

p

ABAP
REPORT

Qual a função do R/3?

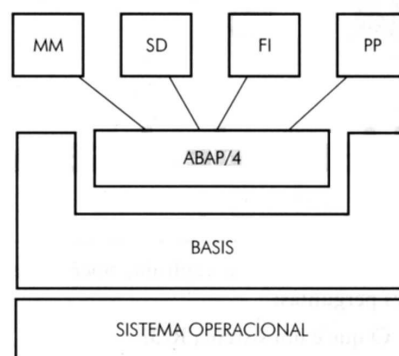
É oferecer um conjunto integrado compacto, de aplicativos empresariais de grande escala. O conjunto padrão de aplicativos integrados com cada sistema R/3 é o seguinte:

- PP (Production Planning - planejamento de produção)
- MM (Materials Management - gerenciamento de materiais)
- SD (Sales and Distribution - vendas e distribuição)
- FI (Financial Accounting - contabilidade financeira)
- CO (Controlling - controladoria)
- AM (Fixed Assets Management - gerenciamento de ativos fixos)
- PS (Project System - sistema de projeto)
- WF (Workflow - fluxo de trabalho)
- IS (Industry Solutions - soluções de indústria)
- HR (Human Resources - recursos humanos)
- PM (Plant Maintenance - manutenção das instalações)
- QM (Quality Management - gerenciamento da qualidade)

Estes aplicativos são chamados áreas funcionais, áreas de aplicativo ou, às vezes, módulos funcionais do R/3. Todos estes termos são sinônimos.

Tradicionalmente, as estruturas empresariais desenvolvem um conjunto de aplicativos de processamento de dados avaliando produtos individuais e comprando esses produtos separadamente de diversos fabricantes de software. Interfaces entre eles são então necessárias. Por exemplo, o sistema de gerenciamento de materiais precisará ser vinculado com as vendas e a distribuição e com os sistemas financeiros, e o sistema de fluxo de trabalho precisará de uma alimentação a partir do sistema do HR. Uma quantidade significativa de tempo e dinheiro de IS é gasta na implementação e manutenção dessas interfaces.

O R/3 vem predefinindo com os aplicativos empresariais básicos necessários à maioria das grandes corporações. Esses aplicativos coexistem em um ambiente homogêneo. Eles são projetados para funcionar utilizando desde um único banco de dados e um conjunto (muito grande) de tabelas. Os tamanhos de banco de dados de produção atuais variam de 12 gigabytes a aproximadamente 3 terabytes. Cerca de 8.000 tabelas de banco de dados são distribuídas com o produto R/3 padrão.



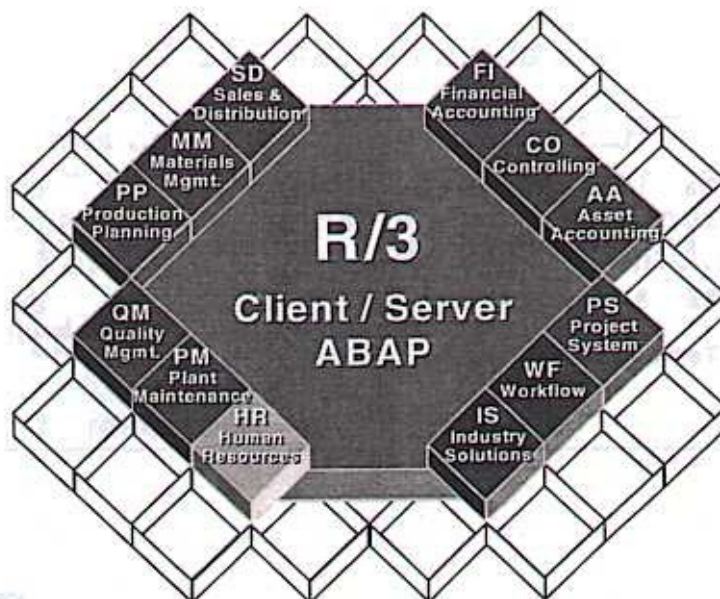
Por que você precisa saber isso?

É importante para você, como um programador de ABAP, saber por que esses aplicativos são todos escritos inteiramente em ABAP. Esses são os aplicativos que você deve entender para ser um bom desenvolvedor de R/3.

Por exemplo, suponha que você conhece ABAP e tenha recebido a incumbência de escrever um relatório financeiro que resume débitos e créditos de cada ano fiscal para cada fabricante na empresa. Talvez você saiba escrever código em ABAP, mas você saberia como começar a resolver essa tarefa?

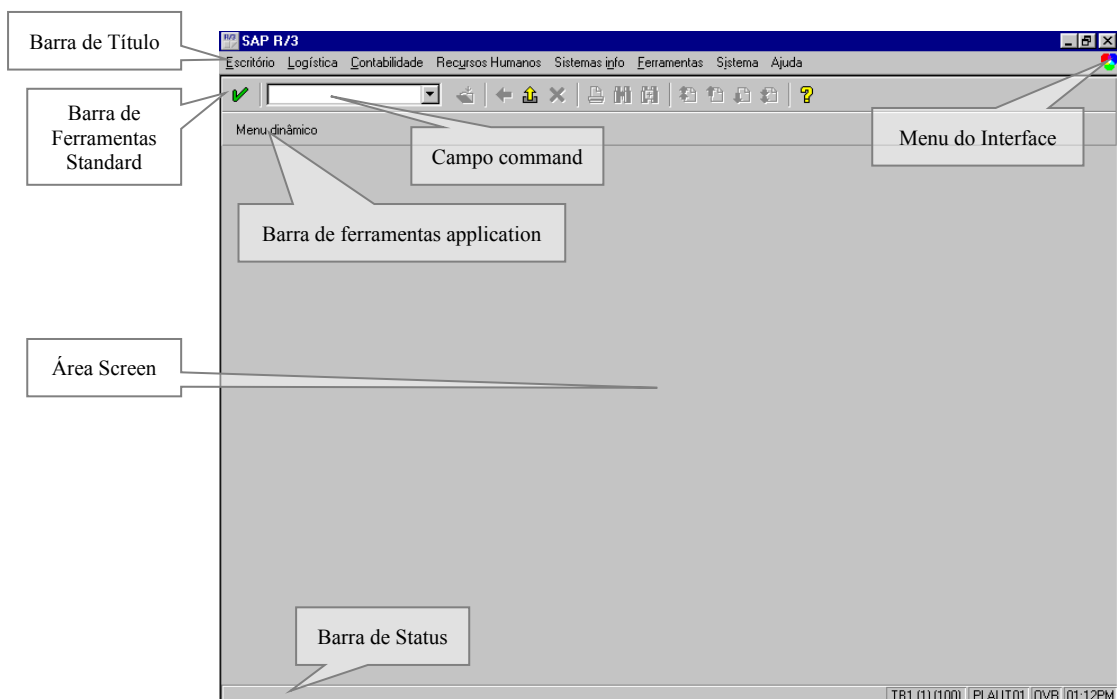
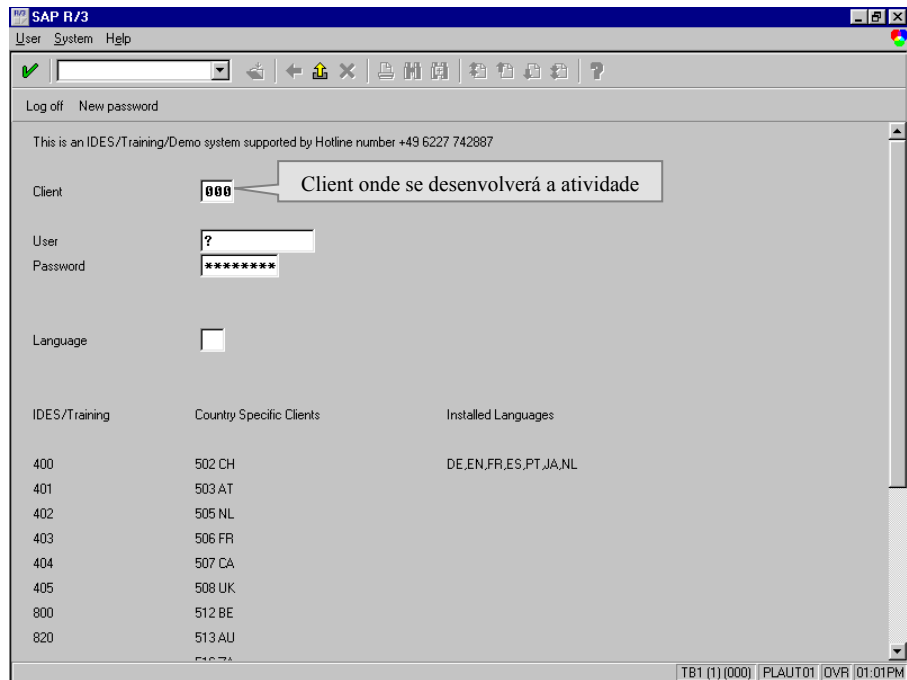
Ou talvez seu trabalho exija um novo desenvolvimento em ABAP/4. Você é incumbido de projetar um sistema que ofereça informações de estoque para potenciais compradores. Se você não conhece os sistemas de vendas e financeiro e de distribuição, você não saberá se está criando algo que já existe em R/3. Tampouco você poderá saber se há tabelas do R/3 que já contêm dados semelhantes ou idênticos aos tipos de dados que você quer recuperar. Esses aplicativos são altamente integrados. Um desenvolvedor que assume a abordagem “eu construirei minhas próprias tabelas e manterei minhas próprias cópias dos dados”, pode logo descobrir que seus dados são redundantes e devem ser rotineiramente sincronizados com o resto do banco de dados. Ele construiu um aplicativo que não tira proveito da natureza altamente integrada do ambiente R/3.

É importante que o ABAP seja desenvolvido com a certeza de que dentro do R/3 não possua nada que já não atenda as necessidades da área funcional.



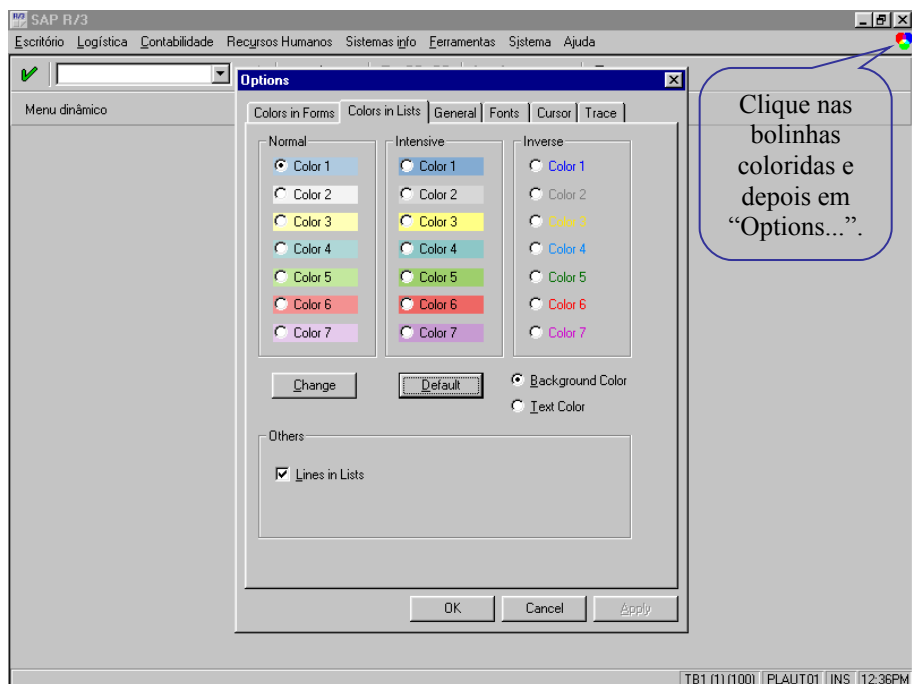
Logon

Através do logon o sistema identificará se o usuário esta autorizado a entrar no SAP no client que está acessando, e é através do seu UserId que é identificado a que módulos e transações você terá acesso.



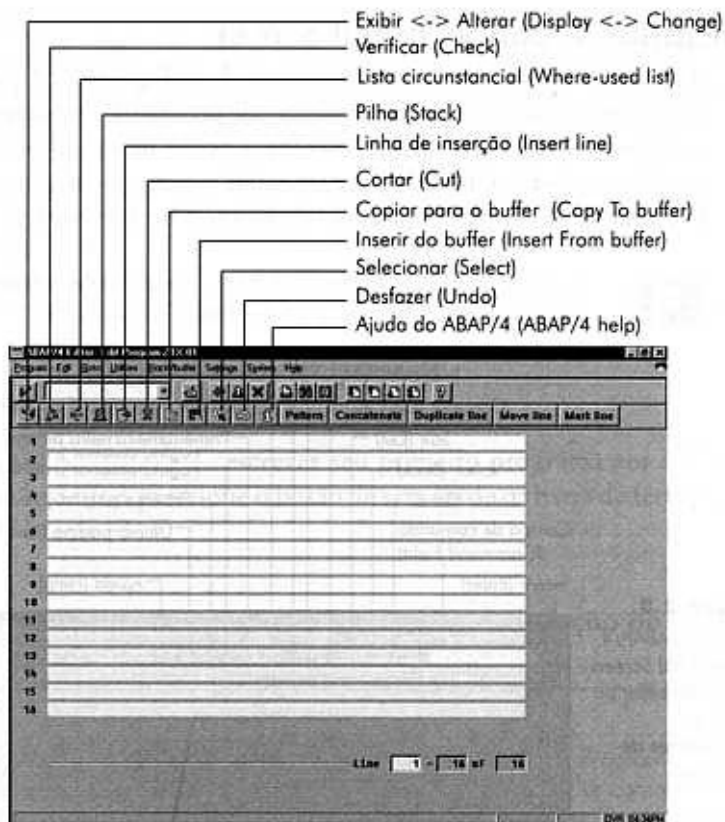
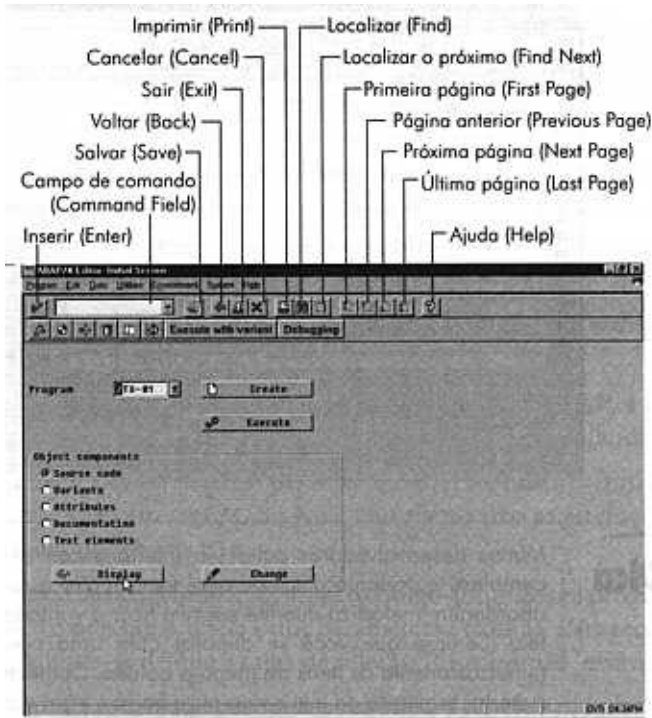
Cores

Para que você possa alterar as cores de seu ambiente tornando mais agradável, utilize o recurso abaixo demonstrado.



Ícones no SAP

Abaixo alguns exemplos, com descrição, dos ícones que você terá contato no SAP.

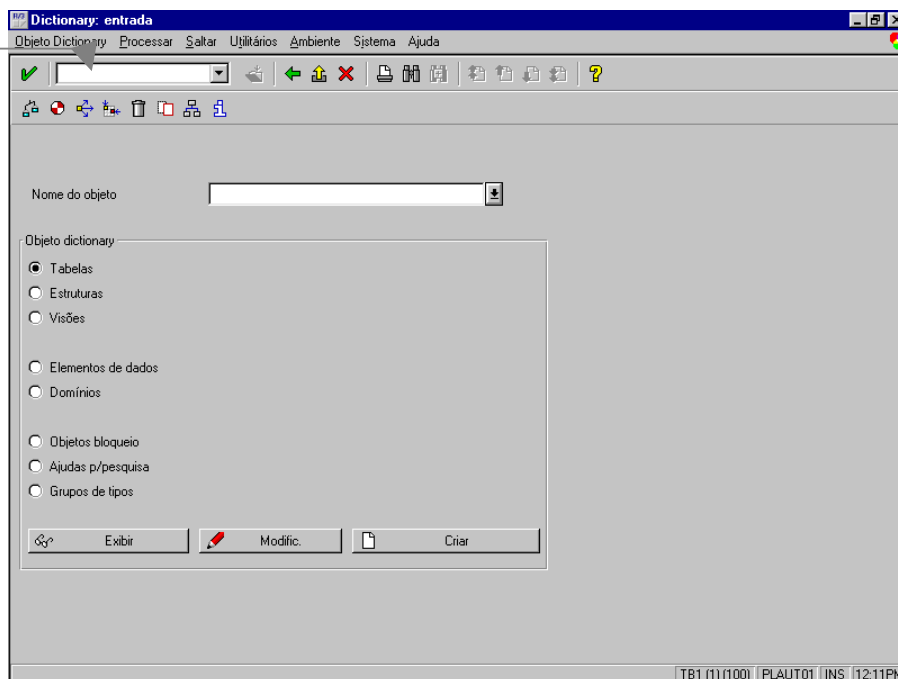


Dicionário

O Dicionário é um utilitário que define objetos de dados, nele você pode criar e armazenar objetos como tabelas, estruturas e visualizações.

Caminho: Clique no menu a opção “*Ferramentas*” > “*ABAP Workbench*” > “*Dictionar*”.

DICA: Se você digitar “SE11” no campo command a tela de entrada do ABAP Dictionary irá surgir.



Tabelas

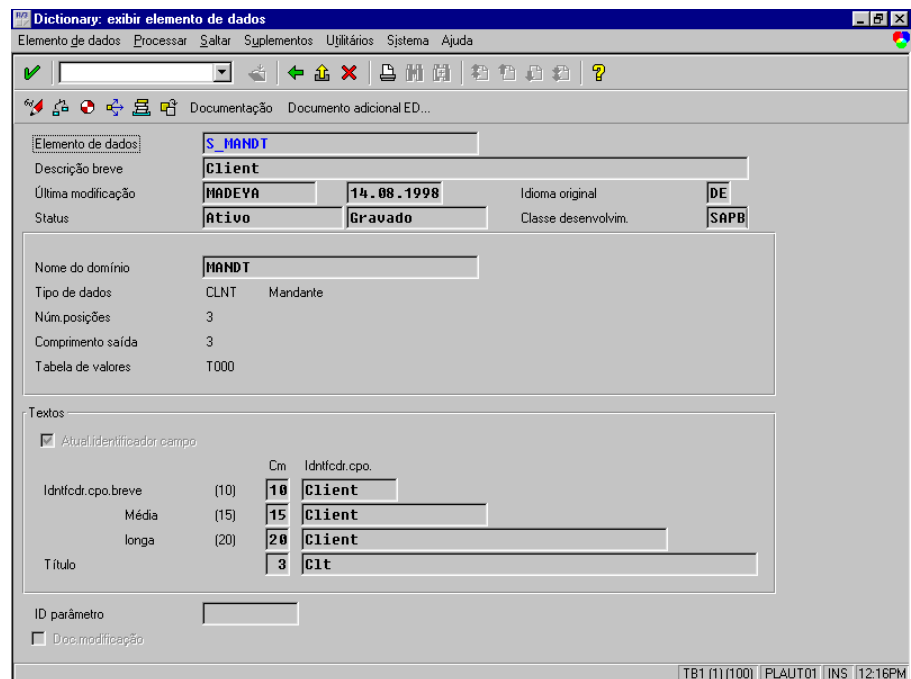
Dentro do R/3, uma **table** é uma coleção de linhas onde em cada linha contém campos ou também chamados de colunas. Em uma **table** são armazenados dados persistentes, se você colocar dados em uma **table**, eles continuarão ai mesmo depois de você finalizar seu programa e ali ficarão até seu ou outro programa altera-los ou exclui-los. O nome de uma **table** é único dentro de todo o sistema. Utilizaremos a **sflight** para exemplificar uma **table**.



Data element

Uma **table** é composta de campos, mas para criar um campo você precisa de um **data element** que contém os rótulos de campo e a documentação online (**FI**) para o campo. Um **data element** pode ser utilizado em mais de um campo e em mais de uma **table**.

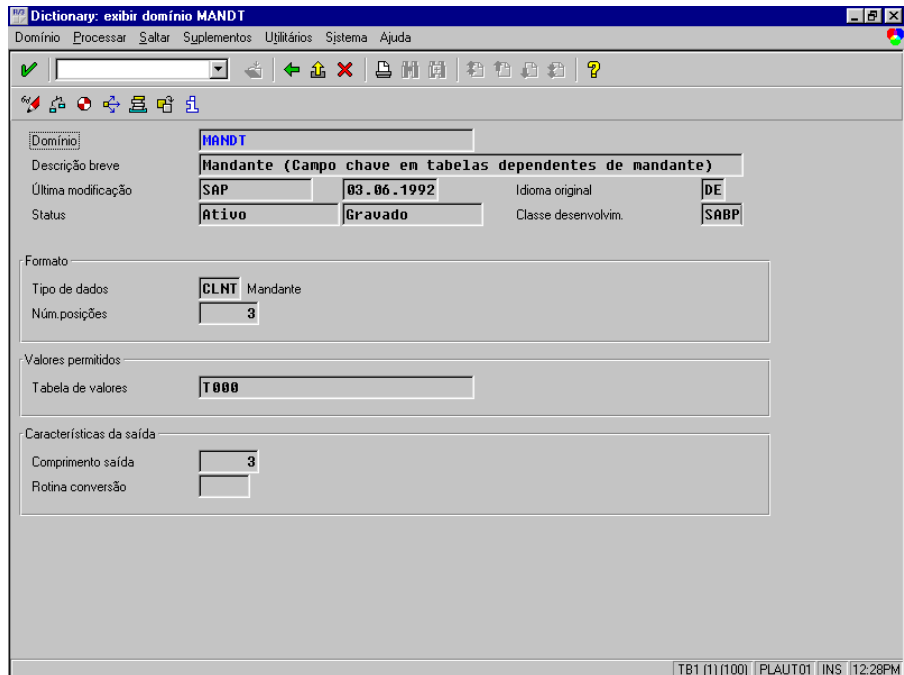
Para que você veja os **data element**, clique duas vezes com o mouse sobre o campo “*Elem. dados*”. Utilizaremos o campo **Mandt** para exemplificar.



Domínios

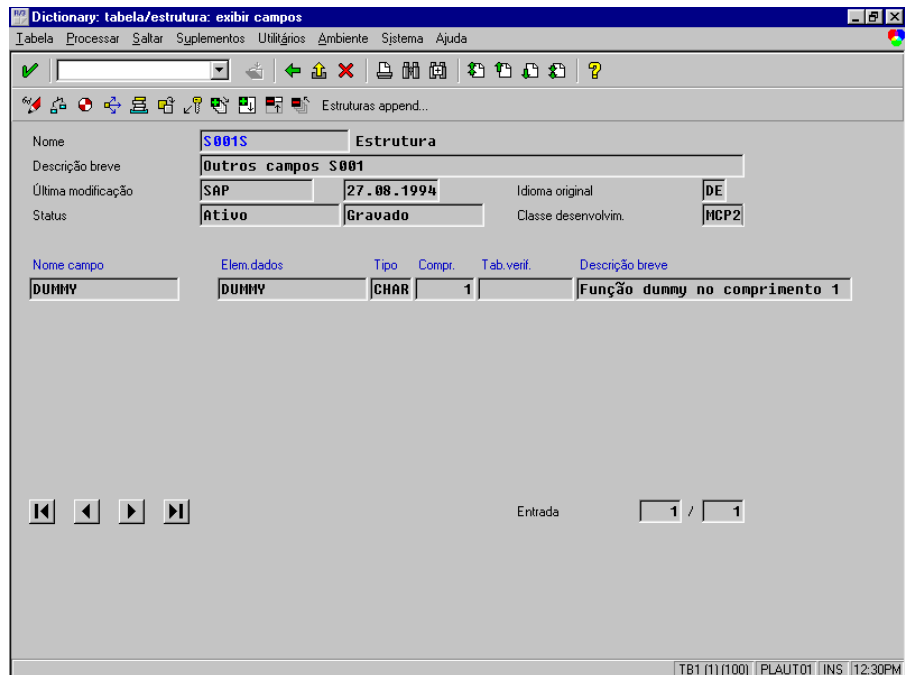
O domínio contém a característica técnica de uma campo, como o comprimento do campo e o tipo de dado. Os domínios também são reutilizáveis podendo ser usados em mais de um **data element**.

Para que você veja os **domínio**, aproveitaremos o exemplo da página anterior, clique duas vezes com o mouse sobre o campo “*Nome do domínio*”. Utilizaremos o campo **Mandt** para exemplificar.



Structure

Uma **structure** é uma descrição de uma série de campos agrupados sob um nome em comum. Ela descreve os nomes de campo, sua sequência e seus tipos e comprimentos de dados. Assim como a **table**, uma **structure** não terá outra com seu nome e nem com o de uma **table**.



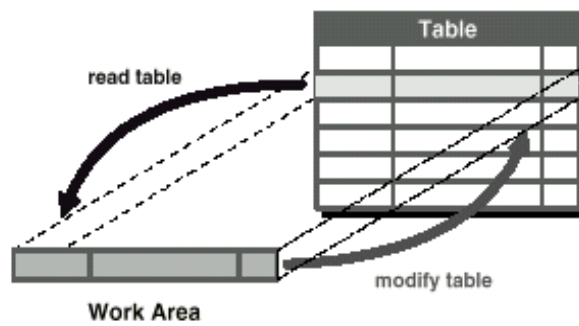
Uma estrutura pode agrupar dados de várias tabelas, ela difere de uma tabela transparente pois só matém os dados em tempo de execução do programa.

Tabelas Internas

Tabelas internas são utilizadas para armazenar em tempo de execução os dados necessários para a processamento do seu programa. Estes dados podem ser selecionados de diversas tabelas transparentes. O número de linhas de uma tabela não é fixo, a medida que a seleção é feita você pode acrescentar os dados na sua tabela.

O acesso a tabela interna é feito linha a linha utilizando uma work area como interface para transferir dados para ou de uma tabela, esta work area é utilizada para armazenar temporariamente uma entrada da tabela de dados.

Quando lemos uma tabela interna o conteúdo de endereçamento é enviado para uma work area. Esta work area tem a mesma estrutura da sua tabela interna.



REPORT

Através deste conceito temos dois tipos de tabelas internas.

WITH HEADER LINE – O sistema automaticamente cria uma work area idêntica a linha da sua tabela interna.

WITHOUT HEADER LINE – O sistema não reconhece a work area, que deve ser definida no programa.

table with header line

```
LOOP AT <itab>.  
.....  
  WRITE <itab>-<field>.  
.....  
ENDLOOP.
```

```
APPEND <itab>.  
MODIFY <itab>.  
READ TABLE <itab>.
```

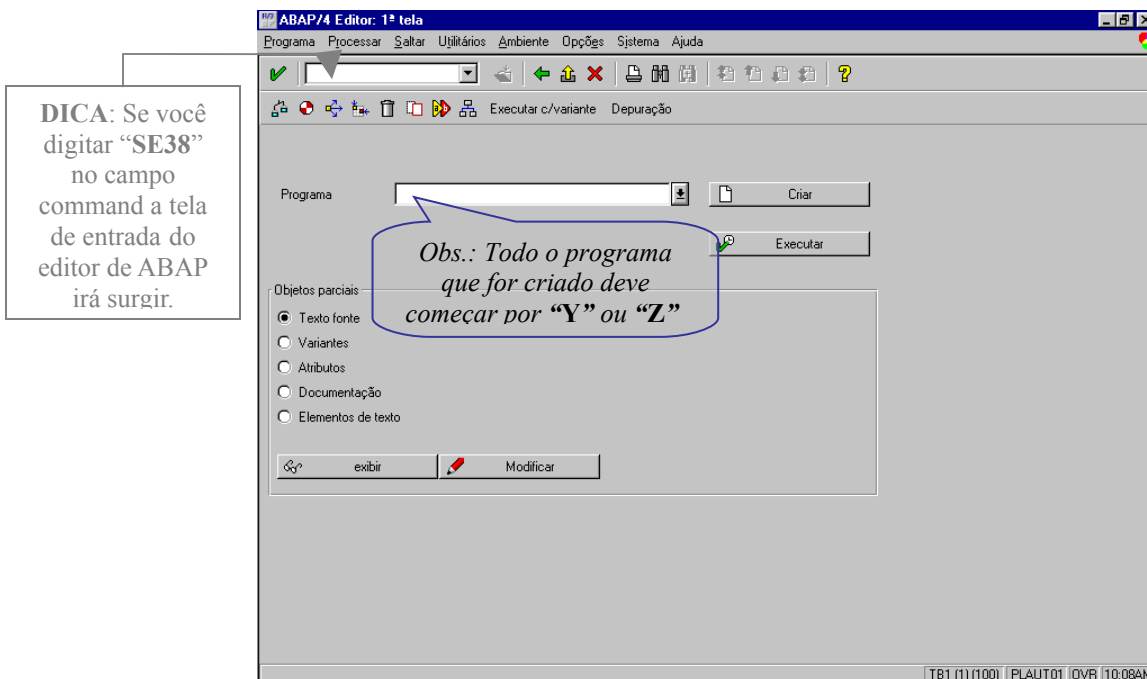
table without header line

```
LOOP AT <itab> INTO <wa>.  
.....  
  WRITE <wa>-<field>.  
.....  
ENDLOOP.
```

```
APPEND <wa> TO <itab>.  
MODIFY <itab> FROM <wa>.  
READ TABLE <itab> INTO <wa>.
```

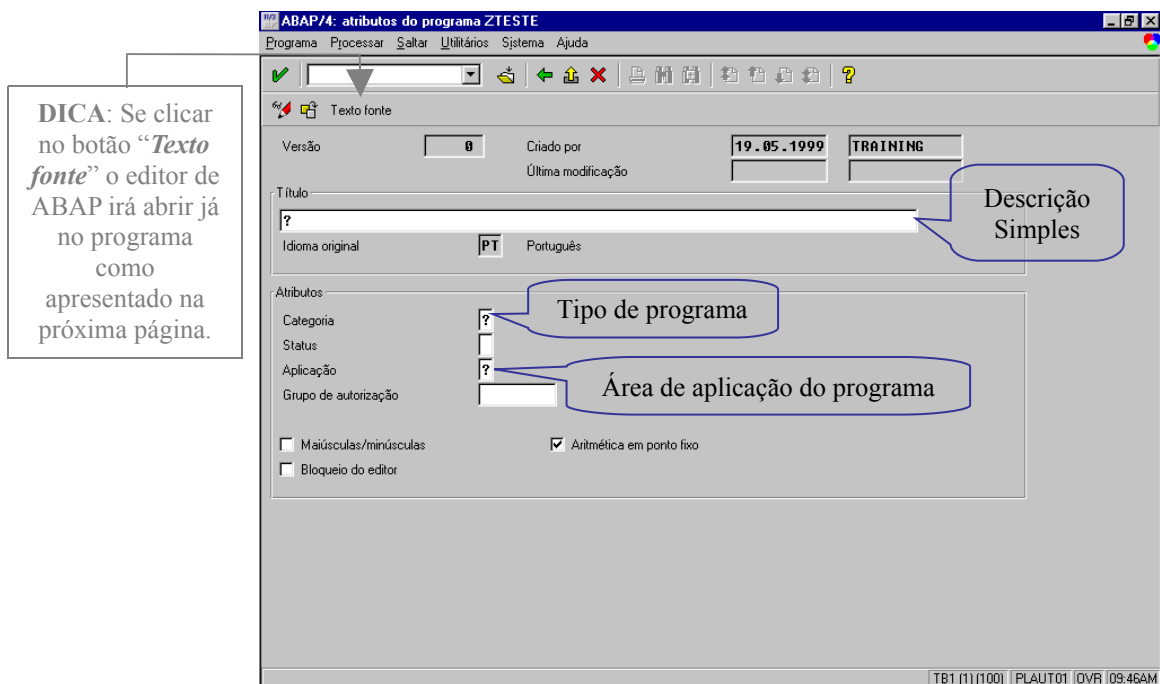
Criar Programas

- Caminho:** Clique no menu a opção “*Ferramentas*” > “*ABAP Workbench*”
Clique no botão “*Editor ABAP*”.
Chegando na tela do “*ABAP/4 Editor*”, deve-se entrar com o nome do programa que será criado.
Clique em criar.



Nesta tela você irá digitar os dados assinalados e clicar em salvar.

Abrirá uma tela menor “*Modificar entrada catálogo objetos*” onde você deverá preencher o campo “*Classe desenvolvim.*” com a sua classe.



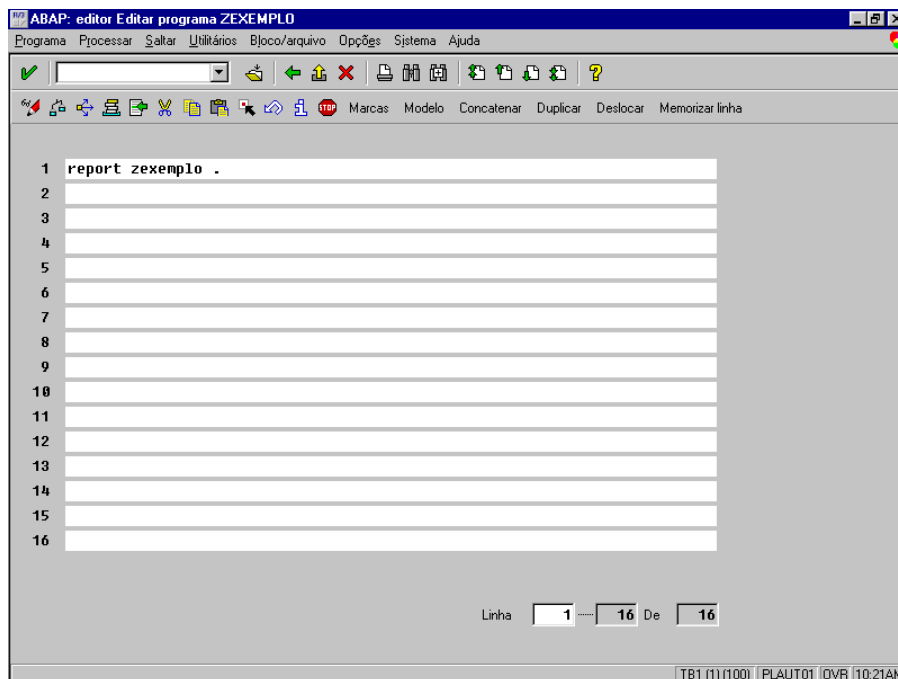
Obs.: Após clicar em salvar na tela “*Modificar entrada catálogo objetos*” uma outra tela surgirá solicitando o número da request, caso você já tenha criado uma request anteriormente, salve o seu programa nela, caso contrário crie uma nova.

Imp.: Uma request permite o transporte do seu programa de uma máquina de desenvolvimento para uma outra de produção fisicamente diferente, pois na mesma máquina os programas são vistos em todos os clients pois são independentes. Caso não haja necessidade de transporte pode-se criar como LOCAL OBJECT.

Retorne para a tela inicial e tecle “*Modificar*”.

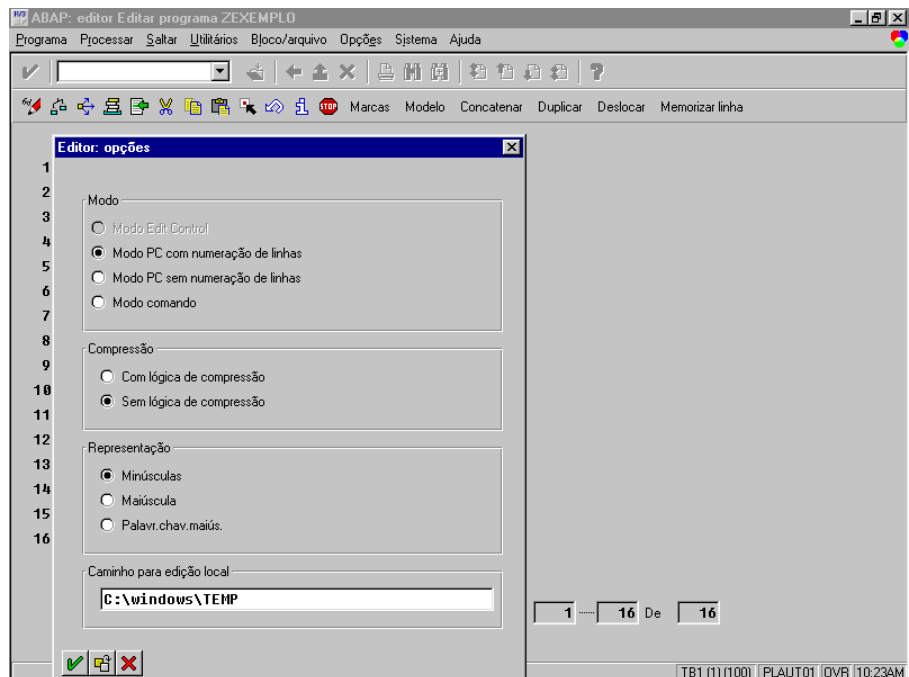
O editor ABAP será aberto.

A primeira linha indicará o nome do programa e é através desta linha que o sistema identificará e executará o programa ou por execução direta ou por um atalho (*Ex.*: Uma transação).



Tipos de Editor

Caminho: No menu clique em “*Opções*”
 Clique em “*Modo de edição...*”
 Abrirá a tela abaixo para que você selecione qual editor irá trabalhar.



Características:

- **Modo PC com numeração de linhas (PC mode with line numbers)**

Proporciona um estilo de “processador de texto” com comandos de cópia, recortar e colar mas com a numeração de linhas incluídas. Este é o modo default do SAP.

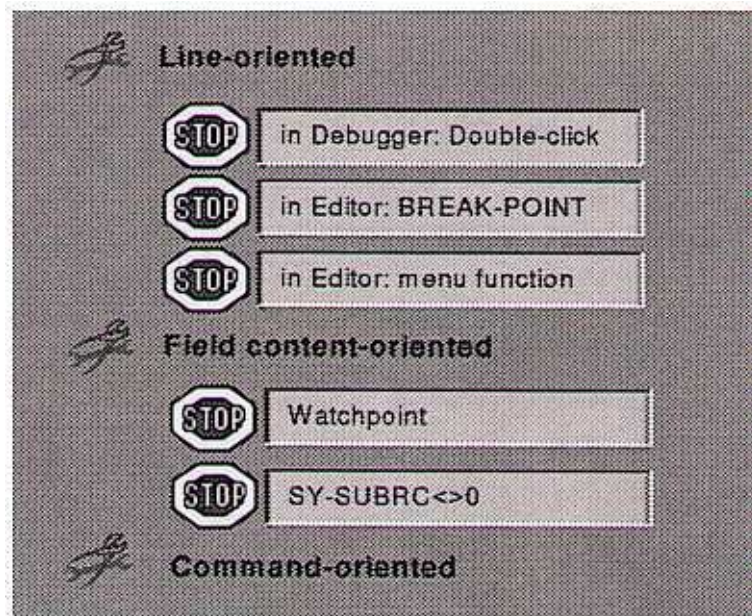
- **Modo PC com numeração de linhas (PC mode without line numbers)**

Proporciona um estilo de “processador de texto” com comandos de cópia, recortar e colar.

- **Modo Comando (Command mode)**

Proporciona a mesma funcionalidade associada a versão R/2 do SAP.

Breakpoint



Utilize o breakpoint quando for necessário analisar o programa linha a linha e verificar o acontecimento dos erros.

A função *Continue* no debugger executa comandos até o próximo ponto de breakpoint.

Pode-se colocar um ponto de breakpoint na posição atual do cursor selecionando a função *Set Breakpoint* ou através de duplo clique. Linhas onde um ponto de breakpoint for fixo estão marcadas com um símbolo de STOP. Exiba tudo sobre os breakpoint atualmente fixados, selecionando *Goto → Breakpoint*. Pode-se apagar pontos de breakpoint através de duplo clique, após posicionar o cursor em um deles ou usando o menu de ponto de breakpoint.

Utilize a declaração BREAK-POINT no Editor ABAP.

Posicione o cursor e então pode escolher *Utilities → Breakpoint → Set*.

Fixe pontos de breakpoint dependente em SY-SUBRC <> 0 através de *Breakpoint → Breakpoint at...*

Comandos orientados de breakpoint: Você pode fixar pontos de breakpoint através do ABAP para palavras-chave, eventos e seqüência de dados escolhendo *Breakpoint → Breakpoint at...*

Selecionar Tabelas

Deve-se declarar as tabelas, como demonstrado na linha 3, que serão utilizadas no programa.

Para leitura da base de dados (Tabelas) você usa o comando **select**.

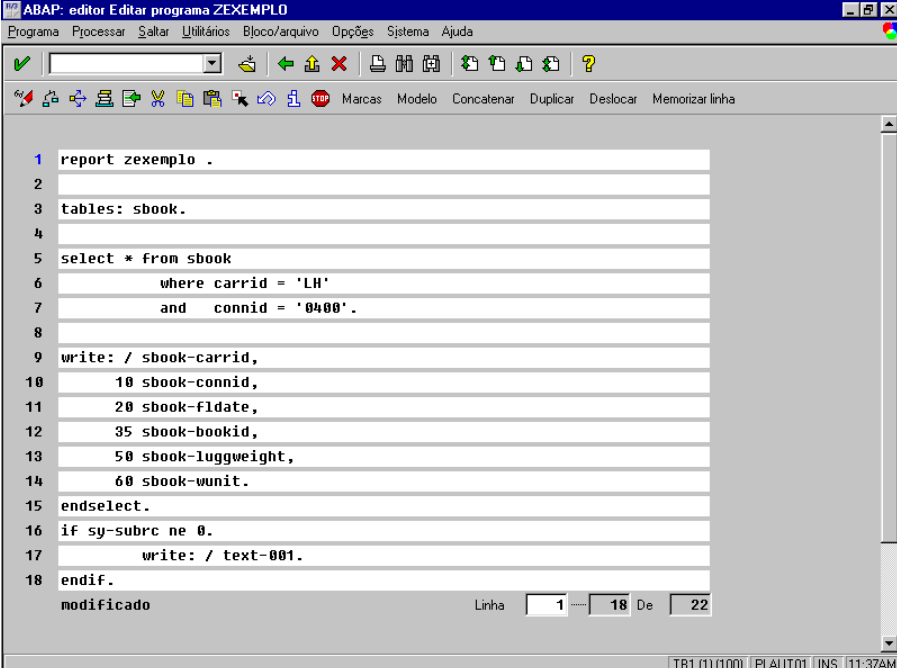
O **select** não é performático sem autorizações de checks.

Leitura de todos os dados de uma simples entrada.

Leitura de dados especificando colunas.

Na linha 5, você vê que foi utilizado o comando **select** para recuperar os registros dos bancos de dados, onde * indica que todos os campos na tabela devem ser recuperados.

Após a figura abaixo serão apresentadas outras opções de **select**.



```
1 report zexemplo .
2
3 tables: sbook.
4
5 select * from sbook
6         where carrid = 'LH'
7         and   connid = '0400'.
8
9 write: / sbook-carrid,
10       10 sbook-connid,
11       20 sbook-fidate,
12       35 sbook-bookid,
13       50 sbook-luggweight,
14       60 sbook-wunit.
15 endselect.
16 if sy-subrc ne 0.
17     write: / text-001.
18 endif.
modificado
```

Linha 1 De 22

TB1 (1) (100) | PLAUT01 | INS | 11:37AM

Sint.:

- **select single * from...** (seleciona um único registro)

–Este comando de **select** é usado para um acesso simples na entrada de dados. Você tem que especificar a chave completa da tabela, para obtenção de um único registro, para usar a chave use dentro do comando a condição **where...**

–Caso você não tenha especificado a chave completa, uma mensagem de Warning aparecerá no avisos do editor com erro de sintaxe (check sua performance), neste caso trará o primeiro registro encontrado na tabela.

–Para check use a variável do sistema **sy-subrc** (retorna 0 se a leitura foi feita com sucesso, retorna 4 se entrada não existe)

–Este comando não tem **endselect**.

- **select * from...**
endselect. (faz um processo de loop sem restrições)

–Este tipo de comando faz uma leitura sem restrições , seria um tipo de leitura seqüência de uma tabela, esse tipo de processo não é performático

–Este comando tem que ser finalizado com **endselect**.

–Para check use a variável do sistema **sy-subrc** (retorna 0 se a leitura foi feita com sucesso, retorna 4 se entrada não existe)

- **select * from...**
where...
endselect. (faz um processo de loop com restrições)

–Este tipo de comando faz a leitura com restrições que estão nas condições do **where**, segue também um processo de Loop tem uma performance muito maior que o anterior.

–Este comando tem que ser finalizado com **endselect**.

–Para check use a variável do sistema **sy-subrc** (retorna 0 se a leitura foi feita com sucesso, retorna 4 se entrada não existe)

–Operadores de comparação para comando **where** .

eq	=	
ge	>=	=>
le	<=	=<
ne	<>	><
gt	>	
lt	<	

REPORT

- `select <a1> <a2> ... into (<f1>, <f2>...) from....`
`where...`
`endselect.` (leitura de colunas simples)
 - Este comando seleciona somente os campos definidos `<a1> <a2>...` com a condição imposta no **where**.
 - Na adição do **into** após o **select** você especifica a sua área de trabalho no caso `<f1> <f2>...` com cada campo preenchido no processo de loop.
 - Os argumentos `<a1> <a2>...` tem que conter o mesmo numero de elementos para `<f1> <f2>...`

- `select max (distance)`
`min (distance)`
`count(*) from <table> into (... , ... , ...)`
`where...` (leitura de colunas agregando funções)
 - Este comando determina o numero total dos registros de uma tabela, retorna o valor máximo ,o valor mínimo, a quantidade total, você pode usar também as funções **avg** (média) e **sum** (soma)
 - Todas estas funções só podem ser usadas para campos numéricos.
 - Este comando não tem **endselect**.
 - Você pode também a opção adicional **distinct**, esta função agrega somente um registro o primeiro que encontrar, só poderá ser usado quando tiver diferentes valores nas colunas.

- `select into corresponding fields of <wa>`.(carrega os dados correspondente a work area definida)
 - Este comando carrega os dados numa area de trabalho que tem que ser definida, os nomes dos campos tem que ser iguais para que sejam transportado os dados.
 - Este comando não é recomendado devido a sua baixa performance pois demanda tempo na comparação dos campos.
 - Este comando não tem **endselect**.

- `select into corresponding fields of table <itab>`.(carrega os dados correspondente a uma tabela interna definida)
 - Este comando carrega os dados numa tabela interna que tem que ser definida, os nomes dos campos tem que ser iguais para que sejam transportado os dados.
 - Este comando sobrescreve os registros.
 - Este comando não é recomendado devido a sua baixa performance pois demanda tempo na comparação dos campos.

REPORT

–Este comando não tem **endselect**.

-
- **select into appendig corresponding fields of table <itab>**.(carrega os dados correspondente a uma tabela interna definida)

–Este comando acrescenta os dados numa tabela interna que tem que ser definida, os nomes dos campos tem que ser iguais para que sejam transportado os dados.

–Este comando não sobrescreve os registro e sim acrescenta novos registro

–Este comando não tem **endselect**.

-
- **select * from <table>**
where <table field>
between <field1>
and <field2>. (leitura de dados com limite de extensão)

–Este comando tem o mesmo processo de um select só que você define um processo de range onde **<field1>** e **<field2>** são os limites dos campos **<table field>**.

–Este comando não tem **endselect**.

-
- **select * from <table>**
where <table field>
like...“...” (leitura de dados com pesquisa)

–Este comando permite que você faça uma pesquisa de campos definida no LIKE usando caracteres especiais.

–Caracteres especiais (coringas) “_”, “%”, você só pode usar esta procura para campos caracteres.

Sint.:

```
...
select * from mara where matnr like '%ca'.
...
```

-
- **select * from <table>**
where <table field>
in (... , ...) (leitura de dados com lista)

–Este comando é usado na clausula **where** para comparação de lista de valores ou valores únicos, os campos não são um intervalo e sim valores fixos

REPORT

- `select * from <table>`
`where <table field>`
`in <itab>` (leitura de dados com operador de uma tabela)

–Este comando é usado na clausula **where** para comparação de lista dentro de uma tabela interna que contenha valores fixos
–A tabela interna deve sempre incluir os campos **sign, option, low, high**.

- `select * from <table>`
`where <table field>`
`in <work area>` (leitura de dados com operador de uma work area)

–Este comando é usado na clausula **where** para comparação de um único parâmetro dentro de uma work area. O conteúdo de um campo definido.

- `select * from (<table>)`
`where (<itab>)` (leitura de dados dinâmica)

–Este comando permite que você crie uma tabela interna com dados montados contendo uma linha com 72 caracteres (seriam dados constantes nessa linha podendo conter parênteses aspas campos em brancos enfim qualquer tipo de caracter. Ele retorna na condição **where** os dados contidos na tabela interna
–O nome das tabelas devem estar entre parênteses com branco separando os nomes. Este tipo de comando não retorna erro de sintaxe.

- `select * from (<table>)`
`into table <itab>`

–Este processo e bastante rápido para leitura de dados de tabela interna, só que não sobrescreve nenhum registro, faz o processo de loop e busca cada registro na tabela interna um a um.
–Este comando não tem **endselect**.

- `select * from (<table>)`
`appending table <itab>`

–Este processo e bastante rápido para leitura de dados de tabela interna, só que acrescenta os registros , faz o processo de loop e busca cada registro na tabela interna um a um.
–Este comando não tem **endselect**.

-
- `select * from (<table>)`
 `for all entries in <itab>`
 `where...`

- Este processo seleciona todos os registros para a tabela interna.
- Se a tabela interna não contem dados não use a condição **where**.
- Os campos da tabela transparente e da tabela interna tem que ser iguais.
- Esta condição **for all entries** exclui a adição do comando **order by...**

-
- `select * from (<table>)`
 `order by <field1> <field2>`
 primary key (leitura de dados com seleção de ordenação)

- Este comando permite que você selecione os dados na ordem da chave primaria da tabela em ordem ascendente ou os campos definidos (<field1>, <field2>), também em ordem ascendente
- Você também pode acrescentar a ordem **ascending** ou **descending**, o default e a ordem ascendente
- Você pode também fazer a ordem de uma tabela interna, neste caso deve conter a lista de campos <f1>, <f2>....
- As entradas da tabela interna tem ser do tipo caracter e no máximo com 72 caracteres.

-
- `select <a1> <a2> ... into (<f1>, <f2>...)`
 `from.<table>`
 group by...

- Este comando permite um grupo de entrada do conteúdo de um campo.
- Ele determina o máximo e o mínimo <a1> <a2> de um determinado campo.

-
- `select * from (<table>)`
 bypassing buffer

- Este comando permite acesso por um determinado período não usando a area de memória.
- Você tem que ter a certeza da leitura dos dados.
- Geralmente esse comando é usado para tabelas em tela em tempo de execução.

DICA: Utilize o botão modelo na execução do programa para gerar um select demonstrando que não haverá erro na ordem da seleção dos campos chaves.

Get

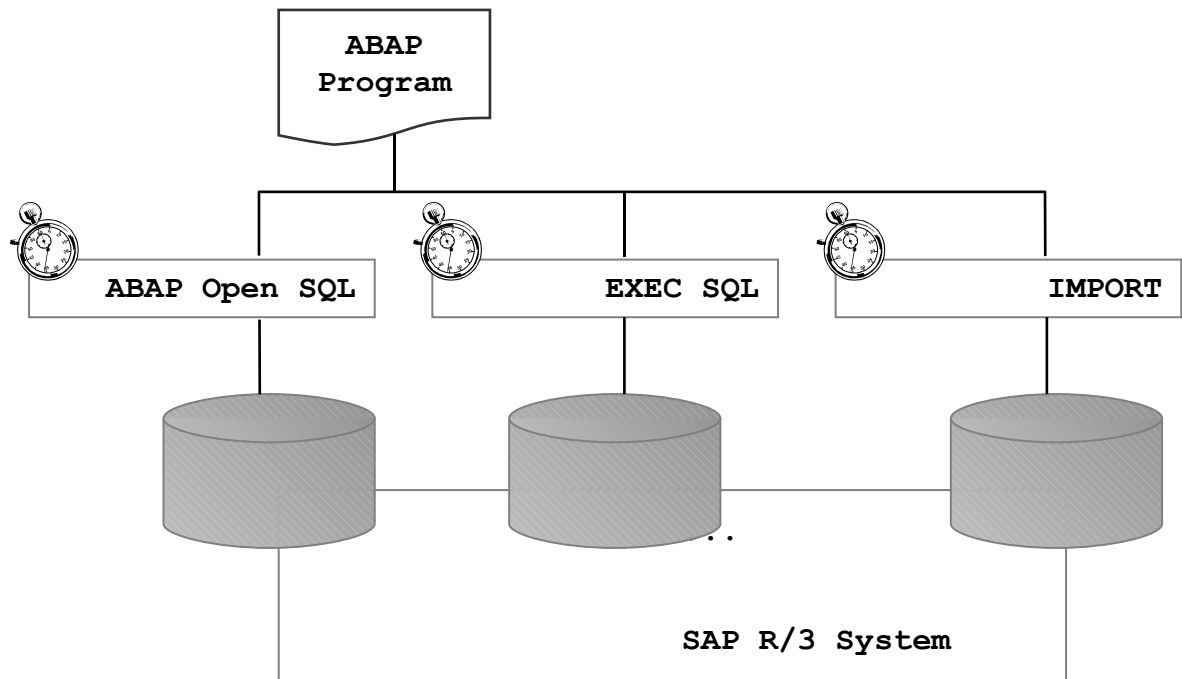
Se em uma base de dados lógica for necessário a seleção de campos para uma tabela, você pode utilizar a declaração GET para especificar somente os campos necessários para executar o processo.

Ex.:

```
report zexemplo.  
tables: spfli, sflight, sbook.  
get spfli fields carrid connid cityfrom cityto deptime aritime.  
    write: spfli-carrid, spfli-connid, spfli-cityfrom, spfli-cityto,  
          spfli-deptime, spfli-aritime.  
get sflight fields fldate price currency.  
    write:/10 sflight-fldate, sflight-price, sflight-currency.  
get sbook fields bookid customid custtype.  
    write:/20 sbook-bookid, sbook-customid, sbook-custtype.
```

Não é recomendado a utilização deste comando por comprometer a performance.

Acessando a base de dados



Utilize a declaração EXEC SQL quando for necessário uma intervenção direta por parte do usuário na base do banco de dados.

Assim que a declaração EXEC SQL for processada em seu programa ABAP, ela torna o programa ABAP inoperante até que a declaração ENDEXEC seja processada.

Sint.:

```
...  
exec sql.  
    <sql statement>  
endexec.  
...
```

Exibir Dados – write

Quando se faz necessário a exibição na tela do resultado de um **report** você utiliza o comando **write** e todo o conteúdo que estiver após a instrução até ser finalizado por “.” gerará um relatório de saída. Como no exemplo abaixo, o **write** irá fornecer uma listagem de dados que foram selecionados dentro da tabela **sflight** produzindo um resultado como demonstrado abaixo.

Ex.:

```
report zexemplo.
tables: sbook.
select * from sbook
  where carrid = 'lh'
     and connid = '0400'.
write: / sbook-carrid,
       10 sbook-connid,
       20 sbook-fldate,
       35 sbook-bookid,
       50 sbook-luggweight,
       60 sbook-wunit.

endselect.
if sy-subrc ne 0.
  write: / 'não existem entradas para esta seleção.'.
endif.
skip 5.
uline.
skip.
```

Carrid	Connid	Fldate	Bookid	Luggweight
LH 0400	28.02.1995	00000001	20,0000	KG
LH 0400	28.02.1995	00000002	20,0000	KG
LH 0400	28.02.1995	00000003	30,0000	KG

Eventos – inicialization, start-of-selection

Um evento é como uma sub-rotina em outras linguagens, trata-se de uma seção independente do código que executa uma tarefa e então retorna para o ponto de chamada, contudo, ao contrário das sub-rotinas, você não codifica a chamada para um evento, ao contrário, o sistema aciona o evento para você quando uma condição específica surge. Um evento é um tag (contador) que identifica uma seção de código, que é associada com um evento começa com um nome de evento e termina quando o próximo nome é encontrado. Os nomes dos eventos são palavras reservadas, não se podendo criar novos eventos. Os nomes dos eventos são:


inicialization

Quando você executa um programa onde um critério de seleção é definido, o sistema normalmente processa a tela primeiro. Se você quer que um bloco de comandos seja executado antes da tela de seleção utilize este comando.

Sint.:

Antes do comando:


```
report sapmztst.  
parameters firstday like sy-datum default sy-datum.  
tables spfli.
```

Carrier ID	<input type="text"/>	To	<input type="text"/>	
From	<input type="text"/>			
To	<input type="text"/>			
FIRSTDAY	<input type="text" value="01/08/1996"/>			

REPORT

Após o comando

```
report sapmztst.
parameters firstday like sy-datum default sy-datum.
tables spfli.
initialization.
  city_fr = 'new york'.
  city_to = 'frankfurt'.
  carrid-sign   = 'i'.
  carrid-option = 'eq'.
  carrid-low    = 'aa'.
  append carrid.
  firstday+6(2) = '01'.
```

Carrier ID	AA	To	<input type="text"/>	
From	NEW YORK			
To	FRANKFURT			
FIRSTDAY	01/01/1996			

Sint.:

```
report zexemplo.
tables sbook.
select-options fl_date for sbook-fldate.
initialization.
  move: 'I'      to fl_date-sign,
        'EQ'     to fl_date-option,
        sy-datum to fl_date-low.
  append fl_date.
  move: 'BT'     to fl_date-option,
        '19960101' to fl_date-low,
        '19960630' to fl_date-high.
  append fl_date.
```

start-of-selection

Se a primeira instrução executável em seu programa não for procedida por um nome de evento, o sistema automaticamente insere **start-of-selection** antes da primeira linha de código executável. Permite o processamento antes da execução de leitura das tabelas transparentes através do comando sempre acompanhado pelo comando **end-of-selection**.

Posicionamento de dados no relatório – `u-line`, `skip`

Especificações para o comando `write`.

`u-line`

Escreve um traço no relatório de saída ou tela de seleção

`skip`

Gera linhas em branco, para mais de uma linha em branco (*Ex.: `skip n`*), também podendo deslocar-se para outro ponto de saída do relatório (*Ex.: `skip n to line n`*).

Sint.:

```
...  
WRITE: 'Text 1 .....'.  
      SKIP.  
WRITE: 'Text 2 .....'.  
...
```

Resultado:

```
Text 1 .....
```

```
Text 2 .....
```

- `/p(1) → / →` pula uma linha, **p** (posição da coluna), **1** (deslocamento a esquerda).

Sint.:

```
...  
write: /5 (30) field,  
...
```

Comandos de saída do programa

stop

Este comando interrompe o processo de leitura e vai para o final do evento **end of selection**, caso não tenha usado o evento ele interrompe o programa, também podendo ser utilizado para interromper a execução de um **loop** indo direto para o **endloop**.

exit

Termina o **loop** imediatamente após a sua utilização, e o processo somente continua na instrução imediatamente após o **endloop**. Normalmente é utilizado quando se quer testar condições dentro do loop.

Ex.:

```
report exemplo.
tables t100.
data sap_count type i.
select * from t100 where sprsl = sy-langu and
                        argb = 'ds'.

  write / t100-text.
  if t100-text cs 'sap'.
    add 1 to sap_count.
    if sap_count = 3.
      exit.
    endif.
  endif.
endselect.
```

Novas páginas – new page

Você vai utilizá-lo para causar a saída do comando `write` para gerar uma nova página. A paginação do sistema é automática, especificado no cabeçalho do relatório.

Sint.:

```
...
new-page [no title | with-title]
         [no heading | with-heading]
         [line-count o n (m)]
         [line-size k]
         [print on | print off]
...
```

–*n*, *m* e *k* são variáveis numéricas ou literais.

–**no title** desativa o título padrão nas páginas seguintes e **with-title** ativa.

–**no heading** desativa o título cabeçalho-padrão de coluna nas páginas seguintes e **with-heading** ativa.

–**line-count** configura o número de linhas por página e o número de linhas reservado na parte inferior de cada página para um rodapé.

–**line-size** configura o número de colunas de saída para as páginas seguintes.

–**print on** faz com que a saída das instruções **write** sejam enviadas para o spool em vez da lista, só permitindo a visualização do relatório através do spool e o **print off** faz o contrário.

O comando **new-page** não aciona o evento **end-of-page**.
Ele pode ser utilizado em qualquer ponto do programa.

MOVE

Para mover um valor de um campo para outro, utilize o comando move. Parte do conteúdo ou todo ele pode ser movido. Ao invés de move é possível utilizar o operador de atribuição =.

Sint.:

```
...  
data: number type i,  
      five   type i.  
move 5 to five.  
move five to number.  
...
```

Do

O comando do correspondente a um mecanismo de loop básico.

Sint.:

```
...
do.
  write: / 'sy-index - begin:', (3) sy-index.
  if sy-index = 10.
    exit.
  endif.
  write: 'end:', (3) sy-index.
enddo.
...
```

Tabelas de gerenciamento do sistema operacional

Algumas variáveis foram criadas para validar a execução e entrada de dados no sistema, estas variáveis encontram-se na estrutura SYST.

Ex.:

sy-subrc Valor de retorno de acordo com determinadas instruções ABAP/4

sy-datum Sistema: Data do dia do logon

sy-uname Sessão: Usuário SAP no SAP logon

sy-langu Código de idioma do logon no sistema SAP

...

Case

O comando `case` executa uma série de comparações onde somente as instruções após a primeira correspondência `when` serão executadas. Sendo muito semelhante a `if/elseif`, sendo sua única diferença que em cada `if/elseif` você pode especificar uma expressão complexa e em `case` só pode especificar um único valor a ser comparado e os valores são sempre comparados quanto a igualdade. As strings de campo são tratadas como sendo variáveis de tipo C

Sint.:

```
...
data: one    type i value 1,
      three type p value 3.
      four   type p value 4.
do 4 times.
  case sy-index.
    when one.
      write / 'that is'.
    when 2.
      write  'a'.
    when three.
      write 'good'.
      write 'example'.
    when others.
      write '!'.
  endcase.
enddo.
...
```


Concatenate

Une duas ou mais linhas ou campos, você terá de informar onde os dados serão posicionados.

Sint.:

```
...
data: one(10)   value 'John',
      two(3)    value 'F.',
      three(10) value 'Kennedy',
      name(20) .
concatenate one two three into name separated by space.
...
```

Resultado:

John F. Kennedy

Condense

Utilize o comando condense para unir campos havendo a necessidade de deslocá-los da posição em que se encontram.

Ex.:

```
report zexemplo.
```

```
data name(30).  
    name(10)      = 'Dr.'.  
    name+10(10)  = 'Michael'.  
    name+20(10)  = 'Hofmann'.  
condense name.  
write name.
```

Resultado:

Dr. Michael Hofmann

Format color / Write color

							INPUT	
No.	Color	INTENSIFIED	INTENSIFIED OFF	INVERSE	INT.	INT. OFF		
0	COL_BACKGROUND	0123456789	0123456789	0123456789	0123456	0123456		
1	COL_HEADING	0123456789	0123456789	0123456789	0123456	0123456		
2	COL_NORMAL	0123456789	0123456789	0123456789	0123456	0123456		
3	COL_TOTAL	0123456789	0123456789	0123456789	0123456	0123456		
4	COL_KEY	0123456789	0123456789	0123456789	0123456	0123456		
5	COL_POSITIVE	0123456789	0123456789	0123456789	0123456	0123456		
6	COL_NEGATIVE	0123456789	0123456789	0123456789	0123456	0123456		
7	COL_GROUP	0123456789	0123456789	0123456789	0123456	0123456		

Para diferenciar ou destacar uma ou mais linhas você utiliza os comandos `format color` e `write color`.

Sint.:

```

...
loop at i_tab.
  if v_flag is initial.
    format color 2 intensified on.
    v_flag = 'X'.
  else.
    format color 2 intensified off.
    clear v_flag.
  endif.
write: /001 i_tab-tabname(15),
      017 i_tab-fieldname(15),
      034 i_tab-rollname(15),
      051 i_tab-domname(15),
      068 i_tab-datatype,
      074 i_tab-leng no-zero,
      082 i_tab-decimals no-zero,
      090 i_tab-tabclass.
endloop.

```

ABAP

REPORT

...

1.00 - 44

Símbolos e ícones

Você tem a opção de criar símbolos e ícones para os seus programas.

Um `symbol` é uma figura bicolor simples de um símbolo comum como um círculo, quadrado, uma pasta ou documento ocupando em sua maioria apenas um caracter na lista de saída, tendo alguns ocupando dois caracteres; já um `icon` é semelhante a um símbolo exceto por ser multicolorido, chegando a ocupar até mais que dois caracteres.

Sint.:

```
...  
INCLUDE <SYMBOL>.  
WRITE: / SYM_RIGHT_HAND AS SYMBOL,      " output as symbol  
        'Tip, Note',  
        SYM_LEFT_HAND AS SYMBOL.      " output as symbol  
...
```

```
...  
INCLUDE <ICON>.  
WRITE: / ICON_OKAY AS ICON,             "output as icon  
        'Text line'.  
...
```

Títulos dos relatórios – text elements

Caminho: Clique no menu a opção “*Ferramentas*” > “*ABAP Workbench*”
Clique no botão “*Editor ABAP*”.

Chegando na tela do “*ABAP/4 Editor*”, selecione a opção “*Text-elements*” e clique em criar ou modificar

Você pode definir os elementos de textos num relatório.

Você edita um título; define um cabeçalho; define colunas.

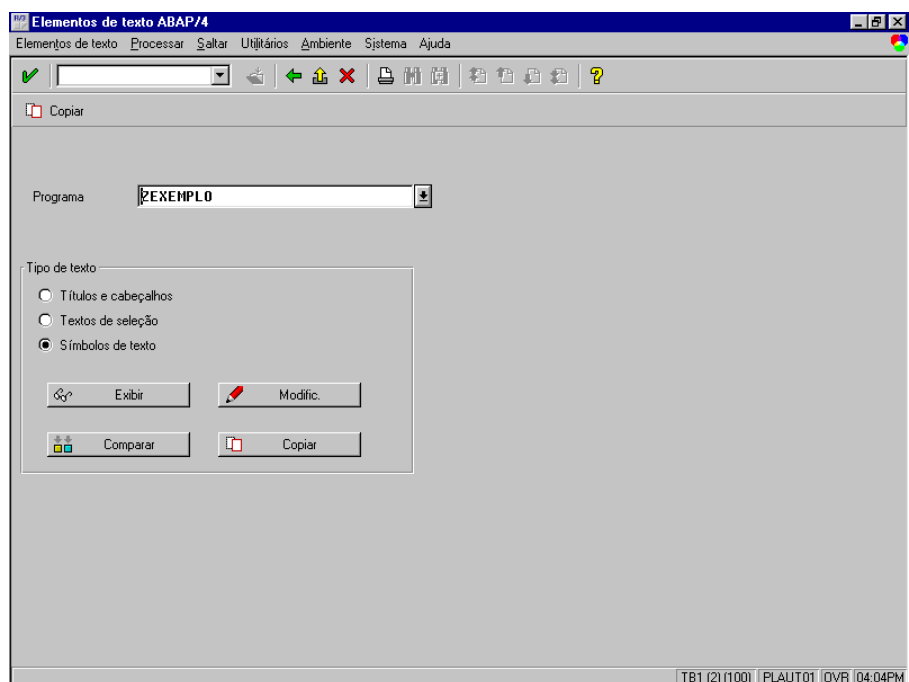
Sint.:

...

```
write va, text-001.
```

...

–**text-001** é o elemento de texto que você trabalhará.



Reservando linhas – reserve

Você, as vezes, pode necessitar escrever de maneira agrupada informações relacionadas que se distribuem por várias linhas, para isso você pode utilizar o comando **reserve**, que reservará linhas em um relatório para que não haja quebra indevida de informação.

Quando o comando **reserve** é executado, o sistema verifica se há n^* linhas disponíveis na página atual, se houver menos que n linhas sobrando na página atual, uma quebra de página ocorrerá acionando o **end-of-page** (se existir), seguido pelo comando **top-of-page** (se existir) fazendo a instrução write começar na parte superior da nova página. Se pelo menos n linhas não estiverem disponíveis quando a instrução **reserve** for executada, o código não fará nada.

* n é um literal numérico e variável.

Sint.:

```
...
select * from <table>.
    reserve 3 lines.
    write: / <table-c1>, <table-c2>,
           / <table-c3>, <table-c4>, <table-c5>,
           / <table-c6>.

skip.
endselect.
top-of-page.
    write / 'Relatório'.
    uline.
...
```

--<table> é uma tabela.

--<table-cn> são os campos da tabela.

Resultado:

Relatório

11111 São Paulo
Rua São Paulo, 9999
011-999-9999

11112 Rio Grande do Sul
Rua Porto Alegre, 9999
041-999-9999

Ex.:

```
1 report zexemplo .
2
3 tables: sflight.
4
5 select * from sflight.
6   reserve 2 lines.
7   write: / sflight-carrid, sflight-connid,
8         / sflight-fldate, sflight-price.
9   skip.
10  endselect.
11
12 top-of-page.
13   write / 'Flight'.
14   uline.
15
16
17
18
```

modificado Linha 1 De 18

TB1 (1) (100) | PLAUT01 | INS | 03:13PM

ZEXEMPLO 1

Flight

LH 0400	28.02.1995	899,00
LH 0454	17.11.1995	1.499,00
LH 0455	06.06.1995	1.090,00
LH 3577	28.04.1995	6.000,00
SQ 0026	28.02.1995	849,00

TB1 (1) (100) | PLAUT01 | INS | 03:18PM

Formato de páginas

Você pode usar parâmetros para controlar o tamanho da página do relatório, para isso você pode utilizar os seguintes comandos:

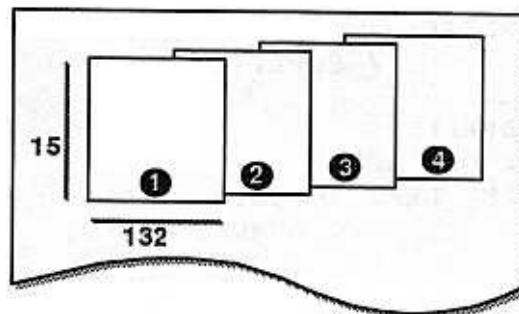
- **line-size** Define a largura da página
- **line-count** Define o número de linhas por página.

Sint.:

```
report zteste line-size 132 line-count 15.
```

```
tables...
```

```
...
```



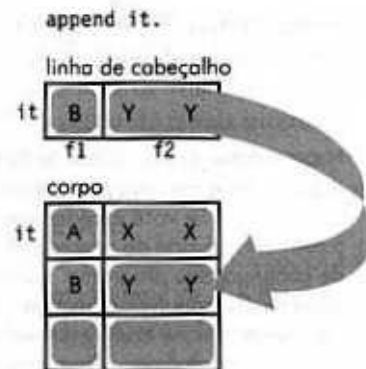
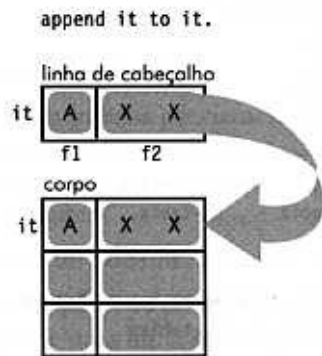
Incluindo dados – append

Para adicionar uma única linha em uma tabela interna, você pode utilizar o comando **append** ele copia uma única linha de qualquer área de trabalho e a coloca no corpo ao final das linhas existentes, podendo ser a linha de cabeçalho ou qualquer outra **string** de campo com a mesma estrutura de uma linha no corpo.

Sint.:

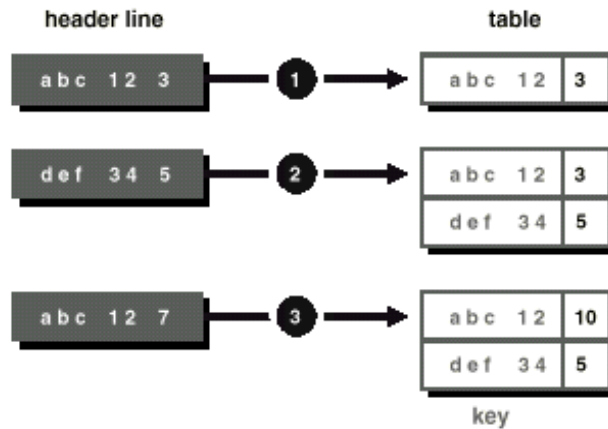
Append wa to it.

- **wa** é o nome de uma área de trabalho
- **it** é o nome de tabela interna.



Agrupando dados – collect

Armazena os dados somando os campos numéricos de uma tabela interna, assegurando que não haverá duplicidade de registros. Quando o **collect** for executado, o sistema formará uma chave a partir dos campos de chave padrão da área de trabalho, o sistema pesquisará o corpo da tabela interna quanto uma linha com uma mesma chave daquela na área de trabalho.



Classificar os dados – sort

Para classificar o conteúdo de uma tabela interna você pode utilizar o comando sort, as linhas podem ser classificadas por uma ou mais colunas em ordem ascendente ou descendente, sendo que a própria sequência de classificação pode ser alterada.

Ex.:

```
report zexemplo.
data: begin of person occurs 5,
      name(10)   type c,
      age        type i,
      country(3) type c,
      end of person.
person-name     = 'Muller'. person-age = 22.
person-country = 'USA'.   append person.
person-name     = 'Moller'. person-age = 25.
person-country = 'FRG'.   append person.
person-name     = 'Möller'. person-age = 22.
person-country = 'USA'.   append person.
person-name     = 'Miller'. person-age = 23.
person-country = 'USA'.   append person.
sort person.
```

Resultado:

```
Miller 23 USA
Moller 25 FRG
Muller 22 USA
Möller 22 USA
```

Ler os dados – loop at

Para ler algumas ou todas as linhas de uma tabela interna, você pode utilizar a instrução `loop at`, ela lê o conteúdo da tabela interna, colocando as linhas do conteúdo, uma por vez, na área de trabalho. O loop termina automaticamente quando a última linha é lida e a instrução após `endloop` é executada.

Sint.:

```
...  
loop at itab into struc.  
    write: / struc-name, struc-blnce.  
endloop.  
...
```

```
...  
loop at itab into struc where blnce <> 0.  
    write: struc-name, struc-blnce.  
endloop.  
...
```

Controle de quebras

at first, at last

Você pode utilizar estes comandos para executar um trecho entre a primeira e a última passagem do loop por uma tabela interna.

Obs.: Todos estes comandos devem ser finalizados com **endat**.

Sist.:

```
...
loop at itab.
  ...
  at first.
    ...
    endat.
  ...
  at last.
    ...
    endat.
  ...
endloop.
...
```

at new, at end of

Utilize estes comandos para detectar uma alteração em uma coluna de uma passagem de loop para a seguinte.

Obs.: Todos estes comandos devem ser finalizados com **endat**.

Sist.:

```
...
sort by c.
loop at itab.
  ...
  at new c.
    ...
    endat.
  ...
  at end of c.
    ...
    endat.
  ...
endloop.
...
```


Seleção Simples – parameters

Este comando quando executado em um programa gera um campo de entrada de dados numa tela de seleção antes do programa realmente em si ser executado, esta tela é onde você digitará a informação que será a chave para a seleção dos dados do seu programa.

Sint.:

```
parameters p1 [(1)] [type t] [decimals d]...
```

–**p1** é o nome do parâmetro.

–**1** é a especificação interna de comprimento.

–**t** é o tipo de dado

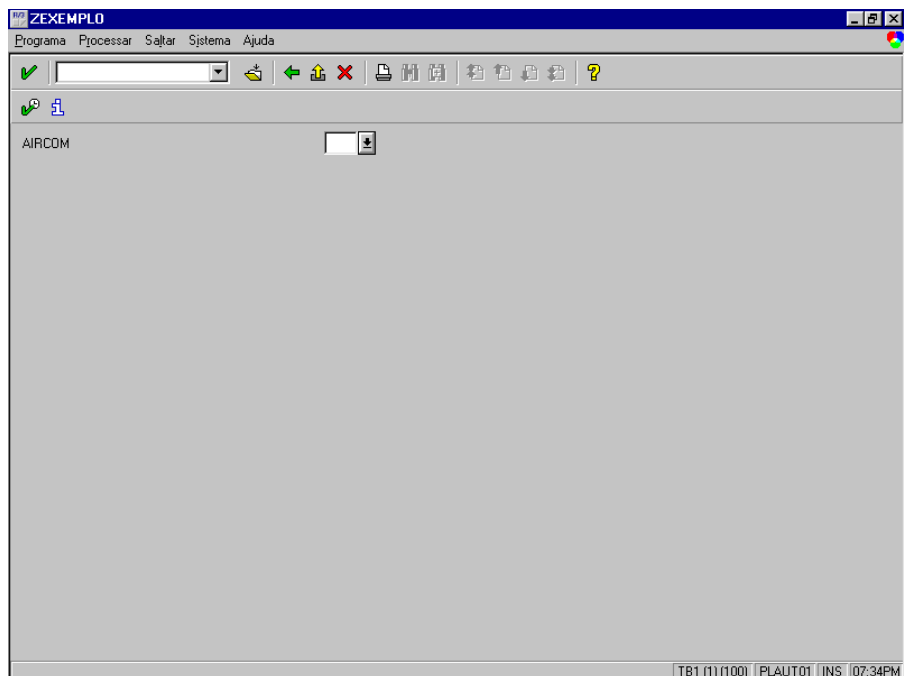
–**d** é o número de espaços decimais (utilizado apenas como tipo p).

Ex.:

```
report zexemplo
```

```
parameters aircom like spfli-carrid.
```

Resultado:



Seleção com um intervalo – select-option

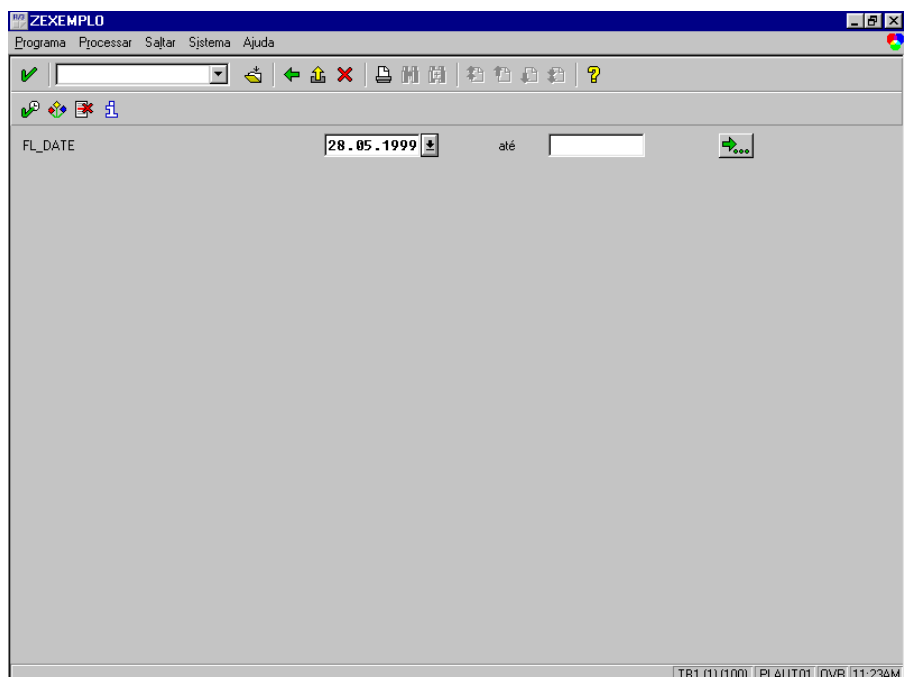
Semelhante ao comando **parameters**, o comando **select-options** cria critérios de seleção para um campo do banco de dados, a diferença é que **select-options** exibe dois campos para entrada de dados com um intervalo entre eles, campo to e campo from, podendo ser restringido a um só campo com **no-intervals** na sintaxe **select-options**.

Ex.:

```
report zexemplo.
tables sbook.
select-options fl_date for sbook-fldate.

initialization.
  move: 'I'      to fl_date-sign,
        'EQ'    to fl_date-option,
        sy-datum to fl_date-low.
  append fl_date.
  move: 'BT'      to fl_date-option,
        '19960101' to fl_date-low,
        '19960630' to fl_date-high.
  append fl_date.
```

Resultado:



Seleção Pré-Definida – checkbox

Os parâmetros podem ser criados como campos de dados que contenham apenas um valor de entrada e também podem ser criados como caixas de seleção. Quando os **parameters** assumem a forma das caixas de seleção, eles são declarados como tipo C e dessa forma armazenam o valor de X quando verificados e da tecla Espaço quando não verificados. Uma boa utilização do parâmetro **checkbox** é solicitar ao usuário para indicar se ele quer que certos componentes de um relatório sejam exibidos.

Seleção em um grupo – radiobutton

Assim como o checkbox, o radiobutton está selecionado ou não, mas somente operam em um ou mais grupos, sendo que apenas um único do grupo pode estar selecionado no momento da execução do programa.

Sint.:

```
...
selection-screen begin of block rad_blk with frame title text-000.
parameters: rad_ex1 radiobutton group one,
             rad_ex2 radiobutton group one,
             rad_ex1 radiobutton group one,
selection-screen end of block rad-blk.
...
```

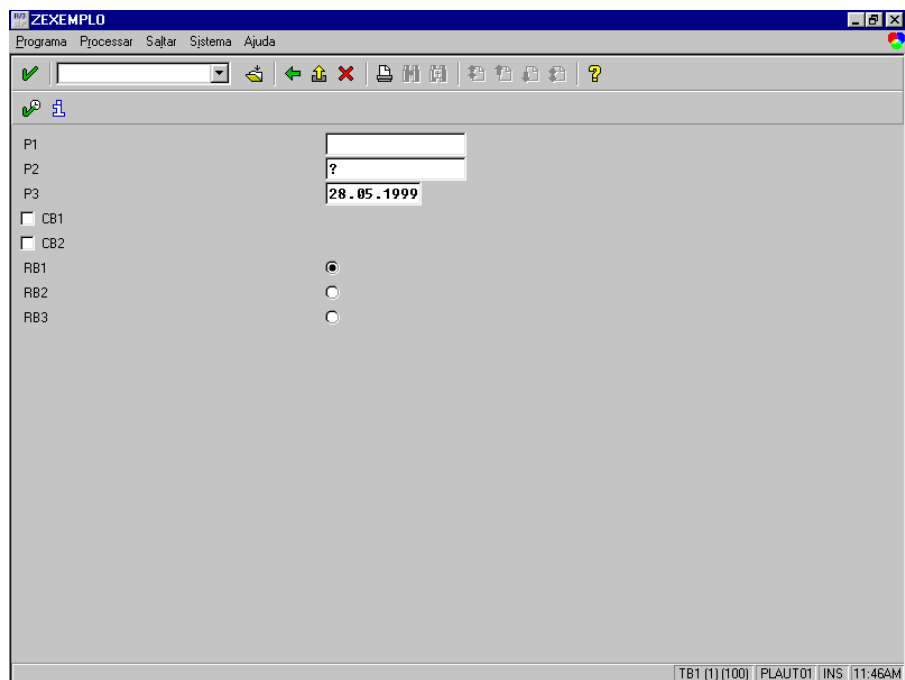
REPORT

No exemplo abaixo estaremos demonstrando **parameters**, **checkbox** e **radiobutton**.

Ex.:

```
report zexemplo.  
parameters: p1(15) type c,  
            p2 like p1 obligatory lower case,  
            p3 like sy-datum default sy-datum,  
            cb1 as checkbox,  
            cb2 as checkbox,  
            rb1 radiobutton group g1 default 'X',  
            rb2 radiobutton group g1,  
            rb3 radiobutton group g1.  
  
write:/ 'You entered:',  
       / \ p1 = \, p1,  
       / \ p2 = \, p2,  
       / \ p3 = \, p3,  
       / \ cb1= \, cb1,  
       / \ cb2= \, cb2,  
       / \ rb1= \, rb1,  
       / \ rb2= \, rb2,  
       / \ rb3= \, rb3.
```

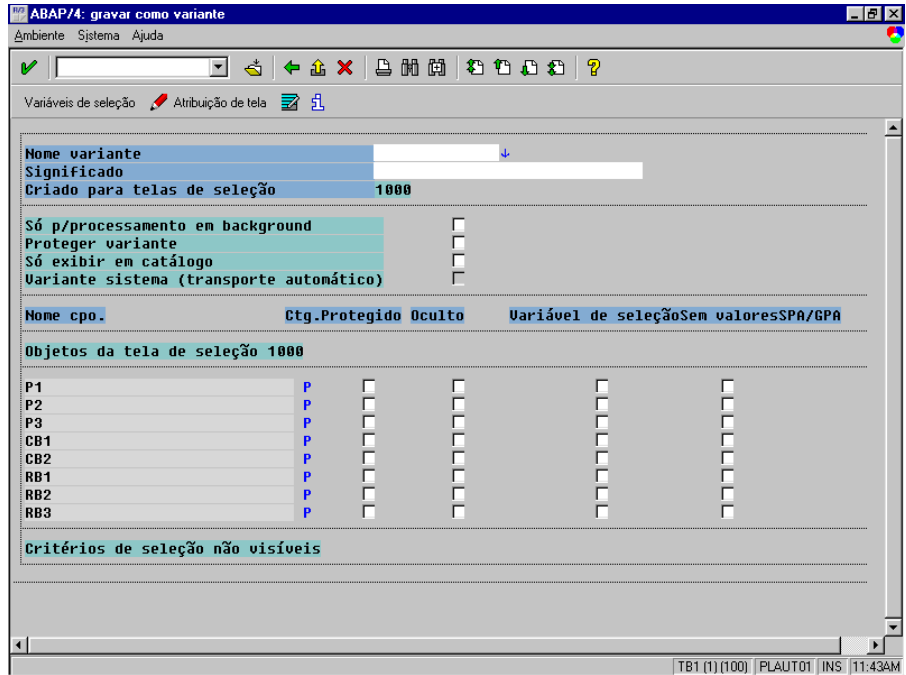
Resultado:



Variantes

Quando você entra com os mesmos dados constantemente você pode salvar estes dados em uma variante.

Para que se salve os dados, você deve estar na tela onde eles serão preenchidos e clique no ícone de salvar, de um nome para sua variante. Preencha a tela como do exemplo abaixo.



Expandir linhas do relatório – at line-selection

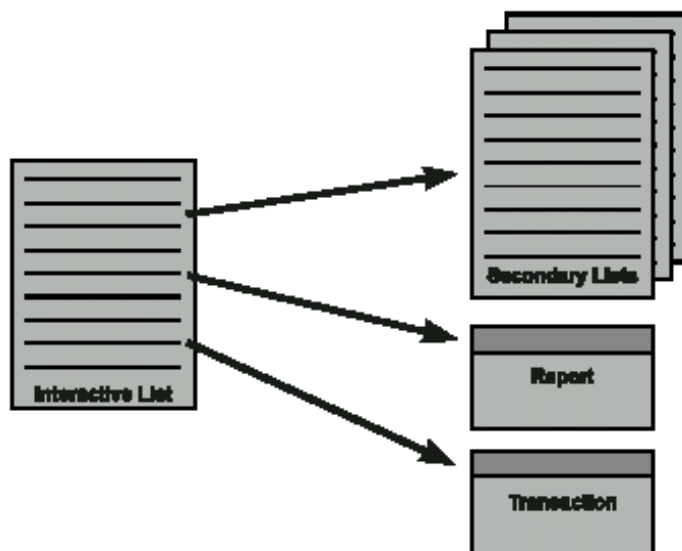
Um programa de relatório consiste de uma simples lista de informações, isto significa que quando o relatório é executado os dados estão disponibilizados de acordo com o que o lhe foi solicitado. Estes relatórios podem se tornar extensos, sendo que muitas vezes seu foco é distorcido mediante tantas informações. Para estas situações utilizamos o recurso de listas secundárias.

Este tipo de relatório não pode ser executado em background, a atuação do usuário é direta.

Quando houver necessidade de informações adicionais, o usuário interage durante a execução.

Relatórios Interativos permitem voce utilizar listas secundárias, outros relatórios e transações passando dados entre eles através do comando HIDE.

Interactive Reporting



Eventos de controle de um Relatório Interativo:

- At line-selection - Ocorre após o usuário clicar duas vezes sobre uma linha do relatório.
- At PF <nn> - Ocorre após o usuário pressionar uma tecla de função.
Exemplos: Help - F1 / Cancel - F13 / Exit - F15
- At user-command - Ocorre após o usuário clicar um botão definido na tela.

REPORT

Para o cabeçalho da lista secundária utilize o comando - Top-of-page during line-selection

Abaixo relacionamos algumas variáveis do sistema que ajudam nos controles das listas secundárias.

System field	Information
SY-LSIND	Index of the list created during the current event (basic list = 0)
SY-LISTI	Index of the list level from which the event was triggered
SY-LILLI	Absolute number of the line from which the event was triggered
SY-LISEL	Contents of the line from which the event was triggered
SY-CUROW	Position of the line in the window from which the event was triggered (counting starts with 1)
SY-CUCOL	Position of the column in the window from which the event was triggered (counting starts with 2)
SY-CPAGE	Page number of the first displayed page of the list from which the event was triggered
SY-STARO	Number of the first line of the first page displayed of the list from which the event was triggered (counting starts with 1). Possibly, a page header occupies this line.
SY-STACO	Number of the first column displayed in the list from which the event was triggered (counting starts with 1)
SY-UCOMM	Function code that triggered the event
SY-PFKEY	Status of the displayed list

Gui interface / Gui status / Set pf-status / Set titlebar / At user comand

gui interface

Esta ferramenta nos permite montar um “*Menu Painter*” onde você define funções como no Windows, tratada como interface com o usuário.

- Title Bar → Título do Menu Painter
- Menu Bar → Menu que sera criado
- Standard ToolBar → Onde você define ícones
- Aplication ToolBar → Botoes novos.

gui status

É a definição do tipo funções de um Menu que serão ativados ou desativados:

- Tipos de Status
- Screen
- Dialog Box
- List
- List in Dialog Box

set pf-status

É um comando para uso da determinação da **GUI** para cada lista criada. Você se refere a um status e tem que definir o “*Menu Painter*”. Para uma lista básica você pode escolher o sua **GUI** com status tipo “LIST”

Na realidade é um nome com máximo de 8 caracteres, para acionar a sua **GUI**

set titlebar

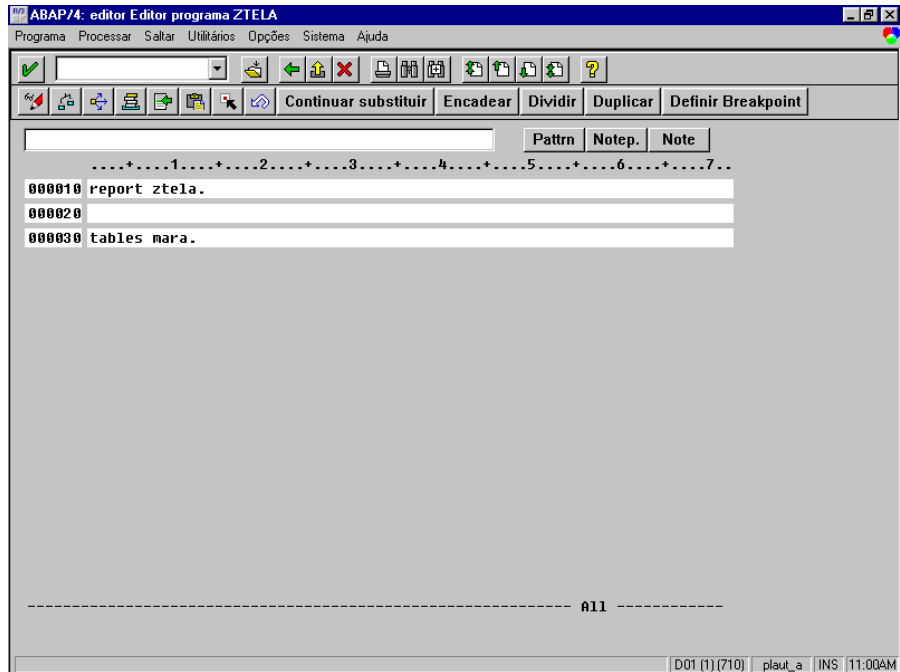
Este comando e para determinação do título da sua **GUI**.

at user-comand

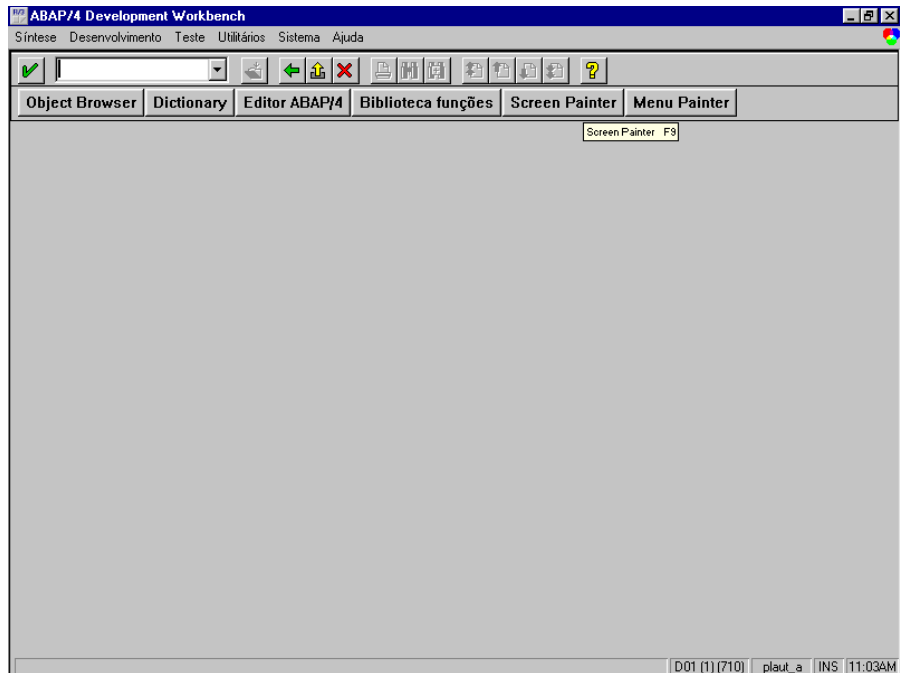
Este comando é que vai controlar a ação do usuário após definida a sua **GUI**.

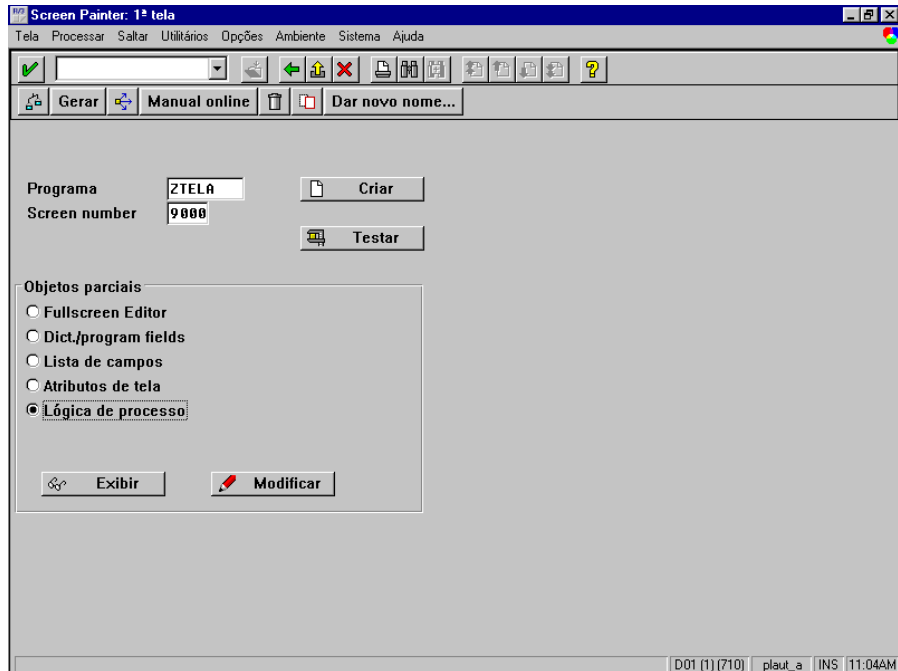
As figuras abaixo demonstram como criar funções em uma tela:

Entre na SE38 e informe as tabelas com as quais irá trabalhar, salve e saia.

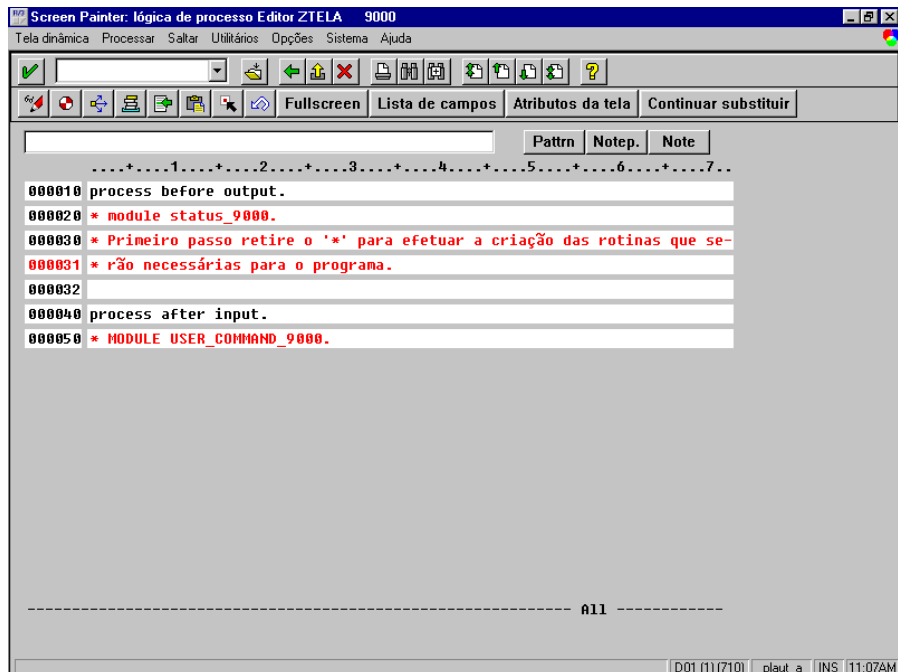


Entre na opção *Menu Painter* para habilitar as funções.
Preencha as informações conforme modelo abaixo.

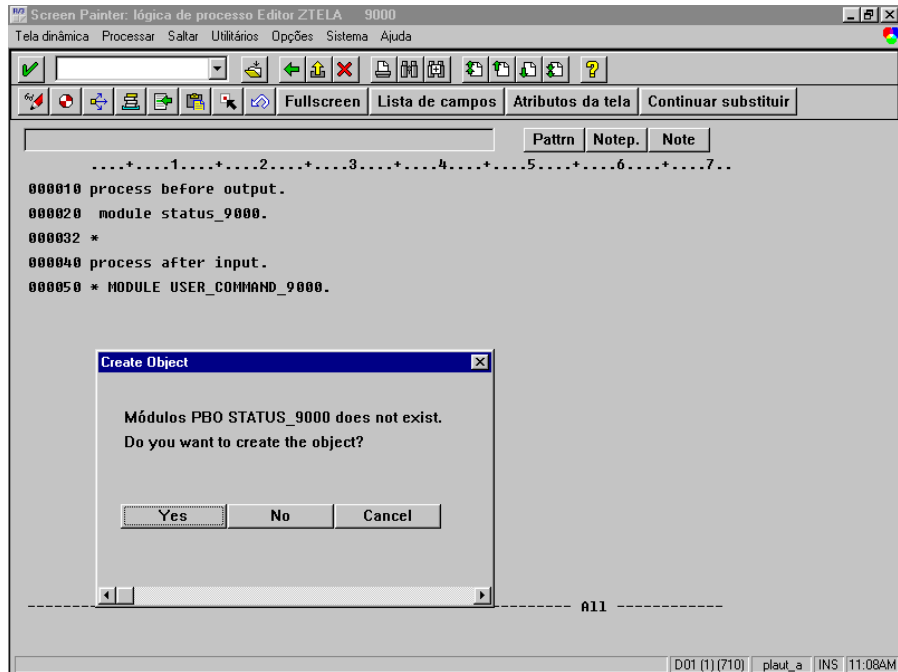




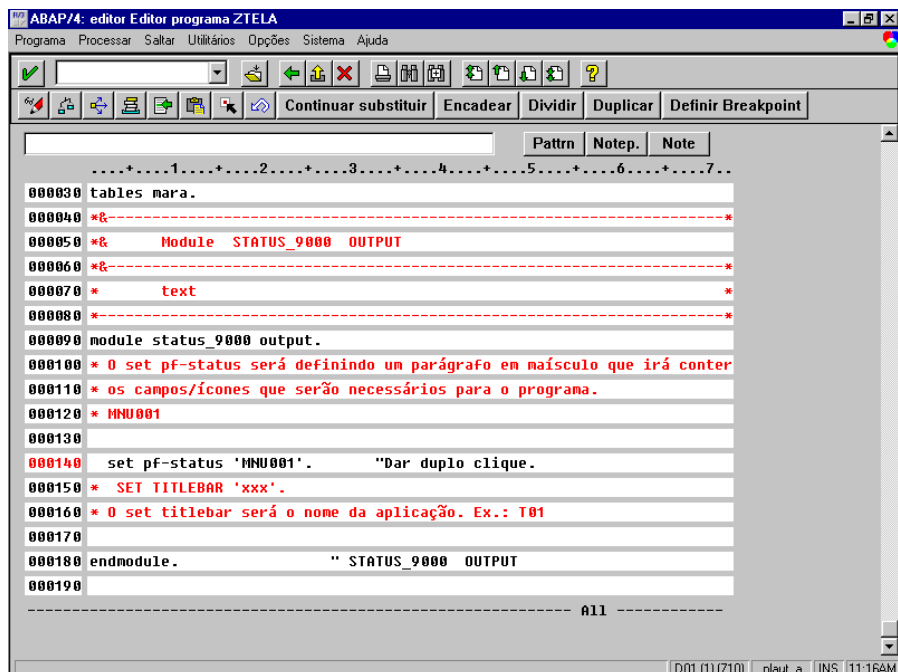
Siga as recomendações que estão com ‘*’ para definir as funções. Observe que existem o PBI (Process after Input) e o PBO (Process before output).



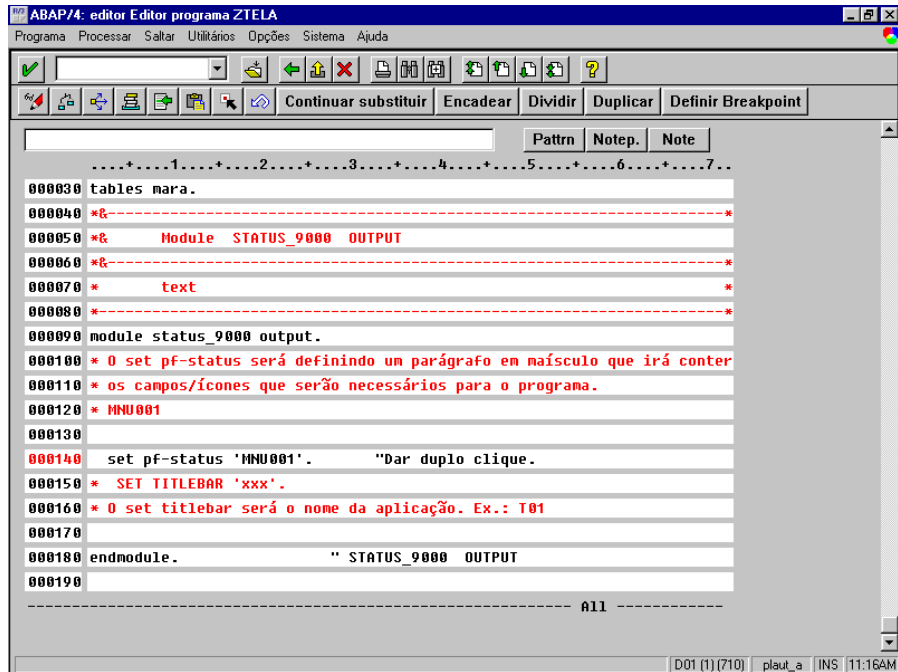
Siga como abaixo.



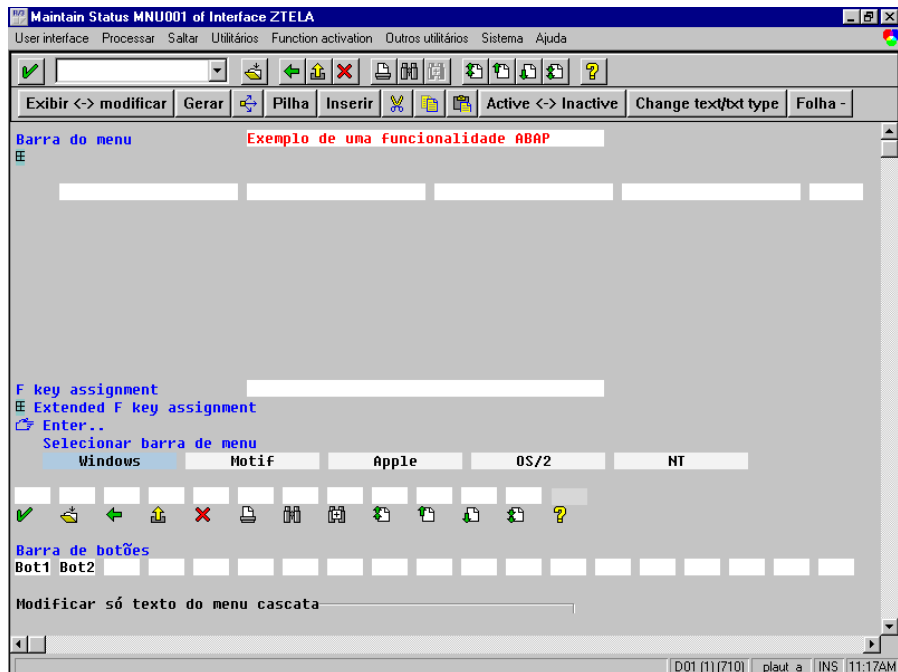
Siga as recomendações que estão com '*' ou após '---'.



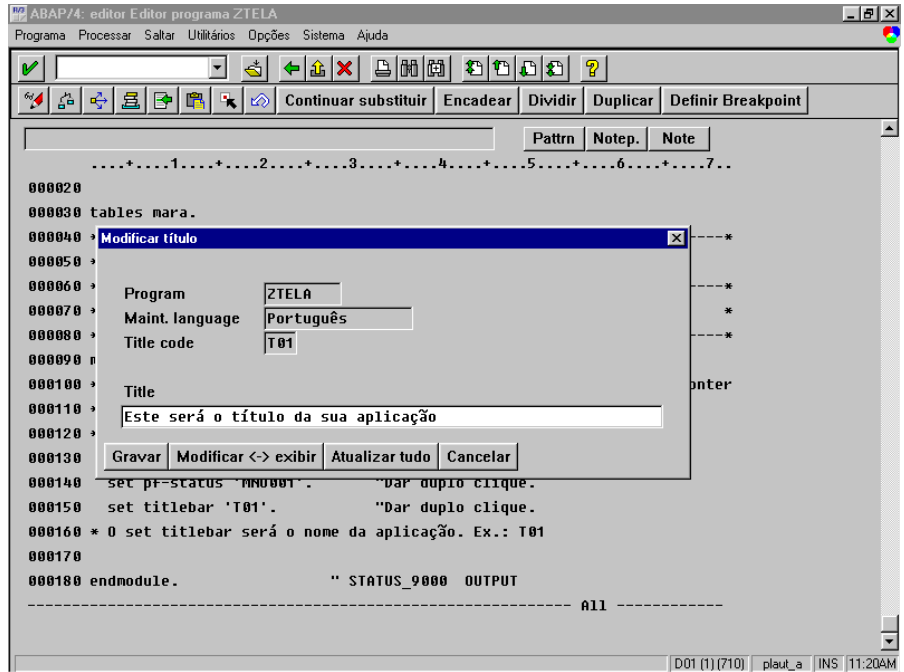
Siga as recomendações que estão com '*' ou após ''.



Esta funcionalidade permite que você crie ícones e funções. Ex.: Bot1 e Bot2.



Está opção deve ser utilizada para criar o título.



Listas secundarias com janelas (Comando Window)

Este comando você especifica as coordenadas de uma janela. A seqüência é início da coluna e número da linha., finaliza fim da coluna e fim da linha. Automaticamente é criada uma janela, ou você pode escolher **list in dialog box** de uma **GUI**, onde você cria título, botões dentro desta janela.

Ex.: **window starting at c1 11**
ending at c2 12

Para gerar um título para window use o adição no comando (**with frame title...**).

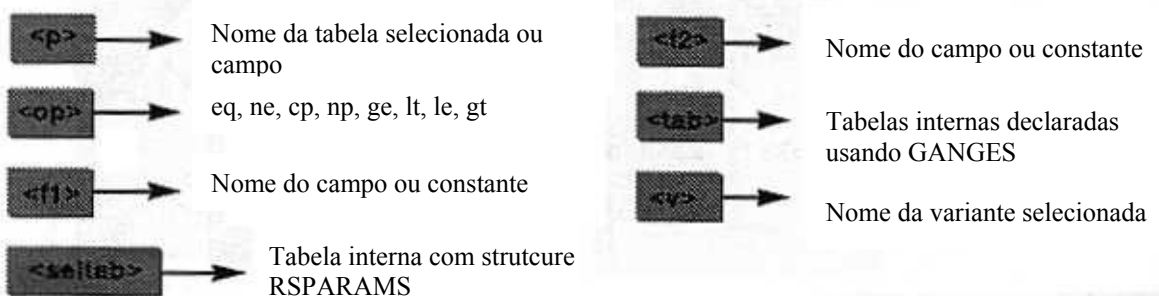
Componentes de ligação

Dentro do programa de ABAP você pode chamar outros programas de report ou transação através dos comandos `call` e `submit`.

Chamando outro relatório – `submit`

Este comando é utilizado quando for necessário dentro de um programa você chamar e processar outro programa. Você pode impor condições para que o `submit` ocorra, também pode declarar o nome do programa que vai ser executado.

```
SUBMIT <report>
AND RETURN
VIA SELECTION-SCREEN
AND RETURN EXPORTING LIST TO MEMORY
WITH <p> <op> <f1>
WITH <p> BETWEEN <f1> AND <f2>
WITH <p> IN <tab>
USING SELECTION-SET <v>
WITH SELECTION-TABLE <seltab>
```



Retorno ao relatório – `submit and return`

Este comando é semelhante ao anterior, sendo sua diferença a opção de retornar ao programa principal após ser processado o programa que foi chamado por **submit**.

Sint.:

```
...  
submit report01  
  via selection-screen  
  using selection-set 'variant1'  
  using selection-sets of program 'report00'  
  and return.  
...
```

Transferência de dados – export / import

Estes comandos podem ser usados para importar ou exportar dados entre dois programas que fazem parte de uma seqüência de chamada.

Não esquecer que você tem que ter um **submit** no programa que vai exportar os dados.

Você pode utilizar o parâmetro **id** que ajudará a gerenciar a memória evitando que no futuro tenha problemas com o grande volume de memória ocupada por esta operação.

Sint.:

EXPORT <datacluster> to memory.

IMPORT <datacluster> to memory.

Os parâmetros **to** e **from** são opcionais.

Ex.:

```
report exemplo.
tables indx.
data: indxkey like indx-srtfd value 'keyvalue',
      f1(4), f2 type p,
      begin of itab3 occurs 2,
        cont(4),
      end of itab3.
indx-aedat = sy-datum.
indx-usera = sy-uname.
* export data.
export f1 f2 itab3 to
      database indx(st) id indxkey.
```

```
report exemplo.
tables indx.
data: indxkey like indx-srtfd,
      f1(4), f2 type p,
      begin of tab3 occurs 10,
        cont(4),
      end of tab3.
indxkey = 'indxkey'.
import f1 f2 tab3 from database indx(st) id indxkey.
```


Constants

Como o nome já diz é uma constante, sendo que é quase idêntica a uma variável, exceto quanto ao fato de que seu valor não poderá ser alterado. A instrução **constants** é semelhante à instrução **data**; entretanto, a adição do comando **value** é necessário. O comando **constants** pode ser usado quando for incluir um literal várias vezes no mesmo programa definindo um valor igual ao da literal e utilizar a constante no corpo do programa e quando for necessário alterar o valor de sua literal basta apenas alterar o valor de sua constante.

Sint.:

```
constants col[(1)] [type t] [decimals d] value 'zaza'.
```

- **col** é o nome da constante
- **(1)** é a especificação interna de comprimento.
- **t** é o tipo de dado.
- **d** é o número de casas decimais (utilizadas apenas com o tipo **p**).
- **'zaza'** é um literal que fornece o valor da constante.

Ex.:

```
...
constants  char1 value 'x'.
...
constants  int    type i value 99.
...
constants: begin of const_rec,
            c(2) type i value 'xx',
            n(2) type n value '12',
            x    type x value 'ff',
            i    type i value 99,
            p    type p value 99,
            f    type f value '9.99e9',
            d    type d value '19950101',
            t    type t value '235959',
            end of const_rec.
...
```

Types

A instrução **types** é utilizada para definir seus próprios tipos de dados e orientar-se nos tipos já existentes.

Sint.:

```
types tp1(c) type t decimals d.
types tp1 like g1.
```

tp1 é o nome do tipo.

g1 é o nome de uma variável previamente definida no programa ou é o nome de um campo que pertence a uma tabela ou estrutura no Data Dictionary.

(d) é a especificação interna de comprimento.

t é o tipo de dado.

d é o número de casas decimais (utilizadas apenas com o tipo p).

Ex.:

```
report zexemplo.
types wa(2) type c.
data: va1 type wa value 'FOME',
      va2 type wa value 'SEDE'.
write va1, va2.
```

Resultado:

FOME SEDE

Tipos de **Types**:

Tipo	Descrição	Tamanho	Valor Inicial
C	Texto (Character)	1	Space
N	Texto Numérico	1	'00...0'
D	Data (YYYYMMDD)	8	'00000000'
T	Hora (HHMMSS)	6	'000000'
X	Hexadecimal	1	x '00'
I	Whole number (integer)	4	0
P	Packed number	8	0
F	Floating point number	8	'0.0'

Types Estruturados

É quando um tipo definido pelo usuário pode ser baseado na definição de uma string de campo, é utilizado para reduzir a redundância e tornar a manutenção mais fácil.

exemplo:

```
report zexemplo.
types: begin of address,
       street(26),
       city(15),
       postal_code(9),
       end of address.
data: customer_addr type address,
      vendor_addr   type address.
customer_addr-street = 'zazzaza'.
write: / customer_addr-street.
```

Clear

O comando **clear** configura o valor de uma variável ou string de campo como zero. Se o dado é do tipo **c**, o valor, por sua vez, será configurado como espaços em branco.

Sint.:

```
clear c1 [with c2 : with 'A': with null]
```

c1 e **c2** são nomes de string de campo ou variáveis.

'A' é um literal de qualquer comprimento.

Ex.:

```
...  
DATA: TEXT(10)          VALUE 'Hello',  
      NUMBER TYPE I    VALUE 12345,  
      ROW(10) TYPE N   VALUE '1234567890',  
      BEGIN OF PLAYER,  
        NAME(10)       VALUE 'John',  
        TEL(8) TYPE N  VALUE '08154711',  
        MONEY TYPE P  VALUE 30000,  
      END   OF PLAYER.  
...  
CLEAR: TEXT, NUMBER, PLAYER.  
...
```

Free

Utilize o comando **free** para excluir todas as linhas de uma tabela interna e liberar a memória associada. Trabalha da mesma forma que o comando **clear**.

Sint.:

free itab.

itab é uma tabela interna com ou sem uma linha de cabeçalho.

todas as linhas são excluídas e toda a memória utilizada pelo corpo da tabela interna é liberada.

a linha de cabeçalho, se existir, permanecerá inalterada.

utilize **free** quando terminar de utilizar uma **itab**.

Ex.:

```
data: begin of itab occurs 3,  
      t1 like sy-index,  
      end of itab.  
do 3 times.  
  itab-t1 = sy-index.  
  append itab.  
enddo.  
loop at itab.  
  write itab-t1.  
endloop.  
free itab.  
if itab[] is initial.  
  write: / 'não existe indicação na itab após instrução free'.  
endif.
```

Resultado:

```
      1          2          3  
não existe indicação na itab após instrução free
```

Refresh

Ele exclui todas as linhas de uma tabela interna, mas deixa a memória alocada, você utiliza quando quiser preencher a tabela novamente. Se você necessita utilizar a tabela imediatamente após limpá-la, o comando **refresh** é mais eficiente do que **free** já que impede alocações desnecessárias de memória.

Sint.:

refresh itab.

itab é uma tabela interna com ou sem uma linha de cabeçalho.

Todas as linhas são excluídas. Toda a memória utilizada pelo corpo da tabela interna permanece alocada.

A linha de cabeçalho, se existir, permanecerá inalterada

Ex.:

```
data: begin of itab occurs 3,  
      t1 like sy-index,  
      end of itab,  
      v like sy-index.  
do 3 times.  
  v = sy-index.  
  do 3 times.  
    itab-t1 = v * sy-index.  
    append itab.  
  enddo.  
  write: / ''.  
  loop at itab.  
    write itab-t1.  
  endloop.  
  refresh itab.  
enddo.
```

Resultado:

1	2	3
2	4	6
3	6	9

Executando Cálculos

`compute`

Instrução utilizada com mais frequência para executar cálculos.

Sint.:

```
compute va3 = va1 op va2
```

va3 é a variável de recebimento do resultado do cálculo.

va1 e **va2** são os operandos.

op é um operador matemático.

Ex.:

```
...  
va3 = va2 + va3.  
va1 = ( va2 + va3 ) * va4  
...
```

Quando for necessário trabalhar com parênteses utilize um espaço, “(“ , “)”, antes e depois do valor dentro do parênteses.

`add` ou `add-corresponding`

O comando **add** é utilizado para adicionar um número a outro, com **add-corresponding** as strings de campo com os dados que têm o mesmo nome podem ser adicionadas juntamente.

Sint.:

```
...  
add va1 to va2.  
add-corresponding st1 to st2  
...
```

REPORT

Assim como demonstrado em **add** (função de adicionar) abaixo demonstraremos subtrair, multiplicar e dividir. As instruções abaixo operam de forma semelhante a **add**.

subtract ou **subtract-corresponding**

Sint.

```
subtract va1 from va2.  
subtract-corresponding st1 from st2.
```

multiply ou **multiply-corresponding**

Sint.:

```
multiply va1 by va2.  
multiply-corresponding st1 by st2.
```

divide ou **divide-corresponding**

Sint.:

```
divide va1 by va2.  
divide-corresponding st1 by st2.
```


Check

Assume a condição de IF mas com menos complexidade, com a mesma eficiência e também podendo ser codificada dentro de um loop, passa o controle imediatamente à instrução de terminação do loop e desviando das instruções intermediárias. Aceita uma expressão lógica. Se a expressão for verdadeira, não fará nada. Se for falsa, pulará para o final do loop.

Sint.:

```
...  
select * from mara where matnr = 1.  
check sy-subrc = 0.  
...
```

Ex.:

```
report zexemplo.  
tables: spfli,  
        sflight.  
...  
select-options:  
    sf_price   for sflight-price,  
    sp_carr    for spfli-carrid,  
    sp_from    for spfli-cityfrom no database selection,  
    sp_dept    for spfli-deptime.  
...  
get sflight.  
check select-options.  
...
```

Read Table

É utilizado para ler e localizar uma única linha que corresponde aos critérios específicos e a coloca em uma área de trabalho. O comando **read table** só pode ser utilizado para a leitura de tabelas internas, não funcionando com banco de dados.

Ex.:

```
report zexemplo.
data: begin of ti occurs 3,
      val(2) type n,
      va2     type i,
      end of ti,
      va3 like ti.
ti-val1 = '5'.  ti-val2 = 2. append ti.
ti-val1 = '10'. ti-val2 = 5. append ti.
read table ti index 2.
write: / 'sy-subrc =', sy-subrc,
      / 'sy-tabix =', sy-tabix,
      /  ti-val1, ti-val2.
```

Resultado:

```
sy-subrc =      0
sy-tabix =      2
10         5
```

Insert

Você pode utilizá-lo para inserir uma única linha em uma tabela interna.

Sint.:

```
...  
insert [wa into] itab [index n]  
...
```

wa é uma área de trabalho com a mesma estrutura de uma linha de tabela interna **itab**.

n é uma constante, literal ou variável numérica.

se **wa** for especificado, o conteúdo dela será inserido em **itab**. **wa** deverá ter a mesma estrutura que **itab**.

se **wa** não for especificado, o conteúdo da linha de cabeçalho será inserido em **itab**. Se **itab** não tiver uma linha de cabeçalho, **wa** deverá ser especificado.

se **index** for especificado, a nova linha será inserida antes da linha **n**. A linha **n** se tornará então a linha **n + 1**.

a instrução **insert** pode ser utilizada dentro ou fora de **loop at itab**. Se utilizada fora, a adição **index** deverá ser especificada. Se utilizada dentro, **index** será opcional. Se a adição não for especificada, a linha atual será assumida.

exemplo:

```
data: begin of itab occurs 5,  
      fal like sy-index,  
      end of itab.  
do 5 times.  
  itab-fal = sy-index.  
  append itab.  
enddo.  
itab-fal = -99.  
insert itab index 3.  
loop at itab.  
  write / itab-fal.  
endloop.  
loop at itab where fal >= 4.  
  itab-fal = -88.  
  insert itab.  
endloop.  
skip.  
loop at itab.  
  write / itab-fal.  
endloop.
```

Resultado:

1
2
99-
3
4
5

1
2
99-
3
88-
4
88-
5

Modify

Serve para modificar o conteúdo de uma ou mais linhas de uma tabela interna

Sint.:

```
modify itab [from wa] [index n] [transporting c1 c2 ... [where exp]]
```

itab é o nome de uma tabela interna com ou sem uma linha de cabeçalho.

wa é uma área de trabalho com a mesma estrutura de uma linha no corpo de itab.

v é uma constante, variável ou literal numérica.

c1 e **cp2** são componentes de **itab**.

e1 é uma expressão lógica envolvendo componentes de itab.

obs.:

se **from wa** for especificada, a linha será sobrescrita com o conteúdo de **wa**.

se **from wa** não for especificada, a linha será sobrescrita com o conteúdo da linha de cabeçalho.

se **index n** for especificada, **n** identificará o número da linha que foi sobrescrita.

modify itab pode ser especificada dentro ou fora de **loop at itab**. Se especificada fora, **index n** deverá ser especificado. Quando especificada dentro, **index n** será opcional. Se não for especificada, a linha atual será modificada.

transporting especifica quais componentes devem ser sobrescritos e o restante permanecerá inalterado. Sem ela, todos eles serão sobrescritos.

usar a condição **where** depois de **transporting** fará com que os componentes especificados sejam sobrescritos em todas as linhas que satisfazem a condição **where**. O lado esquerdo de cada parte de **exp** deve especificar um componente de **itab**. O mesmo componente pode ser especificado depois de **transporting** e em **exp**.

Não poderá utilizar **modify itab** com **where**:

- dentro do **loop at itab**

- com a adição **index**

REPORT

Ex.:

```
report zexemplo.
data: begin of itab occurs 5,
      t1 like sy-index,
      t2,
      end of itab,
      gama(5) value 'plaut'.
do 5 times varying itab-t2 from gama+0 next gama+1.
  itab-t1 = sy-index.
  append itab.
enddo.
itab-t2 = 'v'.
modify itab index 4.
loop at itab.
  write: / itab-t1, itab-t2.
endloop.
loop at itab.
  itab-t1 = itab-t1 * 2.
  modify itab.
endloop.
skip.
loop at itab.
  write: / itab-t1, itab-t2.
endloop.
itab-t2 = 'W'.
modify itab transporting t2 where t1 <> 10.
skip.
loop at itab.
  write: / itab-t1, itab-t2.
endloop.
```

Resultado:

```
1  p
2  l
3  a
5  v
5  t

2  p
4  l
6  a
10 v
10 t

2  W
```

ABAP

1.00 - 87

REPORT

4 W
6 W
10 v
10 t

Delete

Com o comando **delete** você poderá excluir uma ou mais linhas de uma tabela interna. Semelhante as inserções, as exclusões dentro do **loop at itab** não afetarão imediatamente a tabela interna, ao contrário, se tornarão efetivas na próxima passagem do **loop**.

Sint.:

```
Delete itab [index c]  
           [from v] [to l]  
           [where exp]
```

itab é uma tabela interna

c, *v* e *l* são constantes, variáveis e literais numéricos

exp é uma expressão lógica envolvendo os componentes de *itab*.

Occurs

Não limita o número de linhas que pode ser adicionado à tabela interna, teoricamente o número de linhas que pode ser adicionado a uma tabela interna é apenas limitado a memória virtual disponível no servidor do aplicativo. O sistema utiliza o comando **occurs** apenas como uma especificação para determinar quanta memória será alocada.

With Keys

Se a utilização da adição for especificada, o sistema localizará uma linha que corresponda com a expressão key e a colocará na linha de cabeçalho. Usando a expressão key, você pode procurar por uma linha especificando um valor ao invés de índice.

Ex.:

```
report zexemplo
data: begin of itab occurs 3,
      t1(3) type n,
      t2   type i,
      t3(3) type c,
      t4   type p,
      end of itab,
      begin of tabi,
        t1 like itab-t1,
        t2 like itab-t2,
      end of tabi,
      tabt like itab,
      t(8).
itab-t1 = '100'.
itab-t3 = 'pp'.
itab-t2 = itab-t4 = 1.
append itab.
itab-t1 = '200'.
itab-t3 = 'll'.
itab-t2 = itab-t4 = 2.
append itab.
itab-t1 = '300'.
itab-t3 = 'aa'.
itab-t2 = itab-t4 = 3.
append itab.
read table itab with key t1 = '300' t2 = 3.
write: / 'sy-subrc =', sy-subrc,
      / 'sy-tabix =', sy-tabix,
      / itab-t1, itab-t2, itab-t3, itab-t4.
t = 't2'.
read table itab into tabt with key (t) = 2.
write: /,
      / 'sy-subrc =', sy-subrc,
      / 'sy-tabix =', sy-tabix,
      / itab-t1, itab-t2, itab-t3, itab-t4,
      / tabt-t1, tabt-t2, tabt-t3, tabt-t4.
clear tabi.
tabt-t1 = '100'.
tabt-t3 = 'pp'.
```

REPORT

```
tabt-t2 = tabt-t4 = 1.
read table itab with key = tabt.
write: /,
      / 'sy-subrc =', sy-subrc,
      / 'sy-tabix =', sy-tabix,
      / itab-t1, itab-t2, itab-t3, itab-t4.
tabi-t1 = '200'.
tabi-t2 = 2.
read table itab into tabt with key tabi.
write: /,
      / 'sy-subrc =', sy-subrc,
      / 'sy-tabix =', sy-tabix,
      / itab-t1, itab-t2, itab-t3, itab-t4.
```

Resultado:

```
sy-subrc =      0
sy-tabix =      3
300           3  AA           3

sy-subrc =      0
sy-tabix =      2
300           3  AA           3
200           2  LL           2

sy-subrc =      0
sy-tabix =      1
100           1  PP           1

sy-subrc =      0
sy-tabix =      2
100           1  PP           1
```

Form

Define o fim do evento precedente e o começo de uma sub-rotina. As sub-rotinas não podem ser excessivas no interior de eventos.

Sint.:

```
form <sub> tables pa1 pa2...
      using pa3 value pa4.
      ...
endform.
```

<sub> é o nome da sub-rotina.

pa1 e pa2 são parâmetros.

Ex.:

```
TYPES: BEGIN OF FLIGHT_STRUC,
        FLCARRID LIKE SFLIGHT-CARRID,
        PRICE    LIKE SFLIGHT-FLDATE,
        END      OF FLIGHT_STRUC.

DATA: MY_FLIGHT TYPE FLIGHT_STRUC OCCURS 0,
      IBOOK1    LIKE SBOOK          OCCURS 0,
      IBOOK2    LIKE IBOOK1        OCCURS 0,
      STRUC     LIKE SBOOK.

PERFORM DISPLAY USING MY_FLIGHT IBOOK1 IBOOK2 STRUC.

FORM DISPLAY USING P_ITAB LIKE MY_FLIGHT[]
                  P_BOOK1 LIKE IBOOK1[]
                  P_BOOK2 LIKE IBOOK2[]
                  P_STRU  LIKE STRUC.

DATA: L_FLIGHT LIKE LINE OF P_ITAB,
      L_CARRID LIKE L_FLIGHT-FLCARRID.
...
WRITE: / P_STRU-CARRID, P_STRU-CONNID.
...
LOOP AT P_ITAB INTO L_FLIGHT WHERE FLCARRID = L_CARRID.
...
ENDLOOP.
ENDFORM.
```

Perform

É utilizado para chamar uma sub-rotina, no caso o **form**.

Ex.:

```

report zexemplo.
data: number_i type i value 5,
      number_p type p value 4,
      begin of person,
          name(10)      value 'paul',
          age type i    value 28,
      end   of person,
      alpha(10)        value 'abcdefghij'.
field-symbols <pointer>.
assign number_p to <pointer>.
perform change using 1
                        number_i
                        number_p
                        <pointer>
                        person
                        alpha+number_i(<pointer>).
*-----*
*      form change                                           *
*-----*
*      .....                                               *
*-----*
* --> value(par_1)                                           *
* --> par_number_i                                           *
* --> par_number_p                                           *
* --> par_pointer                                            *
* --> par_person                                             *
* --> par_part_of_alpha                                       *
*-----*
form change using value(par_1)
                  par_number_i
                  par_number_p
                  par_pointer
                  par_person structure person
                  par_part_of_alpha.
add par_1 to par_number_i.
par_number_p = 0.
par_person-name+4(1) = alpha.
par_person-age = number_p + 25.
add number_i to par_pointer.
par_part_of_alpha = space.
endform.

```

Include

Um programa **include** é um aplicativo em que o conteúdo é projetado para ser utilizado por outro programa, podendo estar incluído em mais de um programa. Normalmente ele não é completo sozinho. Um programa **include** deve ser do tipo **i**. O código do programa **include** é copiado como está e substitui o comando **include** no tempo de geração do programa.

Ex.:

Programa Principal

```
report zexemplo.  
tables: <tab1>, <tab2>.  
parameters va_var1 like tab1-camp1.  
  
include: zexemp1,  
         zexemp2.  
  
top-of-page.  
  write: / 'Resultado', va_var1.  
  uline.
```

Include ZEXEMP1

```
*** include zexemp1.  
select single * from tab1 where camp1 = va_camp1.
```

Include ZEXEMP2

```
*** include zexemp2.  
select * from tab2 where camp1 = tab1-camp1.  
write: / tab2-camp1.  
endselect.
```

Call function

Este comando é utilizado para chamar um módulo de função. **call function** é um comando específico, não necessitando que seja colocado pontos ou vírgulas depois de parâmetros ou nomes de exceção. O nome do módulo deve ser codificado em letras maiúsculas, se for codificado em letras minúsculas, a função não será localizada e um dump ocorrerá.

exemplo:

```
data begin of bdcdata occurs 100.
    include structure bdcdata.
data end of bdcdata.

data begin of itab occurs 10.
    include structure bdcmsgcoll.
data end of itab.

data program like sy-repid

bdcdata-program = 'sapms38m'.
bdcdata-dynpro  = '0100'.
bdcdata-dynbegin = 'x'.
append bdcdata.
clear bdcdata.
bdcdata-fnam    = 'rs38m-programm'.
bdcdata-fval    = program.
append bdcdata.
...
call transaction 'se38' using bdcdata mode 'n'
                    messages into itab.
```

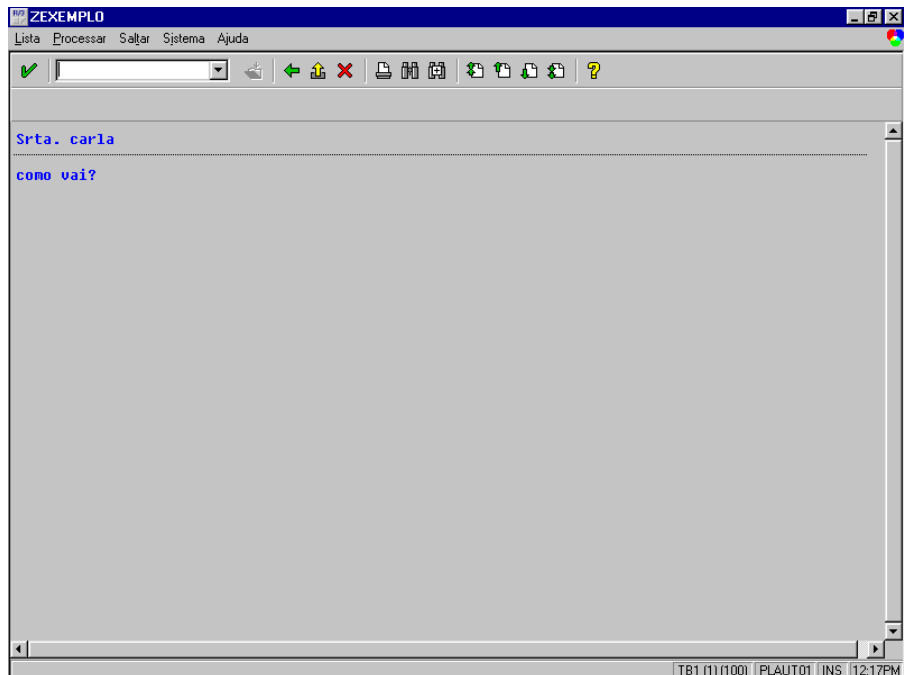
Top-of-page

Este comando é utilizado para criar títulos personalizados (cabeçalhos). Se não houver uma tela de seleção, o primeiro comando **write** executado acionará o **top-of-page**, caso o seu programa possua uma tela de seleção é possível que o **top-of-page** seja acionado duas vezes, primeiro pelo comando **write** executado antes da tela de seleção ser mostrada ou pelo comando **write** executado após a tela de seleção ser mostrada.

Ex.:

```
report zexemplo no standard page heading.  
data: val(5) value 'gata',  
      va2(5) value 'carla'.  
write: / 'Como vai?'.  
top-of-page.  
write: / 'Srta.', va2. uline.
```

Resultado:



End-of-page

Este comando cria rodapés de página, os comandos que estiverem após o **end-of-page** serão executados antes da geração de cada nova página. Você precisa reservar espaço na parte inferior da página (**line-count**) para o **end-of-page**.

Ex.:

```
report zexemplo line-count 15(2) no standard page heading.
data: va1(5) value 'gata',
      va2(5) value 'Carla'.
do 20 times.
  write: / 'como vai', sy-index, 'times'.
enddo.
```

```
top-of-page.
  write: / 'Srta.', va2.
  uLINE.
```

```
end-of-page.
  write: / sy-uline,
        / 'Este é o rodapé com duas linhas reservadas'.
```

Resultado:

```
ZEXEMPLO
Lista Processar Saltar Sistema Ajuda

Srta. Carla
-----
como vai      1 times
como vai      2 times
como vai      3 times
como vai      4 times
como vai      5 times
como vai      6 times
como vai      7 times
como vai      8 times
como vai      9 times
como vai     10 times
como vai     11 times

este é o rodapé com duas linhas reservadas
-----
Srta. Carla

como vai     12 times
como vai     13 times
como vai     14 times
como vai     15 times
como vai     16 times
como vai     17 times
como vai     18 times
como vai     19 times
como vai     20 times

TB1 (1) (100) | PLAUT01 | INS | 12:26PM
```


At selection-screen

É processado depois da entrada do usuário na tela de seleção ativa, podendo ocorrer após o usuário clicar um botão. Além da verificação de validações de dados, mensagens de aviso, alteração do status GUI e até mesmo janelas pop-up podem ser abertas utilizando este evento.

Ex.:

```
...
select-options name for sy-repid modif id xyz.
...
at selection-screen output.
  loop at screen.
    check screen-group1 = 'xyz'.
    screen-intensified = '1'.
    modify screen.
  endloop.
....
```

At user-command

Os botões, assim como muitas outras opções de tela de seleção baseadas em evento, podem ser muito úteis na manutenção da interação com usuário e na validação de entrada do usuário.

```
data: number1 type i value 20,  
      number2 type i value 5,  
      result type i.  
  
start-of-selection.  
  write: / number1, '?', number2.  
  
at user-command.  
  case sy-ucomm.  
    when 'add'.  
      result = number1 + number2.  
    when 'subt'.  
      result = number1 - number2.  
    when 'mult'.  
      result = number1 * number2.  
    when 'divi'.  
      result = number1 / number2.  
    when others.  
      write 'unknown function code'.  
      exit.  
  endcase.  
  write: / 'result:', result.
```

Resultado:

```
20 ?      5
```

Message

As mensagens são mantidas e armazenadas na tabela T100 e podem ser acessadas a partir do ABAP Workbenck. Um programador efetivo emite mensagens que são descritivas e ajudam o usuário a entender a natureza do fluxo do programa.

É possível atribuir a cada instrução de mensagem tipos de mensagem que têm diversos efeitos no resultado do programa.