ABAP Internal Tables & Keys — Deep Cheat Sheet
==

Contents

1. Table Types
2. Declaration Syntax
3. Keys: Primary & Secondary (Unique / Non-Unique / Sorted / Hashed)
4. Table Operations (CRUD)
5. Reading: READ TABLE vs Table Expressions
6. Safe Access: OPTIONAL, FIELD-SYMBOLS, ASSIGN
7. Looping & Iteration Patterns
8. Modern Constructors: VALUE, FOR, COND, REDUCE
9. CORRESPONDING, MAPPING, EXCEPT
10. Ranges (SIGN, OPTION, LOW, HIGH)
11. Sorting & De-duplication
12. Performance Tips & Best Practices
13. Examples (compact)
14. Quick Reference

----
1. Table Types

STANDARD TABLE
- Flexible, unsorted; linear search (O(n)).
- Declaration: DATA lt TYPE STANDARD TABLE OF row_type WITH EMPTY KEY.

SORTED TABLE
- Always sorted by key; binary search possible; fast read by key.
- Declaration: DATA lt TYPE SORTED TABLE OF row_type WITH UNIQUE KEY key_field.

HASHED TABLE
- Unordered; very fast lookup by unique key; no index order.
- Declaration: DATA lt TYPE HASHED TABLE OF row_type WITH UNIQUE KEY key_field.

Index table: only STANDARD supports index operations (INDEX).

----
2. Declaration Syntax

Single table:
```
 DATA lt_people TYPE STANDARD TABLE OF ty_person WITH EMPTY KEY.
```

With key:
```
 DATA lt TYPE SORTED TABLE OF ty_person WITH UNIQUE KEY id.
```

Secondary key (STANDARD only):
```
 DATA lt TYPE STANDARD TABLE OF ty_row
     WITH NON-UNIQUE SORTED KEY keyname COMPONENTS field1 field2.
```

----
3. Keys: Primary & Secondary

PRIMARY key:
- Defined with WITH UNIQUE/SORTED/HASHED KEY at declaration or implicitly when using TYPES.

SECONDARY keys:
- Create indexes for fast access using USING KEY keyname in READ/LOOP.
- Types: SORTED (index maintained), NON-UNIQUE (duplicates allowed), UNIQUE (duplicates prohibited).

Components:

```
 - COMPONENTS field1 field2 defines which fields the key uses.

-------------------------
4. Table Operations (CRUD)
-------------------------
Append:
 APPEND wa TO lt.
 APPEND VALUE #( field = val ) TO lt.

Insert:
 INSERT wa INTO lt INDEX 1.
 INSERT VALUE #( ... ) INTO TABLE lt.

Modify:
 MODIFY lt FROM wa INDEX idx.
 MODIFY TABLE lt FROM wa.

Delete:
 DELETE lt WHERE field = val.
 DELETE ADJACENT DUPLICATES FROM lt COMPARING field.
 DELETE lt INDEX idx.
 DELETE lt WHERE table_line = wa.

Clear / Free:
 CLEAR lt. "keeps memory, empties
 FREE lt.  "releases memory

Clear header line:
 CLEAR wa.

-------------------------
5. Reading: READ TABLE vs Table Expressions
-------------------------
Classic:
 READ TABLE lt INTO wa WITH KEY id = 10.
 IF sy-subrc = 0. ... ENDIF.

Modern table expression:
 wa = lt[ id = 10 ].              "throws exception if not found for STANDARD without key
 wa = lt[ id = 10 ] OPTIONAL.     "safe for SORTED/HASHED UNIQUE keys in procedures (not in some class contexts)

Field-symbol assign (safe in classes):
 ASSIGN lt[ id = lv_id ] TO <fs> OPTIONAL.
 IF <fs> IS ASSIGNED. ... ENDIF.

Using KEY:
 READ TABLE lt INTO wa WITH KEY id = 10.
 READ TABLE lt INTO wa WITH KEY id = 10 USING KEY secondary_key.

Binary search:
 SORT lt BY id.
 READ TABLE lt INTO wa WITH KEY id = iv_id BINARY SEARCH.

-------------------------
6. Safe Access: OPTIONAL, FIELD-SYMBOLS, ASSIGN
-------------------------
- OPTIONAL prevents CX_SY_ITAB_LINE_NOT_FOUND for unique-key lookup (rules apply).
- In class methods prefer FIELD-SYMBOL + ASSIGN ... OPTIONAL:
   FIELD-SYMBOLS <fs> TYPE row_type.
   ASSIGN lt[ id = lv_id ] TO <fs> OPTIONAL.
   IF <fs> IS ASSIGNED. ... ENDIF.
```

- Always UNASSIGN <fs> to clear pointer.

----

7. Looping & Iteration Patterns

```
LOOP AT lt INTO DATA(ls).
 LOOP AT lt INTO DATA(ls) WHERE age > 30.
 LOOP AT lt INTO DATA(ls) FROM 1 TO 10.
 LOOP AT lt INTO DATA(ls) WHERE SOMEFIELD IN lt_range. "use ranges
```

Using KEY:

```
 LOOP AT lt USING KEY connection INTO DATA(ls_row).
   ...
 ENDLOOP.
```

Table expressions in loops:

```
 LOOP AT VALUE #( FOR <r> IN src WHERE ( cond ) ( <r> ) ) INTO DATA(row).
```

----

8. Modern Constructors: VALUE, FOR, COND, REDUCE

VALUE #() - construct tables/structures:

```
 lt = VALUE #( ( id = 1 name = 'A' ) ( id = 2 name = 'B' ) ).
```

FOR in VALUE:

```
 lt_new = VALUE #( FOR <p> IN lt WHERE ( age > 18 ) ( id = <p>-id name = <p>-name ) ).
```

COND #():

```
 ls = COND #( WHEN cond THEN val ELSE other ).
```

REDUCE:

```
 DATA(str) = REDUCE string( INIT res = '' FOR <p> IN lt NEXT res = |{ res },{ <p>-name }| ).
```

Notes: IF inside FOR without ELSE avoids blank rows.

----

9. CORRESPONDING, MAPPING, EXCEPT

Copy fields by name:

```
 target = CORRESPONDING #( source ).
```

Rename while copying:

```
 target = CORRESPONDING #( source MAPPING ( new = old ) ).
```

Exclude fields:

```
 target = CORRESPONDING #( source EXCEPT field_to_skip ).
```

Combine with FOR:

```
 lt_dto = VALUE #( FOR <r> IN lt_db ( CORRESPONDING #( <r> MAPPING ( person_id = id ) EXCEPT salary ) ) ).
```

----

10. Ranges (SIGN/OPTION/LOW/HIGH)

Define:

```
 DATA lr TYPE RANGE OF i.
 lr = VALUE #( ( sign = 'I' option = 'GT' low = 100 )
               ( sign = 'I' option = 'LT' low = 500 ) ).
```

Use in SELECT:

```
 SELECT * FROM tab WHERE field IN @lr INTO TABLE @DATA(result).
```

Options: EQ, NE, GT, LT, GE, LE, BT, NB, CP (contains pattern)

----

11. Sorting & De-duplication

```
SORT lt BY field.
DELETE ADJACENT DUPLICATES FROM lt COMPARING field.
```

Note: For delete duplicates, table must be sorted by same field(s).

----

12. Performance Tips & Best Practices

- Use HASHED/SORTED for frequent key lookups.
- Avoid STANDARD table for heavy key-based searches.
- Use FIELD-SYMBOLS to avoid copying large structures.
- Use VALUE/FOR for clean, declarative transformations.
- Prefer READ TABLE ... USING KEY for secondary keys.
- Release large tables with FREE when done.

----

13. Compact Examples

```
" Copy table rows
 it_copy = VALUE #( FOR <r> IN it_src ( <r> ) ).

" Create range from table
 SORT it_src BY id.
 DELETE ADJACENT DUPLICATES FROM it_src COMPARING id.
 it_range = VALUE #( FOR <r> IN it_src ( sign = 'I' option = 'EQ' low = <r>-id ) ).

" Safe read in class
 FIELD-SYMBOLS <fs> TYPE ty_row.
 ASSIGN it_table[ id = lv_id ] TO <fs> OPTIONAL.
 IF <fs> IS ASSIGNED. out->write( <fs>-name ). ENDIF.

" Map to DTO
 ls_dto = CORRESPONDING #( ls_db MAPPING ( person_id = id first = fname ) EXCEPT salary ).
```

----

14. Quick Reference (One-liners)

Append row: APPEND VALUE #(...) TO lt.
Construct: lt = VALUE #(...).
Filter: lt2 = VALUE #(FOR <r> IN lt WHERE (cond) (<r>)).
Map: dto = CORRESPONDING #(db MAPPING (p = id)).
Safe read: ASSIGN lt[id = val] TO <fs> OPTIONAL.

END OF CHEAT SHEET