Transacciones Dinámicas

Transacciones como "Vistas"

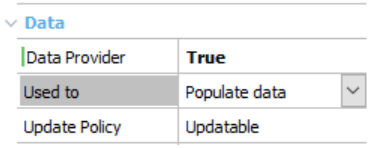
GeneXus

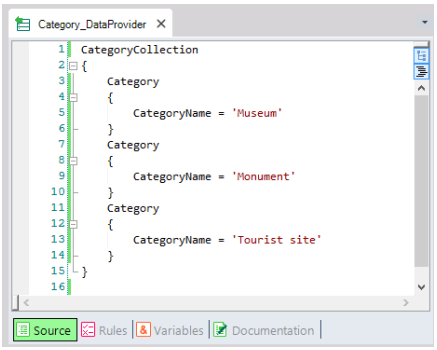
Hasta ahora hemos visto que por cada objeto transacción se crea una tabla por cada nivel, para almacenar sus datos y luego recuperarlos.

Data Provider para inicializar datos

DP

(Usado para poblar datos)





Ya hemos visto que podemos asociar un Data Provider a una transacción con el fin de poblar sus tablas con datos.

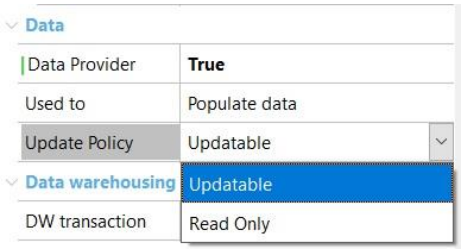
Data Provider para inicializar datos

DP

(Usado para poblar datos)

Usos de una Transacción:

- Insertar, Modificar, Eliminar datos
- Recorrer, recibir datos

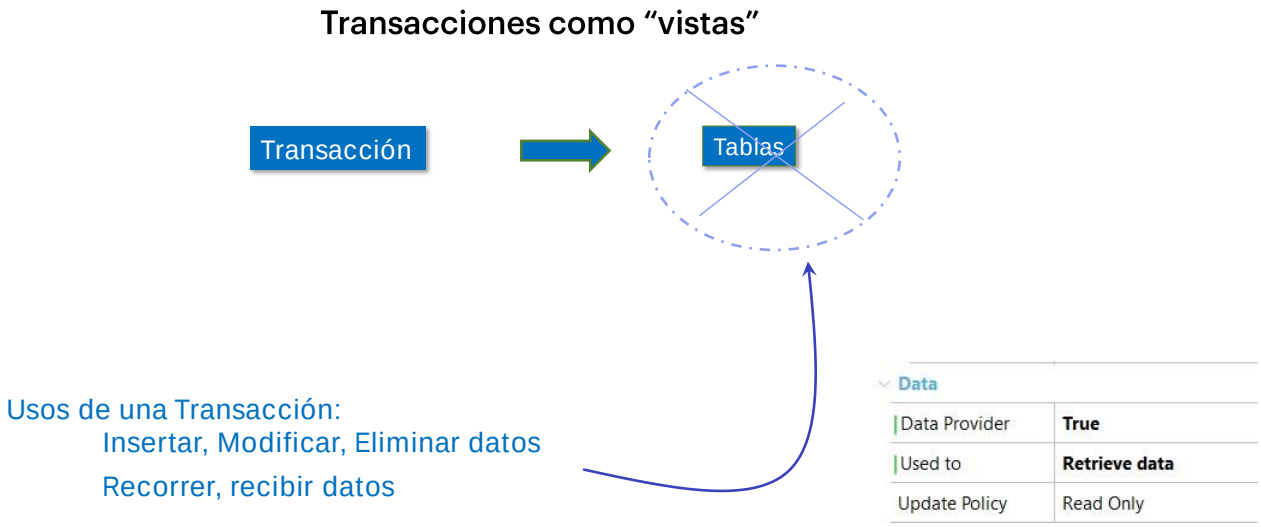


En ese escenario el Data Provider es utilizado únicamente para inicializar. Luego la transacción se comportará normalmente y accederá a sus tablas para recuperar la información y permitirá entonces **insertar, actualizar y eliminar** registros del modo usual.

Observar la propiedad **Update Policy** que toma el valor Updatable que permite estas actualizaciones y este comportamiento.

Pero también podemos modificar este comportamiento de la transacción e impedir que se puedan actualizar los datos. Es decir, una vez inicializadas las tablas con datos, estos mismos no podrán modificarse ni podrán agregarse datos nuevos.
Para esto se deberá cambiar la política de actualización, que por defecto toma el valor Updatable, e indicarla como Read only.

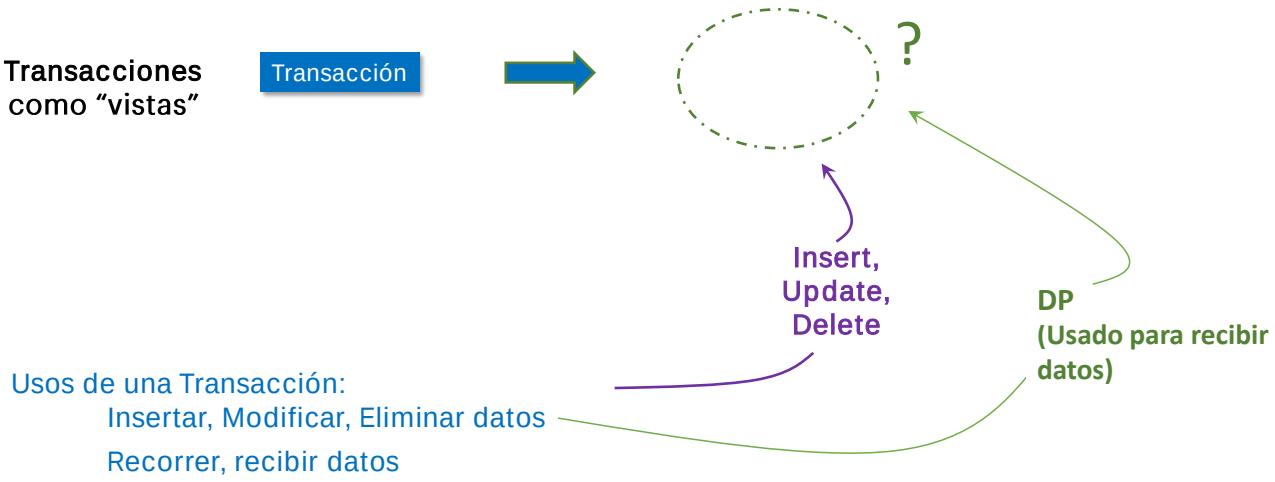
Data Provider para recibir datos



Veremos ahora que es posible conservar los usos habituales de la transacción pero sin que la información se encuentre almacenada en las tablas asociadas. En definitiva, tendremos un caso de transacciones a partir de las cuales no se crean tablas en la base de datos de la aplicación

Esto lo podemos lograr indicando que el Data Provider asociado a la transacción será utilizado para recuperar información, **Retrieve data**.

Data Provider para recibir datos

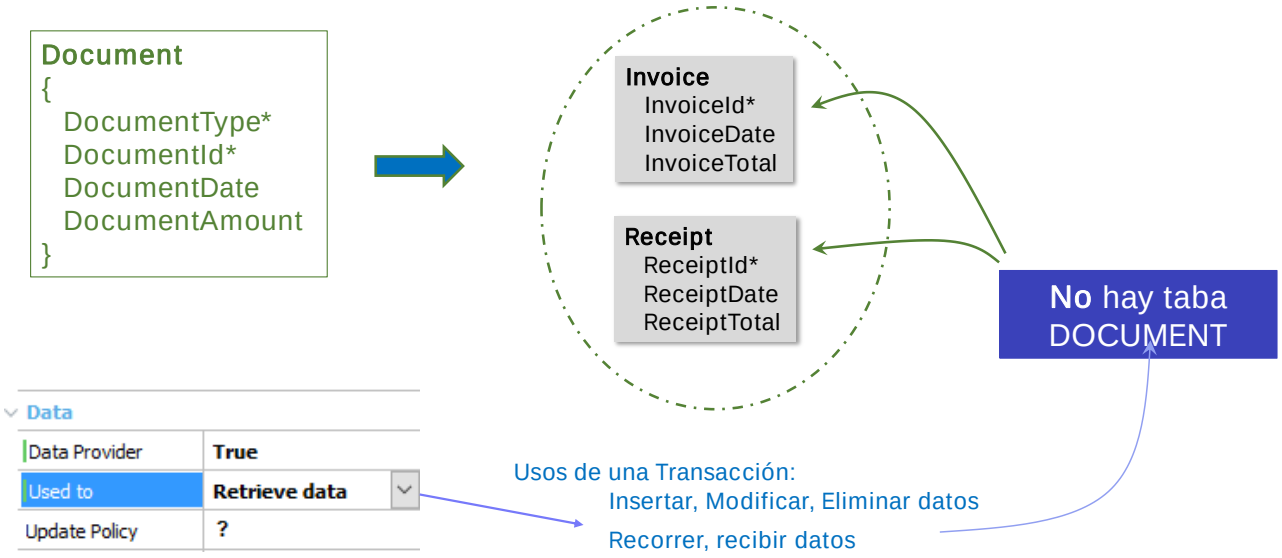


Pero si no se crean tablas, entonces de alguna manera tendremos que especificar de dónde se recuperará la información cada vez que el usuario quiera navegar sus datos. Y también tendremos que especificar qué hacer cuando el usuario ingrese datos en la pantalla y quiera insertarlos, actualizarlos o eliminarlos de las tablas correspondientes.

Para Insertar, Modificar y Eliminar tendremos que programar explícitamente tres eventos con esos nombres.

Para recuperar la información de la transacción tendremos que programar el Data Provider asociado a la misma.
Al igual que en el poblado de datos, la propiedad Update Policy, nos permitirá indicar si podemos utilizar la transacción solo para recuperar información, o también para actualizarla.

Escenario: Relación de integridad de tipo “OR”



Estudiaremos ahora un ejemplo:

Supongamos que tenemos dos transacciones comunes:

- **Factura**, para representar las facturas que emite la agencia de viajes a sus clientes, ya sea por compras de viajes, excursiones, etc. Estas facturas se identifican con un número correlativo.
- **Recibos**, para representar los recibos que la agencia de viajes le emite a sus clientes por sus compras. Estos recibos también se identifican por números correlativos entre sí.

El sistema contable de la Agencia de Viajes necesita poder manipular estas facturas y recibos como Documentos en general, para luego poder manejar esos Documentos en Movimientos contables

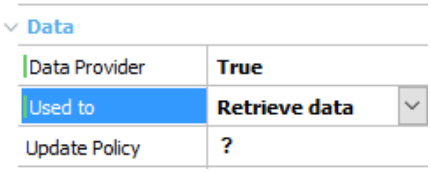
Creamos la transacción Documento, con identificador compuesto por DocumentType y DocumentId. ¿Por qué necesitamos un identificador compuesto? Para poder determinar, por ejemplo, si estamos hablando de la factura 1 o del recibo 1.

Esta transacción entonces será como una “vista” que unificará la información contenida en las tablas de Factura y de Recibo. Es decir, no creará una tabla para contener la información sino que la tomará de las tablas correspondientes a Factura y a Recibo.

Escenario: Relación de integridad de 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 eso declaramos su propiedad Data Provider con valor True, e indicamos que usaremos dicho Data Provider para recibir información.

Hecho esto, automáticamente GeneXus entenderá que no deberá crear la tabla asociada a la transacción pues en ese Data Provider se declarará de dónde obtener los datos. En nuestro caso, será a partir de las tablas de FACTURA y RECIBO asociadas a las correspondientes transacciones.

Veamos ahora el source del Data Provider asociado a esta transacción.

Tenemos un grupo Document para devolver todos los documentos que son facturas, y otro grupo para devolver todos los documentos que son recibos.

A partir de aquí toda vez que ejecutemos la transacción para navegar por sus datos, se ejecutará este Data Provider que será el que cargará la información apropiada en la pantalla de modo absolutamente transparente tanto para el desarrollador y para el usuario, Nadie percibirá que se trata de una transacción sin tabla.

Escenario: Relación de integridad de tipo “OR”

Transacción dinámica como transacción base: Listado de Documentos

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

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

Luego de esto, la transacción dinámica se utiliza como cualquier otra transacción. Por ejemplo, si queremos listar todos los documentos ordenados en forma descendente por fecha, podemos crear un procedimiento con un For each como el que estamos viendo:

Es muy interesante detenerlos a observar que este For each tiene declarada Document como transacción base. Y a partir de esto, dado que los atributos mencionados en el printblock pertenecen a Document, entonces GeneXus determina que la tabla base de este For each es Document.

Pero Document no existe como una tabla en nuestra base de datos sino que es una vista a través del Data Provider.

Escenario: Relación de integridad de tipo “OR”

Actualización de datos: Insertar, Modificar y Eliminar

```
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
```

Ahora bien, las transacciones no se utilizan solamente para recuperar sus datos sino también para actualizarlos. ¿Cómo podemos hacer entonces dado que no tenemos una tabla asociada a esta transacción?

Observemos la propiedad Update Policy. Si esta propiedad toma el valor “Updatable” se ofrecerán entonces los eventos Insert, Update y Delete en la transacción, para programar cómo insertar, actualizar y eliminar la información que el usuario completa en la pantalla. Sólo el desarrollador sabrá qué debe hacer en cada uno de estos casos con esta información.

Dependiendo de la realidad se permitirán estas acciones, o no.

Observemos el Form de la transacción. Cuando el usuario haya terminado de completar los campos de esta pantalla para insertar un nuevo documento y presione el botón Confirm, tendremos que insertar un nuevo registro en la tabla Factura o Recibo según corresponda, dependiendo del valor que haya ingresado en el atributo DocumentType.

Observemos entonces el sector de los Eventos de la transacción. Hemos programado el evento Insert, utilizando para esto las variables Factura y Recibo basadas en los tipos de datos Business Component de Factura y Recibo respectivamente.

. Observemos que no necesitamos escribir el commit, puesto que estamos en el contexto de la transacción Document, que sigue teniendo la propiedad Commit on exit en Yes por defecto, y esto significa que hará su commit en forma implícita.

Reglas y disparo de eventos

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

Reglas →

```
Default( DocumentDate, &Today );

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

- Arbol de evaluación y momentos de disparo sin iguales a una transacción habitual.
- Son especificadas y disparadas igual que en una transacción habitual.

Veamos ahora qué sucede si tenemos reglas especificadas a nivel de la transacción dinámica.

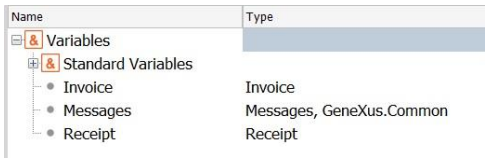
¿En qué momento se van disparando? Qué pasa con el árbol de evaluación?

Bien, tanto el árbol de evaluación como los momentos de disparo de las reglas es idéntico a si tratáramos con una transacción habitual.

Mensajes disparados en la ejecución del Business Component

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

Eventos →



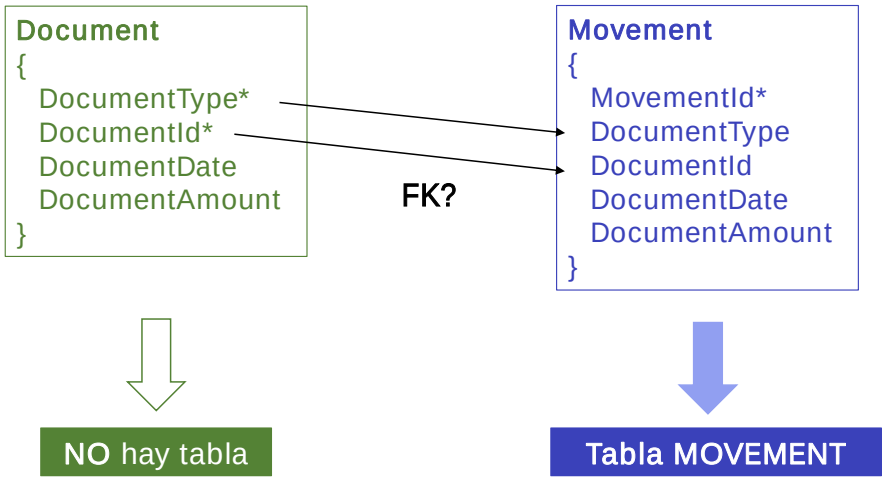

```
Event Insert(&Messages)
     If DocumentType = Type.Invoice
          &Invoice = New()
          &Invoice.InvoiceId = DocumentId
          &Invoice.InvoiceDate = DocumentDate
          &Invoice.InvoiceTotal = DocumentAmount
          &Invoice.Insert()
          &Messages = &Invoice.GetMessages()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

En cuanto a los mensajes de éxito o fracaso disparados en la ejecución de los Business Components Factura y Recibo, también podemos recuperarlos.

Para eso declaramos la variable Messages, basada en el tipo de datos Messages, colección, como parámetro en cada uno de los eventos Insert, Update y Delete a nivel de la transacción.

Estos mensajes son desplegados en el form de la transacción dinámica de forma totalmente transparente.

Integridad referencial



Avanzando un poco más en nuestro ejemplo, recordemos que hemos dicho que el sistema contable de la Agencia de Viajes necesita manejar a todos los documentos como movimientos contables.

Ya que no se creará la tabla DOCUMENTO asociada a la transacción Documento, podríamos suponer que entonces en la tabla MOVIMIENTO asociada a la transacción estándar Movimiento, el par de atributos conformado por DocumentType y DocumentId no podrán constituir la llave foránea que deberían.

Entonces qué va a suceder con el control de la integridad referencial? GeneXus podrá resolverlo?

Como la integridad referencial debe asegurarse, GeneXus genera triggers de SQL para asegurarla. Por lo tanto podemos decir que el par formado por DocumentType y DocumentId constituyen en Movimiento una "pseudo" llave foránea.

En definitiva, no se permitirá eliminar en Document Racturas o Recibos para los que haya un movimiento, y tampoco se permitirá ingresar un Movimiento que no exista como Documento.

Resumen

1. Data Provider: True

2. Used to: Populate Data

DP (para inicializar)

3. Update Policy: Updatable

Permite (o no) hacer actualizaciones sobre las tablas.

1 Data Provider: True

2. Used to: Retrieve Data

DP (para recibir)

3. Update Policy: Updatable

Permite (o no) hacer actualizaciones.
Se deben programar los eventos:
Insert, Update, Delete

Repasemos ahora los conceptos que hemos aprendido:

Cuando la propiedad Data Provider a nivel de una transacción toma el valor True, GeneXus nos pregunta para qué lo vamos a utilizar.
Si indicamos que lo usaremos para poblar la tabla con datos, entonces la transacción generará sus correspondientes tablas asociadas.
A través de la propiedad Update policy permitiremos, o no, la actualización de registros de la forma habitual

Cuando la propiedad Data Provider asociada a una transacción toma el valor True, e indicamos que lo vamos a utilizar para recibir datos, entonces GeneXus entiende que dicha transacción no tendrá tablas asociadas y que realizará una vista a partir de dicho Data Provider.

Luego, según el valor que indiquemos en la propieda Update policy, se deberán programar los correspondientes eventos para permitir insertar, modificar o eliminar registros en las tablas correspondientes.

Más sobre Transacciones dinámicas...

- Otros casos de uso:

  Selección
  Agrupamiento de datos
  Relaciones temporales

- Más ejemplos de uso de transaciones dinámicas:

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

Para finalizar, vale mencionar que hemos visto un único caso de uso de transacciones dinámicas, pero hay múltiples casos, como por ejemplo:
Selección, Agrupación de datos y Relaciones temporales que abordaremos en otros cursos.

Se puede acceder a más información desde el link en pantalla.

GeneXus™

training.genexus.com
wiki.genexus.com