Transações dinâmicas

Transações como "views"

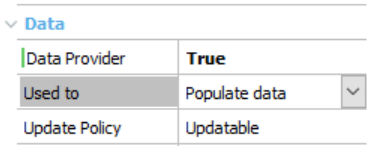
GeneXus

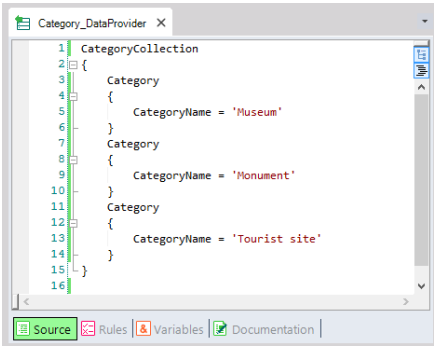
Até agora vimos que para cada objeto transação, é criada uma tabela para cada nível, para armazenar seus dados e depois recuperá-los.

Data Provider para inicializar dados

DP

(Usado para preencher dados)





Já vimos que podemos associar um Data Provider a uma transação a fim de preencher suas tabelas com dados.

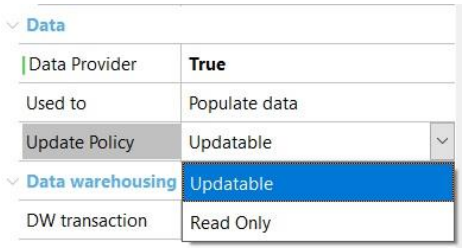
Data Provider para inicializar dados

DP

(Usado para preencher dados)

Usos de uma transação :

- Inserir, modificar e excluir dados
- Navegue, receba dados

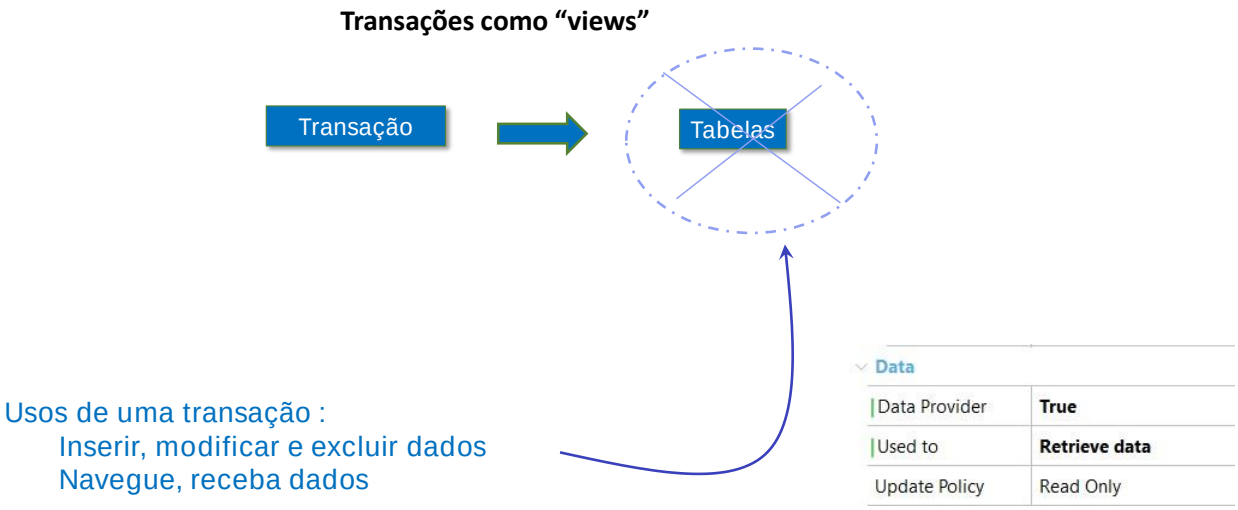


Nesse cenário, o Data Provider é utilizado apenas para inicializar. Depois, a transação se comportará normalmente e acessará suas tabelas para recuperar a informação e permitirá então **inserir, atualizar e excluir** registros da maneira usual.

Observar a propriedade **Update Policy** que assume o valor Updatable que permite estas atualizações e este comportamento

Mas também podemos modificar este comportamento da transação e impedir que os dados possam ser atualizados. Ou seja, uma vez iniciadas as tabelas com dados, estes mesmos não poderão ser alterados, nem poderão ser adicionados dados novos.
Para isto, deverá ser alterada a política de atualização, que por padrão assume o valor Updatable, e indicá-la como Read only.

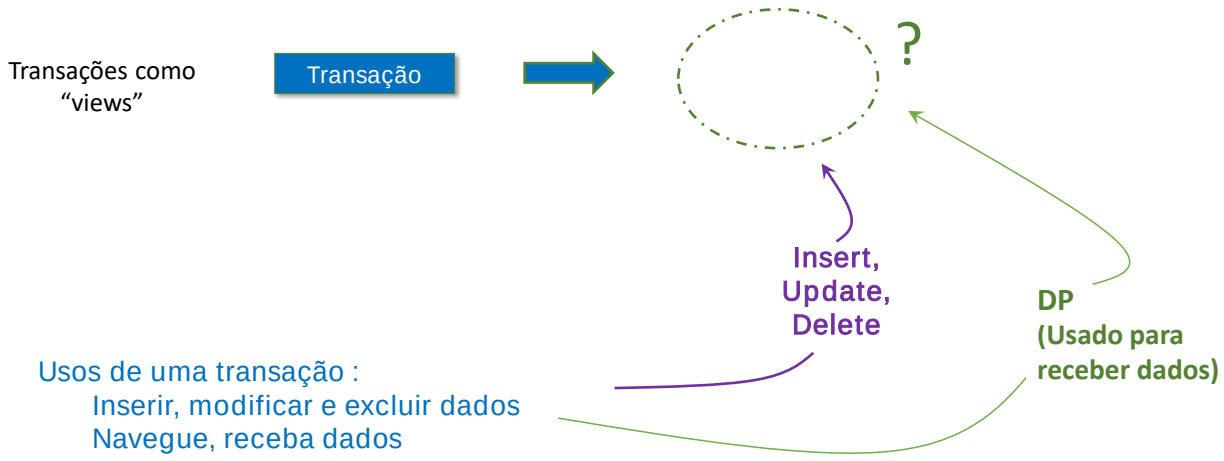
Data Provider para receber dados



Veremos agora que é possível conservar os usos habituais da transação mas sem que a informação se encontre armazenada nas tabelas associadas. Em suma, teremos um caso de transações a partir das quais não se criam tabelas na base de dados da aplicação.

Isto podemos conseguir indicando que o Data Provider associado à transação será utilizado para recuperar informações, **Retrieve data**.

Data Provider para receber dados

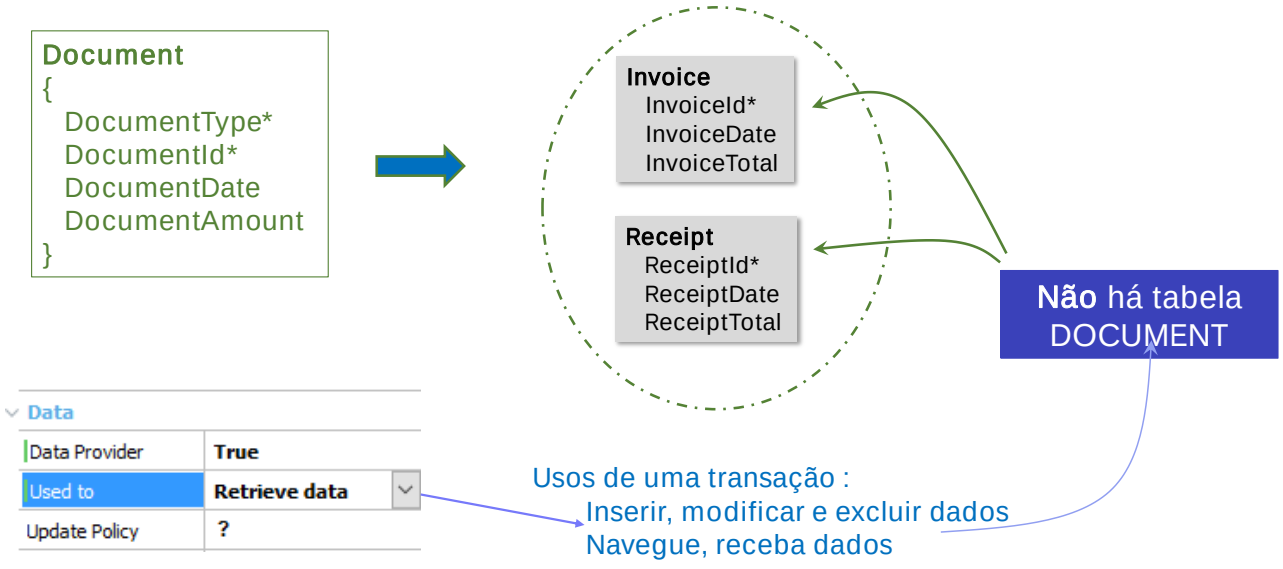


Mas se não são criadas tabelas, então, de alguma forma teremos que especificar de onde se recuperará a informação cada vez que o usuário queira navegar seus dados. E também teremos que especificar o que fazer quando o usuário inserir dados na tela e queira inseri-los, atualizá-los ou excluí-los das tabelas correspondentes.

Para Inserir, Modificar e Excluir, teremos que programar explicitamente três eventos com esses nomes.

Para recuperar a informação da transação teremos que programar o Data Provider associado à mesma.
Como no preenchimento de dados, a propriedade Update Policy, nos permitirá indicar se podemos utilizar a transação apenas para recuperar informações, ou também para atualizá-la.

Cenário: Relação de integridade do tipo **"OR"**



Estudaremos agora um exemplo.

Suponhamos que temos duas transações comuns:

- **Fatura**, para representar as faturas que emite a agência de viagens para seus clientes, seja por compras de viagens, excursões, etc. Estas faturas são identificadas com um número correlativo.
- **Recibos**, para representar os recibos que a agência de viagens emite aos seus clientes por suas compras. Estes recibos também são identificados por números correlativos entre si.

O sistema de contabilidade da Agência de Viagens precisa ser capaz de manipular estas faturas e recibos como Documentos em geral, para depois poder manejar esses Documentos em Movimentos contábeis.

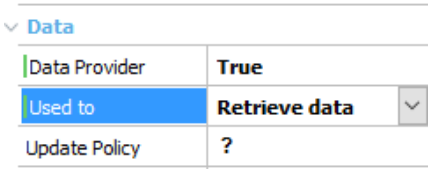
Em primeiro lugar, criamos a transação Documento, com identificador composto por DocumentType e DocumentId. Por que precisamos de um identificador composto? Para poder determinar, por exemplo, se estamos falando da fatura 1 ou do recibo 1.

Esta transação então será como uma "view" que unificará as informações contidas nas tabelas de Fatura e de Recibo. Ou seja, não criará uma tabela para conter a informação, mas a tomará das tabelas correspondentes a Fatura e a Recibo.

Cenário: Relação de integridade do tipo **"OR"**

```
Document
{
  DocumentType*
  DocumentId*
  DocumentDate
  DocumentAmount
}
```

```
DocumentCollection
{
    Document from Invoice
    {
        DocumentId = InvoiceId
        DocumentType = "Invoice"
        DocumentDate = InvoiceDate
        DocumentAmount = InvoiceTotal
    }
    Document from Receipt
    {
        DocumentId = ReceiptId
        DocumentType = "Receipt"
        DocumentDate = ReceiptDate
        DocumentAmount = ReceiptTotal
    }
}
```



Para isso declaramos sua propriedade Data Provider com valor True, e indicamos que usaremos esse Data Provider para receber informação.

Feito isto, automaticamente GeneXus entenderá que não deverá criar a tabela associada à transação pois nesse Data Provider se declarará de onde obter os dados. No nosso caso, será a partir das tabelas de FATURA e RECIBO associadas às correspondentes transações.

Vejamos agora o source do Data Provider associado a esta transação.

Temos um grupo Document para devolver todos os documentos que são faturas, e outro grupo para devolver todos os documentos que são recibos.

A partir daqui, toda vez que executemos a transação para navegar por seus dados, será executado este Data Provider que será o que carregará a informação apropriada na tela de modo absolutamente transparente, tanto para o desenvolvedor e para o usuário, ninguém perceberá que se trata de uma transação sem tabela.

Cenário: Relação de integridade do tipo **"OR"**

Transação dinâmica como transação base: : Lista de Documentos

```
Document
{
  DocumentType*
  DocumentId*
  DocumentDate
  DocumentAmount
}
```

```
For each Document order (DocumentDate)
      print Documents
Endfor
```

Depois disso, a transação dinâmica é utilizada como qualquer outra transação. Por exemplo, se queremos listar todos os documentos ordenados em forma decrescente por data, podemos criar um procedimento com um For each como o que estamos vendo:

É muito interessante pararmos para observar que este For each tem declarado Document como transação base. E, a partir disto, dado que os atributos mencionados no printblock pertencem a Document, então GeneXus determina que a tabela base deste For each é Document.

Mas Document não existe como uma tabela em nossa base de dados, em vez disso é uma view através do Data Provider.

Cenário: Relação de integridade do tipo **"OR"**

Atualização de dados: Inserir, Modificar e Excluir

```
Document
{
  DocumentType*
  DocumentId*
  DocumentDate
  DocumentAmount
}
```

Eventos →

```
Event Insert
     If DocumentType = Type.Invoice
          &Invoice = New()
          &Invoice.InvoiceId = DocumentId
          &Invoice.InvoiceDate = DocumentDate
          &Invoice.InvoiceTotal = DocumentAmount
          &Invoice.Insert()
     else
          &Receipt = New()
          &Receipt.ReceiptId = DocumentId
          &Receipt.ReceiptDate = DocumentDate
          &Receipt.ReceiptTotal = DocumentAmount
          &Receipt.Insert()
     endif
Endevent
```

No entanto, as transações não são utilizadas apenas para recuperar seus dados, mas também para atualizá-los. Como podemos fazer então, uma vez que não temos uma tabela associada a esta transação?

Observemos a propriedade Update Policy. Se esta propriedade assume o valor "Updatable" serão oferecidos então os eventos Insert, Update e Delete na transação, para programar como inserir, atualizar e excluir a informação que o usuário completa na tela. Somente o desenvolvedor saberá o que deve fazer em cada um destes casos com esta informação.

Dependendo da realidade estas ações serão permitidas, ou não.

Suponhamos que, em nosso exemplo, estas ações são permitidas. Observemos o Form da transação. Quando o usuário tiver terminado de completar os campos desta tela para inserir um novo documento e pressionar o botão Confirm, teremos que inserir um novo registro na tabela Fatura ou Recibo conforme corresponda, dependendo do valor que tenha inserido no atributo DocumentType.

Observemos, então, o setor dos Eventos da transação. Programamos o evento Insert, utilizando para isto as variáveis Fatura e Recibo baseadas nos tipos de dados Business Component de Fatura e Recibo, respectivamente.

Vale mencionar que, em vez do método Insert do business component, poderíamos ter utilizado o método Save. Observemos que não precisamos escrever o commit, uma vez que estamos no contexto da transação Document, que ainda tem a propriedade Commit on exit em Yes por padrão, e isso significa que fará seu commit de forma implícita.

Regras e acionamento de eventos

```
Document
{
  DocumentType*
  DocumentId*
  DocumentDate
  DocumentAmount
}
```

Regras →

```
Default( DocumentDate, &Today );

Error( "Document Amount must be greater than 0" )
    if DocumentAmount <= 0;
```

- Arvore de avaliação e momentos de disparo das regras são idênticos a se tratarmos com uma transação habitual.
- Eles são especificados e acionados como em uma transação normal.

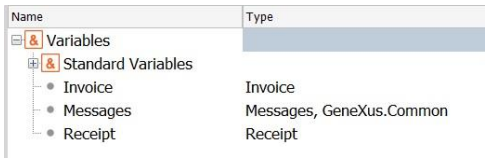
Vejamos agora o que acontece se temos regras especificadas no nível da transação dinâmica
.
Em que momento irão se disparar? O que ocorre com a árvore de avaliação?

Bem, tanto a árvore de avaliação como os momentos de disparo das regras são idênticos a se tratarmos com uma transação habitual.

Mensagens desencadeadas na execução do Business Component

```
Document
{
  DocumentType*
  DocumentId*
  DocumentDate
  DocumentAmount
}
```

Eventos →



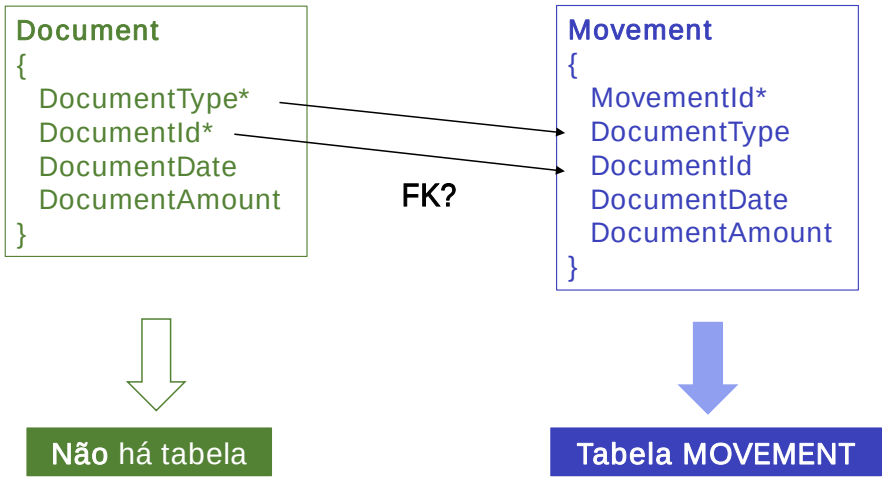

```
Event Insert(&Messages)
     If DocumentType = Type.Invoice
          &Invoice = New()
          &Invoice.InvoiceId = DocumentId
          &Invoice.InvoiceDate = DocumentDate
          &Invoice.InvoiceTotal = DocumentAmount
          &Invoice.Insert()
          &Messages = &Invoice.GetMessages()
     else
          &Receipt = New()
          &Receipt.ReceiptId = DocumentId
          &Receipt.ReceiptDate = DocumentDate
          &Receipt.ReceiptTotal = DocumentAmount
          &Receipt.Insert()
          &Messages = &Receipt.GetMessages()
     endif
Endevent
```

Quanto às mensagens de êxito ou fracasso disparadas na execução dos Business Components Fatura e Recibo, também podemos recuperá-las.

Para isso declaramos a variável Messages, baseada no tipo de dados Messages, coleção, como parâmetro em cada um dos eventos Insert, Update e Delete no nível da transação.

Estas mensagens são exibidas no form da transação dinâmica de forma totalmente transparente.

Integridade referencial



Avançando um pouco mais em nosso exemplo, recordemos que havíamos dito que o sistema contábil da Agência de Viagens precisa tratar todos os documentos como movimentos contábeis.

Já que não se criará a tabela DOCUMENTO associada à transação Documento, poderíamos supor que, então, na tabela MOVIMENTO associada à transação padrão Movimento, o par de atributos formado por DocumentType e DocumentId não poderão constituir a chave estrangeira que deveriam.

Então o que vai acontecer com o controle da integridade referencial? GeneXus poderá resolvê-lo?

Como a integridade referencial deve ser assegurada, GeneXus gera triggers de SQL para assegurá-la. Portanto, podemos dizer que o par formado por DocumentType e DocumentId constituem em Movimento uma "pseudo" chave estrangeira.

Em suma, não será permitido eliminar em Document Faturas ou Recibos para os quais exista um movimento, e tampouco permitirá adicionar um Movimento que não exista como Documento.

Resumo

1. Data Provider: True

2. Used to: Populate Data

DP (para inicializar)

3. Update Policy: Updatable

Permite (ou não) fazer atualizações nas tabelas.

1 Data Provider: True

2. Used to: Retrieve Data

DP (para receber)

3. Update Policy: Updatable

Permitir (ou não) fazer atualizações. Os eventos devem ser programados **Insert, Update, Delete**

Vamos agora revisar os conceitos que aprendemos :

Quando a propriedade Data Provider no nível de uma transação assume o valor True, GeneXus nos pergunta para que vamos utilizá-lo.

Se indicamos que o usaremos para preencher a tabela com dados, então a transação gerará suas correspondentes tabelas associadas.

Através da propriedade Update policy permitiremos, ou não, a atualização de registros da forma habitual

Quando a propriedade Data Provider associada a uma transação assume o valor True e indicamos que vamos utilizá-la para receber dados, então GeneXus entende que esta transação não terá tabelas associadas e que realizará uma view a partir deste Data Provider.

Em seguida, de acordo com o valor que indicamos na propriedade Update policy, deverão ser programados os correspondentes eventos para permitir inserir, modificar ou excluir registros nas tabelas correspondentes.

Mais sobre transações dinâmicas...

- Outros casos de uso:

  Seleção
  Agrupamento de dados
  Relações temporárias

- Mais exemplos de uso de transações dinâmicas

  http://wiki.genexus.com/commwiki/servlet/wiki?28062,Dynamic%20Transactions.

Para finalizar, vale mencionar que vimos um único caso de uso de transações dinâmicas, mas há múltiplos casos como, por exemplo:
Seleção, Agrupamento de dados e Relações temporárias que abordaremos em outros cursos.

Podem ser acessadas mais informações a partir do link na tela.

GeneXus™

training.genexus.com
wiki.genexus.com