Home
Específicos
Flex
Getting Started
Java
Processo Datasul
Progress
Tools
Trilhas
User
Page Operations
Browse Space

SDK - Software Development Kit - v.1.0

Narrativa - Comentários Relacionados

Added by Fabiano Ramos dos Santos, last edited by Fabiano Ramos dos Santos on Out 18, 2010 (view change) SHOW COMMENT
Labels
incubado, componente

Visão Geral

documento incubado

A solução de comentários relacionados permite que sejam adicionados comentários aos registros de um grid.

Características

Os comentários relacionados permitem que sejam inseridas informações pertinentes a cada registro em um grid.

O que a solução oferece:

- criação dinâmica de renderer para chamada à solução de comentários relacionados (vide técnica);
- disponibiliza interface para leitura e manutenção dos comentários.

> **Atenção**
>
> A solução é somente uma camada de apresentação, não cabendo-lhe a leitura ou persistência das informações, e encontra-se disponível para utilização somente com o Metadados via FreeForm/ABLScript.
>
> A janela utilizada para inclusão e visualização dos comentários necessita de um espaço considerável, cabendo ao analista responsável verificar se a sua utilização é adequada para a situação.

Localização

A solução de comentários relacionados encontra-se embutido nos componentes do Metadados e é inserido dinamicamente, mediante a técnica apresentada.

Quando utilizar o Componente?

Quando for necessário adicionar informações sobre um determinado registro.

Pré-Requisitos

- Criação de tabela para armazenar os comentários (a estrutura necessária será apresentada em como utilizar o componente);
- Conhecimentos em Progress e ABLScript.

Como utilizá-lo?

A utilização da solução de comentários relacionados inicia-se com a criação da tabela onde serão persistidos os comentários. Serão mostrados dois exemplos de como a tabela pode ser criada (utilizando o rowid do registro ou utilizando a chave da tabela relacionada, que pode ser composta) ficando a critério do analista definir a forma que julgar mais adequada.

A disponibilização dos comentários dar-se-á, basicamente, através dos passos aqui resumidos e detalhados logo a seguir:

1. Criação da tabela para armazenar os comentários;
2. Criação do script Progress responsável por efetuar a busca e persistência dos comentários;
3. Adição de campo para armazenar se há comentários disponíveis para determinado registro na temp-table que alimenta o grid;
4. Apresentação das informações em tela;
5. Persistência das informações após ser fechada a view dos comentários.

> **Por que necessito criar uma tabela?**
>
> Adotou-se o formato apresentado devido a possibilidade de crescimento geométrico de informações. Como a solução pode ser adotada tanto para o simples registro de comentários como para registro de históricos mais complexos, isto poderia gerar muitos registros comprometendo, com o passar do tempo, a performance da solução.

Passos para a utilização dos comentários

Passo 1 - Tabela para armazenar os comentários

Como todo o processo envolve o ABLScript (desde a apresentação da coluna até a persistência das informações), e o mesmo atualmente suporta somente tipos primitivos, o rowid, por exemplo, tem de ser utilizado como string.

Exemplo - Estrutura de tabela utilizando o rowid

Nome campo	Tipo Campo	Utilização
parentRowid	char(100)	Armazena (como string) o rowid correspondente ao registro pai do comentário.
commentText	char(2000)	Armazena o comentário.
updateUser	char(160)	Armazena o usuário atualizador do comentário.
updateDate	date	Armazena a data da atualização do comentário.

Passo 2 - Script Progress para manipulação dos comentários

Com a tabela para armazenar os comentários criada, deve-se prosseguir com a criação do script Progress responsável pela busca e persistência das informações. O exemplo aqui descrito foi desenvolvido sobre a tabela de módulo (modul_dtsul) e, para os comentários, foi criada uma tabela que foi nomeada modul_comment, a qual utiliza uma estrutura como a apresentada no primeiro exemplo.

```
def temp-table ttModulDtsul no-undo
    field commented as logical
    field codModulDtsul as character
    field desModulDtsul as character.

/*
 O campo eventType armazena o tipo de evento relacionado ao comentário. Esta
```

```
 informação está presente somente no retorno dos dados e deve ser verificada
 para realizar-se a ação correspondente.
*/
def temp-table ttComments no-undo
    field parentRowid as char  /* rowid do registro pai */
    field commentRowid as char /* rowid do comentário, quando já persistido */
    field commentText as char  /* texto do comentário */
    field updateUser as char   /* usuário atualizador */
    field updateDate as date   /* data da atualização */
    field eventType as char.   /* evento do comentário (create, update ou delete) */

procedure getModules:

    def output param table for ttModulDtsul.

    for each modul_dtsul no-lock:
        create ttModulDtsul.
        assign ttModulDtsul.codModulDtsul = modul_dtsul.cod_modul_dtsul
               ttModulDtsul.desModulDtsul = modul_dtsul.des_modul_dtsul.

        if  can-find(first modul_comments
                     where modul_comments.parentRowid = string(rowid(modul_dtsul))) then
            assign ttModulDtsul.commented = true.
        else
            assign ttModulDtsul.commented = false.
    end.

end.

procedure getComments:

    def input param vCodModulDtsul as char.
    def output param table for ttComments.

    find first modul_dtsul no-lock
         where modul_dtsul.cod_modul_dtsul = vCodModulDtsul no-error.
    if  not avail modul_dtsul then
        return.

    for each modul_comments no-lock
        where modul_comments.parentRowid = string(rowid(modul_dtsul)):
        create ttComments.
        assign ttComments.parentRowid  = string(rowid(modul_dtsul))
               ttComments.commentRowid = string(rowid(modul_comments))
               ttComments.commentText  = modul_comments.commentText
               ttComments.updateUser   = modul_comments.updateUser
               ttComments.updateDate   = modul_comments.updateDate.
    end.

end.

procedure setComments:

    def input param vCodModulDtsul as char.
    def input-output param table for ttComments.

    for each ttComments no-lock:

        case ttComments.eventType:
            when "create" then do:
                find first modul_dtsul no-lock
                     where modul_dtsul.cod_modul_dtsul = vCodModulDtsul no-error.

                find first modul_comments exclusive-lock
                     where rowid(modul_comments) = to-rowid(ttComments.commentRowid) no-error.
                if  avail modul_dtsul and not avail modul_comments then do:
                    create modul_comments.
                    assign modul_comments.parentRowid = string(rowid(modul_dtsul))
                           modul_comments.commentText = ttComments.commentText
                           modul_comments.updateUser  = ttComments.updateUser
                           modul_comments.updateDate  = ttComments.updateDate.
                    assign ttComments.parentRowid  = string(rowid(modul_dtsul))
                           ttComments.commentRowid = string(rowid(modul_comments)).
                end.
            end.
            when "update" then do:
                find first modul_comments exclusive-lock
                     where rowid(modul_comments) = to-rowid(ttComments.commentRowid) no-error.
                if  avail modul_comments then
                    assign modul_comments.commentText = ttComments.commentText
                           modul_comments.updateUser  = ttComments.updateUser
                           modul_comments.updateDate  = ttComments.updateDate.
            end.
            when "delete" then do:
                find first modul_comments exclusive-lock
                     where rowid(modul_comments) = to-rowid(ttComments.commentRowid) no-error.
                if  avail modul_comments then
                    delete modul_comments.
            end.
        end case.
```

```
    end.

end.
```

> **Observação**
>
> A temp-table que fará a movimentação dos comentários (aqui nomeada ttComments) deve, obrigatoriamente, conter os campos apresentados. Estes campos são utilizados pelo ABLScript e pela solução de comentários relacionados para apresentação e posteriormente para sua persistência.

Passo 3 - Campo "*commented*" para renderização da coluna de comentários

Conforme pode ser verificado no código apresentado, a temp-table ttModulDtsul contém um campo chamado "*commented*", do tipo logical, que armazenará a informação se o registro possui comentário. Este campo é necessário para manipulação do ícone e, consequentemente, da view que será apresentada ao ser clicado no renderer de comentários.

O campo "*commented*", neste caso, é alimentado com um simples can-find para o registro em questão. Para melhor perfomance, recomenda-se criar um índice para o campo (ou campos, se chave composta) que será a chave da tabela. O nome do campo deve ser exatamente este na temp-table, para que o ABLScript interprete que se trata de uma view com coluna de comentários e renderize a coluna de comentários.

Passo 4 - Leitura dos comentários

Primeiramente, devemos definir em que evento será feito a leitura dos comentários. Para isto, foi disponibilizado o evento **commentsOpened**. Este evento é disparado no momento em que a popup de comentários é aberta. Na janela de edição do Metadados (Metadados IDE), selecione o grid que conterá os comentários e então selecione o evento de **commentsOpened**. Neste evento, deve ser adicionado um código semelhante a este:

```
def var grid as widget-handle.
def var vHandle as handle.
def var idx as integer.
def var module as character.

def temp-table ttModulDtsul
    field commented as logical
    field codModulDtsul as character label "Módulo"
    field desModulDtsul as character label "Descrição".

def temp-table ttComments no-undo
    field parentRowid as char
    field commentRowid as char
    field commentText as char
    field updateUser as char
    field updateDate as date
    field eventType as char.

ttModulDtsul = grid:query-open().
idx = grid:selected-row().
if  idx > -1 then do:
    ttModulDtsul:set-current(idx).
    module = ttModulDtsul.codModulDtsul.

    run boModulDtsul.p persistent set vHandle.
    run getComments in vHandle (input module, output ttComments).

    setproperty(grid, "comments", ttComments).
end.
```

O código descrito no evento **commentsOpened**, basicamente, verifica se há um registro selecionado e adquire a chave (no caso o módulo) do registro. Executa o programa Progress e a procedure responsável por retornar os comentários relacionados ao módulo. O retorno é então setado no grid, através da

propriedade "*comments*", para que seja apresentado na popup aberta.

Passo 5 - Persistência dos comentários

Após a adição (ou edição) dos comentários, é necessário realizar a persistência deles. Esta etapa é realizada nas seguintes situações: ao clicar em **Adicionar** para se inserir um novo registro, ao clicar em **Salvar** quando se está editando um registro ou ao clicar em **Remover**. Para isto, foi disponibilizado o evento **commentsChanged**. O código abaixo exemplifica como pode ser feita a persistência das informações:

```
def var grid as widget-handle.
def var vHandle as handle.
def var idx as integer.
def var module as character.

def temp-table ttModulDtsul
    field commented as logical
    field codModulDtsul as character label "Módulo"
    field desModulDtsul as character label "Descrição".

def temp-table ttComments no-undo
    field parentRowid as char
    field commentRowid as char
    field commentText as char
    field updateUser as char
    field updateDate as date
    field eventType as char.

ttModulDtsul = grid:query-open().
idx = grid:selected-row().
if  idx > -1 then do:
    ttModulDtsul:set-current(idx).
    module = ttModulDtsul.codModulDtsul.
    ttComments = getproperty(grid, "comments").

    run boModulDtsul.p persistent set vHandle.
    run setComments in vHandle (input module, input-output ttComments).
    setproperty(grid, "comments", ttComments).
end.
```

O código descrito no evento **commentsChanged**, basicamente, verifica se há um registro selecionado e adquire a chave (no caso o módulo) do registro. Busca os comentários manipulados na janela de comentários, populando a temp-table. Com os comentários no ABLScript, o script Progress é executado e então a procedure responsável por persistir as informações.

> **Observação**
>
> Note que a função **setComments** possui um parâmetro de input-output porque após a persistência do comentário o mesmo já possui o rowid do registro pai e já possui o rowid do próprio registro, devendo ser retornado para a tela para uma eventual atualização do comentário que acabou de ser criado.