TOTVS

PROGRESS OPENEDGE

Progress Básico (DCA)

2025/ julho

01 Introdução ao Progress OpenEdge

OpenEdge - O Que é?

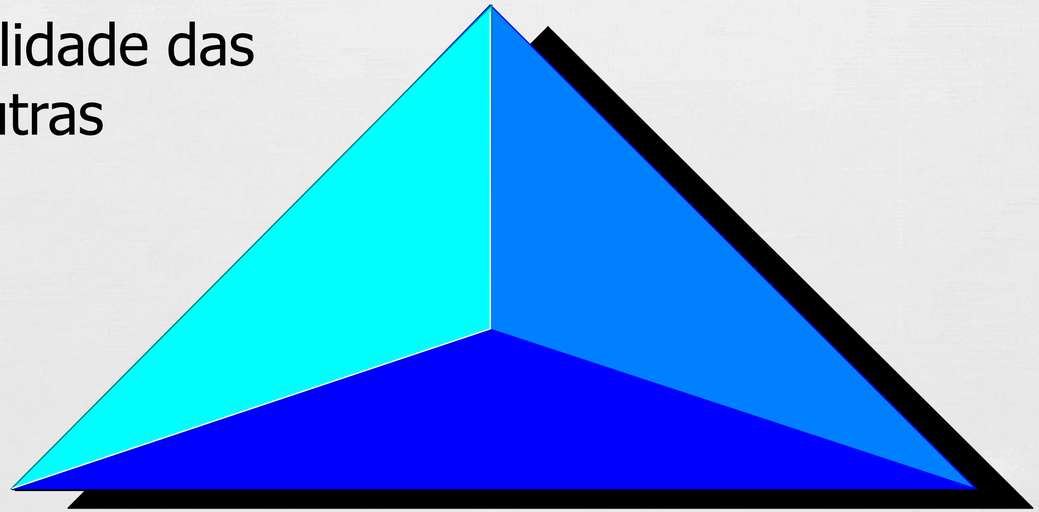
Completo ambiente de desenvolvimento de aplicações, composto por:

- Sistema Gerenciador de BD Relacional
- Linguagem de 4a. Geração
- Ferramentas de Programação

OpenEdge - O Que é?

Flexível

Assegura portabilidade das aplicações em outras plataformas



Aberto

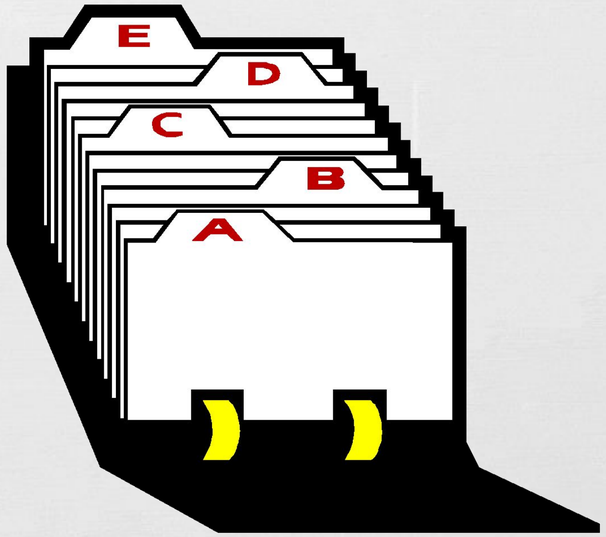
- Suporta o padrão industrial
- Assegura opções de desenvolvimento heterogêneo

Completo

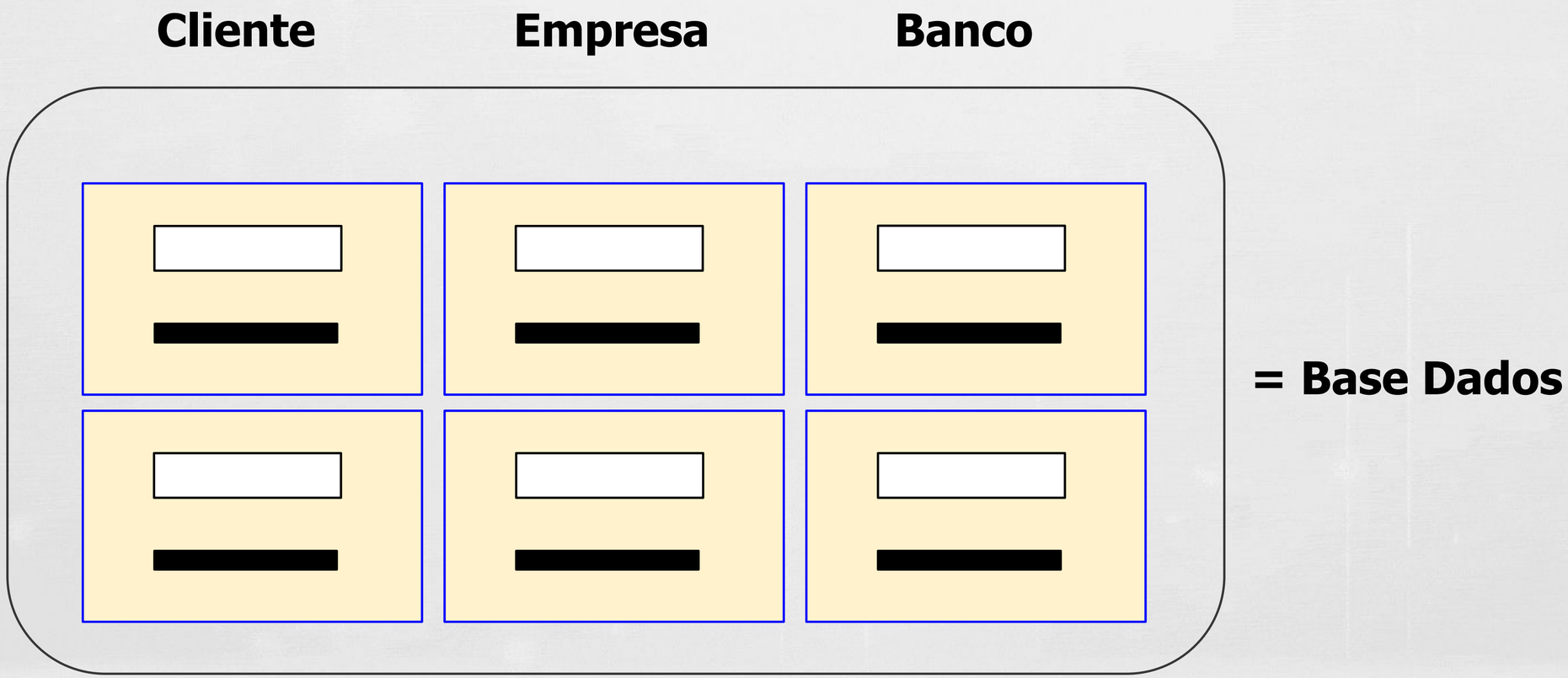
- Permite construção de grandes aplicações
- Produz soluções Client/Server

Base de Dados

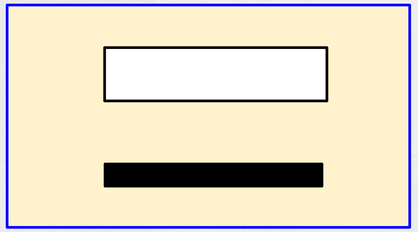
- Tabela
  - Registro
  - Campo
- Índice



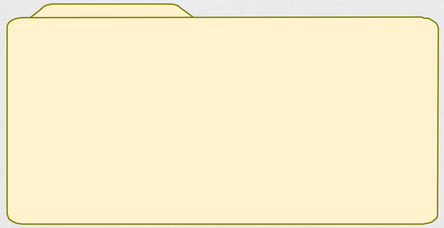
OpenEdge - Banco de Dados



OpenEdge - Banco de Dados

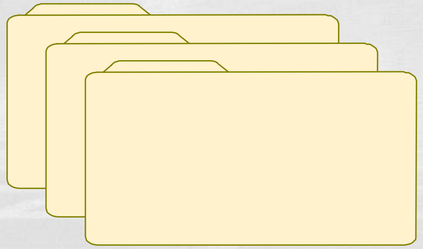


= Tabela



= Registro

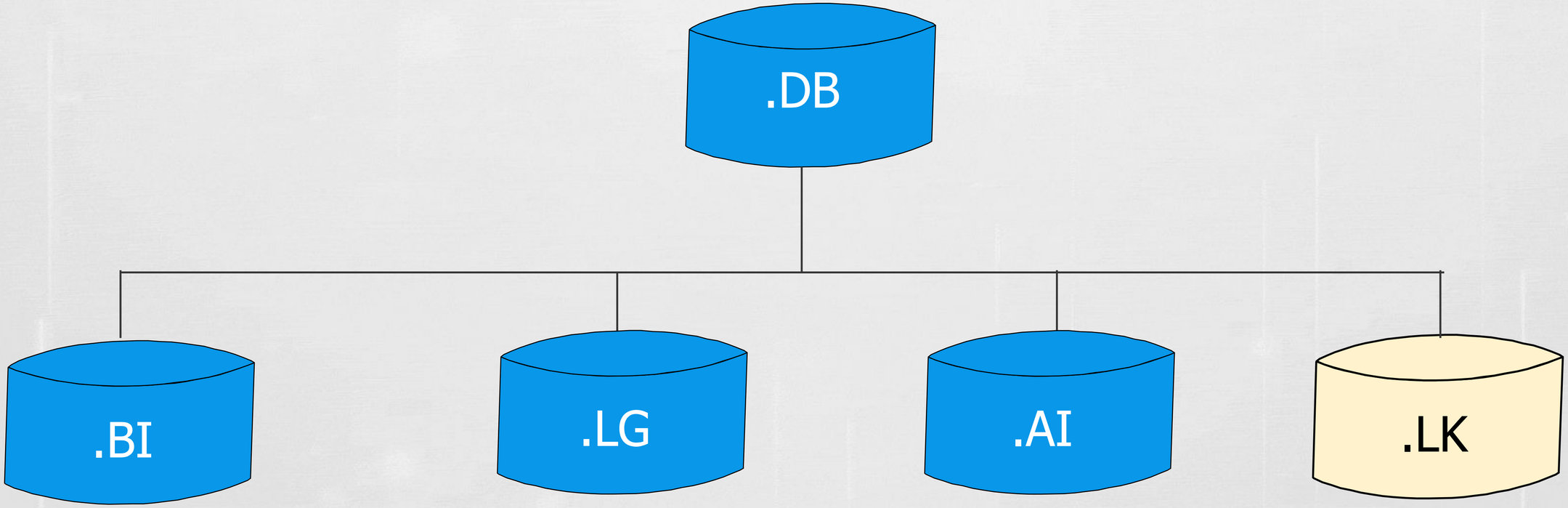
= Campo

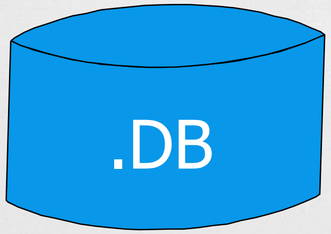


= Índice

COMPONENTES LÓGICOS

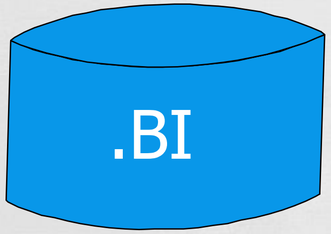
OpenEdge - Banco de Dados





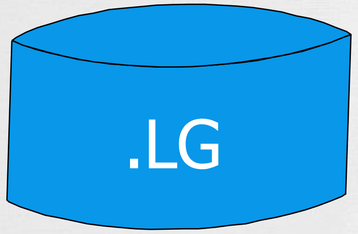
Database

contém os dados atuais e descrições da base



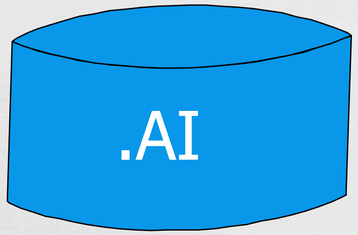
Before-Image

contém a imagem dos dados de uma transação



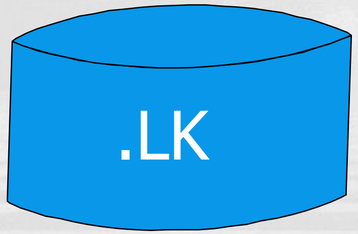
Log

contém as referências dos usuários, hora de entrada, saída e programas acessados (histórico).



After-image

Contém a imagem da base após a transação.



Lock

Informa se a base está sendo acessada.

OpenEdge - Arquivos Importantes (.INI)

Progress.INI - Configurações do Progress

Contém informações importantes sobre a sessão e localização de arquivos do progress.

```
[Startup]
ImmediateDisplay=yes
Use-3D-Size=No
Keep3DFillinBorder=yes
DLC=c:\progress\openedge
PROCFG=c:\progress\openedge\PROGRESS.CFG
PROMSGS=c:\progress\openedge\promsgs
PROPATH=.,c:\progress\openedge\gui,c:\progress\openedge,
    c:\progress\openedge\src
```

OpenEdge - Arquivos Importantes (.PF)

arquivo.PF - Configurações da sessão

Contém as configurações da sessão que será executada, bem como os parâmetros de conexão das bases de dados que serão utilizados na aplicação

-db c:\treinamento\sports.db -ld sports -1
-tok 1600
-s 300 -E -d dmy
-Mm 4096
-cpstream ibm850
-cpterm iso8859-1
-inp 32000 -mmax 6000
-h 800
-T c:\tmp

OpenEdge - Parâmetros do .PF

Principais parâmetros:

-1	carga do banco de dados monousuário
-d	formato de data (-d dmy)
-E	formato europeu de ponto decimal
-T	diretório dos arquivos temporários
-db	nome físico do banco a ser conectado
-ld	nome lógico do banco
-H	nome da máquina onde está o banco
-S	nome do serviço no servidor para o banco
-N	protocolo de comunicação (-N TCP)
-pf	arquivo de parâmetros
-ininame	nome do arquivo .INI utilizado
-basekey	utiliza .INI ou o Registry do Windows (INI)
-cpstream	mapa de caracteres a ser utilizado (ibm850)
-p	programa a ser executado
-Bt	área reservada para temp-tables
-h	número de bases que podem ser conectadas juntas
-RO	Read-Only
-F	Ignora o BI, conectando forçado o DB

OpenEdge - Montando Atalho

- Criar um diretório de trabalho: **c:\treinamento**

- Criar um atalho com:
  Objeto: **c:\progress\openedge\bin\prowin.exe -p _desk.p -d dmy -E**
  Iniciar em: **c:\treinamento**

OpenEdge - Acessando o Desktop

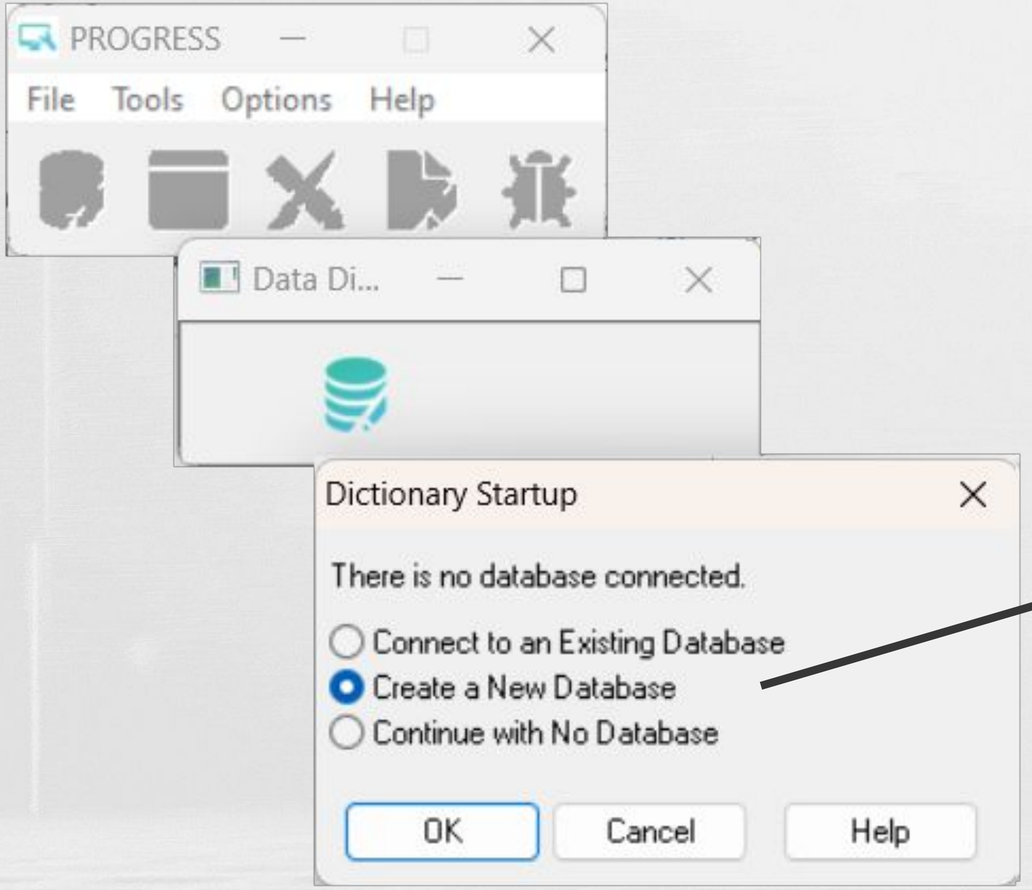
1 2 3 4 5

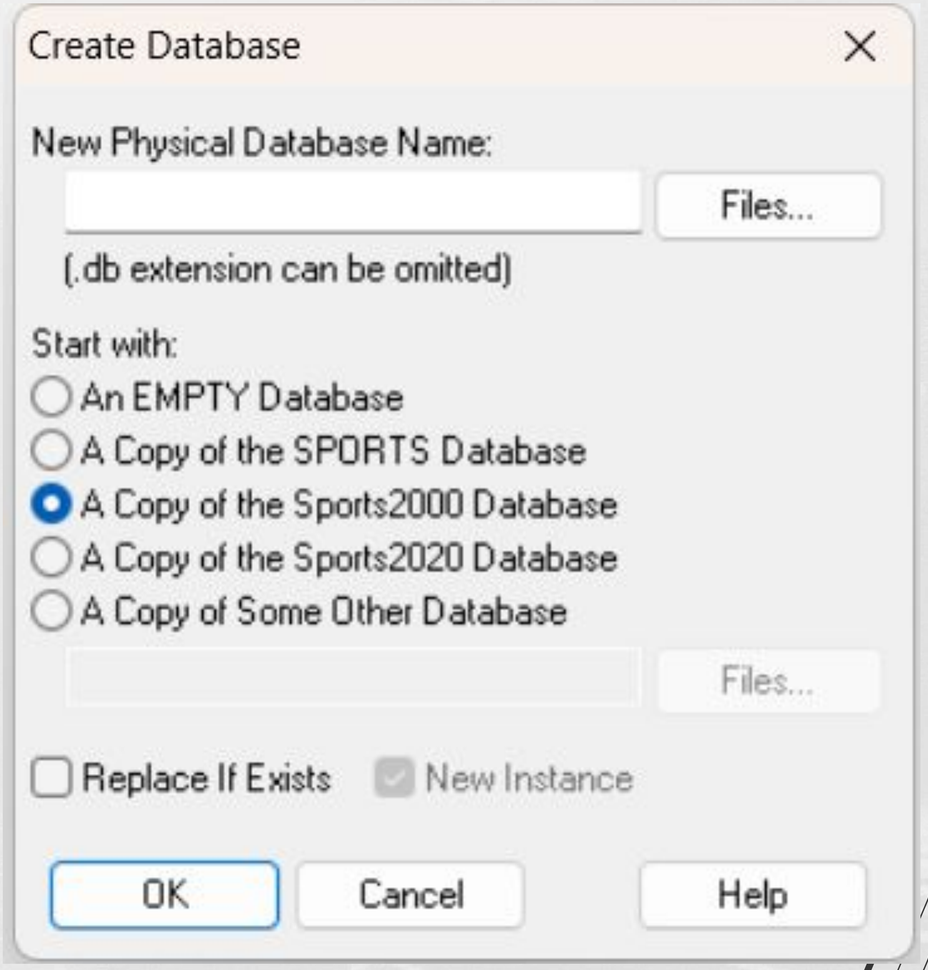
Onde:

1) Data Dictionary
2) Procedure Editor
3) AppBuilder
4) Results (Não utilizado)
5) Debugger

OpenEdge - Dicionário de Bancos de Dados

OpenEdge - Dicionário de Bancos de Dados





OpenEdge - Dicionário de Bancos de Dados

- Acessar o OpenEdge com o atalho criado anteriormente
- Acessar o Dicionário de dados
- Informar o nome do banco de dados: **c:\treinamento\sports.db**

02 Componentes da Linguagem

OpenEdge - Componentes da Linguagem

- Blocos
- Declarações
- Funções
- Operadores
- Variáveis
- Expressões
- Caracteres e símbolos especiais
- Eventos
- Elementos da interface com o usuário

  (atributos e métodos)

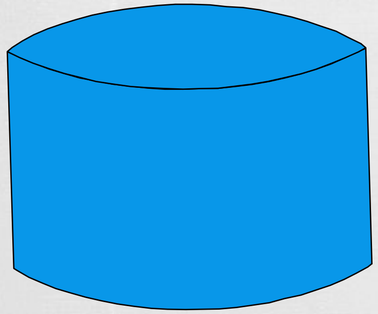
OpenEdge - Tipos de Dados do Progress

Tipo	Formato	Exemplo
Character	x(8) Aceita: A – Alfabetico ! – Maiusculas	“x(40)” –> string alfanumérica “AAA-9999” –> MCZ-9283 “!x(10)” –> STRING ALFANUMÉRICA
Integer	>>>,>>9 Aceita: z – espaço 9 – mostra “0”	“>>>,>>9” –> 1.500 “999,999” –> 001.500 “zzz,zz9” –> 1.500
Decimal	>>>,>>9.99	“>>>,>>9.99” –> 1.500,45 “999,999.99” –> 001.500,45 “zzz,zz9.99” –> 1.500,45
Logical	Yes/No Aceita: True/False	Yes No
Date	MM/DD/AAAA	12/31/9999
Rowid	---	
Recid	>>>>>9	

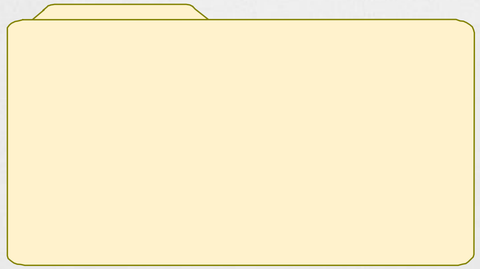
https://docs.progress.com/bundle/abl-reference/page/Data-types.html

03 Movimentação de Dados

Base de Dados → Tela



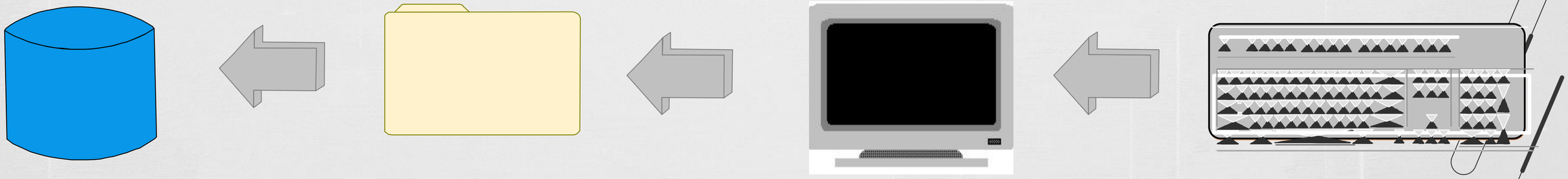
Base de Dados



Buffer Registro

Buffer Tela

Tela → Base de Dados



Base de Dados

Buffer Registro

Buffer Tela

Entrada Usuário

Regras:

Os programas não podem manipular os dados da **base de dados** até que eles sejam copiados para o **buffer de registro**.

O usuário não pode interagir com os dados até que eles sejam copiados do **buffer de registro** para o **buffer de tela**.

As alterações que são feitas no buffer de tela ou buffer de registro não são atualizadas automaticamente em outro buffer. Deve-se programar o controle de movimentação entre os buffers.

OpenEdge - Comandos para Movimentação de Dados

Comando	Banco de Dados	Buffer de Registro	Buffer de Tela	Usuário
ASSIGN		←	●	
CREATE	● →	→		
DELETE	←	●		
DISPLAY		●	→	
ENABLE			←	●
FIND	● →	→		
FOR EACH	● →	→		
GET	● →	→		
INSERT	● →	● → ←	● ←	●
PROMPT-FOR			←	●
RELEASE	←	●		
SET		←	●	← ●
UPDATE		● → ←	→ ●	← ●

04 Manipulando Dados

Comandos - Insert

Cria um novo registro no banco, permite que o usuário faça alterações e atualiza as informações no banco.
Executa as ações de CREATE, DISPLAY, PROMPT-FOR e ASSIGN.

Ex.: **INSERT** customer.

Comandos - Create

Cria um novo registro no banco, inicializa com os valores defaults (padrões) e deixa uma cópia do registro no buffer de registro.

Ex.:
```
CREATE customer.
ASSIGN customer.CustNum = 99.
```

Comandos - Update

Mostra os valores, permite o usuário alterar e atualiza os dados no banco.
Executa as ações de DISPLAY, PROMPT-FOR e ASSIGN.

Ex.: **UPDATE** customer.
UPDATE customer EXCEPT CustNum.

Comandos - Display

Mostra os valores na Tela.
Move os dados do Buffer de Registro para o Buffer de Tela.
Os dados devem sempre ter sido previamente movidos para o Buffer de Registro.

Ex.: **DISPLAY** customer.
DISP customer.CustNum.

Comandos - Set

Permite que o usuário altere as informações e atualiza o banco de dados.
Executa as ações de PROMPT-FOR e ASSIGN.

Ex.: **SET** customer.
SET customer.Name.

Comandos - Prompt-for

Solicita uma entrada de dado ao usuário e disponibiliza no buffer de tela.

Ex.:
```
PROMPT-FOR customer.CustNum.
    FIND customer
        WHERE customer.CustNum =
            INPUT customer.CustNum NO-ERROR.
    IF  AVAILABLE customer THEN
        DISPLAY customer
                EXCEPT customer.Comments
                WITH 1 COLUMN.
```

Comandos - Assign

Atualiza a informação no Banco de Dados.
Copia os dados do buffer de tela para seus respectivos campos e/ou variáveis no buffer de registro.

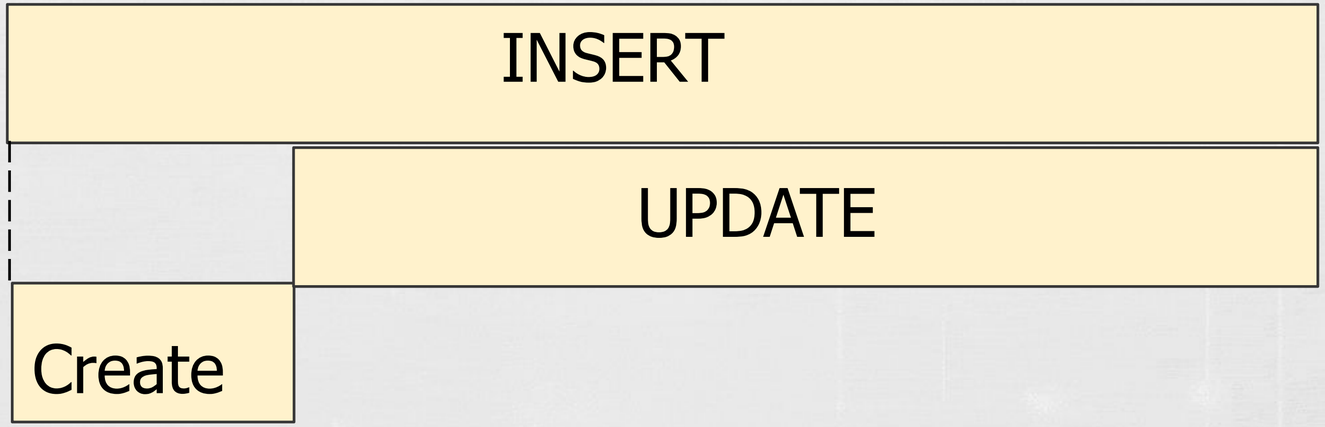
Ex.:
```
DEF VAR i-custnum AS integer NO-UNDO.
PROMPT-FOR i-custnum.
FIND FIRST customer EXCLUSIVE-LOCK NO-ERROR.
ASSIGN customer.CustNum = i-custnum.
```

Comandos - Delete

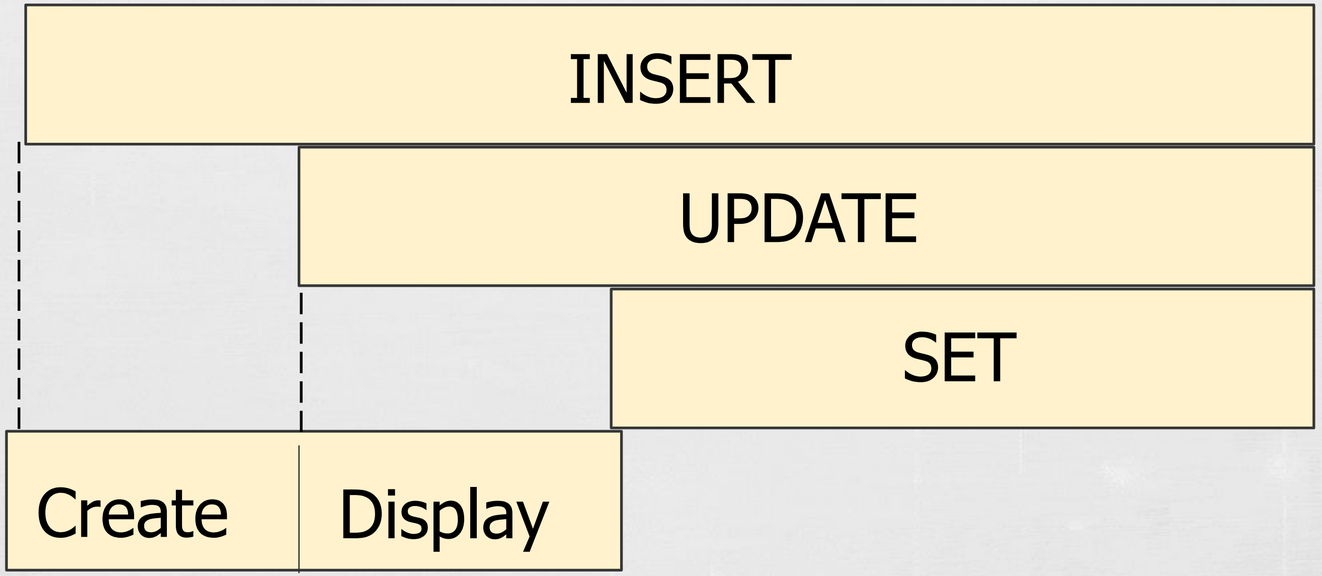
Elimina um registro do buffer de registro e do banco de dados.
Os dados devem sempre ter sido previamente movidos para o Buffer de Registro.

Ex.:
```
REPEAT:
        PROMPT-FOR customer.CustNum.
        FIND customer USING customer.CustNum
            EXCLUSIVE-LOCK NO-ERROR.
        DELETE customer.
END.
```

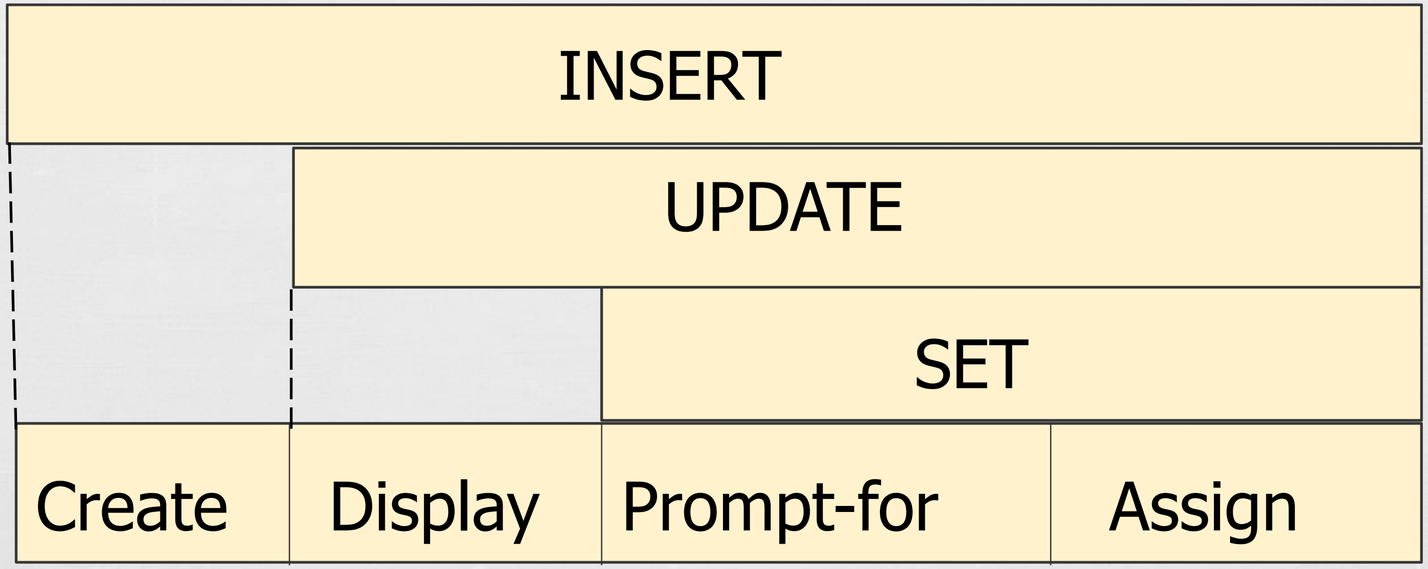
Comandos - Comandos que Substituem Outros Comandos



Comandos - Comandos que Substituem Outros Comandos



Comandos - Comandos que Substituem Outros Comandos



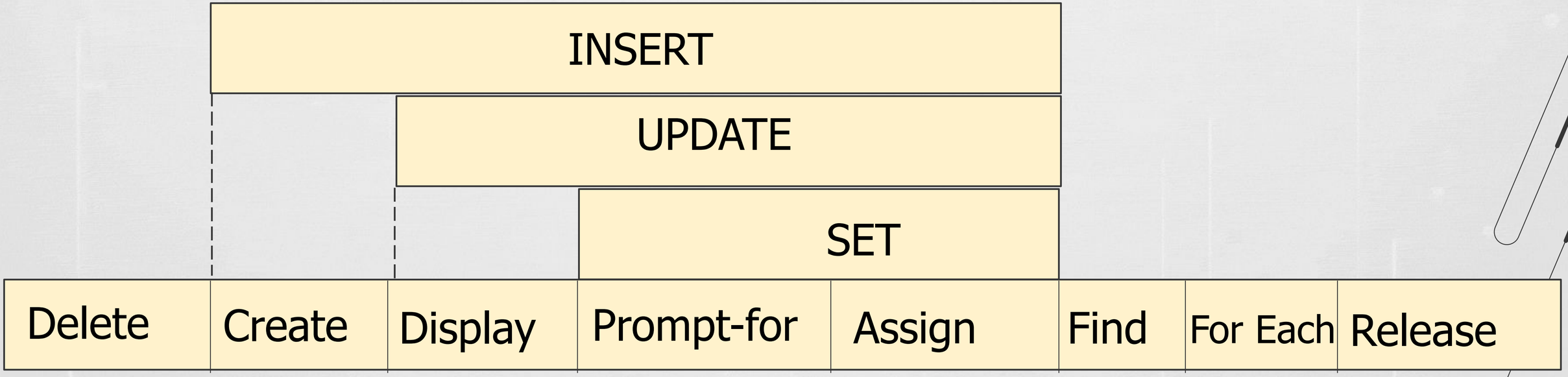
Comandos - Observe a Movimentação

insert customer.

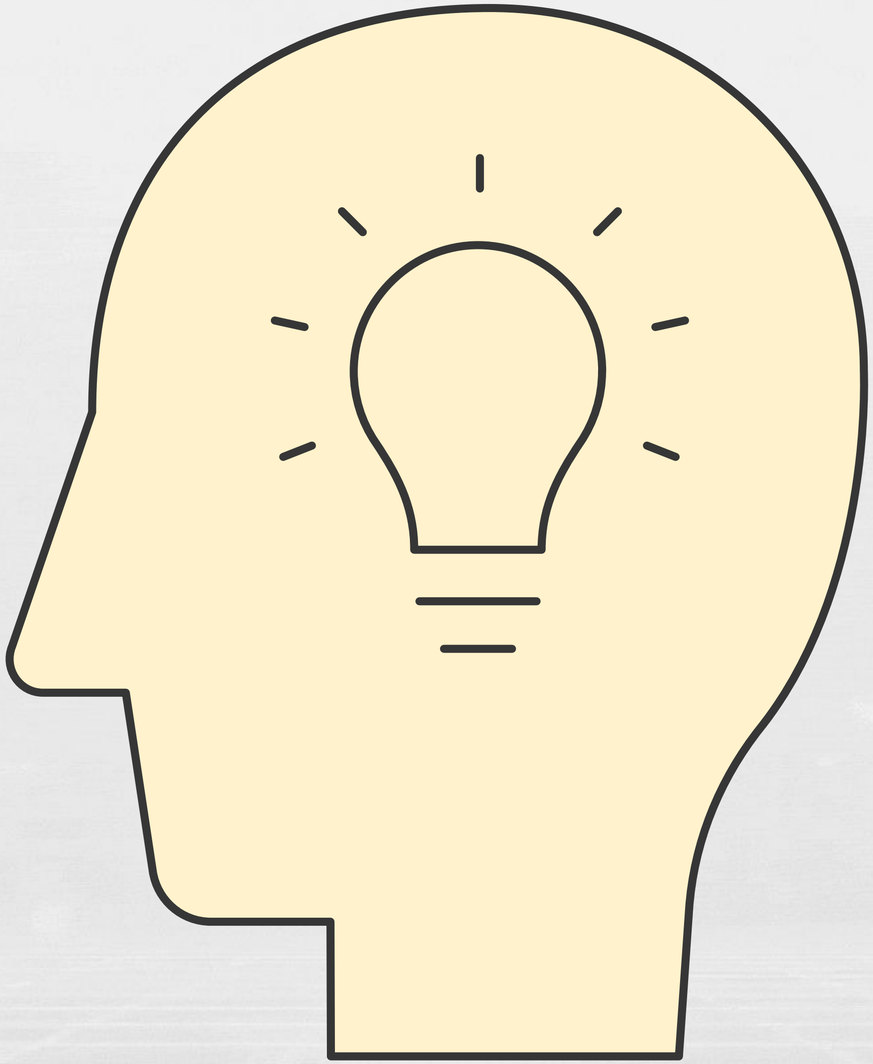
create customer.
update customer.

create customer.
display customer.
prompt-for customer.
assign customer.

Comandos - Quadro de Comandos



Laboratório 1



05 Blocos

Blocos - Tipos de Blocos

- REPEAT
- FOR EACH
- DO
- PROCEDURES
- TRIGGERS

Blocos - Repeat

Laço (repetição) automático. Continua a interagir até encontrar um END-ERROR ou outra condição de término definida pelo usuário.

Ex:

```
REPEAT WITH 1 COLUMN:
        CREATE customer.
        UPDATE customer.
END.
```

Blocos - For Each

Lê um registro a cada interação do bloco e copia os dados do banco para o buffer de registro.

Ex:
```
FOR EACH customer NO-LOCK:
    DISPLAY customer.CustNum
            customer.Name.
END.
```

Blocos - Do

Individualiza um grupo de comandos dentro de um bloco simples.

Ex:
```
FOR EACH customer EXCLUSIVE-LOCK:
        DISPLAY customer.CustNum.
        IF  credit-limit > 15000 THEN DO:
            UPDATE customer.Name
                   customer.Address.
            MESSAGE "Registro Alterado".
        END.
    END.
```

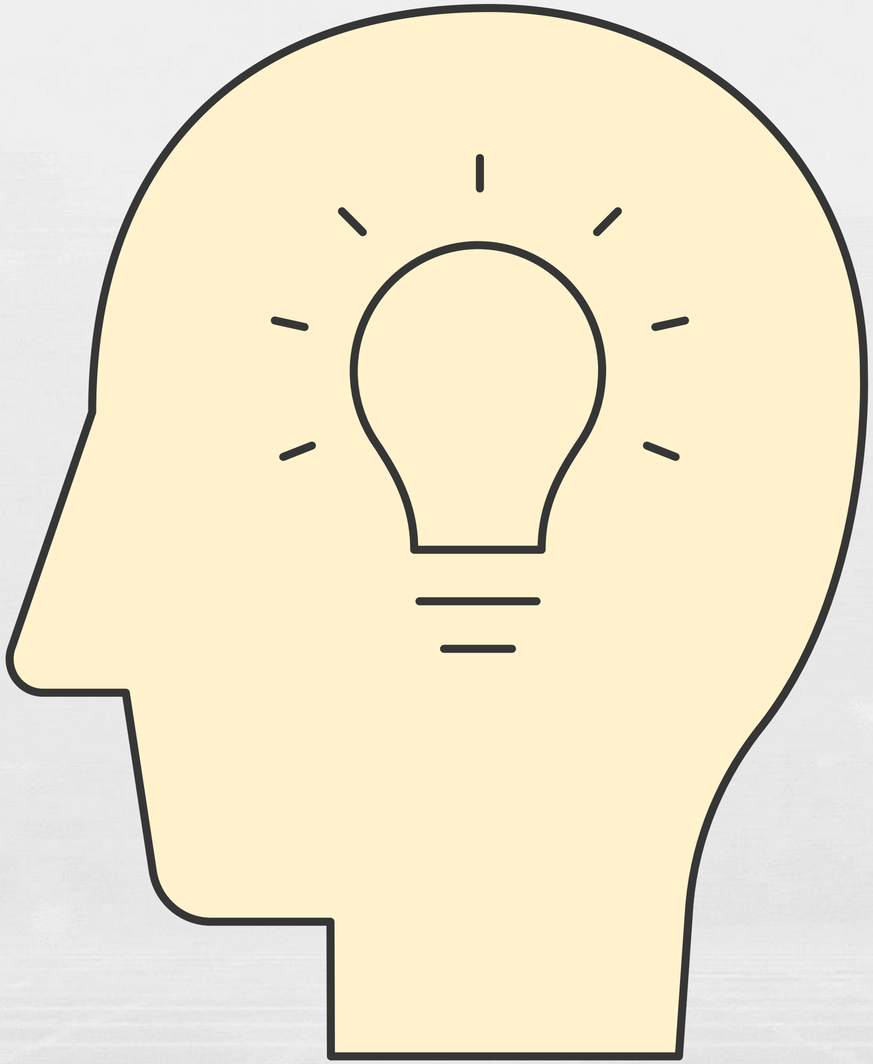
Blocos - Desvio na Execução de Blocos

Permite o desvio na execução do bloco.

NEXT.
LEAVE.

Ex:
```
FOR EACH customer:
        DISPLAY customer.CustNum.
        IF  customer.CreditLimit > 15000 THEN
            NEXT.
        IF  customer.CreditLimit = 10000 THEN
            LEAVE.
        UPDATE customer.Name
                   customer.Address.
    END.
```

Laboratório 2



06 Localizando Registros

Localizando Registros - Find

Busca apenas um registro em uma tabela.

FIRST (primeiro) LAST (último)
NEXT (próximo) PREV (anterior)

Ex.:
```
FIND FIRST customer
        WHERE customer.CreditLimit > 15000
        NO-LOCK NO-ERROR.
    IF  AVAILABLE customer THEN
        DISPLAY customer WITH 1 COLUMN.
```

Localizando Registros - Cláusula Where

Limita os registros a serem mostrados em uma leitura de uma tabela.

```
Ex:  FOR EACH customer NO-LOCK
            WHERE customer.CreditLimit > 15000:
            DISPLAY customer.CustNum
                    customer.CreditLimit.
     END.
```

Operadores:

NOT	**>**	**<**
AND	**>=**	**<=**
OR	**<>**	

Localizando Registros - Cláusula Using

Substitui o WHERE, utilizando o dado informado no buffer de tela.

Ex:
```
REPEAT WITH 1 COLUMN:
        PROMPT-FOR customer.CustNum.
        FIND customer USING customer.CustNum.
        DISPLAY customer.
END.
```

WHERE customer.CustNum = INPUT customer.CustNum

Localizando Registros - Cláusula Of

Substitui o WHERE, utiliza o índice para fazer o relacionamento entre tabelas.

Ex:
```
FIND FIRST customer NO-LOCK NO-ERROR.
FOR EACH order OF customer NO-LOCK:
    DISPLAY order.OrderNum.
END.
```

WHERE order.CustNum = customer.CustNum

Localizando Registros - Cláusula By

Usa-se a opção BY para classificação de registros por um campo não indexado.

```
Ex: FOR EACH customer NO-LOCK
            BY balance DESCENDING:
            DISPLAY customer.Balance
                    customer.Name
                    customer.Phone.
    END.
```

Default: ordenação ascendente

Localizando Registros - Cláusula Use-index

A opção USE-INDEX permite você escolher qualquer índice definido, para selecionar registros.

```
Ex:  FOR EACH customer NO-LOCK
            USE-INDEX name:
            DISPLAY customer.Balance
                    customer.Name
                    customer.Phone.
     END.
```

Localizando Registros - Cláusula Begins

Verifica se a 1. expressão inicia com a 2. expressão.
Utiliza um índice caso exista.

Ex:
```
FOR EACH customer NO-LOCK
        WHERE customer.Name BEGINS "S":
        DISPLAY customer.CustNum
                customer.Name
                customer.Phone.
END.
```

Localizando Registros - Cláusula Matches

Compara uma expressão do tipo caracter com um padrão e retorna o valor verdadeiro se a expressão for atendida.
Não utiliza índice.

Ex:
```
FOR EACH customer NO-LOCK
        WHERE customer.Address MATCHES "*st":
        DISPLAY customer.CustNum
                customer.Name
                customer.Address.
END.
```

Localizando Registros - Leitura de Registros

BEGINS	MATCHES
• Pesquisa por um campo que comece com um campo caracter especificado	• Pesquisa por um campo que atenda um padrão de caracteres especificado
• Usa índice para fazer a pesquisa, não precisa ler todos os registro da tabela	• Busca todos os registros da tabela para poder fazer a comparação
• Não se pode utilizar coringas	• Pode ser usado coringas (* e .)

Localizando Registros - Cláusula Contains

Faz a busca por determinada palavra ou combinação delas, em atributos do tipo word-index.

Ex:
```
FOR EACH item NO-LOCK
        WHERE item.ItemName CONTAINS "ball":
        DISPLAY item.ItemNum
                item.ItemName
                item.Price.
    END.
```

* A tabela deverá possuir um índice do tipo **WORD-INDEX** para o campo pesquisado

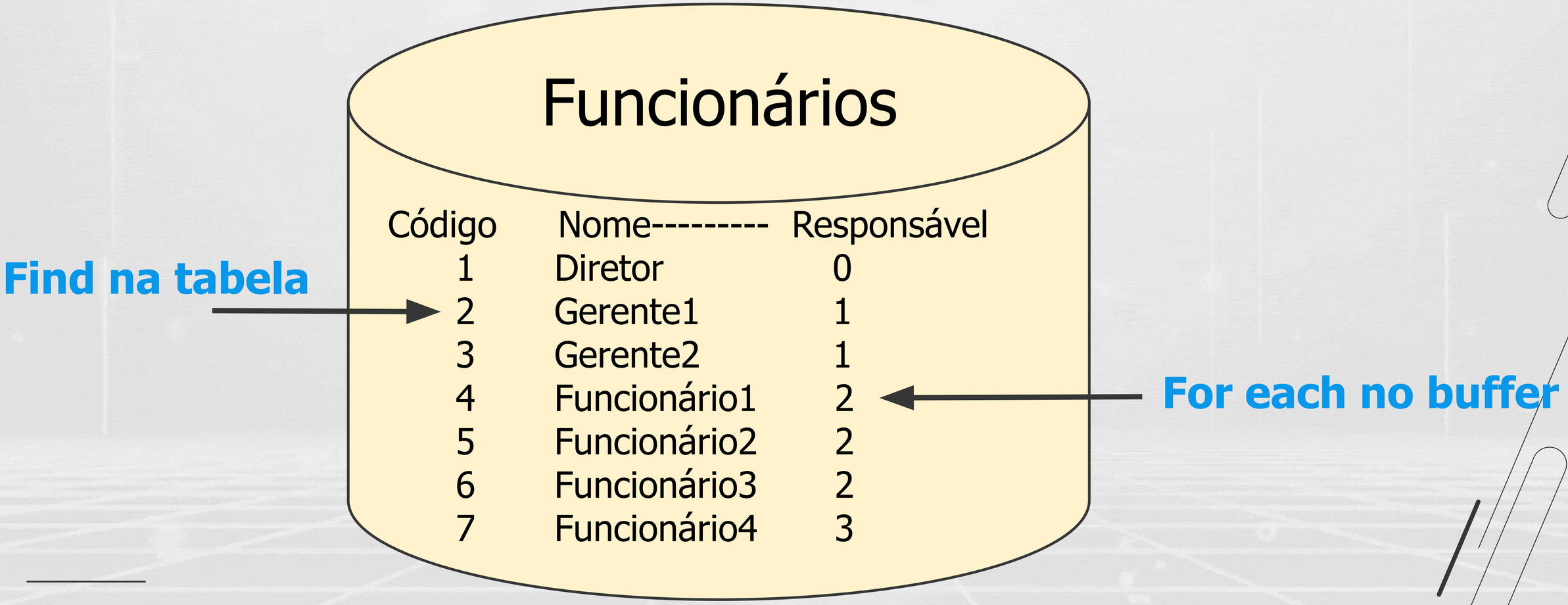
Localizando Registros - Cláusula Fields

- Traz somente os campos desejados
- Funciona somente em leituras do tipo NO-LOCK
- Todos os campos dentro do FOR EACH devem estar na cláusula FIELDS.

Ex:
```
FOR EACH item NO-LOCK
        FIELDS (ItemNum ItemName Price)
        WHERE item.ItemName BEGINS "a":
        DISPLAY item.ItemNum
                item.PtemName
                item.Price.
END.
```

Buffer

- É uma imagem/espelho/ponteiro da tabela
- Todas as alterações feitas na tabela são refletidas no Buffer e vice-versa



Buffer


```
DEFINE BUFFER b-aux FOR customer.

FIND FIRST customer EXCLUSIVE-LOCK NO-ERROR.
UPDATE customer.CustNum
       customer.Name.
CREATE b-aux.
ASSIGN b-aux.CustNum = customer.CustNum + 1
       b-aux.Name    = customer.Name.
DISPLAY customer.CustNum
        customer.Name
        b-aux.CustNum
        b-aux.Name
       WITH 1 COLUMN.
```

Query

Pesquisa agrupada de registro do banco de dados.

- Definir a query
  Ex.: **DEFINE QUERY** query-name
  FOR table **SCROLLING**.

- Abrir a query
  Ex.: **OPEN QUERY** query-name **FOR EACH** table

- Acessar os registros
  Ex.: **GET FIRST** query-name.

Query

Retorna um registro de uma query previamente aberta.

```
GET FIRST query-name.
    NEXT
    LAST
    PREV
```

Query

Ex.:
```
DEFINE QUERY qr-cust
                    FOR customer SCROLLING.

OPEN QUERY qr-cust FOR EACH customer.

GET FIRST qr-cust.
REPEAT:
    DISP customer.
    GET NEXT qr-cust.
END.
```

ÍNDICES

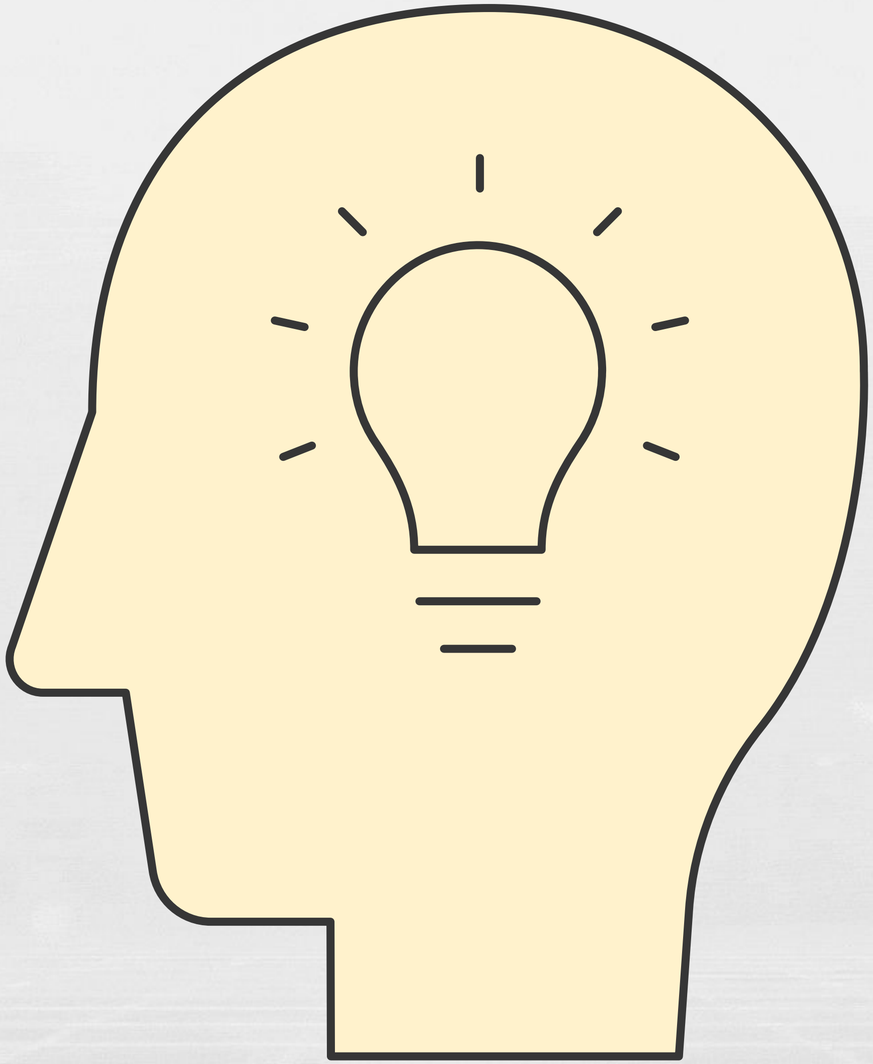
- Equality Match.
- Range Match.
- Sort Match.
- Word-index.

Leitura de Registros

WHERE	Equality Match
order.ordernum = 3	Sim
order.sdate = today + 30	Sim
order.ordernum = customer.maxcredit	Sim
order.sdate = order.pdate	Não

WHERE	Range Match
order.ordernum >= 5	Sim
order.odate < today - 15	Sim
order.ordernum > customer.maxcredit	Sim
order.sdate >= order.odate + 30	Não

Laboratório 3



07 Variáveis e Mensagens

Variáveis

```
DEFINE VARIABLE nome-variavel
        AS tipo-de-dado
    LIKE Tabela.atributo
            INITIAL "xxxx"
            NO-UNDO.
```

Ex.:
```
DEFINE VARIABLE c-customer AS char NO-UNDO.
DEF VAR i-custnum LIKE customer.CustNum NO-UNDO.
```

Mensagens

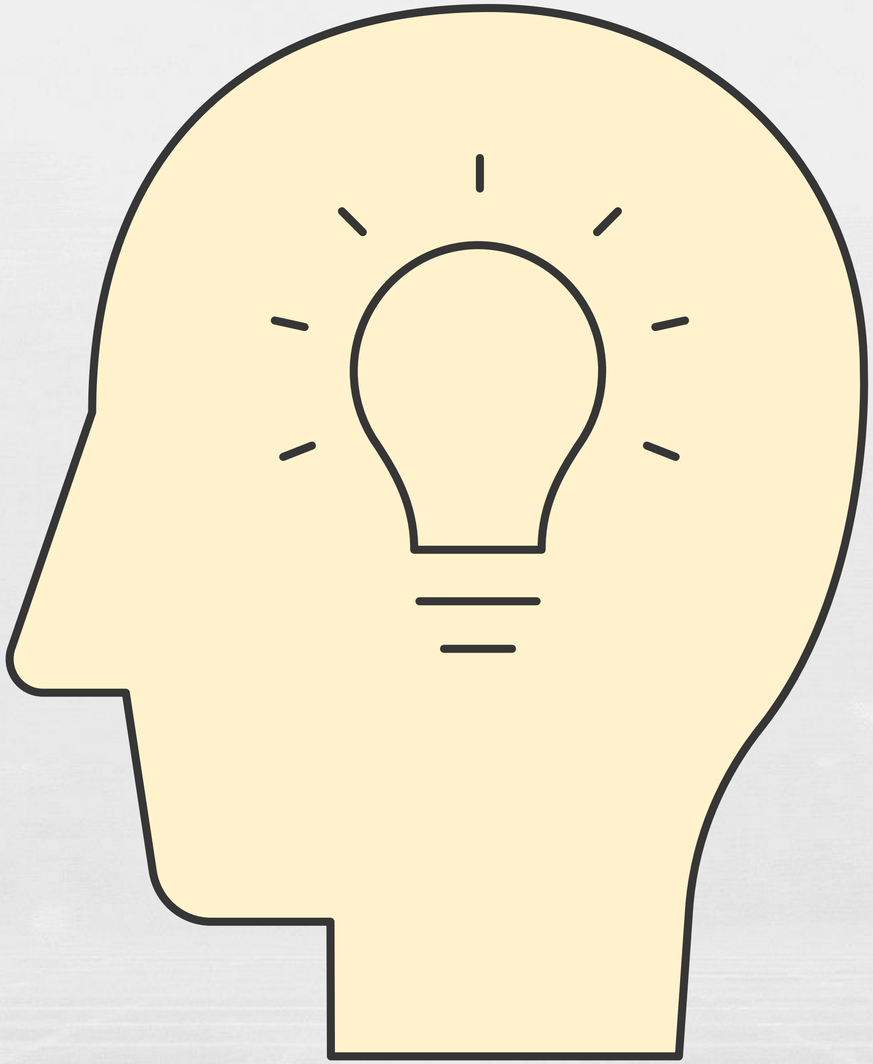
Envia uma mensagem para a tela.

```
MESSAGE "texto" [ UPDATE variavel ]
                [ VIEW-AS ALERT-BOX
                      [QUESTION / INFORMATION
                        WARNING / ERROR ]
                      [ BUTTONS YES-NO
                                YES-NO-CANCEL
                                OK
                                OK-CANCEL
                                RETRY-CANCEL ]
                [ TITLE "título" ].
```

Variáveis e Mensagens

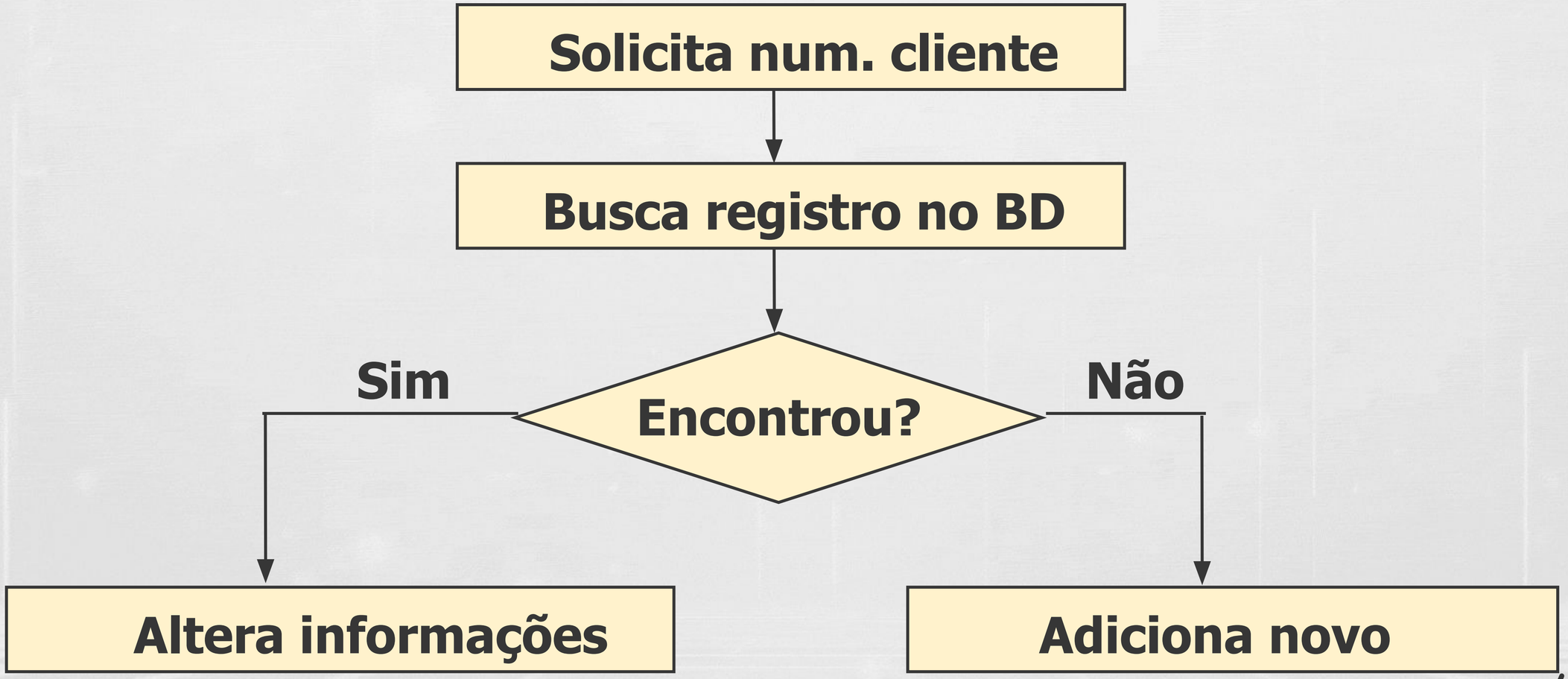
```
DEF VAR l-resp AS logical INITIAL no NO-UNDO.

MESSAGE "Confirma Atualização ?"
        UPDATE l-resp
        VIEW-AS ALERT-BOX
            QUESTION BUTTONS YES-NO
            TITLE "Atualização de Dados".
```

Laboratório 4



08 Processamento Condicional

Processamento Condicional



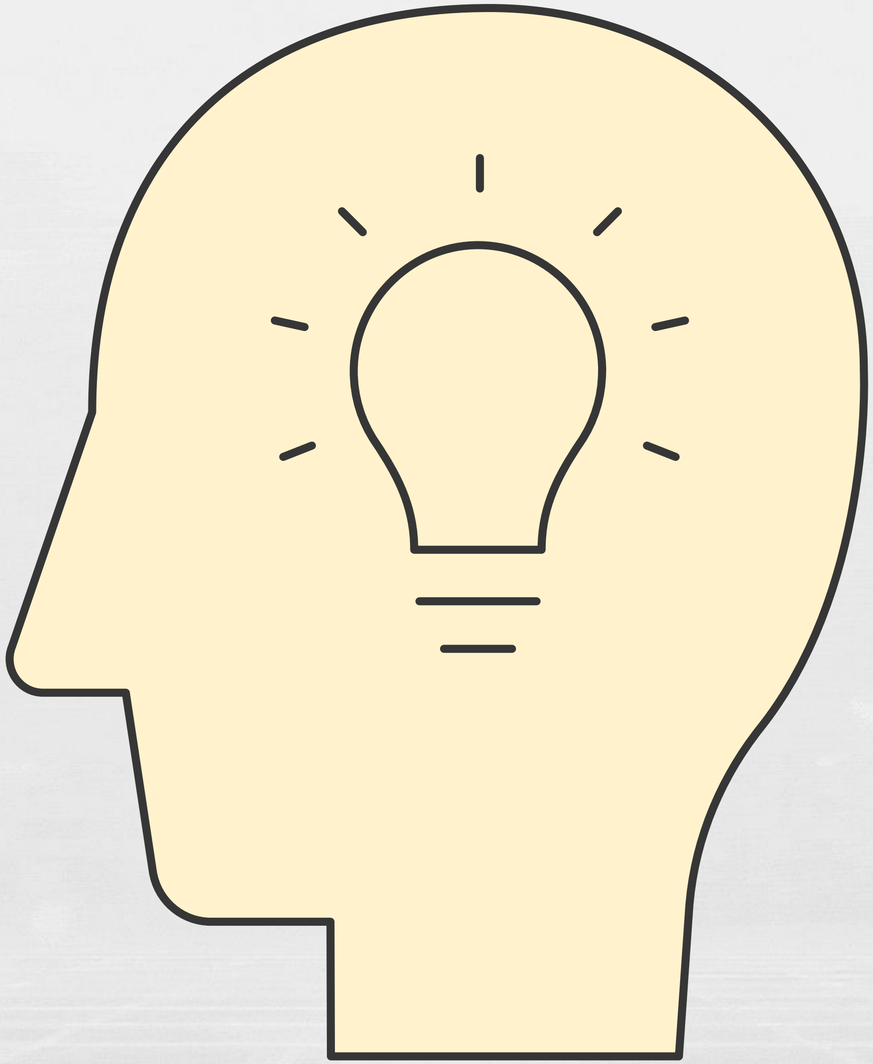
Processamento Condicional - IF


```
IF  expression THEN
        comando.
ELSE DO:
        bloco.
END.
```

Processamento Condicional - Case


```
CASE expression:
    WHEN value THEN
        comando.
    WHEN value THEN DO:
        bloco
    END.
    WHEN value THEN
        comando.
    OTHERWISE DO:
        bloco
    END.
END CASE.
```

Laboratório 5



09 Frames

Frames

É uma área retangular dentro da tela do Progress usada para mostrar informações.

Layout default para seus dados.

Espaço disponível:

- Frame (21 linhas)
- Mensagens (2 linhas)
- Linha de status (1 linha)

Frames

As frames default do progress ocorrem nos seguintes blocos :

- Procedure
- Repeat
- For each
- Do with frame

Frames

```
DISPLAY "Relatório de Clientes"
            WITH FRAME f-abc CENTERED.
FOR EACH customer:
      DISPLAY customer.CustNum customer.Name
                  WITH FRAME xxx DOWN.
END.
```

one-down: dados de uma única interação do bloco

down: dados de múltiplas interações do bloco.

Frames

Define uma frame para uso por um procedimento ou por diversos procedimentos.

Ex.: **DEFINE FRAME** f-dados

```
        customer.CustNum   NO-LABEL   SKIP(2)
        customer.Name      AT 5       SKIP
        customer.Address   AT 5
        WITH SIDE-LABELS OVERLAY
             KEEP-TAB-ORDER NO-VALIDATE.
```

Frames - Form

Permite definir o posicionamento dos campos.

O Form estará sempre contido dentro de uma Frame. Uma Frame pode possuir várias Forms.

Caso a Frame não tenha sido informada ou não exista, o progress cria automaticamente uma Frame.

Ex.:

```
FORM
    customer.CustNum   NO-LABEL   SKIP(2)
    customer.Name      AT 5       SKIP
    customer.Address   AT 5
    WITH FRAME f-dados.
```

Frames - Opções de Posicionamento

AT - Alinhamento pela direita.

F1: Short
Fld2: Medium
Field3: Long

COLON - Alinhamento pelos dois pontos.

F1: Short
Fld2: Medium
Field3: Long

TO - Alinhamento pela esquerda.

F1: Short
Fld2: Medium
Field3: Long

Frames - Manipulando Frames

VIEW - apresenta uma frame.

Ex.: VIEW FRAME frame-name.

HIDE - apaga uma frame.

Ex.: HIDE FRAME frame-name NO-PAUSE.

CLEAR - apaga o conteúdo de uma frame.

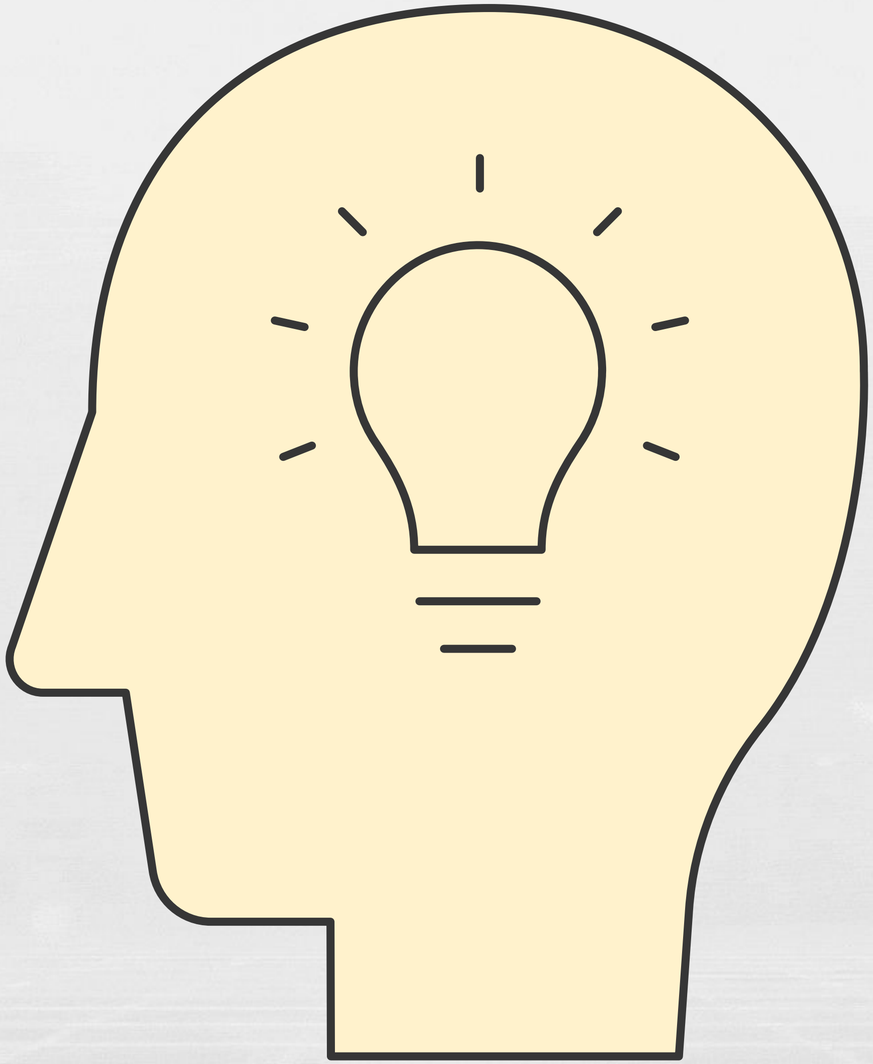
Ex.: CLEAR FRAME f-arq ALL NO-PAUSE.

PAUSE - suspende a execução até que uma tecla seja pressionada ou um certo tempo passar.

Ex.: PAUSE.

PAUSE 5.

Laboratório 6



10 Includes

Includes

Em tempo de compilação PROGRESS lê estes arquivos include como sendo parte do programa.

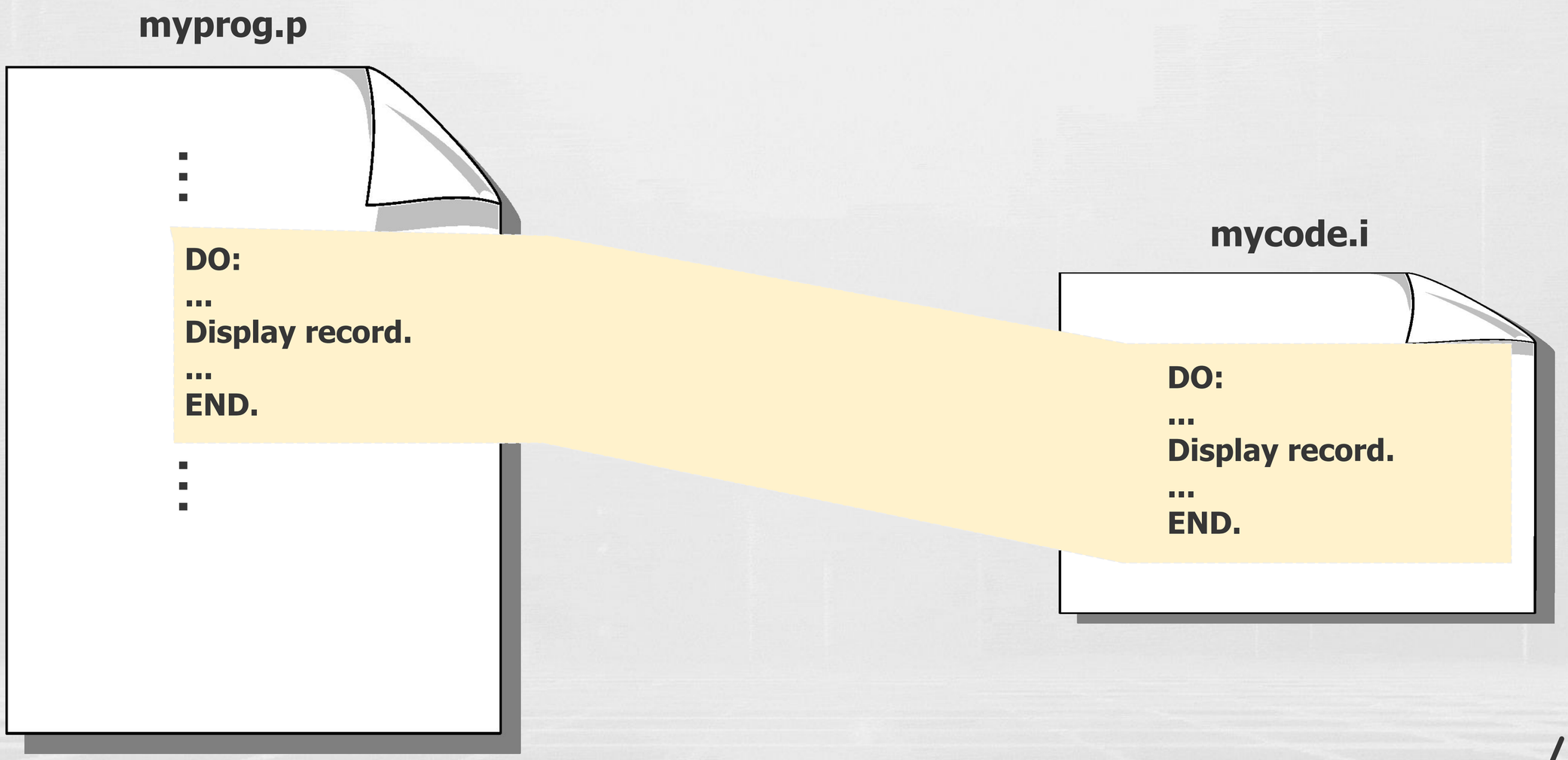
Vantagem: reutilização de código.

Ex.: {i-valida.i}

{i-valida.i "Código Inválido"}

```
/* i-valida.i */
MESSAGE "Validando {1}"
        VIEW-AS ALERT-BOX.
```

Includes - Expansões do Program em Tempo de Compilação



Includes - Código Modular com Arquivos de Includes

myprog.p

```
:
{mycode.i}

:
{mycode.i}
```

mycode.i

```
DO:
   ...
   Display record.
   ...
END.
```

Includes - Usando Arquivos de Include para Definir Variáveis

mycode.i

```
DEFINE VARIABLE...
```

prog1.p

```
{mycode.i}
:
:
```

prog2.p

```
{mycode.i}
:
:
```

Includes

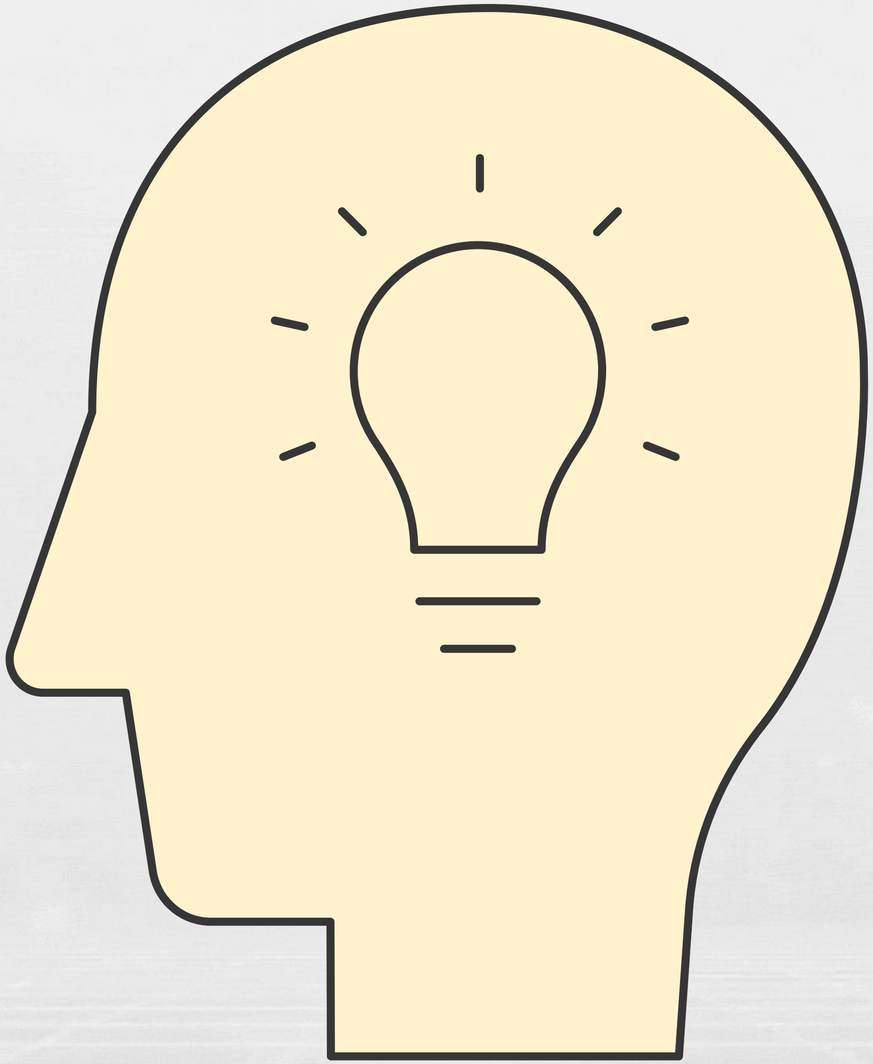

```
{i-mostra.i &table="customer"
            &field="CustNum"}

/* i-mostra.i */
FOR EACH {&table}:
    DISPLAY {&table}.{&field}.
END.
```

Includes - Observação

Não deve ser utilizado a barra "\" nos nomes de arquivos ou includes para informar o diretório, pois causa erro no UNIX.

Quando for necessário utilizar barras para informar diretório, deve-se utilizar a barra "/".

Laboratório 7



Laboratório 7

1. Escreva um programa **l-include.p** que possua uma include genérica para mostrar dados de uma determinada tabela, onde sejam passados como parâmetros o nome da tabela e uma lista com 2 atributos.

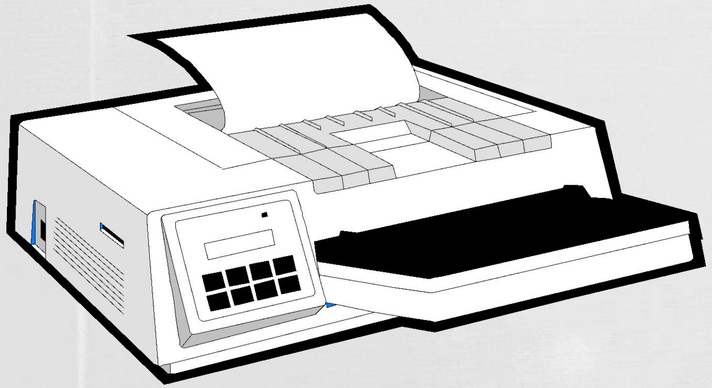
2. Modifique o programa **l-include.p**, chamando-o de **l-inc2.p**, de modo que utilizando a mesma include mostre os clientes e os estados.

11 Relatórios

Relatórios - Redirecionando a Saída

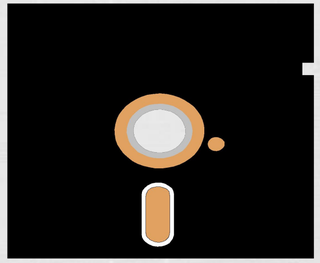
OUTPUT TO <destino>.

PRINTER



TERM

arquivo



Relatórios - Stream

Permite definir saída simultâneas.
O Progress permite até 5 saídas diferentes por programa.

As quatro regras básicas para utilização de stream:

1. Definir
2. Direcionar
3. Utilizar
4. Fechar

```
(1) DEFINE STREAM s1.
(2) OUTPUT STREAM s1 TO arquivo.txt APPEND.
(3) PUT STREAM s1 UNFORMATTED SKIP.
(4) OUTPUT STREAM s1 CLOSE.
```

Relatórios - Funções Automáticas

TOTAL

COUNT

MAXIMUM

MINIMUM

AVERAGE

Os labels são mostrados na ordem especificada acima (em inglês).

Relatórios - Totalizando Valores

ACCUMULATE - Calcula uma ou mais expressões de totalização dentro de um bloco de interação.

ACCUM - Acessa os valores acumulados pela função accumulate.

Relatórios - Quebras

BREAK BY - Define categorias de quebra.

FIRST-OF - Retorna verdadeiro se a iteração atual for a primeira do grupo de quebra.

LAST-OF - Retorna verdadeiro se a iteração atual for a última do grupo de quebra.

FIRST - Retorna verdadeiro se a iteração atual for a primeira.

LAST - Retorna verdadeiro se a iteração atual for a última.

Ex.:

```
FOR EACH customer
        BREAK BY customer.SalesRep:
        IF FIRST-OF(customer.SalesRep) THEN
            DISPLAY customer.SalesRep
                        customer.Name.
END.
```

Relatórios - Quebras

Ex:

Represent.	Num Cliente	Limite Crédito
aaa	1	3.000
aaa	3	60.000
		63.000
bbb	4	5.000
bbb	8	50.000
		55.000
		118.000

Relatórios - Cabeçalhos e Rodapés

Opção HEADER

PAGE-TOP (depende da HEADER)

PAGE-BOTTOM (depende da HEADER)

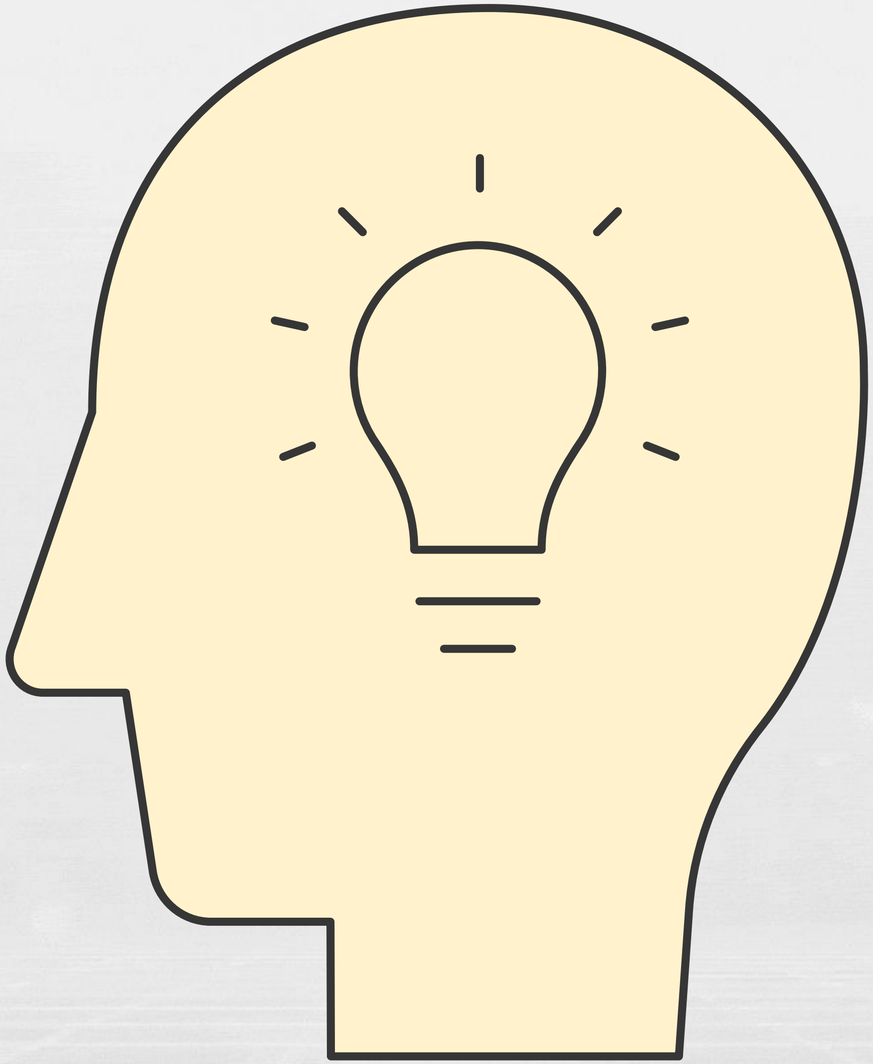
na definição da frame.

VIEW / HIDE - Ativar/Desativar frames.

PAGE-SIZE - número de linhas por página.

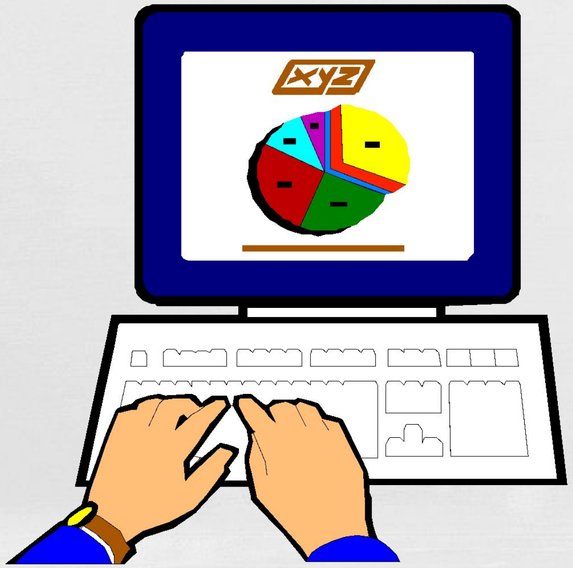
PAGE-NUMBER - retorna o número da página.

Laboratório 8



12 Programação Orientada a Eventos

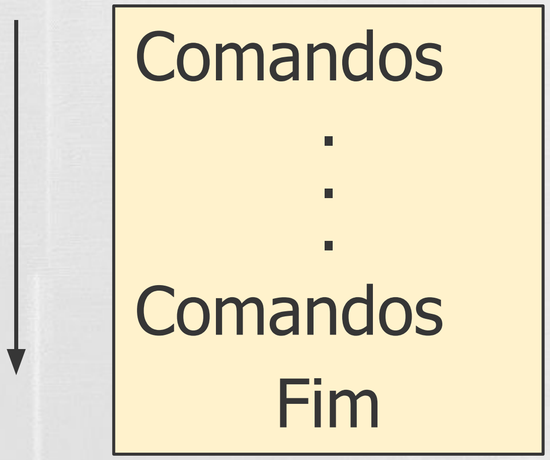
Usuário dirige a execução através do acionamento dos componentes da interface.



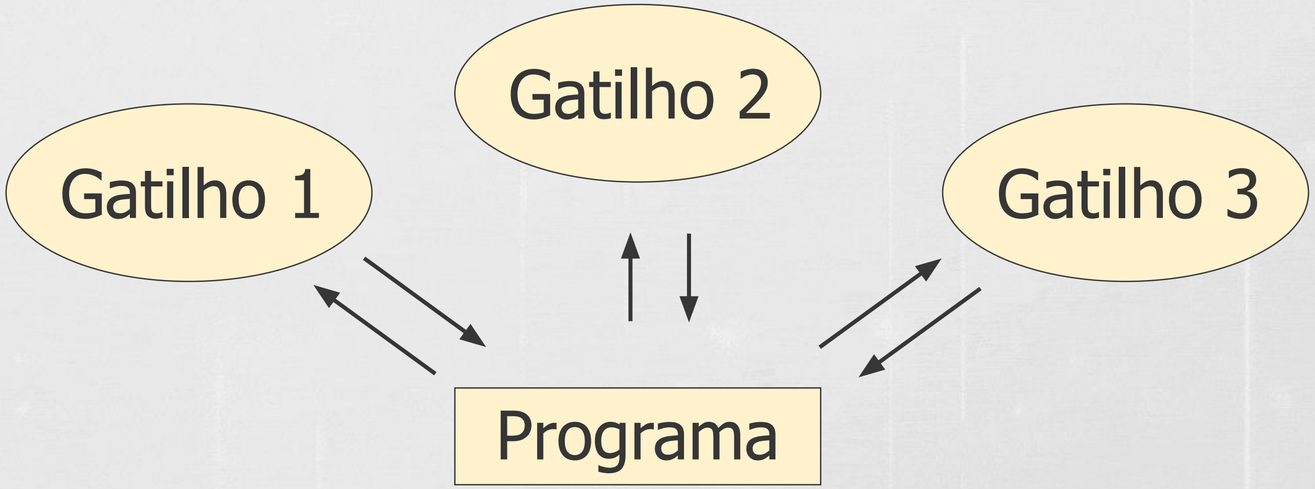
A interface é composta por **widgets**, que geram diferentes eventos.

Comportamento dos Programas

Procedural



Eventos



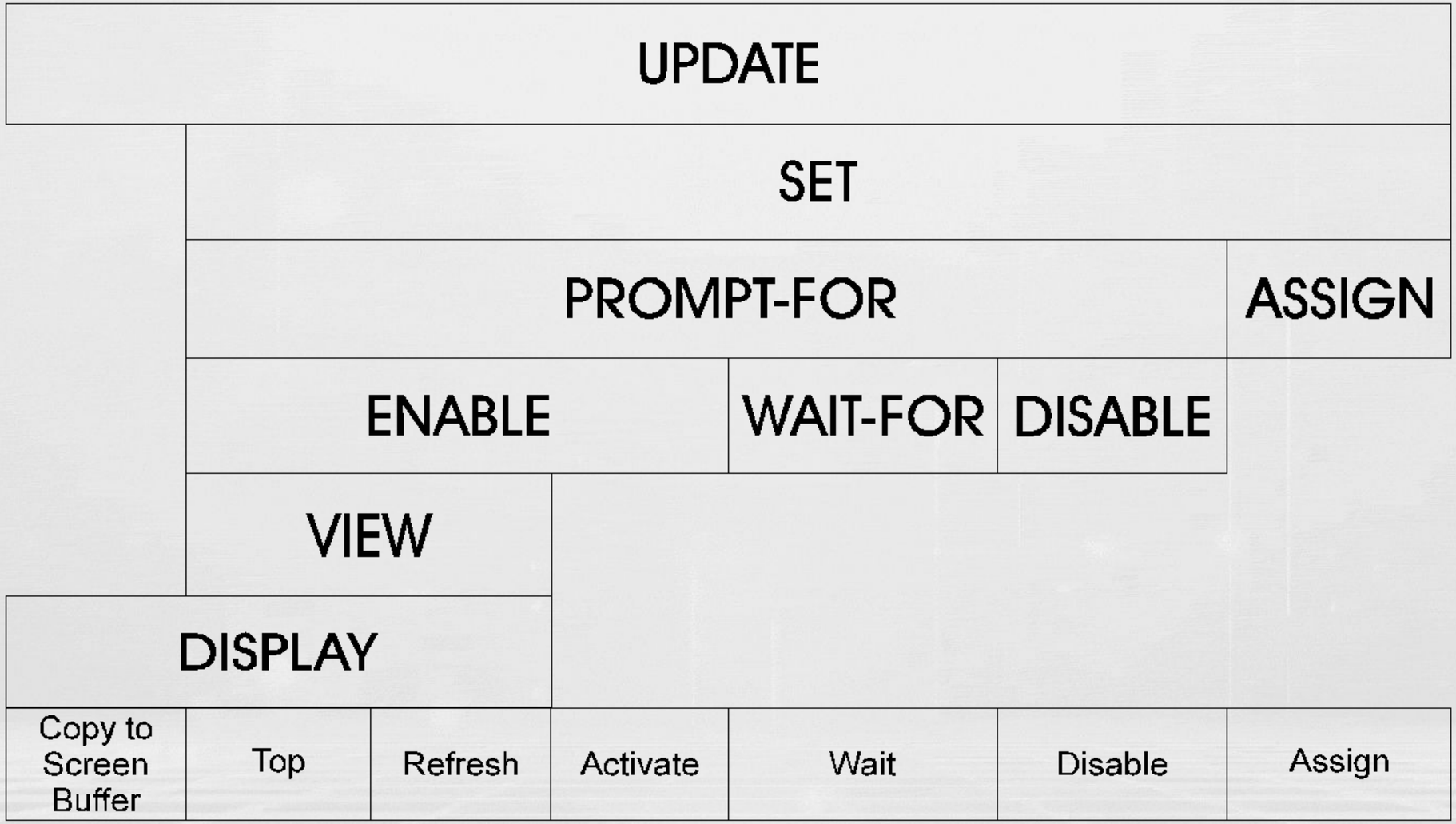
Eventos

Ações do usuário ou do Progress que ocorrem durante a execução de uma aplicação.

Tipos de Eventos:

- Eventos causados pelo usuário (acesso a elementos da interface)
- Eventos relacionados ao banco de dados

Comandos de Construção de Blocos



Como Utilizar um Objeto

Passo	Sintaxe
Definir o objeto e mostrá-lo	DEFINE...
Definir os triggers para o objeto	ON event OF object DO: ... END.
Mostrar o objeto na frame	DISPLAY...WITH FRAME...
Executar o bloco principal do programa	WAIT-FOR event OF object

Existem 4 tipos básicos de Widgets

- **Window Widget**: espaço de trabalho da aplicação. Criada quando a aplicação for iniciada.
- **Frame Widget**: recipiente para outros widgets.
- **Data Widget**: forma de representação dos dados. São eles: fill-in, text, editor, combo-box, etc.
- **Action Widget**: forma de representar os comandos. Botões e menus compõem os widgets de ação. Esses widgets permitem que o usuário dirija a operação de sua aplicação.

ATRIBUTOS

Apresentam/modificam as características dos WIDGETS.

Ex.: ASSIGN wgh_retang1:**bgcolor** IN FRAME f_dados = 8.

MÉTODOS

Retornam/adicionam informações/dados dos WIDGETS.

OBS: Normalmente retorna uma expressão lógica.

Ex.: ASSIGN v_stat = v_estab:**load-mouse-pointer("cross.cur")**
IN FRAME f_dados.

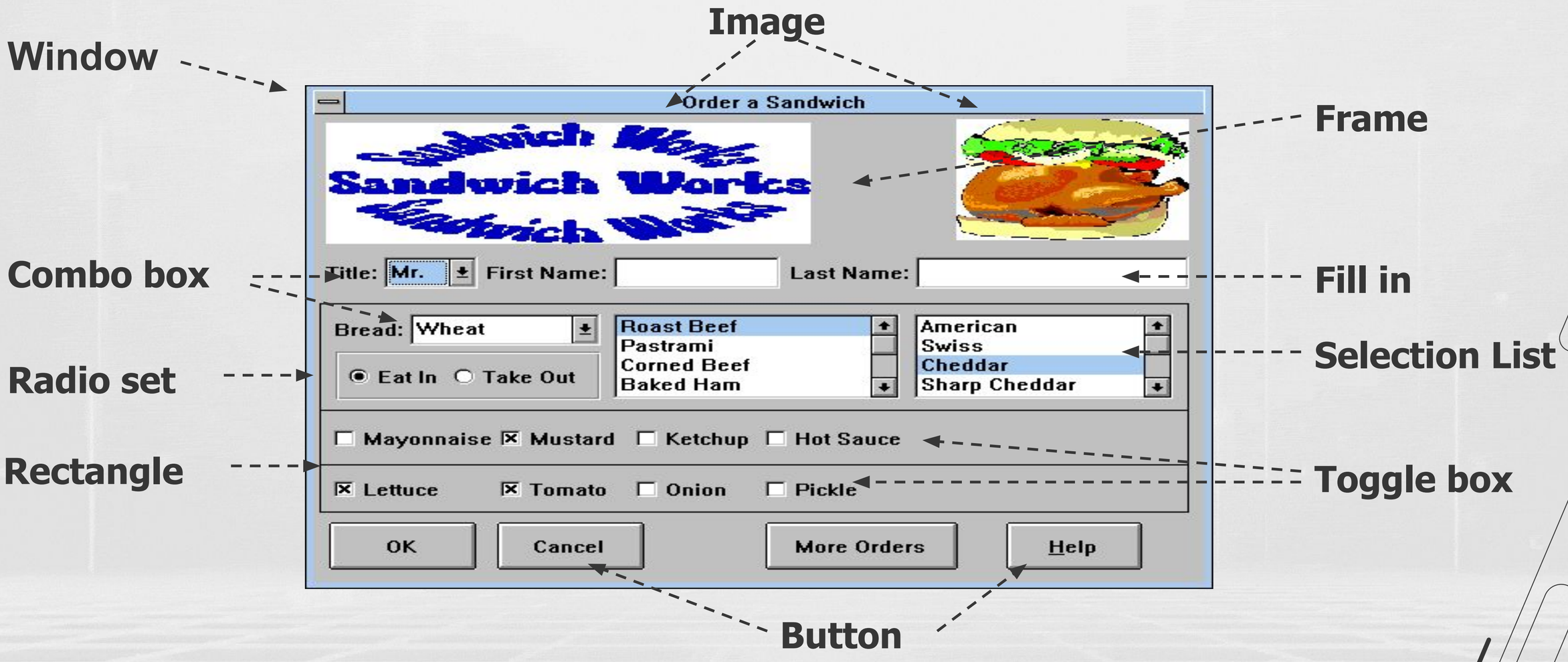
Triggers

Bloco de código executado sempre que um evento ocorre.

```
ON event-list OF widget-list
    trigger-block
```

Ex.:
```
ON choose OF bt-cancela DO:
        MESSAGE "Cancelado pelo usuário"
                VIEW-AS ALERT-BOX INFORMATION.
        QUIT.
    END.
```

Objetos Gráficos



Tipos de Widgets - Button

```
DEF BUTTON bt-sair     LABEL "Sair" AUTO-ENDKEY.
DEF BUTTON bt-ok       LABEL "Ok"  AUTO-GO.

DEF BUTTON bt-atualizar  LABEL "Atualizar".
DEF BUTTON bt-desfazer  LABEL "Desfazer" IMAGE FILE "im-undo".
```

Atributos
- Label
- Sensitive
- Visible
- Tooltip

Métodos
- Load-mouse-pointer
- Load-image
- Load-image-down
- Load-image-up
- Load-image-insensitive

Eventos
- Entry
- Leave
- Choose

View-as

Use VIEW-AS para especificar os widgets de representação dos dados.

Ex. DEFINE VARIABLE l-estudante AS logical
VIEW-AS TOGGLE-BOX LABEL "Estudante?".

Tipos de Widgets - Fill-in


```
DEFINE VARIABLE c-name AS character
        VIEW-AS FILL-IN LABEL "Nome"
        FORMAT "x(30)".
```

Atributos
- Format
- Label
- Screen-value
- Sensitive
- Visible
- Tooltip

Métodos
- Load-mouse-pointer

Eventos
- Entry
- Leave

Tipos de Widgets - Toggle-box

```
DEFINE VAR l-cobertura1 AS logical LABEL "Chocolate"
                                VIEW-AS TOGGLE-BOX.

DEFINE VAR l-cobertura2 AS logical LABEL "Morango"
                                VIEW-AS TOGGLE-BOX.
```

Ao mudar o valor selecionado, o progress executa a trigger de value-changed

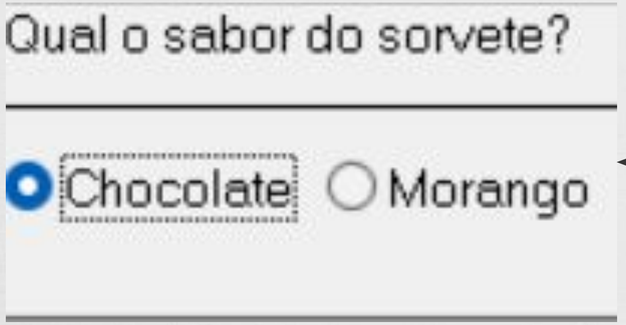
Atributos
- Label
- Screen-value
- Sensitive
- Visible
- Tooltip
- Checked

Métodos
- Load-mouse-pointer

Eventos
- Entry
- Leave
- Value-changed

Tipos de Widgets - Radio-set

```
DEF VAR i-sabor-sorvete  AS integer
        LABEL "Qual o sabor do sorvete?"
        VIEW-AS RADIO-SET HORIZONTAL
        RADIO-BUTTONS "Chocolate", 1 , "Morango", 2.
```



Ao mudar o valor selecionado, o progress executa a trigger de value-changed

Atributos
- Label
- Screen-value
- Sensitive
- Visible
- Tooltip
- Horizontal
- List-items

Métodos
- Load-mouse-pointer
- Add-last
- Delete
- Disable
- Enable
- Replace

Eventos
- Entry
- Leave
- Value-changed

Tipos de Widgets - Selection-list


```
DEF VAR c-coberturas  AS character
    LABEL "Cobertura"
    VIEW-AS SELECTION-LIST MULTIPLE
    LIST-ITEMS "Chantily",
               "Creme",
               "Mel",
               "Morango"
    SCROLLBAR-VERTICAL
    INNER-CHARS 12
    INNER-LINES 4 SORT.
```

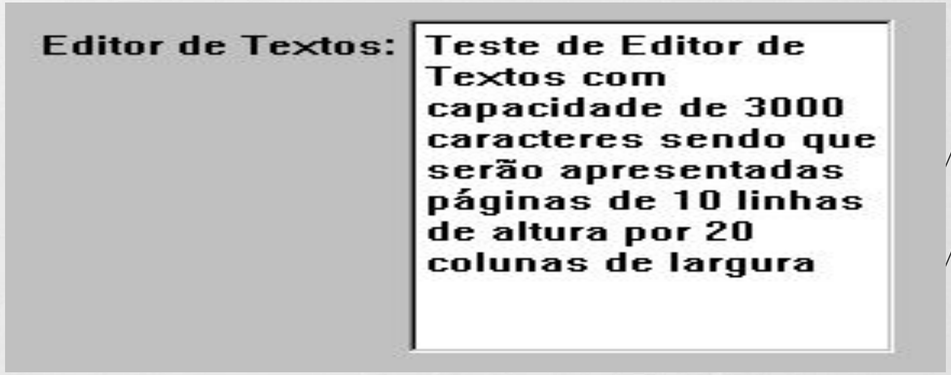
Atributos
- Label
- Screen-value
- Sensitive
- Visible
- Tooltip
- Multiple
- Num-items
- List-items

Métodos
- Load-mouse-pointer
- Add-last
- Add-first
- Delete
- Entry
- Replace
- Lookup

Eventos
- Entry
- Leave
- Value-changed
- Default-action

Tipos de Widgets - Editor


```
DEF VAR c-editor AS char FORMAT "x(3000)"
        LABEL "Editor de Textos"
        VIEW-AS EDITOR INNER-LINES 10
                       INNER-CHARS 20
                  SCROLLBAR-VERTICAL
                  SCROLLBAR-HORIZONTAL.
```



Atributos
- Label
- Screen-value
- Sensitive
- Visible
- Tooltip
- Num-lines
- Read-only
- Scrollbar-vertical
- Scrollbar-horizontal

Métodos
- Load-mouse-pointer
- Read-file
- Save-file

Eventos
- Entry
- Leave

Tipos de Widgets - Combo-box

```
DEF VAR c-coberturas AS character
    LABEL "Cobertura" VIEW-AS COMBO-BOX
    LIST-ITEMS "Chantily",
               "Creme",
               "Mel",
               "Morango" SIZE 15 BY 1.
```

Atributos
- Format
- Label
- Screen-value
- Sensitive
- Visible
- Tooltip
- List-items
- num-items

Métodos
- Load-mouse-pointer
- Add-first
- Add-last
- Delete
- Entry
- Lookup
- Replace

Eventos
- Entry
- Leave
- Value-changed

Tipos de Widgets - Image


```
DEF IMAGE im-logo  FILE "datasul.bmp" SIZE 10 BY 1.
```

Atributos
- Sensitive
- Visible
- Tooltip

Métodos
- Load-mouse-pointer
- Load-image

Tipos de Widgets - Rectangle


```
DEF RECTANGLE rt-key SIZE 70 BY 5
                     EDGE-PIXELS 3
                     BGCOLOR 3 FGCOLOR 15.
```

Atributos
- Filled
- Sensitive
- Visible
- Tooltip

Métodos
- Load-mouse-pointer

Tipos de Widgets - Browse

```
DEF QUERY q-cust FOR customer SCROLLING.
DEF BROWSE b-cust QUERY q-cust DISPLAY
        customer.CustNum    customer.Name
        customer.City       customer.Country
        WITH SEPARATORS 5 DOWN.
DEF FRAME f-dados b-cust WITH NO-LABELS.
OPEN QUERY q-cust FOR EACH customer.
UPDATE b-cust WITH FRAME f-dados.
```

Atributos
- Multiple
- Sensitive
- Visible
- Separators
- Title
- Num-selected-rows

Métodos
- Fetch-selected-row
- Is-row-selected
- Select-row
- Load-mouse-pointer

Eventos
- Entry
- Leave
- Default-action
- Value-changed

Representação de Campos

Widget	Tipo de dados		
	Caracter	Numérico	Lógico
Combo-Box	x	x	x
Fill-in	x	x	x
Radio-set	x	x	x
Text	x	x	x
Selection-list	x		
Editor	x		
Toggle-box			x

Principais Eventos

- CHOOSE (buttons, menus)
- ENTRY
- LEAVE
- VALUE-CHANGED
  (toggle-box, radio-set, selection-list, browse)

Manipulando Widgets

ENABLE - ativa o widget na tela.
Ex.: ENABLE bt-inclui WITH FRAME f-cliente.

DISABLE - desativa o widget na tela.
Ex.: DISABLE bt-mod WITH FRAME f-cliente.

APPLY - Executa um gatilho de um widget.
Ex.: APPLY "choose" TO bt-mod IN FRAME f-cliente.

WAIT-FOR - Para a execução. Fica esperando eventos.
Ex.: WAIT-FOR choose OF bt-sai IN FRAME f-cliente.

Relembrando - Atributos

widget-name:attribute-name
Ex.: DISPLAY c-name:**label** WITH FRAME f-x.

widget-name:attribute-name IN FRAME frame-name = value
Ex. ASSIGN c-name:**help** IN FRAME f-x = "Informe o nome do cliente".

Relembrando - Métodos

ASSIGN v-log = **widget-name:method-name(...)** IN FRAME frame-name.

Ex.: ASSIGN v-log = ed-texto:**load-file**("doc.txt") IN FRAME f-x.

ASSIGN v-log = sl-estados:**add-last**("SC") IN FRAME f-x.

Menus

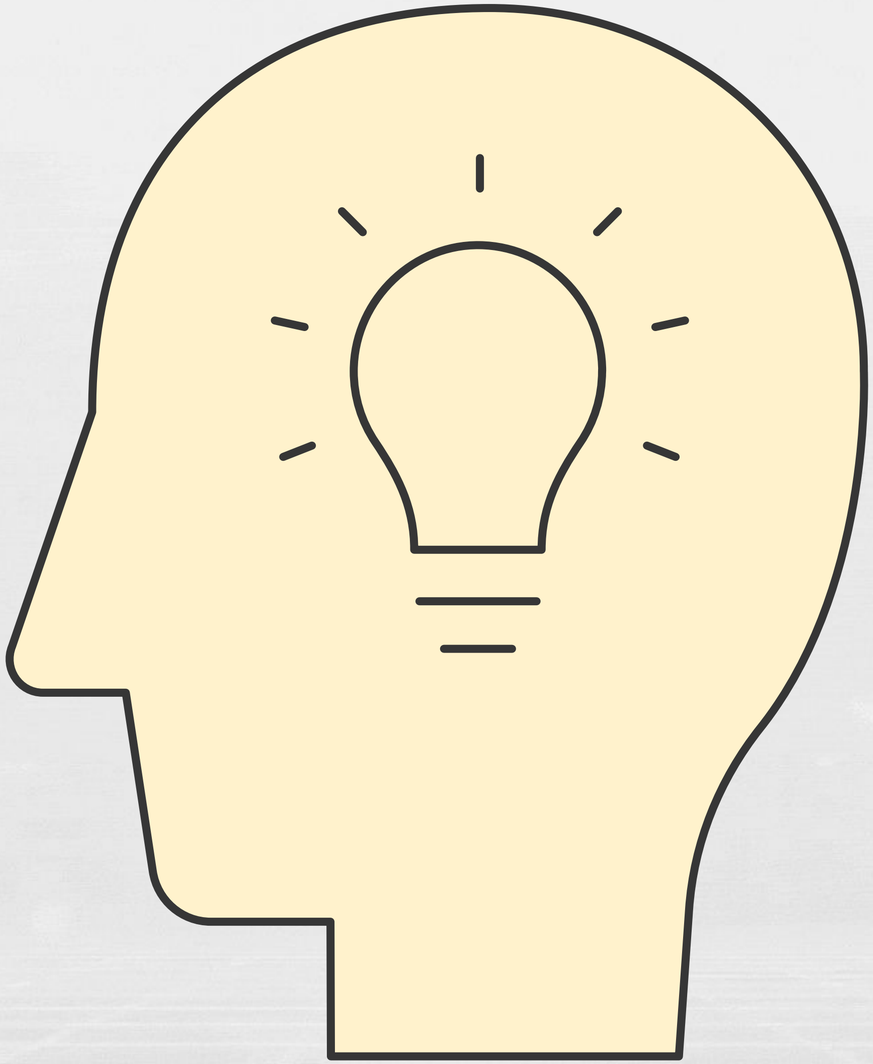
ASSIGN DEFAULT-WINDOW:MENUBAR = MENU mbar:HANDLE.

```
DEFINE SUB-MENU sm-Reports
    MENU-ITEM mi-Labels        LABEL "Customer Labels"
    MENU-ITEM mi-Names         LABEL "Customer Names"
    RULE
    MENU-ITEM mi-Balances      LABEL "Order Totals"
    MENU-ITEM mi-Today         LABEL "Order Items".

DEFINE MENU mbar MENUBAR
    SUB-MENU  sm-Table         LABEL "Tables"
    SUB-MENU  sm-Reports       LABEL "Reports"
    SUB-MENU  sm-Help          LABEL "Help".
```

Menus - Triggers


```
ON CHOOSE OF MENU-ITEM mi-item IN MENU menu-name DO:
      bloco
END.

DEF SUB-MENU sm-sair
     MENU-ITEM mi-sair  LABEL "Sai&r"
     TRIGGERS:
          ON CHOOSE APPLY "window-close" TO current-window.
     END TRIGGERS.
```

Laboratório 9



13 Tabelas Temporárias

Temp-Tables

- São iguais as tabelas
- Possuem índice
- Podem denegrir a performance, caso tenham muitos registros e não possuam índice adequado

```
DEF TEMP-TABLE tt-name   [NO-UNDO]
        [LIKE table-name
        [USE-INDEX index-name [AS PRIMARY]] ... ]
        [FIELD field-name  {AS data-type | LIKE field}] ...
        [INDEX index-name
           [IS [UNIQUE] [PRIMARY] [WORD-INDEX ]]
           {index-field [ASCENDING|DESCENDING]}...]...
```

Temp-Tables

Ex.:

```
DEF TEMP-TABLE tt-cliente NO-UNDO
        LIKE customer
        FIELD SalesRep AS character FORMAT "x(5)" LABEL "Representante"
        FIELD Region     LIKE state.Region
        INDEX i-repres IS PRIMARY SalesRep.

INSERT tt-cliente EXCEPT Comments
            WITH 1 COLUMN.

FOR EACH tt-cliente NO-LOCK:
        DISP tt-cliente EXCEPT Comments
                WITH SIDE-LABELS.
END.
```

Temp-Tables - Like Tabela

- Sempre herda todos os atributos da tabela.
- É possível incluir novos atributos.

Exemplo:

```
DEF TEMP-TABLE tt-cliente LIKE customer
    FIELD c-end-entrega AS char FORMAT "x(40)".
```

Temp-Tables - Triggers

- Temp-table like **NÃO** herda as triggers.
- Temp-table **NÃO** pode ter trigger.

Temp-Tables - Índices

- Pode ser definido quantos índices se desejar em uma temp-table.
- O Primário será aquele que tiver a cláusula PRIMARY, caso não seja definido, será o primeiro definido.
- Temp-Table Like
  - Herda todos os índices da tabela (*)
  - Caso seja definido um índice na temp-table todos os índices serão ignorados.
  - É possível definir qual índice da tabela deve ser utilizado.

Temp-Tables

- Definindo índice:

```
def temp-table tt-cliente no-undo
    field i-cod as int
    field c-nome as char
    index ind-cod is primary i-cod.
```

- Com todos os índices da tabela:

```
def temp-table tt-cliente no-undo like customer.
```

- Somente com um índice:

```
def temp-table tt-cliente no-undo like customer
    index ind-nome Name.
```

- Com um índice específico da tabela:

```
def temp-table tt-cliente no-undo like customer
    index ind-nome Name
    use-index i-custnum.
```

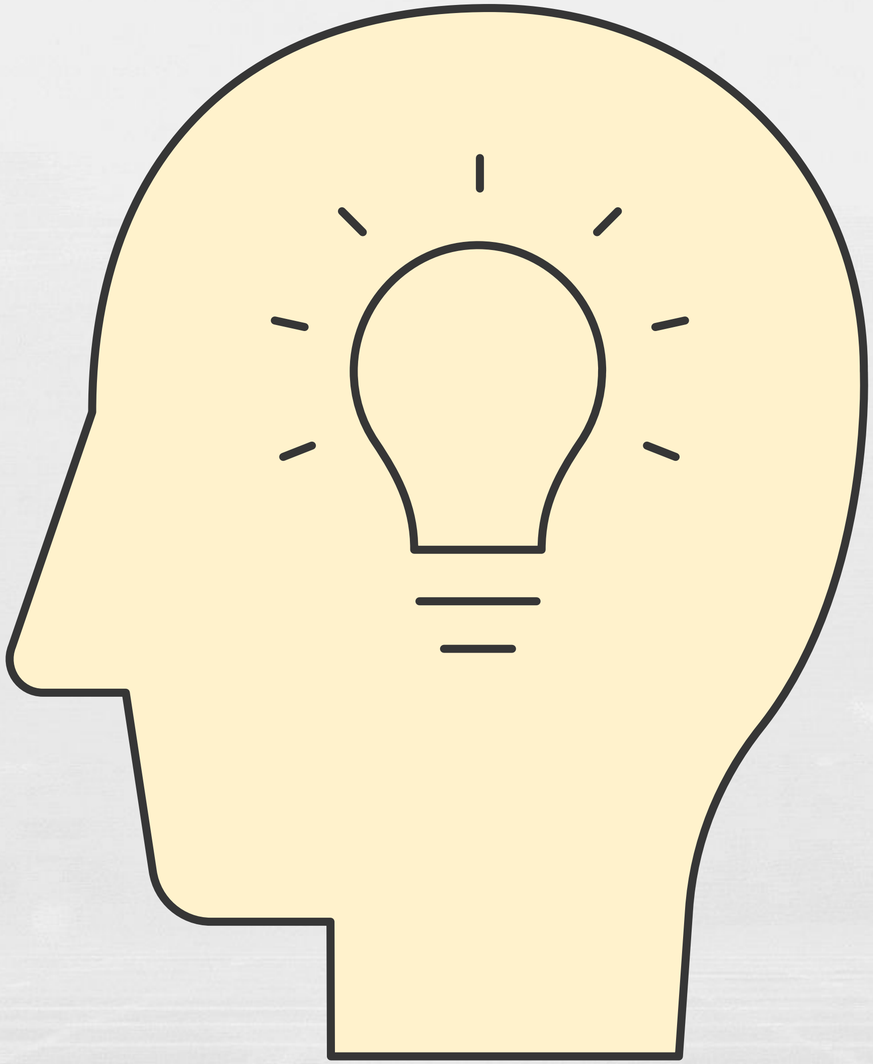
Lock

- As cláusulas NO-LOCK, EXCLUSIVE-LOCK e SHARED-LOCK não tem efeito sobre a leitura.

Transação

- Ocorre o mesmo que uma tabela.
- Não inicia transação.

Laboratório 10



14 Procedures

Procedure Externa: Programas PROGRESS (.p). Pode ser chamada de qualquer outro do programa, mas sem acessar o mesmo dado, e pode haver a necessidade de compartilhamento.

Procedure Interna: Bloco de código definido dentro de um programa PROGRESS. Pode ser chamada de qualquer ponto do programa. Acessam os mesmos dados, sem a necessidade de compartilhamento.

Procedures

Para executar uma procedure é utilizamos o comando RUN.

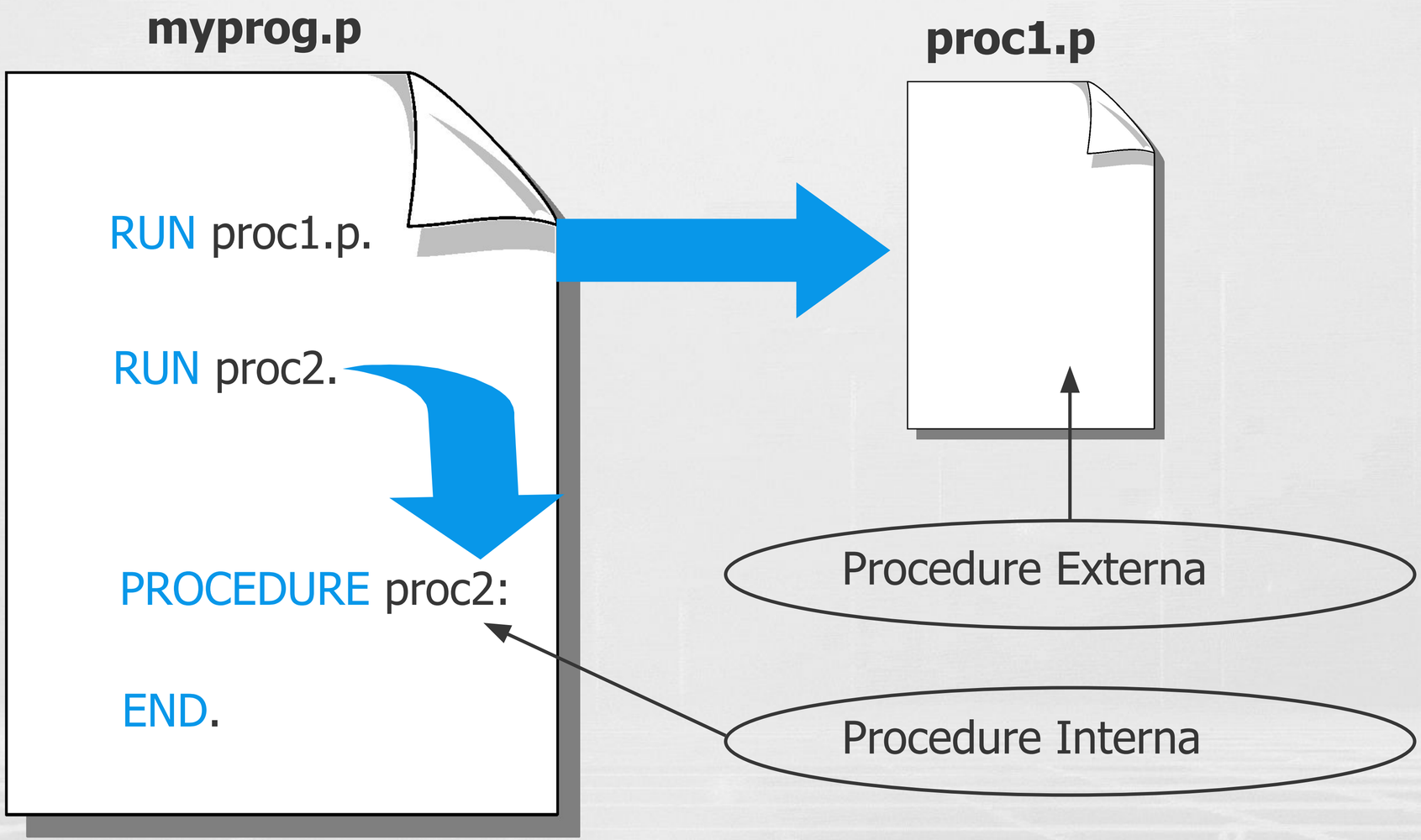
RUN <nome-da-procedure>.

Ex.:

RUN pi-calculo. *(procedure interna)*

RUN proces.p. *(procedure externa)*

Procedures


myprog.p
RUN proc1.p.
RUN proc2.
PROCEDURE proc2:
END.
proc1.p
Procedure Externa
Procedure Interna

Procedures - Definição de PI


```
PROCEDURE <nome-da-procedure>:
    <código fonte>
END PROCEDURE.

Ex.: FOR EACH customer:
         RUN pi-mostra-cliente.
     END.

     PROCEDURE pi-mostra-cliente:
         DISPLAY customer.custnum
                 customer.name.
     END PROCEDURE.
```

Procedures


```
FOR EACH customer:
    DISPLAY  customer.custnum
             customer.name.
    RUN pi-mostra-pedidos.
END.

PROCEDURE pi-mostra-pedidos:
    FOR EACH order OF customer:
        DISPLAY order.ordernum.
    END.
END.
```

Procedures - Retorno


```
FOR EACH customer:
        DISPLAY   customer.custnum
                  customer.name.
       RUN pi-mostra-pedidos.
       IF RETURN-VALUE = "NOK" THEN
          UNDO, NEXT.
END.

PROCEDURE pi-mostra-pedidos:
        DEF VAR lFound AS LOGICAL NO-UNDO.
        FOR EACH order OF customer:
                DISPLAY order.ordernum.
                ASSIGN lFound = TRUE.
        END.
        IF NOT lFound THEN
            RETURN "NOK".
        ELSE
           RETURN "OK".
END.
```

Procedures Persistentes

Programa carregado na memória, permitindo que seja executado suas procedures internas.

```
RUN  <nome-da-procedure>
        PERSISTENT SET <variável-handle>.
```

Ex.:

```
DEF VAR hproces AS handle NO-UNDO.

RUN proces.p PERSISTENT SET hproces.

RUN pi-digito IN hproces.
```

Procedures

COMPILE <nome-do-programa> [SAVE].

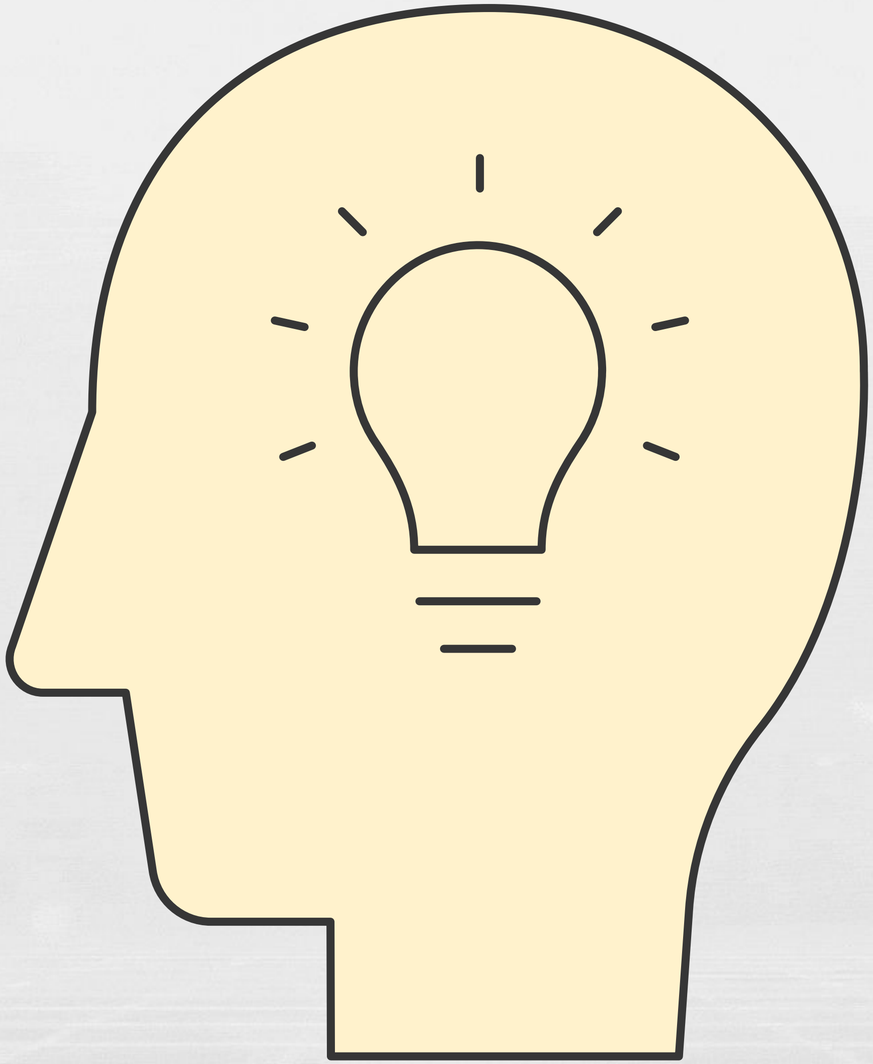
Ex.:

COMPILE proces.p SAVE.

Extensões:

.P
.W
.i
.PY
.R

Laboratório 11



15 Compartilhamento de Dados

Componentes Compartilhados

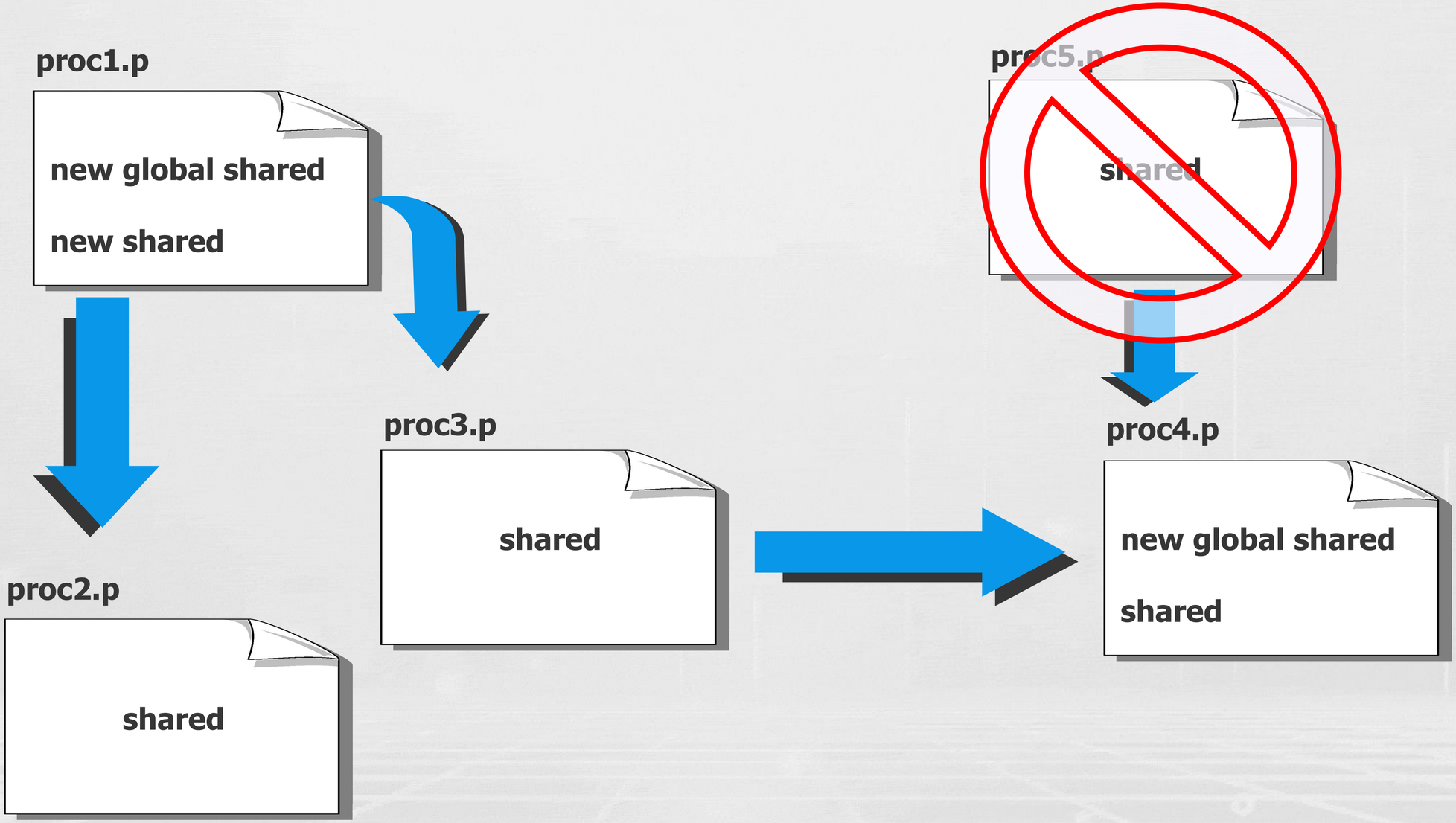
DEFINE [[NEW [GLOBAL]] SHARED] VARIABLE

DEFINE [[NEW] SHARED] FRAME

DEFINE [[NEW] SHARED] BUFFER

DEFINE [[NEW [GLOBAL]] SHARED] STREAM

Componentes Compartilhados



Passagem de Parâmetros

INPUT: recebe um valor do procedimento que o chamou.

Ex.: RUN prog2.p (INPUT c-name).

```
/* prog2.p */
DEF INPUT PARAM c-nome AS char NO-UNDO.
```

Passagem de Parâmetros

OUTPUT: retorna um valor para o procedimento que o chamou.

Ex.: RUN prog2.p (OUTPUT i-cont).

```
/* prog2.p */
DEF OUTPUT PARAM i-num AS integer NO-UNDO.
```

Passagem de Parâmetros

INPUT-OUTPUT: recebe e retorna um valor para o procedimento que o chamou.

Ex. RUN prog2.p (INPUT-OUTPUT c-name).

```
/* prog2.p */
DEF INPUT-OUTPUT PARAM c-nome AS char NO-UNDO.
```

Passagem de Parâmetros

PARAM BUFFER: passa o buffer do registro como parâmetro para o programa chamado.

Ex. RUN prog2.p (BUFFER customer).

```
/* prog2.p */
DEF PARAM BUFFER b-customer FOR customer.
```

Passagem de Parâmetros

PARAM TABLE: passa uma temp-table como parâmetro para o programa chamado.

Ex. RUN prog2.p (INPUT TABLE tt-dados).

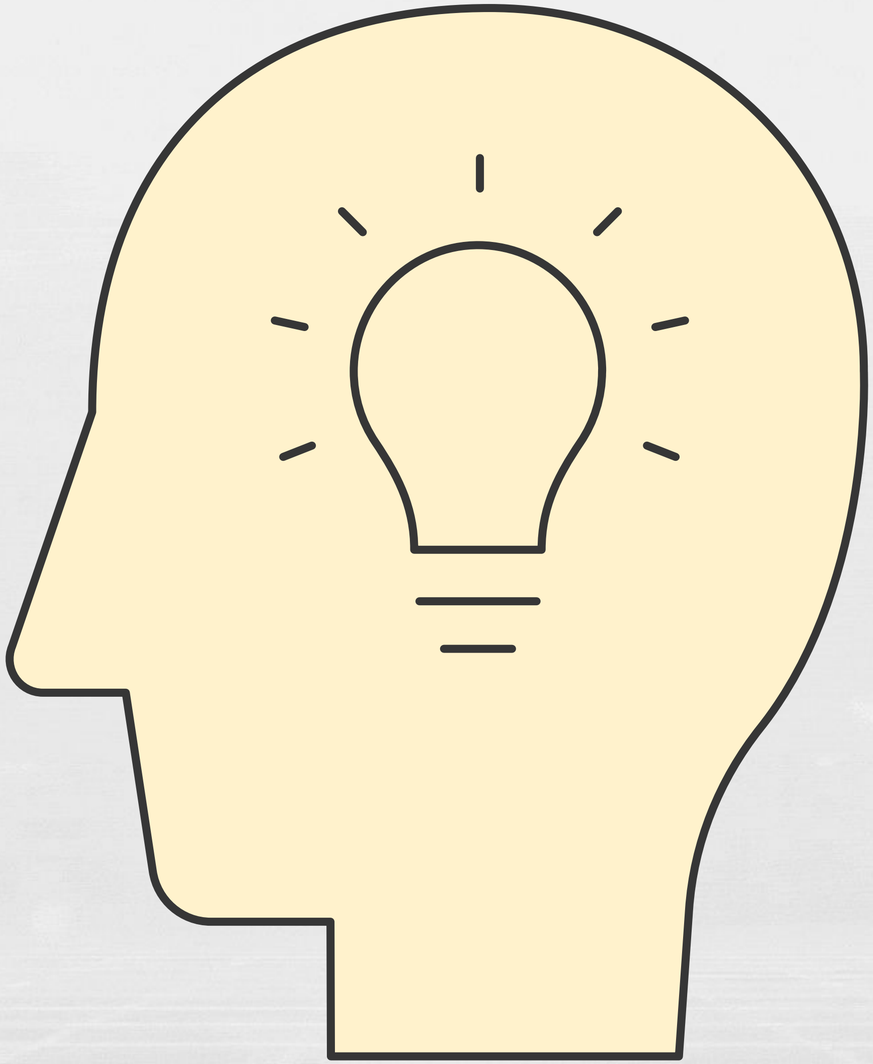
```
/* prog2.p */
DEF TEMP-TABLE tt-dados LIKE customer.

DEF INPUT PARAM TABLE FOR tt-dados.
        OUTPUT
        INPUT-OUTPUT
```

Tabelas Temporárias x Componentes Compartilhados

- Numa passagem de parâmetro ou uma definição Shared/Global a assinatura deve ser mantida.

Altera a assinatura ?	**Param**	**Shared/Global**
Nome da Temp-table	Não	Não
Nome do Campo	Não	Não
Tipo do Campo	Sim	Sim
Formato do Campo	Não	Não
Índice	Não	Sim
Cláusula undo	Não	Sim
Quantidade de Campo	Sim	Sim

Laboratório 12



16 Importação e Exportação

Export


```
OUTPUT TO c:/tmp/state.d.
FOR EACH state:
      EXPORT state.
END.
OUTPUT CLOSE.
```

```
"AK" "Alaska" "West"
"AL" "Alabama" "South"
"AR" "Arkansas" "Central"
"AZ" "Arizona" "West"
"CA" "California" "West"
"CO" "Colorado" "West"
```

Input From

INPUT [STREAM xxx] FROM arquivo.txt.

- Se o arquivo de entrada conter mais campos que o procedimento usa, PROGRESS ignora os campos excedentes.

- Se o arquivo de entrada conter menos campos que o esperado, PROGRESS processa a linha do arquivo, mas não seta os campos que ficaram sem entrada.

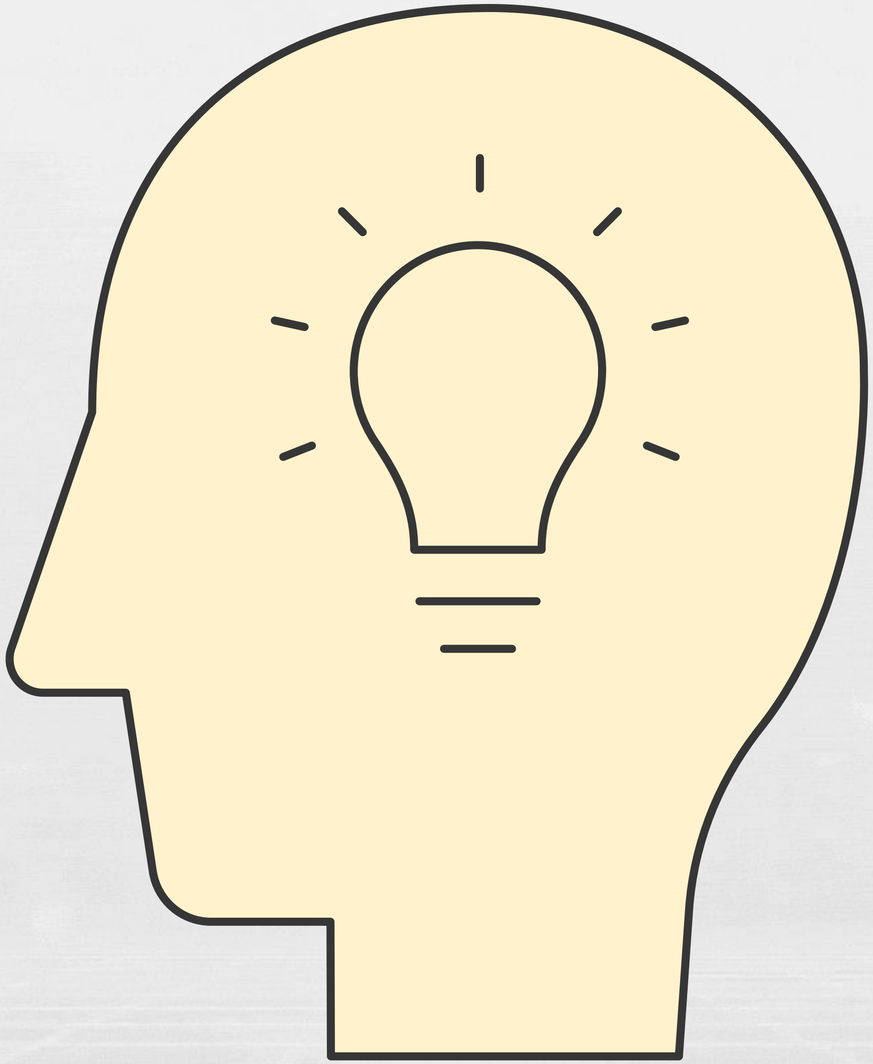
Input From


```
INPUT FROM cust.txt.
REPEAT:
        CREATE customer.
        SET customer.CustNum
            customer.Country
            customer.Name.
END.
INPUT CLOSE.
```

Import


```
DEF VAR i-cust    AS integer    NO-UNDO.
DEF VAR c-name    AS character  NO-UNDO.
DEF VAR c-rep     AS character  NO-UNDO.

INPUT FROM cust.txt.
REPEAT:
    IMPORT i-cust c-name c-rep.
    MESSAGE i-cust   SKIP
            c-name SKIP
            c-rep
            VIEW-AS ALERT-BOX.
END.
INPUT CLOSE.
```

Laboratório 13



17 Tratamento de Erros

Tipos de Erros

- **Erro de sistema**

- **Erro gerado pelo procedimento** - a busca de um registro falha ou o procedimento tenta criar um registro com chave duplicada.

- **Erro de interrupção de processamento** - quando o usuário pressiona as teclas de ERROR, ENDKEY ou END-ERROR.

Como Tratar Erros


```
ON   ERROR    UNDO <bloco>, LEAVE    <bloco>
ON   QUIT     UNDO <bloco>, RETURN "nok"
ON   STOP     UNDO <bloco>, RETRY    <bloco>
ON   ENDKEY   UNDO <bloco>, LEAVE    <bloco>
```

Como Tratar Erros


```
Main_block:
DO  ON ERROR UNDO Main_block, LEAVE Main_block:
      RUN  elimina.p.
END.

Processa:
FOR EACH customer EXCLUSIVE-LOCK
        ON ERROR UNDO, LEAVE:
        IF customer.CustNum > 15 THEN
             UNDO, NEXT.
        DELETE customer.
END.
```

18 Transações

Transações

- Unidade completa de trabalho
- A transação é um trabalho que se for terminada a sua execução de forma anormal, será completamente desfeito, ou seja, todo realizado ou nada feito.

Procedimento
REPEAT
FOR EACH

Blocos que tem processamento de transação

Transações

Uma transação é uma interação do bloco mais externo que contiver declarações que alteram diretamente o banco de dados, ou a leitura de registros com a opção EXCLUSIVE-LOCK.

Escopo de uma transação determina a quantidade de trabalho que PROGRESS desfaz em eventos de falha do sistema.

```
REPEAT:
        CREATE customer.
        UPDATE CustNum Name Phone.
END.

DO TRANSACTION:
        REPEAT:
            CREATE customer.
            UPDATE CustNum Name Phone.
        END.
END.
```

Controle de Transações

- Início e Término

- Pode ser controlado com a palavra-chave **TRANSACTION**

  DO TRANSACTION:

  FOR EACH tabela TRANSACTION:

  REPEAT TRANSACTION:

Aumentando o Tamanho da Transação


```
do  transaction:
        repeat:
            insert order with 2 columns.
            repeat:
                create orderline.
                orderline.OrderNum = order.OrderNum.
                update LineNum ItemNum Qty Price.
            end.
        end.
end.
```

Diminuindo o Tamanho da Transação


```
repeat:
    do transaction:
        insert order with 2 columns.
    end.
    repeat transaction:
        create orderline.
        orderline.OrderNum = order.OrderNum.
        update LineNum ItemNum Qty Price.
    end.
end.
```

Transações - Subtransações

- Se já tem transação ativa e o Progress encontra um bloco de transação é iniciada uma subtransação.
- Implica em gravações no arquivo **.lbi.**

Transações - Subtransações


```
repeat:
    insert order with 2 columns.
    repeat:
        create orderline.
        orderline.OrderNum = order.OrderNum.
        update LineNum ItemNum Qty Price.
    end.
end.
```

Transações

- Se a transação iniciar no trigger/pi ela terminará dentro do mesmo.
- Se ao iniciar um trigger/pi e tiver uma transação ativa, o trigger/pi fará parte da mesma.

Transações - Programas

- Programas e subprogramas estão sujeitos a transação
- Avaliar a transação em tempo de execução

```
repeat:
        find next customer.
        update customer with 2 columns.
        run proc1.p.
end.
```

Transações - Variáveis

- Variáveis estão sujeitas a transação.
- Devem ser definidas sempre com **NO-UNDO**.

Transações - Multi-Bancos

- Uma transação envolve mais de um banco conectado.
- O Progress usa o protocolo TWO-PHASE COMMIT
  - Na primeira fase verifica se todos os bancos estão conectados.
  - Na segunda fase confirma alterações.

- Para determinar se uma transação está ativa pode ser usada a função TRANSACTION.
  - Retorna verdadeiro quando a transação está ativa naquele ponto.

Transações - Função Transaction


```
message "Fora do FOR EACH: " TRANSACTION.

for each customer:
    message "Dentro do FOR EACH:" TRANSACTION.
    update customer with 2 columns.
end.
```

Transações

- Usar NO-UNDO nas variáveis.
- Ao processar campos com extent usar o DO ao invés do REPEAT. O REPEAT inicia transação ou subtransação para cada elemento.
- Quando a lógica do programa permitir, fazer as atualizações mais no nível de transação do que no nível de subtransação.

Transações - WARNING--TRANSACTION

**** WARNING--TRANSACTION keyword given within actual transaction level. (214)**

- Acontece quando é usada a palavra chave TRANSACTION e o programa já é uma transação.
- Corrigir. Pois ao entrar no programa não deveria ter transação ativa.

Transações - WARNING--TRANSACTION


```
do transaction:
        //  atualização do log de programa
end.

/* alguma atualização */

repeat: // ou for each
        // lógica do programa
end.
```

Transações - WARNING--TRANSACTION


```
do transaction:
        //  atualização do log de programa
end.

do transaction:
        /* alguma atualização */
end.

repeat: // ou for each
        // lógica do programa
end.
```

Transações - Gatilhos de Dicionário

- É sempre transação
  - Create
  - Delete
  - Write
  - Assign
- Pode ser ou não
  - Find

19 Bloqueio de Registros

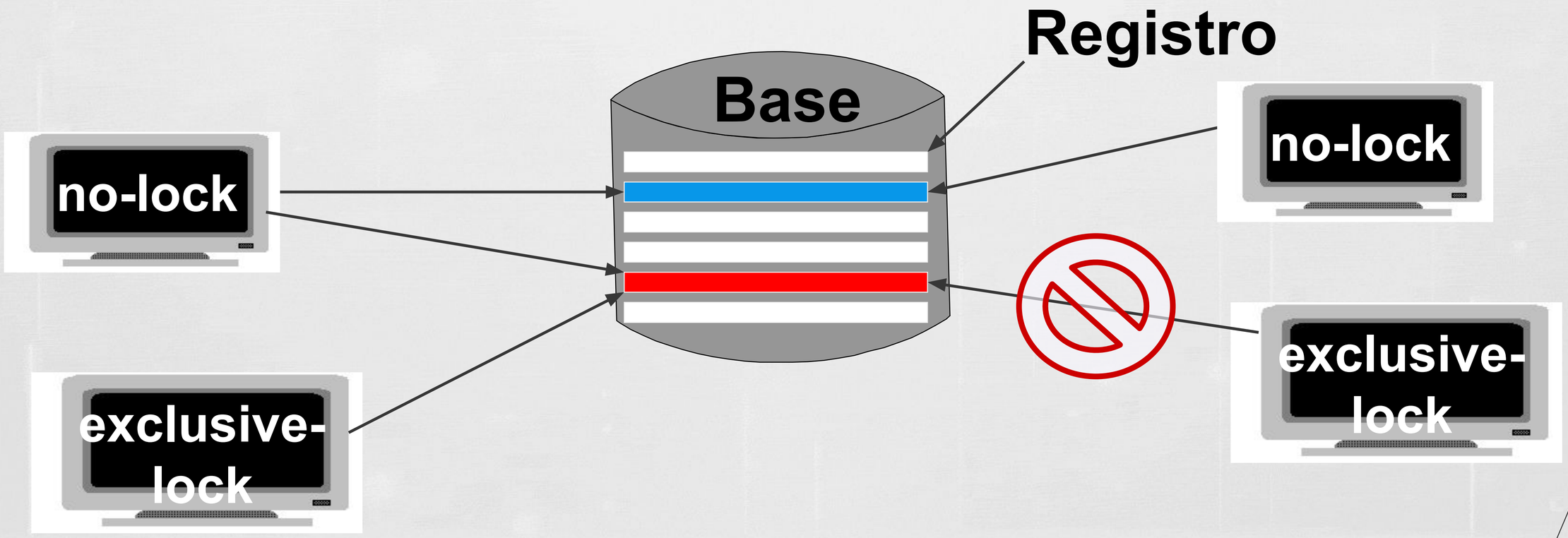
Bloqueio de Registros

NO-LOCK – Somente leitura não bloqueia o registro.

SHARE-LOCK (não aconselhável) – O registro fica livre até que haja uma interação na tabela, neste caso o progress passa o registro para bloqueado. Caso não seja informado o tipo de Lock o progress assume share-lock.

EXCLUSIVE-LOCK – Bloqueia o registro com o usuário corrente.

Bloqueio de Registros



Bloqueio de Registros

- Quando o Progress lê o registro ele usa **SHARE-LOCK** por default.
- Quando o Progress altera o registro ele coloca o registro em **EXCLUSIVE-LOCK**.

```
for each customer:
    update
    customer.
end.
```

Bloqueio de Registros - Como Usar

- Quando ler o registro e não for alterar:
  usar **NO-LOCK**.

- Quando lê o registro para alteração:
  usar **EXCLUSIVE-LOCK**.

- Evitar o uso do SHARE-LOCK.
  Pode causar o DEAD-LOCK.
  Dois usuários tentam alterar o mesmo registro. Um bloqueia o outro.

Bloqueio de Registros - Duração

- A duração do bloqueio é determinada pela transação e pelo escopo do registro.

- **SHARE-LOCK**
  - Liberado no fim da transação ou no fim do escopo do registro. O que for maior.

- **EXCLUSIVE-LOCK**
  - Liberado no fim da transação. Se o escopo do registro for maior é convertido para SHARE-LOCK.

Bloqueio de Registros - Release do Registro

- O Progress libera o registro (release) quando:
  - termina o escopo do registro.
    - é executado o comando RELEASE.
    - quando é executado um CREATE, FIND, FOR EACH.

Bloqueio de Registros - Triggers


```
def button bt-mod label "Modifica".

form
    item.ItemNum item.ItemName bt-mod
    with frame a.

on choose of bt-mod do:
    find first item exclusive-lock.
    update item.ItemName with frame a.
end.

enable all with frame a.
find first item.
display item.ItemNum item.ItemName
        with frame a.
wait-for window-close of current-window.
```

Bloqueio de Registros


```
def button bt-mod label "Modifica".

form
    item.ItemNum item.ItemName bt-mod
    with frame a.

on choose of bt-mod do:
    find first item exclusive-lock.
    update item.ItemName with frame a.
    release item. // libera share-lock do registro
end.

enable all with frame a.
find first item.
display item.ItemNum item.ItemName
        with frame a.
wait-for window-close of current-window.
```

Bloqueio de Registros - Exclusive-lock

- Só é liberado no final da transação.
- Não adianta executar RELEASE (o release libera o SHARE-LOCK se o escopo for maior que a transação).
- Para resolver: diminuir o tamanho da transação.

Bloqueio de Registros - Exclusive-lock

- Transação é todo o programa.
- O bloqueio de todos os registros somente são liberados no fim do programa.

```
find first empresa exclusive-lock.
empresa.dt-calculo = today.

for each conta-contabil exclusive-lock:
  val-saldo = ....
end.
```

Bloqueio de Registros - Exclusive-lock

SYSTEM ERROR: Lock Table Overflow !!!

Bloqueio de Registros - Reduzir o Tamanho da Transação

```
do transaction:
    find first empresa exclusive.
    empresa.dt-calculo = today.
end.

for each conta-contabil exclusive:
    val-saldo = ....
end.
```

Bloqueio de Registros - Lock Table

- Tabela de bloqueio de registros.
- Parâmetro do Servidor: **-L**
- Limitado pelo tamanho da memória ou parâmetros do kernell.

Bloqueio de Registros - Lock Table Overflow

- Soluções paliativas:
  - aumentar o parâmetro -L.
  - executar em mono-usuário.
    - Derrubar o servidor.
- Solução definitiva:
  - verificar o tamanho da transação no programa.
  - rever lógica do programa e da aplicação.

Bloqueio de Registros

- Evitar transações muito grandes.
- No projeto do sistema isto deve ser avaliado
- Exemplo:
  - Atualização da Contabilidade (débito e crédito)
  - Cálculo do preço médio

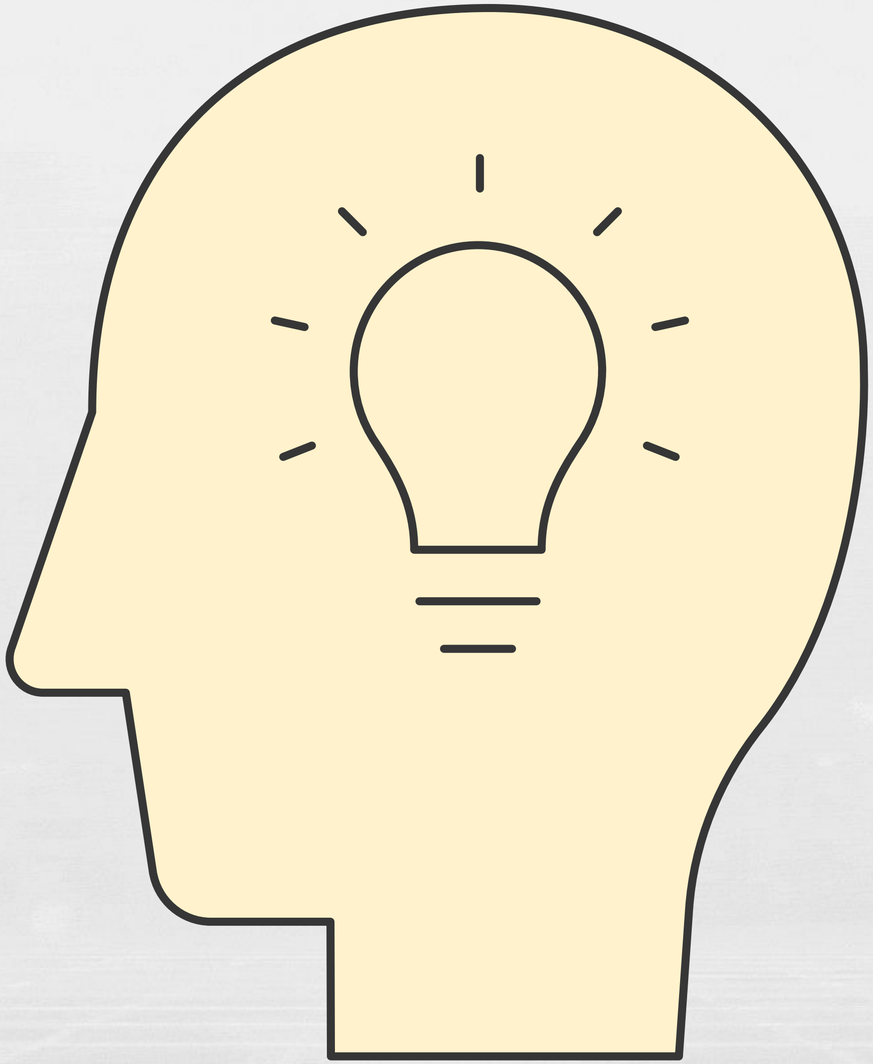
Bloqueio de Registros - Dead-lock

Dois usuários tentam alterar o mesmo registro, um bloqueia o outro. Isto ocorre pois eles acessam a mesma tabela com SHARED-LOCK.

Ambos querem atualizar a tabela e estão esperando que o Progress atualize o registro para EXCLUSIVE-LOCK.O Progress não vai ceder o EXCLUSIVE-LOCK para nenhum usuário, pois existe outro com o registro em SHARED-LOCK.

É necessário que um deles cancele a alteração.

Laboratório 14



20 Classes Úteis

Como Utilizar uma Classe

Para utilizarmos uma classe no Progress, temos que seguir os seguintes passos:

- Adicionar no começo do programa um USING com a classe
- Definir uma variável com o tipo o nome da classe
- Criar a instância da classe com o NEW classe()
- Acessar os atributos e métodos da classe

```
USING PROGRESS.json.ObjectModel.JsonObject.

DEF VAR oObj AS JsonObject NO-UNDO.

oObj = NEW JsonObject().
oObj:add('nome', 'Joao da Silva').
oObj:add('cargo', 'Gerente').
oObj:add('area', 'Engenharia').
```

JsonObject

Para representar um objeto JSON, a Progress fornece a classe Progress.json.ObjectModel.JsonObject.

Neste objeto é permitido adicionar, alterar e remover TAGS e seus valores.

Para utilizarmos essa classe, obrigatoriamente temos que definir no início do programa uma cláusula.

Ex.:

```
{
   "nome": "Joao da Silva",
   "cargo": "Gerente",
   "area": "Engenharia"
}
```

```
USING PROGRESS.json.ObjectModel.JsonObject.

DEF VAR oObj AS JsonObject NO-UNDO.

oObj = NEW JsonObject().
oObj:add('nome', 'Joao da Silva').
oObj:add('cargo', 'Gerente').
oObj:add('area', 'Engenharia').
```

JSonArray

Para representar uma lista de objeto, a Progress fornece a classe Progress.json.ObjectModel.JsonArray.

Neste objeto é permitido adicionar, obter e remover objetos de dentro da lista.
Para utilizarmos essa classe, obrigatoriamente temos que definir no início do programa uma cláusula.

Ex.:

```
[
  {
     "nome": "Joao da Silva",
     "cargo": "Gerente",
     "area": "Engenharia"
  },
  {
     "nome": "Gertrudes",
     "cargo": "Consultora",
     "area": "Implantacao"
  }
]
```

```
USING PROGRESS.json.ObjectModel.JsonObject.
USING PROGRESS.json.ObjectModel.JsonArray.

DEFINE VARIABLE oObj  AS JsonObject NO-UNDO.
DEFINE VARIABLE aList AS JsonArray  NO-UNDO.

aList = NEW JsonArray().
oObj = NEW JsonObject().
oObj:add('nome', 'Joao da Silva').
oObj:add('cargo', 'Gerente').
oObj:add('area', 'Engenharia').
aList:add(oObj).

oObj = NEW JsonObject().
oObj:add('nome', 'Gertrudes').
oObj:add('cargo', Consultora').
oObj:add('area', Implantacao').
aList:add(oObj).
```

Convertendo um JsonObject ou JsonArray para LongChar

Para converter um JsonObject ou um JsonArray em uma string, temos que utilizar o método **getJsonText()** dessas classes, onde será retornado um LongChar com o conteúdo.

Ex.:

```
USING PROGRESS.json.ObjectModel.JsonObject.
USING PROGRESS.json.ObjectModel.JsonArray.

DEFINE VARIABLE oObj  AS JsonObject NO-UNDO.
DEFINE VARIABLE aList AS JsonArray  NO-UNDO.
DEFINE VARIABLE lcTxt AS LONGCHAR   NO-UNDO.

FIX-CODEPAGE(lcTxt) = "utf-8":U.

aList = NEW JsonArray().
oObj = NEW JSonObject().
oObj:add('nome', 'Joao da Silva').
oObj:add('cargo', 'Gerente').
oObj:add('area', 'Engenharia').
aList:add(oObj).

lcTxt = aList:getJsonText().
```

```
DEF VAR cTmp AS CHARACTER NO-UNDO.
cTmp = STRING(lcTxt).
MESSAGE cTmp VIEW-AS ALERT-BOX.
```

```
[
  {
    "nome": "Joao da Silva",
    "cargo": "Gerente",
    "area": "Engenharia"
  }
]
```

Convertendo um LongChar para JsonObject ou JsonArray

Para converter um JsonObject ou um JsonArray em uma string, temos que utilizar o método **getJsonText()** dessas classes, onde será retornado um LongChar com o conteúdo.

Ex.:

```
USING PROGRESS.json.ObjectModel.JsonObject.
USING PROGRESS.json.ObjectModel.JsonArray.
USING Progress.Json.ObjectModel.ObjectModelParser.

DEFINE VARIABLE oObj     AS JsonObject        NO-UNDO.
DEFINE VARIABLE aList    AS JsonArray         NO-UNDO.
DEFINE VARIABLE lcTxt    AS LONGCHAR          NO-UNDO.
DEFINE VARIABLE myParser AS ObjectModelParser NO-UNDO.

FIX-CODEPAGE(lcTxt) = "utf-8":U.
lcTxt = '~{ "nome": "Joao da Silva", "cargo": "Gerente"}'.
myParser = NEW ObjectModelParser().
oObj = CAST(myParser:Parse(lcTxt), JsonObject) NO-ERROR.
delete object myParser.
```

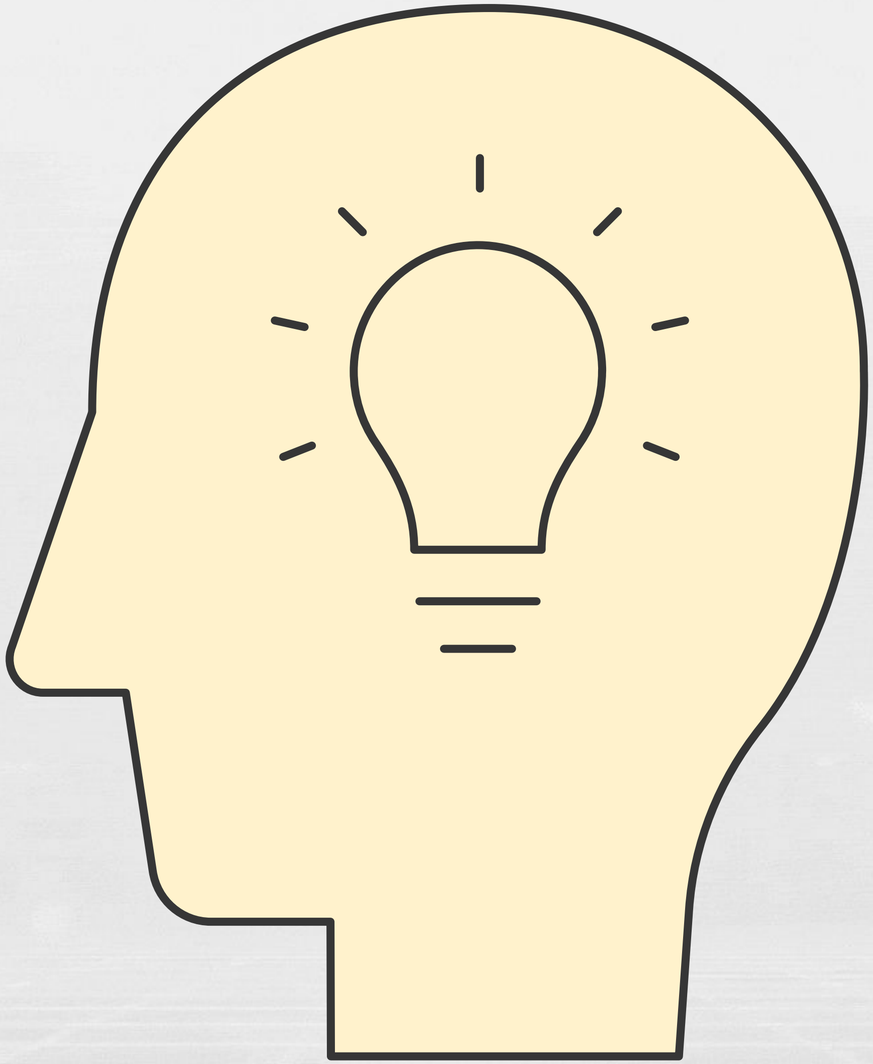
Gerando um Arquivo JSON

A forma mais simples para gerar um arquivo JSON com o conteúdo de uma tabela é utilizando uma Temp-Table.

Ex.:

```
DEFINE TEMP-TABLE ttCustomer NO-UNDO LIKE Customer.
FOR EACH Customer NO-LOCK:
    CREATE ttCustomer.
    BUFFER-COPY Customer TO ttCustomer.
END.
TEMP-TABLE ttCustomer:WRITE-JSON("FILE",
    "c:/tmp/customer.json", TRUE, ?, ?, TRUE).
EMPTY TEMP-TABLE ttClientes.
```

Laboratório 15



21 Funções

CAN-FIND: retorna verdadeiro se um registro foi encontrado.

Ex.: IF **CAN-FIND**(customer
WHERE customer.CustNum = i-cust-num) THEN ...

LOOKUP: retorna um valor inteiro indicando a posição da string dentro de uma lista. Separador default: ",".
LOOKUP(string, lista, [separador]).

Ex.: **LOOKUP**("sc", "am,pr,sc")

ENTRY: retorna uma string caracter baseada em uma lista, a partir de uma posição inteira. Separador default: ",".
ENTRY(posição, lista, [separador]).

Ex.: **ENTRY**(2, c-lista, "/").

NUM-ENTRIES: retorna o número de itens de uma lista.
Separador default: ",".
NUM-ENTRIES(lista, [separador])

Ex.: **NUM-ENTRIES**(c-lista, ".").

SUBSTRING: extrai uma porção de caracteres de uma string.
SUBSTR(string, posição-inicial, posição-final)

Ex.: **SUBSTR**(c-nome, 2, 10).
SUBSTR(c-address, 10).

STRING: converte dados não caracter para o tipo de dado caracter.

STRING(source, [format]).

Ex.: **STRING** (customer.cust-num).

STRING (customer.postal-code, "99999").

LENGTH: retorna o tamanho de uma string.

Ex.: ASSIGN i-tam = **LENGTH**(v_cod_estab).

SUBSTITUTE: troca parâmetros dentro de uma string.

SUBSTITUTE(string, param1, param2, ..., param*N*).

Ex.: v-desc = **SUBSTITUTE**("o usuário &1 não foi encontrado", c-user).

TRIM: retira os espaços em branco do início e do final da string.

Ex.: ASSIGN c-user = **TRIM**(" teste ").

REPLACE: troca *N* ocorrências da string1 pela string2 dentro de um texto.
REPLACE(texto, string1, string2).

Ex.: v-desc = "o usuário não foi encontrado".
v-desc = **REPLACE**(v-desc, "o", "ó").

INDEX: retorna um inteiro, indicando a posição de uma expressão dentro de uma string.

INDEX(string, expressão, posição-inicial p/procura).

Ex.: **INDEX**("curso de progress", "de").
INDEX("curso de progress", "pr", 10).

RECID: endereço físico de um registro.
RECID(tabela)

Ex.: FIND customer
WHERE RECID(customer) = 3848 NO-ERROR.

ROWID: endereço físico de um registro.
ROWID(tabela)

Ex.: ASSIGN r-reg = ROWID(customer).
FIND customer
WHERE ROWID(customer) = r-reg
EXCLUSIVE-LOCK NO-ERROR.

Dicas

Dicas - Habilitando o Debug

Acessar o MS-DOS como administrador
Tecla Windows + R
CMD

Acessar o diretório do OpenEdge
cd c:\Progress\OpenEdge\bin

Executar:
prodebugenable.bat -enable-all

Administrador: Prompt de Comando

```
Microsoft Windows [versão 10.0.26100.4061]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Windows\System32>cd C:\progress\openedge\bin

C:\Progress\OpenEdge\bin>prodebugenable.bat -enable-all
OpenEdge Release 12.8 as of Thu Dec 21 10:17:00 EST 2023

==============================================================================
        PROGRESS Debug Enabler
==============================================================================

Debugging is enabled for the Progress 4GL installed in
C:\Progress\OpenEdge.
C:\Progress\OpenEdge\bin>
```

Dicas

Utilize o HELP do progress, pois ele possui vários exemplos de utilização.

Documentação OpenEdge 12.8
https://docs.progress.com/pt-BR/category/openedge?labelkey=product_openedge_128

KBase Progress
https://community.progress.com/s/global-search/%40uri#t=KnowledgeBase&sort=relevancy&f:@commonproductgroup=[OpenEdge]

OBRIGADO

Ricardo Menna

Arquiteto – Engenharia On-Premise
ricardo.menna@totvs.com.br

totvs.com

@totvs

@totvs

/totvs

/totvs

company/totvs

totvs.store