Ferramentas – Regras LSP - Intermediário

SENIOR
Universidade
Corporativa

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

Apresentação

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

Sumário

CAPÍTULO 01

Cursores Avançados

Objetivos Específicos

- Desenvolver cursores avançados no sistema.

Cursores

Revisão da Estrutura de Cursores

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>";

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( );

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( );

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

Sintaxe: <nome_do_cursor>.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>

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

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

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.

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

Cursores com Recuperação de Dados de Mais de Uma Tabela

Um cursor permite que sejam recuperados dados de mais de uma tabela no mesmo comando SQL. No exemplo abaixo, além do nome de um funcionário, estamos recuperando também o nome da empresa e a descrição de sua nacionalidade. Essas duas últimas informações estão em outras tabelas, porém ligadas na tabela de funcionários pelas chaves estrangeiras.

Exemplo Gestão de Pessoas – Cursor com uso de mais de uma tabela:

```
Definir Cursor Cur_R034FUN;
Definir Data   dData;
Definir Numero nNumEmp;
Definir Numero nTipCol;
Definir Alfa   aMensagem;
Definir Alfa   aEnter; @ Nova Linha @

nNumEmp = 1;
nTipCol = 1;
MontaData(01, 01, 2010, dData);

Cur_R034FUN.SQL "SELECT R030EMP.NOMEMP, R034FUN.NOMFUN, R023NAC.DESNAC \
                   FROM R034FUN, R030EMP, R023NAC \
                  WHERE R030EMP.NUMEMP =  R034FUN.NUMEMP \
                    AND R023NAC.CODNAC =  R034FUN.CODNAC \
                    AND R034FUN.NUMEMP =  :nNumEmp \
                    AND R034FUN.TIPCOL =  :nTipCol \
                    AND R034FUN.DATADM >= :dData";
Cur_R034FUN.AbrirCursor();
Se (Cur_R034FUN.Achou)
  Inicio
  CaracterParaAlfa(10, aEnter);
  aMensagem = "Empresa = " + Cur_R034FUN.NomEmp + aEnter + "Funcionário = " +
              Cur_R034FUN.NomFun + aEnter + "Nacionalidade = " +
              Cur_R034FUN.DesNac;
  Mensagem(Retorna, aMensagem);
  Fim;
Senao
  Inicio
  Mensagem(Retorna, "Nenhum registro foi encontrado!");
```

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

Exemplo Gestão Empresarial – Cursor com uso de mais de uma tabela:

```
Definir Cursor Cur_E075PRO;
Definir Alfa aCodPro;
Definir Alfa aDesPro;
Definir Alfa aNomEmp;
Definir Alfa aDesFam;
Definir Alfa aEnter; @ Nova Linha@
Definir Alfa aMensagem;
aMensagem = "";
nCodEmp = 1;
aCodPro = "1101";
CaracterParaAlfa(10, aEnter);
Cur_E075PRO.SQL "SELECT E075PRO.CODPRO, E075PRO.DESPRO, E070EMP.NOMEMP, \
                       E012FAM.DESFAM \
                  FROM E075PRO, E070EMP, E012FAM    \
                 WHERE E075PRO.CODEMP = E070EMP.CODEMP \
                   AND E075PRO.CODEMP = E012FAM.CODEMP \
                   AND E075PRO.CODFAM = E012FAM.CODFAM \
                   AND E075PRO.CODEMP = :nCodEmp \
                   AND E075PRO.CODPRO = :aCodPro";
Cur_E075PRO.AbrirCursor();
Se (Cur_E075PRO.Achou)
   Inicio
   aDesPro = Cur_E075PRO.DesPro;
   aCodPro = Cur_E075PRO.CodPro;
   aNomEmp = Cur_E075PRO.NomEmp;
   aDesFam = Cur_E075PRO.DesFam;
   aMensagem = "Produto: " + aCodPro + aEnter + "Nome do Produto: " +
               aDesPro + aEnter + "Empresa: " + aNomEmp + aEnter +
               "Família: " + aDesFam;
   Mensagem(Retorna,aMensagem);
   Fim;
Senao
   Mensagem(Retorna,"Produto não Encontrado!!!");
Cur_E075PRO.FecharCursor();
```

Exemplo Gestão de Pessoas – Cursor com Laço de Repetição:

```
Definir Cursor Cur_R074CID;
Definir Alfa aNomCid;
Definir Alfa aCodCid;
Definir Alfa aUF;
Definir Alfa aMensagem;
Definir Alfa aEnter;
aUF       = "AC";
aNomCid   = "";
aCodCid   = "";
aMensagem = "";
RetornaAscII(13,aEnter);
Cur_R074CID.SQL "SELECT CODCID, NOMCID FROM R074CID WHERE ESTCID = :aUF";
Cur_R074CID.AbrirCursor();
Enquanto (Cur_R074CID.Achou)
    Inicio
    nCodCid = Cur_R074CID.CodCid;
```

```
    aNomCid = Cur_R074CID.NomCid;
    ConverteMascara(2,nCodCid,aCodCid,"99.99999");
    aMensagem = aMensagem + aCodCid + " - " + aNomCid + aEnter;
    Cur_R074CID.Proximo();
    Fim;
Cur_R074CID.FecharCursor();
Mensagem(Retorna,aMensagem);
```

Exemplo Gestão Empresarial – Cursor com Laço de Repetição:

```
Definir Cursor Cur_R034FUN; @ Cursor mestre @
Definir Cursor Cur_E008CEP;
Definir Alfa aNomCid;
Definir Alfa aCepIni;
Definir Alfa aUF;
Definir Alfa aMensagem;
Definir Alfa aEnter;
aUF       = "AC";
aNomCid   = "";
aCepIni   = "";
aMensagem = "";
RetornaAscII(13,aEnter);

Cur_E008CEP.SQL "SELECT CEPINI, NOMCID FROM E008CEP WHERE SIGUFS = :aUF";
Cur_E008CEP.AbrirCursor();
Enquanto (Cur_E008CEP.Achou)
    Inicio
    nCepIni = Cur_E008CEP.CepIni;
    aNomCid = Cur_E008CEP.NomCid;
    ConverteMascara(2,nCepIni,aCepIni,"99999-999");
    aMensagem = aMensagem + aCepIni + " - " + aNomCid + aEnter;
    Cur_E008CEP.Proximo();
    Fim;
Cur_E008CEP.FecharCursor();
Mensagem(Retorna,aMensagem);
```

Cursores aninhados: È possível termos cursores aninhados. Um cursor aninhado é útil quando para cada registro retornado por um cursor mais externo queremos fazer um novo select para buscar outros dados referentes a esse registro. Por exemplo, para cada colaborador, queremos obter os seus dependentes.

Exemplo Gestão de Pessoas – Cursores encadeados:

```
Definir Cursor Cur_R034FUN; @ Cursor mestre @
Definir Cursor Cur_R036DEP;  @ Cursor detalhe @
Definir Alfa   aTexto; @ Variável alfanumérica para concatenação @
Definir Alfa   aEnter;  @ Nova Linha @
Definir Numero nNumEmp;
Definir Numero nTipCol;
Definir Numero nNumCad;

CaracterParaAlfa(10, aEnter); @ Caracter especial para quebra de linha @
aTexto = "";

@ Cursor principal para ler os Colaboradores @
Cur_R034FUN.SQL "SELECT R034FUN.NUMEMP, R034FUN.TIPCOL, R034FUN.NUMCAD,\
                       R034FUN.NOMFUN \
                  FROM R034FUN \
              ORDER BY R034FUN.NUMEMP, R034FUN.TIPCOL, R034FUN.NUMCAD";
Cur_R034FUN.AbrirCursor();
Enquanto(Cur_R034FUN.Achou)
    Inicio
```

```
    nNumEmp = Cur_R034FUN.NumEmp;
    nTipCol = Cur_R034FUN.TipCol;
    nNumCad = Cur_R034FUN.NumCad;
    aEnter = aEnter + "Dependentes do colaborador: " + Cur_R034FUN.NomFun +
             aEnter;
    @ Cursor secundário busca registros (dependentes) para @
    @ cada registro do cursor principal (funcionários)     @
    Cur_R036DEP.SQL "SELECT R036DEP.NOMDEP FROM R036DEP \
                      WHERE R036DEP.NUMEMP = :nNumEmp \
                        AND R036DEP.TIPCOL = :nTipCol \
                        AND R036DEP.NUMCAD = :nNumCad \
                   ORDER BY R036DEP.NOMDEP";
    Cur_R036DEP.AbrirCursor();
    Enquanto(Cur_R036DEP.Achou)
        Inicio
        aTexto = aTexto + Cur_R036DEP.NomDep + aEnter;
        Cur_R036DEP.Proximo();
        Fim;
    Cur_R036DEP.FecharCursor();
    aTexto = aTexto + "-------------------------" + aEnter;
    Cur_R034FUN.Proximo();
    Fim;
Cur_R034FUN.FecharCursor();

Se  (aTexto = "")
    Inicio
    aTexto = "Nenhum registro encontrado!";
    Fim;
Mensagem(Retorna, aTexto);
```

Exemplo Gestão Empresarial – Cursores encadeados:

```
Definir Cursor Cur_E083ORI;
Definir Cursor Cur_E012FAM;
Definir Alfa aCodOri;
Definir Alfa aDesOri;
Definir Alfa aNomEmp;
Definir Alfa aCodFam;
Definir Alfa aDesFam;
Definir Alfa aEnter;
Definir Alfa aMensagem;
aMensagem = "";
CaracterParaAlfa (10, aEnter); @ Caracter especial para quebra de linha @
    @ Cursor principal para ler as Origem de Produto @
Cur_E083ORI.SQL "SELECT CODEMP, CODORI, DESORI FROM E083ORI";
Cur_E083ORI.AbrirCursor();
Enquanto (Cur_E083ORI.Achou)
    Inicio
    nCodEmp = Cur_E083ORI.CodEmp;
    aCodOri = Cur_E083ORI.CodOri;
    aDesOri = Cur_E083ORI.DesOri;
    aMensagem = aMensagem + "Origem :" + aCodOri + " - " + aDesOri  +
                ":" + aEnter;
  @ Cursor secundário para buscar registros (famílias) para @
  @ cada registro do cursor principal (origem)              @
    Cur_E012FAM.SQL "SELECT CODFAM, DESFAM FROM E012FAM    \
                      WHERE CODEMP = :nCodEmp \
                        AND CODORI = :aCodOri";
    Cur_E012FAM.AbrirCursor();
    nEntrou = 0;
    Enquanto (Cur_E012FAM.Achou)
        Inicio
```

```
        aCodFam = Cur_E012FAM.CodFam;
        aDesFam = Cur_E012FAM.DesFam;
        Se  (nEntrou = 0)
            Inicio
            nEntrou = 1;
            aMensagem = aMensagem + " - Família(s):  " + aCodFam + " - " +
                        aDesFam +  aEnter;
            Fim;
        Senao
            Inicio
            aMensagem = aMensagem + "                " + aCodFam + " - " +
                        aDesFam +  aEnter;
            Fim;
        Cur_E012FAM.Proximo();
        Fim;
    Cur_E012FAM.FecharCursor();
    aMensagem = aMensagem +  "-------------------------------" + aEnter;
    Cur_E083ORI.Proximo();
    Fim;
Cur_E083ORI.FecharCursor();
Se  (aMensagem <> "")
    Mensagem(Retorna,aMensagem);
Senao
    Mensagem(Retorna,"Nenhuma Origem foi Cadastrada!!!");
```

Macro __inserir nos Cursores: Essa macro permite inserir uma cadeia de caracteres no cursor. Serve para inserir um fragmento de comando SQL dentro de um comando SQL já existente em um cursor em tempo de execução da regra.

Sintaxe: __Inserir(: <nome_variável>)

Onde <nome_variável> é o nome de uma variável alfanumérica definida dentro da regra.

Considerações:

- Este comando é uma macro e deve ser usado dentro do comando SQL do cursor. Somente neste caso ele será reconhecido. Essa macro permite a inserção de valores no comando SQL do cursor em tempo de execução da regra;
- Entre os parêntesis deve ser inserido o nome de uma variável precedido por dois pontos. Se for necessário inserir valores constantes, o comando __Inserir não deve ser usado;
- A variável deve estar declarada dentro da regra e deve ser do tipo ALFA (alfanumérico). Variáveis não declaradas na regra ou que não sejam do tipo ALFA ocasionarão um erro na execução da regra;
- Os valores das variáveis a serem incluídas não devem conter o comando __Inserir. Isto pode ocasionar recursividade no tratamento do comando e instabilidade da regra. Tentativas neste sentido ocasionarão um erro na execução do modelo;
- O comando não deve ser usado na lista de campos do comando SQL. Caso isto for feito a regra não poderá reconhecer os campos para a compilação. A inserção do valor da variável dentro do comando SQL somente pode ser feito durante a execução da regra.

- Faz-se necessário a otimização do uso deste comando. O uso excessivo do mesmo poderá degradar a desempenho durante a execução da regra.

IMPORTANTE

Se a regra estiver usando a linguagem Senior SQL 2, esta macro deve estar no comando de maneira que os conectores não fiquem inválidos no SQL caso a macro seja substituída.

Exemplo 1 - Gestão de Pessoas:

Vamos trabalhar com a regra abaixo:

```
Definir Cursor Cur_R034FUN;
Definir Alfa SQLTipCol;
Definir Alfa SQLNumCad;

@ O retorno da função fictícia abaixo poderia ser algo como: @
@ "and TipCol = 1" ou ainda "and TipCol in (1,2)" @
ObtemWhereTipCol(SQLTipCol);


@ O retorno da função fictícia abaixo poderia ser algo como: @
@ "and NumCad = (select max(Numcad) from R034FUN @
@                 where NumEmp=1 and TipCol=1)"  @

ObtemWhereNumCad(SQLNumCad);

Cur_R034FUN.SQL "SELECT NUMEMP, TIPCOL, NUMCAD, NOMFUN, VALSAL \
                   FROM R034FUN \
                  WHERE NUMEMP = 1 \
                        __INSERIR(:SQLTIPCOL)\
                    AND TIPSEX = 'M' \
                        __INSERIR(:SQLNUMCAD)";
Cur_R034FUN.AbrirCursor();
Se  (Cur_R034FUN.Achou)
    Inicio
    ...
    ...
    Fim;
Cur_R034FUN.FecharCursor();
```

O comando SQL contido no cursor acima funciona bem em Senior SQL 2. Isto acontece porque a macro __inserir está disposta de maneira que seja opcional, ou seja, podem ser retirados sem que o comando SQL seja prejudicado.

Exemplo 2 - Vetorh:

```
Cur_R034FUN.SQL "SELECT NUMEMP, TIPCOL, NUMCAD, NOMFUN, VALSAL \
                   FROM R034FUN \
                  WHERE NUMEMP = 1 \
                    AND TIPCOL = __INSERIR(:SQLTIPCOL) \
                    AND NUMCAD = __INSERIR(:SQLNUMCAD)";
```

Neste caso a macro __inserir está sendo comparada com o valor do campo e se a macro for retirada não funcionará.

Exemplo 3 - Gestão de Pessoas:

```
Cur_R034FUN.SQL "SELECT NUMEMP, TIPCOL, NUMCAD, NOMFUN, VALSAL \
                   FROM R034FUN \
                  WHERE NUMEMP = 1 \
                    AND __INSERIR(:SQLTIPCOL) \
                    AND __INSERIR(:SQLNUMCAD) \
                    AND TIPSEX = 'M'";
```

Neste caso existem os conectores entre a expressão do NumEmp e entre as macros. Da mesma forma se a macro for retirada o comando ficará incompleto e não funcionará.

Para um melhor entendimento de como a macro deve ser usada, pode-se seguir a seguinte regra

IMPORTANTE

Se a macro puder ser removida e o comando ficar sem erros de sintaxe, a macro pode ser utilizada. Caso contrário, não!

Exemplo 4 - Gestão de Pessoas:

```
Cur_R034FUN.SQL "SELECT NUMEMP, TIPCOL, NUMCAD, NOMFUN, VALSAL \
                   FROM R034FUN \
                  WHERE NUMEMP = 1    \
                        __INSERIR(:SQLTIPCOL)\
                    AND TIPSEX = 'M' \
                        __INSERIR(:SQLNUMCAD)";
```

Removendo continuaria compilando.

```
Cur_R034FUN.SQL "SELECT NUMEMP, TIPCOL, NUMCAD, NOMFUN, VALSAL \
                   FROM R034FUN \
                  WHERE NUMEMP = 1 \
                    AND TIPSEX = 'M'";
```

Note que neste caso o comando continua sintaticamente correto depois que a macro foi removida. Portanto, a macro pode ser utilizada neste lugar.

Exemplo 5 - Gestão de Pessoas:

```
Cur_R034FUN.SQL "SELECT NUMEMP, TIPCOL, NUMCAD, NOMFUN, VALSAL \
                   FROM R034FUN \
                  WHERE NUMEMP = 1 \
                    AND TIPCOL = __INSERIR(:SQLTIPCOL) \
                    AND NUMCAD = __INSERIR(:SQLNUMCAD)";
```

Removendo ficaria incorreto e não vai compilar.

```
Cur_R034FUN.SQL "SELECT NUMEMP, TIPCOL, NUMCAD, NOMFUN, VALSAL \
                   FROM R034FUN \
                  WHERE NUMEMP = 1 \
                    AND TIPCOL = AND NUMCAD = ";  @ erro neste ponto @
```

Note que a macro foi removida. Este comando não compilaria mais em Senior SQL 2. Neste caso a macro não poderá ser usada.

Exemplo 6 - Gestão de Pessoas:

```
Cur_R034FUN.SQL "SELECT NUMEMP, TIPCOL, NUMCAD, NOMFUN, VALSAL \
                   FROM R034FUN\
                  WHERE NUMEMP = 1 \
                    AND __INSERIR(:SQLTIPCOL) \
                    AND __INSERIR(:SQLNUMCAD) \
                    AND TIPSEX = 'M'";
```

Removendo ficaria incorreto e não vai compilar.

```
Cur_R034FUN.SQL "SELECT NUMEMP, TIPCOL, NUMCAD, NOMFUN, VALSAL \
                   FROM R034FUN \
                  WHERE NUMEMP = 1 AND AND \ @ erro neste ponto @
                        TIPSEX = 'M'";
```

Desta maneira a macro também não pode ser usada. A remoção da macro causa um erro de compilação neste comando SQL.

Macro __inserir na execução: Durante a execução da regra, o valor que estiver contido dentro da variável indicada pela macro, irá ser inserido dentro do comando SQL do cursor. Como uma boa parte dos cursores são executados dentro do gerador de relatórios e este somente

utiliza Senior SQL 2, os valores inseridos no comando através da macro terão um tratamento adicional. Caso este valor contenha uma função existente na linguagem Senior SQL 2, este será convertido para o comando nativo caso a regra esteja utilizando comando SQL nativo.

Isto foi feito devido ao fato de que o gerador de relatórios somente utilizar Senior SQL 2 e as regras utilizarem tanto SQL Senior 2 quanto nativo. Os valores retornados dentro dos modelos de relatórios e que fazem parte do SQL do gerador, podem eventualmente serem usados em SQL dos cursores. Como a regra pode usar comandos nativos, isto mantém a compatibilidade.

A única função suportada neste formato é a função Day.

Exemplo – Gestão Empresarial – Utilizando clausulas externas:

```
Definir Cursor Cur_E006PAI; @ Tabela E006PAI de países do sistema Gestão
Empresarial.@
Definir Numero nCount;
Definir Alfa aStr;
Definir Alfa aMacro; @ Conterá o SQL a ser inserido no select do cursor.@

@ Condição especial que será adicionada ao SQL do cursor @
@ através da macro __Inserir.@
aMacro = " AND CODMOE = '01'";
Cur_E006PAI.SQL "SELECT CODPAI, NOMPAI \
                   FROM E006PAI \
                  WHERE MERSUL = 'S' \
                        __INSERIR(:aMacro) \
                  ORDER BY CODPAI";
nCount = 0;
Cur_E006PAI.AbrirCursor();
Enquanto(Cur_E006PAI.Achou)
  Inicio
  @ Conta quantos registros encontrou. @
  @ Teste com e sem a macro __Inserir para ver a diferença na quantidade @
  @ de registros que serão encontrados. Com a macro deverá encontrar @
  @ menos registros pois terá mais filtro no where. @
  nCount++;
  Cur_E006PAI.Proximo();
  Fim;
Cur_E006PAI.FecharCursor();

IntParaAlfa(nCount, aStr);
aStr = "Encontrados '" + aStr + "' registros!";

Mensagem(Retorna, aStr);
```

Exemplo – Gestão de Pessoas – Utilizando clausulas externas:

```
Definir Cursor Cur_R074CID; @ Tabela R074CID de Cidades do Gestão de Pessoas.@
Definir Alfa aMacroUF; @ Conterá o SQL a ser inserido no select do cursor.@
Definir Alfa aContador;
Definir Alfa aMensagem;
@ Condição especial que será adicionada ao SQL do cursor @
@ através da macro __Inserir.@
aMacroUF = "WHERE R074CID.ESTCID = 'AC'";
nContador = 0;
```

```
Cur_R074CID.SQL "SELECT CODCID FROM R074CID __Inserir(:aMacroUF)";
Cur_R074CID.AbrirCursor();
Enquanto (Cur_R074CID.Achou)
    Inicio
    @ Conta quantos registros encontrou. @
    @ Teste com e sem a macro __Inserir para ver a diferença na quantidade @
    @ de registros que serão encontrados. Com a macro deverá encontrar @
    @ menos registros pois terá filtro no where. @
    nContador++;
    Cur_R074CID.Proximo();
    Fim;
Cur_R074CID.FecharCursor();

IntParaAlfa(nContador,aContador);
aMensagem = "O Total de Cidades Encontradas foi " + aContador;

Mensagem(Retorna,aMensagem);
```

AUTOATIVIDADE

Agora que você já explorou todos os recursos e informações sobre o gerador de telas, vamos praticar e criar um cursor com a seguinte estrutura :

- Gestão de Pessoas: Mensagem com Código e Descrição do Estado (tabela de estados) SC e RS e abaixo de cada estado a listagem das cidades (código e descrição).

- Gestão Empresarial: Mensagem com Código e Descrição do Estado (tabela de estados) SC e RS e abaixo de cada estado a listagem das cidades (Cep Inicial e descrição).

CAPÍTULO 02

Função ExecSQL

Objetivos Específicos

- Desenvolver regras com o recursos da função ExecSql.

Comando ExecSQL

O comando ExecSQL permite executar comandos DML (Data Manipulation Language - Linguagem de Manipulação de Dados). A DML é um subconjunto da linguagem SQL usada para inserir, atualizar e apagar dados. Os seguintes comandos DML podem ser utilizados pela função ExecSQL nas regras:

Insert: é usado para inserir um novo registro em uma tabela existente.

Update: possibilita mudar os valores de dados em uma ou mais linhas de uma tabela existente.

Delete: permite remover linhas existentes de uma tabela.

Abaixo temos um exemplo de como inserir um novo registro em uma tabela existente no banco de dados através da função ExecSQL.

Exemplo – Gestão Empresarial – Inserção de Dados:

```
Definir Cursor Cur_E006PAI; @E006PAI é a tabela de países do Gestão Empresarial@
Definir Numero nEANPai;
Definir Numero nResult;
Definir Alfa aCodPai;
Definir Alfa aNomPai;
Definir Alfa aMerSul;
Definir Alfa aVisEnt;

aCodPai = "9876";
aNomPai = "País " + aCodPai;
aMerSul = "N";
nEANPai = 0;
aVisEnt = "N";

@Insere um novo registro para a tabela de países. @
ExecSql "INSERT INTO E006PAI \
         (CODPAI, NOMPAI, CODMOE, MERSUL, EANPAI, VISENT, PAISIS) VALUES \
         (:ACODPAI, :ANOMPAI, NULL, :AMERSUL, :NEANPAI, :AVISENT, NULL)";

nResult = 0; @ Não Achou @

@Verificamos se realmente inseriu o novo país na tabela E00PAI,@
@procurando pelo registro inserido com um Cursor.@
Cur_E006PAI.SQL "SELECT CODPAI \
                   FROM E006PAI \
                  WHERE CODPAI = :ACODPAI";

Cur_E006PAI.AbrirCursor();

Se  (Cur_E006PAI.Achou)
    Inicio
    nResult = 1; @ Achou @
    Fim;
@Note que não deixamos para fechar o cursor só depois de exibir a mensagem.@
@E se o usuário fosse tomar um cafezinho com a mensagem ainda na tela?@
@O cursor iria ficar aberto por tempo indeterminado sem necessidade. @
Cur_E006PAI.FecharCursor();
```

```
Se  (nResult = 1)
    Mensagem(Retorna, "Registro encontrado!");
Senao
    Mensagem(Retorna, "O registro não foi encontrado!");
```

Se o comando ExecSQL gerar uma exceção (um erro), será exibida uma mensagem de erro notificando o usuário sobre o erro que ocorreu. Um exemplo clássico é executar a regra acima duas vezes. Na primeira vez o comando insert do ExecSql será executado com sucesso, porém na segunda vez, levantará uma exceção notificando que o registro já existe (erro de violação de chave primária), conforme exibido pela mensagem abaixo:

Exemplo – Gestão de Pessoas – Inserção de Dados:

```
Definir Cursor Cur_R; @R074CID é a tabela de cidades do Gestão de Pessoas@
Definir Alfa aNomCid;
Definir Alfa aEstCid;
Definir Alfa aCodEst;
Definir Alfa aCodMun;
Definir Alfa aDisCid;

nCodCid = 1;
aNomCid = "Cidade ExecSQL";
aEstCid = "AC";
nCodPai = 1;
aCodEst = "AC";
aCodMun = "";
nPerIss = 0;
aDisCid = "";

@Insere um novo registro para a tabela de Cidade. @
ExecSql "INSERT INTO R074CID \
         (CODCID, NOMCID, ESTCID, CodCid, CODEST, CODMUN, PERISS, DISCID) \
          VALUES \
         (:nCodCid, :aNomCid, :aEstCid, :nCodPai, :aCodEst, :aCodMun,
          :nPerIss, :aDisCid)";

@Verificamos se realmente foi inserido uma nova cidade na tabela R074CID,@
@procurando pelo registro inserido com um Cursor.@
Cur_R074CID.SQL "SELECT CodCid \
                   FROM R074CID \
                  WHERE CODCID = :nCodCid";

Cur_R074CID.AbrirCursor();
nResult = 0; @ Não Achou @
```

```
Se  (Cur_R074CID.Achou)
    Inicio
    nResult = 1; @ Achou @
    Fim;
@Note que não deixamos para fechar o cursor só depois de exibir a mensagem.@
@E se o usuário fosse tomar um cafezinho com a mensagem ainda na tela?@
@O cursor iria ficar aberto por tempo indeterminado sem necessidade. @
Cur_R074CID.FecharCursor();

Se  (nResult = 1)
    Mensagem(Retorna, "Registro encontrado!");
Senao
    Mensagem(Retorna, "O registro não foi encontrado!");
```

O próximo exemplo demonstra como executar a função ExecSql para alterar os dados de um registro de uma tabela existente.

Exemplo – Gestão Empresarial – Alteração de Dados:

```
Definir Cursor Cur_E006PAI; @ E006PAI é a tabela de países do Gestão Empresarial
@
Definir Alfa aCodPai;
Definir Alfa aNomPai;

aCodPai = "9876";@Altera o nome do país "9876" da tabela de países.@

ExecSql "UPDATE E006PAI SET NOMPAI = 'NOME ALTERADO' WHERE CODPAI = '9876'";

@Recuperamos o registro alterado apenas para verificar.@
Cur_E006PAI.SQL "SELECT NOMPAI FROM E006PAI WHERE CODPAI = :ACODPAI";

aNomPai = "";
Cur_E006PAI.AbrirCursor();
Se  (Cur_E006PAI.Achou)
    Inicio
    aNomPai = "Nome do país '" + aCodPai + "' alterado para '" +
              Cur_E006PAI.NomPai + "'!";
    Fim;
Cur_E006PAI.FecharCursor();

Se  (aNomPai <> "")
    Mensagem(Retorna, aNomPai);
Senao
    Mensagem(Retorna, "O registro não foi encontrado!");
```

Exemplo – Gestão de Pessoas Alteração de Dados:

```
Definir Cursor Cur_R074CID; @ R074CID é a tabela de países @
Definir Alfa aNomCid;

nCodCid = 1; @Altera o nome do cidade "1" da tabela de Cidades.@
```

```
ExecSql "UPDATE R074CID SET NOMCID = 'NOME ALTERADO EXECSQL' \
          WHERE CodCid = :nCodCid";

@Recuperamos o registro alterado apenas para verificar.@
Cur_R074CID.SQL "SELECT NomCid FROM R074CID WHERE CodCid = :nCodCid";

aNomCid = "";
Cur_R074CID.AbrirCursor();
Se  (Cur_R074CID.Achou)
    Inicio
    aNomCid = "Nome da Cidade alterado para '" + Cur_R074CID.NomCid + "'!";
    Fim;
Cur_R074CID.FecharCursor();

Se  (aNomCid <> "")
    Mensagem(Retorna, aNomCid);
Senao
    Mensagem(Retorna, "O registro não foi encontrado!");
```

No exemplo seguinte, demonstraremos como executar a função ExecSql para excluir um registro de uma tabela.

Exemplo – Gestão Empresarial – Exclusão de Registros:

```
Definir Funcao PaisExiste(Numero End aPaisExiste);
Definir Numero nPaisExiste;
Definir Alfa aCodPai;

aCodPai = "9876";
nPaisExiste = 0; @Assume que o país não existe inicialmente.@

@Primeiramente verificamos se o país existe.@
PaisExiste(nPaisExiste);
Se  (nPaisExiste = 1) @ Se o país existir iremos excluír da base de dados. @
    Inicio
    ExecSql "DELETE FROM E006PAI WHERE CODPAI = :ACODPAI";
    PaisExiste(nPaisExiste); @ Testamos novamente se o país existe. @
    Fim;

@ Função definida pelo programador da regra @
@ que retorna valor no final da sua execução. @
Funcao PaisExiste(Numero End aPaisExiste);
    Inicio
    Definir Cursor Cur_E006PAI;
    Definir Alfa aNomPai;
    aNomPai = "";
    aPaisExiste = 0; @ Não existe @
    Cur_E006PAI.SQL "SELECT NOMPAI FROM E006PAI WHERE CODPAI = :ACODPAI";
    Cur_E006PAI.AbrirCursor();
    Se (Cur_E006PAI.Achou)
       Inicio
       aNomPai = "Nome do país '" + aCodPai + "' = '" + Cur_E006PAI.NomPai +
                 "'!";
       aPaisExiste = 1; @ País existe @
       Fim;
    Senao
```

```
        aNomPai = "País '" + aCodPai + "' não encontrado!";
    Cur_E006PAI.FecharCursor();
    Mensagem(Retorna, aNomPai);
    Fim;
```

Exemplo – Gestão de Pessoas – Exclusão de Registros:

```
Definir Funcao CidadeExiste(Numero End aCidadeExiste);
Definir Alfa aCodCid;
Definir Alfa aMensagem;

nCodCid = 1;
ConverteMascara(1,nCodCid,aCodCid,"99.99999");

nCidadeExiste = 0;

@Primeiramente verificamos se o país existe.@
CidadeExiste(nCidadeExiste);
Se  (nCidadeExiste = 1) @ Se a Cidade existir será excluída da base. @
    Inicio
    ExecSql "DELETE FROM R074CID WHERE CODCID = :nCodCid";
    CidadeExiste(nCidadeExiste); @ Testamos novamente se o país existe. @
    Fim;

Funcao CidadeExiste(Numero End aCidadeExiste);
    Inicio
    Definir Cursor Cur_R074CID;
    Definir Alfa aNomCid;
    aMensagem = "";
    aCidadeExiste = 0; @ Não existe @
    Cur_R074CID.SQL "SELECT NOMCID FROM R074CID WHERE CODCID = :nCodCid";
    Cur_R074CID.AbrirCursor();
    Se (Cur_R074CID.Achou)
       Inicio
       aMensagem = "Cidade '" + aCodCid +"' - '"+ Cur_R074CID.NomCid + "'!";
       aCidadeExiste = 1; @ A Cidade existe @
       Fim;
    Senao
       aMensagem = "Cidade '" + aCodCid + "' não encontrado!";
    Cur_R074CID.FecharCursor();
    Mensagem(Retorna, aMensagem);
    Fim;
```

Função ExecSQLEx

O funcionamento dessa função é muito semelhante ao comando ExecSql. Porém com essa função é possível efetuar tratamento de exceção ao executar o comando SQL.

Sintaxe da função: ExecSqlEx(<Alfa ComandoSQL>, <Numero Sucesso>, <Alfa Mensagem>);

- ComandoSQL: é o comando SQL que se deseja executar, semelhante ao comando ExecSql;
- Sucesso: esse comando retorna 0 (zero) se o comando foi executado com sucesso ou 1 (um) se ocorreu alguma exceção (erro na execução do SQL);
- Mensagem: se ocorreu alguma exceção na execução do comando SQL, esse parâmetro retornará a mensagem de erro, caso contrário não retorna nada.

Exemplo – Gestão Empresarial – Uso de ExecSQLEx:

```
@Variáveis para utilizar na função ExecSqlEx@
Definir Numero nSucesso;
Definir Alfa aMensagem;
Definir Alfa aSQL;
Definir Alfa aCodPai;
Definir Alfa aNomPai;
Definir Alfa aMerSul;
Definir Numero nEANPai;
Definir Alfa aVisEnt;

aCodPai = "9876";
aNomPai = "País " + aCodPai;
aMerSul = "N";
nEANPai = 0;
aVisEnt = "N";
@Insere (tenta) um novo registro para a tabela de países.@
ExecSqlEx("insert into E006PAI (CodPai, NomPai, CodMoe, MerSul, \
          EanPai, VisEnt, PaiSis) values (:aCodPai, :aNomPai, \
          null, :aMerSul, :nEANPai, :aVisEnt, null)", nSucesso, aMensagem);
Se  (nSucesso = 0)
    Mensagem(Retorna, "Registro inserido com sucesso!");
Senao
    Inicio
    aMensagem = "Erro ao inserir registro:" + aMensagem;
    Mensagem(Erro, aMensagem);
    Fim;
```

Exemplo – Gestão de Pessoas – Uso de ExecSQLEx:

```
@Variáveis para utilizar na função ExecSqlEx@
Definir Numero nSucesso;
Definir Alfa aMensagem;
Definir Alfa aSQL;
Definir Cursor Cur_R074CID; @R074CID é a tabela de cidades do Gestão de Pessoas@
Definir Alfa aNomCid;
Definir Alfa aEstCid;
Definir Alfa aCodEst;
Definir Alfa aCodMun;
Definir Alfa aDisCid;

nCodCid = 1;
aNomCid = "Cidade ExecSQL";
aEstCid = "AC";
nCodPai = 1;
```

```
aCodEst = "AC";
aCodMun = "";
nPerIss = 0;
aDisCid = "";

@Insere um novo registro para a tabela de Cidade. @
ExecSqlEx("INSERT INTO R074CID \
          (CODCID, NOMCID, ESTCID, CodCid, CODEST, CODMUN, PERISS, DISCID) \
          VALUES (:nCodCid, :aNomCid, :aEstCid, :nCodPai, :aCodEst,\
                  :aCodMun, :nPerIss, :aDisCid)" , nSucesso, aMensagem);
Se  (nSucesso = 0)
    Mensagem(Retorna, "Registro inserido com sucesso!");
Senao
    Inicio
    aMensagem = "Erro ao inserir registro:" + aMensagem;
    Mensagem(Erro, aMensagem);
    Fim;
```

Utilizando transações

Visão Geral

Uma transação é uma unidade de processamento (atômica) que faz uma alteração no banco de dados, e que não pode ser efetuada parcialmente. Uma transação pode ser um comando ou um conjunto de comandos que atualizam dados em algumas tabelas do banco. No caso de o programador não especificar explicitamente o início e o fim de uma transação, cada comando de atualização no banco é considerado uma transação. O aspecto mais importante de uma transação é que ela não pode ser efetuada parcialmente (garantido pela atomicidade de uma transação). Isso significa que se forem agrupados dez comandos de atualização na mesma transação, ou todos serão executados com sucesso, ou nenhum será. Ou seja, se ocorrer um erro na execução do oitavo comando, todos os sete primeiros serão desfeitos. Transações ajudam a fazer o controle multiusuário, pois cada usuário não enxerga alterações efetuadas por uma transação antes de ela ser concluída.

A LSP disponibiliza 3 procedimentos para manipular transações. Eles serão explicados a baixo e devem ser utilizadas com o comando ExecSql e/ou com a função ExecSQLEx (mais recomendado por permitir verificar se ocorreu algum erro ou não na sua execução), ou com qualquer outro recurso que execute comandos SQL DML nas regras. Abaixo estão listados os métodos para controle de transações nas regras:

- **IniciarTransacao:** Inicia uma transação no banco de dados.
  - **Sintaxe**:IniciarTransacao();
- **FinalizarTransacao:** Finaliza uma transação no banco de dados.
  - **Sintaxe**:FinalizarTransacao();
- **DesfazerTransacao:** Desfaz uma transação iniciada anteriormente.
  - **Sintaxe**: DesfazerTransacao();

Exemplo – Gestão Empresarial – Uso de Transações:

```
@Variáveis representando os campos da tabela E006PAI@
@(países) do Sistema Gestão Empresarial.@
Definir Funcao ValidaErro();
Definir Alfa aCodPai;
Definir Alfa aNomPai;
Definir Alfa aMerSul;
Definir Numero nEANPai;
Definir Alfa aVisEnt;
@Variáveis para alguns campos da tabela E007UFS(Estados) do sistema Gestão
Empresarial.@
Definir Alfa aSigUfs;
Definir Alfa aNomUfs;
@Variáveis utilizadas como parâmetro na função ExecSqlEx@
Definir Alfa aMsgErro;
Definir Numero nSucesso;

aCodPai = "9876";
aNomPai = "País " + aCodPai;
aMerSul = "N";
nEANPai = 0;
aVisEnt = "N";

@Inicio da operação atômica de comandos SQL envolvendo transação.@
IniciarTransacao();
@Excluímos os estados do país em questão.@
ExecSqlEx("delete from E007UFS where CodPai = :aCodPai", nSucesso, aMsgErro);
@Após a execução de um comando SQL sempre validamos se ocorreu algum erro@
@para desfazer a transação. A função ExecSqlEx é ideal para isso, pois@
@ela possibilita sabermos se ocorreu algum erro ou não em sua execução.@
ValidaErro();

@Excluímos o país.@
ExecSqlEx("delete from E006PAI where CodPai = :aCodPai", nSucesso, aMsgErro);
ValidaErro();
@Inserimos novamente o país.@
ExecSqlEx("insert into E006PAI values (:aCodPai, :aNomPai, null, :aMerSul, \
          :nEANPai, :aVisEnt, null)",  nSucesso, aMsgErro);
ValidaErro();
@Para esse país iremos inserir 9 (nove) novos estados.@
Para(i = 1; i <= 9; i++)
    Inicio
    @Monta a sigla do estado@
    aSigUfs = "";
    IntParaAlfa(i, aSigUfs);
    aSigUfs = "A" + aSigUfs;

    @Monta o nome do estado@
    aNomUfs = "Estado " + aSigUfs;
    @Insere o estado na tabela de estados.@
    ExecSqlEx("insert into E007UFS values (:aSigUfs, :aNomUfs, :aCodPai, \
               null)", nSucesso, aMsgErro);
    ValidaErro();
    Fim;
@Aplica definitivamente no banco de dados todas as operações SQL@
@realizadas até aqui. Note que ou todas as operações SQL são bem@
@sucedidas ou nenhuma (atomicidade).@

FinalizarTransacao();
Mensagem(Retorna, "A operação foi realizada com sucesso!");

@Nesse exemplo, essa função deve ser chamada sempre após executar@
@a função ExecSqlEx para validar sua execução.@
Funcao ValidaErro();
  Inicio
  @Ocorreu algum erro na execução da última chamada a função ExecSqlEx.@
```

```
    Se (nSucesso = 1)
       Inicio
       @Todas as operações SQL executadas até esse ponto serão desfeitas.@
       DesfazerTransacao();
       aMsgErro = "Erro na execução da função ExecSqlEx:" + aMsgErro;
       @Gera uma exceção (erro) abortando a execução da regra.@
       Mensagem(Erro, aMsgErro);
       Fim;
    Fim;
```

Exemplo – Gestão de Pessoas – Uso de Transações:

```
Definir Funcao ValidaErro();
Definir Alfa aNomCid;
Definir Alfa aEstCid;
Definir Alfa aCodEst;
Definir Alfa aCodMun;
Definir Alfa aDisCid;
Definir Alfa aNomBai;
Definir Alfa aAux;
Definir Alfa aMsgErro;

nCodCid = 1;
aNomCid = "Cidade ExecSQL";
aEstCid = "AC";
nCodPai = 1;
aCodEst = "AC";
aCodMun = "";
nPerIss = 0;
aDisCid = "";

IniciarTransacao();
ExecSqlEx("DELETE FROM R074BAI WHERE CODCID = :nCodCid", nSucesso, aMsgErro);
ValidaErro();

ExecSqlEx("DELETE FROM R074CID WHERE CODCID = :nCodCid", nSucesso, aMsgErro);
ValidaErro();

ExecSqlEx("INSERT INTO R074CID VALUES (:nCodCid, :aNomCid, :aEstCid, :nCodPai,
:aCodEst, :aCodMun, :nPerIss, :aDisCid)",
  nSucesso, aMsgErro);
ValidaErro();
Para(i = 1; i <= 9; i++)
    Inicio
    aNomBai = "";
    IntParaAlfa(i, aAux);
    aNomBai = "Bairro A" + aAux;
    nCodBai = i;
    nCepBai = 8900000 + i;
    ExecSqlEx("insert into R074BAI values (:nCodCid, :nCodBai, :aNomBai,
:nCepBai)", nSucesso, aMsgErro);
    ValidaErro();
    Fim;

FinalizarTransacao();
Mensagem(Retorna, "A operação foi realizada com sucesso!");

Funcao ValidaErro();
  Inicio
```

```
Se (nSucesso = 1)
   Inicio
   DesfazerTransacao();
   aMsgErro = "Erro na execução da função ExecSqlEx:" + aMsgErro;
   Mensagem(Erro, aMsgErro);
   Fim;
Fim;
```

AUTOATIVIDADE

Desenvolva uma regra que permita a validação comando de update na base para o campo MERSUL = “S” (GEmp E006PAI) e DISCID = “BRA” (GP R074CID).

CAPÍTULO 03

Lista Dinâmica

Objetivos Específicos

- Desenvolver regras com o uso de lista dinâmica.

Lista Dinâmica

A lista dinâmica é um mecanismo virtual (fica alocada apenas na memória temporária do computador) que permite manipular dados em forma de tabela, podendo essa tabela possuir para cada coluna um tipo de dado nativamente suportado pela LSP. A lista dinâmica disponibiliza uma série de métodos, propriedades e funções para manipulá-la, tornando sua utilização muito fácil, prática e flexível.

Exemplo de como seria uma lista dinâmica (se pudéssemos visualizá-la na memória) definida com 6 colunas e já possuindo 4 linhas de dados:

Empresa (Nro)	TipoColaborador (Numero)	Cadastro (Numero)	Nome (Alfa)	DataAdmissao (Data)	Salário (Numero)
IDA (Início De Arquivo)					
1	1	100	João	05/08/2000	**3.859,55**
1	1	200	Antônio	30/01/2005	**1.983,22**
2	2	300	Maria	12/04/2007	**1.100,00**
2	3	400	Carlos	22/11/2003	**2.56,87**
FDA (Fim De Arquivo)					

A lista permite:

- Definir as colunas com seus respectivos nomes e tipo de dados.
- Adicionar novas linhas e atribuir valor para cada coluna conforme seu tipo de dado definido;
- Navegar por todas as linhas da lista, podendo acessar os seus dados e alterá-los se necessário;
- Ir para uma determinada linha;
- Remover linhas;
- Saber quantos registros (linhas) a lista possui, e muito mais.

Termos usados neste documento

Alguns termos usados neste documento precisam ser explicados para o total entendimento deste recurso.

Registro Virtual: É um registro da lista que somente existe logicamente. Este não possui dados. Existem dois em qualquer lista definida na regra: um no início e outro no final. Estes registros sempre existem mesmo que a lista esteja vazia.

IDA: Significa Início De Arquivo. Indica que a lista está no primeiro registro virtual. Qualquer acesso aos dados com a lista neste estado, ocasionará um erro em tempo de execução.

FDA: Significa Fim De Arquivo. Indica que a lista está no último registro virtual. Qualquer acesso aos dados com a lista neste estado ocasionará um erro na execução da regra.

Funcionamento Básico

O funcionamento básico deste novo recurso consiste em:

- Determinar os campos que a lista usará;
- Preencher a lista com valores e;
- Usar estes valores de maneira que atenda as necessidades da lógica implementada pelo programador/usuário.

Comandos

Os comandos implementados por este novo recurso estão relacionados a seguir juntamente com uma explicação:

Comando para definição de listas: São comandos que determinam o formato da mesma. Este formato hoje somente é determinado pelos campos que compõem a lista.

Definição tipo Lista: Este comando serve para determinar o tipo de uma variável que será a lista. Nenhum parâmetro adicional será necessário para esta definição.

Definição dos Campos: procedimento DefinirCampos: Este comando inicia a fase de adição de campos na lista. Somente podem ser adicionados campos durante este período, ou seja, após a chamada deste comando. O mesmo não adiciona nenhuma informação de campos. Isto será feito por um comando que será visto mais adiante.

Confirmação de Campos: procedimento EfetivarCampos: Este comando determinará o fim da adição de campos e informará ao compilador/interpretador que a partir deste ponto a lista será usada efetivamente (receberá valores). Também permitirá ao interpretador criar estruturas internas de controle e manipulação desta lista.

Adição de campos: procedimento AdicionarCampo: Este comando adiciona os campos. Nesta adição também deve ser informado o tipo e o tamanho se necessário.

Sintaxe:

- funcao <lista>.AdicionarCampo(Alfa NomeCampo, <tipo> TipoInterno, Numero Tamanho);
- NomeCampo: indica o nome do campo. Este parâmetro deve ser um literal alfanumérico (constante). O nome do campo não deve conter espaços, acentos e nem número como primeiro caractere;
- TipoInterno: indica o tipo que o campo terá. Deve ser um tipo primitivo interno da regra, ou seja, numero, alfa ou data;
- Tamanho: parâmetro opcional que determina o tamanho do campo. Se informado, somente será aceito para campos alfanuméricos. Neste caso, o campo terá um tamanho limitado. Se não for informado, campos do tipo alfa

não terão limite (podendo ter valores até o limite de memória). Os outros tipos de campos não são afetados.

Acesso aos campos

O acesso aos campos que foram definidos dentro da lista, deve ser feito digitando-se o nome da lista, seguido do ponto (.) e o nome do campo. Este nome deverá ser definido previamente através do comando AdicionarCampo. Caso o nome digitado após o ponto não for um nome de procedimento, função, propriedade ou campo definido na lista, um erro de compilação será gerado.

Comandos para manipulação de registros

Estes comandos permitem adicionar, inserir, gravar, excluir etc. registros da listas para usar todo o potencial dinâmico do recurso.

Adição de registros: procedimento Adicionar: Este comando é o primeiro comando de manipulação de dados do recurso lista. Ele serve para adicionar valores (agrupados em registros) dentro da lista. Ele cria um registro no final dos registros existentes. Este somente respeitará a ordem de adição se não existirem chaves definidas (será visto mais adiante).

Inserção de registros: procedimento Inserir: Este comando tem a mesma função do comando Adicionar, mas ao invés de adicionar um registro no final dos registros existentes, insere-o na posição atual da lista (apontado internamente e acessível pela propriedade NumReg).

Edição de registros: procedimento Editar: Este comando visa a atualização de registros. Para tal é necessário posicionar a lista no registro que se deseja alterar. Após este posicionamento é executado o comando Editar o qual muda os valores desejados.

Gravação de registros: procedimento Gravar: Quando se altera os valores dos campos (após a chamada do comando Adicionar, Inserir ou Editar), pode-se efetivar os dados através do comando Gravar. Este comando grava as informações dentro da lista para posterior recuperação.

Cancelamento de registros: procedimento Cancelar: Quando se altera os valores dos campos e por algum motivo os mesmos não devem ser efetivados, chama-se o comando Cancelar. Os dados que estão sendo alterados ficam em um registro virtual que não é trabalhado até que seja chamado o comando Gravar ou Cancelar. No caso do comando Cancelar este registro virtual é descartado não alterando o conteúdo da lista.

Exclusão de registros: procedimento Excluir: Este comando exclui um registro. Para tal é necessário posicionar a lista no registro que deverá ser excluído e então chamar o comando Excluir. Somente o registro atualmente posicionado será excluído. Para excluir mais registros é necessário chamar o comando mais vezes.

Comandos para posicionamento de listas

Estes comandos existem para que o programador/usuário possa posicionar o registro da lista e permitir uma maior agilidade no uso do recurso.

O primeiro registro: função Primeiro: Este comando posiciona no primeiro registro que estiver na lista. Note que o primeiro registro pode ser o primeiro adicionado ou o primeiro que respeitar a chave que estiver atualmente selecionada. Por exemplo: se existir um campo que for o nome do funcionário e a chave estiver configurada para este campo, o primeiro registros provavelmente será um nome que comece por A. Este comando retorna 1 (um) se a lista pôde ser posicionada no primeiro registro e 0 (zero) caso contrário.

O último registro: função Ultimo: Este comando posiciona a lista no último registro. Da mesma forma como o comando Primeiro, o último registro pode ser o último registro adicionado ou o registro que estiver obedecendo a chave. No exemplo anterior (nome do funcionário) o último registro poderia ser um nome que começasse com Z. Este comando retorna 1 (um) se a lista pôde ser posicionada no final e 0 (zero) caso contrário.

O registro anterior: função Anterior: O comando Anterior posiciona a lista no registro imediatamente anterior ao registro atual. Se não existir registro anterior, será posicionada em IDA. Segue a mesma lógica de chave do comando Primeiro e Ultimo. Se a lista pôde ser posicionada no registro anterior (que não é o IDA), este comando retorna 1 (um). Caso contrário retorna 0 (zero).

O próximo registro: função Próximo: Posiciona a lista no registro imediatamente posterior ao registro atual. Se não existir registro posterior, será posicionada em FDA. A lógica de chave segue o padrão dos comandos de posicionamento anteriores. Retorne 1 (um) se foi possível posicionar no próximo registro e 0 (zero) caso não tenha conseguido.

Comandos para procura de registros

Estes comandos auxiliam ao programador/usuário na procura de registros dentro da lista através de valores previamente conhecidos. Veremos quais são eles:

Configurar valores de chave para a procura: procedimento SetarChave: Este comando coloca a lista em estado de edição de chave para que seja possível a manipulação dos valores da chave. Quando configurados estes valores, será possível procurar os registros que possuem a chave informada. Isto será feito através do comando VaiParaChave que será visto a seguir. Este comando apaga os valores que estiverem na chave no momento da chamada. Para manter os valores da chave use o comando EditarChave.

Editar os valores de chave para a procura: procedimento EditarChave: Este comando tem o mesmo objetivo do comando SetarChave mas sem apagar os valores da chave. Quando este comando for chamado os valores que estiverem contidos na chave neste momento serão mantidos e ainda assim a lista entrará em modo de edição de chave. Este comando serve para procurar por chaves muito parecidas sem que seja necessário informar todos os valores novamente.

Procurar valores dentro da lista: função VaiParaChave: Este comando procura pelo registro que tiver a chave configurada naquele momento. Por exemplo: Consideremos que a chave da lista seja o código de cadastro do funcionário e que o mesmo tenha o valor 10 (dez) após a chamada do comando SetarChave. Quando o comando VaiParaChave for chamado a lista será posicionada no primeiro registro onde o número do cadastro do funcionário for 10 (dez). Se o registro com esta característica não for encontrado, a lista não será reposicionada. Ficara no mesmo lugar. Caso o comando encontre o registro procurado, será retornado 1 (um). Caso contrário será retornado 0 (zero).

Comandos para posicionamento absoluto

Os comandos a seguir, informam e configuram a posição absoluta da lista conforme o número do registro.

Obtenção da posição: propriedade NumReg: Esta propriedade retorna o número do registro (baseado com início em zero) da posição atual da lista. Se a lista estiver posicionada no quarto registro, o valor retornado será 3 (três). Este número de registro é influenciado pela chave que estiver ativa no momento da obtenção deste valor. Por exemplo: Existe um registro na lista que não possui chave definida. O número deste registro é 2 (dois). Quando atribuímos uma chave para a lista, outro registro pode ter o número 2 (dois) e o registro que antes possuía o número 2 (dois) pode ter qualquer outro número, dependendo da chave aplicada.

Configuração da posição: procedimento SetaNumReg: Este procedimento ter como objetivo posicionar a lista de maneira absoluta. A posição da lista é a ordem do registro menos 1 (um). A ordem do registro é influenciada pela chave que estiver ativa no momento da chamada.

Comandos diversos de listas: Os comandos a seguir são de categoria geral, mas são utilizados normalmente com os outros comandos aqui apresentados.

Propriedade IDA: Retorna 1 (um) se a lista estiver em IDA (Início de Arquivo) e 0 (zero) caso contrário.

Propriedade FDA: Retorna 1 (um) se a lista estiver em FDA (Final de Arquivo) e 0 (zero) caso contrário.

Propriedade QtdRegistros: Retorna o número de registros que estão armazenados na lista naquele momento.

Procedimento Limpar: Apaga todos os registros da lista.

Procedimento Chave: Este procedimento configura a chave que a lista terá do momento da atribuição da chave em diante. A chave serve para ordenar os registros pelos campos que a compõe, começando da esquerda para a direita na lista de campos atribuídos como chave. A chave também serve para posicionar em um registro específico da lista. Uma chave pode ser composta por mais de um campo. Ela deve conter os nomes dos campos que estiverem configurados na lista separados por ponto-e-vírgula (;), exemplo: "Empresa;TipoColaborador". Caso não se queira chave nenhuma, deve-se configurar este valor com vazio ("").

Exemplos

Exemplo – Gestão de Pessoas – Uso de Lista Dinâmica:

```
Definir Lista LLista;
Definir Cursor Cur_R074EST;
Definir Cursor Cur_R074CID;
Definir Alfa aUF;
Definir Alfa aNomeEstado;
Definir Alfa aEnter;
Definir Alfa aMensagem;
Definir Alfa aQtdCid;
Definir Alfa aEnter;

RetornaAscII(13,aEnter);

@Definição dos campos da lista@
LLista.DefinirCampos();@Indica que será iniciada a definição dos campos@
LLista.AdicionarCampo("UF", Alfa);
LLista.AdicionarCampo("NomeEstado", Alfa);
LLista.AdicionarCampo("QtdCid", Numero);
LLista.EfetivarCampos();@Indica que será finalizada a definição dos campos@

@ Defini uma chave para a lista, os colaboradores @
@ serão ordenados pelo nome. @
LLista.Chave("UF");

@As listas são muito utilizadas em parceria com os cursores.@
Cur_R074EST.SQL "SELECT CODPAI, CODEST, DESEST FROM R074EST";
Cur_R074EST.AbrirCursor();
Enquanto(Cur_R074EST.Achou)
    Inicio
    @Armazena na lista os dados do cursor. @
    LLista.Adicionar();@ Cria um novo registro vazio na lista @
    LLista.UF = Cur_R074EST.CodEst;
    LLista.NomeEstado = Cur_R074EST.DesEst;
    nCodPai =  Cur_R074EST.CodPai;
    aUF     =  Cur_R074EST.CodEst;
    Cur_R074CID.SQL "SELECT CODCID FROM R074CID \
                     WHERE CODPAI = :nCodPai \
                       AND CODEST = :aUF";
    Cur_R074CID.AbrirCursor();
    nQtdCid = 0;
    Enquanto(Cur_R074CID.Achou)
        Inicio
        nQtdCid++;
        Cur_R074CID.Proximo();
        Fim;
    Cur_R074CID.FecharCursor();
    LLista.QtdCid = nQtdCid;
    LLista.Gravar();@Grava o registro na lista@
    Cur_R074EST.Proximo();
    Fim;
@Liberamos o cursor, porém temos todos os dados na lista.@
Cur_R074EST.FecharCursor();

aMensagem = "Estados: " + aEnter;
LLista.Primeiro();@ Posiciona a lista no primeiro registro. @
Enquanto(LLista.FDA = 0)
    Inicio
    aUF = LLista.UF;
    aNomeEstado = LLista.NomeEstado;
    nQtdCid = LLista.QtdCid;
    ConverteMascara(1,nQtdCid,aQtdCid,"zzz.zz9");
    aMensagem = aMensagem + aUF + " - " + aQtdCid + aEnter;
    LLista.Proximo();@Avança para o próximo registro da lista.@
    Fim;
```

```
Mensagem(Retorna,aMensagem);
```

Exemplo – Gestão Empresarial – Uso de Lista Dinâmica:

```
Definir Lista LLista;
Definir Cursor Cur_E007UFS;
Definir Cursor Cur_E008CEP;
Definir Alfa aUF;
Definir Alfa aNomeEstado;
Definir Alfa aMensagem;
Definir Alfa aQtdCid;
Definir Alfa aEnter;

RetornaAscII(13,aEnter);

@Definição dos campos da lista@
LLista.DefinirCampos();@Indica que será iniciada a definição dos campos@
LLista.AdicionarCampo("UF", Alfa);
LLista.AdicionarCampo("NomeEstado", Alfa);
LLista.AdicionarCampo("QtdCid", Numero);
LLista.EfetivarCampos();@Indica que será finalizada a definição dos campos@

@ Defini uma chave para a lista, os colaboradores @
@ serão ordenados pelo nome. @
LLista.Chave("UF");

@As listas são muito utilizadas em parceria com os cursores.@
Cur_E007UFS.SQL "SELECT SIGUFS, NOMUFS FROM E007UFS";
Cur_E007UFS.AbrirCursor();
Enquanto(Cur_E007UFS.Achou)
    Inicio
    @Armazena na lista os dados do cursor. @
    LLista.Adicionar();@ Cria um novo registro vazio na lista @
    LLista.UF         = Cur_E007UFS.SigUfs;
    LLista.NomeEstado = Cur_E007UFS.NomUfs;
    aUF     =  Cur_E007UFS.SigUfs;
    Cur_E008CEP.SQL "SELECT CEPINI FROM E008CEP WHERE SIGUFS = :aUF";
    Cur_E008CEP.AbrirCursor();
    nQtdCid = 0;
    Enquanto(Cur_E008CEP.Achou)
        Inicio
        nQtdCid++;
        Cur_E008CEP.Proximo();
        Fim;
    Cur_E008CEP.FecharCursor();
    LLista.QtdCid = nQtdCid;
    LLista.Gravar();@Grava o registro na lista@
    Cur_E007UFS.Proximo();
    Fim;
@Liberamos o cursor, porém temos todos os dados na lista.@
Cur_E007UFS.FecharCursor();
aMensagem = "Estados: " + aEnter;
LLista.Primeiro();@ Posiciona a lista no primeiro registro. @
Enquanto(LLista.FDA = 0)
    Inicio
    aUF = LLista.UF;
    aNomeEstado = LLista.NomeEstado;
    nQtdCid = LLista.QtdCid;
    ConverteMascara(1,nQtdCid,aQtdCid,"zzz.zz9");
    aMensagem = aMensagem + aUF + " - " + aQtdCid + aEnter;
    LLista.Proximo();@Avança para o próximo registro da lista.@
    Fim;
Mensagem(Retorna,aMensagem);
```

AUTOATIVIDADE

Desenvolver os itens abaixo:
- ERP: Carregar uma Lista Dinâmica com o Código do Cliente, Nome do Cliente e a Quantidade de Pedidos. O conteúdo da Lista Dinâmica deve ser visualizado através de uma Mensagem.
- Gestão de Pessoas: Carregar uma Lista Dinâmica com o Numero do Local, Nome do Local e a Quantidade de Colaboradores. O conteúdo da Lista Dinâmica deve ser visualizado através de uma Mensagem.

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

SENIOR
Universidade
Corporativa