cliente).
* Sempre da chiudere con BAPI\_TRANSACTION\_COMMIT o ROLLBACK

# 6) ESERCIZI REALI — consegna → soluzione → codice commentato

## 6.1) **YCS\_TESTPP** (Produzione: AUFK/AFKO/AFVC/RESB/MAKT)

**Consegna**  
Dato un intervallo di **AUFNR** (ordini di produzione), costruire un report che mostri: - Testata e dati tecnici (AUFK/AFKO)  
- **Operazioni** (AFVC)  
- **Componenti** per operazione (RESB)  
- **Descrizioni materiali** per materiale ordine (**AFKO-PLNBEZ**) e per componenti (**RESB-MATNR**) da **MAKT** (lingua utente) Visualizzare il tutto in **ALV Grid**. Le operazioni senza componenti devono comparire lo stesso.

**Soluzione – Architettura**  
Programma modulare con include **TOP/SEL/FORM** + ALV. Usiamo **FAE** per evitare SELECT in LOOP, deduplichiamo le chiavi e facciamo **BINARY SEARCH**.

**Codice completo e commentato**

\*&---------------------------------------------------------------------\*  
\*& Report YCS\_TESTPP  
\*&---------------------------------------------------------------------\*  
\*&  
\*&---------------------------------------------------------------------\*  
REPORT ycs\_testpp.  
  
INCLUDE ycs\_testpp\_top.  
INCLUDE ycs\_testpp\_sel.  
  
DATA: lt\_output TYPE ti\_output.  
  
START-OF-SELECTION.  
  PERFORM extract\_data CHANGING lt\_output.  
  
  IF lt\_output IS INITIAL.  
    WRITE: / 'Nessun record trovato'.  
  ELSE.  
    PERFORM display\_data USING lt\_output.  
  ENDIF.  
  
  INCLUDE ycs\_testpp\_form.

**YCS\_TESTPP\_TOP** – tipi, variabili, ALV

\*&---------------------------------------------------------------------\*  
\*&  Include           YCS\_TESTPP\_TOP  
\*&---------------------------------------------------------------------\*  
" Tabelle principali  
TABLES: aufk,  
        afko,  
        afvc,  
        resb,  
        makt.  
  
" Strutture output finale  
TYPES: BEGIN OF ty\_output,  
         aufnr  TYPE aufk-aufnr,  
         werks  TYPE aufk-werks,  
         plnbez TYPE afko-plnbez,  
         zgrid  TYPE aufk-zgridvalue,  
         maktxp TYPE makt-maktx,      " descrizione prodotto  
         gamng  TYPE afko-gamng,  
         gltrs  TYPE afko-gltrs,  
         vornr  TYPE afvc-vornr,  
         ktsch  TYPE afvc-ktsch,  
         ltxa1  TYPE afvc-ltxa1,  
         matnr  TYPE resb-matnr,  
         maktxc TYPE makt-maktx,      " descrizione componente  
         bdmng  TYPE resb-bdmng,  
         lgort  TYPE resb-lgort,  
       END OF ty\_output.  
  
TYPES: ti\_output TYPE STANDARD TABLE OF ty\_output WITH EMPTY KEY.  
  
" Tabelle interne di appoggio  
DATA: lt\_aufk TYPE STANDARD TABLE OF aufk,  
      lt\_afko TYPE STANDARD TABLE OF afko,  
      lt\_afvc TYPE STANDARD TABLE OF afvc,  
      lt\_resb TYPE STANDARD TABLE OF resb,  
      lt\_makt TYPE STANDARD TABLE OF makt.

**YCS\_TESTPP\_SEL** – selezione & avvio

\*&---------------------------------------------------------------------\*  
\*&  Include           YCS\_TESTPP\_SEL  
\*&---------------------------------------------------------------------\*  
"selezione  
SELECT-OPTIONS: s\_aufnr FOR aufk-aufnr.

**YCS\_TESTPP\_FORM** – logica di estrazione, mapping e ALV

FORM extract\_data CHANGING lt\_output TYPE ti\_output.  
  
  DATA: ls\_output           TYPE ty\_output,  
        ls\_makt\_prod        TYPE makt,  
        ls\_makt\_comp        TYPE makt,  
        lv\_component\_found  TYPE abap\_bool.  
  
  " Field-symbols dichiarati UNA SOLA VOLTA  
  FIELD-SYMBOLS: <fs\_aufk> TYPE aufk,  
                 <fs\_afko> TYPE afko,  
                 <fs\_afvc> TYPE afvc,  
                 <fs\_resb> TYPE resb.  
  
  CLEAR lt\_output.  
  
  " 1) AUFK  
  SELECT \* FROM aufk  
    INTO TABLE lt\_aufk  
    WHERE aufnr IN s\_aufnr.  
  
  IF lt\_aufk IS INITIAL.  
    RETURN.  
  ENDIF.  
  
  " 2) AFKO  
  SELECT \* FROM afko  
    INTO TABLE lt\_afko  
    FOR ALL ENTRIES IN lt\_aufk  "Prendo le testate di produzione corrispondenti agli ordini trovati.  
    WHERE aufnr = lt\_aufk-aufnr.  
  
  " 3) AFVC  
  SELECT \* FROM afvc  
    INTO TABLE lt\_afvc  
    FOR ALL ENTRIES IN lt\_afko  
    WHERE aufpl = lt\_afko-aufpl. "Ogni ordine ha una sequenza di operazioni (identificata da AUFPL).  
  SORT lt\_afvc BY aufpl vornr.  
  
  " 4) RESB  
  SELECT \* FROM resb  
    INTO TABLE lt\_resb  
    FOR ALL ENTRIES IN lt\_afko  
    WHERE rsnum = lt\_afko-rsnum.  
  SORT lt\_resb BY rsnum vornr.  
  
 " 5) MAKT (descrizioni per padre + componenti)  
DATA: lt\_matnr TYPE SORTED TABLE OF matnr WITH UNIQUE KEY table\_line.  
  
LOOP AT lt\_afko ASSIGNING <fs\_afko>.  
  INSERT <fs\_afko>-plnbez INTO TABLE lt\_matnr.  
ENDLOOP.  
  
LOOP AT lt\_resb ASSIGNING <fs\_resb>.  
  IF <fs\_resb>-matnr IS NOT INITIAL.  
    INSERT <fs\_resb>-matnr INTO TABLE lt\_matnr.  
  ENDIF.  
ENDLOOP.  
  
IF lt\_matnr IS NOT INITIAL.  
  " allinea tipo riga = MAKT selezionando \*  
  SELECT \*  
    FROM makt  
    INTO TABLE lt\_makt  
    FOR ALL ENTRIES IN lt\_matnr  
    WHERE matnr = lt\_matnr-table\_line  
      AND spras = sy-langu.  
  
  SORT lt\_makt BY matnr.  " unica lingua => OK BINARY SEARCH su MATNR  
ENDIF.  
  
  
  " 6) Costruzione output  
  LOOP AT lt\_aufk ASSIGNING <fs\_aufk>.  
  
    " prendo l'AFKO dell'ordine  
    READ TABLE lt\_afko ASSIGNING <fs\_afko>  
         WITH KEY aufnr = <fs\_aufk>-aufnr.  
    IF sy-subrc <> 0.  
      CONTINUE.  
    ENDIF.  
  
    " scorro le operazioni dell'ordine (per AUFPL)  
    LOOP AT lt\_afvc ASSIGNING <fs\_afvc>  
         WHERE aufpl = <fs\_afko>-aufpl.  
  
      lv\_component\_found = abap\_false.  
  
      " scorro i componenti della specifica operazione (RSNUM+VORNR)  
      LOOP AT lt\_resb ASSIGNING <fs\_resb>  
           WHERE rsnum = <fs\_afko>-rsnum  
             AND vornr = <fs\_afvc>-vornr.  
  
        CLEAR: ls\_output, ls\_makt\_prod, ls\_makt\_comp.  
  
        " Testata  
        ls\_output-aufnr  = <fs\_aufk>-aufnr.  
        ls\_output-werks  = <fs\_aufk>-werks.  
        ls\_output-zgrid  = <fs\_aufk>-zgridvalue.  
        ls\_output-plnbez = <fs\_afko>-plnbez.  
        READ TABLE lt\_makt INTO ls\_makt\_prod  
             WITH KEY matnr = <fs\_afko>-plnbez BINARY SEARCH.  
        ls\_output-maktxp = ls\_makt\_prod-maktx.  
        ls\_output-gamng  = <fs\_afko>-gamng.  
        ls\_output-gltrs  = <fs\_afko>-gltrs.  
  
        " Operazione  
        ls\_output-vornr  = <fs\_afvc>-vornr.  
        ls\_output-ktsch  = <fs\_afvc>-ktsch.  
        ls\_output-ltxa1  = <fs\_afvc>-ltxa1.  
  
        " Componente  
        ls\_output-matnr  = <fs\_resb>-matnr.  
        ls\_output-bdmng  = <fs\_resb>-bdmng.  
        ls\_output-lgort  = <fs\_resb>-lgort.  
        READ TABLE lt\_makt INTO ls\_makt\_comp  
             WITH KEY matnr = <fs\_resb>-matnr BINARY SEARCH.  
        ls\_output-maktxc = ls\_makt\_comp-maktx.  
  
        APPEND ls\_output TO lt\_output.  
        lv\_component\_found = abap\_true.  
  
      ENDLOOP.  
  
      " Operazione senza componenti: riga con campi componente vuoti  
      IF lv\_component\_found = abap\_false.  
        CLEAR: ls\_output, ls\_makt\_prod.  
        ls\_output-aufnr  = <fs\_aufk>-aufnr.  
        ls\_output-werks  = <fs\_aufk>-werks.  
        ls\_output-zgrid  = <fs\_aufk>-zgridvalue.  
        ls\_output-plnbez = <fs\_afko>-plnbez.  
        READ TABLE lt\_makt INTO ls\_makt\_prod  
             WITH KEY matnr = <fs\_afko>-plnbez BINARY SEARCH.  
        ls\_output-maktxp = ls\_makt\_prod-maktx.  
        ls\_output-gamng  = <fs\_afko>-gamng.  
        ls\_output-gltrs  = <fs\_afko>-gltrs.  
        ls\_output-vornr  = <fs\_afvc>-vornr.  
        ls\_output-ktsch  = <fs\_afvc>-ktsch.  
        ls\_output-ltxa1  = <fs\_afvc>-ltxa1.  
  
        APPEND ls\_output TO lt\_output.  
      ENDIF.  
  
    ENDLOOP.  
  ENDLOOP.  
  
  SORT lt\_output BY aufnr vornr.  
  
ENDFORM.

## 6.2) **Z\_FM\_REPORTORDER4** (FM performante)

**Consegna**  
Creare un **Function Module** che, dato un **VBELN**, restituisce una tabella con: **VBAK** (testata), **KNA1** (cliente), **VBAP** (posizioni), **MAKT** (descrizioni materiale), **LIPS** (q.tà consegnate). **No SELECT in LOOP**. Usare **FAE**, **SORT** e **BINARY SEARCH**. Gestire eccezioni.

**Interfaccia proposta**  
- IMPORTING: IV\_VBELN TYPE VBAK-VBELN  
- TABLES (o EXPORTING): ET\_OUTPUT TYPE STANDARD TABLE OF TY\_OUT  
- EXCEPTIONS: ORDER\_NOT\_FOUND, NO\_ITEMS\_FOUND

**Codice completo e commentato**

FUNCTION z\_fm\_reportorder4.  
\*"----------------------------------------------------------------------  
\*"\*"Local Interface:  
\*"  IMPORTING  
\*"     REFERENCE(I\_VBELN) TYPE  VBELN\_VA  
\*"     REFERENCE(I\_KUNNR) TYPE  KUNAG OPTIONAL  
\*"  EXPORTING  
\*"     REFERENCE(E\_OUTPUT) TYPE  YCS\_OUTPUT\_T  
\*"----------------------------------------------------------------------  
  
  DATA: lt\_vbap   TYPE STANDARD TABLE OF vbap,  
        lt\_lips   TYPE STANDARD TABLE OF lips,  
        lt\_makt   TYPE STANDARD TABLE OF makt,  
        ls\_vbak   TYPE vbak,  
        ls\_vbap   TYPE vbap,  
        ls\_kna1   TYPE kna1,  
        ls\_lips   TYPE lips,  
        ls\_makt   TYPE makt,  
        ls\_output TYPE ycs\_output\_s,  
        lv\_lfimg  TYPE lips-lfimg,  
        lv\_kunnr  TYPE kunnr.          " <--- variabile per il cliente  
  
  CLEAR e\_output.  
  
  " 1) Testata ordine  
  SELECT SINGLE \* FROM vbak INTO @ls\_vbak WHERE vbeln = @i\_vbeln.  
  IF sy-subrc <> 0.  
    RETURN.  
  ENDIF.  
  
  " 1.1) Decidi quale KUNNR usare:  
  "     - se I\_KUNNR passato: usa quello  
  "     - altrimenti prendi da VBAK  
  IF i\_kunnr IS NOT INITIAL.  
    lv\_kunnr = i\_kunnr.  
  ELSE.  
    lv\_kunnr = ls\_vbak-kunnr.  
  ENDIF.  
  
  
  " 2) Lettura anagrafica cliente solo se ho un KUNNR valorizzato  
  CLEAR ls\_kna1.  
  IF lv\_kunnr IS NOT INITIAL.  
    SELECT SINGLE kunnr name1  
      FROM kna1  
      INTO CORRESPONDING FIELDS OF ls\_kna1  
      WHERE kunnr = lv\_kunnr.  
  ENDIF.  
  
  " 3) Posizioni ordine  
  SELECT \* FROM vbap INTO TABLE @lt\_vbap WHERE vbeln = @i\_vbeln.  
  IF lt\_vbap IS INITIAL.  
    RETURN.  
  ENDIF.  
  
  SORT lt\_vbap BY vbeln posnr.  
  DELETE ADJACENT DUPLICATES FROM lt\_vbap COMPARING vbeln posnr.  
  
  " 4) LIPS + MAKT  
  IF lt\_vbap IS NOT INITIAL.  
    SELECT \* FROM lips INTO TABLE @lt\_lips  
      FOR ALL ENTRIES IN @lt\_vbap  
      WHERE vgbel = @lt\_vbap-vbeln  
        AND vgpos = @lt\_vbap-posnr.  
    SORT lt\_lips BY vgbel vgpos.  
  
    SELECT matnr, maktx FROM makt INTO TABLE @lt\_makt  
      FOR ALL ENTRIES IN @lt\_vbap  
      WHERE matnr = @lt\_vbap-matnr  
        AND spras = @sy-langu.  
    SORT lt\_makt BY matnr.  
  ENDIF.  
  
  " 5) Composizione output  
  LOOP AT lt\_vbap INTO ls\_vbap.  
    CLEAR: lv\_lfimg, ls\_output.  
  
    " Somma LFIMG  
    READ TABLE lt\_lips TRANSPORTING NO FIELDS  
         WITH KEY vgbel = ls\_vbap-vbeln  
                  vgpos = ls\_vbap-posnr  
         BINARY SEARCH.  
    IF sy-subrc = 0.  
      DATA(lv\_idx) = sy-tabix.  
      WHILE lv\_idx <= lines( lt\_lips ).  
        READ TABLE lt\_lips INTO ls\_lips INDEX lv\_idx.  
        IF sy-subrc <> 0 OR  
           ls\_lips-vgbel <> ls\_vbap-vbeln OR  
           ls\_lips-vgpos <> ls\_vbap-posnr.  
          EXIT.  
        ENDIF.  
        lv\_lfimg = lv\_lfimg + ls\_lips-lfimg.  
        lv\_idx = lv\_idx + 1.  
      ENDWHILE.  
    ENDIF.  
  
    " Descrizione materiale  
    READ TABLE lt\_makt INTO ls\_makt  
         WITH KEY matnr = ls\_vbap-matnr  
         BINARY SEARCH.  
  
    " Output  
    ls\_output-vbeln = ls\_vbap-vbeln.  
    ls\_output-kunnr = lv\_kunnr.           " <--- uso la variabile x kunnr opt  
    ls\_output-name1 = ls\_kna1-name1.  
    ls\_output-erdat = ls\_vbak-erdat.  
    ls\_output-posnr = ls\_vbap-posnr.  
    ls\_output-matnr = ls\_vbap-matnr.  
    ls\_output-arktx = ls\_vbap-arktx.  
    ls\_output-zmeng = ls\_vbap-zmeng.  
    ls\_output-meins = ls\_vbap-meins.  
    ls\_output-lfimg = lv\_lfimg.  
    ls\_output-maktx = ls\_makt-maktx.  
  
    IF ls\_output-maktx IS INITIAL AND ls\_output-arktx IS NOT INITIAL.  
      ls\_output-maktx = ls\_output-arktx.  
    ENDIF.  
  
    APPEND ls\_output TO e\_output.  
  ENDLOOP.  
  
ENDFUNCTION.

**Perché è performante**: tutte le letture massive sono fatte in blocco (**FAE**), le tabelle sono **ordinate** e la ricerca è **binaria**. Nessun SELECT dentro LOOP.

## 6.3) **YCS\_REPORT\_ORDER4** (Report che usa il tuo FM)

**Consegna**  
Creare un report che chieda **VBELN**, chiami Z\_FM\_REPORTORDER4 e mostri i dati in **ALV Grid**.

**Codice completo e commentato**

REPORT ycs\_report\_order4.  
  
\*&---------------------------------------------------------------------\*  
\*& Report YCS\_REPORT\_ORDER4  
\*&---------------------------------------------------------------------\*  
REPORT YCS\_REPORT\_ORDER4.  
  
INCLUDE ycs\_report\_order3\_top.  
INCLUDE ycs\_report\_order3\_sel.  
  
DATA: lt\_output TYPE ti\_output.  
  
PERFORM extract\_data CHANGING lt\_output.  
  
IF lt\_output IS INITIAL.  
  WRITE: / 'Nessun record da estrarre'.  
  EXIT.  
ELSE.  
  PERFORM display\_data USING lt\_output.  
ENDIF.  
  
INCLUDE ycs\_report\_order3\_form.

**TOP:**

\*&---------------------------------------------------------------------\*  
\*&  Include           YCS\_REPORT\_ORDER3\_TOP  
\*&---------------------------------------------------------------------\*  
  
" Tabelle principali  
TABLES: vbak,  
        vbap,  
        kna1,  
        lips,  
        makt.  
  
" Struttura header ordini  
TYPES: BEGIN OF ty\_vbak\_short,  
         vbeln TYPE vbak-vbeln,  
         kunnr TYPE vbak-kunnr,  
         erdat TYPE vbak-erdat,  
       END OF ty\_vbak\_short.  
  
" Struttura posizioni ordini  
TYPES: BEGIN OF ty\_vbap\_short,  
         vbeln TYPE vbap-vbeln,  
         posnr TYPE vbap-posnr,  
         matnr TYPE vbap-matnr,  
         arktx TYPE vbap-arktx,  
         zmeng TYPE vbap-zmeng,  
       END OF ty\_vbap\_short.  
  
" Struttura clienti  
TYPES: BEGIN OF ty\_kna1\_str,  
         kunnr TYPE kna1-kunnr,  
         name1 TYPE kna1-name1,  
       END OF ty\_kna1\_str.  
  
" Riga LIPS  
TYPES: BEGIN OF ty\_lips\_row,  
         vbeln TYPE lips-vgbel,   " riferimento a documento precedente (ordine)  
         posnr TYPE lips-vgpos,  
         lfimg TYPE lips-lfimg,  
       END OF ty\_lips\_row.  
  
TYPES: BEGIN OF ty\_lips\_sum,  
         vbeln TYPE lips-vgbel,  
         posnr TYPE lips-vgpos,  
         lfimg TYPE lips-lfimg,  
       END OF ty\_lips\_sum.  
  
" Struttura output finale  
TYPES: BEGIN OF ty\_output,  
         vbeln TYPE vbak-vbeln,  
         kunnr TYPE vbak-kunnr,  
         name1 TYPE kna1-name1,  
         erdat TYPE vbak-erdat,  
         posnr TYPE vbap-posnr,  
         matnr TYPE vbap-matnr,  
         arktx TYPE vbap-arktx,  
         zmeng TYPE vbap-zmeng,  
         lfimg TYPE lips-lfimg,  
         maktx TYPE makt-maktx,  
       END OF ty\_output.  
  
TYPES: ti\_output TYPE STANDARD TABLE OF ty\_output WITH EMPTY KEY.  
  
  " Dichiarazioni locali  
  DATA: lt\_vbak   TYPE TABLE OF ty\_vbak\_short,  " Tabella ordini con VBELN+KUNNR  
        ls\_vbak   TYPE ty\_vbak\_short,           " Work area per singolo ordine  
        lt\_eoutput TYPE ycs\_output\_t,           " tabella restituita dal FM  
        ls\_eout    TYPE ycs\_output\_s,           " riga restituita dal FM  
        ls\_output  TYPE ty\_output.              " riga dell'output finale del report  
  
  DATA: it\_fieldcat TYPE slis\_t\_fieldcat\_alv,  
        ls\_fieldcat TYPE slis\_fieldcat\_alv.

**SEL:**

\*&---------------------------------------------------------------------\*  
\*&  Include           YCS\_REPORT\_ORDER3\_SEL  
\*&---------------------------------------------------------------------\*  
  
" Selezione degli ordini in base ai criteri di ricerca  
  
  
SELECT-OPTIONS: s\_vbeln FOR vbak-vbeln,  
                s\_kunnr FOR vbak-kunnr,  
                s\_erdat FOR vbak-erdat.  
  
" Gruppo di radiobutton: poi nel form extract data bisogna scrivere la logica  
PARAMETERS: sortxord TYPE c RADIOBUTTON GROUP rad1 DEFAULT 'X',  
            sortxclS  TYPE c RADIOBUTTON GROUP rad1.

**FORM :**

\*&---------------------------------------------------------------------\*  
\*&  Include           YCS\_REPORT\_ORDER3\_FORM  
\*&---------------------------------------------------------------------\*  
  
FORM display\_data USING pt\_output TYPE ti\_output.  
  
  DEFINE append\_fieldcat.  
    CLEAR ls\_fieldcat.  
    ls\_fieldcat-fieldname = &1.  
    ls\_fieldcat-seltext\_m = &2.  
    ls\_fieldcat-col\_pos   = &3.  
    APPEND ls\_fieldcat TO it\_fieldcat.  
  END-OF-DEFINITION.  
  
  " Mostro sia ARKTX sia MAKTX (nessuna normalizzazione)  
  append\_fieldcat:  
    'KUNNR' 'Codice cliente'         1,  
    'NAME1' 'Ragione sociale'        2,  
    'VBELN' 'Numero ordine'          3,  
    'POSNR' 'Numero riga'            4,  
    'MATNR' 'Codice materiale'       5,  
    'ARKTX' 'Descrizione materiale'  6,  
    'ZMENG' 'Quantità ordinata'      7,  
    'LFIMG' 'Quantità spedita'       8,  
    'MAKTX' 'Descr. materiale MAKT'  9.  
  
  CALL FUNCTION 'REUSE\_ALV\_GRID\_DISPLAY'  
    EXPORTING  
      it\_fieldcat        = it\_fieldcat  
      i\_callback\_program = sy-repid  
    TABLES  
      t\_outtab           = pt\_output  
    EXCEPTIONS  
      program\_error      = 1  
      OTHERS             = 2.  
  
  IF sy-subrc <> 0.  
    MESSAGE 'Errore nella visualizzazione ALV' TYPE 'E'.  
  ENDIF.  
  
ENDFORM.  
  
\*---------------------------------------------------------------------\*  
\* Estrazione dati (versione che chiama il FM Z\_FM\_REPORTORDER4)  
\* Nessuna gestione duplicati ((non sono adiacenti e non si cancellano)), nessuna normalizzazione descrizioni  
\*---------------------------------------------------------------------\*  
FORM extract\_data CHANGING lt\_output TYPE ti\_output.  
  
  CLEAR lt\_output.  
  
  " 1) Seleziona la lista degli ordini reali in base ai filtri di selezione  
  SELECT vbeln kunnr erdat  
    FROM vbak  
    INTO TABLE lt\_vbak  
    WHERE vbeln IN s\_vbeln  
      AND kunnr IN s\_kunnr  
      AND erdat IN s\_erdat.  
  
  IF lt\_vbak IS INITIAL.  
    " Nessun ordine trovato con i criteri selezionati  
    RETURN.  
  ENDIF.  
  
  " Rimuovi eventuali duplicati e ordina per numero d'ordine  
  SORT lt\_vbak BY vbeln.  
  DELETE ADJACENT DUPLICATES FROM lt\_vbak COMPARING vbeln.  
  
  " 2) Per ogni ordine, chiama il Function Module e accumula i risultati  
  LOOP AT lt\_vbak INTO ls\_vbak.  
    CLEAR lt\_eoutput.  
  
    " Chiamata al FM Z\_FM\_REPORTORDER4 con parametri ordine e cliente (MODIFICA)  
    CALL FUNCTION 'Z\_FM\_REPORTORDER4'  
      EXPORTING  
        i\_vbeln = ls\_vbak-vbeln    " Numero ordine  
        i\_kunnr = ls\_vbak-kunnr    " Codice cliente  
      IMPORTING  
        e\_output = lt\_eoutput  
      EXCEPTIONS  
        OTHERS   = 1.  
    IF sy-subrc <> 0.  
      " Gestione errore; qui proseguiamo col prossimo ordine  
      CONTINUE.  
    ENDIF.  
  
    IF lt\_eoutput IS NOT INITIAL.  
      " Trasferisci i campi dalla tabella restituita dal FM all'output del report  
      LOOP AT lt\_eoutput INTO ls\_eout.  
        CLEAR ls\_output.  
        MOVE-CORRESPONDING ls\_eout TO ls\_output.  
        APPEND ls\_output TO lt\_output.  
      ENDLOOP.  
    ENDIF.  
  ENDLOOP.  
"qua faccio gli ordinamenti per i casi della checkbox  
    IF sortxord = 'X'.  
    SORT lt\_output BY vbeln posnr.         " Ordina per ordine  
  ELSEIF sortxcls = 'X'.  
    SORT lt\_output BY kunnr vbeln posnr.   " Ordina per cliente -> ordine  
  ENDIF.  
  
  
ENDFORM.

# 7) ERRORI COMUNI (molto dettagliato) – con spiegazione & fix

1. **Tipo riga ≠ lista SELECT**  
   Esempio: lt\_makt TYPE TABLE OF makt ma fai SELECT matnr maktx FROM makt INTO TABLE lt\_makt.  
   **Perché NON va:** il tipo riga si aspetta **tutti** i campi di MAKT; tu ne dai solo 2 → mismatch.  
   **Fix:** crea un tipo con solo matnr/maktx oppure usa CORRESPONDING con struttura compatibile.
2. **FOR ALL ENTRIES con tabella driver vuota**  
   Se lt\_keys IS INITIAL, la SELECT legge **tutto**.  
   **Fix:** IF lt\_keys IS NOT INITIAL. SELECT ... ENDIF.
3. **BINARY SEARCH senza SORT coerente**  
   Funziona solo se la tabella è **ordinata** per **quella stessa chiave** (e ordine).  
   **Fix:** SORT lt\_tab BY k1 k2. Poi READ ... WITH KEY k1 = ... k2 = ... BINARY SEARCH.
4. **JOIN che moltiplica righe (duplicati)**  
   Un JOIN 1:N genera più righe del previsto.  
   **Fix:** capisci la cardinalità; se ti servono solo le chiavi, usa **FAE** separati o DISTINCT lato DB.
5. **SELECT dentro LOOP (N+1 query)**  
   **Fix:** estrai chiavi → usa **FAE** o **JOIN**.
6. **Dimenticare SPRAS su MAKT**  
   Senza lingua potresti leggere descrizioni sbagliate.  
   **Fix:** WHERE spras = sy-langu.
7. **INTO CORRESPONDING FIELDS OF TABLE con nomi non allineati**  
   Se i nomi campo differiscono, i campi restano vuoti.  
   **Fix:** allinea i nomi o mappa manualmente.
8. **Chiave multipla in ordine sbagliato**  
   SORT lt BY matnr werks ma cerchi werks, matnr → non funziona.  
   **Fix:** stesso ordine tra SORT e READ.
9. **Non svuotare work area**  
   Riusi gs\_out senza CLEAR → valori della riga precedente.  
   **Fix:** CLEAR gs\_out prima di ogni nuovo APPEND.
10. **Usare APPEND quando serviva COLLECT**  
    Se vuoi **aggregare per chiave**, COLLECT somma le quantità per la stessa chiave.  
    **Fix:** usa COLLECT o aggrega tu con **hashed table**.
11. **Full scan per mancanza di condizione-chiave**  
    Senza WHERE su chiavi → lentezza.  
    **Fix:** filtra sempre su campi indicizzati/chiave.
12. **Nested LOOP O(n²)**  
    LOOP in LOOP senza indici → lento.  
    **Fix:** SORT + BINARY SEARCH o **HASHED TABLE**.
13. **SELECT \* inutile**  
    Porti a casa colonne che non usi.  
    **Fix:** seleziona solo i campi necessari.
14. **Mancato controllo sy-subrc**  
    Non sai se hai trovato qualcosa.  
    **Fix:** controllalo sempre dopo READ/SELECT.

# 8) SNIPPET RIUTILIZZABILI (spiegati) – + nuovi utili

Ogni snippet dice **a cosa serve** e **cosa ottieni**.

### (8.1) Macro Field Catalog ALV

**Serve a:** creare rapidamente colonne per ALV.  
**Ottieni:** meno codice ripetuto e testi coerenti.

DEFINE m\_fcat.  
 CLEAR gs\_fieldcat.  
 gs\_fieldcat-fieldname = &1.  
 gs\_fieldcat-seltext\_m = &2.  
 APPEND gs\_fieldcat TO gt\_fieldcat.  
END-OF-DEFINITION.

### (8.2) FAE sicuro (driver non vuoto)

**Serve a:** leggere in blocco per tante chiavi.  
**Ottieni:** niente SELECT in LOOP, grandi guadagni di performance.

IF lt\_keys IS NOT INITIAL.  
 SELECT col1 col2 FROM ztable INTO TABLE @DATA(lt\_out)  
 FOR ALL ENTRIES IN @lt\_keys  
 WHERE key = @lt\_keys-key.  
 SORT lt\_out BY col1.  
ENDIF.

### (8.3) Ricerca veloce con BINARY SEARCH

**Serve a:** trovare una riga in tempo logaritmico.  
**Ottieni:** ricerche rapidissime anche con migliaia di righe.

SORT lt\_tab BY k1 k2.  
READ TABLE lt\_tab ASSIGNING FIELD-SYMBOL(<s>)  
 WITH KEY k1 = lv\_k1 k2 = lv\_k2 BINARY SEARCH.

### (8.4) Costruire RANGES per WHERE IN

**Serve a:** passare più valori a una SELECT.  
**Ottieni:** filtro pulito e riusabile.

DATA: lr\_vbeln TYPE RANGE OF vbak-vbeln.  
APPEND VALUE #( sign = 'I' option = 'EQ' low = '0000123456' ) TO lr\_vbeln.

### (8.5) Misurare performance

**Serve a:** confrontare due soluzioni con numeri reali.  
**Ottieni:** la scelta più veloce basata su dati.

GET RUN TIME FIELD DATA(t1).  
" ... codice ...  
GET RUN TIME FIELD DATA(t2).  
WRITE: / 'µs:', t2 - t1.

### (8.6) Fieldcat da struttura DDIC

**Serve a:** generare field catalog dai metadati del DDIC.  
**Ottieni:** zero errori di naming/tipo.

CALL FUNCTION 'REUSE\_ALV\_FIELDCATALOG\_MERGE'  
 EXPORTING i\_structure\_name = 'VBAK'  
 CHANGING ct\_fieldcat = gt\_fieldcat.

### (8.7) Aggregazione per chiave (senza LOOP annidati)

**Serve a:** sommare valori per chiave senza O(n²).  
**Ottieni:** codice più pulito e prestazioni migliori.

TYPES: BEGIN OF ty\_k,  
 matnr TYPE mara-matnr,  
 qty TYPE lips-lfimg,  
END OF ty\_k.  
DATA lt\_aggr TYPE SORTED TABLE OF ty\_k WITH UNIQUE KEY matnr.  
  
LOOP AT lt\_lips INTO DATA(s).  
 READ TABLE lt\_aggr ASSIGNING FIELD-SYMBOL(<a>) WITH KEY matnr = s-matnr BINARY SEARCH.  
 IF sy-subrc = 0.  
 <a>-qty = <a>-qty + s-lfimg.  
 ELSE.  
 INSERT VALUE ty\_k( matnr = s-matnr qty = s-lfimg ) INTO TABLE lt\_aggr.  
 ENDIF.  
ENDLOOP.

### (8.8) Gestione messaggi centralizzata

**Serve a:** non spargere MESSAGE ovunque.  
**Ottieni:** manutenzione semplice.

FORM msg USING p\_text TYPE string.  
 MESSAGE p\_text TYPE 'S'.  
ENDFORM.

### (8.9) Safe READ con default

**Serve a:** evitare IF annidati dopo READ.  
**Ottieni:** codice più lineare.

READ TABLE lt\_makt INTO DATA(ls\_makt) WITH KEY matnr = lv\_matnr BINARY SEARCH.  
DATA(lv\_maktx) = COND makt-maktx( WHEN sy-subrc = 0 THEN ls\_makt-maktx ELSE '' ).

### (8.10) Builder di WHERE dinamico

**Serve a:** costruire condizioni complesse senza concatenare stringhe.  
**Ottieni:** chiarezza su filtri multipli.

DATA lr\_kunnr TYPE RANGE OF vbak-kunnr.  
APPEND VALUE #( sign = 'I' option = 'BT' low = '0000001000' high = '0000001999' ) TO lr\_kunnr.  
" ...poi WHERE kunnr IN lr\_kunnr

## 9) Modulepool –

**SAPMYPP\_CSTEST COS E’**

È un programmino SAP “a schermate”. Hai **una schermata (0100)** con 3 campi:

* **Material** → GV\_MATNR (lo scrivi tu)
* **Plant** → GV\_WERKS (lo scrivi tu)
* **Description** → GV\_MAKTX (la riempiamo noi se i dati sono corretti)

Premi **ENTER** → il programma controlla i dati → se è tutto ok legge la descrizione e la mostra.

**Com’è fatto (4 “cassetti” di codice)**

Nel programma principale, in cima, ci sono 4 INCLUDE:

1. **TOP**: solo **dichiarazioni** (variabili, costanti, i 3 campi dello screen).
2. **PBO** (*Process Before Output*): codice che gira **prima** di mostrare lo screen (titolo, pulsanti, cursore).
3. **PAI** (*Process After Input*): codice che gira **dopo** che tocchi la tastiera (ENTER/BACK/RESET).
4. **F01**: le **FORM** (mini-funzioni) che fanno i controlli e leggono dal database.

Pensa ai 4 cassetti: **nomi**, **prepara schermo**, **reagisci all’utente**, **lavoro vero**.

**Lo screen 0100 (come respira)**

Nel **Layout** ci sono i 3 campi collegati alle variabili:

* GV\_MATNR (input), GV\_WERKS (input), GV\_MAKTX (solo output)

**TOP (le “cose” che il programma usa)**

* GV\_MATNR, GV\_WERKS, GV\_MAKTX → i 3 campi
* GV\_OK\_CODE → memorizza **che tasto** hai premuto (ENTER/BACK/RESET…)
* GV\_CURSOR → dice **dove rimettere il cursore** dopo un errore (es. “torna su Material”)
* Costanti: GC\_ENTER, GC\_BACK, GC\_RESE, … per leggere GV\_OK\_CODE in modo semplice

**PBO (prima di mostrare lo schermo)**

* STATUS\_0100 → mette il **titolo** e la **barra dei tasti** (PF-STATUS)
* CURSOR\_SET → se GV\_CURSOR ha un nome campo (es. 'GV\_MATNR'), il programma **sposta il cursore** lì, poi lo pulisce

Serve per l’usabilità: se sbagli Material, ti riportiamo *subito* su quel campo.

**PAI (dopo che premi ENTER/BACK/RESET)**

Ci sono due moduli:

**a) USER\_COMMAND (AT EXIT-COMMAND)**

Se premi un tasto di **uscita forte** (tipo BACK/EXIT contrassegnato come “E” in Menu Painter), il sistema **esce subito**: rilascia lock, fa LEAVE PROGRAM, ecc.  
È la “uscita di emergenza” pulita.

**b) USER\_COMMAND\_0100**

Qui avviene la magia normale:

1. **Salva il tasto premuto** in GV\_OK\_CODE (con SAVE\_OK\_CODE)
2. **CASE** su GV\_OK\_CODE:
   * **ENTER**:
     + se hai scritto un Material, lo **normalizza** (aggiunge zeri davanti)  
       → es. 123 diventa 000000000000000123
     + poi chiama **CONTROL\_DATA\_0100** (che fa i controlli veri e, solo se tutto ok, legge la descrizione)
   * **BACK** → LEAVE TO SCREEN 0 (torna indietro)
   * **RESET** → chiama RESET\_HEADER (pulisce tutti i campi) e rimette il cursore su Material

Nota importante: **qui NON leggiamo MAKT** (la descrizione). Lo fa **F01** soltanto quando i dati sono validi. Così evitiamo di mostrare una descrizione se il plant è sbagliato o il legame non c’è.

**F01 (le FORM: controlli + letture DB)**

Le FORM principali sono 3 (+1 facoltativa):

**SAVE\_OK\_CODE**

* Copia SY-UCOMM (tasto premuto) in GV\_OK\_CODE, e pulisce la variabile grezza.
* Serve per non “sporcare” il giro dopo.

**RESET\_HEADER**

* Svuota Material, Plant, Description e i flag → usata quando premi **RESET**.

**CONTROL\_DATA\_0100 (il cuore)**

Qui facciamo i **controlli** e la **lettura della descrizione**.

**Ordine:**

1. Se **non hai messo nulla** (Material e Plant vuoti) → **Description vuota** e **RETURN**.
2. Se hai messo **Material** → deve esistere in **MARA**.
   * Se **non esiste** → messaggio rosso, cursore su Material, **svuota Description**, **RETURN**.
3. Se hai messo **Plant** → deve esistere in **T001W**.
   * Se **non esiste** → messaggio rosso, cursore su Plant, **svuota Description**, **RETURN**.
4. Se hai messo **entrambi** → deve esistere il **legame** in **MARC** (Material-per-Plant).
   * Se **non esiste** → messaggio rosso (“Materiale non assegnato al plant”), cursore su Plant, **svuota Description**, **RETURN**.
5. **Solo se tutti i controlli sono passati** → leggiamo **MAKT** (descrizione):
   * lingua utente SY-LANGU, se non c’è proviamo in inglese 'E'
   * mettiamo il risultato in **GV\_MAKTX** → e lo screen la mostra.

Per capire se una SELECT ha trovato qualcosa, guardiamo **SY-SUBRC**:  
0 = trovato (OK) — <> 0 = non trovato (KO).  
In caso KO: messaggio + cursore sul campo giusto + **svuotiamo la Description** + usciamo dalla FORM.

**GET\_MATERIAL\_DESCRIPTION (opzionale)**

Fa solo la SELECT su **MAKT**. Puoi tenerla se vuoi riusare la logica in più punti.

**Le tabelle che consultiamo (chi fa cosa)**

* **MARA** → esistenza **materiale** (chiave: MATNR)
* **T001W** → esistenza **plant** (chiave: WERKS)
* **MARC** → legame **materiale–plant** (chiave: MATNR + WERKS)
* **MAKT** → **descrizione** materiale per **lingua** (chiave: MATNR + SPRAS)

MAKT **non dipende dal plant**, ma **noi la mostriamo solo se** anche il plant è valido (e il legame MARC c’è), così l’utente non vede descrizioni quando i dati non stanno insieme.

**Mini-flusso “storia”**

* Tu scrivi MATNR e/o WERKS → **ENTER**
* PAI:
  + normalizza MATNR
  + chiama CONTROL\_DATA\_0100
* F01:
  + controlla MARA → T001W → MARC
  + se qualcosa è sbagliato: **messaggio rosso**, **svuota descrizione**, **cursore sul campo sbagliato**, **stop**
  + se tutto ok: legge MAKT → mette la **Description**
* Torna al PBO → se avevi settato GV\_CURSOR, il cursore va al punto giusto

**Prove tipiche (per allenarti)**

* Vuoti → niente descrizione (ok).
* MATNR buono, WERKS vuoto → descrizione ok (perché plant non è richiesto, ma materiale sì).
* MATNR buono, WERKS inesistente → messaggio su plant, **descrizione vuota**.
* MATNR+WERKS buoni ma **non** collegati in MARC → messaggio, **descrizione vuota**.
* Tutto buono → descrizione compilata.

## 10) LUW, SAP LUW

**Cos’è una LUW (Logical Unit of Work)**

**DB LUW (livello database)**

È la “unità atomica” del DB: inizia con la prima operazione che cambia dati e **finisce con un COMMIT WORK o ROLLBACK WORK**.  
Se fai più UPDATE/INSERT e poi **COMMIT**, il DB salva tutto insieme; con **ROLLBACK** annulla tutto.

**SAP LUW (livello applicazione SAP)**

È una “macro-operazione” dell’utente che può durare **più dialog step** (più pressioni di ENTER da più schermate).  
**Alla fine di *ogni* dialog step SAP fa un commit implicito del DB** per le operazioni fatte *in quel passo*.  
Quindi, se fai UPDATE nel passo 1, a fine passo 1 è **già scritto a DB**: non puoi annullare nel passo 2.

**Perché il “bundling”?**

Per evitare salvataggi parziali tra step: **raccolgo** le modifiche e **le eseguo tutte insieme in un punto preciso** (es. quando premi “Salva”).

**Cosa fa IN UPDATE TASK in pratica?**

Non scrive subito. **Accoda** la chiamata del Function Module di update in una **coda di update**.  
La coda viene eseguita **al prossimo COMMIT WORK**:

* Per default gira in **un altro work process** (asincrono).
* Non puoi usare schermate/commit dentro l’update FM.
* Puoi renderlo sincrono con **COMMIT WORK AND WAIT** o con **SET UPDATE TASK LOCAL** (vedi sotto).

**Quando useresti SET UPDATE TASK LOCAL invece del default?**

Quando vuoi che gli update **girino nello stesso work process** (locale) e **il COMMIT WORK aspetti** che finiscano:

* **Debug** più semplice (stesso processo).
* **Errore immediato**: se l’update fallisce, fallisce il commit → **rollback atomico**.
* Utile per **update piccoli/rapidi** o in **ambiente di test**.  
  (Attenzione in produzione: può allungare la durata del commit dell’utente.)

Esempio:

SET UPDATE TASK LOCAL.      " esegui gli update FM localmente  
CALL FUNCTION 'Z\_UPD' IN UPDATE TASK EXPORTING ...  
COMMIT WORK AND WAIT.       " aspetta l'esito

**“Non si fa mai un commit esplicito in bundling FM/subroutine”, è vero?**

**Dipende da cosa intendi “dove”. Facciamola chiara e pratica:**

**A) Bundling con subroutine (scrivo io direttamente su DB):**

* **Dove fai il commit?**  
  → **Nel punto di salvataggio** (quando l’utente preme “Salva”).  
  È normale fare **un solo COMMIT WORK lì**.
* **Cosa NON fare:** commit “sparsi” prima del Salva.
* Nota: potresti anche “affidarti” al commit **implicito** di fine dialog step, ma **meglio un commit esplicito al Salva** (più chiaro/controllato).

**Riassunto:** subroutine → **SÌ** un **(solo) COMMIT WORK** al Salva.

**B) Bundling con FM IN UPDATE TASK**

* **Obbligatorio**: serve sempre un **COMMIT WORK** nel **chiamante** per far partire gli update accodati.  
  (Senza commit, la coda **non** viene eseguita.)
* **Vietato**: **mai** COMMIT/ROLLBACK **dentro** il **Function Module di update**.
* Se vuoi sincrono: **COMMIT WORK AND WAIT** oppure **SET UPDATE TASK LOCAL + COMMIT WORK [AND WAIT]**.

**Riassunto:** update FM → **SÌ** COMMIT WORK nel chiamante; **NO** commit dentro l’update FM.

**C) Call in altri sistemi (tRFC/qRFC IN BACKGROUND TASK DESTINATION …)**

* Anche qui la spedizione parte **al COMMIT WORK** nel chiamante.
* Quindi **serve** un **commit** nel punto di salvataggio.
* “Non fate **commit sparsi** durante i vari step.” ✅
* “Non fate **commit dentro gli update FM**.” ✅
* Ma **al punto di salvataggio** (quando vuoi davvero scrivere):
  + **subroutine** → fai **un** COMMIT WORK
  + **update FM** → **devi** fare COMMIT WORK (o AND WAIT) per eseguirli

Se vuoi una regola d’oro: **un solo commit, nel punto “Salva”**.  
Né prima, né dentro gli update FM, né sparsi

**Mini-esempi super sintetici**

**Subroutine (bundling in memoria)**

" Passi utente: accumulo dati in GT\_ITEMS (niente DB)  
APPEND gs\_item TO gt\_items.  
  
" Tasto Salva:  
FORM save\_all.  
  LOOP AT gt\_items INTO gs\_item.  
    MODIFY ztable FROM gs\_item.  
  ENDLOOP.  
  COMMIT WORK.   " unico commit  
ENDFORM.

**Update FM (bundling via coda di update)**

" Accodo le scritture (ancora niente DB)  
CALL FUNCTION 'Z\_UPDATE\_ITEMS' IN UPDATE TASK  
  TABLES it\_items = gt\_items.  
  
" Punto di salvataggio (obbligatorio)  
COMMIT WORK.        " oppure: COMMIT WORK AND WAIT.

**Dentro Z\_UPDATE\_ITEMS** (update FM):

FUNCTION Z\_UPDATE\_ITEMS.  
\*" TABLES IT\_ITEMS STRUCTURE ZMI\_ITEM  
  DATA ls TYPE zmi\_item.  
  LOOP AT it\_items INTO ls.  
    MODIFY ztable FROM ls.  
  ENDLOOP.  
ENDFUNCTION.        " (nessun COMMIT/ROLLBACK qui!)

Fine. Questo è il **manuale completo** come lo volevi: consegna/soluzione/codice per **YCS\_TESTPP**, **Z\_FM\_REPORTORDER4**, **YCS\_REPORT\_ORDER4**, con **errori comuni dettagliati** e **snippet spiegati**. È pronto per essere esportato in **PDF**.