



# Fundamentos de ABAP (2)

# Índice

## **01** Introdução

03

## **02** Elementos de Linguagem

10

**01**

# Introdução

# Objectivos do módulo

- Descrever a estrutura de um programa ABAP no seu formato mais básico
- Ser capaz de criar pequenos programas, conhecendo as sintaxes para formatação de *output*, tratamento de *strings*, elementos de controlo, ciclos e subrotinas. Utilizar estes programas para para extracção das bases de dados no seu formato mais simples
- Os formandos terão um primeiro contacto com o *Function Builder*, ficando aptos a criarem módulos de função simples bem, como o tratamento de excepções
- Ser capaz de utilizar as técnicas de programação de tabelas internas e de bases de dados/Open SQL
- Ser capazes de identificar tabelas internas com ou sem *headerline*, ciclos de leitura e instruções para extracção de dados, ordenação e funções diversas

# Estrutura de um programa (1)

## ***Cabeçalho:***

```
REPORT <nome> [NO STANDARD PAGE HEADING]
              [LINE-SIZE <nr_colunas>]
              [LINE-COUNT <nr_linhas>]
              [MESSAGE-ID <classe_mensagens>]
```

## ***Parte declarativa:***

CONSTANTS

TYPES

DATA

TABLES

TYPE-POOLS

## ***Parâmetros de selecção:***

PARAMETERS/SELECT-OPTIONS

## ***Inicialização de variáveis e tabelas internas, intervalos fixos:***

INITIALIZATION

# Estrutura de um programa (2)

***Bloco principal do processamento:***

START-OF-SELECTION

***Fim do bloco principal de processamento:***

END-OF-SELECTION

***Outros eventos:***

AT USER-COMMAND

AT SELECTION-SCREEN [options]

TOP-OF-PAGE

# Regras de sintaxe (1)

- As regras de sintaxe do ABAP/4 são idênticas às das linguagens comuns: o código do programa é constituído por comandos

- Um comando é uma sequência de palavras que termina com um ponto

```
131|      REPORT abaptest.
132|      WRITE: 'customer list'.
```

- Uma palavra é sempre delimitada por brancos em ambos os lados ou por ponto do lado direito

```
134|      IF sy-subrc <> 0.
```

- Um comando começa sempre por uma palavra chave de ABAP/4 como por ex: *report* ou *write*

```
131|      REPORT abaptest.
```

- Um literal é uma sequência de caracteres entre plicas/apóstrofos

```
132|      WRITE: 'customer list'.
```

- Se for necessário usar apóstrofos numa literal deve-se duplicá-los

```
9|      WRITE 'Customer's Name'.
```

# Regras de sintaxe (2)

- A maioria dos comandos em ABAP permite ou requer parâmetros

```
12 MESSAGE e010 WITH var1.
```

- Todos os textos após um \* na primeira coluna ou " em qualquer coluna são considerados comentários

```
15 *toda a linha é comentário
16 MOVE a TO b. "move valor de a para o b
```

- Os comandos são independentes da posição: podem-se desenvolver por várias linhas, ou vários na mesma linha

```
18 MOVE var1
19     TO var2.
20 IF a = 2. MOVE a TO b. ENDIF.
```

- Por regra os comandos cujo início seja idêntico podem ser combinados separando a parte diferente por vírgula

```
22 WRITE 'lista de clientes'.
23 WRITE 'da empresa'.
```



```
22 WRITE: 'lista de clientes',
23     'da empresa'.
```



# Regras de sintaxe (3)

- Os comandos são organizados por tipo:
  - Elementos Declarativos
    - *Tables, Data, etc.*
  - Elementos Operacionais
    - *Add, Move, etc.*
  - Elementos de Controle
    - *Do, If, Case, Perform, Loop, etc.*
  - Elementos de Eventos
    - *TOP-OF-PAGE, AT USER-COMMAND, etc*

02

# Elementos de Linguagem

# Elementos declarativos

- TABLES
- TYPES
- DATA
- CONSTANTS
- STATIC

# Tables

- Declara uma tabela do Dicionário de Dados (SE11) para uso no programa
- Aloca uma variável com o mesmo nome e estrutura da tabela da BD para utilização como *buffer* para comunicar com a tabela

# Types

- O conceito “*TYPE*” fornece a possibilidade de definir estruturas complexas
- É o suporte básico para as futuras técnicas de programação orientadas ao objecto em ABAP

```

25  TYPES type.
26  TYPES type(len).
27  □ TYPES: BEGIN OF rectype,
28  |
29  |         END OF rectype.

```

## Opções:

```

... TYPE type1
... TYPE type2 LENGTH length [DECIMALS n]
... TYPE RANGE OF data1
... LIKE f1
... TYPE LINE OF itabtype
... LIKE LINE OF itab

```

# Tipos elementares

Categoria	Tipo	Descrição	Comprimento (byte)	Valor inicial
caracteres	c	<i>character</i>	1	Space
	n	<i>numeric text</i>	1	'0'
numéricos	i	<i>integer</i>	4	0
	p	<i>packed integer</i>	8	0
	f	<i>floating point number</i>	8	0.0
data/ hora	d	<i>date</i>	8	'00000000'
	t	<i>time</i>	6	'000000'
hexadecimal	x	<i>hexadecimal</i>	1	X'00'

- Recomendações:
  - I – Contadores
  - P – Cálculo comercial
  - F – Cálculo científico

# Data

- Permite declarar variáveis para serem utilizadas no programa. Nenhuma variável pode ser utilizada no programa sem ter sido previamente declarada

```
33 DATA <var>.  
34 DATA: BEGIN OF <var>,  
35  
36     END OF <var>.  
--
```

## Adições:

```
... TYPE type1  
... TYPE type2 LENGTH length [DECIMALS n]  
... TYPE RANGE OF data1  
... TYPE REF TO data2  
... LIKE f1  
... TYPE LINE OF itabtype  
... LIKE LINE OF itab
```

# Constants

- Permite declarar constantes para serem utilizadas no programa. Nenhuma constante pode ser utilizada no programa sem ter sido previamente declarada

```
39 ▶  CONSTANTS c TYPE type [length length] VALUE val.
40  □  CONSTANTS: BEGIN OF crec,
41  |
42  |          END OF crec.
```

Adições iguais às do comando DATA excepto:  
 ... TYPE type OCCURS n,  
 ... LIKE f1 OCCURS n,  
 ... WITH HEADER LINE



# Statics

- Define uma variável local que mantém o seu valor depois de deixar o espaço onde foi definido

```
44 | STATICS f TYPE type [length LENGTH decimals n].
```

## Adições:

```
... TYPE type1
... TYPE type2 LENGTH length [DECIMALS n]
... TYPE RANGE OF data1
... TYPE REF TO data2
... LIKE f1
... TYPE LINE OF itabtype
... LIKE LINE OF itab
```

# Notas sobre declarativas

- Existem várias formas de se definir um elemento complexo
- Os campos das estruturas acessem-se usando um hífen entre o nome da estrutura e o elemento: *struct-field*

Exemplo: MARA-MATNR (tabela MARA, campo MATNR)

# Variáveis de sistema

- Podem ser consultadas na estrutura SYST acessível através do dicionário de dados

Nome	Tipo	Tamanho	Descrição breve
Sy-datum	D	8	Data atual do servidor de aplicação
Sy-zeit	T	6	Hora atual do servidor de aplicação
Sy-uname	C	12	Nome do usuário actual
Sy-subrc	X	2	Código de retorno de instruções ABAP
Sy-langu	C	2	Código de idioma do ambiente de texto
Sy-linct	I	4	comprimento da página da lista atual
...			

# Elementos operacionais

- Existem inúmeros comandos operacionais que podem ser consultados *online*.
- De entre os vários existentes consideram-se, para efeitos de exemplo, os comandos *WRITE*, *MOVE* e *COMPUTE*.
- Estes comandos permitem observar o mecanismo ABAP de conversão entre tipos de variáveis.

# Comandos *WRITE*, *MOVE* e *COMPUTE*

- *Write:*

```
46 WRITE <format> <value> <options>.
```

```
Hello classical list
Hello classical list |
HELLO CLASSICAL LIST
42
```

- *Compute:* Cálculos aritméticos

```
50 COMPUTE var1 = expressão aritmética.
```

Nota: O comando compute é o único comando em ABAP em que se pode omitir a keyword.

- *Move:* – Atribuição de valores

```
48 MOVE var1 TO var2.
```

```
9 TYPES:
10 BEGIN OF ENUM number,
11     n0, n1, n2,
12 END OF ENUM number.
```

```
13
14 DATA num TYPE number.
```

```
15
16 "num = 1.
```

```
17 "MOVE 1 TO num.
```

```
18 MOVE EXACT 1 TO num.
```

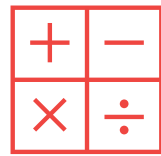
```
19
```

```
20 num = EXACT number( 1 ).
```

Nota  
de u  
camp

campos  
nas os  
uturas

# Operadores



+ / ADD (soma)

```
52 total = var1 + var2.
53 ADD var1 TO var2.
```

- / SUBTRACT (subtracção)

```
52 total = var1 - var2.
53 SUBTRACT var1 FROM var2.
```

\* / MULTIPLY (multiplicação)

```
52 total = var1 * var2.
53 MULTIPLY var1 BY var2.
```

/ / DIVIDE (resultado da divisão com parte decimal)

```
52 total = var1 / var2.
53 DIVIDE var1 BY var2.
```

DIV (resultado arredondado da divisão)

```
9 total = var1 DIV var2.
```

MOD (resto da divisão)

```
9 total = var1 MOD var2.
```

# Funções

$\sqrt{\phantom{x}}$  SQRT (raíz quadrada)

*exp* EXP (exponencial)

*log* LOG (logaritmo)

*sin* SIN (seno)

*COS* COS (coseno)

STRLEN (comprimento de *strings*)

# Operadores relacionais

EQ	=	(igual)
NE	<>	(diferente)
GT	>	(maior que)
GE	>=	(maior ou igual que)
LT	<	(menor que)
LE	<=	(menor ou igual que)
BETWEEN val1 AND val2		(entre val1 e val2)
IS INITIAL		(diferente de 0)
NOT		(negativa)



# Operadores booleanos / para *strings*

## Operadores Booleanos:

AND = (e)

OR <> (ou)

## Operadores para *strings*:

st1 CO st2 (contém apenas)

st1 CA st2 (contém pelo menos)

st1 CS st2 (contém a *string*)

st1 CP st2 (contém o padrão)

### Exemplos:

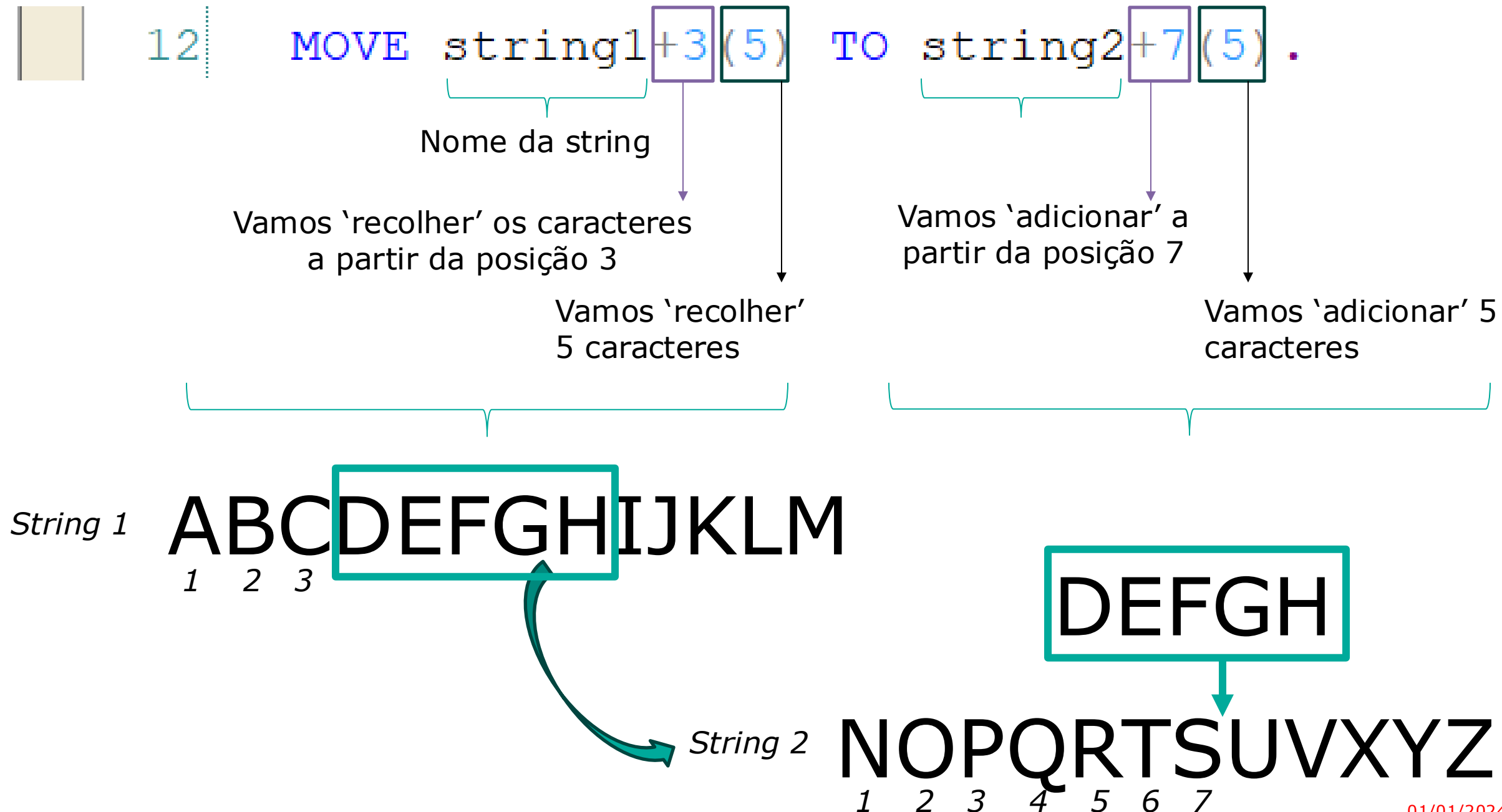
v\_texto CO 'ABCD' -> *true* se v\_texto só contém caracteres entre A, B, C e D

v\_texto CA 'ABCD' -> *true* se v\_texto contém pelo menos um caracter entre A, B, C e D

v\_texto CS 'Academia' -> *true* se v\_texto contém a *string* 'Academia'

V\_texto CP '\*INETUM\*' -> *true* se v\_texto contém o padrão INETUM

# Operações com *strings* (1)



# Operações com *strings* (2)

- SHIFT <string> [BY <n> PLACES] [LEFT | RIGHT] |
- [DELETE LEADING/TRAILING <padrão>].
- REPLACE [FIRST/ALL] OCURRENCES OF <string1> IN [TABLE] <variável>
- WITH <string2>.
- TRANSLATE <variável> TO UPPER/LOWER CASE.
- FIND [FIRST/ALL] OCURRENCES OF <string1> IN [TABLE] <variável>.
- CONDENSE <variável> (NO-GAPS).
- CONCATENATE <f1>...<fn> INTO <variável>.
- SPLIT <string> AT <separador> INTO <var1> <var2> ... <varn>.
- SPLIT <string> AT <separador> INTO TABLE <tabela>.

# Formatação de *output*

```

8  FORMAT INTENSIFIED OFF.
9  ULINE.
10 SKIP 2.
11 WRITE: /5 'Data'.
12  FORMAT INTENSIFIED.  Coloca as letras de cor azul
13 WRITE: 20 SY-DATUM DD/MM/YYYY.
14  FORMAT INTENSIFIED OFF.
15 WRITE: /5 'Hora'.
16  FORMAT INTENSIFIED.
17 WRITE: 20 SY-UZEIT.
18 NEW-PAGE.
19 WRITE: / 'Utilizador:', SY-UNAME UNDER SY-DATUM.
20 SKIP 2.
21 POSITION 10.
22 WRITE '* Fim de Programa *'.
23 RESERVE 3 lines.

```

## Output

Data	19.09.2023
Hora	13:33:46
Formação ABAP; Programa N°3 - Operadores Condicionais	
Utilizador:	ROFFSDF
* Fim de Programa *	


# Conversão de *output* e *substrings*

```

8  DATA: account(10) TYPE c,
9      start      TYPE d.
10
11  account = '1234567890'.
12  MOVE '19970305' TO start.
13
14  WRITE: account+8(2), '**', start(4).
15
16  * Obter o dia na data (2 posições a partir da 6ª)
17  start+6(2) = '01'.
18  account+6 = '9999'.
19
20  * Escrita da data com formatação
21  WRITE: / account, '****', start DD/MM/YYYY.

```

Output



```

90 ** 1997
1234569999 **** 01.03.1997

```

# Conversões e formatos

```

285 DATA: valor          TYPE p DECIMALS 2 VALUE 1250,
286       c_valor(10) .
287
288 MOVE valor TO c_valor.
289 WRITE: 40 c_valor.
290 WRITE: valor.
291
292 WRITE valor TO c_valor.
293 NEW-LINE.
294 WRITE AT 10 c_valor.

```

## Output

```

1250.00
1.250,00

```



# Strings: exemplo

```

8 DATA:gv_frase TYPE c LENGTH 40,
9     gv_tam     TYPE i,
10    gv_mes(8)  TYPE c VALUE 'Novembro',
11    moff       TYPE i,
12    mlen       TYPE i,
13    gv_result  TYPE c LENGTH 40,
14    n_moff(2)  TYPE n,
15    n_mlen(2)  TYPE n.
16
17 START-OF-SELECTION.
18 CLEAR: gv_frase,
19     gv_tam,
20     moff,
21     mlen,
22     gv_result,
23     n_moff,
24     n_mlen.
25
26 gv_frase = 'Lisboa, 10 de Novembro de 2009'.
27
28 COMPUTE gv_tam = strlen( gv_frase ).
29 WRITE:/ 'Tamanho total da string com a data:', gv_tam.
30 SKIP 1.
31
32 FIND ALL OCCURRENCES OF gv_mes IN gv_frase
33 IGNORING CASE
34 MATCH OFFSET moff
35 MATCH LENGTH mlen.
36
37 IF sy-subrc = 0.
38     n_moff = moff.
39     n_mlen = mlen.
40     CONCATENATE 'Encontrada na posição:'
41     n_moff
42     'tamanho'
43     n_mlen INTO gv_result SEPARATED BY space.
44     WRITE:/ gv_result.
45 ENDIF.

```

Declaração de  
variáveis, constantes

Limpeza de variáveis

Definir variável gv\_frase e  
obter o seu tamanho

Encontrar o início da palavra  
"Novembro" na *string* e  
registrar onde foi encontrada

Utilização da variável de  
sistema sy-subrc para  
avaliação de sucesso/insucesso  
da instrução de código anterior

## Output

```

Tamanho total da string com a data:      30
Encontrada na posição: 14 tamanho 08

```

# Elementos de controlo

- Condicionais

- *IF*
- *CASE*

- Ciclos

- *DO*
- *WHILE*
- *LOOP*

- Interrupções de Ciclos

- *CONTINUE*
- *CHECK*
- *EXIT*

```

8 CASE <f>.
9   WHEN <f1>.
10    <statement block>.
11   WHEN <f2>.
12    <statement block>.
13   WHEN <f3>.
14    <statement block>.
15   WHEN OTHERS.
16    <statement block>.
17   ENDCASE.
18
19 LOOP AT <v_01>.
20   <statement block>.
21   ENDLOOP.
22
23 .....
24 WHEN OTHERS.
25   <statement block>.
26   ENDCASE.
27
28
29

```

```

IF NUM > 30.
CASE SIZE TIPO.
WHEN 'A'.
DO MSG 'A' TO wa_mara.
WHEN 'B'.
DO MSG 'B' TO wa_mara.
WHEN 'C'.
DO MSG 'C' TO wa_mara.
WHEN 'D'.
DO MSG 'D' TO wa_mara.
WHEN 'E'.
DO MSG 'E' TO wa_mara.
WHEN 'F'.
DO MSG 'F' TO wa_mara.
WHEN 'G'.
DO MSG 'G' TO wa_mara.
WHEN 'H'.
DO MSG 'H' TO wa_mara.
WHEN 'I'.
DO MSG 'I' TO wa_mara.
WHEN 'J'.
DO MSG 'J' TO wa_mara.
WHEN 'K'.
DO MSG 'K' TO wa_mara.
WHEN 'L'.
DO MSG 'L' TO wa_mara.
WHEN 'M'.
DO MSG 'M' TO wa_mara.
WHEN 'N'.
DO MSG 'N' TO wa_mara.
WHEN 'O'.
DO MSG 'O' TO wa_mara.
WHEN 'P'.
DO MSG 'P' TO wa_mara.
WHEN 'Q'.
DO MSG 'Q' TO wa_mara.
WHEN 'R'.
DO MSG 'R' TO wa_mara.
WHEN 'S'.
DO MSG 'S' TO wa_mara.
WHEN 'T'.
DO MSG 'T' TO wa_mara.
WHEN 'U'.
DO MSG 'U' TO wa_mara.
WHEN 'V'.
DO MSG 'V' TO wa_mara.
WHEN 'W'.
DO MSG 'W' TO wa_mara.
WHEN 'X'.
DO MSG 'X' TO wa_mara.
WHEN 'Y'.
DO MSG 'Y' TO wa_mara.
WHEN 'Z'.
DO MSG 'Z' TO wa_mara.
WHEN OTHERS.
DO MSG 'Desconhecido' TO wa_mara.
ENDCASE.
ENDIF.

```

<b>CONTINUE</b>	Termina incondicionalmente a iteração actual.
<b>CHECK</b>	Termina condicionalmente a iteração actual.
<b>EXIT</b>	Termina a execução do ciclo.



# Elementos para interrupção de ciclos

- *CHECK* <condição>

- Dentro de um ciclo

```

8  WHILE counter > 0.
9      bloco1.
10     CHECK flag <> space.
11     bloco2.
12 ENDWHILE.

```

Enquanto o contador 'counter' for superior a zero irá correr o ciclo *loop*. Em cada iteração do ciclo verifica se a condição 'check' se verifica. Se se verificar corre o código do bloco 2 e passa para a próxima iteração do ciclo

- Fora de um ciclo

```

15  bloco1.
16  CHECK flag <> space.
17  bloco2.

```

**Não recomendado!**  
**SAP recomenda utilizar o *RETURN***

# Elementos para interrupção de ciclos

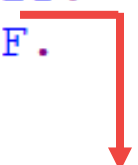
- *EXIT*

- Dentro de um ciclo

```

8  DO.
9  IF counter > 10.
10  EXIT.
11  ENDIF.
12  ENDDO.

```



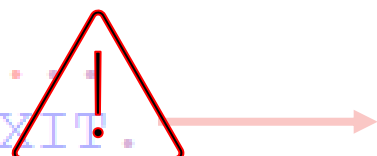
Dentro do ciclo 'do', se a condição do 'if' for verificada (neste caso ser superior a 10) então sai do ciclo 'do'

- Fora de um ciclo

```

8  IF ...
9  EXIT.
10 ENDIF.

```



Não recomendado!  
SAP recomenda utilizar o *RETURN*

# Subrotinas

- Secções de encapsulação de código que são re-usáveis: usadas sempre que chamadas
- 'Chamadas' pelo comando *PERFORM*
- Definidas dentro de *FORM...ENDFORM.*

```

174  PERFORM form.
175  PERFORM form(prog) .
176  PERFORM form IN PROGRAM prog IF FOUND.
177  PERFORM n OF form1 form2.
178  PERFORM form USING p1 p2.
179  PERFORM form CHANGING p1 p2

```

# Subrotinas: algumas notas

- A utilização da adição *TYPE* ou *LIKE* obriga a que as variáveis e os correspondents parâmetros sejam do mesmo tipo ou convertíveis.
- A adição *CHANGING* é puramente documental visto que as variáveis são sempre passadas por referência a não ser que seja usada a adição *VALUE*.
- Mesmo quando as variáveis são passadas por valor se a variável for alterada o novo conteúdo será passado para a variável na saída da subrotina.

# Subrotinas: exemplo

- *Cálculo da raiz quadrada de um número*

```

6  REPORT zfabap_jul2023_demo03_alc.
7
8  PARAMETERS: p_valor(8) TYPE p DECIMALS 3.
9  DATA: gv_raiz(8) TYPE p DECIMALS 3,
10         gv_except.
11
12  START-OF-SELECTION.
13    CLEAR: gv_raiz, gv_except.
14
15    PERFORM raiz_quadrada USING p_valor
16          CHANGING gv_raiz
17          gv_except.
18
19  IF gv_except IS INITIAL.
20    WRITE:/ 'Raiz quadrada = ', gv_raiz.
21  ENDIF.
22
23  END-OF-SELECTION.

```

```

*&-----*
*&  Form RAIZ_QUADRADA
*&-----*
FORM raiz_quadrada USING  f_valor
                        CHANGING  f_raiz
                        f_except.

IF f_valor < 0.
  f_except = 'X'.
  MESSAGE s002.
ELSE.
  f_raiz = SQRT( f_valor ).
ENDIF.

ENDFORM.          " RAIZ_QUADRADA

```

# Módulos de função

## ? Funções vs subrotinas

*As funções distinguem-se das subrotinas nos seguintes pontos:*

- *São objectos que pertencem ao repositório.*
- *Apenas são acedidos pela sua interface.*
- *São visíveis e acessíveis por todos os restantes objectos ABAP.*
- *São definidas através de um editor próprio onde se definem as suas características.*
- *Têm mecanismos próprios específicos para lidar com situações de erro.*

# Chamadas de módulos de função em ABAP

```

8      DATA: field(30) VALUE 'Example: This is a field.',
9            head(30).
10
11      CALL FUNCTION 'STRING_SPLIT'
12        EXPORTING
13          delimiter = ':'
14          string    = field
15        IMPORTING
16          head      = head
17          tail      = field
18        EXCEPTIONS
19          not_found = 1
20          OTHERS   = 2.
21
22      CASE sy-subrc.
23      WHEN 1.
24          ...
25      WHEN 2.
26          ...
27      ENDCASE.

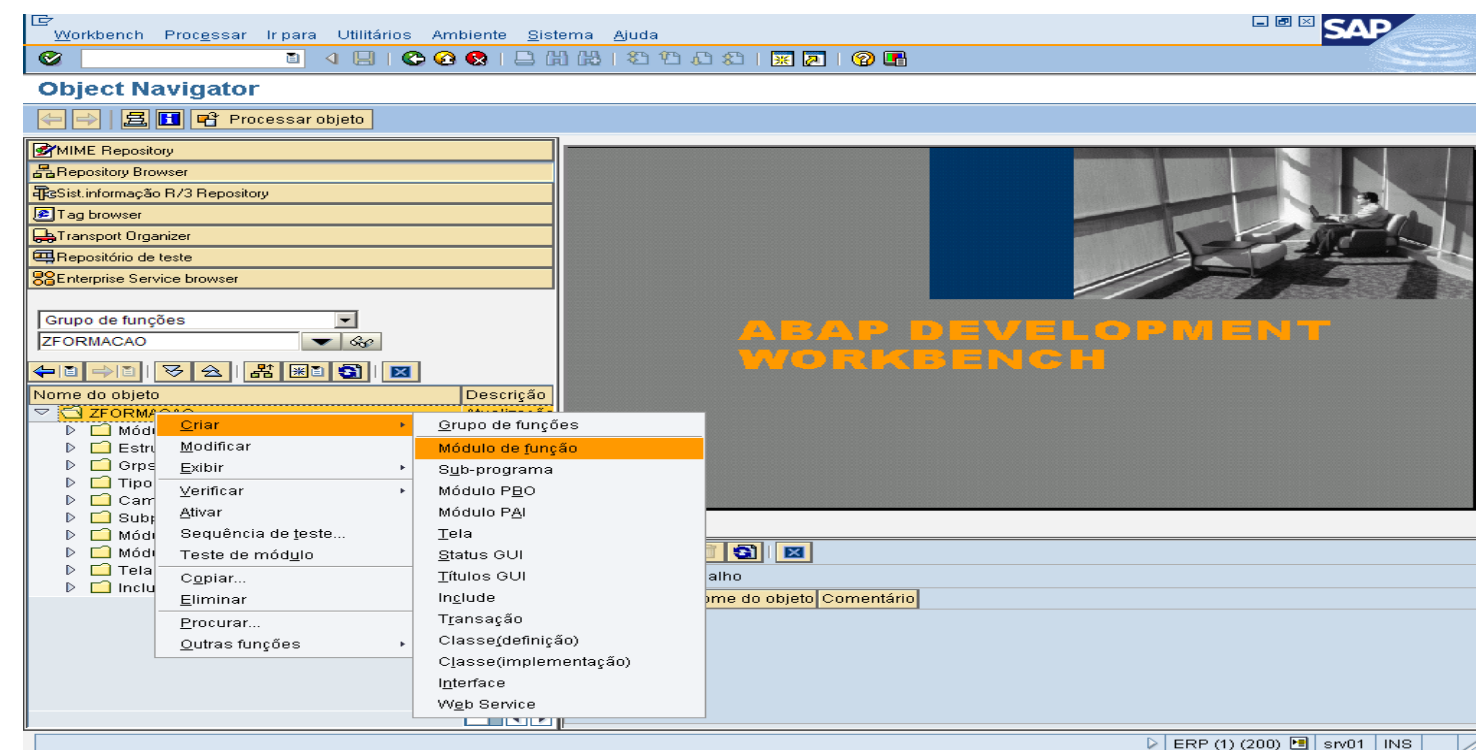
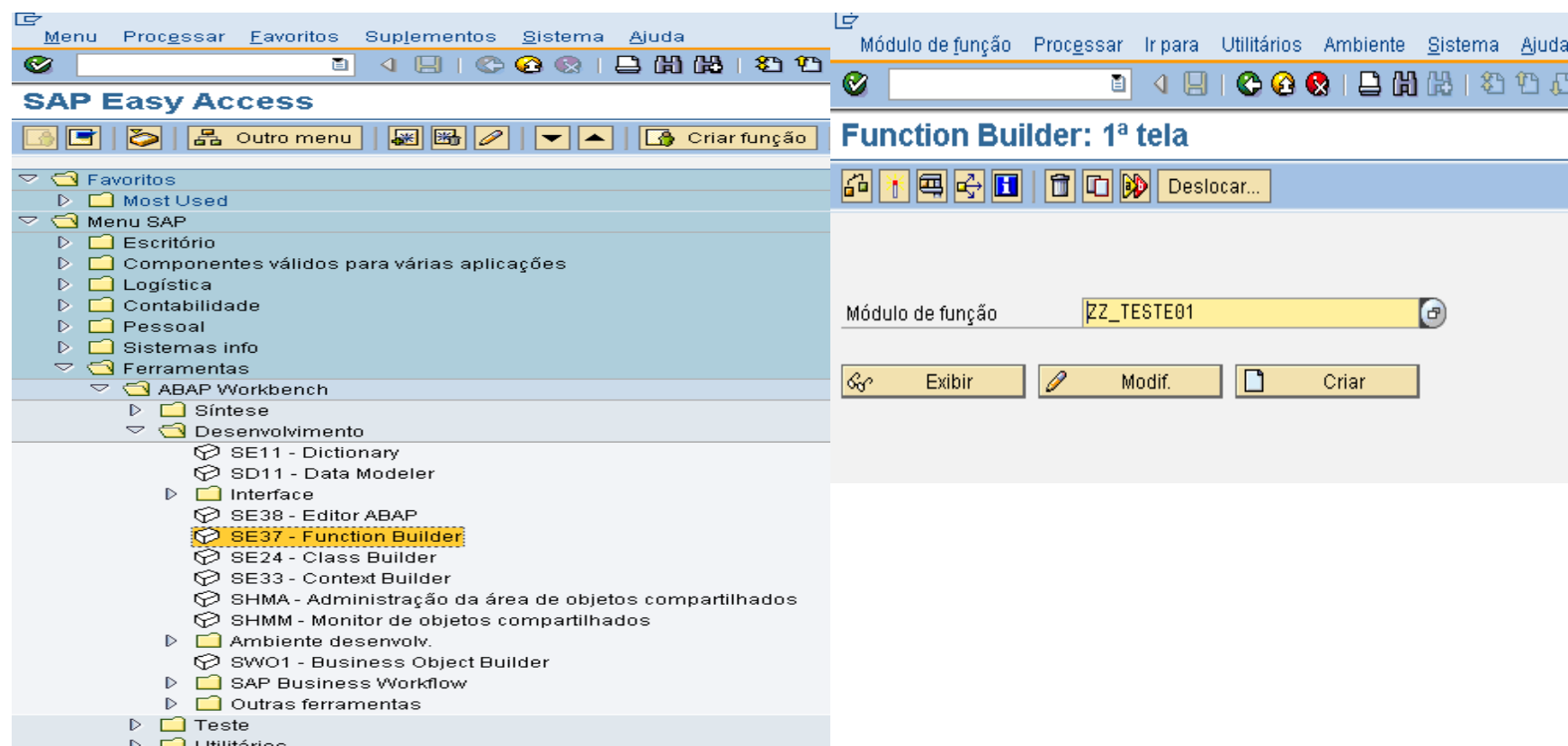
```

# Módulos de função

- Gerados através do Function Builder (Transacção SE37) ou através da transacção SE80

SE37

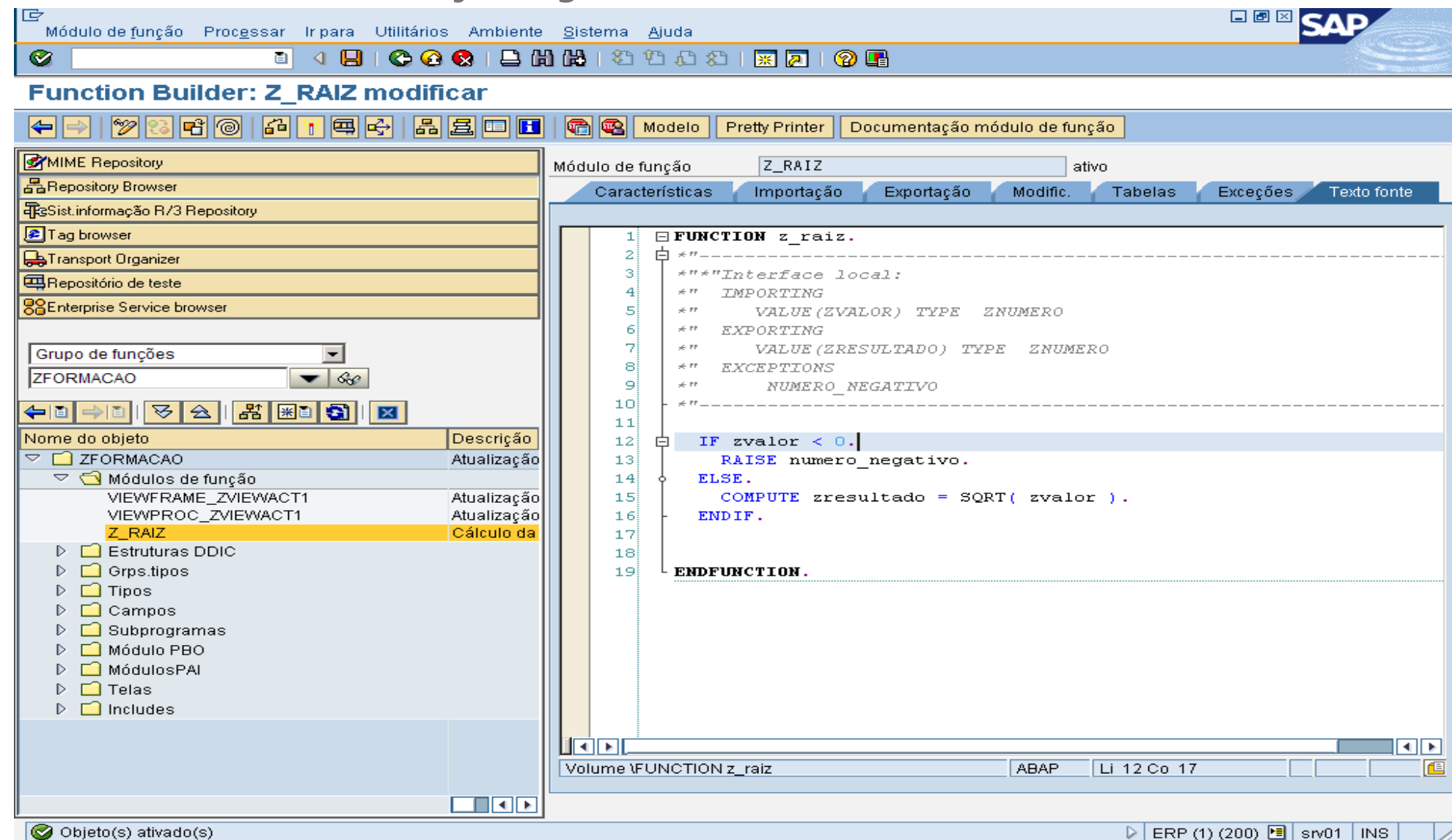
SE80





# Módulos de função: exemplo (1)

- Criar módulo de função que receba um parâmetro numérico e devolva a raiz quadrada, validando que o parâmetro recebido não seja negativo.



# Módulos de função: exemplo (2)

- Criar módulo de função que receba um parâmetro numérico e devolva a raiz quadrada, validando que o parâmetro recebido não seja negativo.

SAP

Módulos de função Processar Ir para Utilitários Sistema Ajuda

Testar módulo de função: tela de entrada

Depuração Diretório de dados de teste

Teste p/grupo de funções ZFORMACAO  
Módulo de função Z\_RAIZ  
Maiúsculas/minúsculas ☐

Parâmetro importação	Valor
ZVALOR	2,00

ERP (1) (200) srv01 INS

SAP

Módulos de função Processar Ir para Utilitários Sistema Ajuda

Testar módulo de função: tela de resultado

Teste p/grupo de funções ZFORMACAO  
Módulo de função Z\_RAIZ  
Maiúsculas/minúsculas ☐

Tmp.exec.: 219 Microsegundos

Parâmetro importação	Valor
ZVALOR	2,00

Parâms.export.	Valor
ZRESULTADO	1,41

ERP (1) (200) srv01 INS

# Tabelas

- Internas
- Base de Dados

# Tabelas Internas

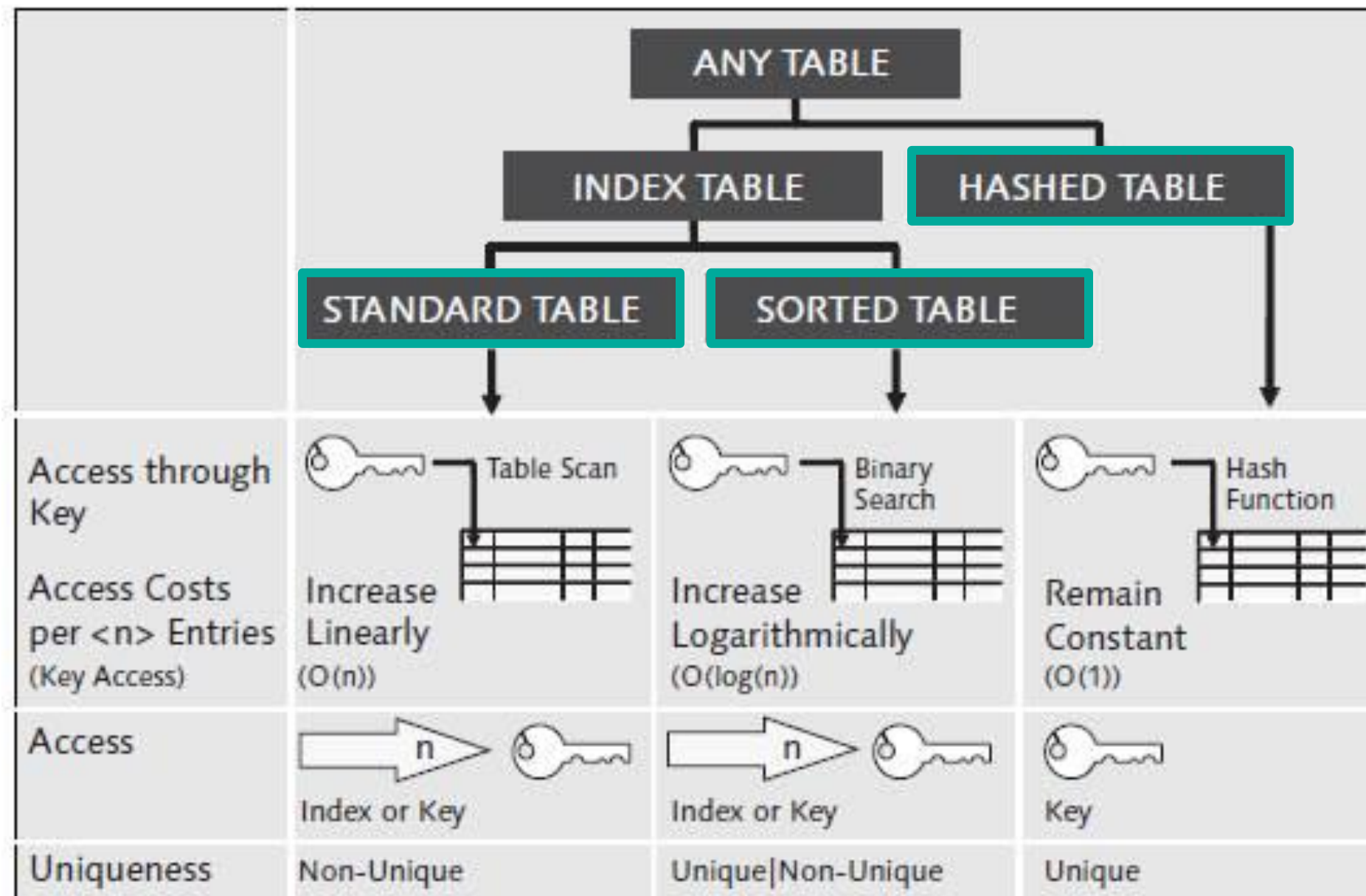
- Apenas existem durante a execução de um programa
- O número de linhas não é limitado
- Têm especial utilidade para as seguintes operações:
  - Melhorar a *performance* lendo as linhas significativas das tabelas da BD em apenas uma operação
  - Criação de visões temporárias com dados de várias tabelas
  - Maior facilidade de construção de listagens

```

9  TYPES: BEGIN OF t_line,
10      column1 TYPE i,
11      column2 TYPE i,
12      column3 TYPE i,
13      END OF t_line.
14
15  TYPES itab TYPE STANDARD TABLE OF t_line.
16
17  DATA tab1 TYPE itab.
18  DATA tab2 TYPE STANDARD TABLE OF t_line.

```

# Tipos de Tabelas Internas



# Tabelas Internas c/ cabeçalho

- linha de cabeçalho + conjunto de linhas estruturadas de uma tabela
- linha de *Header* tem a mesma estrutura que um registro
- linha de *Header* é a área de trabalho de uma tabela interna
- quando se preenche uma tabela interna, os dados são primeiro colocados dentro da “*Header Line*” e só depois copiados para a tabela. O mesmo acontece com as outras operações que envolvem tabelas internas.

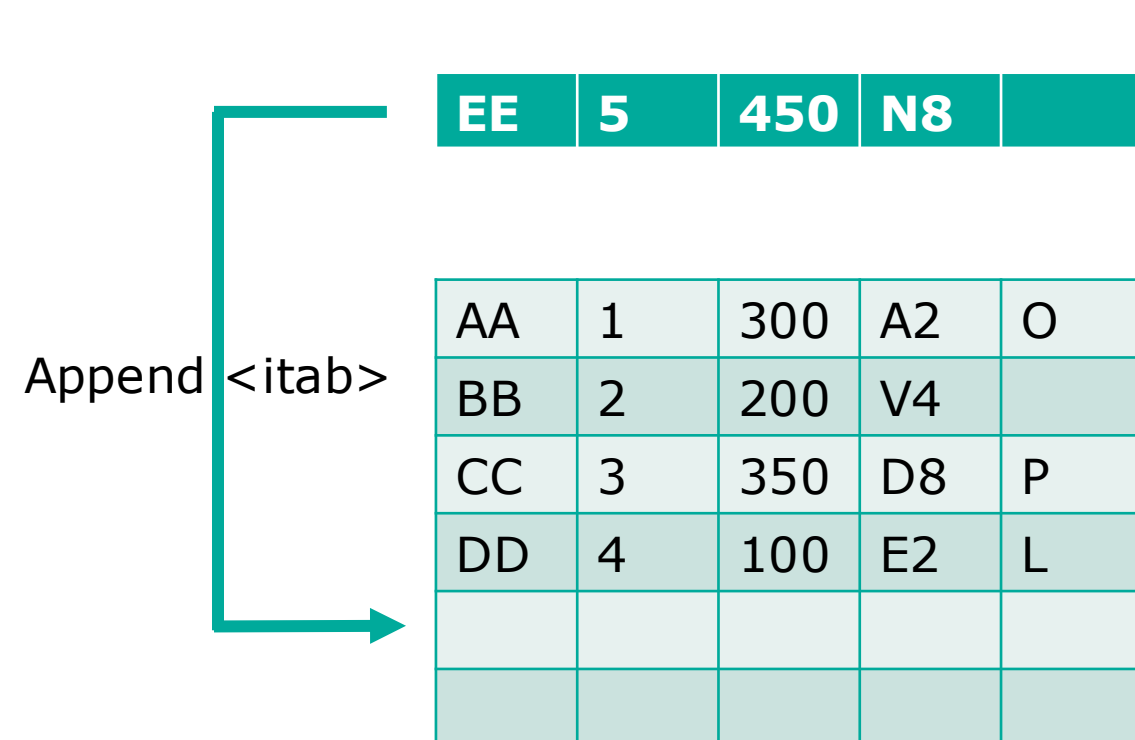
Exemplo: DATA: ADDR\_TAB6 TYPE ADDR\_TAB WITH HEADER LINE.

# Tabelas Internas s/ cabeçalho

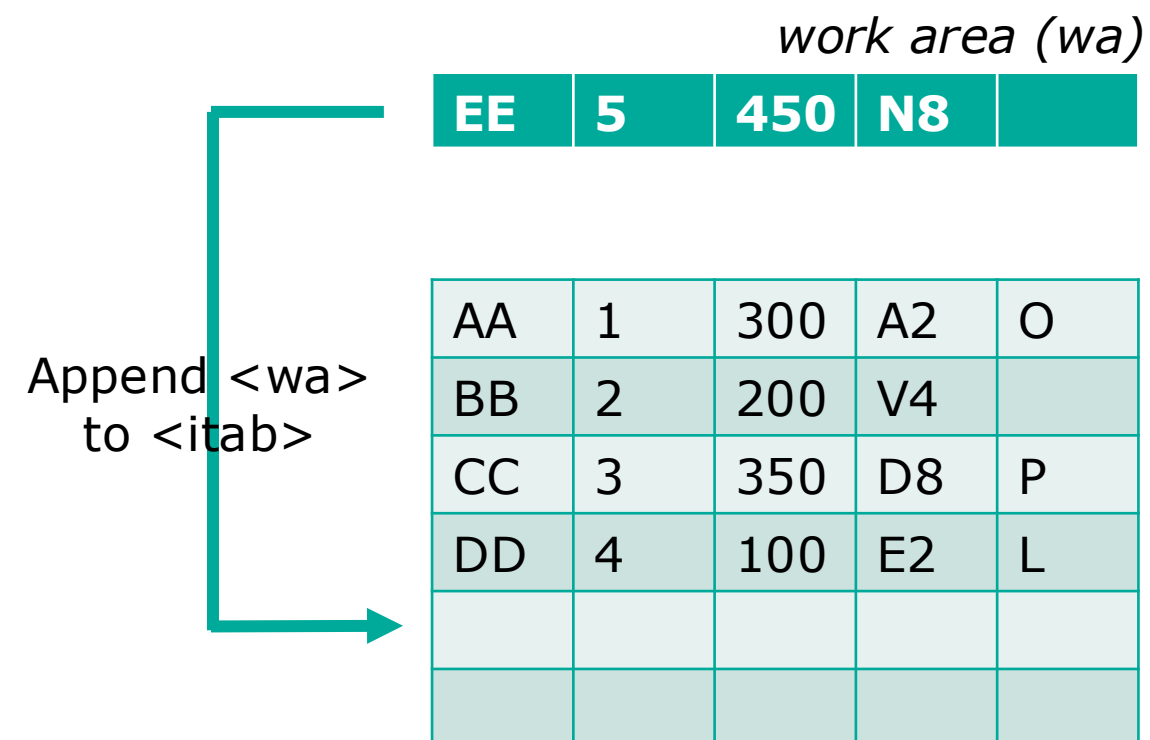
- É possível declarar Tabelas internas sem *Header Line*
- Uma tabela com este formato consiste num conjunto de linhas com estrutura idêntica e previamente especificada no programa através de um TYPE. Contudo, é necessário o recurso a uma estrutura auxiliar do mesmo tipo que a da tabela interna para aceder aos seus dados.

Exemplo: DATA: ADDR\_TAB TYPE ADDR\_TAB.

# Tabelas Internas c/ e s/ cabeçalho



**Com cabeçalho**



**Sem cabeçalho**



# Tabelas Internas: operações (1)

- LOOP Lê uma ou mais linhas da tabela
- AT Acede a uma área específica da tabela
- READ TABLE Lê uma linha da tabela
- APPEND Insere uma linha no fim da tabela
- MODIFY Modifica uma linha da tabela
- INSERT Insere uma linha na tabela
- DELETE Apaga uma linha da tabela
- CLEAR <itab> Apaga a Header Line
- CLEAR <itab>[ ] Apaga todas as entradas da tabela; O espaço de memória continua ocupado
- REFRESH <itab> Apaga todas as entradas da tabela; O espaço de memória continua ocupado
- FREE <itab> Apaga todas as entradas da tabela; O espaço de memória é libertado.

# Tabelas Internas: operações (2)

- *COLLECT*      Adição/inserção de linha
- *SORT*      Ordena a tabela
- *DESCRIBE*      Obtém informações técnicas da tabela
- *SY-TABIX*      Variável de sistema que indica qual a última linha acedida da tabela
- *SY-SUBRC*      Variável de sistema que contém o código de retorno da última operação executada sobre a tabela

# Tabelas Internas: LOOP

- Processamento de tabelas internas (leitura de dados)

```

8 LOOP AT <itab> INTO <wa> FROM <n1> TO <n2>
9                               WHERE <condition>.
10 .....
11 ENDLOOP.

```

opcionais

Passagem dos valores de cada linha da tabela para uma estrutura

```

13 LOOP AT <itab> ASSIGNING <fs> FROM <n1> TO <n2>
14                               WHERE <condition>.
15 .....
16 ENDLOOP.

```

opcionais

Associar um apontador

Exemplo:

```

19 LOOP AT tab1.
20     WRITE: /1 sy-tabix,
21            10 tab1-code.
22     IF sy-tabix > 10.
23         EXIT.
24     ENDIF.
25 ENDLOOP.

```


# Tabelas Internas: *AT*

- Acesso a uma área específica da tabela interna

```

27  LOOP AT <itab>.
28      AT <line>.
29      <statement block>.
30  ENDLOOP.

```

- 
- *AT FIRST*                      Primeira linha da tabela interna
  - *AT LAST*                        Última linha da tabela interna
  - *AT NEW <f>*                    Início dum grupo de linhas que contêm os mesmos valores no campo <f>
  - *AT END Of <f>*                Fim dum grupo de linhas que contêm os mesmos valores no campo <f>

# Tabelas Internas: exemplo

```

18 TABLES actfli.
19 DATA my_flights LIKE actfli OCCURS 10 WITH HEADER LINE.
20 DATA fldate TYPE datum.
21
22 START-OF-SELECTION.
23     SELECT * FROM actfli INTO TABLE my_flights ORDER BY PRIMARY KEY.
24
25     LOOP AT my_flights.
26     |   AT NEW fldate.
27     |   |   WRITE: / 'date', fldate.
28     |   |   ENDAT.
29     |
30     |   WRITE: / my_flights-carrid, my_flights-seatsocc.
31     |
32     |   AT END OF seatsocc.
33     |   |   SUM.
34     |   |   WRITE: / 'Occupied seats total:', my_flights-seatsocc.
35     |   |   ENDAT.
36     |
37     ENDLOOP.

```

# Tabelas Internas: outros comandos complementares

- *APPEND wa TO itab.*
- *COLLECT wa INTO itab.*
- *INSERT wa INTO itab.*
- *MODIFY itab FROM wa.*
- *READ TABLE itab INTO wa.*
- *LOOP AT itab INTO wa.*
- *SORT itab [ASCENDING/DESCENDING] BY <key>.*

# Tabelas de Base de Dados: *SELECT*

Declaração:

```
18 TABLES tabela.
```

Exemplo:

```
20 TABLES actfli.
```

Leitura de dados da tabela de base de dados: comando *SELECT*

- \* indica a selecção de todas as colunas.
- As variáveis são colocadas no header da tabela (se não existir opção INTO) desde que esta tenha sido declarada como TABLES.

Exemplo:

```
8 DATA: itab TYPE STANDARD TABLE OF t001,
9      wa TYPE t001.
10 SELECT * FROM t001
11 INTO TABLE itab.
```

# Tabelas de Base de Dados:

## ***SELECT SINGLE***

- A opção *SINGLE* indica que se vai ler apenas uma linha da tabela.
- É recomendável especificar toda a chave primária.
- As variáveis são colocadas no header da tabela (se não existir opção INTO).

Exemplo:

```

10  SELECT SINGLE *
11      FROM t001
12      WHERE bukrs = 'pt01'.
13
14  IF sy-subrc = 0.
15      WRITE / t001-bukrs.
16  ELSE.
17      WRITE / 'código de companhia não existe'.
18  ENDIF.

```



# Tabelas de Base de Dados: *selecção de alguns campos/columnas*

- É obrigatório usar o comando *INTO*
- Desempenho:
  - Mais eficiente ler apenas as colunas que interessam
  - Se for lida apenas uma linha não se deve usar *INTO TABLE*
  - Se for preciso ler muitas linhas, *INTO TABLE* é muito mais eficiente
  - *INTO CORRESPONDING FIELDS* deve ser evitado p/ grandes volumes

Exemplo:

```

6  REPORT zfabap_jul2023_demo03_alc.
7
8  TABLES t001.
9
10 DATA   xbukrs LIKE t001-bukrs.
11
12  SELECT bukr> INTO (xbukrs)
13    FROM t001.
14    WRITE / xbukrs.
15  ENDSELECT.

```

# Tabelas de Base de Dados: funções de agregação

- Max(<fields>)                      Máximo
- MIN(<fields>)                      Mínimo
- SUM(<fields>)                      Soma
- AVG(<fields>)                      Média
- COUNT(<fields>)                      Contagem

Exemplo:

```

8  TABLES t001.
9
10 DATA: countfield TYPE i,
11         maxfield   LIKE t001-bukrs,
12         minfield   LIKE t001-bukrs.
13
14 SELECT COUNT( DISTINCT bukrs )
15        MAX( bukrs )
16        MIN( bukrs )
17        FROM t001 INTO (countfield, maxfield, minfield).
18
19
20 WRITE: / countfield, maxfield, minfield.

```

# Tabelas de Base de Dados: funções de agregação – *GROUP BY*

Exemplo:

```

10 DATA: count TYPE i,
11          sum  TYPE p DECIMALS 2,
12          avg  TYPE f,
13          connid LIKE sbook.
14
15 SELECT connid COUNT( * )
16          SUM( luggweight )
17          AVG( luggweight )
18          INTO (connid, count, sum, avg)
19          FROM sbook
20          WHERE carrid = 'LH' AND fldate = '19950228'
21          GROUP BY connid.
22
23 WRITE: / connid, count, sum, avg.
24 ENDSELECT.

```

# Tabelas de Base de Dados: ***DISTINCT***

- Elimina duplicações durante a selecção
- Obriga o *SELECT* a executar um *SORT* implícito

```

8  SELECT SINGLE * ...
9  SELECT [distinct] * ...
10 ► SELECT [distinct] a1 ... an ...
11

```

# Tabelas de Base de Dados: *INTO*

- coloca os valores numa variável
- todas as linhas podem ser obtidas utilizando *INTO TABLE*

... *INTO structure*  
 ... *INTO CORRESPONDING FIELDS OF struct*  
 ... *INTO TABLE itab*  
 ... *INTO CORRESPONDING FIELDS OF TABLE itab*

Exemplo:

```
8  TABLES t001.
9  DATA itab LIKE t001 OCCURS 10 WITH HEADER LINE.
10 SELECT * FROM t001 INTO TABLE itab.
```

# Tabelas de Base de Dados: *FOR ALL ENTRIES*

- Permite uma restrição de dados utilizando outra tabela interna que esteja obrigatoriamente preenchida

*... FOR ALL ENTRIES IN itab WHERE cond*

Exemplo:

```

8  □ DATA: BEGIN OF ftab OCCURS 10,
9      bukrs LIKE t001-bukrs,
10     lang  LIKE t001-spras,
11     END OF ftab.
12
13     ftab-bukrs = '0001'.
14     ftab-lang = 'D'.
15     APPEND ftab.
16
17     ftab-bukrs = '0002'.
18     ftab-lang = 'E'.
19     APPEND ftab.
20
21 □ SELECT * FROM t001
22     FOR ALL ENTRIES IN ftab
23     WHERE bukrs NE ftab-bukrs AND
24     spras = ftab-lang AND butxt LIKE 'A%'.
25
26 ENDSELECT.
```



[inetum.com](https://inetum.com)

FRANCE | SPAIN | PORTUGAL | BELGIUM | SWITZERLAND | LUXEMBOURG | ENGLAND |  
POLAND | ROMANIA | MOROCCO | TUNISIA | SENEGAL | CÔTE D'IVOIRE | ANGOLA |  
CAMEROON | USA | BRAZIL | COLOMBIA | MEXICO | RP OF PANAMA | PERU | CHILE |  
COSTA RICA | DOMINICAN REPUBLIC | ARGENTINA | SINGAPORE | UAE

