



Ferramentas – Regras LSP - Básico

**SENIOR**  
Universidade  
Corporativa

Todos os direitos reservados e protegidos pela Lei nº 9.610, de 19/02/1998.  
Nenhuma parte deste material, sem a prévia e expressa autorização da Senior, poderá ser reproduzida ou transmitida sejam quais forem os meios empregados: eletrônicos, mecânicos, fotográficos, gravação ou quaisquer outros, sob pena de constituir violação aos direitos autorais”.

1ª Edição deste material – 10 de Outubro de 2016 - Versão do Sistema 6.2.31.

## Apresentação

---

Este material tem como objetivo apresentar os conhecimentos básicos para o desenvolvimento de regras na Linguagem Senior de Programação (LSP).

## Sumário

BOAS PRÁTICAS PARA O DESENVOLVIMENTO DE REGRAS .....	7
Uso de Regras: Quando? .....	7
Desenhar e Depois Desenvolver .....	7
Comentários em Regras .....	8
Indentação .....	8
Informações Fixas.....	9
EDITOR DE REGRAS .....	13
O Editor de Regras.....	13
Criando uma Nova Regra .....	14
Compilando uma Regra .....	14
Salvando uma Regra .....	15
Abrindo uma Regra Existente.....	15
Menus do Editor de Regras .....	16
Modo de Seleção .....	17
Macros.....	18
Indentar .....	19
Barra de Ferramentas .....	21
Marcadores .....	21
COMANDOS,VARIÁVEIS E FUNÇÕES. ....	26
Comandos.....	26
Operadores.....	27
Comando Se .....	28
Laço de repetição Para .....	30
Laço de repetição Enquanto .....	31
Definição de variáveis .....	32
Conversão de variáveis.....	34
Conversão de máscaras.....	35
Entrada de Valor.....	36
Cancel .....	36
Funções .....	36
Variáveis Disponibilizadas pelos Sistemas .....	39

Constantes.....	40
Tabelas e Campos.....	40
Depuração de Regras .....	41
ESTRUTURA BÁSICA DE CURSOR .....	46
Estrutura Básica.....	51
IDENTIFICADORES DE REGRAS .....	55
IDENTIFICADORES DE REGRAS - CADASTRAMENTO E ATIVAÇÃO .....	55
IDENTIFICADORES DE REGRAS - VARIÁVEIS DISPONÍVEIS.....	56

# CAPÍTULO 01

## Boas Práticas de Desenvolvimento

### Objetivos Específicos

- Informar as melhores práticas para o desenvolvimento de regras com a linguagem LSP

## BOAS PRÁTICAS PARA O DESENVOLVIMENTO DE REGRAS

Fazendo uma analogia, é possível dizer que desenvolver é como construir uma casa: muitos podem saber levantar paredes (os chamados “pedreiros de fim de semana”), mas somente bons profissionais a farão reta, dentro do esquadro, seguindo exatamente a planta e com a certeza de que não vai cair.

A adoção de más práticas no desenvolvimento de Regras não significa que não funcionem. Funcionam sim, mas as perguntas são: Funcionam até quando? E o que dizer da manutenção? Se voltarmos a abrir o código-fonte daqui a 2 anos, vamos nos entender nele?

Abaixo seguem alguns princípios de boas práticas na construção de Regras. É muito importante fazer bom uso disso pois bons profissionais adotam métodos de excelência.

### Uso de Regras: Quando?

Os sistemas da Senior são soluções completas para o mercado. Os recursos usuais de uma empresa já são contemplados nativamente.

Regras são úteis sim, mas elas só devem ser utilizadas como última opção, caso uma análise indique que a rotina padrão não atende as necessidades do processo. Muitas vezes a consultoria precisa ser muito mais de processo do que de desenvolvimento.

Portanto, sempre é necessário se perguntar: será que não seria o caso de a empresa adequar seus processos aos recursos nativos do sistema, ao invés de querer personalizá-lo?

Talvez, após as análises, a resposta seja negativa. Neste caso é necessário partir para a customização e é para esta finalidade que a Senior desenvolveu a ferramenta gerador de regras para este objetivo.

### Desenhar e Depois Desenvolver

Existem programadores que são afoitos em colocar no Editor de Regras os seus conhecimentos de programação. Porém, antes de digitar sequer uma linha de código-fonte, é prudente desenhar o processo que irá construir e validar com o cliente, tantas vezes quanto forem necessárias até haver consenso e aprovação do que será desenvolvido.

Não é o caso de discutir com o cliente o nome das tabelas que serão criadas, se serão utilizados dois ou três cursores no projeto, ou a função “xyz”. O que é necessário validar é se a análise da solução, uma vez colocada em produção (desenvolvida), irá atender ao processo do cliente. Se houver retrabalho na fase de desenho, este é sempre mais “barato” do que na fase de execução. E isto é válido independente do tamanho do projeto e da customização.

## Comentários em Regras

Pontuar suas Regras com comentários irá facilitar em muito as futuras manutenções. Pode até ser uma Regra que você mesmo fez e a domina com clareza, tenha certeza de que daqui 2, 3 anos terá dificuldades em fazer a manutenção se ela não estiver bem documentada. E se a manutenção da Regra for executada por um terceiro, então, haverá muitas dificuldades.

Para comentários de uma linha, utilize o símbolo @ no início e no final da frase.

Exemplo: **Definir Cursor** Cur\_Clientes; @Criação do cursor@

Para comentários de várias linhas, utilize os símbolos /\* no início e \*/ no final da frase.

Exemplo: **Definir Cursor** Cur\_Clientes; /\*Criação do cursor que será  
chamado na rotina de recalculo.\*/

## Indentação

Assim como os parágrafos são de fundamental importância para compreensão das idéias de um livro, a indentação em uma Regra auxilia seu entendimento. Ela não é obrigatória, assim como os comentários também não o são, mas auxiliam em muito. Procure usar o recurso de Tabulação para fazer a indentação.

Exemplo de Regra sem comentário e sem indentação:

```
Definir Alfa aTipoCliente;
Definir Alfa aNomeCliente;
Definir Alfa aEnderecoCliente;
Definir Alfa aMsg;
aTipoCliente = "J";
Definir Cursor Cur_Clientes;
Cur_Clientes.sql "SELECT CodCli, NomCli, EndCli FROM E085CLI WHERE
(TipCli =:aTipoCliente) ORDER BY CodCli";
Cur_Clientes.AbrirCursor();
ENQUANTO (Cur_Clientes.Achou)
Inicio
aNomeCliente = Cur_Clientes.NomCli;
aEnderecoCliente = Cur_Clientes.EndCli;
aMsg = aNomeCliente + " END.: "+ aEnderecoCliente;
Mensagem (Retorna, aMsg);
Cur_Clientes.Proximo();
Fim;
Cur_Clientes.FecharCursor();
```

Veja no exemplo abaixo a mesma Regra, agora com indentação e comentários. Qual é melhor para dar manutenção?

A mesma Regra, agora com comentários e com indentação:



```

@=====@
@ Cliente: Senior Sistemas @
@ Rotina: Exibir Nome e Endereço do Cliente em tela @
@ Código da Regra: 12 @
@ Desenvolvedor: George Silva @
@ Data criação: 26/09/2009 @
@ Data última alteração: 01/10/2009 @
@ Histórico alteração: Alterado o Tipo de Cliente @
@=====@
/*Definição das variáveis*/
Definir Alfa aTipoCliente;
Definir Alfa aNomeCliente;
Definir Alfa aEnderecoCliente;
Definir Alfa aMsg;
aTipoCliente = "J"; @Somente carregaremos Clientes do tipo Pessoa
Jurídica.@

/*Criação do Cursor de Clientes*/
Definir Cursor Cur_Clientes;

Cur_Clientes.sql "SELECT CodCli, NomCli, EndCli\
                  FROM E085CLI\
                  WHERE (TipCli =:aTipoCliente)\
                  ORDER BY CodCli";

/*Execução do Cursor de Clientes*/
Cur_Clientes.AbrirCursor();

ENQUANTO (Cur_Clientes.Achou)
  Inicio
    aNomeCliente = Cur_Clientes.NomCli;
    aEnderecoCliente = Cur_Clientes.EndCli;
    aMsg = aNomeCliente + " END.: "+ aEnderecoCliente; @Monta o
    que será exibido na mensagem.@
    Mensagem (Retorna, aMsg);
    Cur_Clientes.Proximo();
  Fim;

/* Fechamento do Cursor de Clientes*/
Cur_Clientes.FecharCursor();

```

## Informações Fixas

Um hábito de desenvolvimento que deve ser evitado é o de fixar no código-fonte informações que deveriam ser parametrizáveis.

Exemplo:

- Se Cliente 1 e Valor NF >= 10.000,00 então NF modelo 1.
- Se Cliente 1 e Valor NF < 10.000,00 então NF modelo 2.
- Se Cliente 2 e Valor NF >= 10.000,00 então NF modelo 2.

- Se Cliente 2 e Valor NF < 10.000,00 então NF modelo 4.

Em uma primeira análise, é mais rápido desenvolver a Regra com as informações fixas. No entanto é necessário ser perguntar: Mesmo que haja um ganho de tempo inicial, como ficará a manutenção desta rotina?

Usando o exemplo acima, quando o Cliente 1 não usar mais o Modelo de NF 1 para valores superiores a R\$ 10.000,00, mas somente para valores superiores a R\$ 15.000,00, o usuário ficará refém de ajuste na Regra. E isso talvez até possa parecer bom, mas não é!

O que se exige de um sistema profissional é, dentre outras coisas, a possibilidade de o usuário parametrizar suas necessidades.

Ainda usando o exemplo acima, como seria possível resolver essa questão de não fixar informações no fonte da Regra? Primeiro de tudo, verificando se nativamente já existe um recurso que atenda a necessidade do cliente.

Supondo que para o exemplo citado realmente não existe nada nativo no ERP. Então poderíamos fazer assim para deixar a rotina personalizável:

- Criar uma tabela de usuário com os campos Código Cliente (chave), Valor Superior, Modelo de NF Superior, Valor Inferior, Modelo de NF Inferior.
- Criar uma tela SGI dispondo os campos da tabela de usuário criada.
- Criar um acesso de menu para o usuário poder fazer a manutenção dos parâmetros.
- Desenvolver a Regra apontando para esta tabela de usuário para definir seu comportamento.
- Colocar segurança de acesso através dos recursos de segurança dos sistemas da Senior para que somente usuários autorizados possam fazer manutenção das parametrizações.

Ainda outras soluções poderiam ser propostas, sempre nesta idéia de manter a rotina personalizável o máximo possível pelo usuário.



## AUTOATIVIDADE

Agora que você já explorou todos os recursos e informações sobre o gerador de telas, vamos praticar e criar uma tela idêntica a que foi apresentada acima no sistema Gerador de Telas do produto Gestão de Pessoas. Depois poderemos disponibilizar a tela no menu do sistema Colaboradores.

# CAPÍTULO 02

## Editor de Regras

### Objetivos Específicos

- Informar os recursos do editor de regras da sênior.

## EDITOR DE REGRAS

O Editor de Regras da Senior Sistemas é uma ferramenta muito útil e produtiva para desenvolver rotinas complementares e auxiliares as que os sistemas já possuem. A Senior possui sua própria linguagem de programação (LSP – Linguagem Senior de Programação) e seu próprio compilador. Os comandos da LSP são no idioma português e sua sintaxe é semelhante a outras linguagens de programação comumente utilizadas no desenvolvimento de sistemas, o que facilita muito a aprendizagem e produtividade. Com ela é possível desde exibir uma caixa de mensagem com um aviso ao usuário, como executar um relatório ou até mesmo um web services. Manipular informações de banco de dados fica muito fácil e simples com as regras. Leitura de arquivos, envio de e-mail, manipulação de dados em memória entre muitos outros recursos, permitem muita flexibilidade segurança e robustez nos sistemas da Senior.

Ferramentas como Gerador de Relatórios, Gerador de Telas, Importador/Exportador de Arquivos Texto dentre outros recursos, permitem a customização através da Regra. Isto garante agilidade e flexibilidade no desenvolvimento de soluções integradas ao ambiente dos sistemas da Senior.

A LSP tem uma sintaxe própria, com comandos em português. Não há distinção de letras maiúsculas e minúsculas. Portanto a LSP possui cerca de 50 palavras reservadas que não poderão ser usadas pelo programador para outros fins.

Alguns exemplos de palavras reservadas pelo compilador são: Inicio, Fim, Definir, Para, VaPara, Se, Senao, Cancela, Data, Proximo, Achou, Tabela, Pare, E, Ou, Abrir, Fechar, Ler, etc...

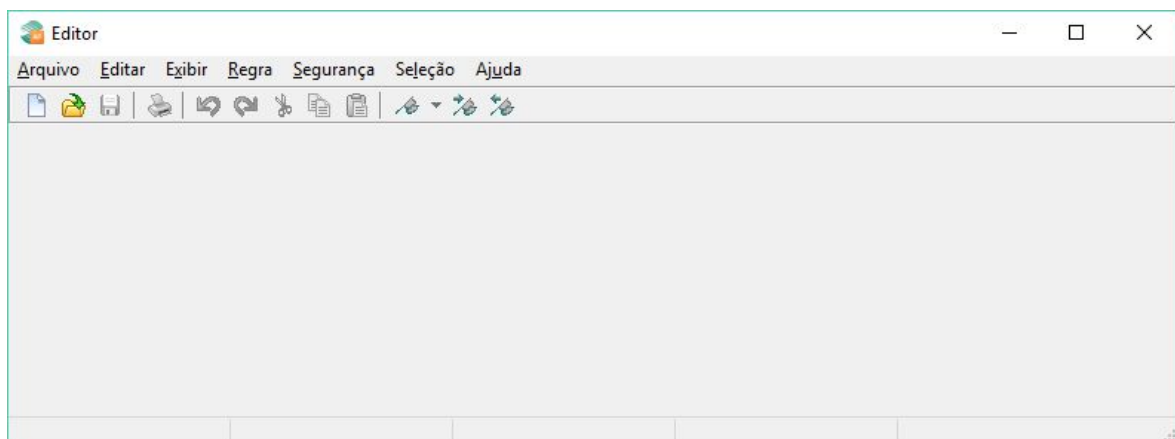
Para identificar estas palavras, poderá consultar a documentação da Linguagem Senior de Programação. No entanto, uma maneira fácil de identificar se a palavra é reservada ou não é verificar se ficou destacada em negrito, pois o Editor de Regras coloca em negrito as palavras reservadas.

## O Editor de Regras

Para abrirmos o editor de regras, temos que acessar o seguinte menu de sistema:

Gestão de Pessoas: Recursos/Implementações/Editor de Regras;

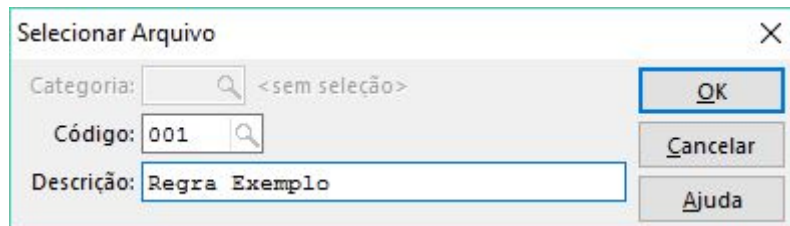
Gestão Empresarial: Recursos/Implementações/Regras/Definir;



Na figura anterior visualizamos a tela principal do Editor de Regras. É através deste editor que serão desenvolvidas as regras em todas as ferramentas que utilizam regras. Para o desenvolvimento das regras vamos utilizar a Linguagem Senior de Programação (LSP).

### Criando uma Nova Regra

Para criar uma nova regra deve-se acessar o menu Arquivo/Novo(Ctrl+N) ou clicar no botão “Novo” na barra de ferramentas. Vai abrir uma tela conforme figura a seguir:



A imagem mostra uma janela de diálogo intitulada "Selecionar Arquivo". Ela possui três campos de entrada: "Categoria:" com o valor "<sem seleção>", "Código:" com o valor "001", e "Descrição:" com o valor "Regra Exemplo". Cada campo tem um ícone de lupa à direita. À direita dos campos, há três botões empilhados: "OK", "Cancelar" e "Ajuda".

Informe um código e uma descrição para a regra e clique no botão “OK”. A regra será salva no diretório configurado na chave "Regras" do aplicativo "Editor de Configuração" com a extensão LSP.

O editor é composto por um menu de opções, logo abaixo está a barra de ferramentas para as opções mais usadas e abaixo encontra-se a área de edição. No lado esquerdo da área de edição encontra-se a numeração das linhas.

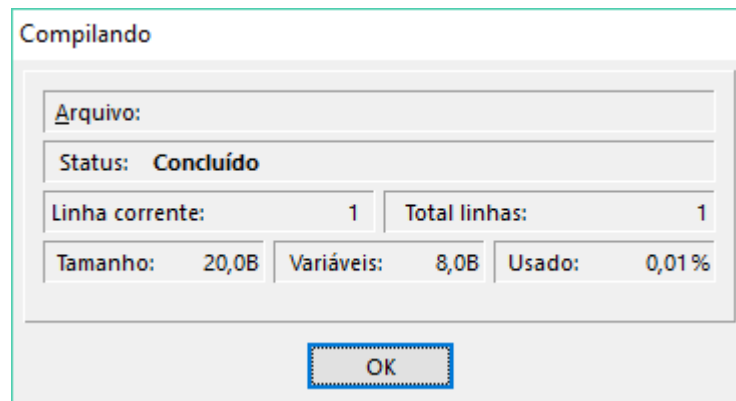
Para um exemplo simples, pode ser digitado no editor o comando abaixo:

```
Mensagem(Retorna, "Olá mundo!");
```

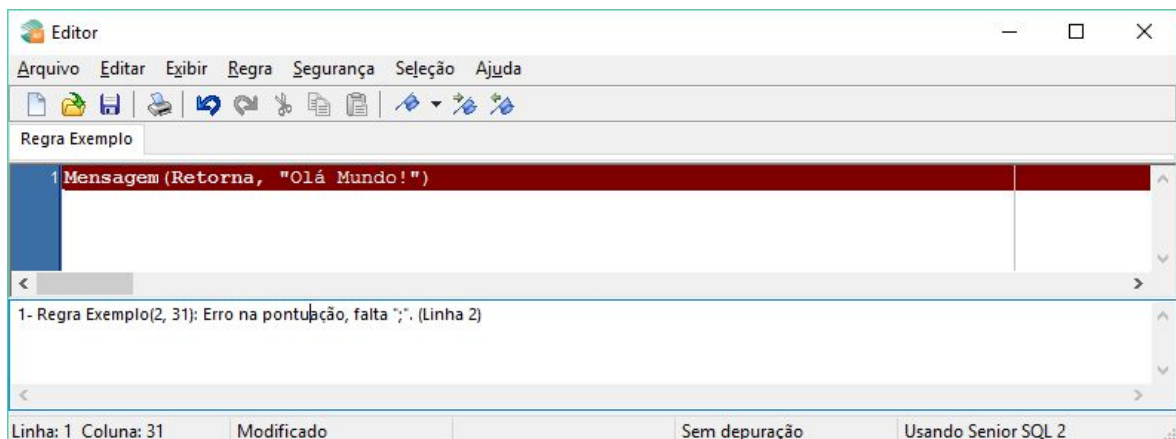
### Compilando uma Regra

Compilar uma regra significa submeter o código fonte escrito, a uma análise criteriosa do compilador, que verificará se a sintaxe de todos os comandos (os comando serão abordados mais adiante) estão corretos. Se os comandos estão corretos, o compilador irá gerar as instruções de execução e irá adicioná-las ao arquivo binário do sistema (.BIN) para posterior execução. O arquivo binário também é gerado ao compilar as regras no diretório configurado na chave "Regras" do aplicativo "Editor de Configuração".

Para compilar essa regra de exemplo, acesse o menu “Regra/Compilar Arquivo” ou pressione as teclas de atalho Shift+F9, vai abrir a tela de compilação conforme a figura a seguir:



O Status “Concluído” indica que a compilação da regra foi efetuada com sucesso. Já o Status “Existem erros...”, indica que a compilação não foi bem sucedida por existir algum erro na construção da regra. O cursor será posicionado na primeira linha e coluna que possuir algum problema. A linha ficará destacada com a cor bordô, conforme a figura a seguir:



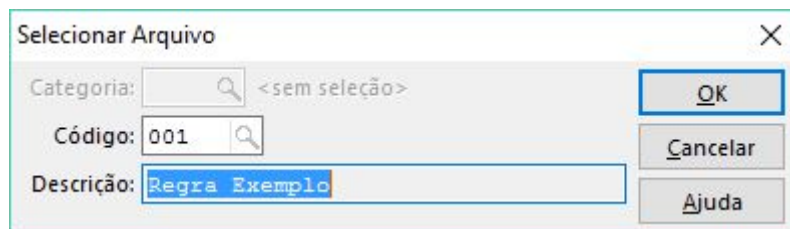
Sempre que a compilação não ocorrer com sucesso, será exibida na parte inferior do editor, uma mensagem descrevendo o problema. No caso acima, aconteceu um erro pela falta de “;” (ponto e vírgula). Para corrigir deve ser informado um ponto e vírgula (;) no final do comando. Após corrigidos todos os problemas deve-se compilar a regra novamente, até obter sucesso.

### Salvando uma Regra

Para salvar uma regra, basta clicar no botão “Salvar”, ou acessar o menu “Arquivo/Salvar”, ou ainda pressionar as teclas de atalho “Ctrl+S”.

### Abrindo uma Regra Existente

Para abrir uma regra, abra o Editor de Regras, através do menu correspondente do sistema. No editor clique no botão abrir ou acesse o menu “Arquivo/Abrir”, ou ainda pressione as teclas de atalho “Ctrl+A”. Vai abrir a tela “Selecionar Arquivo”, conforme figura a seguir:



Selecionar Arquivo

Categoria: <sem seleção>

Código: 001

Descrição: Regra Exemplo

OK Cancelar Ajuda

Nessa tela informe o código da regra, ou clique no botão de reticências no campo “Código” e selecione a regra desejada. Para concluir a operação, clique no botão “OK”.

### Menus do Editor de Regras

Através dos menus do Editor de Regras é possível efetuar o gerenciamento das regras de forma rápida e fácil. A seguir serão mostrados os principais menus e uma breve descrição da função de cada um.

Menu Arquivo			
Ação	Descrição	Atalho	
 Novo	Possibilita criar uma nova regra.		Ct
 Abrir	Abre uma regra existente.		Ct
 Salvar	Salva a regra corrente.		Ct
 Salvar	Salva todas as regras abertas.		
Todos			
Como Salvar	Salva a regra corrente com outro nome.		S
 Excluir	Exclui a regra corrente. Solicita confirmação antes de excluir.		
 Fechar	Fecha a regra corrente.		Ct
			rl+F4
Todos Fechar	Fecha todas as regras abertas.		
Configurar Impressora	Possibilita ao usuário alterar as configurações de impressão.		
 Imprimir	Imprimir o texto da regra corrente.		Ct
			rl+I
 Sair	Sai do Editor de Regras.		Alt
			t+F4

Menu Editar		
Ação	Descrição	Atalho



 <b>Desfazer</b>	Desfaz a última ação realizada.	<b>Ctrl+Z</b>
 <b>Refazer</b>	Refaz a última ação realizada.	<b>Shift+Ctrl+Z</b>
 <b>Recortar</b>	Move o texto selecionado no editor para a área de transferência do Windows.	<b>Ctrl+X</b>
 <b>Copiar</b>	Copia o texto selecionado no editor para a área de transferência do Windows.	<b>Ctrl+C</b>
 <b>Colar</b>	Insere no editor logo após a posição do cursor o texto armazenado na área de transferência do Windows.	<b>Ctrl+V</b>
<b>Selecionar Tudo</b>	Seleciona todo o texto contido no editor de regras.	<b>Ctrl+T</b>
<b>Modo de Seleção</b>	Permite definir diferentes modos de seleção para o texto.	
<b>Macros</b>	Permite gravar ações para reproduzi-las posteriormente de forma automatizada.	
<b>Indentar</b>	Permite deslocar blocos inteiros de texto para direita ou para esquerda.	
 <b>Localizar</b>	Permite localizar um texto em específico no texto do editor.	<b>Ctrl+L</b>
 <b>Localizar Próxima</b>	Repete a última busca após a última ocorrência encontrada.	<b>F3</b>
<b>Substituir</b>	Substitui um texto por outro.	<b>Ctrl-U</b>
<b>Ir Para Linha</b>	Posiciona o cursor na linha especificada.	<b>Alt+G</b>
<b>Retornar Última Versão</b>	<b>Salva Volta o texto da regra exatamente como após ter sido salvo pela última vez.</b>	<b>Ctrl+M</b>

A maioria dos itens do menu Editar são auto-explicativos, porém alguns não. A seguir serão abordados alguns itens que necessitam de mais detalhamento.

### Modo de Seleção

Permite definir diferentes modos de seleção para o texto. Os modos de seleção possíveis são:

#### *Normal*

O texto é selecionado desde a posição do clique inicial até a posição onde liberou o botão do mouse.

Exemplo:

```

20 Definir Alfa ClaSal;
21 Definir Alfa NivSal;
22 Definir Alfa aClaIni;
23 Definir Alfa aClaFim;
24 Definir Alfa aNivIni;
25 Definir Alfa aNivFim;
26 Definir Alfa alfaEstCar;

```

### Coluna

Permite selecionar colunas sempre em blocos retangulares.

Exemplo:

```

20 Definir Alfa ClaSal;
21 Definir Alfa NivSal;
22 Definir Alfa aClaIni;
23 Definir Alfa aClaFim;
24 Definir Alfa aNivIni;
25 Definir Alfa aNivFim;
26 Definir Alfa alfaEstCar;

```

### Macros

Este menu permite gravar e executar macros. Uma macro representa um conjunto de ações realizadas no código do Editor de Regras, que pode ser aplicado de forma automática a outra parte qualquer de código com a execução dessa macro.

Para iniciar a gravação de uma macro, acesse o menu Editar/Macros/Iniciar Gravação, ou pressione as teclas de atalho Shift+Ctrl+R. Exemplo:



A partir de agora o editor estará gravando as ações que você efetuar no texto. Após ter concluído as ações desejadas, acesse novamente o menu Editar/Macros/Terminar Gravação, ou pressione as teclas de atalho Shift+Ctrl+R. **Exemplo:**



Para executar a macro gravada nos passos anteriores, acesse o menu Editar/Macros/Executar Macro ou pressione as teclas de atalho Shift+Ctrl+P.

Exemplo:

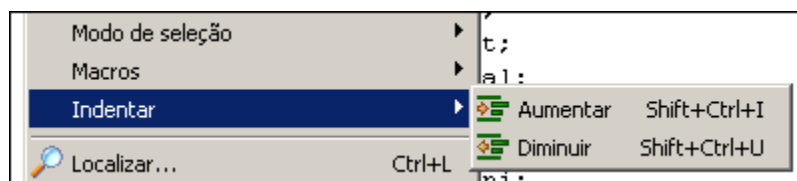


### DICA

Dependendo da ação gravada, para aplicar a macro corretamente pode ser necessário selecionar o texto no qual se pretende aplicar a macro ou posicionar o cursor no início do texto onde a macro deve ser aplicada.

### Indentar

Esta opção de menu permite deslocar blocos inteiros de texto para direita ou para a esquerda. O menu apresenta as duas opções conforme imagem abaixo:



Para efetuar a indentação de um bloco de texto para a direita, proceda da seguinte forma:

- Selecione o bloco de texto desejado;
- Vá até o menu Editar/Indentar/Aumentar, ou pressione as teclas de atalho Shift+Ctrl+I.

Para efetuar a indentação de um bloco de texto para a esquerda, proceda da seguinte forma:

- Selecione o bloco de texto desejado;
- Vá até o menu Editar/Indentar/Diminuir, ou pressione as teclas de atalho Shift+Ctrl+U.

Menu Exibir		
Ação	Descrição	Atalho
		ho

<b>Barra de Ferramentas</b>	Ocultar/Exibir a Barra de Ferramentas do Editor de Regras.	-
<b>Barra de Status</b>	Ocultar/Exibir a Barra de Status do Editor de Regras.	-
<b>Editor Avançado</b>	Habilitar/Desabilitar os recursos avançados do Editor de Regras.	-
<b>Menu Regra</b>		
<b>Ação</b>	<b>Descrição</b>	<b>Atalho</b>
<b>Depurar Arquivo</b>	Habilitar/Desabilitar a depuração da regra.	<b>Ctrl+F8</b>
<b>Ativar Depuradores</b>	Habilitar a depuração de todas as regras.	<b>Shift+F10</b>
<b>Desativar Depuradores</b>	Desabilitar a depuração de todas as regras.	<b>Ctrl+F10</b>
<b>Usar SQL 2</b>	Habilitar/Desabilitar a utilização da linguagem SQL Senior versão 2.	<b>Ctrl+F12</b>
<b>Visualizar Regras</b>	Permite a visualização de todas as regras no editor sem poder efetuar alterações.	
<b>Compilar Arquivo</b>	Compila a regra corrente.	<b>Shift+F9</b>
<b>Compilar Todos</b>	<b>Compila todas as regras.</b>	<b>Ctrl+F9</b>

<b>Menu Segurança</b>		
<b>Ação</b>	<b>Descrição</b>	<b>Atalho</b>
<b>Definir</b>	<b>Possibilita definir permissões de acesso para uma determinada regra.</b>	-

<b>Menu Seleção</b>		
<b>Ação</b>	<b>Descrição</b>	<b>Atalho</b>
<b>Comandos</b>	Permite selecionar comandos de programação de regra.	<b>Shift+F5</b>
<b>Funções</b>	Permite selecionar funções de programador.	<b>Shift+F6</b>
<b>Variáveis</b>	Permite selecionar variáveis de sistema para utilizar na regra.	<b>Shift+F7</b>
<b>Tabelas e Campos</b>	<b>Permite selecionar tabelas e campos do sistema para utilizar na regra.</b>	<b>Shift+F8</b>

<b>Menu Ajuda</b>		
<b>Ação</b>	<b>Descrição</b>	<b>Atalho</b>

o	Conteúdo	Abre a tela de ajuda sobre o Editor de Regras.
		1


## Barra de Ferramentas



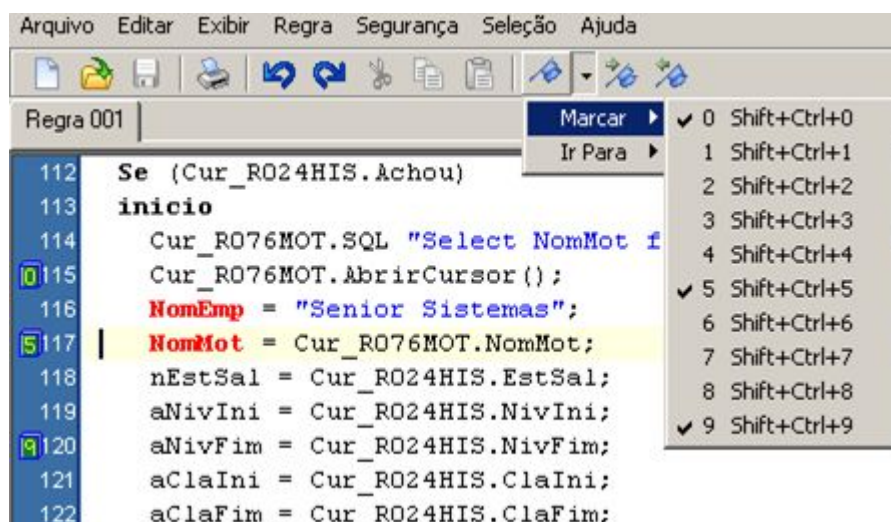
A barra de ferramentas possibilita acesso rápido às principais funcionalidades do Editor de Regras. A maioria das opções da barra de ferramentas já foram abordadas nos itens de menu anteriormente, com exceção dos marcadores que serão abordados a seguir.

## Marcadores


Os marcadores servem para marcar linhas do editor nas quais deseja-se ir posteriormente de forma rápida e prática.

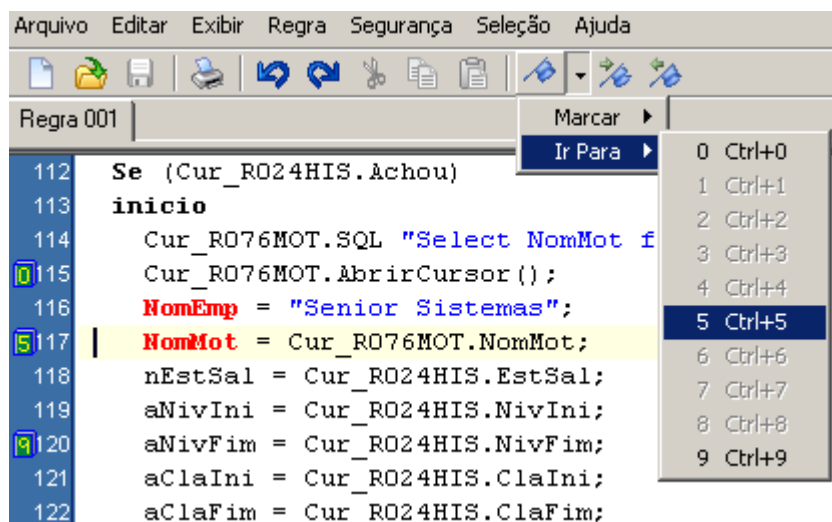
Para inserir um novo marcador, clique na linha onde deseja inseri-lo, vá no botão  da barra de ferramentas e clique na setinha preta apontando para baixo. No menu que se abre, clique em Marcar e em seguida escolha um dos marcadores disponíveis. É possível inserir até 10 marcadores cujos números vão de 0 até 9. Quando clicar sobre um deles, aparecerá um sinal de checado do lado esquerdo do menu e na margem esquerda numerada do editor aparecerá um pequeno quadrado com o número correspondente no centro. Para remover um marcador, basta acessar o menu Marcar na barra de ferramentas e clicar sobre o item checado anteriormente.



Também é possível inserir ou remover marcadores através das teclas de atalho que estão no próprio menu Shift+Ctrl+ número de 0 até 9, ou ainda clicando com o botão direito do mouse no editor escolhendo o menu Marcadores/Marcar/Escolha o marcador desejado.





Após inserir marcadores é possível utilizá-los para movimentar-se rapidamente pelas linhas marcadas.

Para ir a uma linha marcada clique no botão  na barra de ferramentas, no menu que aparece clique em Ir Para, apenas os marcadores já inseridos estarão habilitados, basta clicar sobre um deles ou utilizar as teclas de atalho que o próprio menu indicar (Ctrl+Números de 0 até 9). Ou ainda clicando com o botão direito do mouse no editor escolhendo o menu Marcadores/Ir Para/Escolha o marcador desejado.



Após adicionar marcadores também é possível mover-se por eles através dos botões  e .

- : Vai para o próximo marcador, se já estiver posicionado no último marcador, vai para o primeiro.
- : Vai para o marcador anterior, se já estiver no primeiro, vai para o último marcador.





### AUTOATIVIDADE

Agora que você já explorou todos os materiais disponíveis vamos praticar a navegação na regra através de marcadores.



# CAPÍTULO 03

## Comandos, Variáveis e Funções

### Objetivos Específicos

- Identificar os comandos, variáveis e funções nas regras LSP

## COMANDOS, VARIÁVEIS E FUNÇÕES.

### Comandos

Em razão do grande número de comandos serão apresentados na apostila apenas uma lista dos comandos mais utilizados na linguagem LSP, estas palavras são reservadas em qualquer ponto de regra no sistema e em todas as ferramentas de implementação, e por este motivo não podem ser utilizadas como variáveis definidas no sistema.

Resumo dos Comandos	
Comando	Descrição breve
Início ou { (abre chaves)	Início de comando de bloco.
Fim ou } (fecha chaves)	Fim de comando de bloco.
Definir	Usado para declarar variáveis e funções.
Para	Comando para laço de repetição com quantidade de iterações conhecidas.
Enquanto	Comando para laço de repetição com quantidade de iterações indeterminadas.
VaPara	Desvia a execução do programa para um identificador determinado.
Regra	Executa outra regra identificada pelo número da regra.
Usar	Possibilita utilizar outras regras do mesmo arquivo binário dentro de uma já existente. Deve ser o primeiro comando da regra. E: Usar 1, 2, 3;. Onde 1, 2, 3 são os números das regras (arquivos LSP).
Se	Executa o bloco a seguir se a condição for verdadeira.
Senao	Usado com o comando se. Se a condição for falsa o bloco após o senao será executado.
Cancel	utilizado para cancelar a regra, aceita os parâmetros: 1, 2 e 3.
Alfa	Usado com o comando "Definir" para declarar uma variável alfanumérica. Pode ser especificado o tamanho máximo de caracteres que a variável comportará.
Numero	Usado com o comando "Definir" para definir uma variável numérica.
Tabela	Variável que armazena valores em forma de tabela, onde tem-se linhas e colunas. Cada coluna é um nome com um tipo específico de informação. As linhas são indexadas de "1" até "N". É utilizado com o comando "Definir" para criar uma tabela de variáveis.
Funcao	Permite definir uma função pelo próprio usuário.
Continue	Utilizado em laços de repetição (Para ou Enquanto) para abandonar a iteração atual, voltando para o início da iteração.
Pare	Interrompe a execução de laços de repetição (Para ou Enquanto). O comando "Pare" faz com que o sistema abandone o laço de repetição e continue a execução do restante da regra.
End	Indica passagem de parâmetro por referência em uma função.

E	Utilizado com o comando “Se”, para ligar várias condições, onde todas devem ser verdadeiras para que o resultado da comparação seja verdadeiro.
Ou	Utilizado com o comando “Se”, para ligar várias condições, onde pelo menos uma das condições, deve ser verdadeira para que o resultado da comparação seja verdadeiro.
Abrir	Abre o arquivo informado em “nome do arquivo” para o “modo de abertura” informado (Ler/Gravar). Se o arquivo não existir, ele é criado. Ele retorna um manipulador de arquivos.
Ler	Lê uma quantidade de caracteres especificados em “tamanho”, de um arquivo especificado no “manipulador de arquivo” e joga o valor lido na “variável” especificada.
Fechar	Fecha um arquivo aberto anteriormente pelo comando “Abrir”.
Gravar	Grava o valor da “constante” ou da “variável” com uma quantidade de caracteres especificados em “tamanho” no arquivo especificado no “manipulador de arquivo”.
Lernl	Lê uma linha do arquivo indicado pelo “manipulador de arquivo” e joga o valor lido para a “variável” indicada.
Formatar	Retorna um dado no formato especificado.
FormatarN	Usado para formatar números comumente com casas decimais, tem seu funcionamento semelhante ao da função “Formatar”. Entretanto, implementa o terceiro parâmetro, referente ao separador de casas decimais e o quarto parâmetro, que armazena o resultado da formatação em uma variável.
Mensagem	Exibe uma caixa de mensagem para o usuário.
AtualizarCampos	Atualiza os dados de todos os campos de uma tela de cadastramento.
IniciarTransacao	Inicia uma transação no banco de dados.
FinalizarTransacao	Finaliza a transação no campo de dados.
DesfazerTransacao	Desfaz uma transação iniciada anteriormente no banco de dados.
ValRet	Retorna um valor numérico para a aplicação.
ValStr	Retorna um valor alfanumérico para a aplicação.
AbriCursor	Abri um cursor definido pelo comando Cursor.
FecharCursor	Fecha um cursor definido pelo comando Cursor.
Achou	Retorna verdadeiro se achou alguma linha no cursor.
NaoAchou	Retorna verdadeiro se não achou nenhuma linha no cursor.
SQL	Usado para definir um comando SQL para o cursor.
Próximo	Lê o próximo registro do cursor.

## Operadores

Os operadores que são utilizados nas regras e fórmulas podem ser:

### Operadores Lógicos



inal	Descrição
	Sinal igual. Utilizado em comparações/operações aritméticas
	Maior que. Utilizado nas comparações
	Menor que. Utilizado nas comparações
>	Diferente de. Utilizado nas comparações
=	Maior ou igual a. Utilizado nas comparações
=	Menor ou igual a. Utilizado nas comparações
	“E”. Utilizado com o comando “Se”, para ligar várias condições, onde todas devem ser verdadeiras para que o resultado da comparação seja verdadeiro.
u	“OU”. Utilizado com o comando “Se”, para ligar várias condições, onde pelo menos uma das condições deve ser verdadeira para que o resultado da comparação seja verdadeiro.

### Operadores Aritméticos

inal	S	Descrição
	=	Sinal igual. Utilizado em comparações/operações aritméticas
	+	Sinal de Somar. Utilizado nas operações aritméticas de somar
	-	Sinal de Subtrair. Utilizado operações aritméticas de subtrair
	/	Sinal dividir. Utilizado nas operações aritméticas de divisão
	*	Asterisco. Utilizado nas operações aritméticas de multiplicação
+	+	Incremento de +1. Utilizado para aumentar o valor de uma variável, de um em um.
-	-	Decremento de - 1. Utilizado para diminuir o valor de uma variável, de um em um.

### Operadores Extras

ímbolo	S	Descrição
	(	<b>Delimitador utilizado para incluir comentários que tenham no máximo uma linha.</b>
*	/	<b>Início de comentário de bloco de comandos.</b>
/	*	<b>Final de comentário de bloco de comandos.</b>

### Comando Se

Se (<condição verdadeira>)

**Inicio**

<executa comando 1>;

<executa comando 2>;

**Fim;**

Ou

**Se** (<condição verdadeira>)

**Inicio**

<executa comando 1>;

<executa comando 2>;

**Fim;**

**Senao**

**Inicio**

<executa comando 3>;

<executa comando 4>;

**Fim;**

Exemplo:

**Definir Alfa** aDiaSemana;

**Definir Numero** nDia;

aDiaSemana = "";

nDia = 3;

**Se** (nDia = 1)

aDiaSemana = "Segunda-Feira";

**Senao**

**Se** (nDia = 2)

aDiaSemana = "Terça-Feira";

**Senao**

**Se** (nDia = 3)

aDiaSemana = "Quarta-Feira";

**Senao**

**Se** (nDia = 4)

aDiaSemana = "Quinta-Feira";

**Senao**

**Se** (nDia = 5)

aDiaSemana = "Sexta-Feira";

**Senao**

**Se** (nDia = 6)

aDiaSemana = "Sábado";

**Senao**

**Se** (nDia = 7)

aDiaSemana = "Domingo";

**Senao**

aDiaSemana = "Dia da Semana Inválido!";

**Mensagem** (Retorna, aDiaSemana);

O comando “Se” executa um ou mais comandos se a condição avaliada for verdadeira. Pode ser utilizado em conjunto com o “Senao” que executa comandos se a condição for falsa.



### IMPORTANTE

Início ou { (abre chaves): Indica o início de um bloco de comandos o qual permite a inclusão de outros blocos. Fim ou } (fecha chaves): Indica o fim de um bloco de comandos.

### Laço de repetição Para

Esse comando permite fazer um loop (repetição) de comandos, ou seja, que vários comandos sejam executados determinados número de vezes. Indica-se um <valor inicial> e esse valor é incrementado pelo valor do <contador> até que a <condição> seja falsa. Esse comando é empregado normalmente quando é conhecido o número de iterações a realizar, no exemplo abaixo  $i \leq 7$  (sete vezes)

#### Sintaxe:

**Para** (<valor inicial>; <condição>; <contador>)

##### Início

<executa comando 1>;

<executa comando 2>;

<executa comando n>;

##### Fim;

#### Exemplo:

**Definir Alfa** aDiaSemana;

**Definir Numero** nDia;

aDiaSemana = "";

nDia = 30;

**Para** (i = 1; i <= nDia; i++)

##### Início

**Se** (nDia = 1)

aDiaSemana = "Segunda-Feira";

##### Senao

**Se** (nDia = 2)

aDiaSemana = "Terça-Feira";

##### Senao

**Se** (nDia = 3)

aDiaSemana = "Quarta-Feira";

##### Senao

**Se** (nDia = 4)

aDiaSemana = "Quinta-Feira";

```

Senao
Se (nDia = 5)
    aDiaSemana = "Sexta-Feira";
Senao
Se (nDia = 6)
    aDiaSemana = "Sábado";
Senao
Se (nDia = 7)
    aDiaSemana = "Domingo";
Senao
    Pare;
Fim;
Mensagem (Retorna, aDiaSemana);

```

## Laço de repetição Enquanto

Também é um comando utilizado para fazer uma repetição (loop) de comandos. Ou seja, fazer com que um bloco de comandos seja executado determinado número de vezes até que a <condição> seja falsa (normalmente emprega-se o comando enquanto quando não é conhecido o número de iterações que se deseja fazer).

**Sintaxe:**

**Enquanto** (<condição verdadeira>)

**Início**

<executa comando 1>;

<executa comando 2>;

<executa comando n>;

**Fim;**

**Exemplo:**

@ -- Regra para inversão de um texto de uma variável e exibição -- @

@ -- da mesma de forma invertida -- @

Definir Alfa aTexto;

**Definir Alfa** aNovoTexto;

Definir Alfa aTextoAux;

aTexto = "String que deve ser escrita de forma invertida!!!";

aNovoTexto = "";

aTextoAux = "";

**Enquanto** (aTexto <> "")

Início

aTextoAux = aTexto;

**TamanhoAlfa**(aTextoAux,nTamanho); @ recupera o tamanho da string @

**CopiarAlfa**(aTextoAux,nTamanho,1);@ copia o último caractere @

aNovoTexto = aNovoTexto + aTextoAux;

aTextoAux = aTexto;

**DeletarAlfa**(aTextoAux,nTamanho,1); @ exclui o último caractere @

```
aTexto = aTextoAux;  
Fim;  
Mensagem(Retorna,aNovoTexto);
```

## Definição de variáveis

Variáveis são palavras que podemos definir e utilizar na elaboração das Regras, tornando-se assim especiais, ou reservadas, dentro daquela Regra. Atribuímos além de um NOME também um TIPO a cada variável criada, geralmente sendo dos tipos numérico, alfa ou data.

A vantagem das variáveis é que podemos atribuir VALORES as mesmas (que chamamos de “Constantes”), em tempo de execução da Regra, desde que respeitando-se seu tipo definido. Por exemplo, não conseguiremos atribuir a palavra “Senior” a uma variável do tipo numérica; nem tampouco a data 15/09/2011 a uma variável do tipo Alfa, e assim por diante.

Praticamente podemos afirmar que não existem Regras de média ou alta complexidade sem o uso de variáveis; mesmo as mais simples costumam usar variáveis.

O “nome” da variável pode ser composto de letras e/ou números ou o caractere “\_” (underscore), não podendo ter mais de 30 caracteres e tendo que iniciar, obrigatoriamente, por uma letra.

Uma sugestão é que crie-se o nome da variável indicando-se o seu tipo. Por exemplo, ao invés de criar uma variável do tipo numérica com o nome NrDias, criar como nNrDias, sendo que o “n” identificaria que ela é do tipo Numérica. Ou uma variável do tipo alfa ser precedida de “a”, como NomeCliente ser identificado como aNomeCliente. Isso facilitaria durante a Regra identificar seu tipo apenas observando seu nome.

A sintaxe para definir uma variável no Editor de Regras é: Definir <tipo da variável> <nome da variável>;

Exemplo a seguir:

```
@ Definição de Variáveis @  
Definir Numero nCodigo;  
Definir Alfa aNome;  
Definir Data dDataCadastro;  
Definir Tabela tMeses[12] = {Alfa aNomeMes[10]; Numero nNumeroDias;};  
Definir Cursor Cur_NomeTabela;  
Definir Funcao FTriplo(Numero nValor, Alfa aTexto);  
Definir Lista LCidade;
```

**Numero:** Usado para definir variáveis do tipo numéricas que representam tanto números inteiros como de ponto flutuante.

**Alfa:** Utilizado para definir variáveis que irão armazenar texto, pode-se determinar a quantidade máxima de caracteres que a variável conterà colocando entre colchetes no final do nome da variável, ex:Definir Alfa aNome[50].



**Data:** Defini variáveis que conterão datas.

**Cursor:** Esse tipo de variável permite que através de comandos SQL sejam recuperados e manipulados dados de tabelas do banco de dados. Devido à importância e utilidade desse comando existe o módulo II de treinamento que o aborda com mais detalhes.

**Função:** Permite que o próprio programador das regras crie e utilize funções auxiliares conforme as suas necessidades. A vantagem da função, é que se existe uma operação que é repetida em muitas regras, pode-se criar a função e chamá-la em cada regra, sem precisar implementá-la novamente. Uma função pode receber parâmetros e retornar valores.

**Lista:** É um tipo que permite adicionar campos com outros tipos em memória e manipular esses campos de várias formas. Pode-se incluir dados, alterar, remover, pesquisar e utilizá-los para atender necessidades de manipulações diversas. Esse tipo é abordado com mais ênfase no módulo II do treinamento.

**Tabela:** Permite definir variáveis do tipo tabela. Nesse tipo de dados, as informações serão armazenadas no formato de linha e coluna. Cada coluna é um nome com um tipo específico de informação. Esse tipo é abordado com mais ênfase no módulo III do treinamento.

Exemplo:

```
@ --- Definição de Variáveis --- @
Definir Alfa aNomeCliente;
Definir Numero nNumeroPedido; @ As variáveis numéricas não precisam
ser definidas, @
Definir Numero nTotalPedido; @ por padrão, toda variável não
definida é numérica. @
Definir Data dDataVenda;
Definir Alfa aNumeroPedido;
Definir Alfa aTotalPedido;
Definir Alfa aDataVenda;
Definir Alfa aMensagem;
Definir Alfa aComDesconto;

aNomeCliente = "Cliente Exemplo"; @ Atribuição de texto a variável @
nNumeroPedido = 100; @ Atribuição de Número a variável @
nTotalPedido = 234.56; @ Atribuição de Valor a variável @
@ --- Uso de variável numérica não declarada para receber o valor ---@
nComDesconto = nTotalPedido * 0.9;
MontaData(08,12,2011,dDataVenda); @ função que monta uma data em uma
variável @
IntParaAlfa(nNumeroPedido,aNumeroPedido); @ Converte um inteiro para
alfanumérico @
@ --- Conversão de Valores para texto de acordo com uma máscara --- @
ConverteMascara(1,nTotalPedido,aTotalPedido,"zzz.zz9,99");
ConverteMascara(1,nComDesconto,aComDesconto,"zzz.zz9,99");
@ --- Conversão de Data para texto de acordo com uma máscara --- @
ConverteMascara(3,dDataVenda,aDataVenda,"DD/MM/YYYY");
@ --- Concatenação de Instrings --- @
aMensagem = "Pedido:" + aNumeroPedido + ", Cliente: " + aNomeCliente +
", Total Pedido: " + aTotalPedido + ", Com Desconto: " +
aComDesconto +
", Data de Venda: " + aDataVenda;
Mensagem(Retorna,aMensagem); @ Exibição da Mensagem para o usuário @
```

A regra acima exemplifica a declaração de variáveis, o emprego de comentários e o uso de funções auxiliares para conversão de valores numéricos para texto. Note que no final de cada comando deve-se colocar um ponto e vírgula (;). É possível atribuir um valor diretamente a uma variável ex: "nTotalPedido = 158.75;" ou através de uma expressão: "nComDesconto = nTotalPedido \* 0.9;". Também foram utilizadas várias funções auxiliares para manipular as informações, essas funções serão abordadas mais adiante. Na linha "29" temos uma concatenação de textos, cujo operador de concatenação é o símbolo "+". Note que o texto deve estar entre aspas duplas ("").

Os textos em verde são comentários, eles servem para documentar o código. Existem dois tipos de comentários: de linha e de bloco. Para o comentário de linha o texto deve ser precedido e sucedido pelo símbolo "@". Já para os comentários de bloco o texto deve estar entre "/\*" e "\*/", sem aspas duplas.

Para definir uma variável deve ser empregado o comando "Definir" seguido do tipo da variável, (ex: Alfa para texto) e em seguida o nome da variável. O compilador das regras não diferencia letras maiúsculas de minúsculas, portanto escrever o nome de uma variável "Nome", "NOME" ou "NoMe" será a mesma variável.

### Conversão de variáveis

Durante a execução da Regra, perceberá que muitas vezes precisará converter o conteúdo de variáveis de um tipo para outro. Para este fim, existem funções prontas de conversão.

Algumas situações de exemplo em que é necessária conversão de tipo de variável:

**IntParaAlfa:** Se quiser exibir uma Mensagem, você já sabe que o conteúdo de sua exibição deve sempre ser do tipo alfa. Então se estiver manipulando valores numéricos, terá de convertê-los para alfa antes de colocá-los no parâmetro do comando "Mensagem".

Exemplo:

```
Definir Alfa aCodigoCliente;  
Definir Numero nCodigoCliente;  
Definir Alfa aMsg;  
nCodigoCliente = 150;  
IntParaAlfa (nCodigoCliente, aCodigoCliente);  
aMsg = "Cliente = " + aCodigoCliente;  
Mensagem (Retorna, aMsg);
```

**AlfaParaInt:** Se estiver lendo o conteúdo de um campo alfa e souber que o conteúdo deste campo for sempre um número, poderá desejar convertê-lo de alfa para numero a fim de utilizá-lo em alguma conta aritmética.

Exemplo:

```

Definir Alfa aProduto;
Definir Numero nProduto;
aProduto = "9701";
AlfaParaInt (aProduto, nProduto);
nCodigoProduto = nCodigoProduto + 1;

```

## Conversão de máscaras

Muitas vezes deseja-se exibir ou gravar o valor de uma variável em um formato diferente do que está carregada na memória. Para isto temos a função `ConverteMascara`. Sua sintax é:

```
ConverteMascara (Tipo_dado,Valor_Origem,Alfa_Origem/Destino,Mascara);
```

Por exemplo, o número "02412438939" armazenado em uma variável na verdade é um CPF e gostaríamos de exibi-lo mascarado. No caso do CPF, usamos um comando semelhante a este:

```
ConverteMascara (1,nNumCPF,aNumCPF,"999.999.999-99");
```

Existem outros Tipos de conversão de máscara na função `ConverteMascara`, como por exemplo, 1-número, 2-dinheiro (valor), 3-data, 4-hora, 5-alfa.

Se carregar para uma variável o valor da função `DatSis`, o ERP carregará a data atual do Sistema para dentro da variável.

No entanto, o ERP utiliza as datas em formato número, sempre somando 1 a partir de 31/12/1900.

Isto significa que 01/01/1901 = 1, 10/01/1901 = 10 e 26/09/2009 = 39716, e assim sucessivamente.

No entanto, talvez queira exibir (ou gravar) a data em formato dd/mm/aaaa e não em número (ou ainda em outro formato, como aaaa/mm/dd, etc). Neste caso, também terá de converter a variável do tipo número para tipo data.

Exemplo:

```

Definir Alfa aDataHoje;
Definir Numero nDataHoje;
nDataHoje = DatSis;
Convertemascara(3,nDataHoje,aDataHoje,"DD/MM/YYYY");
Mensagem (Retorna,aDataHoje);

```

## Entrada de Valor

Se desejar permitir que o usuário entre com um valor durante a execução da Regra, poderá utilizar a função `EntradaValor`. O valor digitado será armazenado em uma variável, podendo assim ser utilizado durante o restante da Regra.

Os tipos possíveis de entrada de valores são: 1-Numero, 2-Dinheiro, 3-Data, 4-Hora, 5-Alfa e 6-Senha.

Exemplo:

```
Definir Alfa aRetorno1;
Definir Numero nCodCliente;
Definir Numero nRetorno2;
Definir Numero nRetorno3;
EntradaValor("Entrada de Dados","Informe o Cliente: ",5,"ZZZZZZZ","
",50,aRetorno1,nRetorno2,nRetorno3);
Mensagem (Retorna,aRetorno1);
```

## Cancel

O cancel é uma variável que permite o cancelamento nas regras LSP. Segue uma relação do comportamento do cancel nas regras LSP.

Evento/Cancel	Cancel(1)	Cancel(2)	Cancel(3)
Na Impressão	- Cancela a impressão do campo; - Considera para os Totais.	- Cancela a impressão do campo ou lista o conteúdo da variável ValStr; - Considera para os Totais.	- Cancela a impressão do campo; - Não considera para os Totais.
Antes de Imprimir	- Cancela a impressão da Seção; - Considera para os Totais.	- Cancela a impressão da seção; - Considera para os Totais.	- Cancela a impressão da seção, ou do registro se estiver no detalhe; - Não considera para os Totais.
Seleção	- Cancela a impressão do Registro	- Cancela a impressão do registro	- Cancela a Impressão do registro
Pré-Seleção	- Aborta a execução do relatório	- Aborta a execução do relatório	- Aborta a execução do relatório

## Funções

Funções são comandos pré-definidos no sistema, que efetuam operações específicas, e que podem ou não retornar valores. Geralmente, uma função quando executada, necessita de parâmetros.

Podemos acessar a lista de funções disponíveis nos sistemas através das teclas Shift+F6 no Editor de Regras.

Existem diversas funções pré-definidas que facilitam a vida do desenvolvedor de Regras. Por exemplo, imagine que queira validar se determinada sequência de números é um CNPJ válido ou não. Normalmente teríamos de criar uma rotina para fazer esta validação. No entanto, existe a função chamada CNPJValido que já faz exatamente isto.

Este é apenas um exemplo, pois existe uma grande gama de funções disponíveis, para os mais diversos objetivos, como por exemplo: CarregaSaldoProjeto, ConverteParaMaiusculo, CriaNovoServico, CriaNovoProduto, ExtensoMoeda, VerificaDiaUtil, CarregaSaldoContábil, EnviarEmail dentre outras muitas.

### *Funções definidas pelo usuário*

Este conteúdo será apresentado no treinamento de Regras LSP - Intermediário.

### *Funções do Editor de Regras*

O Editor de Regras já possui várias funções declaradas internamente para auxiliar no desenvolvimento das regras. No arquivo Ferramentas de Apoio (arquivo de ajuda contendo o manual do usuário) encontra-se a documentação completa de como utilizá-las. Essas funções podem ser divididas em grupos, conforme listadas a seguir:

### *Funções específicas do Gerador de Relatórios*

Este conjunto de funções é específico para utilização no Gerador de relatórios e será abordado em outro treinamento. A documentação completa sobre essas funções pode ser encontrada em: Ferramentas de Apoio/Linguagem Senior de Programação/Funções Gerais/Específicas do Gerador de Relatórios.

### *Funções SQL das regras*

Com essas funções é possível acessar e manipular dados provenientes do banco de dados. Esse conjunto de funções também possui um treinamento dedicado somente a sua utilização. A documentação completa sobre essas funções pode ser encontrada em: Ferramentas de Apoio/Linguagem Senior de Programação/Funções Gerais/Funções SQL em regra.

### *Funções para manipulação de usuários e grupos*

Com as funções de manipulação de usuários pode-se fazer praticamente qualquer ação como, criar usuários e grupos, alterar informações, bloquear usuário e muito mais. Possibilita efetuar as principais ações do SGU, porém através de regras. A documentação completa sobre

essas funções pode ser encontrada em: Ferramentas de Apoio/Linguagem Senior de Programação/Funções Gerais/Para Manipulação de Usuários e Grupos.

### *Funções das regras em geral*

Funções de propósitos gerais, várias delas foram utilizadas em exemplos nessa apostila. A documentação completa sobre essas funções pode ser encontrada em: Ferramentas de Apoio/Linguagem Senior de Programação/Funções Gerais/Regras em Geral.

### *Funções para manipulação de arquivos*

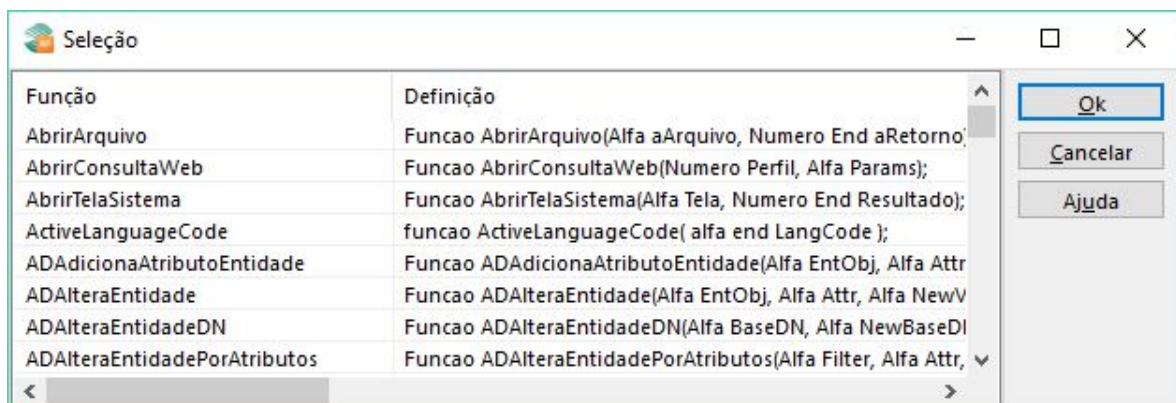
Conjunto de funções que permitem abrir, ler, manipular e gravar arquivos, tanto no formato texto como no formato binário. A documentação completa sobre essas funções pode ser encontrada em: Ferramentas de Apoio/Linguagem Senior de Programação/Funções Gerais/Para Manipular Arquivos Texto.

### *Funções para requisições HTTP*

Essas funções possibilitam a execução de requisições HTTP, oferecendo suporte à utilização de servidor proxy e requisições sobre SSL, permitindo que sejam acessados sites da WEB que utilizem HTTPS. A documentação completa sobre essas funções pode ser encontrada em: Ferramentas de Apoio/Linguagem Senior de Programação/Funções Gerais/ Funções para requisições HTTP.

### *Selecionar uma função existente*

É possível escolher uma função a partir da lista de funções. Para isso, no menu do Editor de Regras escolha: Seleção/Funções ou pressione as teclas Shift+F6. Vai abrir a tela “Seleção” contendo a lista das funções disponíveis. Essa lista de funções pode variar de acordo com o sistema que se está utilizando. Para facilitar a busca da função desejada é possível clicar sobre uma delas e digitar as letras iniciais da função procurada. A seleção ira para a primeira ocorrência que coincida com o que foi digitado. Abaixo está a imagem da tela de seleção de função:



Após selecionar a função desejada é só clicar em “Ok” para fazer com que a função seja adicionada a regra.

### Variáveis Disponibilizadas pelos Sistemas

Os sistemas Senior disponibilizam uma série de funções e variáveis que podem ser utilizadas no Editor de Regras. O processo para selecionar uma função de sistema é o mesmo descrito anteriormente. Para selecionar uma variável de sistema, acesse o menu: Seleção/Variáveis ou pressione as teclas Shift+F7.

Vai abrir a tela “Seleção” com todas as variáveis de sistemas disponíveis. Os passos para selecionar uma variável são os mesmos das funções. Abaixo segue a imagem da tela de seleção de variáveis.



The screenshot shows a window titled 'Seleção' with a table of system variables. The table has three columns: 'Nome', 'Tipo', and 'Ocorr...'. The variables listed are: AbaFgt (Numérico, 0), AbrTelSel (Cadeia de caracteres, 154), Acu\_SitEmp (Numérico, 0), AcuCal (Numérico, 9600), AcuTotCod1 (Numérico, 0), AcuTotCod2 (Numérico, 0), AcuTotDes1 (Cadeia de caracteres, 0), AcuTotDes2 (Cadeia de caracteres, 0), AcuTotFG1 (Cadeia de caracteres, 0), and AcuTotFG2 (Cadeia de caracteres, 0). To the right of the table are three buttons: 'Ok', 'Cancelar', and 'Ajuda'.

Nome	Tipo	Ocorr...
AbaFgt	Numérico	0
AbrTelSel	Cadeia de caracteres	154
Acu_SitEmp	Numérico	0
AcuCal	Numérico	9600
AcuTotCod1	Numérico	0
AcuTotCod2	Numérico	0
AcuTotDes1	Cadeia de caracteres	0
AcuTotDes2	Cadeia de caracteres	0
AcuTotFG1	Cadeia de caracteres	0
AcuTotFG2	Cadeia de caracteres	0



#### DICA

As variáveis de sistema se destacam das outras variáveis pela cor vermelha e não precisam ser declaradas.

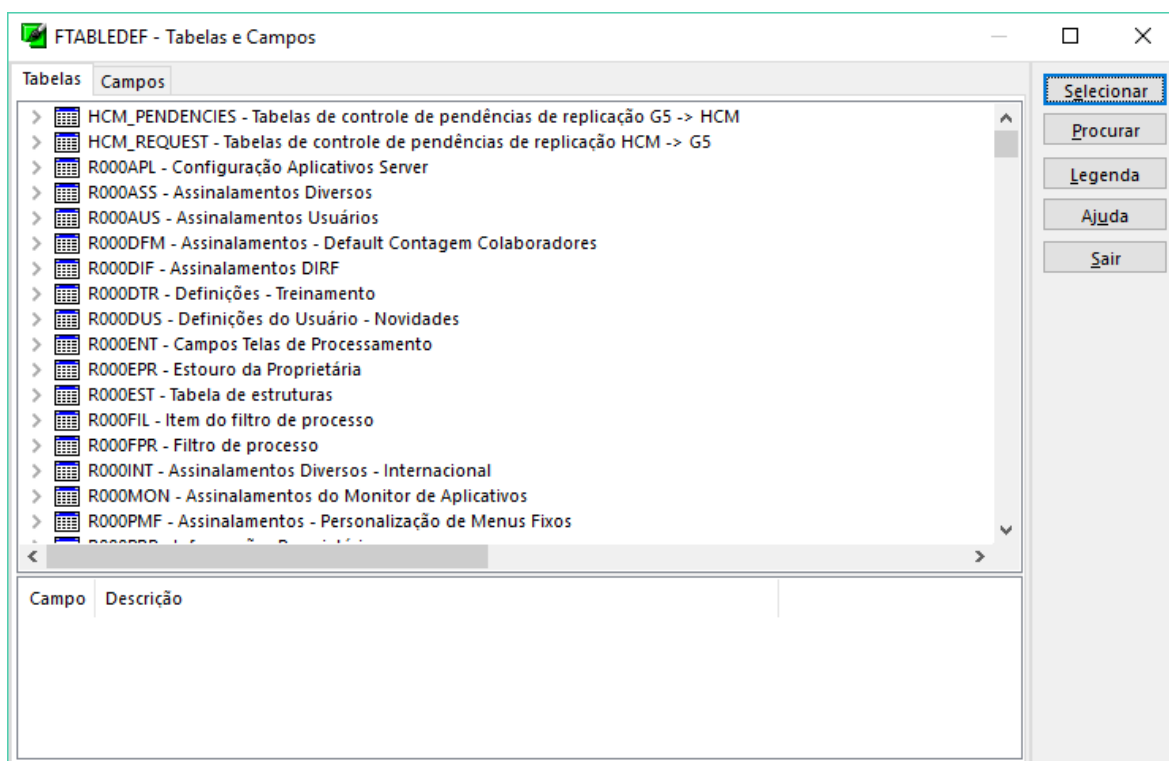


## Constantes

Constantes são informações que não têm seu valor alterado durante a execução da Regra, ou seja, é um valor não calculado. Por exemplo, a data 15/09/2012, o número 720 e o texto “Senior Sistemas” são constantes. O valor resultante de um cálculo ou de uma fórmula não é uma constante.

## Tabelas e Campos

Essa opção abre a tela “Tabelas e Campos” listando as tabelas e campos que podem ser utilizadas nas regras. Para abrir essa tela acesse o menu “Seleção/Tabelas e Campos” ou pressione as teclas Shift+F8. Vai abrir a tela “Tabelas e Campos” conforme figura a seguir:



A tela abre inicialmente com a guia “Tabelas” selecionada. Para selecionar um ou mais campos, primeiro deve ser selecionado a tabela aos quais os campos pertençam, em seguida devem ser selecionados os campos desejados entre aqueles que aparecerão na listagem abaixo da listagem das tabelas. O próximo passo é clicar no botão “Selecionar”. Isso fará com que todos os campos selecionados sejam adicionados a regra.



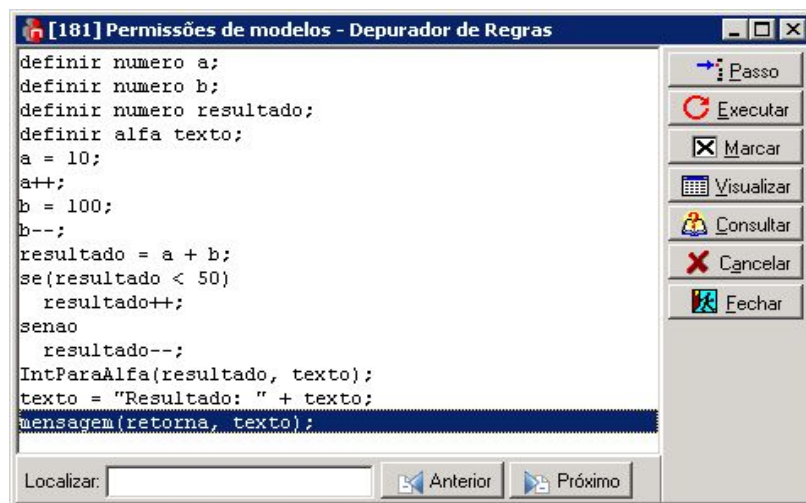
### DICA

Na guia “Campos” ao lado da guia “Tabelas” pode-se procurar por algum campo e suas ocorrências em uma ou mais tabelas, basta clicar no botão “Procurar” e digitar o nome do campo desejado.



## Depuração de Regras

Uma regra pode ser executada de duas formas: com e sem depuração. Uma regra sem depuração é executada sem nenhuma interrupção durante sua execução. Já uma regra com depuração, no início da sua execução abre-se uma tela com o código da regra sendo possível executar passo a passo cada instrução da regra, inclusive pode-se visualizar e alterar os conteúdos das variáveis. A depuração de uma regra pode ser ativada através do menu “Regra/Depurar Arquivo”, ou “Regra/Ativar Depuradores”, ou através das teclas de atalho Ctrl+F8 e Shift+F10 respectivamente. Habilitada a depuração da regra, ao executá-la será aberta a tela de depuração conforme imagem abaixo:



Na sequência será explicada cada opção disponível nessa tela:

**Passo:** Executa apenas uma instrução da regra por vez, permite que a regra seja executada passo a passo.

**Executar:** Executa toda a regra sem interrupções, ou até encontrar um ponto de parada.

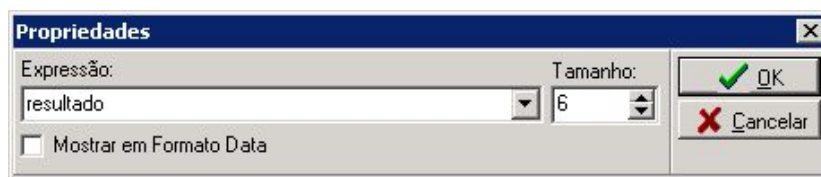
**Marcar:** Marca um ponto de parada, quando o compilador atingir essa instrução a execução é interrompida, permitindo assim que o usuário examine o contexto das variáveis.

**Visualizar:** Abre a tela “Visualização de Variáveis” conforme figura a seguir:



Essa tela permite Visualizar e alterar o conteúdo das variáveis enquanto executa a regra. Abaixo segue a explicação de cada opção:

**Adicionar:** Abre a tela “Propriedades” que permite adicionar uma variável da regra para inspecionar seu conteúdo, conforme figura a seguir:



Em “**Expressão**” informe o nome da variável ou campo de tabela ou ainda uma expressão como “a + b”. Em “**Tamanho**” informe a quantidade de casas decimais para variáveis numéricas. Para mostrar o resultado no formato de data, marque a opção “Mostrar em Formato Data”.

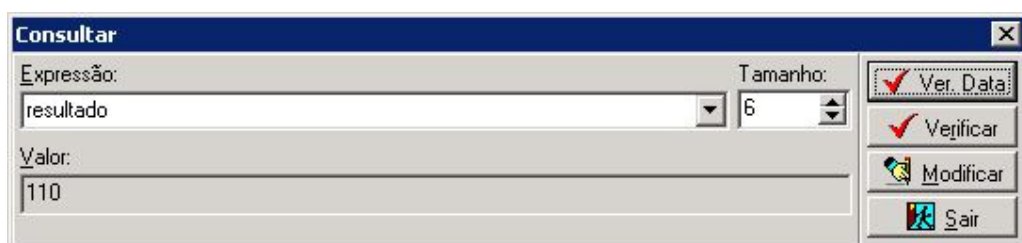
**Editar:** Também abre a tela “Propriedades”, porém já com os valores da variável selecionada. Altere o valor da variável e clique em “OK”.

**Excluir:** Exclui a variável selecionada da tela “Visualização de Variáveis”. **Ex. Todas:** Exclui todas as variáveis da tela “Visualização de Variáveis”.

**Sair:** Fecha a tela “Visualização de Variáveis”.

Na sequência segue as definições da tela “Depurador de Regras”:

**Consultar:** Essa tela tem por objetivo visualizar e alterar os valores de uma variável, ou campo de tabela ou expressão, porém não permite continuar a execução da regra enquanto a tela estiver aberta, Conforme figura a seguir:



Seu funcionamento é semelhante à tela “Propriedades”, em “Expressão” deve ser informado a expressão a ser analisada. O resultado da expressão será exibido no campo “Valor”. Abaixo seguem as explicações das opções dessa tela:

**Ver. Data:** Permite visualizar o conteúdo da expressão no formato data (apenas para os tipos numéricos).

**Verificar:** Visualiza o conteúdo da expressão no seu formato original.

**Modificar:** Pode-se informar um novo valor para a expressão no campo “Valor” e depois clicar no botão “Modificar”. Esse novo valor será atribuído a variável selecionada desse ponto em diante da regra.

**Sair:** Fecha a tela “Consultar”.

**Cancelar:** Cancela a depuração e a execução da regra.

**Fechar:** Fecha o depurador de regras, porém a regra não será cancelada, apenas a depuração.

**Localizar:** No canto inferior da tela de depuração existe uma opção para localizar texto dentro da regra que está sendo depurada. Basta digitar um texto qualquer e pressionar a tecla “Enter” ou clicar nos botões de navegação “Anterior” ou “Próximo”. Caso localizado, o texto será destacado dentro do código da regra. Essa localização não diferencia maiúsculas de minúsculas.



### AUTOATIVIDADE

Agora que você já explorou todos os materiais disponíveis vamos praticar a depuração das regras com o depurador Senior.



# CAPÍTULO 04

## Estrutura Básica de Cursor

### Objetivos Específicos

- Desenvolver um cursor básico.

## ESTRUTURA BÁSICA DE CURSOR

Na LSP, os cursores são recursos que permitem através de comandos SQL obter e manipular registros de uma ou mais tabelas do banco de dados.

A sintaxe para definir uma variável do tipo cursor é a seguinte: definir cursor <nome\_do\_cursor>;

Exemplo:

```
Definir Cursor Cur_R034FUN;
```

Os cursores possuem os seguintes comandos:

**SQL:** Usado para definir um comando SQL para o cursor.

**Sintaxe:** <nome\_do\_cursor>.SQL "<comando\_SQL>";

**Exemplo Gestão de Pessoas:**

```
Definir Cursor Cur_R034FUN;
```

```
Cur_R034FUN.SQL "SELECT NUMCAD \
                  FROM R034FUN \
                  WHERE R034FUN.NUMEMP = 1";
```

**Exemplo Gestão Empresarial:**

```
Definir Cursor Cur_E085CLI;
```

```
Cur_E085CLI.SQL "SELECT NUMCAD \
                  FROM R034FUN \
                  WHERE R034FUN.NUMEMP = 1";
```

**AbrirCursor :** Abre um cursor. Abrir um cursor executa o comando SQL no banco de dados. O banco de dados retorna os dados se houverem dados como resultado do comando select. A partir desse ponto em diante os dados retornados pelo cursor podem ser manipulados na regra.

**Sintaxe:**

<nome\_do\_cursor>.AbrirCursor( );

**Exemplo Gestão de Pessoas:**

```
Cur_R034FUN.AbrirCursor( );
```

**Exemplo Gestão Empresarial:**

```
Cur_E085CLI.AbrirCursor( );
```

**FecharCursor:** Fecha um cursor. A partir desse momento os dados retornados do select não estão mais disponíveis para manipulação através do cursor.

**Sintaxe:**

```
<nome_do_cursor>.FecharCursor( );
```

**Exemplo Gestão de Pessoas:**

```
Cur_R034FUN.FecharCursor( );
```

**Exemplo Gestão Empresarial:**

```
Cur_E085CLI.FecharCursor( );
```

**Proximo:** Posiciona no próximo registro do cursor e lê seu conteúdo.

**Sintaxe:**

```
<nome_do_cursor>.Proximo( );
```

**Exemplo Gestão de Pessoas:**

```
Cur_R034FUN.Proximo( );
```

**Exemplo Gestão Empresarial:**

```
Cur_E085CLI.Proximo( );
```

**Campos:** Os campos que um cursor terá dependerão do comando SQL atribuído a ele. Por exemplo, se um cursor tiver o comando “select \* from R034FUN” o cursor terá todos os campos da tabela de funcionários. Já se tiver o comando “select NomFun, DatAdm from R034FUN” terá apenas os campos NomFun e DatAdm.

Os campos pertencentes ao cursor podem ser acessados diretamente. Os campos são tipo "Somente Leitura", ou seja, não podem ser alterados.

**Sintaxe:**

```
<nome_do_cursor>.<nome_do_campo>
```

**Exemplo Gestão de Pessoas – Cursor simples utilizando valores fixos:**

```
Definir Cursor Cur_R034FUN;
Definir Alfa aNomFun;
Definir Data dDatAdm;
Definir Alfa aMensagem;
Definir Alfa aDatAdm;

aNomFun = "";
dDatAdm = 0;
aMensagem = "Não encontrou o Colaborador!!";
Cur_R034FUN.SQL "SELECT NOMFUN, DATADM FROM R034FUN \
                WHERE NUMEMP = 1 \
                AND TIPCOL = 1 \
```

```

        AND NUMCAD = 1";
Cur_R034FUN.AbrirCursor();
Se (Cur_R034FUN.Achou)
    Início
        aNomFun = Cur_R034FUN.NomFun; @Lendo o valor do campo NomFun do cursor.@
        dDatAdm = Cur_R034FUN.DatAdm; @Lendo o valor do campo DatAdm do cursor.@
        ConverteMascara (3,dDatAdm,aDatAdm,"DD/MM/YYYY");
        aMensagem = "Colaborador: " + aNomFun + ", Data de Admissão: " + aDatAdm;
    Fim;
Cur_R034FUN.FecharCursor();

Mensagem(Retorna,aMensagem);

```

### Exemplo Gestão Empresarial – Cursor simples utilizando valores fixos:

```

Definir Cursor Cur_E085CLI;
Definir Alfa    aNomCli;
Definir Alfa    aUF;
Definir Alfa    aMensagem;

aNomCli = "";
aUF = "";
aMensagem = "Não encontrou o Cliente!!";

Cur_E085CLI.SQL "SELECT NOMCLI, SIGUFS \
                  FROM E085CLI          \
                  WHERE NUMEMP = 1      \
                  AND TIPCOL = 1        \
                  AND NUMCAD = 1";

Cur_E085CLI.AbrirCursor();
Se (Cur_E085CLI.Achou)
    Início
        aNomCli = Cur_E085CLI.NomCli; @Lendo o valor do campo NomCli do cursor.@
        aUF = Cur_E085CLI.SigUfs; @Lendo o valor do campo SigUfs do cursor.@
        aMensagem = "Cliente: " + aNomCli + ", Estado: " + aUF;
    Fim;
Cur_E085CLI.FecharCursor();

Mensagem(Retorna,aMensagem);

```

**Achou:** Retorna verdadeiro se achou alguma linha do cursor. Isso indica que o cursor retornou algum dado do banco de dados.

### Sintaxe:

<nome\_do\_cursor>.Achou

### Exemplo Gestão de Pessoas:

```

Se (Cur_R034FUN.Achou)
    Início
        aNome = Cur_R034FUN.NomFun;
        dData = Cur_R034FUN.DatAdm;
    Fim;

```

### Exemplo Gestão Empresarial:

```

Se (Cur_E085CLI.Achou)
    Início
        aNome = Cur_E085CLI.NomCli;
        aUF = Cur_E085CLI.SigUfs;
    Fim;

```



**NaoAchou:** Retorna verdadeiro se não achou nenhuma linha do cursor. Isso indica que nenhum dado foi retornado pelo banco de dados em resposta ao comando select atribuído ao cursor.

**Sintaxe:** <nome\_do\_cursor>.NaoAchou

Exemplo Gestão de Pessoas:

```
Se (Cur_R034FUN.NaoAchou)
  Inicio
  Mensagem(Retorna, "Nenhum dado foi encontrado!");
  Fim;
```

Exemplo Gestão Empresarial:

```
Se (Cur_E085CLI.NaoAchou)
  Inicio
  Mensagem(Retorna, "Nenhum dado foi encontrado!");
  Fim;
```

Quando um comando SQL for muito extenso, pode-se utilizar a barra (\) para efetuar a quebra de linha. Isso facilita a visualização do comando na tela.

**Exemplo Gestão de Pessoas:**

```
Cur_R034FUN.SQL "SELECT NUMEMP, NUMCAD, NOMFUN, DATADM \
                FROM R034FUN \
                WHERE NUMEMP = (SELECT MAX(NUMEMP) \
                                FROM R034FUN) \
                AND DATADM = (SELECT MIN(DATADM) \
                              FROM R034FUN \
                              WHERE NUMEMP = (SELECT MAX(NUMEMP) \
                                                FROM R034FUN)) ";
```

É possível a inclusão de variáveis diretamente dentro do comando SQL, para isso é necessário utilizar dois pontos (:) antes da variável.



### IMPORTANTE

Não pode ser utilizado espaço após os dois pontos (:).

**Exemplo Gestão de Pessoas – Cursor Simples utilizando variáveis externas:**

```

Definir Cursor Cur_R034FUN;
Definir Alfa aNomFun;
Definir Data dDatAdm;
Definir Alfa aMensagem;
Definir Alfa aDatAdm;

aNomFun = "";
dDatAdm = 0;
aMensagem = "Não encontrou o Colaborador!!";
nNumEmp = 1;
nTipCol = 1;
nNumCad = 10;

Cur_R034FUN.SQL "SELECT NOMFUN, DATADM FROM R034FUN \
                WHERE NUMEMP = :nNumEmp\
                AND TIPCOL = :nTipCol\
                AND NUMCAD = :nNumCad";

Cur_R034FUN.AbrirCursor();
Se (Cur_R034FUN.Achou)
  Inicio
    aNomFun = Cur_R034FUN.NomFun; @Lendo o valor do campo NomFun do cursor.@
    dDatAdm = Cur_R034FUN.DatAdm; @Lendo o valor do campo DatAdm do cursor.@
    ConverteMascara (3,dDatAdm,aDatAdm,"DD/MM/YYYY");
    aMensagem = "Colaborador: " + aNomFun + ", Data de Admissão: " + aDatAdm;
  Fim;
Cur_R034FUN.FecharCursor();
Mensagem(Retorna,aMensagem);

```

**Exemplo Gestão Empresarial – Cursor Simples utilizando variáveis externas:**

```

Definir Cursor Cur_E085CLI;
Definir Alfa aNomCli;
Definir Alfa aUF;
Definir Alfa aMensagem;

aNomCli = "";
aUF = "";
aMensagem = "Não encontrou o Cliente!!";
nCodCli = 2;

Cur_E085CLI.SQL "SELECT NOMCLI, SIGUFS \
                FROM E085CLI \
                WHERE CODCLI = :nCodCli";

Cur_E085CLI.AbrirCursor();
Se (Cur_E085CLI.Achou)
  Inicio
    aNomCli = Cur_E085CLI.NomCli; @Lendo o valor do campo NomCli do cursor.@
    auf = Cur_E085CLI.SigUfs; @Lendo o valor do campo SigUfs do cursor.@
    aMensagem = "Cliente: " + aNomCli + ", Estado: " + aUF;
  Fim;
Cur_E085CLI.FecharCursor();
Mensagem(Retorna,aMensagem);

```

## Estrutura Básica

Sintaxe de um cursor com retorno de apenas um registro:

```
Definir Cursor Cur_Nome;  
  
Cur_Nome.SQL "<comando select>";  
Cur_Nome.AbrirCursor();  
Se      (Cur_Nome.Achou)  
        Inicio  
                                < Bloco de Comando >  
        Fim;  
Cur_Nome.FecharCursor();
```

Sintaxe de um cursor com retorno de vários registros:

```
Definir Cursor Cur_Nome;  
  
Cur_Nome.SQL "<comando select>";  
Cur_Nome.AbrirCursor();  
Enquanto (Cur_Nome.Achou)  
        Inicio  
                                < Bloco de Comando >  
                                Cur_Nome.Proximo()  
        Fim;  
Cur_Nome.FecharCursor();
```



### AUTOATIVIDADE – GESTÃO DE PESSOAS

Agora que você já explorou todos os materiais disponíveis vamos praticar a criação de cursores no sistema Gestão de Pessoas:

- Criar um cursor que retorne em mensagem o código, descrição e UF da cidade.



### AUTOATIVIDADE – GESTÃO EMPRESARIAL – ERP

Agora que você já explorou todos os materiais disponíveis vamos praticar a criação de cursores no sistema Gestão Empresarial:

- Criar um cursor que retorne em mensagem o CEP inicial, descrição e UF da cidade.

# CAPÍTULO 05

## Identificadores de Regras

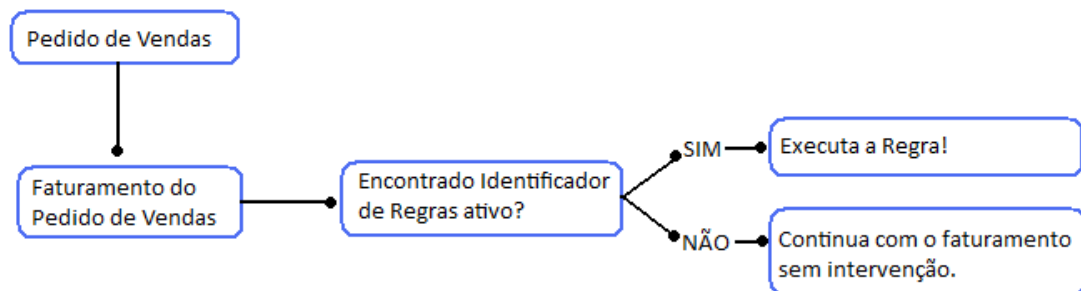
### Objetivos Específicos

- Desenvolver Regras com Uso de Identificadores de Regras.

## IDENTIFICADORES DE REGRAS

Uma coisa é desenvolver a Regra; outra é executá-la. Em que momento sua Regra será executada? Como fazer que ela seja executada, por exemplo, na preparação do faturamento de Pedidos de Venda? Ou ainda no momento em que o usuário está cadastrando uma nova Família de Produtos, para que sejam feitas algumas consistências que a empresa considera importantes?

É então que entram os Identificadores de Regra. Eles são como pontos de verificação dentro do ERP. Sempre que o ERP encontrar um destes pontos, verá se tem o Identificador de Regras ativo e, se estiver ativo, se há uma Regra associada cadastrada. Caso houver, chamará e executará a Regra. Na Figura a seguir temos um fluxograma exemplo do comportamento explicado.



Identificadores de Regras não são criados por usuários e consultores. Eles são desenvolvidos e definidos pela fábrica da Senior. Consultores e usuários criam apenas as Regras, que interagem ou são chamadas por estes Identificadores.

### Identificadores de Regras - Cadastramento e ativação

Os Identificadores podem estar ligados ou não a Transações. Se o Identificador estiver ativo e ligado a uma ou mais Transações, isto significa que somente será executado quando estas Transações forem utilizadas no processo.

A documentação de cada Identificador indica se terá ou não uma Transação relacionada. Alguns Identificadores precisam ter uma Transação associada, outros permitem estar ligados a Transações mas de forma opcional, ou seja, se não for informada uma Transação no cadastramento do Identificador, todas as Transações interagirão com o mesmo. Consulte sempre o Help do Identificador e faça testes usando diversas configurações para avaliar o comportamento do Identificador que pretende usar.

Não é permitido excluir Identificadores uma vez que foram cadastrados, apenas inativá-los. Um log está disponível para auxiliar a identificar o histórico de cadastramento/ativação/inativação do Identificador.

Existem Identificadores de Regras que não tem Regra associada. Por exemplo, o Identificador de Regras **CRE-301AVPNC01**, que quando ativo exibe ao invés do apelido do Fornecedor o nome do Fornecedor no grid de Títulos de Contas a Receber. Não há realmente uma Regra que precisa ser executada neste caso, pois o Identificador já tem as funções necessárias embutidas.

Outros Identificadores de Regras necessitam obrigatoriamente ter Regra associada. Por exemplo, o Identificador de Regras **COM-000ALSTR01** permite alterar a sugestão do Código da Situação Tributária e ICMS do Item sempre que é feita a sugestão do mesmo em Contratos e Notas Fiscais de Compra e Venda. Mas alterar para o que? Bem, precisa ter uma Regra associada a este Identificador, definindo qual será essa alteração.

Os Identificadores estão separados por módulos (COM, CPR, CPA, FIN, VEN, etc). Sua nomenclatura é definida pela fábrica, e obedece a seguinte estrutura: **000COMIS00**

- As 3 primeiras posições são numéricas e identificam a rotina a qual se aplica.

**Exemplo:** **120** indica Pedidos. Quando é utilizado em várias rotinas utiliza-se 000.

- As 5 próximas posições identificam o que irá tratar.

**Exemplo:** **COMIS** indica Comissões.

- As 2 próximas posições para uma numeração seqüencial, pois podem existir várias regras para tratar o mesmo assunto.

**Exemplo:** **120COMIS02**.

É possível consultar todos os Identificadores ativos no sistema e seu histórico de uso através do atalho Ctrl+i.

### Identificadores de Regras - Variáveis disponíveis

Caso o Identificador de Regra seja um do tipo que chama e executa uma Regra, normalmente terá **variáveis de sistema** disponíveis tanto para **consulta** quanto para **retorno**.

Variáveis que retornam valor permitem que a Regra altere o seu conteúdo em “tempo de execução”, retornando esse valor para o ERP.

Por exemplo, o Identificador de Regras **COM-140PRECO02** deve estar associado a uma Regra e tem como função alterar o **preço unitário** e o **percentual de desconto** do item do pedido de serviço que é gravado na Nota Fiscal, quando este é transferido para a mesma.

Isto quer dizer que a Regra associada ao Identificador poderá manipular o **preço unitário (VSPREUNI)** e/ou o **percentual de desconto (VSPERDSC)** dos Itens na preparação da NF.

Ou seja, usando esse Identificador associado com uma Regra, poderemos alterar o valor de cada Item de Serviço para mais 5%, por exemplo. De que outra forma isso poderia ser feito? Manualmente talvez, mas estaria sujeito a intervenção de usuário, com possibilidade de erros, não é verdade? A Regra é uma forma mais segura de fazer isso.

Mas suponhamos que queiramos ativar esse reajuste de 5% apenas se o Código do Serviço for 3501 e 3503. Neste caso, temos as **Variáveis de Sistema** disponibilizadas pelo Identificador de Regras. Se observarmos o *Help* deste Identificador, perceberemos que temos a Variável **VSCODSER**, que nos informa qual o Código do Serviço está sofrendo intervenção do



Identificador no momento da execução. Com isto, conseguiremos fazer o reajuste de 5% apenas nos Serviços desejados.

A lógica da Regra ficaria assim para este exemplo:

Se **VsCodSer** = 3501 ou 3503  
Então **VsPreUni** = **VsPreUni** + 5%

## Identificadores de Regras – Exemplo de Regra

Será desenvolvido um exemplo de Regra ligado a um Identificador de Regra. Imaginemos que nos é solicitado resolver a seguinte situação:

*A empresa deseja **cobrar os impostos** originados dos **Serviços** dos seus clientes. Para isso, ela combina com seus clientes de que praticará tal cobrança, mas isso deve ser controlado a nível de **Pedido de Venda**, pois alguns Pedidos de Venda de clientes podem ter a prática, outros Pedidos não.*

*A **alíquota** que será cobrada é de 5,2%. Na prática isto quer dizer o seguinte: um Serviço que custa R\$ 1.000,00, se tiver sido acordado com o cliente a cobrança dos impostos no Pedido de Venda, deve ser faturado (**emissão da NF**) em R\$ 1.052,00.*

Antes de mais nada, é preciso analisar se há alguma solução nativa para no sistema ERP. Tendo conhecimento que não há tal solução no produto padrão, será necessário partir para o desenho da análise. Uma vez validada a análise junto ao cliente, será iniciado o desenvolvimento da solução.

- 1- Será criada uma tabela de usuários para armazenar a alíquota. Sim, porque hoje é 5,2%, mas se passar a ser outro valor mês que vem, o cliente deve ter condições de alterar sem interferência no código-fonte da Regra.
  - a. Chamaremos essa tabela de **USU\_TAliCob (Alíquota de Cobrança)**.
  - b. Criar os campos:
    - **USU\_SeqTab** (number, ZZZ9, campo chave): toda tabela precisa de um campo chave, e este será o nosso, mesmo que a princípio haverá somente um registro na tabela.
    - **USU\_AliCob** (number, ZZ,ZZ9): campo para armazenar a alíquota praticada.
    - **USU\_ObsCob** (string, 50 posições): campo para observação sobre a alíquota praticada.

**OBS:** uma forma de melhorar a análise seria permitir aos usuários irem acrescentando novas alíquotas e a data em que aconteceu a manutenção e a Regra sempre usar a alíquota mais recente, mas não vamos sofisticar a este ponto nossa Regra.
- 2- Criar uma tela de interface (SGI) para alimentar a tabela criada para manutenção de alíquota.
- 3- Criar na tabela **E120PED** (Pedidos de Venda) o campo **USU\_CobAli** (string, 1 posição, enumeração *LSimNao*), para definir no **Pedido de Venda** se haverá a cobrança da alíquota ou não (S/N).

- 4- Desenvolver a Regra (*código 25-Cobrança de Impostos em Pedidos de Venda*), respeitando as boas práticas que vimos até aqui.
- 5- A Regra criada estará associada ao Identificador de Regras **VEN 140PRECO02**, que precisará estar ativo. Vide *Help* do Identificador para conhecer seu funcionamento.
- 6- Uma das soluções possíveis para esta Regra é listada abaixo. Experimente criar a sua solução, com variações:

```
@=====
@ Cliente: Senior Sistemas                                     @
@ Rotina: Cobrança Impostos em Pedidos de Venda              @
@ Código da Regra: 25                                         @
@ Desenvolvedor: Instrutor GCS                               @
@ Data criação: 01/10/2009                                    @
@ Data última alteração:                                       @
@ Histórico alteração:                                         @
@=====

@Definição de variáveis@
Definir Alfa aDefineCobAliq; @ Indicativo carregado a partir do Serviço de Ped.Venda se haverá ou não Cob.Alíquota@
Definir Alfa VSCODSER; @ Variável de sistema do Código de Serviço @
Definir Numero VsNumPed; @ Variável de sistema de Numero do Pedido @
Definir Numero VSPreUni; @ Variável de sistema do Preço Unitário @
Definir Numero VSCODEMP; @ Variável de sistema do Código Empresa @
Definir Numero VsCodFil; @ Variável de sistema do Código Filial @
Definir Numero nPercAliq; @ Armazena percentual da alíquota carregado a partir da tabela USU_TAliCob @

INICIO
@Cursor para carregar o percentual da alíquota de cobrança.@
Definir Cursor Cur_PercAliq;
Cur_PercAliq.sql "SELECT USU_AliCob\
FROM USU_TAliCob\
WHERE USU_SeqTab = 1";
Cur_PercAliq.AbrirCursor();
nPercAliq = Cur_PercAliq.USU_AliCob; @Seta para a variável o valor de usu_AliCob carregado no Cursor@
Cur_PercAliq.FecharCursor();

@Cursor para verificar se o Pedido de Venda indica que haverá Cobrança da Alíquota (E120PED.USU_CCIPV = 'S') @
Definir Cursor Cur_PedidoServAliq; @Cursor para tratar quando no Pedido de Venda indica que haverá Cobrança CCI@
Cur_PedidoServAliq.sql "SELECT USU_CobAli\
FROM E120PED\
WHERE CodEmp = :VsCodEmp\
AND codfil = :VsCodFil\
AND numped = :VsNumPed\
AND Usu_CobAli = 'S'";
Cur_PedidoServAliq.AbrirCursor();

@Se o Pedido de Vendas está definido como cobrança de alíquota = 'N', encerra a regra@
Se (Cur_PedidoServAliq.NaoAchou)
Início
Cur_PedidoServAliq.FecharCursor();
VaPara Encerrar;
Fim;

@Se o Pedido de Vendas está definido como cobrança de alíquota = 'S', entra na rotina abaixo@
Se (Cur_PedidoServAliq.Achou)
Início
VsPreUni = VsPreUni + (VsPreUni*nPercAliq/100); @Aplica na variável de sistema de Preço Unitário do Serviço o fator da
alíquota@
Cur_PedidoServAliq.FecharCursor();
Fim;

Encerrar: @Rotulo para encerrar a regra@
Cancel;

FIM;
```

Rua São Paulo, 825 - Victor Konder  
Blumenau - SC - CEP: 89012-001  
Telefone: +55 47 3039-3580  
[universidade.corporativa@senior.com.br](mailto:universidade.corporativa@senior.com.br)  
[www.senior.com.br](http://www.senior.com.br)



**SENIOR**  
Universidade  
Corporativa