

Classic ABAP → Modern ABAP (on HANA) — 52 Practical Comparisons

Notes:

- All modern examples assume ABAP 7.40 SP05+ / 7.50+ style features (inline DATA, FILTER, REDUCE, VALUE, table expressions, NEW, COND, SWITCH, CDS/SQL optimizations).
- For brevity some classic examples use representative snippets rather than full programs.
- When using table expressions that may raise exceptions (e.g., itab[key = ...]) handle exceptions in production code; examples show principal idioms.

1. Inline Declarations for SELECT COUNT

Classic ABAP

```
DATA lv_total TYPE i.  
SELECT COUNT(*) INTO lv_total FROM sflight.
```

Modern ABAP

```
SELECT COUNT(*) INTO @DATA(lv_total) FROM sflight.
```

Advantage: Eliminates pre-declaration; variable declared and used inline.

2. Inline Declarations in SELECT SINGLE

Classic ABAP

```
DATA ls_spfli TYPE spfli.  
SELECT SINGLE * FROM spfli INTO ls_spfli WHERE carrid = 'LH' AND connid = '0400'.
```

Modern ABAP

```
SELECT SINGLE * FROM spfli INTO @DATA(ls_spfli) WHERE carrid = 'LH' AND connid = '0400'.
```

Advantage: Fewer lines; scope-limited variable.

3. Table expressions for INDEX access

Classic ABAP

```
READ TABLE lt_spfli INDEX 1 INTO DATA(ls_spfli).  
WRITE ls_spfli-carrid.
```

Modern ABAP

```
WRITE: / lt_spfli[ 1 ]-carrid.
```

Advantage: Direct, readable, less temp vars.

4. Table expression for KEY access

Classic ABAP

```
READ TABLE lt_spfli INTO ls_spfli WITH KEY carrid = 'LH'.
```

Modern ABAP

```
DATA(ls_spfli) = lt_spfli[ carrid = 'LH' ].
```

Advantage: Single-line; clearer semantics (throws if missing unless handled).

5. FILTER operator

Classic ABAP

```
LOOP AT lt_spfli INTO DATA(ls).
  IF ls-countryfr = 'DE'.
    APPEND ls TO lt_de.
  ENDIF.
ENDLOOP.
```

Modern ABAP

```
DATA(lt_de) = FILTER #( lt_spfli WHERE countryfr = 'DE' ).
```

Advantage: Declarative, concise, faster on HANA-friendly structures.

6. VALUE with FOR (table comprehension)

Classic ABAP

```
LOOP AT lt_src INTO DATA(ls).
  IF ls-active = abap_true.
    MOVE-CORRESPONDING ls TO ls_new.
    APPEND ls_new TO lt_new.
  ENDIF.
ENDLOOP.
```

Modern ABAP

```
DATA(lt_new) = VALUE #( FOR ls IN lt_src WHERE ( active = 'X' ) ( CORRESPONDING #( ls )
) ).
```

Advantage: Functional, compact, expressive transformations.

7. REDUCE for aggregation

Classic ABAP

```
DATA lv_tot TYPE i.
LOOP AT lt_sflight INTO DATA(ls).
  lv_tot = lv_tot + ls-seatsmax.
ENDLOOP.
```

Modern ABAP

```
DATA(lv_tot) = REDUCE i( INIT acc = 0 FOR row IN lt_sflight NEXT acc = acc + row-seatsmax ).
```

Advantage: Functional style; avoids loop bookkeeping.

8. COND for conditional assignment

Classic ABAP

```
IF sy-subrc = 0.  
  lv_text = 'OK'.  
ELSE.  
  lv_text = 'ERR'.  
ENDIF.
```

Modern ABAP

```
DATA(lv_text) = COND string( WHEN sy-subrc = 0 THEN 'OK' ELSE 'ERR' ).
```

Advantage: Single expression, easier to scan.

9. SWITCH expression

Classic ABAP

```
IF lv_m > 90.  
  lv_grade = 'A'.  
ELSEIF lv_m > 80.  
  lv_grade = 'B'.  
ELSE.  
  lv_grade = 'C'.  
ENDIF.
```

Modern ABAP

```
DATA(lv_grade) = SWITCH #( lv_m WHEN 90 UP TO 100 THEN 'A' WHEN 80 UP TO 89  
  THEN 'B' ELSE 'C' ).
```

Advantage: Declarative and compact.

10. Inline field-symbol in LOOP AT

Classic ABAP

```
FIELD-SYMBOLS <fs> TYPE spfli.  
LOOP AT lt_spfli ASSIGNING <fs>.  
  <fs>-price = <fs>-price * 1.05.  
ENDLOOP.
```

Modern ABAP

```
LOOP AT lt_spfli ASSIGNING FIELD-SYMBOL(<fs>).  
  <fs>-price = <fs>-price * 1.05.  
ENDLOOP.
```

Advantage: Cleaner declaration in loop header.

11. Inline READ TABLE INTO DATA(...)

Classic ABAP

```
DATA ls TYPE spfli.  
READ TABLE lt_spfli INTO ls WITH KEY carrid = 'LH'.  
IF sy-subrc = 0.  
    WRITE ls-connid.  
ENDIF.
```

Modern ABAP

```
READ TABLE lt_spfli INTO DATA(ls) WITH KEY carrid = 'LH'.  
IF sy-subrc = 0.  
    WRITE ls-connid.  
ENDIF.
```

Advantage: Fewer upfront declarations.

12. VALUE for structure init

Classic ABAP

```
CLEAR: ls.  
ls-carrid = 'LH'.  
ls-connid = '0400'.
```

Modern ABAP

```
DATA(ls) = VALUE spfli( carrid = 'LH' connid = '0400' ).
```

Advantage: Single initialization expression.

13. APPEND VALUE(...) to table

Classic ABAP

```
CLEAR ls.  
ls-carrid = 'LH'.  
ls-connid = '0400'.  
APPEND ls TO lt.
```

Modern ABAP

```
APPEND VALUE #( carrid = 'LH' connid = '0400' ) TO lt.
```

Advantage: No temp variable; short and readable.

14. CORRESPONDING #() mapping

Classic ABAP

```
MOVE-CORRESPONDING src TO dest.
```

Modern ABAP

```
dest = CORRESPONDING #( src ).
```

Advantage: Expression-based mapping—useful inside VALUE / FOR contexts.

15. Inline NEW object creation

Classic ABAP

```
DATA lo TYPE REF TO lcl_demo.  
CREATE OBJECT lo.
```

Modern ABAP

```
DATA(lo) = NEW lcl_demo( ).
```

Advantage: Cleaner instantiation; supports constructor parameters inline.

16. RAISE EXCEPTION NEW

Classic ABAP

```
DATA lx TYPE REF TO zcx_my_error.  
CREATE OBJECT lx.  
RAISE EXCEPTION lx.
```

Modern ABAP

```
RAISE EXCEPTION NEW zcx_my_error( msg = 'Something wrong' ).
```

Advantage: Single-line raise with constructor params.

17. CATCH INTO DATA(...)

Classic ABAP

```
TRY.  
  CALL METHOD something.  
  CATCH zcx_error INTO lx_err.  
    WRITE lx_err->get_text( ).  
ENDTRY.
```

Modern ABAP

```
TRY.  
  CALL METHOD something.  
  CATCH zcx_error INTO DATA(lx_err).  
    WRITE lx_err->get_text( ).  
ENDTRY.
```

Advantage: Inline exception var simplifies scope.

18. SELECT JOIN (pushdown) instead of nested selects

Classic ABAP

```
SELECT * FROM spfli INTO TABLE @DATA(lt_spfli).
```

```
LOOP AT lt_spfli INTO DATA(ls).
  SELECT SINGLE carrname FROM scarr INTO ls-carrname WHERE carrid = ls-carrid.
ENDLOOP.
```

Modern ABAP

```
SELECT a~carrid, a~connid, b~carrname
  FROM spfli AS a
  INNER JOIN scarr AS b
  ON a~carrid = b~carrid
  INTO TABLE @DATA(lt_join).
```

Advantage: Fewer round-trips; DB does join work (HANA-optimized).

19. GROUP BY / Aggregation in DB

Classic ABAP

```
SELECT carrid SUM( seatsmax ) AS total_price
  FROM sflight
  INTO TABLE @DATA(lt_sum)
  GROUP BY carrid.
```

Modern ABAP

-- Same SQL; point is to prefer DB aggregation rather than looping

Advantage: Large performance gains — avoid client-side aggregation.

20. CDS view for reusable DB logic

Classic ABAP

```
SELECT carrid, SUM( seatsmax ) AS total FROM sflight GROUP BY carrid INTO TABLE
@DATA(lt).
```

Modern ABAP

```
@AbapCatalog.sqlViewName: 'ZV_SFL_SUM'
define view Z_CDS_SFL_SUM as select from sflight
{
  carrid,
  sum( seatsmax ) as total
}
group by carrid
```

Advantage: Reusable, annotated model usable from ABAP & UI layers with semantics and associations.

21. FOR ALL ENTRIES -> prefer IN / CDS / JOIN

Classic ABAP

IF lt_carr IS NOT INITIAL.

```
  SELECT * FROM spfli INTO TABLE @DATA(lt_out) FOR ALL ENTRIES IN lt_carr WHERE
```

```
carrid = lt_carr-carrid.  
ENDIF.
```

Modern ABAP

```
SELECT * FROM spfli INTO TABLE @DATA(lt_out) WHERE carrid IN @lt_carr[ * ].
```

Advantage: Avoid FOR ALL ENTRIES pitfalls (duplicates, empty table); better DB optimization.

22. STRING templates for concatenation

Classic ABAP

```
CONCATENATE 'Flight:' lv_conn ' Cost:' lv_price INTO lv_text SEPARATED BY space.
```

Modern ABAP

```
DATA(lv_text) = |Flight: { lv_conn } Cost: { lv_price }|.
```

Advantage: Readable, flexible formatting (alignment, decimals).

23. ESCAPED string templates (HTML/XML)

Classic ABAP

```
lv_html = '<b>' && lv_name && '</b>'.
```

Modern ABAP

```
DATA(lv_html) = |<b>{ lv_name }</b>|.
```

Advantage: Inline HTML/XML building without CONCATENATE.

24. VALUE for nested structures

Classic ABAP

```
ls_header-field1 = 'A'.  
APPEND ls_items TO ls_header-items.
```

Modern ABAP

```
DATA(ls) = VALUE #( field1 = 'A' items = VALUE #( ( itemfield = 1 ) ( itemfield = 2 ) ) ).
```

Advantage: Build complex literal structures in one expression.

25. FILTER / FOR chain to transform and filter in single pipeline

Classic ABAP

```
LOOP AT lt_src INTO ls.  
  IF cond.  
    ls-new = transform.  
    APPEND ls TO lt_target.  
  ENDIF.  
ENDLOOP.
```

Modern ABAP

```
DATA(lt_target) = VALUE #( FOR ls IN lt_src WHERE ( cond ) ( newfield = ls-old * 2 ) ).
```

Advantage: Declarative, easier to verify.

26. ABAP function chaining (fluent) with inline DATA

Classic ABAP

```
DATA lt = zcl_util=>get_table( ).  
CALL FUNCTION '...'
```

Modern ABAP

```
DATA(result) = zcl_util=>get_table( )->transform( )->filter( ).
```

Advantage: Readable chains; fewer temporaries (when class API supports fluent pattern).

27. Table secondary keys + USING KEY

Classic ABAP

```
READ TABLE lt_tab WITH KEY connid = '400'.
```

Modern ABAP

```
READ TABLE lt_tab WITH KEY connid = '400' USING KEY connid_idx INTO DATA(ls).
```

Advantage: Performance boost with secondary index.

28. LOOP AT ... INDEX DATA(...)

Classic ABAP

```
LOOP AT lt_tab INTO ls.  
  idx = sy-tabix.  
ENDLOOP.
```

Modern ABAP

```
LOOP AT lt_tab INTO DATA(ls) INDEX DATA(idx).  
ENDLOOP.
```

Advantage: Cleaner index access.

29. READ TABLE ... TRANSPORTING NO FIELDS (existence check)

Classic ABAP

```
READ TABLE lt_tab WITH KEY id = val TRANSPORTING NO FIELDS.  
IF sy-subrc = 0.  
  ...  
ENDIF.
```

Modern ABAP

```
IF lines( FILTER #( lt_tab WHERE id = val ) ) > 0.
```

...
ENDIF.

Advantage: Expression-friendly checks; sometimes clearer intent.

30. CORRESPONDING #() inside VALUE FOR

Classic ABAP

```
LOOP AT it_src INTO ls_src.  
  MOVE-CORRESPONDING ls_src TO ls_dest.  
  APPEND ls_dest TO it_dest.  
ENDLOOP.
```

Modern ABAP

```
DATA(it_dest) = VALUE #( FOR ls IN it_src ( CORRESPONDING #( ls ) ) ).
```

Advantage: Compact mapping across whole table in expression form.

31. Inline JSON (de)serialization with /ui2/cl_json

Classic ABAP

```
CALL METHOD /ui2/cl_json=>deserialize EXPORTING json = lv_json CHANGING data =  
gs_struct.
```

Modern ABAP

```
/ ui2/cl_json=>deserialize( EXPORTING json = lv_json CHANGING data = DATA(gs_struct) ).
```

Advantage: Inline DATA usage; concise.

32. SQL: SELECT ... INTO TABLE @DATA(...) with proper typing

Classic ABAP

```
DATA lt TYPE TABLE OF spfli.  
SELECT * FROM spfli INTO TABLE lt.
```

Modern ABAP

```
SELECT * FROM spfli INTO TABLE @DATA(lt).
```

Advantage: Single-line and local scope.

33. Use of `@` host expressions in Open SQL for ABAP variables

Classic ABAP

```
SELECT * FROM spfli WHERE carrid = lv_carrid INTO TABLE @DATA(lt).
```

Modern ABAP

```
SELECT * FROM spfli WHERE carrid = @lv_carrid INTO TABLE @DATA(lt).
```

Advantage: Makes data flow explicit and safer; required modern syntax.

34. GROUP BY + HAVING in CDS

Classic ABAP

```
SELECT carrid, SUM( seatsmax ) AS tot FROM sflight GROUP BY carrid HAVING SUM( seatsmax ) > 100 INTO TABLE @DATA(lt).
```

Modern ABAP

Create CDS with `group by` + `having` or use Open SQL with same; point is to push logic to DB and reuse via CDS.

Advantage: Reusable, annotated model; HANA optimizations.

35. Use `INTO CORRESPONDING FIELDS OF TABLE` replaced by VALUE with CORRESPONDING

Classic ABAP

```
LOOP AT it_src INTO ls_src.  
MOVE-CORRESPONDING ls_src TO ls_dest.  
APPEND ls_dest TO it_dest.  
ENDLOOP.
```

Modern ABAP

```
DATA(it_dest) = VALUE #( FOR ls IN it_src ( CORRESPONDING #( ls ) ) ).
```

Advantage: Compact mapping across whole table in expression form.

36. Use `ASSIGNING FIELD-SYMBOL(<fs>)` with WHERE in LOOP AT

Classic ABAP

```
LOOP AT lt INTO ls.  
IF ls-status = 'X'.  
ls-flag = 'Y'.  
MODIFY lt FROM ls.  
ENDIF.  
ENDLOOP.
```

Modern ABAP

```
LOOP AT lt ASSIGNED FIELD-SYMBOL(<fs>) WHERE status = 'X'.  
<fs>-flag = 'Y'.  
ENDLOOP.
```

Advantage: Direct in-place modification, no MODIFY calls.

37. Use `FILTER` + `FOR` + `VALUE` to create unique lists

Classic ABAP

```
LOOP AT lt INTO ls.  
READ TABLE lt_keys WITH KEY key = ls-field TRANSPORTING NO FIELDS.  
IF sy-subrc <> 0.
```

```
APPEND ls-field TO lt_keys.  
ENDIF.  
ENDLOOP.
```

Modern ABAP

```
DATA(lt_keys) = VALUE string_table( FOR row IN lt ( row-field ) ).  
DELETE ADJACENT DUPLICATES FROM lt_keys.
```

Advantage: Use DB distinct when possible or concise expressions client-side.

38. Use `LOOP AT ... WHERE` instead of checking inside loop

Classic ABAP

```
LOOP AT lt INTO ls.  
IF ls-type = 'A'.  
    " process  
ENDIF.  
ENDLOOP.
```

Modern ABAP

```
LOOP AT lt INTO DATA(ls) WHERE type = 'A'.  
    " process  
ENDLOOP.
```

Advantage: Cleaner and marginally faster (condition evaluated once per row by runtime).

39. Use `TRY ... CATCH INTO DATA(...)` and `RAISE EXCEPTION NEW` together

Classic ABAP

```
TRY.  
    CALL METHOD something.  
CATCH zcx_demo INTO lx.  
ENDTRY.
```

Modern ABAP

```
TRY.  
    IF error.  
        RAISE EXCEPTION NEW zcx_demo( msg = 'bad' ).  
    ENDIF.  
    CATCH zcx_demo INTO DATA(lx).  
        WRITE lx->get_text( ).  
    ENDTRY.
```

Advantage: Exception objects created and caught inline.

40. Use `COND` inside expressions for nested assignment

Classic ABAP

```
IF a = 1.
```

```
lv = 'One'.
ELSE.
lv = 'Many'.
ENDIF.
```

Modern ABAP

```
lv = COND string( WHEN a = 1 THEN 'One' ELSE 'Many' ).
```

Advantage: Embeddable inside larger expressions.

41. Table expressions + FIELD-SYMBOL for quick mutate

Classic ABAP

```
READ TABLE lt INTO ls WITH KEY id = '123'.
IF sy-subrc = 0.
ls-value = 'X'.
MODIFY lt FROM ls.
ENDIF.
```

Modern ABAP

```
ASSIGN lt[ id = '123' ] TO FIELD-SYMBOL(<row>).
IF sy-subrc = 0.
<row>-value = 'X'.
ENDIF.
```

Advantage: In-place assignment; no extra copy/modify.

42. Use `LOOP AT <itab> INTO DATA(...) FROM ... TO ... INDEX ...` (slicing)

Classic ABAP

```
LOOP AT lt INTO ls FROM 10 TO 20.
" process
ENDLOOP.
```

Modern ABAP

```
LOOP AT lt INTO DATA(ls) FROM 10 TO 20.
" process
ENDLOOP.
```

Advantage: Inline DATA; same semantics but cleaner.

43. `VALUE` for nested table-of-table initialization in parameter passing

Classic ABAP

```
DATA lt TYPE TABLE OF ty.
" fill lt
CALL METHOD some->m EXPORTING it = lt.
```

Modern ABAP

```
CALL METHOD some->m( it = VALUE ty_table( ( field = 1 ) ( field = 2 ) ) ).
```

Advantage: Literal data passed directly; good for tests/mock calls.

44. `SELECT ... ORDER BY` with `@` host var and `UP TO ... ROWS`

Classic ABAP

```
SELECT * FROM spfli INTO TABLE @DATA(lt) ORDER BY carrid ASCENDING UP TO 100 ROWS.
```

Modern ABAP

```
SELECT * FROM spfli INTO TABLE @DATA(lt) ORDER BY carrid UP TO @lv_rows ROWS.
```

Advantage: Safe, clear, and parameterized paging.

45. Use of `STRING` or `CL_ABAP_STRUCTDESCR` with `CONV` for conversions

Classic ABAP

```
lv_string = lv_number.
```

Modern ABAP

```
lv_string = CONV string( lv_number ).
```

Advantage: Explicit and safe conversions.

46. Use `LOOP AT ... WHERE ... INDEX DATA(idx)` to produce index-aware iteration

Classic ABAP

```
LOOP AT lt INTO ls.
```

```
idx = sy-tabix.
```

```
" do stuff
```

```
ENDLOOP.
```

Modern ABAP

```
LOOP AT lt INTO DATA(ls) WHERE cond INDEX DATA(idx).
```

```
" use idx and ls
```

```
ENDLOOP.
```

Advantage: Avoid using global sy-tabix; clearer locality.

47. `ABAP-ON-HANA` optimized CDS calculations (calculated fields)

Classic ABAP

```
SELECT price, qty, price*qty AS amount FROM zsales INTO TABLE @DATA(lt).
```

Modern ABAP

```
define view z_sales as select from zsales {  
    price,
```

```
qty,  
cast( price * qty as dec(15,2) ) as amount  
}
```

Advantage: Reuse calculation; DB-side computation, less repeated code.

48. Use `FOR` expression with GROUP BY inside VALUE (complex grouping)

Classic ABAP

" complex loops and maps

Modern ABAP

```
DATA(grouped) = VALUE #( FOR group_key IN DISTINCT lt[ field ] (  
    key = group_key  
    items = VALUE table_of_items( FOR row IN lt WHERE ( field = group_key ) ( row ) )  
) ).
```

Advantage: Expressive grouping without auxiliary maps.

49. `DELETE ADJACENT DUPLICATES` replaced by `SELECT DISTINCT` where possible

Classic ABAP

SORT lt BY field.

DELETE ADJACENT DUPLICATES FROM lt COMPARING field.

Modern ABAP

```
SELECT DISTINCT field INTO TABLE @DATA(lt_unique) FROM ztab.
```

Advantage: Use DB for large sets; less client CPU.

50. Use `INTERFACES` and `TRY/CATCH` for cleaner error propagation in modern APIs

Classic ABAP

```
IF NOT lo->do_something().  
  RAISE EXCEPTION TYPE zcx_failed.  
ENDIF.
```

Modern ABAP

```
TRY.  
  lo->do_something().  
CATCH zcx_failed INTO DATA(lx).  
  " handle  
ENDTRY.
```

Advantage: Exception-driven control flow; clearer contract from methods.

51. Use `ABAP SQL GROUP BY ... WITH ROLLUP` (HANA feature via CDS/SQL)

Classic ABAP

" multiple selects and aggregation loops

Modern ABAP

```
SELECT field, SUM(amount) as total FROM zsales GROUP BY field WITH ROLLUP INTO  
TABLE @DATA(lt).
```

Advantage: DB computes rollups; far simpler than multi-step aggregation in ABAP.

52. Use `LET` in CDS (or calculation view) and `ASSOCIATIONS` for readability (CDS-specific)

Classic ABAP

" repeated expressions in SELECT

Modern ABAP (CDS)

```
define view z_view as select from sflight {  
    @ObjectModel.readOnly: true  
    carrid,  
    connid,  
    @EndUserText.label: 'Full route'  
    cast( carrid || '-' || connid as abap.char(20) ) as full_route  
}
```

Advantage: Semantic model, annotations usable by UI frameworks; move logic to DB layer and declare metadata.