



PeopleSoft PeopleCode

Desenvolvido por: Giancarlo Fernandes

Data: Junho/2011

Conteúdo

Visão Geral	8
Onde usar um PeopleCode	8
Fluxo do Processador de Componente	8
Record	8
Page	8
Component	8
Menu	9
Integração	9
Workflow	9
Segurança de Signon	9
Application Engine PeopleCode	9
Component Interface	9
Ferramentas de Desenvolvimento de PeopleCode	10
Modos de Conexão	10
Application Designer	10
Área de Trabalho do Projeto	11
Área de Trabalho do Objeto	11
Janela de Saída	11
Configurações do Application Designer	11
Aba:Projeto	11
Aba: Controle de Alteração	12
Aba: Validação	13
Aba: Editores	14
Aba: Geral	15
Aba: ID Proprietário	15
Aba: Imagem	16
Aba: Navegador	16
Passos de Desenvolvimento de Aplicativos PeopleSoft	17
Exercícios	17
Usando as Ferramentas de Desenvolvimento	24
Localizando Programas PeopleCode	24
Record Field	24
Component	24
Page	25
Editor do PeopleCode	25
Características do Editor	25
Exercícios	25
Ferramentas Adicionais de Desenvolvimento	25
Find Definition References	25
Find In	26
Mensagens de tela	26
Limitações	26
MessageBox	26
WinMessage	27
Warning	28
Error	28
Entendendo o Fluxo do Processador de Componente	29
Tipos de Processamento	29
Interactive (interativo)	29
Deferred (atrasado)	29

Interactive ou Deferred	29
Exercícios	29
Carregando Dados no Buffer	30
Adicionando registros.....	30
Onde é possível programar eventos.....	30
Eventos do Fluxo do Processador de Componentes	31
SearchInit	31
SearchSave	31
RowSelect	32
PreBuild.....	32
FieldDefault.....	33
FieldFormula	33
RowInit	33
PostBuild	34
Activate	34
FieldEdit	34
FieldChange	34
SaveEdit	35
SavePreChange	35
Workflow	35
SavePostChange.....	36
RowInsert.....	36
RowDelete	36
Exercícios	36
Modos de Processamento	37
%Mode.....	37
Exercícios	38
Codificando em PeopleCode	39
Comentários de PeopleCode	39
Comentário de Linha.....	39
Comentário de Bloco	39
Comentário de Bloco Aninhado.....	39
Declaração de Variáveis e Constantes.....	40
Tipos de Escopo	40
Tipos de Dados Convencionais	41
Variáveis de Sistema.....	42
%Component	42
%ComplntfcName	42
%Date.....	43
%DateTime.....	43
%EmployeeID	43
%Language	43
%Menu	43
%OperatorID	43
%Page.....	43
%Session	44
%SQLRows.....	44
%Time	44
%UserID	44
Exercícios	44
Tipos de Dados de Objetos	45
Array	45

CreateArray.....	46
Push	46
Pop	47
Unshift	47
Shift.....	47
CreateArrayAny.....	47
CreateArrayRept	48
Exercícios	49
Estruturas Condicionais.....	49
IF Then Else	49
Evaluate	49
Operadores Relacionais.....	50
Operadores Matemáticos.....	50
Ordem de Precedência	50
Operadores Lógicos	50
Ordem de Precedência	50
Estruturas de Repetição	51
For...End-For	51
While...End-While	51
Repeat...Until	51
Tratamento de Erros.....	52
Try...Catch	52
Exercícios	52
Acesso ao Banco de Dados.....	53
SQLExec	53
Exercícios	53
Classe SQL.....	54
GetSQL / CreateSQL.....	54
Fetch	54
Execute.....	55
Close.....	55
Reutilização de Cursores	55
SQL Injection.....	56
Exercícios	56
Meta-SQL.....	56
%Abs	57
%Concat	57
%CurrentDateIn	57
%CurrentDateOut	57
%CurrentDateTimeln	57
%CurrentDateTimeOut	57
%DateAdd	57
%DateDiff.....	58
%DateIn.....	58
%DateNull	58
%DateOut.....	58
%DatePart	58
%DateTimeDiff	58
%DateTimeln.....	58
%DateTimeNull	58
%DateTimeOut.....	59
%Delete.....	59

%DecDiv	59
%DecMult.....	59
%DTTM.....	59
%FirstRows.....	59
%Insert	59
%InsertSelect	60
%InsertValues	60
%Join	60
%KeyEqual.....	60
%NumToChar	60
%Round	61
%Select.....	61
%SelectAll.....	61
%Substring	61
%Table.....	61
%Truncate	61
%Update	62
%Upper	62
Exercícios	62
Classe Record.....	62
CreateRecord	62
GetRecord	63
CopyFieldsTo.....	63
ExecuteEdits.....	63
Delete.....	63
FieldCount.....	63
Insert.....	63
IsChanged.....	64
SelectByKey.....	64
Update	64
Record Derived/Work.....	64
Exercícios	65
Funções Internas	67
Catálogo de Mensagens	67
MsgGet.....	67
MsgGetText.....	67
MsgGetExplainText	67
MessageBox	67
Funções de Validação	68
All	68
None.....	68
AllOrNone	68
OnlyOne	69
OnlyOneOrNone	69
Exercícios	69
Funções de Texto.....	70
Left	70
Right.....	70
Substring.....	70
LTrim	70
RTrim.....	71
Len.....	71

Rept.....	71
Find	71
Char.....	71
String.....	72
Value	72
Funções de Data e Hora	72
Date3.....	72
AddToDate	72
DateTime6.....	72
AddToDateTime	73
Exercícios	73
Criação de Funções Personalizadas.....	75
Categorias de Funções.....	75
Funções Internas PeopleCode	75
Funções Externas PeopleCode.....	75
Funções Externas Não PeopleCode	76
Exercícios	76
Trabalhando com Níveis em uma Página	77
Acessando o Nível 0.....	77
GetLevel0.....	77
Acessando Níveis 1, 2 e 3	77
GetRowset	77
Classe RowSet.....	78
CreateRowset.....	78
GetRowset	78
ActiveRowCount	78
Fill.....	78
Flush.....	78
GetRow	79
HideAllRows	79
RowCount	79
Select.....	79
ShowAllRows.....	79
Sort.....	80
Classe Row	80
GetRecord	80
GetRowset	80
IsChanged.....	80
IsDeleted.....	80
IsNew	80
RecordCount	81
RowNumber	81
Selected	81
Visible.....	81
Classe Field	82
SearchClear	82
SearchDefault	82
SearchEdit	82
SetDefault	82
SetCursorPos.....	82
DisplayOnly	82
Enabled	83

FormattedValue	83
IsChanged.....	83
IsRequired	83
Label.....	83
OriginalValue	83
Value	84
Visible.....	84
Exercícios	84
Classes	85
Application Package	85
Criando uma Classe.....	85
Protected	86
Private	86
%This	86
%Super	87
Usando uma Classe	87
Acessando Métodos e Propriedades de uma Classe	87
Exercícios	88
Numeração Automática	90
Criando Tabela de Setup	90
Codificando a Autonumeração.....	90
GetNextNumberWithGapsCommit.....	90
Migração de Projeto.....	92
Revisando o Projeto Antes de Migrar.....	92
Aba Development	92
Aba Upgrade	92
Não Esqueça de Objetos de Estrutura do Portal	92
Não Migre Segurança	92
Comparando Bases Origem e Destino.....	93
Preparando a Migração	94
Migrando o Projeto	94
Para Outro Banco de Dados.....	94
Para Arquivo	95
Ativando a Segurança na Base de Destino	95
Referências Bibliográficas	96

Visão Geral

PeopleCode é uma linguagem proprietária da PeopleSoft que cobre cada aspecto de aplicativos PeopleSoft.

Com PeopleCode é possível:

- Controlar a exibição de dados para usuários e validar suas digitações
- Executar cálculos e manipular dados
- Manter integridade de dados
- Atualizar dados no banco de dados
- Gerenciar o portal de navegação
- Administrar segurança
- Integrar aplicativos PeopleSoft com aplicativos de terceiros
- Gerenciar workflow

Onde usar um PeopleCode

Um PeopleCode pode ser usado em:

- Fluxo do Processador de Componente: Record, Page, Component e Menu
- Integração
- Workflow
- Segurança de Signon
- Application Engine
- Component Interface

Fluxo do Processador de Componente

O fluxo do processador de componente é quem executa um aplicativo PeopleSoft a partir do momento em que se clica em um item de menu no portal de navegação, ele monta a estrutura da página executando todos os PeopleCodes que existirem em menus, componentes, páginas e records que existirem no aplicativo.

Record

Peoplecodes em records são o mais comum de ocorrer, mas quando uma record está em várias páginas, este peoplecode será executado em todas essas páginas, para restringir a execução de determinado peoplecode a uma página, componente ou menu, é aconselhável movê-lo para os eventos destes objetos para que somente seja executado quando tal página, componente ou menu for acessado.

Page

Peoplecodes em páginas somente são executados quando tal página é executada, mas quando uma página está em vários componentes, esse peoplecode será executado em todos esses componentes, para restringir a execução de determinado peoplecode a um componente ou menu, é aconselhável movê-lo para os eventos destes objetos para que somente seja executado quando tal componente ou menu for acessado.

Component

Peoplecodes em componentes somente são executados quando tal componente é executado, mas quando um componente está em vários menus, esse peoplecode será executado em todos esses menus, para restringir a execução de determinado peoplecode a

um menu, é aconselhável movê-lo para os eventos deste menu para que somente seja executado quando tal menu for acessado.

Menu

Peoplecodes em menus executam somente quando este menu é acessado pelo usuário.

Integração

Pela integração é possível trocar dados com sistemas de terceiros e até mesmo com outros aplicativos PeopleSoft através de mensagens XML.

As mensagens podem ser:

- Estruturadas: são mensagens baseadas em records, formando um rowset, e serve tanto para envio, quanto recebimento de mensagens.
- Não estruturadas: são mensagens organizadas em qualquer estrutura diferente de records, neste tipo de estrutura os dados devem estar em formato HTML ou SOAP.

As formas de transmissão de mensagens podem ser:

- Assíncronas: As mensagens são enviadas e processadas no destino que por sua vez não retorna nenhum tipo de mensagem.
- Síncronas: As mensagens são enviadas e processadas no destino que por sua vez retorna outra mensagem indicando algum resultado do processamento, que pode ser uma confirmação se o processo foi executado com sucesso ou erro.

Workflow

O roteamento workflow fornece duas funções PeopleCode que são VirtualApprover e VirtualRouter que facilitam a funcionalidade do processo de regra de aprovação.

Segurança de Signon

O PeopleCode de Signon roda sempre que um usuário efetua login no PeopleSoft através do Pure Internet Architecture (PIA), portal ou estação de trabalho em modo 3 camadas.

Application Engine PeopleCode

Application Engine é uma ferramenta que permite desenvolver aplicativos que rodam em processamento batch.

Ele executa um conjunto de PeopleCode e SQL que definem uma regra de negócio, é muito usado em tarefas que demandam muito tempo na execução, como por exemplo, na geração/leitura de arquivos grandes ou tarefas que devem ser agendadas para rodar em determinados momentos

Component Interface

Component Interface é quem externaliza acesso a componentes internos PeopleSoft fornecendo sincronização em tempo real às regras de negócio e aos dados associados ao componentes.

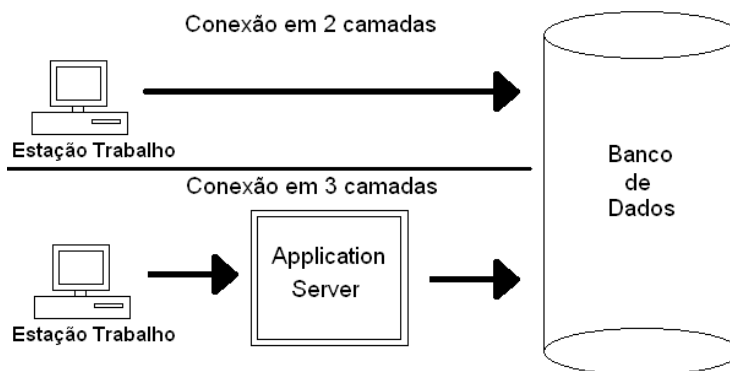
Um Component Interface mapeia um PeopleSoft Component e pode ser visto como uma “caixa preta” que encapsula dados e regras de negócio escondendo a complexidade da implementação e estrutura dos dados.

Ferramentas de Desenvolvimento de PeopleCode

A principal ferramenta de desenvolvimento de Peoplecode é o Application Designer.

Modos de Conexão

O Application Designer pode ser usado em modo 2 camadas ou 3 camadas:

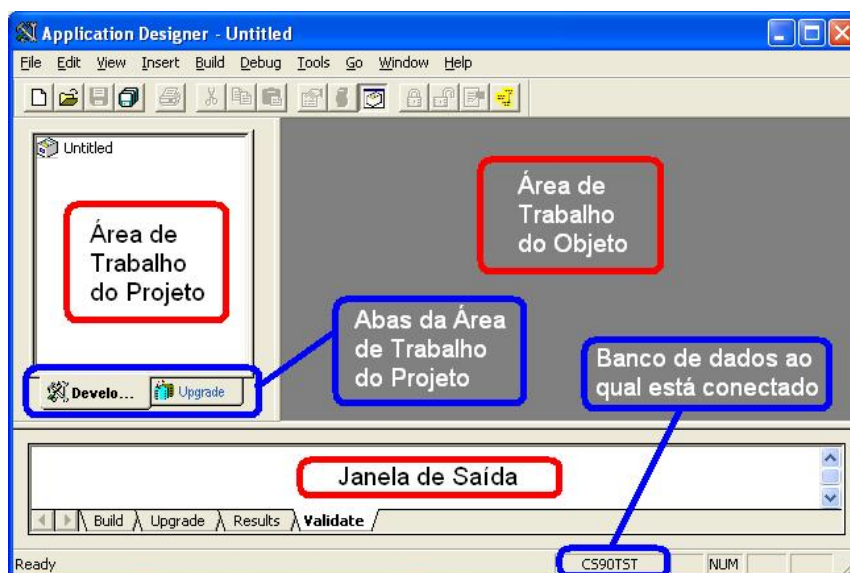


Application Designer

O Application Designer é um ambiente de desenvolvimento integrado que permite aos desenvolvedores PeopleSoft trabalharem com várias definições de objetos em uma única área de trabalho.

A principal vantagem do acesso a todas as definições de desenvolvimento em uma ferramenta é que isso permite que o desenvolvedor PeopleSoft visualize todos os relacionamentos entre as várias partes de um aplicativo, facilitando o processo de desenvolvimento.

O Application Designer organiza as definições de objeto de um aplicativo de negócios em uma Área de Trabalho do Objeto, uma Área de Trabalho do Projeto e uma Janela de Saída



Área de Trabalho do Projeto

Todas as definições associadas ao seu projeto ficam agrupadas neste local de modo organizado, facilitando encontrar qualquer definição facilmente, você pode adicionar ou remover definições do seu projeto a qualquer momento.

Existem 2 abas nesta área de trabalho:

- Development: é onde você encontra todas as definições existentes no seu projeto.
- Upgrade: é onde você define como as definições do seu projeto serão migradas para outro banco de dados.

Área de Trabalho do Objeto

Ao abrir uma definição de objeto, é neste local que é possível ver e editar todas as propriedades desse objeto.

Janela de Saída

Esta janela permite ver resultados de várias operações que são realizadas dentro do seu ambiente do Application Designer.

Configurações do Application Designer

Para acessar as configurações, acesse o menu TOOLS, e clique na opção OPTIONS, abrirá uma nova janela contendo as seguinte abas:

- Projeto
- Controle de Alteração
- Validação
- Editores
- Geral
- ID Proprietário
- Imagem
- Navegador

Aba:Projeto

Grupo Inserir Definição no Projeto:

- Quando a definição for aberta: insere automaticamente no seu projeto qualquer definição que você abrir.
- Quando a definição for modificada e salva, ou excluída: insere automaticamente no seu projeto qualquer definição que você alterar e salvar ou excluir.
- Manualmente: Não insere automaticamente nenhuma modificação no seu projeto, qualquer definição que você queira adicionar deverá ser feita através do menu Insert.

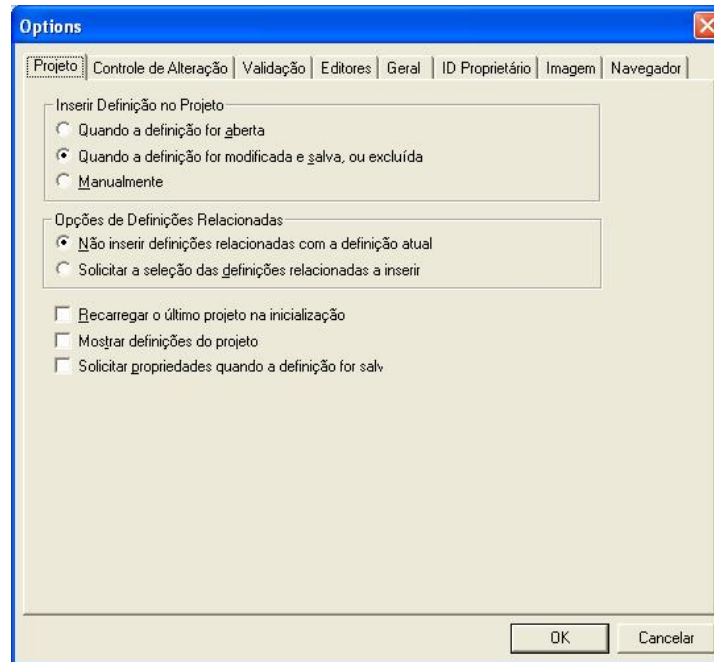
Grupo Opções de Definições Relacionadas:

- Não inserir definições relacionadas com a definição atual: não adiciona nenhuma definição relacionada ao projeto, neste caso, somente a definição em questão é inserida.
- Solicitar a seleção das definições relacionadas a inserir: após inserir uma definição no projeto, uma caixa de diálogo abrirá para você escolher e decidir se deve inserir também definições relacionadas àquela que você acabou de inserir.

Outras opções:

- Recarregar o último projeto na inicialização: Quando marcada, faz com que o Application Designer ao ser aberto, automaticamente carregue o projeto que estava aberto na sessão anterior.

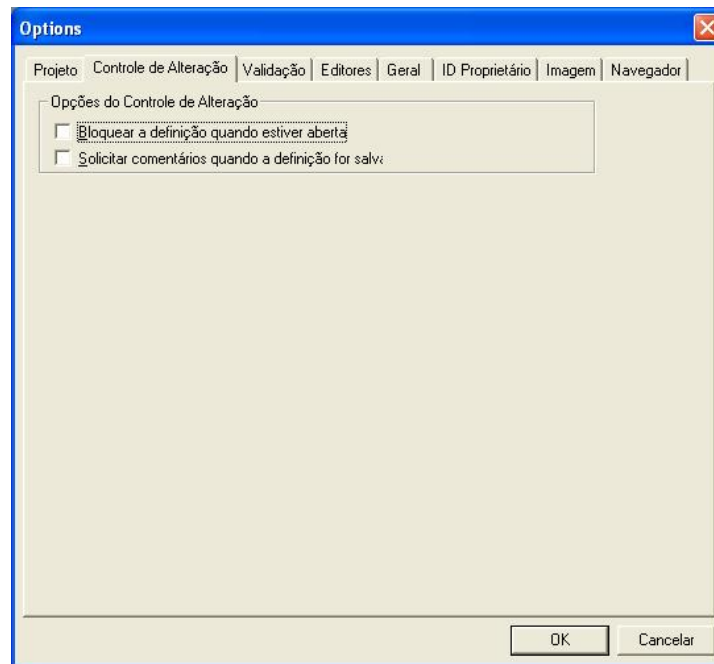
- Mostrar definições do projeto: exibe um pequeno ponto preto no ícone de cada definição que faça parte do projeto, aquelas definições adicionadas, mas ainda não salvas ao projeto, não mostrarão esse ponto preto.
- Solicitar propriedades quando a definição for salva: abre a caixa de diálogo das definições que estão sendo salvas para que sejam digitadas as propriedades da definição.



Aba: Controle de Alteração

Grupo Opções do Controle de Alteração:

- Bloquear a definição quando estiver aberta: automaticamente bloqueia a definição quando estiver aberta.
- Solicitar comentários quando a definição for salva: ao salvar, uma caixa de diálogo abrirá para que sejam preenchidos comentários sobre a alteração que foi feita.



Aba: Validação

Grupo Opções:

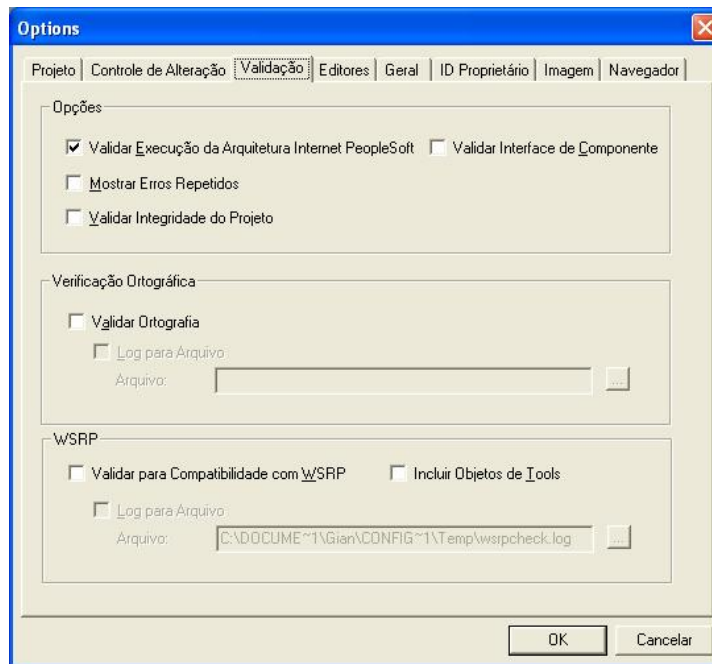
- Validar execução da Arquitetura Internet PeopleSoft: valida se o projeto está em conformidade com a arquitetura internet PeopleSoft.
- Validar Interface de Componente: Valida definições Component Interface, se houverem no seu projeto.
- Mostrar Erros Repetidos: Exibe somente uma vez cada erro encontrado ou exibe repetidamente todos os erros encontrados.
- Validar Integridade do Projeto: Valida se o projeto está íntegro.

Grupo Verificação Ortográfica:

- Validar Ortografia: Valida a ortografia em todos os textos, labels, translates, etc, o resultado será enviado para um arquivo de log, onde poderá ser consultado facilmente.

Grupo WSRP (Web Services for Remote Portlets):

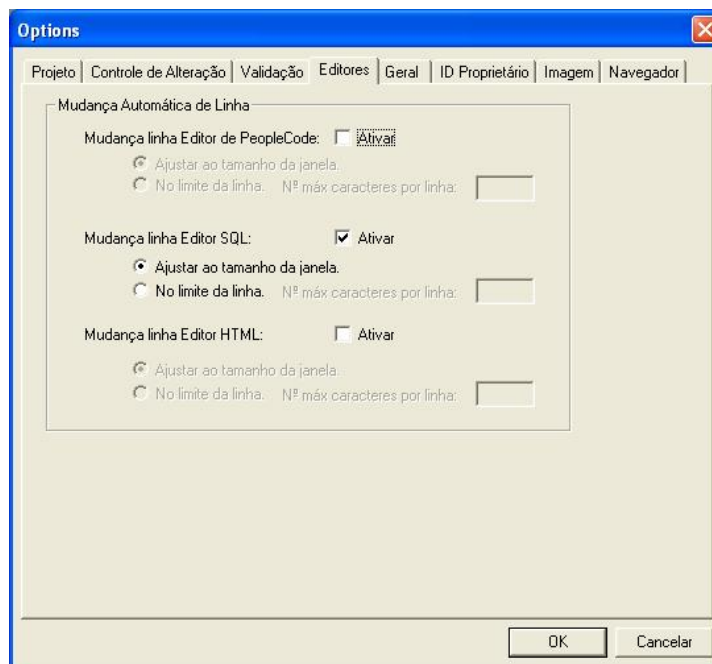
- Validar para Compatibilidade com WSRP: Valida se o projeto está compatível o WSRP.
- Incluir Objetos de Tools: Verifica definições criadas em outras ferramentas também estão compatíveis com o WSRP.



Aba: Editores

Grupo Mudança Automática de Linha:

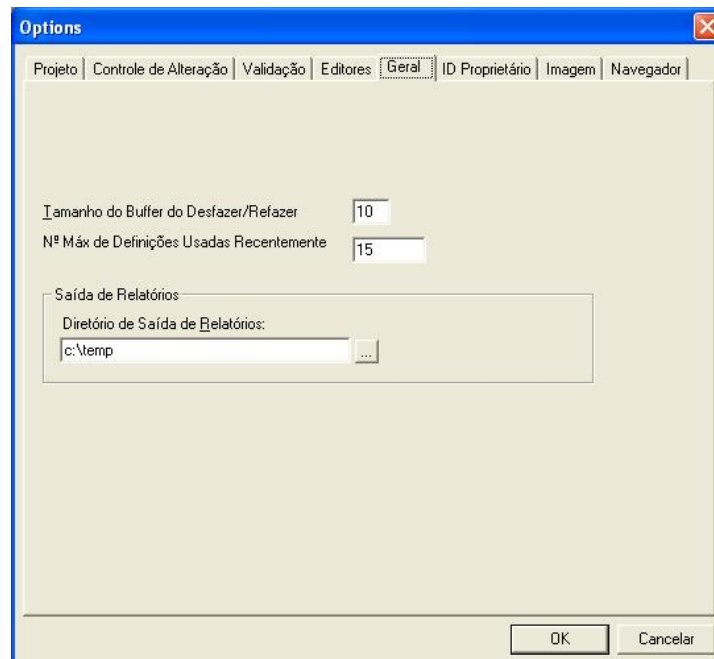
- Mudança linha Editor de PeopleCode: Quando ativado, faz com textos de PeopleCode à direita que estiverem ocultos pelo tamanho da janela, sejam exibidos na linha de baixo.
- Mudança linha Editor SQL: Quando ativado, faz com textos de códigos SQL à direita que estiverem ocultos pelo tamanho da janela, sejam exibidos na linha de baixo.
- Mudança linha Editor HTML: Quando ativado, faz com textos de códigos HTML à direita que estiverem ocultos pelo tamanho da janela, sejam exibidos na linha de baixo.



Aba: Geral

Opções:

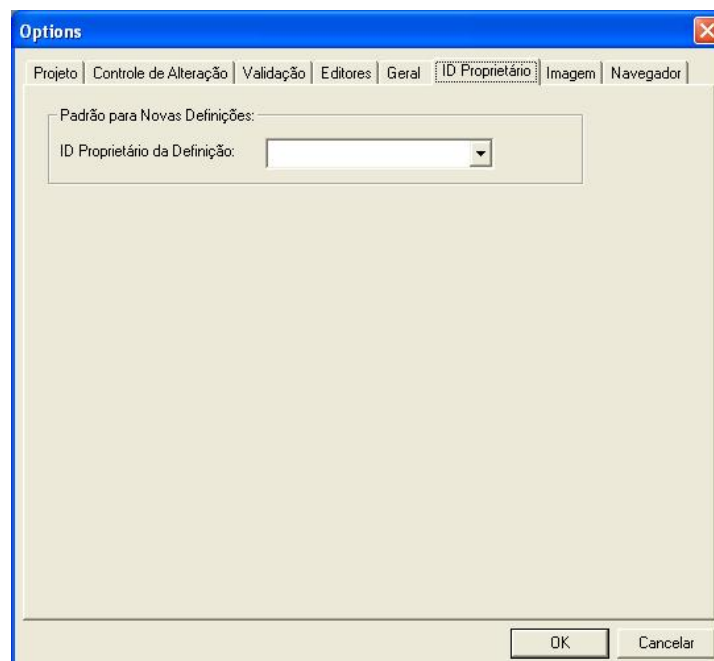
- Tamanho do Buffer do Desfazer/Refazer: Número máximos de alterações que ficam armazenadas no buffer de desfazer/refazer.
- Nº Máx. de Definições Usadas Recentemente: Exibe as últimas “N” definições no menu File, para acesso rápido.
- Diretório de Saída de Relatórios: Onde os logs serão gerados.



The screenshot shows the 'Options' dialog box with the 'Geral' tab selected. The dialog has a title bar with a close button. Below the title bar is a tabbed interface with tabs: 'Projeto', 'Controle de Alteração', 'Validação', 'Editores', 'Geral' (selected), 'ID Proprietário', 'Imagem', and 'Navegador'. The main area contains three settings: 'Tamanho do Buffer do Desfazer/Refazer' with a text box containing '10', 'Nº Máx. de Definições Usadas Recentemente' with a text box containing '15', and a section titled 'Saída de Relatórios' containing a label 'Diretório de Saída de Relatórios:' and a text box with 'c:\temp' and a browse button (...). At the bottom right are 'OK' and 'Cancelar' buttons.

Aba: ID Proprietário

- ID Proprietário da Definição: Indica a que se refere as novas definições.

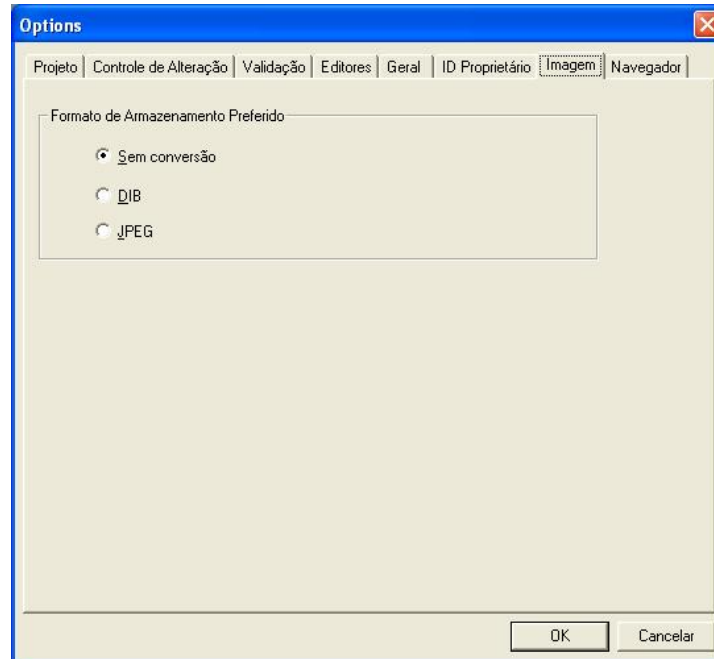


The screenshot shows the 'Options' dialog box with the 'ID Proprietário' tab selected. The dialog has the same title bar and tabbed interface as the previous screenshot, with the 'ID Proprietário' tab now selected. The main area contains a section titled 'Padrão para Novas Definições:' with a label 'ID Proprietário da Definição:' and a dropdown menu. At the bottom right are 'OK' and 'Cancelar' buttons.

Aba: Imagem

Opções:

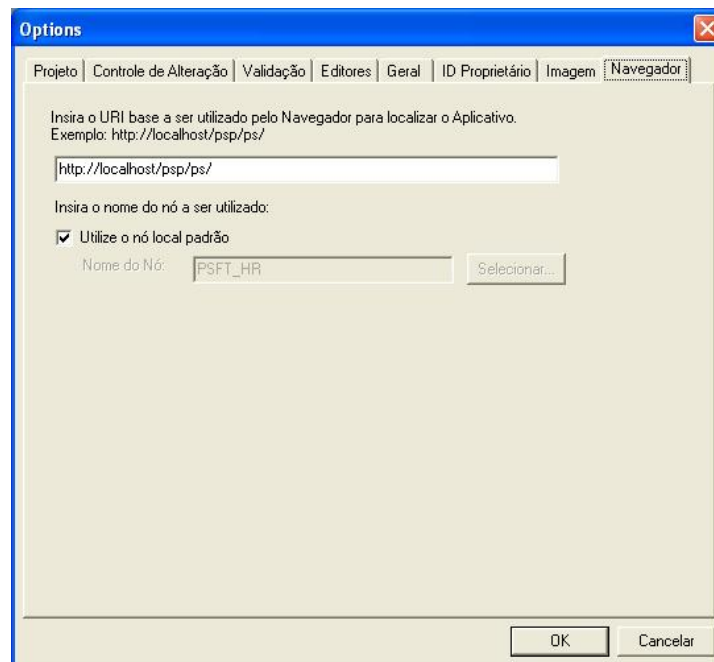
- Formato de Armazenamento Preferido: Indica se imagens devem ser salvar no formato DIB, JPEG, ou no formato original.



Aba: Navegador

Opções:

- URI: URL que o Application Designer deve usar para acessar o PeopleSoft.
- Utilize o nó local padrão: se desmarcado, permite escolher o nó das definições.



Passos de Desenvolvimento de Aplicativos PeopleSoft

Para desenvolver um aplicativo PeopleSoft são necessários pelo menos 8 etapas:

- 1) Planejamento
- 2) Definição de Fields
- 3) Definição de Records
- 4) Build
- 5) Definição de Pages
- 6) Definição de Components
- 7) Definição de Menus
- 8) Definição de segurança
- 9) Testar o aplicativo

Vamos relembrar todos os passos desenvolvendo um pequeno aplicativo que nos ajudará a seguir o conteúdo desta apostila de PeopleCode.

Exercícios

A instituição de ensino “Faculdade S.A.” possui 2 campi (Zona Sul e Zona Norte) e cada campus possui a sua própria biblioteca. A instituição precisa de um aplicativo que gerencie a locação de livros pelos alunos em cada biblioteca.

Para nosso exercício, criaremos 3 páginas de cadastro:

- Cadastro de alunos
- Cadastro de livros
- Controlar as locações de livros

Também criaremos 2 páginas de relatórios:

- Livros que um aluno já locou
- A lista dos 5 livros mais alugados em cada campus

Como regra de locação, temos que um aluno somente poderá alugar livros da biblioteca do campus no qual estuda.

- 1) Abra o Application Designer
- 2) Crie um projeto novo
- 3) Acesse o menu File / Project Properties
- 4) Digite no campo Description: “Exercicio PeopleCode” e clique em OK
- 5) Clique no botão Salvar e dê o nome EXERCICIO_PEOPLECODE ao seu projeto
- 6) Crie a record abaixo:

Field	Field Label	Keys	Required	Edit Type
EMPLID	STUDENT_ID	Key Search Key List Box Item	Sim	
INSTITUTION	INSTITUTION1	Alternate Search Key List Box Item	Sim	Prompt Table Edit INSTITUTION_TBL
CAMPUS	CAMPUS	Alternate Search Key List Box Item	Sim	Prompt Table Edit CAMPUS_TBL
NAME_DISPLAY	NAME2	Alternate Search Key List Box Item	Sim	
ADDRESS1	ADDRESS		Sim	
CITY	(rótulo padrão)		Sim	
STATE	(rótulo padrão)		Sim	

7) Salve a record com o nome EXERC_ALUNO

8) Crie a definição de field abaixo:

Field Type	Label ID	Long Name	Short Name	Format Type
Character (3)	ASSUNTO	Assunto do Livro	Assunto	Uppercase

9) Adicione os valores de tradução:

Field Value	Effective Date	Long Name	Short Name
ADM	01/01/1900	Administração de Empresas	Adm. Emp.
CC	01/01/1900	Ciências da Computação	Cie. Comp.
BIO	01/01/1900	Biologia	Biologia
FIS	01/01/1900	Física	Física
GEO	01/01/1900	Geografia	Geografia

10) Salve o field com o nome EXERC_ASSUNTO

11) Crie a definição de field abaixo:

Field Type	Label ID	Long Name	Short Name	Format Type
Character (1)	TEMPO_MAXIMO	Tempo Máximo de Locação	Tempo Máx	Uppercase

12) Adicione os valores de tradução:

Field Value	Effective Date	Long Name	Short Name
A	01/01/1900	2 dias	2 dias
B	01/01/1900	1 semana	1 semana
C	01/01/1900	2 semanas	2 semanas
D	01/01/1900	1 mês	1 mês
E	01/01/1900	2 meses	2 meses

13) Salve o field com o nome EXERC_TEMPO_MAX

14) Crie a record abaixo:

Field	Field Label	Keys	Required	Edit Type	Default
BOOK_CODE	BOOK_ID	Key Search Key List Box Item	Sim		
INSTITUTION	INSTITUTION1	Altern Search Key List Box Item	Sim	Prompt Table INSTITUTION_TBL	
CAMPUS	CAMPUS	Altern Search Key List Box Item	Sim	Prompt Table CAMPUS_TBL	
TITLE	TITLE	Altern Search Key List Box Item	Sim		
EXERC_ASSUNTO	(rótulo padrão)	Altern Search Key List Box Item	Sim	Translate	
EXERC_TEMPO_MAX	(rótulo padrão)		Sim	Translate	E
AMOUNT_SF	(rótulo padrão)		Sim		0
RATE_AMOUNT	(rótulo padrão)		Sim		2

15) Salve a record com o nome EXERC_LIVRO

16) Crie a record abaixo:

Field	Field Label	Keys	Required	Edit Type	Default
INSTITUTION	INSTITUTION1	Key Search Key List Box Item	Sim	Prompt Table INSTITUTION_TBL	
CAMPUS	CAMPUS	Key Search Key List Box Item	Sim	Prompt Table CAMPUS_TBL	
BOOK_CODE	BOOK_ID	Key Search Key List Box Item	Sim	Prompt Table EXERC_LIVRO	
EMPLID	STUDENT_ID	Key Search Key List Box Item	Sim	Prompt Table EXERC_ALUNO	

DATE_FROM	DATE_FROM	Key Descending Key List Box Item	Sim		%date
DATE_TO	DATE_TO		Sim		

- 17) Salve a record com o nome EXERC_LOCACAO
- 18) Execute o Build para cada record que você criou
- 19) Crie a página abaixo:

EXERC_ALUNO (Página)

Layout Order

ID Aluno: NNNNNNNN

Instituição: *

Campus: *

Nome:

Endereço:

Cidade:

Estado:

EXERC_ALUNO (Página)

Layout Order

	ID	Lv	Label	Type	Field	Record	Display Control	Related Field	Control Field
1	1	0	ID Aluno	Edit Box	EMPLID	EXERC_ALUNO	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
2	2	0	Instituição	Edit Box	INSTITUTION	EXERC_ALUNO	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
3	8	0	Descrição	Edit Box	DESCR	INSTITUTION_TBL	<input type="checkbox"/>	<input checked="" type="checkbox"/>	2
4	3	0	Campus	Edit Box	CAMPUS	EXERC_ALUNO	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
5	10	0	Descrição	Edit Box	DESCR	CAMPUS_TBL	<input type="checkbox"/>	<input checked="" type="checkbox"/>	4
6	4	0	Nome	Edit Box	NAME_DISPLAY	EXERC_ALUNO	<input type="checkbox"/>	<input type="checkbox"/>	
7	5	0	Endereço	Edit Box	ADDRESS1	EXERC_ALUNO	<input type="checkbox"/>	<input type="checkbox"/>	
8	6	0	Cidade	Edit Box	CITY	EXERC_ALUNO	<input type="checkbox"/>	<input type="checkbox"/>	
9	7	0	Estado	Edit Box	STATE	EXERC_ALUNO	<input type="checkbox"/>	<input type="checkbox"/>	

- 20) Crie a página abaixo:

EXERC_LIVRO (Página)

Layout Order

Código do Livro: NNNN

Instituição: *

Campus: *

Título:

Assunto:

Tempo Máx.:

Valor da Locação:

Valor Multa p/ Dia Atraso:

EXERC_LIVRO (Página)

Layout Order

	ID	Lv	Label	Type	Field	Record	Display Control	Related Field	Control Field	Pa
1	1	0	Código do Livro	Edit Box	BOOK_CODE	EXERC_LIVRO	<input checked="" type="checkbox"/>	<input type="checkbox"/>		
2	2	0	Instituição	Edit Box	INSTITUTION	EXERC_LIVRO	<input checked="" type="checkbox"/>	<input type="checkbox"/>		
3	9	0	Descrição	Edit Box	DESCR	INSTITUTION_TBL	<input type="checkbox"/>	<input checked="" type="checkbox"/>	2	
4	3	0	Campus	Edit Box	CAMPUS	EXERC_LIVRO	<input checked="" type="checkbox"/>	<input type="checkbox"/>		
5	10	0	Descrição	Edit Box	DESCR	CAMPUS_TBL	<input type="checkbox"/>	<input checked="" type="checkbox"/>	4	
6	4	0	Título	Edit Box	TITLE	EXERC_LIVRO	<input type="checkbox"/>	<input type="checkbox"/>		
7	5	0	Assunto	Drop Down Lst	EXERC_ASSUNTO	EXERC_LIVRO	<input type="checkbox"/>	<input type="checkbox"/>		
8	6	0	Tempo Máx.	Drop Down Lst	EXERC_TEMPO_MAX	EXERC_LIVRO	<input type="checkbox"/>	<input type="checkbox"/>		
9	7	0	Valor da Locação	Edit Box	AMOUNT_SF	EXERC_LIVRO	<input type="checkbox"/>	<input type="checkbox"/>		
10	8	0	Valor Multa p/ Dia Atraso	Edit Box	RATE_AMOUNT	EXERC_LIVRO	<input type="checkbox"/>	<input type="checkbox"/>		

- 21) Crie a página abaixo:

EXERC_LOCACAO (Página)

Layout Order

Instituição: NNNNN : AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Campus: NNNNN : AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Código do Livro: NNNN : AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

ID Aluno: NNNNNNNN : AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Data Inicial: []

Data Final: []

Tempo Máximo de Locação: []

Valor da Locação: -22222222

Valor Multa p/ Dia Atraso: -22222,22

EXERC_LOCACAO (Página)

Layout Order

ID	Lvl	Label	Type	Field	Record	Display Control	Related Field	Control Field
1	0	Instituição	Edit Box	INSTITUTION	EXERC_LOCACAO	<input checked="" type="checkbox"/>		
2	7	Descrição	Edit Box	DESCR	INSTITUTION_TBL	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	1
3	2	Campus	Edit Box	CAMPUS	EXERC_LOCACAO	<input checked="" type="checkbox"/>		
4	8	Descrição	Edit Box	DESCR	CAMPUS_TBL	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	3
5	3	Código do Livro	Edit Box	BOOK_CODE	EXERC_LOCACAO	<input checked="" type="checkbox"/>		
6	9	Título	Edit Box	TITLE	EXERC_LIVRO	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	5
7	4	ID Aluno	Edit Box	EMPLID	EXERC_LOCACAO	<input checked="" type="checkbox"/>		
8	10	Nome	Edit Box	NAME_DISPLAY	EXERC_ALUNO	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	7
9	5	Data Inicial	Edit Box	DATE_FROM	EXERC_LOCACAO	<input checked="" type="checkbox"/>		
10	6	Data Final	Edit Box	DATE_TO	EXERC_LOCACAO	<input checked="" type="checkbox"/>		
11	13	Tempo Máximo de Locação	Drop Down Lst	EXERC_TEMP_MAX	EXERC_LIVRO	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	5
12	11	Valor da Locação	Edit Box	AMOUNT_SF	EXERC_LIVRO	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	5
13	12	Valor Multa p/ Dia Atraso	Edit Box	RATE_AMOUNT	EXERC_LIVRO	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	5

- 22) Crie o componente abaixo, colocando EXERC_ALUNO como Search Record e default search como Advanced:

EXERC_ALUNO.GBL (Componente)

Definition Structure

Page Name	Item Name	Hidden	Item Label	Folder Tab Label	Allow Deferred Processing
EXERC_ALUNO	EXERC_ALUNO	<input type="checkbox"/>	Cadastro de Alunos		<input checked="" type="checkbox"/>

- 23) Crie o componente abaixo, colocando EXERC_LIVRO como Search Record e default search como Advanced:

EXERC_LIVRO.GBL (Componente)

Definition Structure

Page Name	Item Name	Hidden	Item Label	Folder Tab Label	Allow Deferred Processing
EXERC_LIVRO	EXERC_LIVRO	<input type="checkbox"/>	Cadastro de Livros		<input checked="" type="checkbox"/>

- 24) Crie o componente abaixo, colocando EXERC_LOCACAO como Search Record e default search como Advanced:

EXERC_LOCACAO.GBL (Componente)

Definition Structure

Page Name	Item Name	Hidden	Item Label	Folder Tab Label	Allow Deferred Processing
EXERC_LOCACAO	EXERC_LOCACAO	<input type="checkbox"/>	Controle de Locações		<input checked="" type="checkbox"/>

- 25) Registre cada componente usando:
- Nome do Menu: TREINAMENTO
 - Nome da Barra: USE
 - Nome da Pasta: TREINAMENTO
 - Nº Seqüência: 10 (aluno), 20 (livro) e 30 (locação)
 - Lista de Permissão: TREINAMENTO
- 26) Teste seu aplicativo cadastrando pelo menos a quantidade de itens abaixo para que possamos ver funcionando os relatórios que serão criados a seguir
- 6 livros
 - 6 alunos
 - 10 locações

27) Crie a record do tipo view que trará os livros alugados por um aluno:

Field	Field Label	Keys	Required	Edit Type	Default
INSTITUTION	INSTITUTION1	Key Search Key List Box Item	Não	Prompt Table INSTITUTION_TBL	
CAMPUS	CAMPUS	Key Search Key List Box Item	Não	Prompt Table CAMPUS_TBL	
EMPLID	STUDENT_ID	Key Search Key List Box Item	Não	Prompt Table EXERC_ALUNO	
NAME_DISPLAY	NAME2	Alternate Search Key List Box Item	Não		
BOOK_CODE	BOOK_ID	Key	Não		
TITLE	TITLE		Não		
AMOUNT_SF	(rótulo padrão)		Não		

Código SQL:

```

Select  A.INSTITUTION
        , A.CAMPUS
        , A.EMPLID
        , B.NAME_DISPLAY
        , C.BOOK_CODE
        , C.TITLE
        , C.AMOUNT_SF
From    PS_EXERC_LOCACAO A
        Inner Join PS_EXERC_ALUNO B
            On B.INSTITUTION = A.INSTITUTION
            And B.CAMPUS      = A.CAMPUS
            And B.EMPLID      = A.EMPLID
        Inner Join PS_EXERC_LIVRO C
            On C.INSTITUTION = A.INSTITUTION
            And C.CAMPUS      = A.CAMPUS
            And C.BOOK_CODE   = A.BOOK_CODE

```

28) Salve a record com o nome EXERC_LIVALU_VW

29) Crie a página que irá apresentar o relatório:

EXERC_LIVALU_VW (Página)

Layout | Order

Instituição: NNNN | AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Campus: NNNN | AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Dados do Aluno

ID Aluno: NNNN | AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Endereço: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Cidade: AAAAAAAAAAAAAAAAAAAAAA | Estado: NNNN

	Código do Livro	Título	Valor
1	1	1	1
2	2	2	2

EXERC_LIVALU_VW (Página)										
Layout		Order								
	ID	Lv	Label	Type	Field	Record	Display Control	Related Field	Control Field	Page Field Name
1	1	0	Instituição	Edit Box	INSTITUTION	EXERC_LIVALU_VW	<input checked="" type="checkbox"/>			
2	5	0	Descrição	Edit Box	DESCR	INSTITUTION_TBL	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	1	
3	2	0	Campus	Edit Box	CAMPUS	EXERC_LIVALU_VW	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
4	6	0	Descrição	Edit Box	DESCR	CAMPUS_TBL	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	3	
5	11	1	Dados do Aluno	Scroll Area			<input checked="" type="checkbox"/>			ALUNOS
6	3	1	ID Aluno	Edit Box	EMPLID	EXERC_LIVALU_VW	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
7	4	1	Nome	Edit Box	NAME_DISPLAY	EXERC_LIVALU_VW	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
8	12	1	Endereço	Edit Box	ADDRESS1	EXERC_ALUNO	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	6	
9	13	1	Cidade	Edit Box	CITY	EXERC_ALUNO	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	6	
10	14	1	Estado	Edit Box	STATE	EXERC_ALUNO	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	6	
11	7	2	Livros que Alugou	Grid		EXERC_LIVALU_VW	<input checked="" type="checkbox"/>			LIVROS
12	8	2	Código do Livro	Edit Box	BOOK_CODE	EXERC_LIVALU_VW	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
13	9	2	Título	Edit Box	TITLE	EXERC_LIVALU_VW	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
14	10	2	Valor	Edit Box	AMOUNT_SF	EXERC_LIVALU_VW	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		

30) Crie o componente abaixo, colocando EXERC_LIVALU_VW como Search Record e default search como Advanced, marque somente os botões *Return to List*, *Next in List* e *Previous in List*:

EXERC_LIVALU_VW.GBL (Componente)					
Definition		Structure			
	Page Name	Item Name	Hidden	Item Label	Folder Tab Label
1	EXERC_LIVALU_VW	EXERC_LIVALU_VW	<input type="checkbox"/>	Livros que o Aluno Alugou	<input checked="" type="checkbox"/>

- 31) Registre o componente
- Nome do Menu: TREINAMENTO
 - Nome da Barra: USE
 - Nome da Pasta: TREINAMENTO
 - Nº Sequência: 40
 - Lista de Permissão: TREINAMENTO

32) Teste a nova página no seu aplicativo

33) Crie a record do tipo view que trará os livros mais alugados por campus:

Field	Field Label	Keys	Required	Edit Type	Default
INSTITUTION	INSTITUTION1	Key Search Key List Box Item	Não	Prompt Table INSTITUTION_TBL	
CAMPUS	CAMPUS	Key Search Key List Box Item	Não	Prompt Table CAMPUS_TBL	
BOOK_CODE	BOOK_ID	Key	Não		
TITLE	TITLE		Não		
QUANTITY	(rótulo padrão)		Não		

Código SQL:

```

Select A.INSTITUTION
      , A.CAMPUS
      , A.BOOK_CODE
      , B.TITLE
      , Count(1)
From   PS_EXERC_LOCACAO A
       Inner Join PS_EXERC_LIVRO B
               On B.INSTITUTION = A.INSTITUTION
               And B.CAMPUS      = A.CAMPUS
               And B.BOOK_CODE   = A.BOOK_CODE
Group By A.INSTITUTION
        , A.CAMPUS
        , A.BOOK_CODE
        , B.TITLE

```

34) Salve a record com o nome EXERC_MAIS_VW

35) Crie a página que irá apresentar o relatório:

EXERC_MAIIS_VW (Página)

Layout Order

Instituição: NNNNNN AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Campus: NNNNNN AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

	Código do Livro	Título	Quantidade
1			
2			

EXERC_MAIIS_VW (Página)

Layout Order

	ID	Lvl	Label	Type	Field	Record	Display Control	Related Field	Control Field	Page Field Name
1	1	0	Instituição	Edit Box	INSTITUTION	EXERC_MAIIS_VW	<input checked="" type="checkbox"/>			
2	4	0	Descrição	Edit Box	DESCR	INSTITUTION_TBL	<input type="checkbox"/>	<input checked="" type="checkbox"/>	1	
3	2	0	Campus	Edit Box	CAMPUS	EXERC_MAIIS_VW	<input checked="" type="checkbox"/>			
4	5	0	Nome Fictício	Edit Box	DESCR	CAMPUS_TBL	<input type="checkbox"/>	<input checked="" type="checkbox"/>	3	
5	3	1	Livros Mais Alugados	Grid		EXERC_MAIIS_VW	<input type="checkbox"/>			EXERC_MAIIS_VW
6	6	1	Código do Livro	Edit Box	BOOK_CODE	EXERC_MAIIS_VW	<input type="checkbox"/>			
7	7	1	Título	Edit Box	TITLE	EXERC_MAIIS_VW	<input type="checkbox"/>			
8	8	1	Quantidade	Edit Box	QUANTITY	EXERC_MAIIS_VW	<input type="checkbox"/>			

36) Crie o componente abaixo, colocando EXERC_MAIIS_VW como Search Record e default search como Advanced, marque somente os botões *Return to List*, *Next in List* e *Previous in List*:

EXERC_MAIIS_VW.GBL (Componente)

Definition Structure

	Page Name	Item Name	Hidden	Item Label	Folder Tab Label	Allow Deferred Processing
1	EXERC_MAIIS_VW	EXERC_MAIIS_VW	<input type="checkbox"/>	Os 5 Livros Mais Alugados		<input checked="" type="checkbox"/>

- 37) Registre o componente
- Nome do Menu: TREINAMENTO
 - Nome da Barra: USE
 - Nome da Pasta: TREINAMENTO
 - Nº Sequência: 50
 - Lista de Permissão: TREINAMENTO
- 38) Teste a nova página no seu aplicativo

Usando as Ferramentas de Desenvolvimento

Localizando Programas PeopleCode

Programas PeopleCode ficam dentro de eventos de definições e são disparados conforme sua função.

Cada evento de uma definição de ter ou não um programa PeopleCode.

Podemos associar um PeopleCode nas seguintes definições:

Record Field

Primeiro, vamos entender a diferença entre *Field*, e *Record Field* e *Component Record Field*:

- Field: é uma definição isolada cujas propriedades podem ser compartilhadas entre várias records, suas propriedades quando alteradas afetarão todas as records que possuem este field.
- Record Field: pertence à definição de Record que o inclui, suas propriedades quando alteradas afetarão todas as páginas e componentes que possuem esta record.
- Component Record Field: é um *record field* usado dentro de um componente. Esta definição aparece na aba *Structure* de um componente, suas propriedades quando alteradas afetarão somente a record onde está inserido.

É possível acessar o PeopleCode de um record field das seguintes formas:

- Na área do trabalho do projeto, ao expandir uma definição de record, o campo que tiver PeopleCode poderá ser expandido também, dentro dele todos os eventos que possuem PeopleCode serão exibidos. Para ver o PeopleCode basta um duplo clique no nome do evento.
- Clicar com botão direito num campo dentro de uma record e escolher a opção *View PeopleCode*.
- Selecionar o botão de exibição de PeopleCode na barra de ferramentas e efetuar um duplo clique na célula de interseção entre *field* e *event*.
- Selecionar um campo e escolher a propriedade View PeopleCode a partir do menu principal.
- Dentro de uma página, clicar com botão direito num campo e escolher *View PeopleCode*.

Component

PeopleCodes de components são associados a uma definição de componente e a um evento.

É possível acessar o PeopleCode de um componente das seguintes formas:

- Na área do trabalho do projeto, clicar com o botão direito no nome do componente e escolher *View PeopleCode*.
- Abrir um componente e selecionar a aba *Structure*.
- Clicar com botão direito num campo dentro de uma record e escolher a opção *View PeopleCode*.
- Dentro de uma definição de componente, ao expandir os níveis de rolagem, os campos que tiverem PeopleCode terão um símbolo de um raio ao lado do nome da definição.

Page

É possível acessar o PeopleCode de uma página das seguintes formas:

- Na área do trabalho do projeto, clicar com o botão direito no nome da página e escolher *View PeopleCode*.
- Efetuar um clique com botão direito em qualquer lugar de uma página e escolher a opção *View PeopleCode*.
- Selecionar uma página na área de trabalho do projeto e ir no menu Editar do Application Designer e então escolher a opção *View PeopleCode*.

Editor do PeopleCode

Qualquer tentativa de acessar um PeopleCode automaticamente abre o editor do PeopleCode.

A janela do editor se divide em 3 partes:

- No topo, a metade da esquerda indica o nome da definição
- No topo, a metade da direita indica o nome do evento
- Abaixo do topo, fica o próprio PeopleCode

Características do Editor

- O editor automaticamente formata o código digitado
- Valida a sintaxe e exibe um relatório de erros de sintaxe
- Permite o recurso de arrastar e soltar objetos
- Permite abrir qualquer definição, bastando efetuar um clique com botão direito em cima do nome da definição
- Fornece ajuda sensível ao contexto
- Possui o recurso de desfazer
- Editor possui código colorido
 - Palavras chave, funções internas e definições de classes são azuis
 - Textos entre aspas são vermelhos
 - Comentários são verdes

Exercícios

- 1) Abra a record EXERC_ALUNO, clique com botão direito no campo INSTITUTION e escolha *View PeopleCode*
- 2) Escolha o evento *FieldChange* no topo a direita do seu editor de PeopleCode.
- 3) Digite o PeopleCode abaixo:
`If ^ = "ABC" Then Warning("Foi digitado ABC");`
- 4) Salve e verifique que o Editor valida o código digitado apresentando um erro
- 5) Adicione a linha abaixo
`End-If;`
- 6) Salve novamente e observe que o Editor substituiu o símbolo “^” pelo nome do campo onde você está digitando
- 7) Posicione o cursor na função Warning e aperte F1 no teclado, o editor abrirá o PeopleBooks exibindo a ajuda da função Warning.

Ferramentas Adicionais de Desenvolvimento

Find Definition References

Permite ver todas as referências que utilizam uma determinada definição, desta forma é possível, por exemplo, saber todas as records, páginas, componentes, projetos, etc, que fazem uso de uma determinada definição.

Clicando com o botão direito sobre uma definição, é possível acessar este comando e pesquisar as referências da definição.

Find In

Faz uma pesquisa em todo o banco de dados ou em um projeto específico buscando uma porção de texto.

Este recurso é muito útil quando se deseja saber quais definições possuem um determinado trecho de código ou uma frase/palavra em específico.

Este comando está disponível através do menu *Edit*.

Mensagens de tela

Os comandos WinMessage e MessageBox e seus parentes, Error e Warning são formas simples de inserir breakpoints dentro de um programa e ver o que está acontecendo num trecho de código, quando o Debugger não está ativo em uma determinada base.

Obs: Error e Warning existem apenas por compatibilidade de código de versões anteriores e não devem ser usados em programas PeopleCode rodando em produção. Use-os apenas para debugar seu código, depois remova-os do código para migração ou substitua pelo comando MessageBox.

Limitações

Estas funções não podem ser usadas nos seguintes eventos:

- SavePreChange
- Workflow
- RowSelect
- SavePostChange
- Qualquer evento disparado pelas chamadas das funções ScrollSelect, ScrollSelectNew, RowScrollSelect e RowScrollSelectNew
- Qualquer evento disparado pelos métodos Select e SelectNew da classe RowSet

MessageBox

Ele abre uma janela popup na forma definida pela mensagem no catálogo de mensagens. Quando não usado o catálogo de mensagens, a mensagem será exibida na forma de aviso.

Para cadastrar uma mensagem no catálogo de mensagens é necessário se conectar ao PeopleSoft pelo navegador web. Acesse o menu PeopleTools / Utilitários / Administração / Catálogo de Mensagens.

Limitações sobre o uso desta função, veja o item acima “Limitações”.

Sintaxe:

```
[<result> =] MessageBox(<style> , <title>, <msg_set> , <msg_nbr> ,<default_txt>);
```

Exemplos:

```
MessageBox(0, "", 0, 0, "O valor da variável &X é " | &X);
```

```
MessageBox(0, "", 40, 33, " ");
```

```
MessageBox(%MsgStyle_OK, "", 0, 0, "O valor da variável &X é " | &X);
```

```

&R = MessageBox(%MsgStyle_YesNo + %MsgStyle_Second, "", 0, 0, "Deseja continuar ?");
If &R = %MsgResult_Yes Then
    /* usuário clicou no botão SIM */
Else
    /* usuário clicou no botão NÃO */
End-If;

&R = MessageBox(%MsgStyle_RetryCancel, "", 0, 0, "Erro ao ler arquivo.");
Evaluate &R
When %MsgResult_Retry
    /* usuário clicou no botão REPETIR */
When %MsgResult_Cancel
    /* usuário clicou no botão CANCELAR */
End-Evaluate;

```

Valores válidos para o parâmetro <style>:

Categoria	Valor	Constante
Botão	0	%MsgStyle_OK
Botão	1	%MsgStyle_OKCancel
Botão	2	%MsgStyle_AbortRetryIgnore
Botão	3	%MsgStyle_YesNoCancel
Botão	4	%MsgStyle_YesNo
Botão	5	%MsgStyle_RetryCancel
Botão Default	0	%MsgStyle_First
Botão Default	256	%MsgStyle_Second
Botão Default	512	%MsgStyle_Third

Valores válidos para o parâmetro <result>:

Valor	Constante
-1	%MsgResult_Warning
1	%MsgResult_OK
2	%MsgResult_Cancel
3	%MsgResult_Abort
4	%MsgResult_Retry
5	%MsgResult_Ignore
6	%MsgResult_Yes
7	%MsgResult_No

WinMessage

Similar ao MessageBox, este comando abre uma nova página com a mensagem do primeiro parâmetro, a execução do programa PeopleCode é suspensa até que o usuário aperte algum botão desta nova página, quando então a página se fecha, voltando a página anterior e continuando o fluxo normal do processamento.

Sintaxe:

```
[<result> =] WinMessage (<message> [, <style>] [,<title>]);
```

Exemplos:

```
WinMessage ("O valor da variável &X é " | &X, 0);
```

Valores válidos para o parâmetro <style>: ver MessageBox

Valores válidos para o parâmetro <result>: ver MessageBox

Warning

Ele abre uma janela popup exibindo a mensagem do parâmetro na forma de aviso, a execução do programa PeopleCode é suspensa até que o usuário aperte algum botão desta janela, após fechá-la a execução do fluxo do processamento continua normalmente.

Sintaxe:

```
Warning (<mensagem>);
```

Exemplos:

```
Warning ("O valor da variável &X é " | &X);
```

Error

Ele abre uma janela popup exibindo a mensagem do parâmetro na forma de erro, uma janela popup é exibida e após fechá-la, a execução do programa PeopleCode é interrompida, sendo que o resto de código existente não será executado.

Sintaxe:

```
Error (<mensagem>);
```

Exemplos:

```
Error ("O valor da variável &X é " | &X);
```

Entendendo o Fluxo do Processador de Componente

Tipos de Processamento

Há 2 tipos de processamento: *Interactive* (interativo ou online) e *Deferred* (atrasado)

Interactive (interativo)

No processamento *interactive* (interativo ou online), os eventos são disparados no momento que as coisas acontecem.

A vantagem deste modo é a validação online assim que o usuário digita alguma informação em um campo.

A desvantagem é o tráfego de dados constante com o servidor, especialmente em redes mais lentas, que tornam o uso da página difícil.

Deferred (atrasado)

No processamento *deferred* (atrasado), os eventos não disparam no momento que imaginamos que ele deveria disparar, ao invés disso, como o objetivo deste modo é reduzir tráfego de rede, eles são colocados numa fila para execução no próximo acesso ao servidor, que pode ser quando salvamos a página ou quando clicamos com o mouse numa prompt, por exemplo.

A vantagem deste modo é a quase ausência de tráfego de rede durante o preenchimento da página, tornando mais rápido para o usuário seu trabalho e aliviando a carga de trabalho do servidor.

A desvantagem é perder a validação online dos campos.

Interactive ou Deferred

É possível ter um misto de campos *interactive* e *deferred* numa mesma página, o que pode ser a melhor solução para determinadas situações, neste caso, configuramos campos que necessitam validação online como *interactive* e campos que não necessitam, como *deferred*.

O melhor a fazer é planejar bem a página para que não pese na carga do servidor, nem fique cansativa para o usuário trabalhar com interrupções freqüentes ao servidor, mas que ao mesmo tempo, possa ter todo o tipo de validação necessária no momento apropriado.

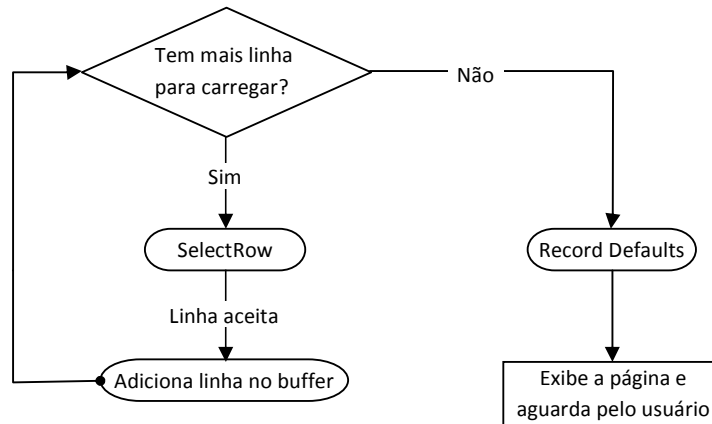
Mantenha em mente: Quanto mais objetos em modo *interactive*, mais lenta a página.

Exercícios

- 1) Abra o Application Designer e abra o seu projeto
- 2) Abra a página EXERC_ALUNO, execute um duplo clique no campo INSTITUTION, na aba *Use*, desmarque a opção *Allow Deferred Mode*
- 3) Teste seu aplicativo digitando algo neste campo e pressionando a tecla *Tab*
- 4) Volte no Application Designer
- 5) Agora marque a opção que você havia desmarcado no exercício anterior
- 6) Teste novamente seu aplicativo e veja a diferença

Carregando Dados no Buffer

Quando o usuário pesquisa por uma chave única, o processador do componente retorna todos os dados necessários para o componente inteiro, somente depois monta o componente, exibe a primeira página e aguarda interação do usuário.



Quando o usuário pesquisa por uma chave única, o processador do componente retorna todos os dados necessários para o componente inteiro, somente depois monta o componente, exibe a primeira página e aguarda interação do usuário.

Se houverem níveis dentro de uma página, todas as linhas de cada nível são colocadas no buffer antes da página ser exibida. Isto significa que quantos mais níveis, mais linhas e quanto mais linhas, mais lenta a página para ser carregada.

Adicionando registros

Ao montar a página do aplicativo, deve-se atentar para deixar os campos chaves do nível 0 da página, como display-only, pois se permitir que o usuário altere uma chave de nível 0, os registros de níveis abaixo ficarão órfãos.

Onde é possível programar eventos

O fluxo do processador de componente executa PeopleCodes nos eventos das seguintes definições:

- Components
- Pages
- Records

Também é possível programar nos eventos das definições abaixo, porém, elas não estão associadas ao fluxo do processador de componentes:

- Message
- Application Engine
- Component Interface
- Application Package

Eventos do Fluxo do Processador de Componentes

Página de Pesquisa	Construção da página	Alteração de campos	Ação de linha	Salvamento
SearchInit	RowSelect	FieldEdit	RowInsert	SaveEdit
SearchSave	PreBuild	FieldChange	RowDelete	SavePreChange
	FieldDefault			Workflow
	FieldFormula			SavePostChange
	RowInit			
	PostBuild			
	Activate			

SearchInit

Este evento dispara em cada campo da *Search Record* que é exibido na página de pesquisa, quando ela é criada, porém antes de ser exibida.

Este tipo de evento é muito usado para definir valores default ou desabilitar a edição de determinados campos de pesquisa.

Coloque PeopleCode somente nos campos que tenham as flags *Search Key* ou *Alternate Search Key* marcadas na record.

Comandos muito utilizados neste evento são:

- SearchDefault – habilita os valores default do campo
- SearchEdit – habilita a prompt do campo
- SetSearchDialogBehavior – define o comportamento da página

Exemplos:

```
Local Field &Campo;  
&Campo = GetField(MinhaRecord.MeuCampo);  
&Campo.SearchDefault = True;  
&Campo.SearchEdit = True;  
  
MinhaRecord.MeuCampo.SearchDefault = True;  
MinhaRecord.MeuCampo.SearchEdit = True;  
  
SetSearchDialogBehavior(0);
```

Restrições

- ⊗ Este evento não dispara se a página estiver sendo executada via component interface.
- ⊗ As seguintes funções não podem ser executadas dentro deste evento:
 - DoModal
 - DoModalComponent
 - Transfer
 - TransferExact
 - TransferNode
 - TransferPage
 - TransferPortal

SearchSave

Este evento dispara em cada campo da *Search Record* no momento em que o botão *Pesquisar* ou *Adicionar* é clicado, porém, antes da pesquisa/adição ser efetuada. Isto permite que possamos validar os dados antes do processamento continuar.

Este evento também é usado para forçar o usuário a preencher um determinado campo para a pesquisa, evitando que a pesquisa retorne linhas demais.

Exemplos:

```
Local Field &Campo;  
&Campo = MinhaRecord.MeuCampo;  
If &Campo.Value < 0 Then  
    Error "Este campo não pode ser negativo.";  
End-if;  
  
If MinhaRecord.MeuCampo.Value < 0 Then  
    Error "Este campo não pode ser negativo.";  
End-if;
```

Restrições

- ⊗ Este evento não dispara se a página estiver sendo executada via component interface.
- ⊗ Este evento não dispara quando um valor é selecionado na lista de pesquisa, neste caso, faça a validação no evento *PreBuild* do componente.
- ⊗ Não use a variável de sistema *%Menu* neste evento, pode causar resultados não esperados.

RowSelect

Este evento é disparado no início do processo de criação de componente e é usado para filtrar linhas de dados que estão carregadas no buffer do componente. Este evento também é disparado quando a função *ScrollSelect* é executada.

Executar a função *DiscardRow* dentro deste evento fará com que o processador do componente ignore a linha atual pulando para a próxima.

Executar a função *StopFetching* dentro deste evento fará com que o processador do componente aceite a linha atual e depois interrompa a leitura das linhas restantes.

Executar as funções *DiscardRow* e *StopFetching* fará com que o processador do componente não carregue a linha atual e interrompa a leitura das linhas restantes.

O uso deste evento não é recomendado por ser ineficiente, para melhor desempenho filtre os dados diretamente em uma SQL View e use-a para carregar os dados.

Exemplos:

```
/* somente carrega linha quando valor é positivo, descartando os negativos */  
If MinhaRecord.MeuCampo.Value < 0 Then  
    DiscardRow();  
End-if;
```

Restrições

- ⊗ Este evento não dispara se a página estiver sendo executada via component interface.

PreBuild

Este evento é iniciado antes da construção da página e é muito utilizado para esconder/exibir páginas/campos, bem como definir valores iniciais em campos. Também é usado para criação de variáveis do tipo *Component*.

Se um erro ocorrer neste evento, o usuário será redirecionado novamente para a página de pesquisa.

Exemplos:

```
MinhaRecord.OPRID.Value = %UserId;
```

Restrições

- ⊗ Este evento não possui restrições.

FieldDefault

Este evento poderá ocorrer na inicialização da página ou quando uma nova linha em uma grid/scroll for criada, porém, o evento somente é disparado se o campo não possuir valor, desta forma, será possível preencher o campo com algum valor antes que ele seja exibido na tela.

O Peoplesoft entende que um campo não possui valor nas seguintes situações:

- Quando o campo possui valor NULL
- Quando o campo possui somente espaços
- Quando o campo possui valor zero (0), em casos de campos numéricos

Exemplos:

```
MinhaRecord.MeuCampo.Value = "blablabla";
```

Restrições

- ⊗ Este evento não dispara caso o campo já possua valor no banco de dados.
- ⊗ Quando o valor de um campo é alterado, seja via PeopleCode ou por usuário, o campo é marcado como alterado (*IsChanged = True*), porém, ao alterar o valor de um campo dentro dos eventos *FieldDefault* e *FieldFormula*, o campo não será marcado como alterado (*IsChanged = False*).

FieldFormula

Este evento normalmente não é usado, pois é disparado em diferentes contextos e triggers PeopleCode em cada campo de cada linha do buffer do componente, e com isso, pode causar séria degradação de desempenho da aplicação.

Por convenção, este evento é usado como local de armazenagem de funções.

Restrições

- ⊗ Este evento pode causar séria degradação de desempenho, evite usá-lo, exceto para armazenagem de funções.
- ⊗ Use os eventos *RowInit* e *FieldChange* ao invés deste.
- ⊗ Quando o valor de um campo é alterado, seja via PeopleCode ou por usuário, o campo é marcado como alterado (*IsChanged = True*), porém, ao alterar o valor de um campo dentro dos eventos *FieldDefault* e *FieldFormula*, o campo não será marcado como alterado (*IsChanged = False*).

RowInit

Este evento é disparado na primeira vez que o processador do componente encontra uma linha de dados ou quando uma nova linha é adicionada numa grid/scroll. Também é disparado quando as funções *Select*, *SelectAll* e *ScrollSelect* são executadas.

Este evento é usado para definir o estado inicial de objetos na página.

Este evento normalmente faz par com o evento *FieldChange*, sendo que no evento *RowInit* o valor do campo é checado na inicialização e no *FieldChange* o valor do campo é re-chechado após digitação do usuário.

Exemplos:

```
MinhaRecord.TOTAL.Value = MinhaRecord.VALOR1.Value + MinhaRecord.VALOR2.Value;
```

```
If MinhaRecord.Campo1.Value = "N" Then  
    MinhaRecord.Campo2.Enabled = False;  
End-if;
```

Restrições

- ⊗ Este evento não dispara quando todos os campos do registro de pesquisa são preenchidos e retornam uma linha de dados.

- ⊗ Este evento não dispara quando as 3 situações abaixo ocorrem ao mesmo tempo:
 - O registro está no nível zero
 - Todos os campos chave da record foram preenchidos na tela de pesquisa
 - Todos os campos da record são *display-only*

PostBuild

Este evento dispara depois que todos os outros eventos do componente já ocorreram.

Normalmente este evento é usado para exibir/esconder páginas e também para declaração de variáveis de componente.

Exemplos:

```
If %Component = Component.MeuComponente Then
  MinhaPagina.Visible = False;
End-if;
```

Restrições

- ⊗ Este evento não possui restrições.

Activate

Este evento é ativado sempre que uma página é ativada, incluindo quando a página é exibida pela primeira vez ou quando um usuário acessa diferentes páginas dentro de um componente (clicando na aba da página).

Cada página possui seu próprio evento *Activate*.

Restrições

- ⊗ A forma como o PeopleSoft cria uma página quando esta possui uma grid/scroll não permite executar as funções *GetRecord*, *GetRow* e similares para buscar dados da página sem definir maiores níveis de detalhamento de contexto.
- ⊗ A função *MessageBox*, quando o tipo da mensagem for **Erro**, **Aviso** ou **Cancelar**, irá parar o processamento enquanto a mensagem estiver sendo exibida.
- ⊗ A função *MessageBox*, quando o tipo da mensagem for **Mensagem**, não irá parar o processamento.

FieldEdit

Este evento é usado para a validação do conteúdo de um campo.

Se houver necessidade de validar vários campos da página use o evento *SaveEdit* ao invés deste.

Exemplos:

```
Local Field &Campo;
&Campo = MinhaRecord.MeuCampo;
If &Campo.Value < 0 Then
  Error "Este campo não pode ser negativo.";
End-if;
```

```
If MinhaRecord.MeuCampo.Value < 0 Then
  Error "Este campo não pode ser negativo.";
End-if;
```

Restrições

- ⊗ Este evento não dispara na página de pesquisa, nem de adicionar novo registro

FieldChange

Este evento é usado para recalcular valores ou executar regras de negócio específicas à alteração de um determinado campo.

Para validar o conteúdo de um campo use o evento *FieldEdit*.

Este evento normalmente faz par com o evento *RowInit*, sendo que no evento *RowInit* o valor do campo é checado na inicialização e no *FieldChange* o valor do campo é re-checado após digitação do usuário.

Exemplos:

```
MinhaRecord.TOTAL.Value = MinhaRecord.VALOR1.Value + MinhaRecord.VALOR2.Value;
```

```
If MinhaRecord.Campo1.Value = "N" Then  
    MinhaRecord.Campo2.Enabled = False;  
End-if;
```

Restrições

- ⊗ Este evento não dispara na página de pesquisa, nem de adicionar novo registro

SaveEdit

Este evento é disparado sempre que o usuário tenta salvar o componente. Use para validar consistência de dados entre vários campos.

Uma exibição de mensagem de erro neste evento aborta o processamento impedindo que o salvamento dos dados aconteça.

Use a função *SetCursorPos* para indicar ao usuário qual campo está causando o erro, mas tenha a certeza de chamar esta função antes da exibição do erro, pois, após a exibição da mensagem de erro, o código que estiver a seguir não será executado.

Exemplos:

```
If MinhaRecord.MeuCampo1.Value < 0 And MinhaRecord.MeuCampo2.Value = "" Then  
    /* primeiro coloca o cursor no campo com problema */  
    SetCursorPos(%Page, MinhaRecord.MeuCampo2, CurrentLineNumber());  
    /* depois mostra mensagem na tela */  
    Error "Quando MeuCampo1 for menor que zero, o MeuCampo2 deverá estar preenchido.";   
End-if;
```

Restrições

- ⊗ Este evento não dispara em linhas que estejam marcadas como *Deleted*.
- ⊗ A ocorrência de erro neste evento impede o salvamento dos dados.
- ⊗ Este evento não dispara na página de pesquisa, nem de adicionar novo registro

SavePreChange

Este evento é disparado somente se o evento *SaveEdit* for concluído sem erro.

Este evento oferece a última oportunidade para manipulação de dados antes que o salvamento (insert, update e delete), efetivamente, ocorra no banco de dados.

Exemplos:

```
MinhaRecord.TOTAL.Value = MinhaRecord.VALOR1.Value + MinhaRecord.VALOR2.Value;
```

Restrições

- ⊗ A ocorrência de erro neste evento impede o salvamento dos dados.

Workflow

Este evento é disparado somente se o evento *SavePreChange* for concluído sem erro e antes que o salvamento, efetivamente, ocorra no banco de dados.

Este evento deve conter peoplecode exclusivamente relacionado a *workflow*, como por exemplo, a função *TriggerBusinessEvent*.

Restrições

- ⊗ A ocorrência de erro neste evento impede o salvamento dos dados.

SavePostChange

Este evento é disparado somente após o banco de dados ter sido atualizado (insert, update e delete).

Se você tiver código no evento *Workflow* e este falhar, o evento *SavePostChange* não será disparado, neste caso, considere a possibilidade de mover o código do evento *Workflow* para o evento *SavePreChange*.

Restrições

- ⊗ Este evento não é disparado caso o sistema não consiga salvar no banco de dados, como conflito de dados, por exemplo.
- ⊗ Nunca force um *Commit* ou *Rollback* manualmente através da função *SQLExec*.

RowInsert

Este evento é disparado somente quando o usuário adiciona uma linha nova.

Como este evento é disparado logo após o evento *RowInit*, não repita um código nesses 2 eventos para evitar que ele execute 2 vezes.

Se nenhum campo for alterado após a linha ser adicionada, ao salvar a página, esta linha nova não será salva no banco de dados.

Para evitar que uma linha nova seja adicionada em uma grid/scroll, marque a opção *No RowInsert* nas propriedades da grid/scroll. Não é possível evitar inserção de linha nova condicionalmente.

Restrições

- ⊗ Se a propriedade *ChangeOnInit* da classe *Rowset* for *False*, mesmo que os campos sejam alterados a linha não será salva no banco de dados.
- ⊗ Não use as funções *Error* e *Warning* neste evento.

RowDelete

Este evento é disparado somente quando o usuário tenta remover uma linha de uma grid/scroll

Este evento deve ser usado para evitar a remoção de linhas que não podem ser removidas.

Este evento marcará cada campo da linha a ser removida com o status de *Deleted*.

Ao remover a única linha de uma grid/scroll, automaticamente, o PeopleSoft executará um *RowInsert* e adicionará uma linha vazia, disparando a execução de todos os eventos que essa nova linha possa gerar. Caso você não queira esses eventos executando, mova o *peoplecode* do evento *RowInit* para o evento *FieldDefault*.

Exemplos:

```
If MinhaRecord.MeuCampo1.Value > 0 Then  
    Error "Esta linha não pode ser removida quando MeuCampo1 for maior que zero."  
End-if;
```

Restrições

- ⊗ Este evento não causa a execução de eventos em records do tipo *Derived/Work*.

Exercícios

- 1) Abra o Application Designer e abra o seu projeto
- 2) Abra a record EXERC_ALUNO, clique com botão direito no campo INSTITUTION e escolha *View PeopleCode*
- 3) No evento *SearchInit*, digite o código:
`EXERC_ALUNO.INSTITUTION.Value = "ABCD";`
- 4) Teste seu aplicativo verificando a página de pesquisa

- 5) No evento *SearchSave*, digite o código:
`Error "Executando o evento SearchSave";`
- 6) Teste seu aplicativo e pesquise um registro
- 7) Remova o código do exercício 5.
- 8) No evento *RowInit*, digite o código:
`WinMessage ("Executando o evento RowInit");`
- 9) Teste seu aplicativo e pesquise um registro
- 10) No evento *FieldChange*, digite o código:
`WinMessage ("Executando o evento FieldChange");`
- 11) No evento *FieldEdit*, digite o código:
`WinMessage ("Executando o evento FieldEdit");`
- 12) Teste seu aplicativo digitando no campo INSTITUTION e apertando a tecla *Tab*.
- 13) No evento *SaveEdit*, digite o código:
`WinMessage ("Executando o evento SaveEdit");`
- 14) No evento *SavePreChange*, digite o código:
`WinMessage ("Executando o evento SaveEdit");`
- 15) No evento *SavePostChange*, digite o código:
`Error "Executando o evento SavePostChange";`
- 16) Teste seu aplicativo cadastrando dados novos e clicando em Salvar.
- 17) Remova todos os códigos que você adicionou nestes exercícios
- 18) No evento *Activate*, digite o código:
`Error "Executando o evento Activate";`
- 19) Teste seu aplicativo.
- 20) Remova o código que você adicionou no exercício 18.
- 21) No evento *PreBuild*, digite o código:
`Error "Executando o evento PreBuild";`
- 22) Teste seu aplicativo.
- 23) Remova o código do exercício 21
- 24) No evento *PostBuild*, digite o código:
`EXERC_ALUNO.INSTITUTION.Value = "POSTB";`
- 25) Teste seu aplicativo.
- 26) Remova todos os códigos que você adicionou do exercício 1 ao 24.
- 27) Teste seu aplicativo para ter certeza que todos os códigos foram removidos.

Modos de Processamento

Os modos de processamento indicam como a página está sendo tratada e através deles podemos tomar diferentes ações.

Valor	Constante	Descrição
A	%Action_Add	Adicionar
U	%Action_UpdateDisplay	Atualizar/Consultar
L	%Action_UpdateDisplayAll	Atualizar/Consultar Todos
C	%Action_Correction	Corrigir Histórico
E	%Action_DataEntry	Entrada de Dados
P	%Action_Prompt	Prompt

%Mode

É com esta variável de sistema que podemos identificar em qual modo de processamento um determinado Peoplecode está rodando.

Sintaxe:

```
&Var = %Mode;
```

Exemplos:

```
If %Mode = %Action_Add Then
    WinMessage ("Executando em modo de adicionar");
```

```
End-if;

If %Mode = %Action_UpdateDisplay Then
    WinMessage ("Executando em modo de atualizar/consultar");
End-if;
```

Exercícios

- 1) Abra o Application Designer e carregue o seu projeto
- 2) Abra a record EXERC_ALUNO, clique com botão direito no campo INSTITUTION e escolha *View PeopleCode*
- 3) No evento *RowInit*, digite o código:

```

If %Mode = %Action_Add Then
    WinMessage ("Executando em modo de adicionar");
End-if;

If %Mode = %Action_UpdateDisplay Then
    WinMessage ("Executando em modo de atualizar/consultar");
End-if;
```
- 4) Teste seu aplicativo pesquisando um registro existente
- 5) Teste seu aplicativo adicionando um registro novo

Codificando em PeopleCode

Toda linha de PeopleCode, com algumas raras exceções, devem, obrigatoriamente, terminar com um ponto e vírgula (;).

Casos de exceções serão mostrados adiante neste curso.

Comentários de PeopleCode

Existem 3 tipos de comentários em PeopleCode:

Comentário de Linha

O comentário de linha é feito através dos comandos REM e Remark, e ambos exigem que o comentário termine com um ponto e vírgula.

Os 2 comandos possuem exatamente a mesma utilização, mas a diferença entre o REM e o Remark é que o REM pinta o código de verde, facilitando a identificação do comentário no PeopleCode, enquanto o Remark mantém o código nas cores originais de cada comando. Por este motivo, dê preferência ao uso do REM ao invés do Remark.

Sintaxe:

```
REM <comentário>;  
Remark <comentário>;
```

Exemplos:

```
REM Este texto é um comentário;  
Remark Este texto é um comentário;
```

Comentário de Bloco

O comentário de bloco usa o mesmo estilo de comentário da linguagem C, inicia por `/*` e termina por `*/`, e é usado para comentar porções grandes de texto.

É possível comentar trechos de PeopleCode, porém o tamanho máximo de cada trecho de comentário é 16.383 bytes, se o tamanho do PeopleCode que deseja comentar for maior que este, será necessário quebrar o comentário em vários blocos.

O bloco comentado ficará identificado pela cor verde.

Sintaxe:

```
/* <comentário> */
```

Exemplos:

```
/* Este texto é um comentário */  
  
/*  
    Este texto é um comentário  
*/
```

Comentário de Bloco Aninhado

O comentário de bloco aninhado inicia por `<*` e termina por `*>`, e é usado para comentar trechos de código que já contenham comentário de bloco (`/* */`).

Este tipo de comentário deve ser evitado ao máximo, e usado somente para comentar blocos de código temporariamente ou blocos de código que devam futuramente ser removidos do projeto.

Este tipo de comentário não deve chegar até o ambiente de produção.

Sintaxe:

```
<* <comentário> *>
```

Exemplos:

```
<* Este texto é um comentário *>
```

```
<*  
  Este texto é um comentário  
*>
```

Declaração de Variáveis e Constantes

A declaração de variáveis em Peoplecode é opcional, porém, é altamente recomendado fazer a declaração.

A não declaração de variáveis pode tornar a busca por um erro bastante demorada, principalmente quando, sem notar, digitamos errado o nome de uma variável, nesta situação, se a variável estiver declarada, o PeopleSoft se encarregará de mostrar o nosso erro, tornando muito fácil e rápido achar o defeito do código e então corrigi-lo.

O comprimento máximo de uma variável ou constante é 1000 caracteres e deverá sempre começar com o símbolo &

Uma variável ou constante pode conter letras maiúsculas, minúsculas, números e os seguintes caracteres: #, @, \$ e _

Constantes podem ser apenas string, number ou boolean.

Tipos de Escopo**Local**

Tem o menor escopo entre os tipo de escopo, se declarada dentro de um evento, terá uma vida somente dentro deste evento, se declarada dentro de uma função, existirá somente dentro da função.

Dê preferência para variáveis com este tipo de escopo.

Component

Uma variável deste escopo existirá somente dentro do componente, como um componente pode conter muitas páginas e records, em todos esses eventos de páginas e record fields essa variável será visível.

Deve ser declarada na primeira linha de cada evento em que for usada.

Sempre que possível, evite variáveis com este tipo de escopo.

Global

Tem o maior escopo entre os tipo de escopo, é válida enquanto a sessão estiver aberta.

Deve ser declarada na primeira linha de cada evento em que for usada.

Sempre que possível, evite variáveis com este tipo de escopo, use somente se não houver alternativa possível.

Tipos de Dados Convencionais

Boolean

Uma variável declarada como boolean aceitará somente 2 possíveis valores: TRUE ou FALSE.

Exemplos:

```
Local boolean &A;  
Component boolean &B;  
Global Boolean &C;  
Constant &D = True;
```

Float

Este tipo de dados aceita valores numéricos com decimais e na comparação entre Float e Number, o tipo Float leva vantagem no quesito desempenho em PeopleCode, devendo ser a preferida ao invés de Number.

Este tipo de dados possui 32 bits de tamanho, por isso, possui limitação de valores que consegue armazenar, se o valor que você pretende armazenar é muito grande ou possui muitas casas decimais, use o tipo Number.

Integer

O tipo Integer aceita somente valores numéricos e inteiros, como é uma variável de 32 bits, possui limitação de valores que consegue armazenar, esse limite de valores vai de - 2.147.483.648 até 2.147.483.647, qualquer tentativa de armazenar numero fracionário neste tipo de variável a parte fracionária será descartada, somente a parte inteira será armazenada.

Este é o tipo de variável numérica que oferece o melhor desempenho na execução de PeopleCode, use sempre este tipo quando possível.

Number

Este tipo de dados aceita valores numéricos com decimais, porém possui o pior desempenho em PeopleCode entre todas as numéricas, por isso, sempre que possível, substitua este tipo de variável por Float ou Integer.

Exemplos:

```
Local number &A;  
Component number &B;  
Global number &C;  
Constant &D = 1234;
```

Date

Este tipo de variável armazena somente valores de datas, desconsiderando o horário.

Exemplos:

```
&A = Date3(2011, 01, 15);    => 15/janeiro/2011  
&B = Date3(2011, 01, 20);    => 20/janeiro/2011  
&C = &A + 1;                 => 16/Janeiro/2011  
&C = &A - 1;                 => 14/janeiro/2011  
&C = &B - &A;                 => 5 (dias)
```

Time

Este tipo de variável armazena somente valores de horas, desconsiderando a data.

Exemplos:

```
&A = Time3(18, 15, 12.3);    => 18:15:12.300000  
&B = Time3(14, 10, 12.3);    => 14:10:12.300000  
&C = &A + 1;                 => 18:10:13.300000
```

```

&C = &A - 1;           => 18:10:11.300000
&C = &B - &A;          => 14700 (segundos)

```

DateTime

O tipo DateTime consegue armazenar simultaneamente data e hora.

Exemplos:

```

&A = Date3(2011, 5, 20);           => 20/mai/2011
&B = Time3(14, 10, 12.3);          => 14:10:12.300000
&C = &A + &B;                       => 20/mai/2011 14:10:12.300000
&C = DateTime6(2011, 5, 20, 14, 10, 12.3); => 20/mai/2011 14:10:12.300000

```

String

O tipo String armazena qualquer tipo de character, pode ser numérico, alfa-numérico, símbolos, etc.

Exemplos:

```

Local string &A;
Component string &B;
Global string &C;
Constant &D = "ABC";

```

Object

O tipo object é um tipo genérico que permite armazenar qualquer tipo de dados não convencional.

Evite declarar variável com este tipo de dados, pois declarar a variável com o tipo certo de dados garante melhor velocidade no aplicativo.

Any

O tipo Any permite que a variável consiga armazenar qualquer coisa, seja um tipo convencional ou não.

Quando uma variável não é declarada no código, o PeopleSoft, automaticamente, declara a variável como sendo deste tipo de dado.

Evite este tipo de dados, pois ele possui o pior desempenho entre todos os tipos de dados existentes no PeopleCode. Usar este tipo de dados (ou não declarar uma variável) é garantia de aplicativo lento.

Exemplos:

```

local Any &A, &B;
Component Any &C;

```

Variáveis de Sistema

%Component

Esta variável retorna o componente no qual o aplicativo está rodando.

Exemplos:

```

If %Component = Component.MEU_COMPONENTE Then
    /* executa peoplecodes */
End-If;

```

%ComplntfcName

Esta variável retorna o componente interface no qual o aplicativo está rodando.

Exemplos:

```
If %CompIntfcName = CompIntfcName.MEU_COMPONENTE_INTERFACE Then
    /* executa peoplecodes */
End-If;
```

%Date

Esta variável retorna a data local do servidor.

Exemplos:

```
Local date &A;
&A = %Date;
```

%DateTime

Esta variável retorna a data/hora local do servidor.

Exemplos:

```
Local datetime &A;
&A = %DateTime;
```

%EmployeeID

Esta variável retorna o ID do usuário que fez login no sistema.

Exemplos:

```
Local string &A;
&A = %EmployeeID;
&A = %OperatorID;
&A = %UserID;
```

%Language

Esta variável retorna o idioma no qual o aplicativo está rodando.

Exemplos:

```
If %Language = "POR" Then
    WinMessage("Rodando em idioma Português");
End-If;
```

%Menu

Esta variável retorna o menu no qual o aplicativo está rodando.

Exemplos:

```
If %Menu = Menu.MEU_MENU Then
    /* executa peoplecodes */
End-If;
```

%OperatorID

Esta variável retorna o ID do usuário que fez login no sistema.

Exemplos:

```
Local string &A;
&A = %EmployeeID;
&A = %OperatorID;
&A = %UserID;
```

%Page

Esta variável retorna a página na qual o aplicativo está rodando.

Exemplos:

```
If %Page = Page.MINHA_PAGINA Then
    /* executa peoplecodes */
End-If;
```

%Session

Esta variável retorna a sessão na qual o aplicativo está rodando.

Exemplos:

```
Local compintfc &Customer;  
&Customer = %Session.GetCompIntfc(COMPINTFC.CUSTOMER);  
&Customer.Name = "Cliente";
```

%SQLRows

Esta variável retorna o número de linhas afetadas por um *SELECT*, *INSERT*, *UPDATE* ou *DELETE*.

Exemplos:

```
SQLExec("Update PS_TABELA Set CAMPO1 = CAMPO2 Where CAMPO3 = 0");  
WinMessage("O numero de linhas afetadas pelo Update é: " | %SQLRows);
```

%Time

Esta variável retorna a hora local do servidor.

Exemplos:

```
Local time &A;  
&A = %Time;
```

%UserID

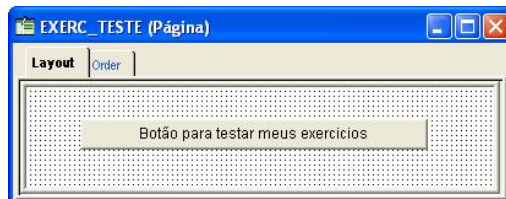
Esta variável retorna o ID do usuário que fez login no sistema.

Exemplos:

```
Local string &A;  
&A = %EmployeeID;  
&A = %OperatorID;  
&A = %UserID;
```

Exercícios

- 1) Abra o Application Designer e carregue o seu projeto
- 2) Crie uma record do tipo Derived/Work
- 3) Adicione o field BUTTON1 na record que você criou
- 4) Salve a sua record com o nome EXERC_TESTE_WRK
- 5) Crie a página abaixo:



- 6) Crie o componente abaixo, colocando EXERC_TESTE_WRK como Search Record:



- 7) Registre o componente usando:
- 8) Nome do Menu: TREINAMENTO

- 9) Nome da Barra: USE
- 10) Nome da Pasta: TREINAMENTO
- 11) Nº Sequência: 60
- 12) Lista de Permissão: TREINAMENTO
- 13) No evento *FieldChange* do BUTTON1, digite o código:


```
Local integer &A, &B, &Soma;
&A = 5;
&B = 10;
&Soma = &A + &B;
WinMessage("A soma de " | &A | " e de " | &B | " é: " | &Soma);
```
- 14) Teste seu aplicativo
- 15) No evento *FieldChange* do BUTTON1, comente o código existente
- 16) Adicione o seguinte código:


```
WinMessage("Este código está rodando no componente " | %Component);
```
- 17) Comente o código anterior, adicione o seguinte código e teste:


```
WinMessage("Este código está rodando na página " | %Page);
```
- 18) Comente o código anterior, adicione o seguinte código e teste:


```
WinMessage("Hoje é " | %date);
```
- 19) Comente o código anterior, adicione o seguinte código e teste:


```
WinMessage("Agora é " | %datetime);
```
- 20) Comente o código anterior, adicione o seguinte código e teste:


```
WinMessage("Agora são: " | %Time);
```
- 21) Comente o código anterior, adicione o seguinte código e teste:


```
WinMessage("O usuário que fez login é: " | %UserID);
```
- 22) Comente o código anterior, adicione o seguinte código e teste:


```
WinMessage("O usuário que fez login é: " | %OperatorID);
```

Tipos de Dados de Objetos

Existem muitos tipos de objetos no PeopleCode e todos os objetos podem ser representados por PeopleCode, entre os objetos mais comumente usados estão:

- Array
- Field
- Record
- Row
- Rowset
- Grid
- SQL
- Page
- Session
- Response
- Message

Array

Array é um tipo de variável que consegue armazenar múltiplos valores, arrays podem ser unidimensionais ou multidimensionais.

Arrays crescem e encolhem dinamicamente, a medida que novos elementos são inseridos ou removidos, logo não é necessário definir o tamanho de um array, o tamanho dele é limitado pela quantidade de memória RAM disponível.

Entretanto, definir o tamanho de um array pode melhorar o desempenho do seu aplicativo.

Sintaxe:

```
/* declara array unidimensional */
<escopo> array of <tipo de dados> <nome_variavel>;

/* declara array bidimensional */
<escopo> array of array of <tipo de dados> <nome_variavel>;
```

```
/* declara array tridimensional */
<escopo> array of array of array of <tipo de dados> <nome_variavel>;
```

Exemplos:

```
/* declara array unidimensional */
Local array of string &A;

/* declara array bidimensional */
Local array of array of number &A;

/* declara array tridimensional */
Local array of array of array of boolean &A;

/* acessa um array unidimensional */
&A[1] = "Oi";

/* acessa um array bidimensional */
&A[1,1] = "Oi";
&A[1][1] = "Oi";

/* acessa um array tridimensional */
&A[1,1,1] = "Oi";
&A[1][1][1] = "Oi";
```

CreateArray

CreateArray é uma função que constrói um array de um determinado tipo de dados e inicializa ele com alguns elementos.

Sintaxe:

```
<variável_array> = CreateArray([<param1>][, <param2>][, <paramN>]);
```

Exemplos:

```
Local array of number &A;
/* cria o array com 3 elementos: 10, 33 e 42 */
&A = CreateArray(10, 33, 42);
/* coloca o numero 29 onde o array tinha o numero 33 */
&A[2] = 29;

&Msg = "O conteúdo do array é:";
&Msg = &Msg | Char(10) | &A[1];
&Msg = &Msg | Char(10) | &A[2];
&Msg = &Msg | Char(10) | &A[3];
WinMessage (&Msg);

Local array of array of number &A;
/* cria o array com 3x2 elementos */
&A = CreateArray(CreateArray(10, 15), CreateArray(33, 41), CreateArray(42, 17));
/* coloca o numero 29 onde o array tinha o numero 41 */
&A[2, 2] = 29;

&Msg = "O conteúdo do array é:";
&Msg = &Msg | Char(10) | &A[1, 1] | " - " | &A[1, 2];
&Msg = &Msg | Char(10) | &A[2, 1] | " - " | &A[2, 2];
&Msg = &Msg | Char(10) | &A[3, 1] | " - " | &A[3, 2];
WinMessage (&Msg);
```

Push

O método Push adiciona elementos no final do array.

Sintaxe:

```
<variável_array>.Push(<elemento>);
```

Exemplos:

```
Local array of number &A;
&A = CreateArray();
&A.Push(10);
&A.Push(20);
&A.Push(30);
```

```
/* este array sera criado assim: 10, 20, 30 */
```

Pop

O método Pop remove o ultimo elemento do array retornando seu valor.

Sintaxe:

```
<variavel> = <variável_array>.Pop();
```

Exemplos:

```
Local array of number &A;  
&A = CreateArray();  
&A.Push(10);  
&A.Push(20);  
&A.Push(30);  
/* este array neste momento está assim: 10, 20, 30 */  
&Item = &A.Pop();  
/* este array agora está assim: 10, 20 */
```

Unshift

O método Unshift adiciona um elemento na primeira posição de um array, todos os demais elementos são deslocados 1 posição para frente.

Sintaxe:

```
<variável_array>.Unshift(<lista_param>);
```

Exemplos:

```
Local array of number &A;  
&A = CreateArray();  
&A.Push(10);  
&A.Push(20);  
&A.Push(30);  
/* este array neste momento está assim: 10, 20, 30 */  
&A.Unshift(40);  
&A.Unshift(50);  
/* este array agora está assim: 50, 40, 10, 20, 30 */
```

Shift

O método Shift remove o primeiro elemento de um array e retorna seu valor, todos os demais elementos são deslocados 1 posição para trás.

Sintaxe:

```
<variavel> = <variável_array>.Shift();
```

Exemplos:

```
Local array of number &A;  
&A = CreateArray();  
&A.Push(10);  
&A.Push(20);  
&A.Push(30);  
/* este array neste momento está assim: 10, 20, 30 */  
&Item = &A.Shift();  
/* este array agora está assim: 20, 30 */
```

CreateArrayAny

CreateArrayAny é uma função que constrói um array do tipo Any e inicializa ele com alguns elementos, como o tipo Any, pode ser qualquer coisa, o array poderá armazenar qualquer tipo de dados em qualquer elemento.

Um array do tipo Any é mais flexível em termos de armazenagem de dados, porém, este tipo de array pode reduzir o desempenho do seu aplicativo.

Use este tipo de array somente se não houver outra alternativa.

Sintaxe:

```
<variável_array> = CreateArrayAny([<param1>][, <param2>][, <paramN>]);
```

Exemplos:

```
Local array of Any &A;
/* cria o array com 5 elementos */
&A = CreateArrayAny(10, True, "Bom Dia", False, 15);
/* coloca o texto "Boa tarde" onde o array tinha o valor False */
&A[4] = "Boa tarde";

&Msg = "O conteúdo do array é:";
&Msg = &Msg | Char(10) | &A[1];
&Msg = &Msg | Char(10) | &A[2];
&Msg = &Msg | Char(10) | &A[3];
&Msg = &Msg | Char(10) | &A[4];
&Msg = &Msg | Char(10) | &A[5];
WinMessage (&Msg);
```

CreateArrayRept

CreateArrayRept é uma função que constrói um array preenchendo todos os elementos do array com o mesmo valor.

Sintaxe:

```
<variável_array> = CreateArrayRept(<valor>, <quantidade>);
```

Exemplos:

```
Local array of number &A;
/* cria array com 4 elementos: 123, 123, 123, 123 */
&A = CreateArrayRept(123, 4);

Local array of string &A;
/* cria array com 4 elementos: "ABC", "ABC", "ABC", "ABC" */
&A = CreateArrayRept("ABC", 4);

Local array of string &A;
/* cria array com 4 elementos vazios: "", "", "", "" */
&A = CreateArrayRept("", 4);

/* cria array bidimensional 4x2 */
Local array of array of number &B;
&B = CreateArrayRept(CreateArray(25, 87), 4);
&Msg = "O conteúdo do array é:";
&Msg = &Msg | Char(10) | &C[1, 1] | " - " | &C[1, 2];
&Msg = &Msg | Char(10) | &C[2, 1] | " - " | &C[2, 2];
&Msg = &Msg | Char(10) | &C[3, 1] | " - " | &C[3, 2];
&Msg = &Msg | Char(10) | &C[4, 1] | " - " | &C[4, 2];
WinMessage (&Msg);

/* cria array bidimensional 2x4 */
Local array of array of number &C;
&C = CreateArrayRept(CreateArray(11, 23, 57, 99), 2);
&Msg = "O conteúdo do array é:";
&Msg = &Msg | Char(10) | &C[1,1] | " - " | &C[1,2] | " - " | &C[1,3] | " - " | &C[1,4];
&Msg = &Msg | Char(10) | &C[2,1] | " - " | &C[2,2] | " - " | &C[2,3] | " - " | &C[2,4];
WinMessage (&Msg);

/* cria array unidimensional sem limite de tamanho e inicializa com vazio */
Local array of string &D;
&D = CreateArrayRept("", 0);
&D.Push("ABC"); /* adiciona a string "ABC" no primeiro elemento do array */
&D.Push("DEF"); /* adiciona a string "DEF" no segundo elemento do array */
&D.Push("XYZ"); /* adiciona a string "XYZ" no terceiro elemento do array */
&Msg = "O conteúdo do array é:";
&Msg = &Msg | Char(10) | &D[1];
&Msg = &Msg | Char(10) | &D[2];
&Msg = &Msg | Char(10) | &D[3];
WinMessage (&Msg);

/* cria array bidimensional sem limite de tamanho e inicializa com vazio */
Local array of array of string &E;
&E = CreateArrayRept(CreateArrayRept("", 0), 0);
&E.Push(CreateArray(CreateArrayRept("", 0)));
```



```

&E[1,1] = "ABC";
&E[1,2] = "DEF";
&E[1,3] = "GHI";
&E.Push(CreateArray(CreateArrayRept("", 0)));
&E[2,1] = "XXX";
&E[2,2] = "YYY";
&E[2,3] = "ZZZ";

```

Exercícios

- 1) Abra o Application Designer e carregue o seu projeto
- 2) No evento *FieldChange* do BUTTON1, apague o código existente
- 3) Adicione o código exemplo do *CreateArray* e teste:
- 4) No evento *FieldChange* do BUTTON1, apague o código existente
- 5) Adicione o código exemplo do *CreateArrayAny* e teste:
- 6) No evento *FieldChange* do BUTTON1, apague o código existente
- 7) Adicione o código exemplo do *CreateArrayRept* e teste:
- 8) No evento *FieldChange* do BUTTON1, apague o código existente

Estruturas Condicionais

IF Then Else

Desvia o fluxo da execução do programa, dependendo do resultado da condição ser falsa ou verdadeira.

O IF-Then-Else processa todas as condições do critério, independente das primeiras condições já darem um resultado final ao critério. Isso significa se houve chance da segunda condição acarretar em erro, você deverá aninhar um IF-Then-Else, dentro de outro.

Sintaxe:

```

If <condição> Then
    <executa caso condição seja verdadeira>
Else
    <executa caso condição seja falsa>
End-If;

```

Exemplos:

```

If &A >= 0 Then
    WinMessage("A é valor positivo");
End-If;

If &A = &B Then
    WinMessage("A é igual a B");
Else
    WinMessage("A é diferente a B");
End-If;

```

Evaluate

Desvia o fluxo da execução do programa, dependendo do resultado da condição sendo testada.

Sintaxe:

```

Evaluate <variável>
When <condição 1>
    <executa PeopleCodes>
    [break;]
When <condição 2>
    <executa PeopleCodes>
    [break;]
When <condição N>
    <executa PeopleCodes>
    [break;]
When-Other

```

```

    <executa PeopleCodes>
End-Evaluate;

```

Exemplos:

```

Evaluate &A
When 1
    WinMessage("A é igual a 1");
When 2
    WinMessage("A é igual a 2");
When-Other
    WinMessage("A não é 1, nem 2");
End-Evaluate;

```

```

Evaluate &A
When < 0
    WinMessage("A é negativo");
When = 0
    WinMessage("A é zero");
When > 0
    WinMessage("A é positivo");
End-Evaluate;

```

Operadores Relacionais

Operador	Significado
=	Igual
>	Maior que
>=	Maior ou Igual a
<=	Menor ou Igual a
<	Menor que
<>	Diferente
!=	Diferente

Operadores Matemáticos

Operador	Função	Exemplo
+	Adição	$\&X = \&A + \&B + 10;$
-	Subtração	$\&X = \&A + \&B - 10;$
*	Multiplicação	$\&X = \&A + \&B * 2;$
/	Divisão	$\&X = \&A + \&B / 2;$
**	Exponenciação	$\&X = \&A + \&B ** 2;$
	Concatenação	$\&X = \&A \text{"texto"} \&B;$

Ordem de Precedência

Primeiro o PeopleSoft calcula as exponenciações, depois as multiplicações e divisões e depois somas e subtrações, sempre da esquerda para a direita.

Operadores Lógicos

Operador	Função	Exemplo
AND	E	$\text{If } \&A=\&B \text{ AND } \&B=\&C \text{ Then}$
OR	Ou	$\text{If } \&A=\&B \text{ OR } \&B=\&C \text{ Then}$
NOT	Negação	$\text{If NOT } \&A \text{ Then}$

Ordem de Precedência

Primeiro o PeopleSoft calcula o NOT, depois AND e por ultimo o OR.

Estruturas de Repetição

For...End-For

Repete um trecho de código por um determinado número de vezes.

Se a variável do seu *For...End-For* for um número inteiro e se couber dentro dos limites do tipo de dados inteiro, declare essa variável como inteiro e o desempenho do loop será bem superior.

Sintaxe:

```
For <variavel> = <numero_inicial> To <numero_final> [Step <incremento>]  
    <executa PeopleCodes>  
End-For;
```

Exemplos:

```
For &I = 1 To 5  
    &A = &A * &I;  
End-For;  
  
For &I = 2 To 10 Step 2  
    &A = &A * &I;  
End-For;  
  
For &I = 10 To 5 Step -1  
    &A = &A * &I;  
End-For;
```

While...End-While

Repete um trecho de código por um determinado número de vezes, sendo que o trecho de código pode não ser executado.

A variação de execução no While...End-While vai de zero a “N” vezes.

Sintaxe:

```
While <condicao>  
    <executa PeopleCodes>  
End-While;
```

Exemplos:

```
While &A < 10  
    <executa PeopleCodes>  
End-While;  
  
While &A = &B  
    <executa PeopleCodes>  
End-While;
```

Repeat...Until

Repete um trecho de código por um determinado número de vezes, sendo que o trecho de código obrigatoriamente é executado, no mínimo uma vez.

A variação de execução no Repeat...Until vai de 1 a “N” vezes.

Sintaxe:

```
Repeat  
    <executa PeopleCodes>  
Until <condicao>;
```

Exemplos:

```
Repeat  
    <executa PeopleCodes>  
Until &A >= 10;
```

```
Repeat
    <executa PeopleCodes>
Until &A <> &B;
```

Tratamento de Erros

Try...Catch

Esta estrutura permite capturar erros impedindo que o aplicativo aborte. Isso torna nosso aplicativo mais confiável, além disso, o usuário final poderá ver mensagens de erro amigáveis ao invés de mensagens de erro complexas que somente são úteis a desenvolvedores.

Sintaxe:

```
Try
    <executa PeopleCodes que podem causar erro>
Catch Exception <variável>
    <executa PeopleCodes>
End-For;
```

Exemplos:

```
Try
    &A = 1 / 0;
Catch Exception &Ex
    WinMessage ("Ocorreu o seguinte erro: " | &Ex.ToString());
End-Try;
```

Exercícios

- 1) Abra o Application Designer e carregue o seu projeto
- 2) No evento *FieldChange* do BUTTON1, apague o código existente
- 3) Adicione o código exemplo do *Try...Catch* e teste:
- 4) No evento *FieldChange* do BUTTON1, apague o código existente
- 5) Crie um array
- 6) Adicione os seguintes elementos nesta ordem:
 - a. 10, 200, 50, 15, 300, 5, 80, 170, 110, 20
- 7) Agora crie um aplicativo para ordenar esse array, em ordem crescente, e exibi-lo na tela.

Acesso ao Banco de Dados

SQLExec

O comando SQLExec permite executar acesso direto ao banco de dados passando por cima do processador de componente.

Quando o código SQL executado pelo SQLExec for um *SELECT*, este código deverá trazer apenas uma linha, pois demais linhas serão ignoradas.

Caso um *SELECT* traga mais de uma linha e você desejar acessar todas as linhas retornadas por ele, será necessário usar a classe SQL ao invés do comando SQLExec.

Limitações:

SQLExec não gera um erro que interrompa a execução do programa caso o usuário não tenha privilégios para executar o SQL no banco de dados. Use a variável de ambiente %SQLRows para saber se um *INSERT*, *UPDATE* ou *DELETE* afetou alguma linha no banco de dados.

Ao usar um *INSERT*, *UPDATE* ou *DELETE*, o comando SQLExec somente poderá ser executado nos seguintes eventos:

- SavePreChange
- Workflow
- SavePostChange
- FieldChange

Sintaxe:

```
SQLExec(<codigo_SQL>, <parâmetros_entrada>, <parametros_saida>);
```

Exemplos:

```
SQLExec("Select Nome, Idade From PS_TABELA Where Codigo=:1", &Codigo, &Nome, &Idade);
```

```
SQLExec("Update PS_TABELA Set Nome=:1 Where Codigo=:2", &Nome, &Codigo);
If %SQLRows = 0 Then
    Error "O Update não atualizou nenhuma linha, talvez você não tenha privilégios.";
End-if;
```

```
&A = CreateArrayAny("Joao", 20);
SQLExec("Update PS_TABELA Set Nome=:1 Where Codigo=:2", &A);
If %SQLRows = 0 Then
    Error "O Update não atualizou nenhuma linha, talvez você não tenha privilégios.";
End-if;
```

Exercícios

- 1) Abra o Application Designer e carregue o seu projeto
- 2) No evento *FieldChange* do BUTTON1, apague o código existente e adicione o código abaixo.

```
Local string &Select, &Instituicao, &Programa, &Descricao, &Curta;
```

```
&Instituicao = "SENAC";
&Programa = "00016";
```

```
&Select = "Select DESCR, DESCRSHORT ";
&Select = &Select | "From PS_ACAD_PROG_TBL A ";
&Select = &Select | "Where INSTITUTION = :1 ";
&Select = &Select | " And ACAD_PROG = :2 ";
&Select = &Select | " And EFFDT = (Select Max(EFFDT) ";
&Select = &Select | " From PS_ACAD_PROG_TBL ";
&Select = &Select | " Where INSTITUTION = A.INSTITUTION ";
&Select = &Select | " And ACAD_PROG = A.ACAD_PROG ";
&Select = &Select | " And EFFDT <= Sysdate) ";
SQLExec(&Select, &Instituicao, &Programa, &Descricao, &Curta);
```

```
MessageBox(%MsgStyle_OK, "", 0, 0, "As descrições são: %1 e %2 ", &Descricao, &Curta);
```

3) Teste o seu código

Classe SQL

A classe SQL é útil quando executar um *SELECT* no banco de dados retorna mais de uma linha, neste caso, o *SQLExec* não pode ser usado, pois retorna apenas a primeira linha, ignorando as demais.

A classe SQL usa um cursor para acessar os dados.

Limitações:

Não é possível ter mais de um cursor aberto de cada vez, se necessário salve todos os dados do primeiro cursor em um array antes de abrir o próximo. Entretanto, é possível reutilizar um cursor, reaproveitando-o com outro código SQL sem precisar fechá-lo.

Ao usar um *INSERT*, *UPDATE* ou *DELETE*, o comando a classe SQL somente poderá ser executada nos seguintes eventos:

- *SavePreChange*
- *Workflow*
- *SavePostChange*
- *FieldChange*

GetSQL / CreateSQL

A função *GetSQL* e a função *CreateSQL* abrem um cursor que permite acessar todas as linhas que um *SELECT* retornar.

Estas funções deveram ser usadas em conjunto com a função *Fetch* que fará o carregamento de cada linha retornada e a função *Close* que fechará o cursor.

Sintaxe:

```
<variável> = GetSQL(<codigo_SQL>, <parâmetros_entrada>);  
<variável> = CreateSQL(<codigo_SQL>, <parâmetros_entrada>);
```

Exemplos:

```
Local SQL &SQL;  
Local string &Estado, &Nome, &Cidade;  
Local integer &Idade;  
&Estado = "SP";  
&SQL = GetSQL("Select Nome, Idade, Cidade From PS_TABELA Where Estado=:1", &Estado);  
While &SQL.Fetch(&Nome, &Idade, &Cidade)  
    If &Idade < 18 Then  
        WinMessage("Cliente " | &Nome | " mora em " | &Cidade | " é menor idade");  
    End-If;  
End-While;  
&SQL.Close();  
  
Local SQL &SQL;  
Local string &Estado;  
Local array of any &Dados;  
&Estado = "SP";  
&SQL = GetSQL("Select Nome, Idade, Cidade From PS_TABELA Where Estado=:1", &Estado);  
While &SQL.Fetch(&Dados)  
    If &Dados[2] < 18 Then  
        WinMessage("Cliente " | &Dados[1] | " mora em " | &Dados[3] | " é menor idade");  
    End-If;  
End-While;  
&SQL.Close();
```

Fetch

A função *Fetch* busca a próxima linha disponível em um cursor e retorna *True* indicando que o *fetch* foi bem sucedido.

Quando o cursor está na ultima linha, a execução da função *Fetch* retornará *False* indicando que o *fetch* não conseguiu buscar nova linha de dados.

Esta função deverá ser usada em conjunto com a função *GetSQL* que fará o carregamento de cada linha retornada e a função *Close* que fechará o cursor.

Sintaxe:

```
<variável> = Fetch(<parâmetros_saida>);
```

Exemplos:

Veja o comando *GetSQL* explicado anteriormente.

Execute

A função *Execute* permite executar comandos de banco de dados, como Insert, Update e Delete.

Sintaxe:

```
<variável>.Execute(<parâmetros_entrada>);
```

Exemplos:

```
Local SQL &SQL;
Local integer &Codigo;
&SQL = CreateSQL("Delete From PS_MINHA_RECORD Where Codigo = :1");

&Codigo = 100;
&SQL.Execute(&Codigo);

&Codigo = 150;
&SQL.Execute(&Codigo);

&Codigo = 2000;
&SQL.Execute(&Codigo);
```

Close

A função *Close* fecha o cursor aberto, caso ainda haja linhas não lidas pelo *Fetch*, elas serão descartadas.

Sintaxe:

```
<variável>.Close();
```

Exemplos:

Veja o comando *GetSQL* explicado anteriormente.

Reutilização de Cursores

É possível reutilizar um cursor evitando fechar e recriar, para isto deve-se definir a propriedade SQL *ReuseCursor* para True.

Sintaxe:

```
<variável>.ReuseCursor = <valor>;
```

Exemplos:

```
Local SQL &SQL;
Local string &Estado;
Local array of any &Dados;

&Dados = CreateArrayAny();
&Estado = "SP";

&SQL = CreateSQL("Select Nome, Idade, Cidade From PS_TABELA Where Estado=:1", &Estado);
&SQL.ReuseCursor = True;

While &SQL.Fetch(&Dados)
    /* peoplecodes */
End-While;
```

```

/* como cursor está sendo reutilizado, não é preciso fechar */
&Estado = "RS";

&SQL.Execute(&Estado);
While &SQL.Fetch(&Dados)
    /* peoplecodes */
End-While;

/* não vai mais usar o cursor, fecha ele */
&SQL.Close();

```

SQL Injection

É uma técnica onde usuários mal intencionados tentam inserir código SQL no seu aplicativo através dos campos da tela.

O PeopleSoft evita que injeção SQL ocorra em records, porém quando um programador usa código SQL em Peoplecode através de SQLExec ou da classe SQL, este código pode ser vítima de SQL Injection.

Para saber se o código que você desenvolveu tem potencial para SQL Injection, use o Application Designer para detectá-lo e reescrever seu código.

Para fazer a detecção, no Application Designer, clique no menu *Edit*, depois na opção *Find In*, ao abrir a janela, no campo *Find Type*, mude para *SQL Injection In PeopleCode* e no campo *Project*, escolha o seu projeto. O resultado será exibido na janela de saída.

Exercícios

- 1) Abra o Application Designer e carregue o seu projeto
- 2) No evento *FieldChange* do BUTTON1, apague o código existente e adicione o código abaixo.

```

Local string &Select, &Instituicao, &Programa, &Descricao, &Curta, &Msg;
Local SQL &SQL;

&Instituicao = "SENAC";

&Select = "Select ACAD_PROG, DESCR, DESCRSHORT ";
&Select = &Select | "From    PS_ACAD_PROG_TBL A ";
&Select = &Select | "Where   INSTITUTION = :1 ";
&Select = &Select | "      And EFFDT      = (Select Max(EFFDT) ";
&Select = &Select | "                      From    PS_ACAD_PROG_TBL ";
&Select = &Select | "                      Where   INSTITUTION = A.INSTITUTION ";
&Select = &Select | "                      And ACAD_PROG  = A.ACAD_PROG ";
&Select = &Select | "                      And EFFDT      <= Sysdate) ";
&SQL = GetSQL(&Select, &Instituicao);

&Msg = "A lista de programas cadastrados é:";

While &SQL.Fetch(&Programa, &Descricao, &Curta)
    &Msg = &Msg | Char(10) | &Programa | " - " | &Descricao | " - " | &Curta;
End-While;
&SQL.Close();

WinMessage (&Msg);

```

- 3) Teste o seu código

Meta-SQL

O PeopleSoft é um ambiente que pode rodar em cima de diferentes bancos de dados, como Oracle, SQL Server, DB2, entre outros, como cada banco de dados tem sua própria sintaxe, pode acontecer de um determinado código SQL rodar em apenas um tipo de banco de dados. Para evitar essa situação o PeopleSoft possui um conjunto de instruções Meta-SQL que abstraem o banco de dados, rodando em qualquer banco.

Para traduzir um Meta-SQL para um código SQL que possa ser testado em aplicativos como TOAD ou SQL Developer, clique com o botão direito no editor de SQL e escolha o

comando *Resolve Meta-SQL*, uma nova aba na janela de saída irá surgir com o código traduzido.

A lista de Meta-SQL é bem extensa, por isso, veremos os comandos mais usados:

%Abs

Esta função retorna o valor absoluto de um numérico.

Exemplos:

```
Select CAMPO From PS_TABELA Where %Abs (VALOR1 - VALOR2) > VALOR3
```

%Concat

Esta variável serve para concatenar texto em um SQL.

Exemplos:

```
Select 'R$' %Concat VALOR From PS_TABELA  
Select CIDADE %Concat '/' %Concat ESTADO From PS_TABELA
```

%CurrentDateIn

Esta variável representa a data atual em cláusulas *WHERE* ou em comandos *INSERT* ou *UPDATE*, e equivale ao comando *Trunc(SYSDATE)* em Oracle.

Exemplos:

```
Select NOME, ENDEREÇO From PS_TABELA Where NASCIMENTO < %CurrentDateIn  
Update PS_TABELA Set DATA_VENDA = %CurrentDateIn Where CODIGO = 1
```

%CurrentDateOut

Esta variável representa a data atual em códigos SQL do tipo *SELECT*, e equivale ao comando *Trunc(SYSDATE)* em Oracle.

Exemplos:

```
Select NOME, IDADE, %CurrentDateOut, USERID From PS_TABELA
```

%CurrentDateTimeln

Esta variável representa a data atual em cláusulas *WHERE* ou em comandos *INSERT* ou *UPDATE*, e equivale ao *SYSDATE* em Oracle.

Exemplos:

```
Select NOME, ENDEREÇO From PS_TABELA Where NASCIMENTO < %CurrentDateIn  
Update PS_TABELA Set DATA_VENDA = %CurrentDateIn Where CODIGO = 1
```

%CurrentDateTimeOut

Esta variável representa a data atual em códigos SQL do tipo *SELECT*, e equivale ao *SYSDATE* em Oracle.

Exemplos:

```
Select NOME, IDADE, %CurrentDateTimeOut, USERID From PS_TABELA
```

%DateAdd

Esta função permite acrescentar/subtrair dias em uma data.

Exemplos:

```
Select %DateAdd(DATA_VENDA, 1) From PS_TABELA  
Select %DateAdd(DATA_VENDA, -1) From PS_TABELA  
Select %DateAdd(%DateIn('2011-01-30'), 1) From PS_TABELA
```

%DateDiff

Esta função retorna em dias a diferença entre 2 datas.

Exemplos:

```
Select %DateDiff(DATA_INICIO, DATA_FINAL) From PS_TABELA  
Select %DateDiff(%DateIn('2011-01-01'), %DateIn('2011-01-15')) From PS_TABELA
```

%DateIn

Esta função converte um texto em data.

O texto deverá estar no formato YYYY-MM-DD.

Exemplos:

```
Select %DateDiff(%DateIn('2011-01-01'), %DateIn('2011-01-15')) From PS_TABELA  
Update PS_TABELA Set CAMPO_DATA = %DateIn('2011-02-20')
```

%DateNull

Esta variável permite inserir valor *Null* em campos *Date*. Use-o somente em comandos *INSERT* e *UPDATE*.

Exemplos:

```
Update PS_TABELA Set CAMPO_DATA = %DateNull
```

%DateOut

Esta função expande um campo data em um comando *SELECT*.

Exemplos:

```
Select NOME, %DateOut(NASCIMENTO) From PS_TABELA
```

%DatePart

Esta função retorna a parte *Date* de um campo *DateTime*.

Exemplos:

```
Select %DateOut(%DatePart(DATA_HORA_ACESSO)) From PS_TABELA
```

%DateTimeDiff

Esta função retorna os minutos entre 2 campos *DateTime*.

Exemplos:

```
Select %DateTimeDiff(DATA_VENDA, %CurrentDateIn) From PS_TABELA
```

%DateTimeIn

Esta função converte um texto em data/hora.

O texto deverá estar no formato YYYY-MM-DD-hh-mm-ss.ssssss.

Exemplos:

```
Update PS_TABELA Set CAMPO_DATA = %DateTimeIn('2011-02-20-23-59-59.000000')
```

%DateTimeNull

Esta variável permite inserir valor *Null* em campos *DateTime*. Use-o somente em comandos *INSERT* e *UPDATE*.

Exemplos:

```
Update PS_TABELA Set CAMPO_DATA = %DateTimeNull
```

%DateTimeOut

Esta função expande um campo *DateTime* em um comando *SELECT*.

Exemplos:

```
Select NOME, %DateTimeOut(HORARIO) From PS_TABELA
```

%Delete

Esta função faz o papel do DELETE em uma tabela no banco de dados, o parâmetro é o nome da tabela no PeopleSoft, não no banco de dados.

Exemplos:

```
Local SQL &SQL_Select, &SQL_Delete;  
&SQL_Select = CreateSQL("%SelectAll(:1) Where EMPLID = :2", &ABS_HIST, &EMPLID);  
&SQL_Delete = CreateSQL("%Delete (:1)");  
While &SQL_Select.Fetch(&ABS_HIST);  
    If &SQL_Select.CAMPO.Value = "Y" Then  
        &SQL_Delete.Execute(&ABS_HIST);  
    End-If;  
End-While;  
&SQL_Delete.Close();
```

%DecDiv

Esta função executa a divisão do primeiro valor pelo segundo.

Exemplos:

```
Select %DecDiv(10, 2) From PS_TABELA -- resultado será 5
```

%DecMult

Esta função executa a multiplicação do primeiro valor pelo segundo.

Exemplos:

```
Select %DecMult(10, 2) From PS_TABELA - resultado será 20
```

%DTTM

Esta função combina um campo *Date* com um campo *Time*, produzindo um campo *DateTime*.

Exemplos:

```
Select %DTTM(DATA, HORA) From PS_TABELA
```

%FirstRows

Esta função retorna as primeiras “N” linhas de um *SELECT*, porém em alguns tipos de bancos de dados, não tem efeito.

Exemplos:

```
Select %FirstRows(10) NOME, SOBRENOME From PS_TABELA
```

%Insert

Esta função faz o papel do INSERT em uma record, o parâmetro é o nome da tabela no PeopleSoft, não no banco de dados.

Exemplos:

```
Local SQL &SQL_Select, &SQL_Insert;  
&SQL_Select = CreateSQL("%SelectAll(:1) Where EMPLID = :2", &ABS_HIST, &EMPLID);  
&SQL_Insert = CreateSQL("%Insert (:1)");  
While &SQL_Select.Fetch(&ABS_HIST);  
    If &SQL_Select.CAMPO.Value = "Y" Then  
        &SQL_Insert.Execute(&ABS_HIST);  
    End-If;  
End-While;
```

```
End-While;
&SQL_Insert.Close();
```

%InsertSelect

Esta função gera um *INSERT* a partir de um *SELECT*, este comando tem limite de 99 campos e ignora campos *Long Character*, *Image* e *Attachment*.

Exemplos:

```
%InsertSelect(TABELA_INSERT, TABELA_SELECT1, TABELA_SELECT2)
From PS_TABELA_SELECT1 A, PS_TABELA_SELECT2 B
Where %Join(COMMON_KEYS, TABELA_SELECT1 A, TABELA_SELECT2 B)
```

Será convertido em:

```
Insert Into TABELA_INSERT
Select CAMPO1, CAMPO2, . . . . .
From PS_TABELA_SELECT1 A, PS_TABELA_SELECT2 B
Where A.CHAVE1 = B.CHAVE1
And A.CHAVE2 = B.CHAVE2 . . . . .
```

%InsertValues

Esta função converte o parâmetro no formato esperado pelo banco de dados.

Exemplos:

```
Insert Into PS_TABELA (CAMPO_TEXTO, CAMPO_DATA, CAMPO_NUMERO)
Values (%InsertValues(:1), %InsertValues(:2), %InsertValues(:3))
```

Será convertido em:

```
Insert Into PS_TABELA (CAMPO_TEXTO, CAMPO_DATA, CAMPO_NUMERO)
Values ('BLA BLA', %DateIn('2010-12-25'), 10)
```

%Join

Esta função cria dinamicamente um *JOIN* entre 2 tabelas.

O primeiro parâmetro pode ser definido como:

- COMMON_KEYS: faz join somente pelas chaves
- COMMOM_FIELDS: faz join por todos os campos de mesmo nome

Exemplos:

```
%Join(COMMOM_KEYS, TABELA1 A, TABELA2 B)
```

Será convertido em:

```
A.CHAVE1 = B.CHAVE1
A.CHAVE2 = B.CHAVE2
A.CHAVE3 = B.CHAVE3 . . . . .
```

%KeyEqual

Esta função expande em condições que serão usadas em clausulas *WHERE*.

Exemplos:

```
Local record &REC;
&REC = CreateRecord(RECORD.MINHA_RECORD);
&REC.NUMERO.Value = 27;
&REC.DATA.Value = %Date;
&REC.TEXTO.Value = "ABC";
SQLExec("Delete From MINHA_RECORD A Where %KeyEqual(:1, A)", &REC);
```

Será convertido em:

```
Delete
From PS_MINHA_RECORD
Where A.NUMERO = 27
And A.DATA = %DateIn('2010-12-25')
And A.TEXT = "ABC"
```

%NumToChar

Esta função converte números para caracter.

Exemplos:

```
Select %NumToChar(CAMPO_NUM) From PS_TABELA
```

%Round

Esta função arredonda numeros.

Exemplos:

```
%Round(10.1234, 2)    ➔  10.12
%Round(10.9876, 2)    ➔  10.99
Select %Round(CAMPO_NUM) From PS_TABELA
```

%Select

Esta função faz o papel do SELECT buscando campos específicos em uma tabela no banco de dados, o parâmetro é o nome da tabela no PeopleSoft, não no banco de dados.

Esta função é principalmente usada em Application Engines.

Exemplos:

```
%Select (INSTITUTION, EMPLID)
Select INSTITUTION, EMPLID
From   PS_TABELA
Where  ACAD_CAREER = :1
```

%SelectAll

Esta função faz o papel do SELECT buscando todos os campos em uma tabela no banco de dados, o parâmetro é o nome da tabela no PeopleSoft, não no banco de dados.

Exemplos:

```
Local SQL &SQL_Select, &SQL_Update;
&SQL_Select = CreateSQL("%SelectAll(:1) Where EMPLID = :2", &ABS_HIST, &EMPLID);
&SQL_Update = CreateSQL("%Update(:1)");
While &SQL_Select.Fetch(&ABS_HIST);
  If &SQL_Select.CAMPO.Value = "Y" Then
    &SQL_Update.Execute(&ABS_HIST);
  End-If;
End-While;
&SQL_Update.Close();
```

%Substring

Esta função retorna a partir da posição X, a quantidade de Y caracteres.

Exemplos:

```
%Substring("ABCDEFGH", 2, 3) ➔ "BCD"
Select %Substring(CAMPO_TEXTO, 5, 10) From PS_TABELA
```

%Table

Esta função retorna o nome SQL de uma tabela definida no PeopleSoft.

Exemplos:

```
Local record &Record;
&Record = GetRecord(Record.MINHA_RECORD);
SQLExec("Delete From %Table(:1) Where CAMPO1 = 0", &Record.Name);
```

%Truncate

Esta função trunca numeros.

Exemplos:

```
%Truncate(10.1234, 2) ➔ 10.12
%Truncate(10.9876, 2) ➔ 10.98
Select %Truncate(CAMPO_NUM) From PS_TABELA
```

%Update

Esta função faz o papel do UPDATE em uma tabela no banco de dados, o parâmetro é o nome da tabela no PeopleSoft, não no banco de dados.

Exemplos:

```
Local SQL &SQL_Select, &SQL_Update;  
&SQL_Select = CreateSQL("%SelectAll(:1) Where EMPLID = :2", &ABS_HIST, &EMPLID);  
&SQL_Update = CreateSQL("%Update(:1)");  
While &SQL_Select.Fetch(&ABS_HIST);  
    If &SQL_Select.CAMPO.Value = "Y" Then  
        &SQL_Update.Execute(&ABS_HIST);  
    End-If;  
End-While;  
&SQL_Update.Close();
```

%Upper

Esta função converte textos para maiúsculo.

Exemplos:

```
%Upper("PARAlelepiPEdo") → "PARALELEPIEDO"  
Select CAMPO1, CAMPO2 From PS_TABELA Where %Upper(CAMPO3) = 'DIGISYSTEM'
```

Exercícios

- 1) Abra o Application Designer e carregue o seu projeto
- 2) No evento *FieldChange* do BUTTON1, apague o código existente e adicione o código abaixo.

```
Local string &Select, &Msg, &Aluno, &Nome;  
Local date &Nascimento;  
Local SQL &SQL;  
  
&Select = "Select EMPLID, NAME_DISPLAY, %DateOut(BIRTHDATE) ";  
&Select = &Select | "From PS_PERSONAL_DATA ";  
&Select = &Select | "Where %Upper(STATE) = 'SP' ";  
&Select = &Select | " And BIRTHDATE < %DateIn('1970-01-01') ";  
&SQL = GetSQL(&Select);  
  
&Msg = "Alunos de SP que nasceram antes de 1970 são:";  
  
While &SQL.Fetch(&Aluno, &Nome, &Nascimento)  
    &Msg = &Msg | Char(10) | &Aluno | " - " | &Nome | " - " | &Nascimento;  
End-While;  
&SQL.Close();  
  
WinMessage(&Msg);
```

- 3) Teste o seu código
- 4) Crie uma record que armazene o código acima
- 5) Salve com o nome de EXEMPLO_META_SQL
- 6) Teste a record no seu aplicativo
- 7) Resolva o Meta-SQL da record no Application Designer, depois pegue o código resolvido e teste no TOAD/SQL Developer ou outra ferramenta de banco de dados

Classe Record

A classe *Record* é usada quando precisamos manipular uma record, seja ela com dados vindos do banco de dados ou com dados gerados na aplicação somente em memória RAM.

CreateRecord

Muito útil quando queremos criar uma record vazia em memória RAM, essa record poderá ser posteriormente gravada no banco de dados, ou não.

Exemplos:

```
Local record &R;
&R = CreateRecord(Record.MINHA_RECORD);
```

GetRecord

Esta função busca uma record já existente no buffer do componente permitindo que se trabalhe nos dados já existentes.

Exemplos:

```
Local record &R;
&R = GetRecord(Record.MINHA_RECORD);
```

CopyFieldsTo

Esta função copia todos os campos de uma record para outra record, caso as duas records não tenham os mesmos campos, somente os campos de mesmo nome serão copiados.

Exemplos:

```
Local record &R1, &R2;
&R1 = GetRecord(Record.MINHA_RECORD);
&R2 = CreateRecord(Record.OUTRA_RECORD);
&R1.CopyFieldsTo(&R2);
```

ExecuteEdits

Executa a validação de todos os campos de uma record, sendo que esta validação se dá somente a nível de prompts, XLAT's e campos obrigatórios, nenhum PeopleCode será executado.

Exemplos:

```
Local record &R;
&R = CreateRecord(Record.MINHA_RECORD);
&R.CAMPO1.Value = 1;
&R.CAMPO2.Value = "ABC";
&R.ExecuteEdits();
If &R.IsEditError Then
    /* pelo menos 1 campo falhou na validação */
End-if;
```

Delete

Remove uma linha de dados de uma record numa tabela de banco de dados.

Exemplos:

```
Local record &R;
&R = CreateRecord(Record.MINHA_RECORD);
&R.CAMPO1.Value = 1;
&R.CAMPO2.Value = "ABC";
If &R.SelectByKey() Then
    &R.Delete();
Else
    WinMessage("Dados não encontrados no banco de dados.");
End-If;
```

FieldCount

Retorna a quantidade de fields em uma record.

Exemplos:

```
Local record &R;
&R = GetRecord(Record.MINHA_RECORD);
WinMessage("A record possui " | &R.FieldCount | " campos");
```

Insert

Insere os dados de uma record numa tabela de banco de dados.

Exemplos:

```
Local record &R;  
&R = CreateRecord(Record.MINHA_RECORD);  
&R.CAMPO1.Value = 1;  
&R.CAMPO2.Value = "ABC";  
If &R.SelectByKey() Then  
    &R.Update();  
Else  
    &R.Insert();  
End-If;
```

IsChanged

Retorna TRUE ou FALSE, indicando se algum campo da record foi alterado desde que a página foi carregada.

Exemplos:

```
If MINHA_RECORD.IsChanged Then  
    WinMessage("A record teve pelo menos um campo alterado");  
Else  
    WinMessage("Nenhum campo foi alterado");  
End-If;
```

SelectByKey

Efetua a busca no banco de dados pelas chaves de pesquisa preenchidas, caso encontre mais de uma linha, somente a primeira será retornada, para acessar as demais linhas use a classe SQL.

Exemplos:

```
Local record &R;  
&R = CreateRecord(Record.MINHA_RECORD);  
&R.CAMPO1.Value = 1;  
&R.CAMPO2.Value = "ABC";  
If &R.SelectByKey() Then  
    &R.Update();  
Else  
    &R.Insert();  
End-If;
```

Update

Atualiza os dados de uma record numa tabela de banco de dados.

Exemplos:

```
Local record &R;  
&R = CreateRecord(Record.MINHA_RECORD);  
&R.CAMPO_CHAVE1.Value = 1;  
&R.CAMPO_CHAVE2.Value = "ABC";  
If &R.SelectByKey() Then  
    &R.Update();  
Else  
    &R.Insert();  
End-If;
```

Record Derived/Work

Este tipo de record é usado principalmente para:

- Armazenamento de valores temporários em componentes
- Exibição de dados que não precisam ser gravados no banco de dados, como resultados de cálculos
- Troca de valores entre páginas dentro de um componente
- Operação de Push Buttons e Hyperlinks

Um campo em uma record Derived/Work possui o mesmo escopo de uma variável *Component*.

O campo de uma record Derived/Work deve ser colocado em uma página e terá exatamente a mesma funcionalidade de um campo qualquer de outro tipo de record.

Esta record é alocada em memória exatamente como outra record de banco de dados e pode ser colocada em qualquer nível em uma página.

Exercícios

- 1) Abra o Application Designer e carregue o seu projeto
- 2) Crie uma record to tipo SQL Table
- 3) Adicione os campos: EMPLID, FIRST_NAME e LAST_NAME
- 4) Configure o EMPLID como Key e Search Key
- 5) Salve a record com o nome TESTE
- 6) Faça o build da record
- 7) No evento *FieldChange* do BUTTON1, apague o código existente e adicione o código abaixo.

```
Local record &Record;

/* cria uma record vazia em memória RAM */
&Record = CreateRecord(Record.TESTE);

/* preenche as chaves de pesquisa da record */
&Record.EMPLID.Value = "1";

/* executa a pesquisa na record */
If &Record.SelectByKey() Then
    /* preenche campos para atualizar dados existentes */
    &Record.FIRST_NAME.Value = "JOAO";
    &Record.LAST_NAME.Value = "SILVA";
    &Record.Update();
    WinMessage("EMPLID já existia, então foi apenas atualizado");
Else
    /* preenche campos para inserir linha nova */
    &Record.EMPLID.Value = "1";
    &Record.FIRST_NAME.Value = "JOAO";
    &Record.LAST_NAME.Value = "SILVA";
    &Record.Insert();
    WinMessage("EMPLID não existia, então foi inserido linha nova");
End-If;
```

- 8) Teste o seu código
- 9) Abra o TOAD ou o SQL Developer e faça um *Select* na sua record
- 10) Altere o FIRST_NAME para "MARIA" e o LAST_NAME para "JOSEFINA" e teste novamente o código
- 11) Retorne ao TOAD ou o SQL Developer e faça um *Select* na sua record
- 12) No evento *FieldChange* do BUTTON1, comente o código existente e adicione o código abaixo.

```
Local record &Record;
Local boolean &Achou;

/* cria uma record vazia em memória RAM */
&Record = CreateRecord(Record.TESTE);

/* preenche as chaves de pesquisa da record */
&Record.EMPLID.Value = "2";

/* executa a pesquisa na record */
&Achou = &Record.SelectByKey();

/* preenche campos da record antes de salvar */
&Record.EMPLID.Value = "2";
&Record.FIRST_NAME.Value = "JOSELITO";
&Record.LAST_NAME.Value = "FERREIRA";

/* verifica se deve atualizar dados existentes ou adicionar linha nova */
If &Achou Then
```

```
        &Record.Update();  
CommitWork();  
        WinMessage("EMPLID já existia, então foi apenas atualizado");  
Else  
        &Record.Insert();  
CommitWork();  
        WinMessage("EMPLID não existia, então foi inserido linha nova");  
End-If;
```

- 13) Teste seu código
- 14) Retorne ao TOAD ou o SQL Developer e faça um *Select* na sua record
- 15) Altere o FIRST_NAME para “ADELINO” e o LAST_NAME para “ADELOSO” e teste novamente o código
- 16) Retorne ao TOAD ou o SQL Developer e faça um *Select* na sua record
- 17) Observe que os 2 códigos acima fazem exatamente a mesma coisa, porém no segundo caso, o código é mais limpo e menos repetitivo.

Funções Internas

Catálogo de Mensagens

As vantagens do catálogo de mensagens são:

- Fornece um local central para armazenamento de mensagens
- Mensagens não ficam hardcode no aplicativo
- Facilita a manutenção de mensagens
- Facilita a tradução de mensagens
- Permite que mensagens possam ser reutilizadas

MsgGet

Busca o conjunto de uma determinada mensagem no catálogo de mensagens e caso precise passar parâmetros para a mensagem, adicione %1 para o primeiro parâmetro, %2 para o segundo parâmetro e assim por diante.

Este conjunto contém a mensagem principal, a mensagem de explicação e o tipo de severidade da mensagem;

Sintaxe:

```
<variável> = MsgGet(<message_set>, <message_num>, <default_msg_txt> [, <paramlist>]);
```

Exemplos:

```
Warning MsgGet(1234, 123, "Mensagem não foi encontrada");
```

```
/* Exemplo: Suponha que mensagem (10, 1) = "O cliente %1 possui status %2" */  
Error MsgGet(10, 1, "Mensagem não foi encontrada", &Nome, &Status);
```

MsgGetText

Busca somente o texto da mensagem principal de uma determinada mensagem no catálogo de mensagens. É possível passar parâmetros assim como na função MsgGet.

Sintaxe:

```
<variável> = MsgGetText(<msg_set>, <msg_num>, <default_txt> [, <paramlist>]);
```

Exemplos:

```
Warning MsgGetText(1234, 123, "Mensagem não foi encontrada");
```

```
/* Exemplo: Suponha que mensagem (10, 1) = "O cliente %1 possui status %2" */  
Error MsgGetText(10, 1, "Mensagem não foi encontrada", &Nome, &Status);
```

MsgGetExplainText

Busca somente o texto explicativo da mensagem principal de uma determinada mensagem no catálogo de mensagens. É possível passar parâmetros assim como na função MsgGet.

Sintaxe:

```
<variável> = MsgGetExplainText(<msg_set>, <msg_num>, <default_txt> [, <paramlist>]);
```

Exemplos:

```
Warning MsgGetExplainText(1234, 123, "Mensagem não foi encontrada");
```

```
/* Exemplo: Suponha que mensagem (10, 1) = "O cliente %1 possui status %2" */  
Error MsgGetExplainText(10, 1, "Mensagem não foi encontrada", &Nome, &Status);
```

MessageBox

Exibe mensagens configuradas no catálogo de mensagens.

O parâmetro <title> não é usado, existe apenas por compatibilidade com códigos antigos, por isso, passe uma string vazia neste parâmetro.

Sintaxe:

```
<variável> = MessageBox(<style>, <title>, <msg_set>, <msg_num>, <default> [, <param>]);
```

Exemplos:

```
MessageBox(%MsgStyle_OK, "", 1234, 123, "Mensagem não foi encontrada");

/* Exemplo: Suponha que mensagem (10, 1) = "Deseja salvar dados do cliente %1 ?" */
&Resposta = MessageBox(%MsgStyle_YesNo, 10, 1, "Mensagem não encontrada", &Nome);
If &Resposta = %MsgResult_Yes Then
    /* usuário respondeu SIM */
Else
    /* usuário respondeu NÃO */
End-If;
```

Funções de Validação

All

Valida se todos os parâmetros estão preenchidos. Para saber se um parâmetro está preenchido, considera-se:

- Texto: não for vazio ou não contiver somente espaços
- Numero: diferente de zero
- Outros tipos: não Null

Sintaxe:

```
<variável> = All(<param1>[, <param2>][, <param_N>]);
```

Exemplos:

```
If All(&Nome, &Sobrenome, &Idade) Then
    /* todos os parâmetros estão preenchidos */
Else
    /* pelo um parâmetro está vazio */
End-If;
```

None

Valida se todos os parâmetros estão vazios, é oposto à função *All*. Para saber se um parâmetro está vazio, considera-se:

- Texto: não possui nenhum caractere ou todos são espaço
- Numero: valor é zero
- Outros tipos: valor é Null

Sintaxe:

```
<variável> = None(<param1>[, <param2>][, <param_N>]);
```

Exemplos:

```
If None(&Nome, &Sobrenome, &Idade) Then
    /* todos os parâmetros estão vazios */
Else
    /* pelo menos um parâmetro está preenchido */
End-If;
```

AllOrNone

Valida se os parâmetros estão ou todos preenchidos, ou todos estão vazios.

Sintaxe:

```
<variável> = AllOrNone(<param1>[, <param2>][, <param_N>]);
```

Exemplos:

```
If None(&Nome, &Sobrenome, &Idade) Then
    /* todos os parâmetros estão vazios ou todos os parâmetros estão preenchidos */
Else
    /* pelo menos um parâmetro está preenchido e pelo menos um está vazio */
End-If;
```

OnlyOne

Valida se apenas um parâmetro está preenchido.

Sintaxe:

```
<variável> = OnlyOne(<param1>[, <param2>][, <param_N>]);
```

Exemplos:

```
If OnlyOne(&Nome, &Sobrenome, &Idade) Then
    /* apenas um dos parâmetros está preenchido */
Else
    /* mais de um parâmetro está preenchido ou todos estão vazios */
End-If;
```

OnlyOneOrNone

Valida se apenas um parâmetro está preenchido ou todos estão vazios.

Sintaxe:

```
<variável> = OnlyOneOrNone(<param1>[, <param2>][, <param_N>]);
```

Exemplos:

```
If OnlyOneOrNone(&Nome, &Sobrenome, &Idade) Then
    /* apenas um dos parâmetros está preenchido ou todos estão vazios */
Else
    /* mais de um parâmetro está preenchido */
End-If;
```

Exercícios

- 1) Abra o Application Designer e carregue o seu projeto
- 2) No evento *FieldChange* do BUTTON1, apague o código existente e adicione o código abaixo.

```
Local string &Nome, &Sobrenome;
Local integer &Idade;

&Nome = "Joao";
&Sobrenome = "Silva";
&Idade = 15;

If All(&Nome, &Sobrenome, &Idade) Then
    WinMessage ("Executou o TRUE");
Else
    WinMessage ("Executou o FALSE");
End-If;
```

- 3) Teste seu código
- 4) No evento *FieldChange* do BUTTON1, apague o código existente e adicione o código abaixo.

```
Local string &Nome, &Sobrenome;
Local integer &Idade;

&Nome = "Joao";
&Sobrenome = "Silva";
&Idade = 0;

If All(&Nome, &Sobrenome, &Idade) Then
    WinMessage ("Executou o TRUE");
Else
    WinMessage ("Executou o FALSE");
End-If;
```

- 5) Teste seu código

- 6) No evento *FieldChange* do BUTTON1, apague o código existente e adicione o código abaixo.

```
Local string &Nome, &Sobrenome;  
Local integer &Idade;  
  
&Nome = " ";  
Sobrenome = "";  
&Idade = 0;  
  
If None(&Nome, &Sobrenome, &Idade) Then  
    WinMessage ("Executou o TRUE");  
Else  
    WinMessage ("Executou o FALSE");  
End-If;
```

- 7) Teste seu código

- 8) No evento *FieldChange* do BUTTON1, apague o código existente e adicione o código abaixo.

```
Local string &Nome, &Sobrenome;  
Local integer &Idade;  
  
&Nome = "";  
Sobrenome = "";  
&Idade = 15;  
  
If None(&Nome, &Sobrenome, &Idade) Then  
    WinMessage ("Executou o TRUE");  
Else  
    WinMessage ("Executou o FALSE");  
End-If;
```

- 9) Teste seu código

Funções de Texto

Left

Esta função retorna os primeiros “N” caracteres de um texto.

Sintaxe:

<variável> = Left(<texto>, <quantidade>);

Exemplos:

Left("ABCDEFGH", 3); ➔ "ABC"

Right

Esta função retorna os últimos “N” caracteres de um texto.

Sintaxe:

<variável> = Right(<texto>, <quantidade>);

Exemplos:

Right("ABCDEFGH", 3); ➔ "FGH"

Substring

Esta função retorna uma porção de texto dentro de outro texto.

Sintaxe:

<variável> = Substring(<texto>, <posição>, <quantidade>);

Exemplos:

Substring("ABCDEFGH", 3, 2); ➔ "CD"

LTrim

Esta função retorna uma string sem os espaços do início.

Sintaxe:

```
<variável> = LTrim(<texto>);
```

Exemplos:

```
LTrim("   ABCDEFGH   "); ➔ "ABCDEFGH"
```

RTrim

Esta função retorna uma string sem os espaços do final.

Sintaxe:

```
<variável> = RTrim(<texto>);
```

Exemplos:

```
RTrim("   ABCDEFGH   "); ➔ "   ABCDEFGH"
```

Len

Esta função retorna o comprimento de uma string.

Sintaxe:

```
<variável> = Len(<texto>);
```

Exemplos:

```
Len("ABCDEFGH"); ➔ 8
```

Rept

Esta função retorna uma porção de texto repetida "N" vezes.

Sintaxe:

```
<variável> = Rept(<texto>, <quantidade>);
```

Exemplos:

```
Rept("0", 3); ➔ "000"
Rept("Oi", 3); ➔ "OiOiOi"
```

Find

Esta função pesquisa uma porção de texto dentro de outro texto e retorna a posição onde esta porção se encontra, caso não exista, retorna zero.

Sintaxe:

```
<variável> = Find(<texto_a_pesquisar>, <texto>[, <posição_inicial>]);
```

Exemplos:

```
Find("SYS", "DIGISYSTEM"); ➔ 5
Find("I", "DIGISYSTEM"); ➔ 2
Find("I", "DIGISYSTEM", 3); ➔ 4
```

Char

Esta função retorna um caractere a partir da numeração ASCII.

Sintaxe:

```
<variável> = Char(<numero_ASCII>);
```

Exemplos:

```
Char(65); ➔ "A"
Char(32); ➔ " " (espaço)
Char(10); ➔ caracter de nova linha
```

String

Converte um valor numérico para texto.

Sintaxe:

```
<variável> = String(<numero>);
```

Exemplos:

```
String(1);      ➔  "1"  
String(1.2);    ➔  "1.2"
```

Value

Converte um valor texto para numérico.

Sintaxe:

```
<variável> = Value(<numero>);
```

Exemplos:

```
Value("1.00");  ➔  1  
Value("1.20");  ➔  1.2
```

Funções de Data e Hora

Date3

Retorna um valor do tipo *Date*.

Sintaxe:

```
<variável> = Date3(<ano>, <mês>, <dia>);
```

Exemplos:

```
Date3(2010, 12, 25); ➔ 25/12/2010
```

AddToDate

Adiciona ou subtrai dias, meses ou anos em uma data.

Para adicionar, os parâmetros devem ser positivos, enquanto que para subtrair, os parâmetros devem ser negativos.

Sintaxe:

```
<variável> = AddToDate(<data>, <ano>, <mês>, <dia>);
```

Exemplos:

```
Local date &A, &B;  
&A = Date3(2010, 12, 20);  
&B = AddToDate(&A, 0, 0, 7); ➔ adiciona 7 dias = 27/12/2010  
  
&A = Date3(2011, 1, 31);  
&B = AddToDate(&A, 0, 1, 0); ➔ adiciona 1 mês = 28/02/2011  
  
&A = Date3(2011, 1, 31);  
&B = AddToDate(&A, 0, -1, -15); ➔ subtrai 1 mês e 15 dias = 16/12/2010
```

DateTime6

Retorna um valor do tipo *DateTime*.

Sintaxe:

```
<variável> = DateTime6(<ano>, <mês>, <dia>, <hora>, <minuto>, <segundo>);
```

Exemplos:

```
DateTime6(2010, 12, 25, 23, 59, 59); ➔ 25/12/2010 23:59:59
```


AddToDateTime

Adiciona ou subtrai anos, meses, dias, horas, minutos ou segundos em um campo *datetime*.

Para adicionar, os parâmetros devem ser positivos, enquanto que para subtrair, os parâmetros devem ser negativos.

Sintaxe:

```
<variável> = AddToDateTime(<data>, <ano>, <mês>, <dia>, <hora>, <minuto>, <segundo>);
```

Exemplos:

```
Local datetime &A, &B;
&A = DateTime6(2010, 12, 20, 14, 30, 00);
&B = AddToDateTime(&A, 0, 0, 7, 0, 0, 0); ➔ adiciona 7 dias = 27/12/2010 14:30:00

&A = DateTime6(2010, 12, 20, 14, 30, 00);
&B = AddToDateTime(&A, 0, 0, 7, 8, 0, 0); ➔ ad. 7 dias e 8 horas = 27/12/2010 22:30:00

&A = DateTime6(2010, 12, 20, 14, 30, 00);
&B = AddToDateTime(&A, 0, 0, 0, 0, 0, 45); ➔ adiciona 45 segundos = 28/02/2011 14:30:45

&A = DateTime6(2010, 12, 20, 14, 30, 00);
&B=AddToDateTime(&A, 0, 0, 0, -1, -30, 0); ➔ sub 1 hora e 30 min = 16/12/2010 13:00:00
```

Exercícios

- 1) Abra o Application Designer e carregue o seu projeto
- 2) No evento *FieldChange* do BUTTON1, apague o código existente e adicione o código abaixo.

```
Local date &A, &B;
&A = Date3(2010, 12, 20);
&B = AddToDate(&A, 0, 0, 7);
WinMessage("AddToDate Exemplo 1: A = " | &A | " - B = " | &B);

&A = Date3(2011, 1, 31);
&B = AddToDate(&A, 0, 1, 0);
WinMessage("AddToDate Exemplo 2: A = " | &A | " - B = " | &B);

&A = Date3(2011, 1, 31);
&B = AddToDate(&A, 0, -1, -15);
WinMessage("AddToDate Exemplo 3: A = " | &A | " - B = " | &B);
```

- 3) No evento *FieldChange* do BUTTON1, apague o código existente e adicione o código abaixo.

```
Local datetime &A, &B;
&A = DateTime6(2010, 12, 20, 14, 30, 00);
&B = AddToDateTime(&A, 0, 0, 7, 0, 0, 0);
WinMessage("AddToDateTime Exemplo 1: A = " | &A | " - B = " | &B);

&A = DateTime6(2010, 12, 20, 14, 30, 00);
&B = AddToDateTime(&A, 0, 0, 7, 8, 0, 0);
WinMessage("AddToDateTime Exemplo 2: A = " | &A | " - B = " | &B);

&A = DateTime6(2010, 12, 20, 14, 30, 00);
&B = AddToDateTime(&A, 0, 0, 0, 0, 0, 45);
WinMessage("AddToDateTime Exemplo 3: A = " | &A | " - B = " | &B);

&A = DateTime6(2010, 12, 20, 14, 30, 00);
&B = AddToDateTime(&A, 0, 0, 0, -1, -30, 0);
WinMessage("AddToDateTime Exemplo 4: A = " | &A | " - B = " | &B);
```

- 4) No evento *FieldChange* do BUTTON1, apague o código existente e adicione o código abaixo.

```
Local integer &A;
&A = Find("SYS", "DIGISYSTEM");
WinMessage("Achou o texto SYS na posição " | &A);

Local integer &A;
&A = Find("I", "DIGISYSTEM", 3);
WinMessage("Achou o texto I na posição " | &A);
```

- 5) No evento *FieldChange* do BUTTON1, apague o código existente e adicione o código abaixo.

```
Local string &Texto, &Esquerda, &Direita;
Local integer &Tamanho;

&Texto = "Este é o Curso de PeopleCode"
WinMessage("O texto é: " | &Texto);

&Esquerda = Left(&Texto, 10);
WinMessage("Os 10 primeiros caracteres do texto são: " | &Esquerda);

&Direita = Right(&Texto, 10);
WinMessage("Os 10 ultimos caracteres do texto são: " | &Direita);

&Tamanho = Len(&Texto);
WinMessage("O tamanho do texto é: " | &Tamanho);
```

- 6) No evento *FieldChange* do BUTTON1, apague o código existente e adicione o código abaixo.

```
Local string &A;
&A = Rept("0", 15)
WinMessage("O texto 0 repetido 15 vezes é: " | &A);
```

Criação de Funções Personalizadas

Categorias de Funções

O PeopleCode consegue usar 3 tipos de funções:

- Internas PeopleCode
- Externas PeopleCode
- Externas Não-PeopleCode

Funções Internas PeopleCode

Funções internas são criadas dentro de um evento e utilizadas dentro deste mesmo evento.

Nas funções internas/externas PeopleCode os parâmetros são sempre por referência, o que significa que se o valor do parâmetro for alterado dentro da função, ele se manterá alterado quando a função terminar e retornar à linha seguinte que chamou a função.

Sintaxe:

```
Function <nome> (<parametros>) [Returns <tipo_dados>]  
    [peoplecodes]  
    [Return <variável>];  
End-Function;
```

Exemplos:

```
--- código dentro do evento FieldChange do campo MINHA_RECORD.MEU_CAMPO ---  
Function Soma(&Valor1 As number, &Valor2 As number) Returns number  
    Local number &A;  
    &A = &Valor1 + &Valor2;  
    Return &A;  
End-Function;  
  
Local number &A;  
&A = Soma(2, 3);  
WinMessage("O resultado é: " | &A);
```

Funções Externas PeopleCode

Funções externas são criadas dentro de um evento e utilizadas dentro de outro mesmo evento.

Nas funções internas/externas PeopleCode os parâmetros são sempre por referência, o que significa que se o valor do parâmetro for alterado dentro da função, ele se manterá alterado quando a função terminar e retornar à linha seguinte que chamou a função.

Importante:

Após criar a função, salve a record para que a função seja armazenada no banco de dados, pois caso contrário, ao declarar esta função em outro evento o PeopleCode não conseguirá validar o código no banco de dados e apresentará a mensagem que a função não foi encontrada.

Sintaxe:

```
Declare Function <nome> PeopleCode <record>.<campo> <evento>;
```

Exemplos:

```
--- código dentro do evento FieldFormula do campo MINHA_RECORD.MEU_CAMPO ---  
Function Soma(&Valor1 As number, &Valor2 As number) Returns number  
    Local number &A;  
    &A = &Valor1 + &Valor2;  
    Return &A;  
End-Function;  
  
--- código dentro do evento FieldChange do campo MINHA_RECORD.MEU_CAMPO ---
```

```

Declare Function Soma PeopleCode MINHA_RECORD.MEU_CAMPO FieldFormula;
Local number &A;
&A = Soma(2, 3);
WinMessage("O resultado é: " | &A);

```

Funções Externas Não PeopleCode

Funções externas não PeopleCode são criadas em outras linguagens de programação, como Java, C++, Visual Basic, etc.

A função externa deverá estar em arquivo formato DLL e deverá estar em uma pasta que faça parte do *PATH* do sistema operacional de todos os computadores configurados como Application Server.

Sintaxe:

```

Declare Function <nome_funcao> Library <lib_name>;

```

Exemplos:

```

Declare Function LogMsg Library "testdll" (string, string) Returns integer;
&Ok = LogMsg("C:\temp\test.log", "Este é um teste");

```

Exercícios

- 1) Abra o Application Designer e carregue o seu projeto
- 2) No evento *FieldFormula* do BUTTON1, apague o código existente (caso exista) e adicione o código abaixo.

```

Function Soma(&Valor1 As number, &Valor2 As number) Returns number
    Local number &A;
    &A = &Valor1 + &Valor2;
    Return &A;
End-Function;

```

```

Function Multiplicacao(&Valor1 As number, &Valor2 As number) Returns number
    Local number &A;
    &A = &Valor1 * &Valor2;
    Return &A;
End-Function;

```

- 1) No evento *FieldChange* do BUTTON1, apague o código existente (caso exista) e adicione o código abaixo.

```

Declare Function Soma PeopleCode EXERC_TESTE_WRK.BUTTON1 FieldFormula;
Declare Function Multiplicacao PeopleCode EXERC_TESTE_WRK.BUTTON1 FieldFormula;

```

```

Local number &A;

&A = Soma(9, 5);
WinMessage("O resultado da soma é: " | &A);

&A = Multiplicacao(9, 5);
WinMessage("O resultado da multiplicação é: " | &A);

```

- 2) Teste seu código

Trabalhando com Níveis em uma Página

Acessando o Nível 0

GetLevel0

Quando há níveis em uma página, nem sempre é possível acessar campos diretamente em outro nível, para resolver este problema surge a função `GetLevel0`, permitindo de forma estruturada a partir dela acessar os dados em qualquer nível.

Esta função retorna um *Rowset* contendo os dados do nível 0 do buffer do componente.

Sintaxe:

```
<variavel> = GetLevel0();
```

Exemplos:

```
Local Rowset &Nivel0;  
&Nivel0 = GetLevel0();  
&Nivel0(1).MINHA_RECORD.MEU_CAMPO.Value = "BLA BLA";
```

```
Local Rowset &Nivel0;  
&Nivel0 = GetLevel0();  
&Nivel0.GetRow(1).MINHA_RECORD.MEU_CAMPO.Value = "BLA BLA";
```

Acessando Níveis 1, 2 e 3

GetRowset

Como podemos ter mais de uma record por nível, esta função permite acessar uma record em específico em um determinado nível.

O exemplo abaixo é uma “receita de bolo”, você pode salvá-lo em local de fácil acesso para referência futura.

Sintaxe:

```
<variavel> = GetRowset(Scroll.<nome_da_record>);
```

Exemplos:

```
Local Rowset &Nivel0, &Nivel1, &Nivel2, &Nivel3;  
Local integer &I, &J, &K;  
  
/* pega o nível 0 do componente (RECORD PAI) */  
&Nivel0 = GetLevel0();  
  
/* pega o nível 1 do componente (filho do nível 0) */  
&Nivel1 = &Nivel0(1).GetRowset(Scroll.MINHA_RECORD_FILHA);  
  
/* percorre todas as linhas do nível 1 */  
For &I = 1 to &Nivel1.ActiveRowCount  
    &Nivel1(&I).MINHA_RECORD_FILHA.MEU_CAMPO.Value = "ABC ABC";  
  
/* pega cada nível 2 do componente (filhos do nível 1) */  
&Nivel2 = &Nivel1(&I).GetRowset(Scroll.MINHA_RECORD_NETA);  
  
/* percorre todas as linhas do nível 2 */  
For &J = 1 to &Nivel2.ActiveRowCount  
    &Nivel2(&J).MINHA_RECORD_NETA.MEU_CAMPO.Value = "DEF DEF";  
  
/* pega cada nível 3 do componente (filhos do nível 2) */  
&Nivel3 = &Nivel2(&J).GetRowset(Scroll.MINHA_RECORD_BISNETA);  
  
/* percorre todas as linhas do nível 3 */  
For &K = 1 to &Nivel3.ActiveRowCount  
    &Nivel3(&K).MINHA_RECORD_BISNETA.MEU_CAMPO.Value = "GHI GHI";  
End-For;
```

```
End-For;  
End-For;
```

Classe RowSet

A classe *Rowset* é usada quando precisamos manipular uma estrutura de dados, seja ela com dados vindos do banco de dados ou com dados gerados no pela aplicação somente em memória RAM.

CreateRowset

Muito útil quando queremos criar uma rowset vazia em memória RAM que não está amarrada a nenhum dado do banco de dados.

Exemplos:

```
Local Rowset &RS;  
&RS = CreateRowset(Record.MINHA_RECORD);
```

GetRowset

Esta função busca uma rowset já existente no buffer do componente permitindo que se trabalhe nos dados já existentes.

Exemplos:

```
Local Rowset &RS;  
&RS = GetRowset(Record.MINHA_RECORD);
```

ActiveRowCount

Esta propriedade retorna a quantidade de linhas ativas dentro de uma rowset, considera-se linhas ativas, aquelas que não foram marcadas para exclusão.

Exemplos:

```
Local Rowset &RS;  
&RS = GetRowset(Record.MINHA_RECORD);  
&Linhas = &RS.ActiveRowCount;
```

Fill

Esta função preenche uma rowset com dados vindos do banco de dados, e retorna o numero de linhas retornadas.

Exemplos:

```
Local Rowset &RS;  
&RS = CreateRowset(Record.MINHA_RECORD);  
/* retorna todas as linhas da record */  
&RS.Fill();  
  
Local Rowset &RS;  
Local number &Linhas;  
&RS = CreateRowset(Record.MINHA_RECORD);  
/* preenche o rowset com as linhas filtradas e retorna o numero de linhas encontradas */  
&Linhas = &RS.Fill("Where CAMPO1 > 0");
```

Flush

Esta função limpa uma rowset apagando todos os dados carregados, como uma rowset nunca pode ter zero linhas, ao apagar todos os dados, o Flush criará uma linha com dados em branco.

Exemplos:

```
Local Rowset &RS;  
&RS = CreateRowset(Record.MINHA_RECORD);  
&Linhas = &RS.Fill("Where CAMPO1 > 0");  
[<peoplecodes>]  
&RS.Flush();
```

```
&Linhas = &RS.Fill("Where CAMPO1 <= 0");
```

GetRow

Esta função retorna uma row (linha) de um rowset. Este é o método padrão da classe Rowset.

Exemplos:

```
Local Row &Row;  
&Row = GetRowset(Record.MINHA_RECORD) (5);  
  
Local Row &Row;  
&Row = GetRowset(Record.MINHA_RECORD).GetRow(5);
```

HideAllRows

Esta função esconde todas as linhas de uma rowset.

Exemplos:

```
Local Rowset &RS1, &RS2;  
&RS1 = GetRowset();  
&RS2 = GetRowset(SCROLL.EMPL_CHKLIST_ITM);  
For &I = 1 to &RS1.ActiveRowCount  
    If &RS1.GetRow(&I).MINHA_RECORD.MEU_CAMPO.Value = "N" Then  
        &RS2.HideAllRows();  
    End-If;  
End-For;
```

RowCount

Esta propriedade retorna a quantidade total de linhas dentro de uma rowset, não importa se a linha está marcada para exclusão ou não.

Exemplos:

```
Local Rowset &RS;  
&RS = GetRowset(Record.MINHA_RECORD);  
&Linhas = &RS.RowCount;
```

Select

Esta função preenche uma rowset com dados vindos do banco de dados, e retorna o numero de linhas retornadas.

Funciona de forma semelhante ao Fill, porém somente pode ser usada em rowsets instanciadas a partir de um GetRowset, enquanto o Fill deve ser usado em rowsets instanciadas através de CreateRowset.

Exemplos:

```
Local Rowset &RS;  
Local number &Linhas;  
&RS = GetRowset(Record.MINHA_RECORD);  
/* preenche o rowset com as linhas filtradas e retorna o numero de linhas encontradas */  
&Linhas = &RS.Select(Record.MINHA_RECORD, "Where CAMPO1 > 0");
```

ShowAllRows

Esta função exibe todas as linhas de uma rowset.

Exemplos:

```
Local Rowset &RS1, &RS2;  
&RS1 = GetRowset();  
&RS2 = GetRowset(SCROLL.EMPL_CHKLIST_ITM);  
For &I = 1 to &RS1.ActiveRowCount  
    If &RS1.GetRow(&I).MINHA_RECORD.MEU_CAMPO.Value = "Y" Then  
        &RS2.ShowAllRows();  
    End-If;  
End-For;
```

Sort

Esta função permite ordenação de dados em uma scroll ou grid baseada em critérios personalizados.

Exemplos:

```
Local Rowset &RS1;  
&RS1 = GetLevel0() (1).GetRowset(Scroll.MINHA_RECORD);  
  
/* ordena pelo campo 3 (crescente) e desempata pelo campo 6 (decrescente) */  
&RS1.Sort(MINHA_RECORD.CAMPO3, "A", MINHA_RECORD.CAMPO6, "D");
```

Classe Row

A classe *Row* é usada quando precisamos manipular uma única linha de dados dentro de uma rowset.

GetRecord

Pega uma record da linha.

Exemplos:

```
&Rec = &Row.GetRecord(Record.MINHA_RECORD);
```

GetRowset

Pega uma rowset da linha.

Exemplos:

```
&RS = &Row.GetRowset(Scroll.MINHA_RECORD);
```

IsChanged

Indica se uma linha foi alterada ou não. Ao inserir uma linha nova a propriedade *IsChanged* será sempre *TRUE*.

Exemplos:

```
Local Rowset &RS;  
&RS = GetRowset(Record.MINHA_RECORD);  
If &RS(1).IsChanged Then  
    /* dados não foram salvos */  
End-If;
```

IsDeleted

Indica se uma linha foi apagada ou não.

Exemplos:

```
Local Rowset &RS;  
&RS = GetRowset(Record.MINHA_RECORD);  
If &RS(1).IsDeleted Then  
    /* esta linha foi apagada */  
End-If;
```

IsNew

Indica se uma linha foi inserida desde a página ter sido exibida ou não.

Exemplos:

```
Local Rowset &RS;  
&RS = GetRowset(Record.MINHA_RECORD);  
If &RS(1).IsNew Then  
    /* esta linha é nova */  
End-If;
```


RecordCount

Conta a quantidade de records em uma linha de dados.

Exemplos:

```
For &I = 1 To &Row.RecordCount
    &Rec = &Row.GetRecord(&I);
    /* peoplecodes */
End-For;
```

RowNumber

Retorna o numero da linha dentro do rowset.

Exemplos:

```
&Linha = GetRowset(Scroll.MINHA_RECORD).GetCurrEffRow().RowNumber;
```

Selected

Retorna TRUE se a linha estiver selecionada, somente terá efeito em grids configuradas para seleção de linha ou múltipla seleção de linhas.

Exemplos:

```
For &I = 1 To &RS.ActiveRowCount
    If &RS.GetRow(&I).Selected Then
        /* esta linha está selecionada */
    End-If;
End-For;

/* como o método GetRow é o método default de um Rowset, o código abaixo é o mesmo */
For &I = 1 To &RS.ActiveRowCount
    If &RS(&I).Selected Then
        /* esta linha está selecionada */
    End-If;
End-For;
```

Visible

Permite identificar se uma linha está visível ou escondida.

Se definir valor FALSE para esta propriedade a linha será escondida e movida para o final do rowset, isso significa que o número da linha mudará ao ser escondida.

Ao definir valor TRUE para esta propriedade em uma linha escondida, a linha ficará visível, porém a linha não será movida para seu local original, ao invés disso, ela será movida para a 1ª posição após a última linha visível no momento.

Para exemplificar:

Suponha uma grid com 6 linhas visíveis, ao esconder a linha 3, as linhas 4, 5 e 6 serão movidas 1 posição para frente e a linha 3 será movida para a posição 6. A posição dos dados na grid ficará assim: **1, 2, 4, 5, 6, 3**.

Ao esconder novamente a linha 3, todo o processo repete, ficando os dados da grid na posição **1, 2, 5, 6, 3, 4**.

Neste momento temos 4 linhas visíveis (valores 1, 2, 5, 6) e 2 linhas escondidas (valores 3 e 4), ao tornar visível a última linha (valor 4), ela será movida para o final das linhas visíveis, ficando assim os dados na grid: **1, 2, 5, 6, 4, 3**.

Exemplos:

```
For &I = 1 to &RS.ActiveRowNumber
    If &RS(&I).MEU_CAMPO.Value = "N" Then
        &RS(&I).Visible = False;
    End-If;
End-For;
```

Classe Field

SearchClear

Limpa o conteúdo do campo caso ele esteja definido como *Search Key* na record. Muito usado nos eventos *SearchInit*.

Exemplos:

```
MINHA_RECORD.MEU_CAMPO.SearchClear();
```

SearchDefault

Quando definido como *TRUE*, preenche o valor default do campo na tela de pesquisa. Muito usado nos eventos *SearchInit*.

Exemplos:

```
MINHA_RECORD.MEU_CAMPO.SearchDefault = True;
```

SearchEdit

Quando definido como *TRUE*, habilita a validação do campo fazendo com que somente valores válidos possam ser digitados. Muito usado nos eventos *SearchInit*.

Exemplos:

```
MINHA_RECORD.MEU_CAMPO.SearchEdit = True;
```

SetDefault

Define o conteúdo do campo como Null ou com o valor default, dependendo de como o campo estiver configurado na Record.

O PeopleSoft possui a função *SetDefault* para esta ação, porém esta função está em desuso e não deve mais ser usada.

Exemplos:

```
MINHA_RECORD.MEU_CAMPO.SetDefault();
```

```
/* não use a função abaixo por estar em desuso */  
SetDefault(MINHA_RECORD.MEU_CAMPO);
```

SetCursorPos

Permite definir o foco da página em um campo específico.

Exemplos:

```
If None(MINHA_RECORD.MEU_CAMPO.Value) Then  
  MINHA_RECORD.MEU_CAMPO.SetCursorPos(%Page);  
  Error "O campo MEU_CAMPO não foi preenchido";  
End-If;
```

DisplayOnly

Transforma um campo em somente leitura ou editável, quando configurado em somente leitura o campo fica acessível para seleção do conteúdo pelo mouse.

Exemplos:

```
If None(MINHA_RECORD.MEU_CAMPO1.Value) Then  
  MINHA_RECORD.MEU_CAMPO2.DisplayOnly = True;  
Else  
  MINHA_RECORD.MEU_CAMPO2.DisplayOnly = False;  
End-If;
```

Enabled

Transforma um campo em habilitado ou desabilitado, quando configurado em desabilitado não é possível selecionar o texto com o mouse.

O PeopleSoft possui as funções Gray e UnGray para habilitar/desabilitar campos na tela, porém estas funções estão em desuso e não devem mais ser usadas.

Exemplos:

```
If None (MINHA_RECORD.MEU_CAMPO1.Value) Then
    MINHA_RECORD.MEU_CAMPO2.Enabled = True;
Else
    MINHA_RECORD.MEU_CAMPO2.Enabled = False;
End-If;

/* não use as funções Gray e UnGray abaixo por estarem em desuso */
If None (MINHA_RECORD.MEU_CAMPO1.Value) Then
    UnGray (MINHA_RECORD.MEU_CAMPO2);
Else
    Gray (MINHA_RECORD.MEU_CAMPO2);
End-If;
```

FormattedValue

Retorna o conteúdo do campo exatamente como exibido na tela, com todas os caracteres de formatação.

Exemplos:

```
WinMessage (MINHA_RECORD.MEU_CAMPO1.FormattedValue);
```

IsChanged

Retorna *TRUE* se o campo foi alterado desde que foi carregado pelo buffer do componente.

Exemplos:

```
If Not MINHA_RECORD.MEU_CAMPO.IsChanged Then
    Error "Você precisa alterar o MEU_CAMPO antes de continuar";
End-If;
```

IsRequired

Retorna *TRUE* se o campo foi definido com obrigatório na record.

Exemplos:

```
If None (MINHA_RECORD.MEU_CAMPO.Value) And MINHA_RECORD.MEU_CAMPO.IsRequired Then
    Error "Preencha o campo antes de continuar";
End-If;
```

Label

Retorna o *label* de um campo.

Exemplos:

```
If None (MINHA_RECORD.MEU_CAMPO.Value) And MINHA_RECORD.MEU_CAMPO.IsRequired Then
    Error "Preencha o campo " | MINHA_RECORD.MEU_CAMPO.Label | " antes de continuar";
End-If;
```

OriginalValue

Retorna o valor original de um campo no banco de dados.

Como records do tipo Derived/Work não existem no banco de dados, esta propriedade não funciona neste tipo de record.

Também é possível buscar o valor original de um campo através da função *PriorValue*, porém esta função está em desuso, ela existe apenas por motivos de compatibilidade com versões anteriores e seu uso não é aconselhado.

Exemplos:

```
&Original = MINHA_RECORD.MEU_CAMPO.OriginalValue;  
&Atual = MINHA_RECORD.MEU_CAMPO.Value;  
WinMessage("O valor será alterado de " | &Original | " para " | &Atual);
```

Value

Retorna o valor atual de um campo.

Exemplos:

```
&Original = MINHA_RECORD.MEU_CAMPO.OriginalValue;  
&Atual = MINHA_RECORD.MEU_CAMPO.Value;  
WinMessage("O valor será alterado de " | &Original | " para " | &Atual);
```

Visible

Esta propriedade exibe ou esconde um campo na tela.

O PeopleSoft possui as funções Hide e UnHide para exibir/esconder campos na tela, porém estas funções estão em desuso e não devem mais ser usadas.

Exemplos:

```
If None(MINHA_RECORD.MEU_CAMPO1.Value) Then  
    MINHA_RECORD.MEU_CAMPO2.Visible = True;  
Else  
    MINHA_RECORD.MEU_CAMPO2.Visible = False;  
End-If;  
  
/* não use as funções Hide e UnHide abaixo por estarem em desuso */  
If None(MINHA_RECORD.MEU_CAMPO1.Value) Then  
    UnHide(MINHA_RECORD.MEU_CAMPO2);  
Else  
    Hide(MINHA_RECORD.MEU_CAMPO2);  
End-If;
```

Exercícios

- 1) Abra o Application Designer e carregue o seu projeto
- 2) Na record EXERC_TESTE_WRK, adicione o campo *AMOUNT*
- 3) Na página EXERC_LIVALU_VW, coloque o campo *AMOUNT*, da record EXERC_TESTE_WRK, abaixo da grid de modo que fique alinhado com o campo "Valor" da grid.
- 4) Configure este campo como Display Only.
- 5) Adicione código neste campo para que exiba a soma do campo "Valor" da grid.
- 6) Na página EXERC_MAIS_VW, coloque o campo *AMOUNT*, da record EXERC_TESTE_WRK, abaixo da grid de modo que fique alinhado com o campo "Quantidade" da grid.
- 7) Configure este campo como Display Only.
- 8) Adicione código neste campo para que exiba a soma do campo "Quantidade" da grid sem interferir na funcionalidade do campo de exibir a soma do campo "Valor" da outra tela.

Classes

Classes em PeopleSoft são criadas através do Application Class usando programação orientada a objetos, e servem para:

- Armazenamento de código reutilizável
- Estender classes já existentes para criar novas
- Encapsular estruturas ou regras de negócio complexas
- Organizar logicamente diferentes códigos

Application Package

Uma classe não pode existir isoladamente, por isso, deve-se criar um Application Package, que é um agrupador de classes, lá podemos ver todas as classes e métodos de forma organizada.

Para criar um Application Package deve-se usar o editor de Application Packages que fica dentro do Application Designer.

No exemplo abaixo, *PSXP_XMLGEN* é o nome do Application Package, enquanto os itens *Collection*, *DSEException*, *OutputField*, *QueryDS* e *RowSetDS* são Classes.



Criando uma Classe

Primeiro crie um *Application Package*, no menu *File*, acesse *New* e depois *Application Package* na janela que surgir, depois no menu *Insert*, clique em *Application Class* e dê um nome a sua classe, lembre-se de salvar o Application Package para que seja possível dar um nome a ele.

Para acessar o PeopleCode da classe, use o botão direito sobre o nome da classe, uma classe possui somente um evento, o “*On Execute*”, toda a codificação da classe ocorrerá nele.

Sintaxe:

```
Class <nome_classe> [Extends | Implements <outra_classe>]
    <declaração_de_metodos>
    <declaração_de_propriedades>
[Protected
    <declaração_de_metodos>
    <declaração_de_instancias>]
[Private
    <declaração_de_metodos>
    <declaração_de_instancias>]
End-Class;
```

Exemplos:

```
/* use o mesmo nome que você escolheu ao criar a classe */
Class MinhaClasse
    /* método construtor da classe */
    Method MinhaClasse();

    /* métodos públicos da classe */
    Method MeuMetodo1() returns integer;
    Method MeuMetodo2(&Codigo As String, &Idade As Integer) Returns Boolean;

    /* propriedades públicas da classe */
    Property string Nome;
    Property number Valor;
    Property integer Idade Get Set;
```

```

Protected
    /* método protegido da classe */
    Method MeuMetodo3(&Valor as number);

    /* propriedade protegida da classe */
    Instance number &Codigo;

Private
    /* método privado da classe */
    Method MeuMetodo4();

    /* propriedade privada da classe */
    Instance string &Temp;
    Instance integer &mIdade;
End Class;

/* le propriedade */
Get Idade
    Return &mIdade;
End-get;

/* grava propriedade */
Set Idade
    &mIdade = &NewValue;
End-set;

/* código do método construtor */
Method MinhaClasse
    <peoplecodes>
End-method;

/* código do método MeuMetodo1 */
Method MeuMetodo1
    <peoplecodes>
End-method;

/* código do método MeuMetodo2 */
Method MeuMetodo2
    <peoplecodes>
End-method;

/* código do método MeuMetodo3 */
Method MeuMetodo3
    <peoplecodes>
End-method;

/* código do método MeuMetodo4 */
Method MeuMetodo4
    <peoplecodes>
End-method;

```

Protected

A instrução Protected na declaração da classe indica que métodos e instancias declaradas a partir dele serão privadas da classe quando acessadas externamente, porém são visíveis a classes herdadas.

Private

A instrução Private na declaração da classe indica que métodos e instancias declaradas a partir dele serão sempre privadas da classe, ou seja, não serão acessíveis por classes herdadas nem quando acessadas externamente.

%This

O objeto %This permite dentro da classe, usar recursos da própria classe.

Exemplos:

```

Method MeuMetodo
    Local string &Nome;

    /* coloca na variável local Nome, o conteúdo da propriedade Nome */
    &Nome = %This.Nome;

```

```

        /* chama outro método da classe */
        %This.MeuOutroMetodo();
End-If;

```

%Super

O objeto %Super permite acessar métodos e propriedades da classe pai (quando a classe atual “extends” outra classe.

Exemplos:

```

class Grafico
    method Grafico(&X1 As integer, &Y1 As integer);
protected
    property integer X;
    property integer Y;
end-class;

method Grafico
    %This.X = &X1;
    %This.Y = &Y1;
end-method;

=====

Import MEU_APPLICATION_PACKAGE:Grafico;

class Grafico3D extends Grafico
    method Grafico3D(&X1 As integer, &Y1 As integer, &Z1 As integer);
protected
    property integer Z;
end-class;

method Grafico3D
    %Super = Create MEU_APPLICATION_PACKAGE:Grafico(&X1, &Y1);
    %This.Z = &Z1;
    WinMessage("X = " | %Super.X | " Y = " | %Super.Y | " Z = " | %This.Z);
end-method;

```

Usando uma Classe

Para usar uma classe é necessário 3 etapas:

1. Importá-la no seu evento de PeopleCode
2. Criar uma variável do tipo da classe
3. Instanciar essa variável

Exemplos:

```

/* importa todas as classes do application package para uso no peoplecode */
Import MEU_APPLICATION_PACKAGE:*;

/* declara uma variável do tipo da classe */
Local MEU_APPLICATION_PACKAGE:MinhaClasse &Var;

/* instancia a classe */
&Var = Create MEU_APPLICATION_PACKAGE:MinhaClasse();

=====

/* importa somente a classe MinhaClasse para uso no peoplecode */
Import MEU_APPLICATION_PACKAGE:MinhaClasse;

/* declara uma variável do tipo da classe */
Local MEU_APPLICATION_PACKAGE:MinhaClasse &Var;

/* instancia a classe */
&Var = Create MEU_APPLICATION_PACKAGE:MinhaClasse();

```

Acessando Métodos e Propriedades de uma Classe

Uma vez que a classe foi importa, declarada e instanciada, você pode acessar somente os métodos e propriedades públicas da classe.

Os métodos e propriedades *Protected* e *Private* não ficam disponíveis para uso.

Exemplos:

```
/* importa somente a classe MinhaClasse para uso no peoplecode */
Import MEU_APPLICATION_PACKAGE:MinhaClasse;

/* declara uma variável do tipo da classe */
Local MEU_APPLICATION_PACKAGE:MinhaClasse &Var;

/* instancia a classe */
&Var = Create MEU_APPLICATION_PACKAGE:MinhaClasse();

/* chama um método da classe */
&A = &Var.MeuMetodo1();

/* acessa uma propriedade da classe */
&Var.Valor = 123;
```

Exercícios

- 1) Abra o Application Designer e carregue o seu projeto
- 2) No evento *FieldChange* do BUTTON1, apague o código existente
- 3) Crie um Application Package chamado *EXEMPLO*
- 4) Crie uma Classe chamada *Matematica*

```
Class Matematica
/* construtor da classe */
Method Matematica();
End-Class;
```

- 5) Adicione as definições dos métodos Soma e Multiplicacao como segue:

```
Method Soma(&Numero1 as number, &Numero2 as number) returns number;
Method Multiplicacao(&Valor1 as number, &Valor2 as number) as number;
```

- 6) Adicione código para os métodos Soma e Multiplicacao como segue:

```
Method Soma
Local number &Total;
&Total = &Numero1 + &Numero2;
Return &Total;
End Method;
```

```
Method Multiplicacao
Local number &Total;
&Total = &Valor1 * &Valor2;
Return &Total;
End Method;
```

- 7) Dentro desse application package, crie uma nova Classe chamada *Fisica*

```
Class Fisica
/* construtor da classe */
Method Fisica();
End-Class;
```

- 8) Adicione as definições dos métodos como segue:

```
Method AreaRetangulo() returns number;
Method PotenciaEletrica();
```

- 9) Adicione as propriedades como segue:

```
Property number Largura;
Property number Altura;
Property number Potencia;
Property number Corrente;
Property number Voltagem;
```

- 10) Adicione código para o método construtor:

```
Method Fisica
%this.Voltagem = 110;
End Method;
```

- 11) Adicione código para os métodos como segue:

```
Method AreaRetangulo
Local number &Area;
&Area = %this.Largura * %this.Altura;
Return &Area;
End Method;
```

```
Method PotenciaEletrica
%this.Potencia = %this.Corrente * %this.Voltagem;
```


End Method;

12) No evento *FieldChange* do BUTTON1 apague o código existente e adicione o código abaixo

```
Import EXEMPLO:*;  
  
Local EXEMPLO:Matematica &ClassMatematica;  
Local EXEMPLO:Fisica &ClassFisica;  
Local number &Resultado;  
  
&ClassMatematica = Create EXEMPLO:Matematica();  
&ClassFisica = Create EXEMPLO:Fisica();  
  
&Resultado = &ClassMatematica.Soma(2, 3);  
WinMessage("A soma de 2 e 3 é: " | &Resultado);  
  
&Resultado = &ClassMatematica.Multiplicacao(2, 3);  
WinMessage("A multiplicação de 2 e 3 é: " | &Resultado);  
  
&ClassFisica.Largura = 10;  
&ClassFisica.Altura = 10;  
&Resultado = &ClassFisica.AreaRetangulo();  
WinMessage("A área do retângulo é: " | &Resultado);  
  
&ClassFisica.Corrente = 15;  
&ClassFisica.PotenciaEletrica();  
&Resultado = &ClassFisica.Potencia;  
WinMessage("Potência em " | &ClassFisica.Voltagem | "v é: " | &Resultado);  
  
&ClassFisica.Voltagem = 220;  
&ClassFisica.PotenciaEletrica();  
&Resultado = &ClassFisica.Potencia;  
WinMessage("Potência em " | &ClassFisica.Voltagem | "v é: " | &Resultado);
```

13) Teste seu código

Numeração Automática

Muitas vezes precisamos ativar a numeração automática de um campo em uma tabela, como cada banco de dados possui sua própria sintaxe para criação de campos autonumeração, não podemos usar a estrutura existe do banco de dados para este fim, pois, como sabemos o PeopleSoft é multi-plataforma e um mesmo código deve rodar em todos os bancos, então o PeopleCode é quem assume este papel de autonumeração.

Criando Tabela de Setup

Para conseguir controlar qual é foi ultimo numero usado, ou mais recente, precisamos criar uma tabela de setup e armazenar o campo de controle lá.

Esta tabela de setup deverá ser do tipo *SQL Table* e normalmente conterá apenas uma linha. Somente quando possuir chave(s) poderá conter mais de uma linha, e esta(s) chave(s) deverá(ão) ser usada(s) na busca do último número usado.

É interessante posteriormente criar uma página para esta tela de setup e verificar com o analista, em qual caminho da interface web colocar esta página.

Normalmente os campos de autonumeração são do tipo Character, mesmo que armazenem somente números, pois caso, haja necessidade de ampliação da numeração, letras poderão ser usadas e o campo não precisará ter seu tamanho modificado, uma vez que mudar o tamanho de um campo pode ser mais complexo do que inicialmente previsto, pelo fato de existir em muitas outras tabelas, ou de existirem possíveis regras de negócio aplicadas nesse campo.

Também normalmente, o campo autonumeração possui o valor default *NEW* ou *NEXT*, para que o usuário identifique que ali é um campo automático.

Na página de adicionar nova linha, assim como na página do aplicativo, este campo deverá estar com a opção *Display Only* marcada, pois, campos autonumeração não podem ser editáveis pelo usuário.

Codificando a Autonumeração

Para codificar a autonumeração, é recomendado usar o evento *SavePreChange* do campo autonumerável.

GetNextNumberWithGapsCommit

Esta função retorna o número da sequência acrescido do incremento, atualizando a tabela de setup automaticamente para o novo número.

Esta função pode ser colocada em qualquer evento, porém o evento mais indicado é o *SavePreChange*.

Esta função usa uma segunda conexão ao banco de dados para melhorar o desempenho, dessa forma um único aplicativo consome mais de uma conexão ao servidor.

Sintaxe:

```
GetNextNumberWithGapsCommit(<Record.field>, <maximo>, <incremento>[, <where>]);
```

Exemplos:

```
Local number &Proximo;

/* verifica se deve auto numerar o campo */
If MINHA_RECORD.MEU_CAMPO_AUTONUMERACAO.Value = "NEXT" Then

    /* pega o proximo numero livre */
    &Proximo = GetNextNumberWithGapsCommit(RECORD_SETUP.CAMPO, 999999999, 1);

    /* verifica resultado da geração do proximo numero */
    Evaluate &Proximo
    When %GetNextNumber_SQLFailure
        Error "Não foi possível gerar o próximo número.";
    When %GetNextNumber_TooBig
        Error "Número atingiu o valor máximo: 9.999.999.999";
    When %GetNextNumber_NotFound
        try
            /* cria o setup, pois tabela está vazia */
            Local Record &Record;
            &Record = CreateRecord(Record.RECORD_SETUP);
            &Record.CAMPO.Value = 1;
            &Record.Insert();
            CommitWork();

            /* exibe o numero gerado */
            MINHA_RECORD.MEU_CAMPO_AUTONUMERACAO.Value = 1;
        catch Exception &E
            Error "Não foi possível criar setup de auto-numeração";
        end-try;
    When-Other
        /* exibe o numero gerado */
        MINHA_RECORD.MEU_CAMPO_AUTONUMERACAO.Value = &Proximo;
    End-Evaluate;
End-If;
```

Migração de Projeto

Após terminar o desenvolvimento do projeto na base de desenvolvimento, será necessário migrar este projeto para uma base de teste de desenvolvimento, onde o analista irá testar se o aplicativo cumpre todas as solicitações, antes de migrar para a base de teste do cliente.

Revisando o Projeto Antes de Migrar

Para fazer esta migração alguns cuidados precisam ser levados em conta:

- Ter certeza que todos os objetos alterados fazem parte do projeto
- Ter certeza que o projeto não contenha objetos sem alteração (algo que foi aberto somente para consulta)
- Ter o cuidado necessário em objetos que foram excluídos no projeto para que sejam devidamente excluídos na base de destino
- Ter certeza que os objetos do projeto não estão sendo usados por outro desenvolvedor em um desenvolvimento que ainda não foi concluído, pois causará erros em outros locais do sistema, que neste momento não serão detectados

É muito importante revisar as abas *Development* e *Upgrade* para identificar os objetos que fazem parte do projeto. A aba *Upgrade* possui todas as definições de objetos que serão migradas, ela pode conter mais objetos que a aba desenvolvimento.

Aba Development

- Contém menos definições que a aba *Upgrade*
- Possui somente objetos que podem ser editados pelo Application Designer

Aba Upgrade

- Possui todas as definições usadas pelo projeto (editáveis pelo Designer ou não)
- Possui as definições excluídas do banco para que sejam excluídas no destino
- Deve-se prestar mais atenção nesta aba antes de migrar

Não Esqueça de Objetos de Estrutura do Portal

Após concluir o projeto, muitas vezes criamos pastas e referências de conteúdo no portal PeopleTools, esses objetos também devem ser migrados para que a mesma estrutura de pastas e referências de conteúdo seja criada na base de destino.

Para isso, adicione todos os objetos de pastas, estruturas e conteúdo no seu projeto antes de iniciar a migração.

Não Migre Segurança

Não devemos migrar segurança, pois, esse objeto de segurança poderá parar em produção, e se isso ocorrer, pode abrir brechas de segurança para acesso indevido na produção, ou até mesmo, pode causar a revogação de permissões de usuários que já possuíam permissão.

Revise seu projeto para ter certeza que ele não possui nenhum objeto de segurança na aba *Upgrade*, caso tenha, remova este objeto do seu projeto.

Comparando Bases Origem e Destino

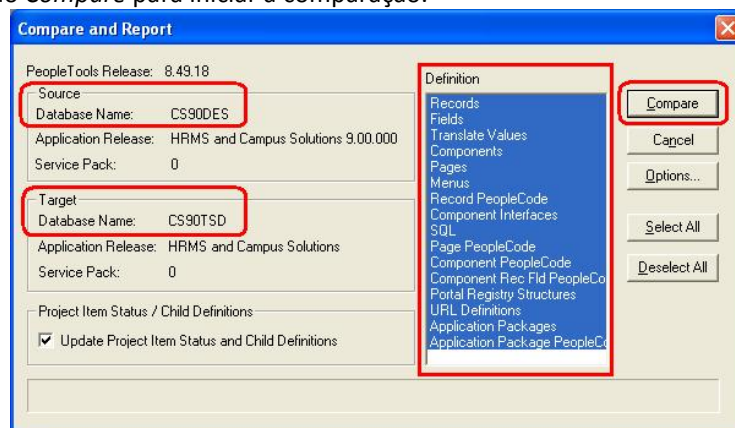
A migração é atualização da base destino a partir de objetos vindos de uma base origem, sendo assim, antes de iniciar a migração é interessante (mas não é obrigatório) comparar o conteúdo do projeto entre as bases origem e destino para que seja

A vantagem de executar a comparação é ter um mapa das diferenças que o nosso projeto apresenta nas bases origem e destino.

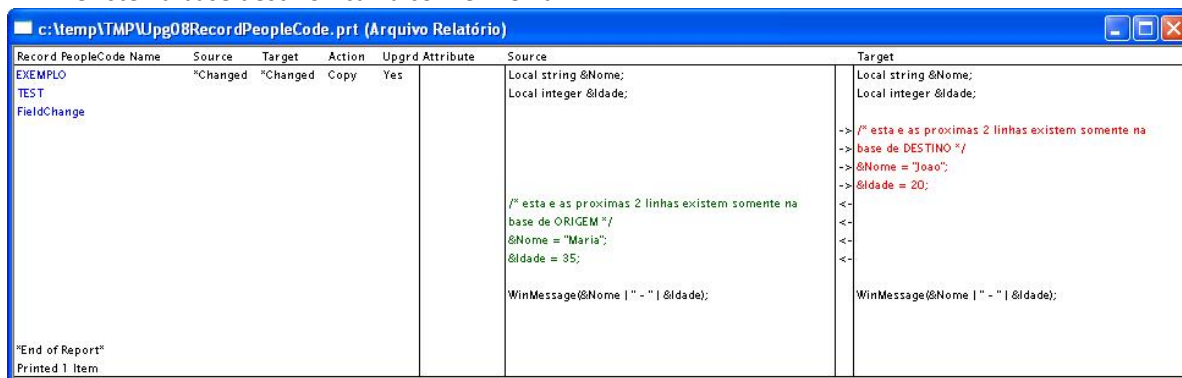
Somente podemos comparar e migrar nosso projeto com outra base que tenhamos acesso, isso evita que um desenvolvedor por descuido envie algo para base de produção ou outra base crítica. O time de desenvolvimento, a rigor, somente tem acesso as bases de desenvolvimento (acesso total) e a base de teste de desenvolvimento (acesso restrito para migrações).

Para executar a comparação de bases vá ao menu *Tools*, escolha a opção *Compare And Report*, e depois a opção *To Database*. Uma tela de login surgirá, é nela que escolheremos qual base de destino será usada na comparação, entre com suas credenciais e faça o login.

Identifique se a base origem e a base destino são as bases corretas, escolha os itens de definição que deverão ser comparados (normalmente todos devem ser selecionados) e por fim clique no botão *Compare* para iniciar a comparação.



Ao terminar, o Application Designer mostrará várias telas, uma para cada tipo de definição, contendo as diferenças entre as bases origem e destino, nas telas de PeopleCode, o código que somente existe na origem fica na cor verde, enquanto que o código que somente existe na base destino fica na cor vermelha.

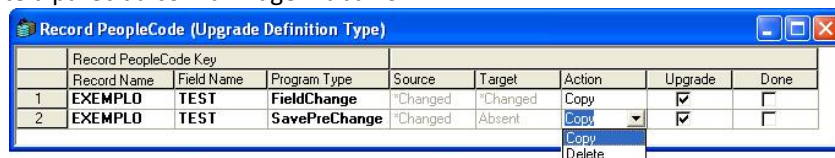


Fazer a migração desse PeopleCode significa apagar o código em vermelho copiando o que está em verde para o destino.

Preparando a Migração

Após a comparação terminar, cada item da aba Upgrade estará configurado nas opções padrão, o que não significa que estão necessariamente corretas.

Assim que a comparação entre bases terminar, abra cada item da aba Upgrade, você verá uma tela parecida com a imagem abaixo.



Os campos *Source* e *Target*, indicam como a definição se encontra nas 2 bases:

- **Same/Same**: Indica que a definição está igual nas 2 bases
- **Changed/Changed**: Indica que a definição está diferente nas 2 bases
- **Changed/Absent**: Indica que a definição existe na origem, mas não no destino
- **Absent/Changed**: Indica que a definição existe no destino, mas não na origem
- **Changed/Unchanged**: Indica que a definição na base origem foi alterada, enquanto que na base destino nunca foi alterada (standard PeopleSoft)
- **Unchanged/Changed**: Indica que a definição na base destino foi alterada, enquanto que na base origem nunca foi alterada (standard PeopleSoft)

O campo *Action* é quem define o que deverá ocorrer com a definição durante a migração, se ela será copiada na base de destino, ou excluída na base de destino.

O campo *Upgrade* é quem indica se a definição será copiada/excluída na base destino ou não.

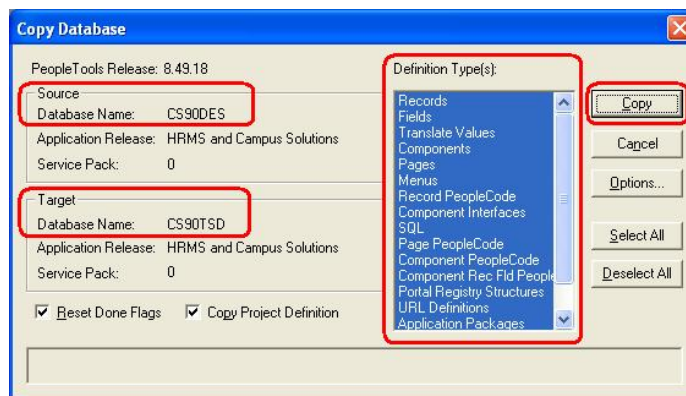
O campo *Done* indica se a operação de migração ocorreu ou não.

Migrando o Projeto

Para Outro Banco de Dados

Para executar a cópia entre bancos de dados (migração) de um projeto vá ao menu *Tools*, escolha a opção *Copy Project*, e depois a opção *To Database*. Uma tela de login surgirá, é nela que escolheremos qual base de destino será usada na cópia, entre com suas credenciais e faça o login.

Identifique se a base origem e a base destino são as bases corretas, escolha os itens de definição que deverão ser copiados (normalmente todos devem ser selecionados) e por fim clique no botão *Copy* para iniciar a migração.



Após a cópia terminar, verifique o campo *Done* de cada item da aba *Upgrade* para identificar se todos os itens marcados para migração foram efetivamente migrados.

Record PeopleCode (Upgrade Definition Type)								
Record PeopleCode Key								
	Record Name	Field Name	Program Type	Source	Target	Action	Upgrade	Done
1	EXEMPLO	TEST	FieldChange	"Changed"	"Changed"	Copy	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
2	EXEMPLO	TEST	SavePreChange	"Changed"	Absent	Copy	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Também é possível refazer a comparação entre bases para ter certeza que todos os itens a serem migrados, foram efetivamente migrados. Assim ao terminar a nova comparação, abra novamente cada tipo de definição e confirme se todas foram migradas conforme planejado.

Para Arquivo

Processo semelhante à cópia para outro banco de dados, porém a cópia para arquivo é um bom método de fazer backup de um projeto, pois caso haja alguma modificação na base é possível restaurar este backup.

Antes de executar a cópia para arquivo é importante:

- Abrir cada item da aba *Upgrade*
- Marcar a opção *Upgrade* de cada linha
- Desmarcar todas as linhas que possuírem a flag *Done* marcada
- Definir a ação de cada linha, como *Copy* ou *Delete* (ação *None* não é feito backup)

Para executar a cópia para arquivo vá ao menu *Tools*, escolha a opção *Copy Project*, e depois a opção *To File*. Escolha a pasta onde será feito backup e clique em *Copy*.

Ativando a Segurança na Base de Destino

Como não migramos segurança, é necessário ativar a segurança na base de destino após a migração ser concluída.

Caso você não tenha permissão de dar segurança aos objetos do seu projeto na base de destino, é necessário informar o analista para que ele solicite a liberação de segurança para a equipe de infra-estrutura.

Referências Bibliográficas

- Enterprise PeopleTools PeopleBooks: Getting Started With PeopleTools
(*Documentação PeopleSoft*)
- Enterprise PeopleTools PeopleBooks: PeopleCode Developers Guide
(*Documentação PeopleSoft*)
- Enterprise PeopleTools PeopleBooks: PeopleCode Language Reference
(*Documentação PeopleSoft*)
- Enterprise PeopleTools PeopleBooks: PeopleSoft Application Designer
(*Documentação PeopleSoft*)