ABAP INTERNAL TABLES & KEYS – MASTER CHEAT SHEET

1. DEFAULT KEY

If you create an internal table WITHOUT specifying a key:

DATA itab TYPE TABLE OF ty_row.

Default Key = all NON-NUMERIC fields (string, char, date, time, etc.)

Numeric fields (I, INT4, DEC, etc.) are NOT part of the default key.

Example:

TYPES: BEGIN OF ty_row,

id TYPE i,

name TYPE string,

city TYPE string,

END OF ty_row.

DATA itab TYPE TABLE OF ty_row.

Default Key = name + city (id excluded)

2. TYPES OF INTERNAL TABLES

A. STANDARD TABLE

- No automatic sorting
- Slow linear search (O(n))
- Fast append
- Default table type

Syntax:

DATA itab TYPE STANDARD TABLE OF ty WITH KEY field1 field2.

B. SORTED TABLE

- Automatically sorted by key
- Fast binary search (O(log n))
- Insert only in sorted order

Syntax:

DATA itab TYPE SORTED TABLE OF ty WITH UNIQUE KEY id.

DATA itab TYPE SORTED TABLE OF ty WITH NON-UNIQUE KEY id.

C. HASHED TABLE

- Key-based hashing
- Fastest access (O(1))

- Must have UNIQUE key
- No sorting possible

Syntax:
DATA itab TYPE HASHED TABLE OF ty WITH UNIQUE KEY id.

3. KEY TYPES & EXAMPLES

A. PRIMARY KEY (MAIN KEY)
Defined at table creation.

Example:
DATA itab TYPE STANDARD TABLE OF ty WITH NON-UNIQUE KEY id.

B. AUTO DEFAULT PRIMARY KEY
When no key is defined:
DATA itab TYPE STANDARD TABLE OF ty.
→ Default primary key used.

C. SECONDARY KEYS
Used to speed up searches or loops.

Example:
DATA itab TYPE STANDARD TABLE OF ty
WITH NON-UNIQUE SORTED KEY city COMPONENTS city.

Read via secondary key:
READ TABLE itab USING KEY city WITH KEY city = 'Berlin'.

4. KEY VARIANTS

A. UNIQUE KEY
No duplicates allowed.
DATA itab TYPE SORTED TABLE OF ty WITH UNIQUE KEY id.

B. NON-UNIQUE KEY
Duplicates allowed.
DATA itab TYPE STANDARD TABLE OF ty WITH NON-UNIQUE KEY id.

C. SORTED KEY (SECONDARY)
Allows sorted access on a STANDARD table.
DATA itab TYPE STANDARD TABLE OF ty
WITH NON-UNIQUE SORTED KEY age COMPONENTS age.

D. HASHED SECONDARY KEY

Fast key access:

```
DATA itab TYPE STANDARD TABLE OF ty
WITH UNIQUE HASHED KEY id COMPONENTS id.
```

5. FIELD-SYMBOL BASED TABLE READ

```
READ TABLE itab ASSIGNING FIELD-SYMBOL() WITH KEY id = 10.

IF IS ASSIGNED.
" record found
ENDIF.
```

6. VALUE CONSTRUCTOR

Complete table in one shot:

```
DATA itab TYPE TABLE OF ty.
itab = VALUE #(
( id = 10 name = 'A' city = 'Berlin' )
( id = 20 name = 'B' city = 'Paris' )
).
```

7. FOR EXPRESSION (Like list comprehension)

```
Output table = VALUE #( FOR line IN itab ( line-city ) ).

Filtered FOR:
VALUE #( FOR line IN itab WHERE ( id > 100 ) ( line ) ).

Let variable inside FOR:
VALUE #( FOR line IN itab LET x = strlen( line-name ) IN
IF x > 3 THEN line ).
```

8. CORRESPONDING (DTO Mapping)

```
DTO = CORRESPONDING #( db_row
MAPPING ( first = fname
last = lname
age = age )
EXCEPT salary ).
```

9. RANGE TABLE CREATION USING FOR

```
DATA it_range TYPE RANGE OF /dmo/connection_id.
```

```
it_range = VALUE #(
FOR row IN itab
( sign = 'I' option = 'EQ' low = row-connection_id )
).
```

10. DELETE OPERATIONS

A. Delete rows:

```
DELETE itab WHERE city = 'Berlin'.
```

B. Delete adjacent duplicates:

```
SORT itab BY id.
DELETE ADJACENT DUPLICATES FROM itab COMPARING id.
```

11. TABLE EXPRESSIONS

Read directly:

```
DATA row TYPE ty.
row = itab[ id = 100 ].
```

Optional:

```
row = itab[ id = 888 ] OPTIONAL.
```

If not found → row = initial.