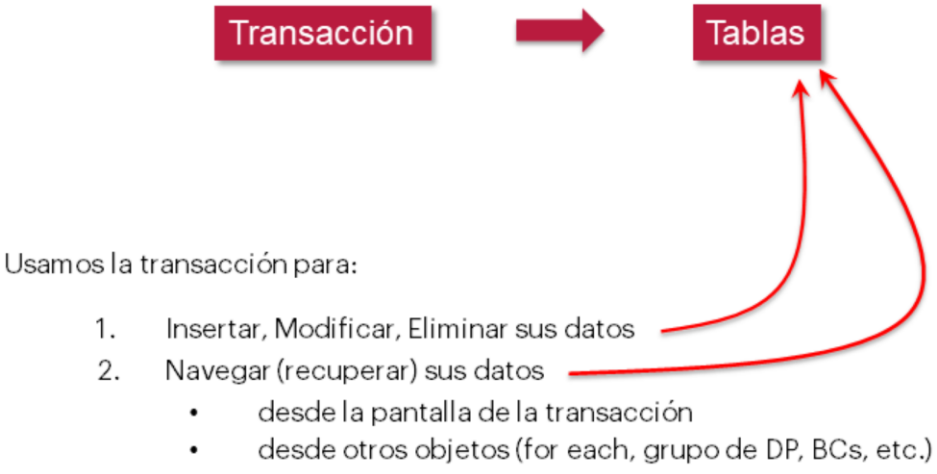
Transacciones dinámicas
Recuperación de datos a demanda
Transacciones como “vistas”

GeneXus™ 16

Transacción estándar

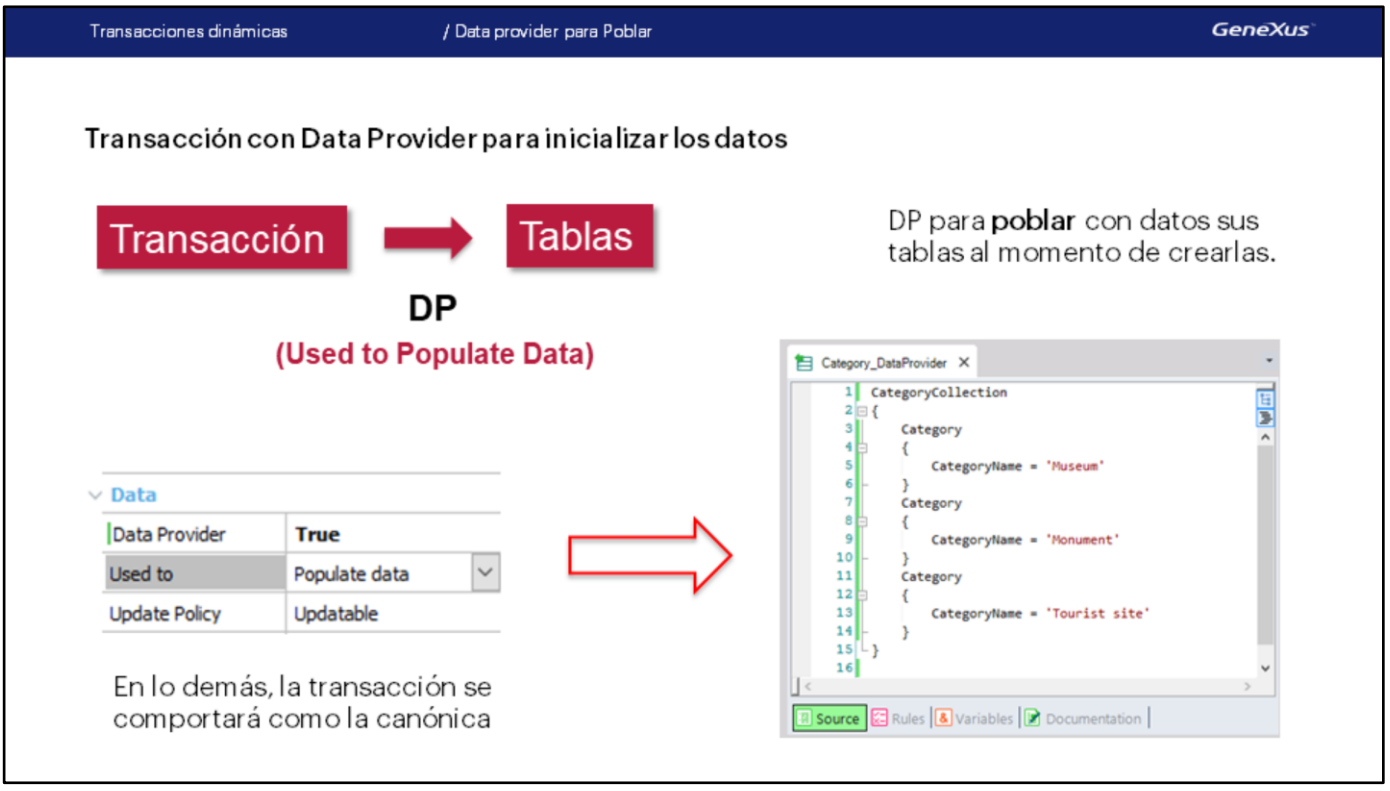


Usamos la transacción para:

1. Insertar, Modificar, Eliminar sus datos
2. Navegar (recuperar) sus datos
   - desde la pantalla de la transacción
   - desde otros objetos (for each, grupo de DP, BCs, etc.)

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

La transacción, en su forma canónica, se utiliza para poder realizar las operaciones de **inserción**, **modificación** y **eliminación** en esas tablas a través de su pantalla, y, por otro lado, para poder navegar (**recuperar**) los datos de esas tablas.

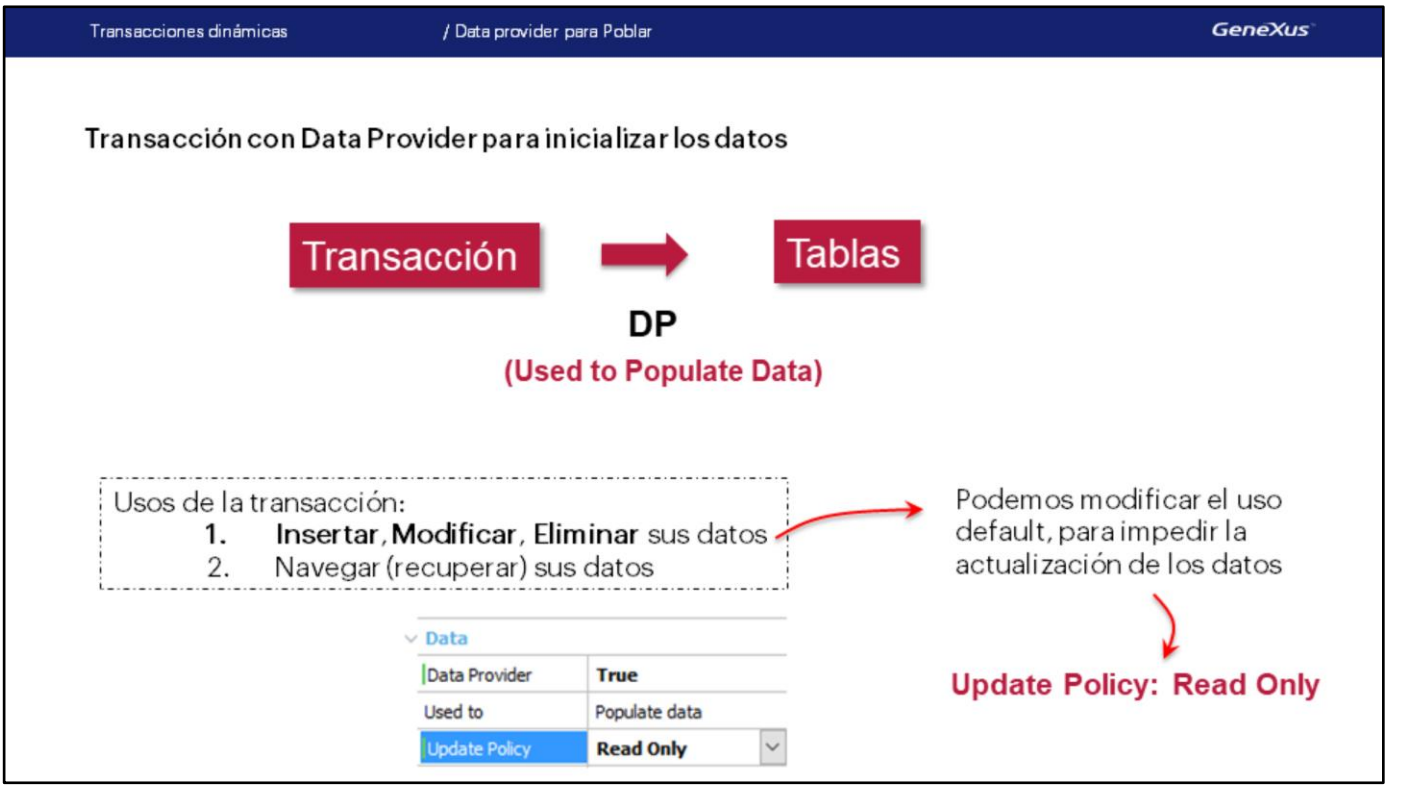


Ya habíamos visto cómo podíamos asociar un **Data Provider a la transacción** a los efectos de poder **poblar** su/s tabla/s con datos.

Recordemos que el Data Provider usado para ("Used to") poblar con datos ("Populate data") se ejecutará en la reorganización, cuando las tablas asociadas a la transacción se creen.

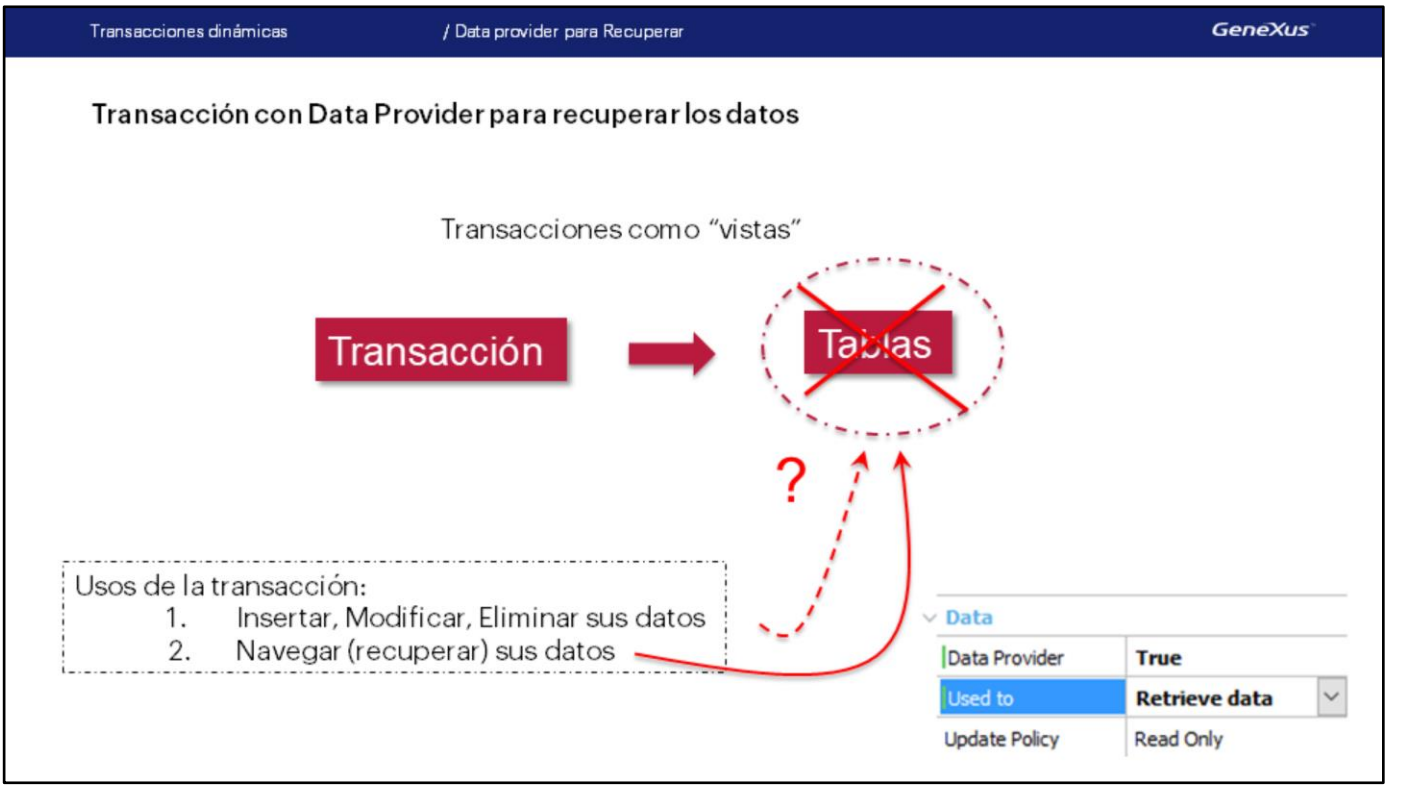
Nota: Si ese Data Provider es modificado en algún momento posterior será ejecutado nuevamente en el próximo F5. Por lo que hay que tener cuidado con los datos que ya estuvieran presentes en las tablas.

El Data Provider es únicamente utilizado para inicializar. Luego la transacción se comportará como la canónica, es decir, accederá normalmente a sus tablas para recuperar la info y permitirá **insertar**, **actualizar** y **eliminar** registros del modo usual. Observar la propiedad **Update Policy** y su valor **Updatable**.



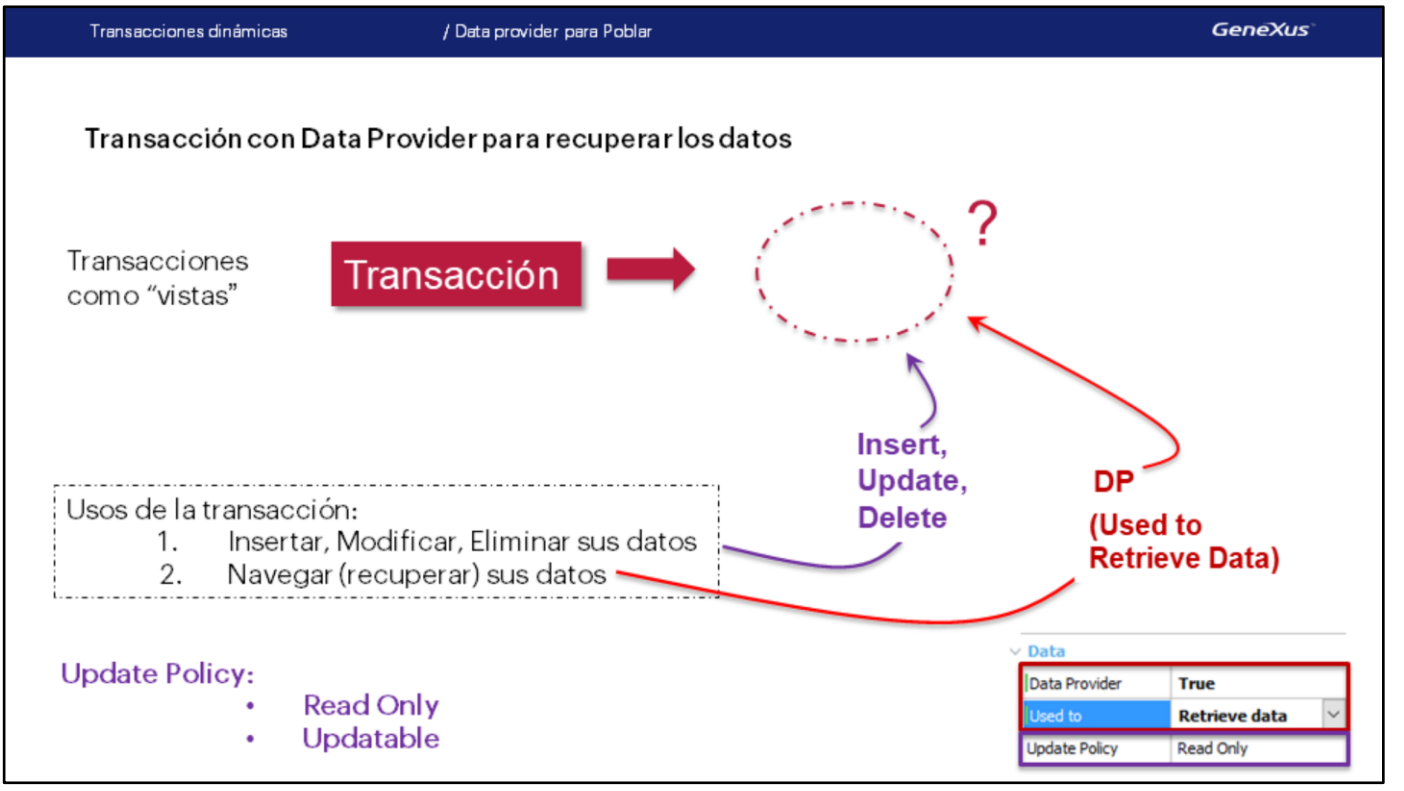
Pero también podemos modificar el uso default de la transacción de modo de impedir que se puedan actualizar sus datos. Es decir, una vez inicializadas las tablas con datos, esos datos no podrán modificarse ni podrán agregarse nuevos.

Para ello, habiendo especificado que la transacción tendrá un Data Provider asociado, Usado para "Populate Data", se cambiará la política de actualización que por defecto es "Updatable" a "Read Only".



Aquí veremos que es posible conservar los usos de la transacción —insertar, modificar, eliminar, así como navegar (recuperar) sus datos— sin que la información esté almacenada en las tablas canónicas. En definitiva tendremos un caso de transacciones que no crean tablas en la base de datos de la aplicación.

Si no crean tablas entonces tendremos que especificar de dónde se recuperará la información cada vez que se quieran navegar sus datos. Y también tendremos que especificar qué hacer cuando el usuario ingrese datos en la pantalla y quiera "insertarlos" en "las tablas" (o "actualizarlos" o "eliminarlos").

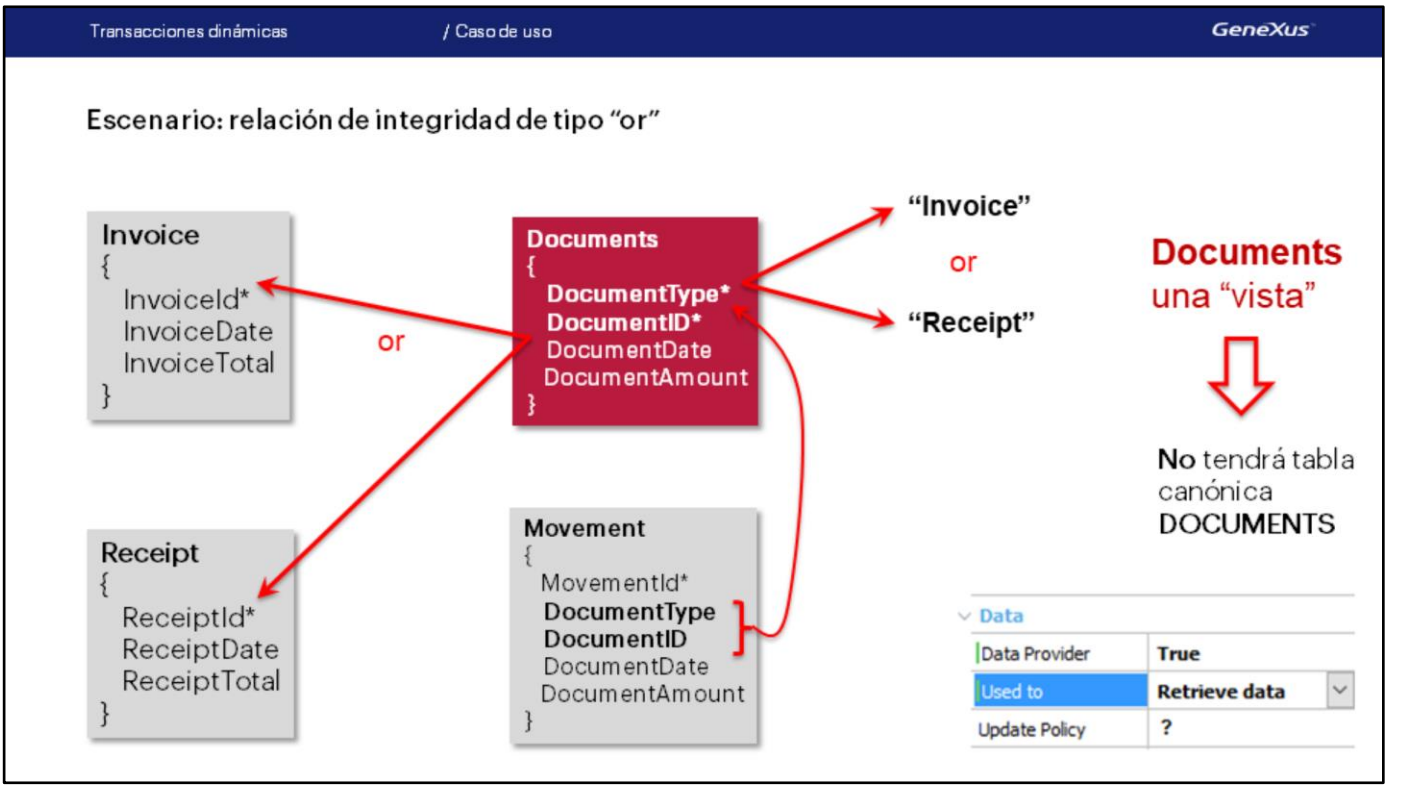


Para Insertar, Modificar, Eliminar, tendremos que programar explícitamente tres eventos con esos nombres.

Para Navegar (recuperar) la información de la transacción, tendremos que programar el Data Provider asociado a la transacción.

Así como en el caso en el que usamos un Data Provider sólo para poblar con datos las tablas podíamos evitar las actualizaciones indicando con una propiedad que la política sería Read Only, aquí también podemos querer utilizar la transacción sólo para recuperar su información, y no para actualizarla. La propiedad será la misma, Update Policy, que asume los dos valores que mostramos.

Estudiaremos en lo que sigue un ejemplo para clarificar el qué y el cómo.

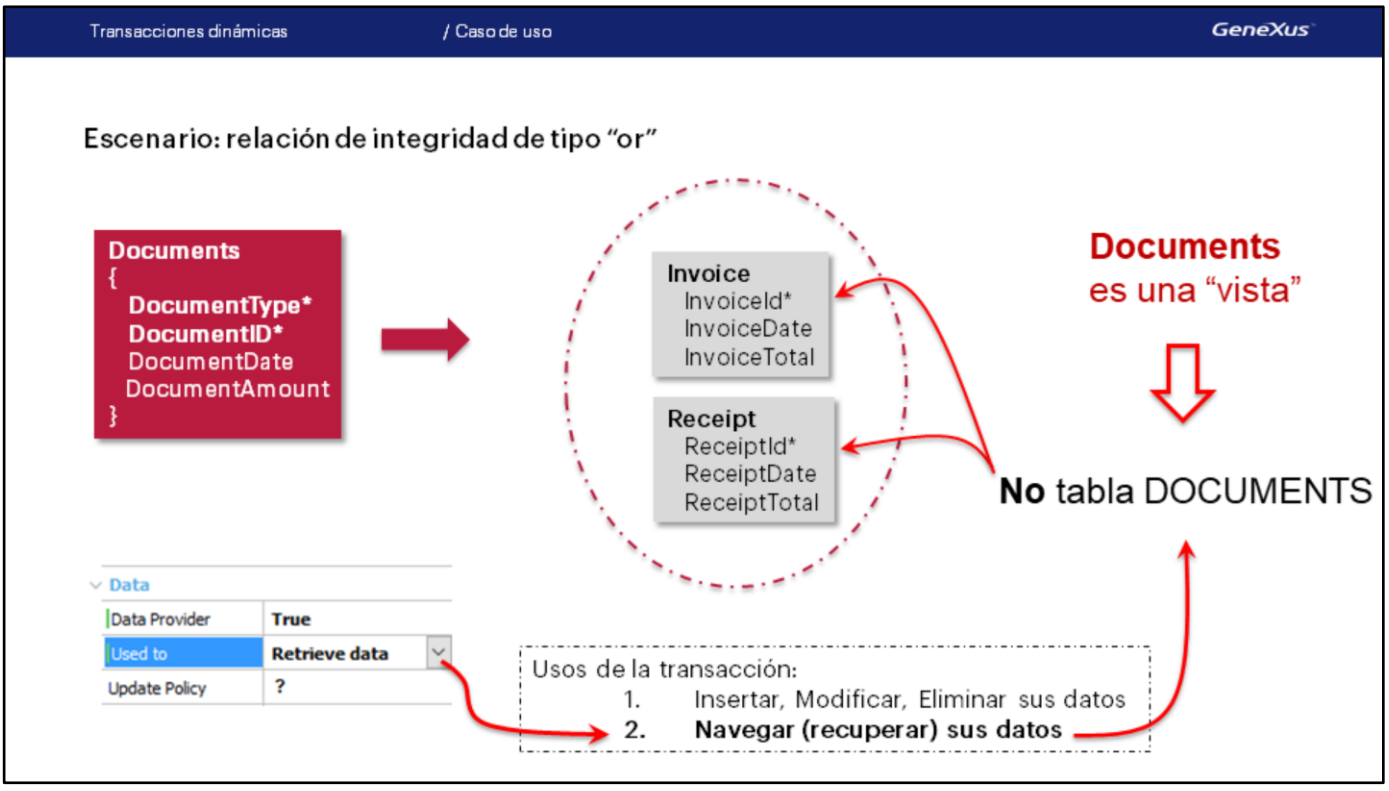


Supongamos que tenemos dos transacciones comunes:

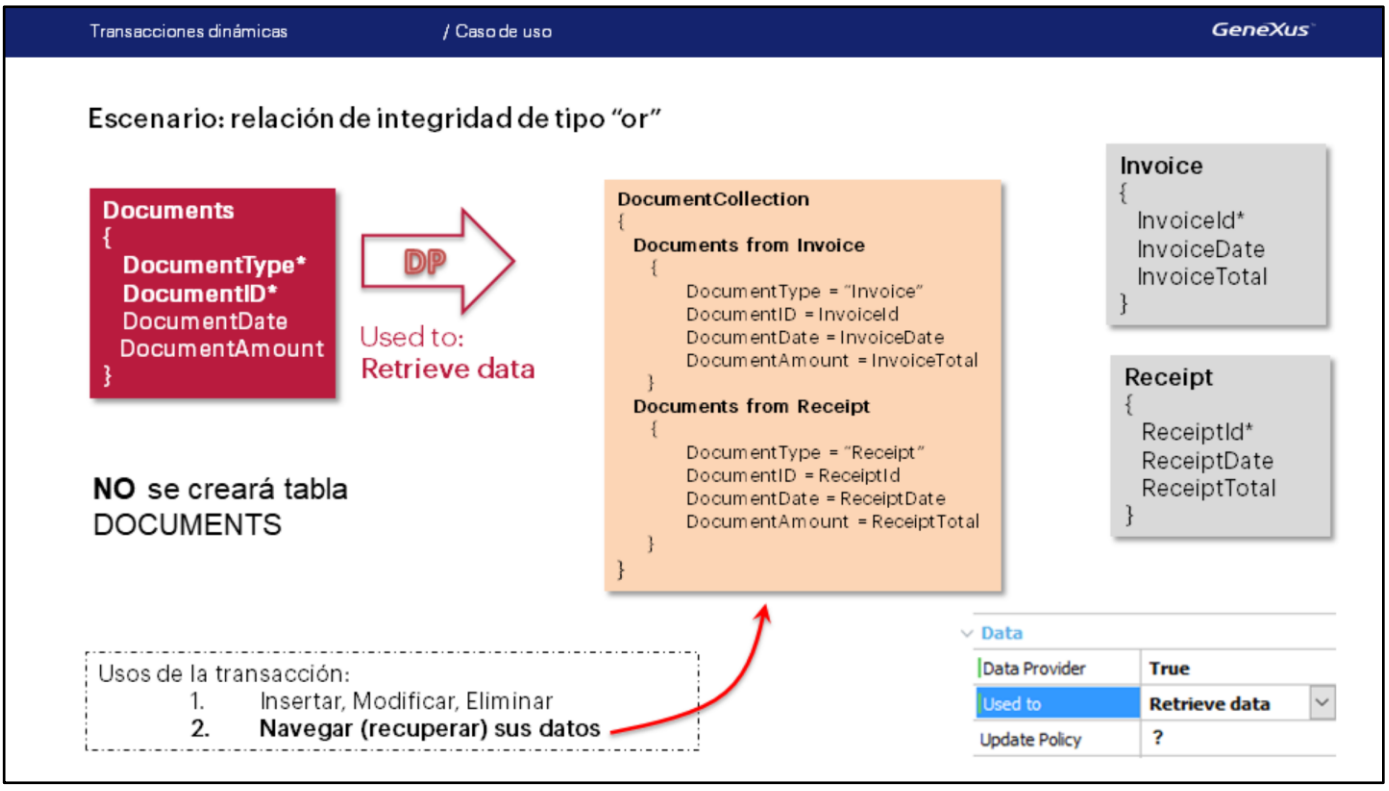
- Invoice, para representar las facturas que emite la agencia de viajes a sus clientes por compras de viajes, excursiones, etc. Estas facturas se identifican con un número correlativo.
- Receipt, para representar los recibos que la agencia de viajes le emite a sus clientes por sus compras. Los recibos también se identifican por números correlativos entre sí.

El sistema contable de la agencia de viajes necesitará manipular facturas y recibos en "movimientos". En los movimientos los recibos son un tipo de documento, así como lo son las facturas. Los movimientos se identifican por un id único, autonumerado. Obsérvese que el movimiento 1 puede ser de la factura 1, y el movimiento 2 del recibo 1. La transacción de movimientos homogeiniza la información de facturas y recibos.

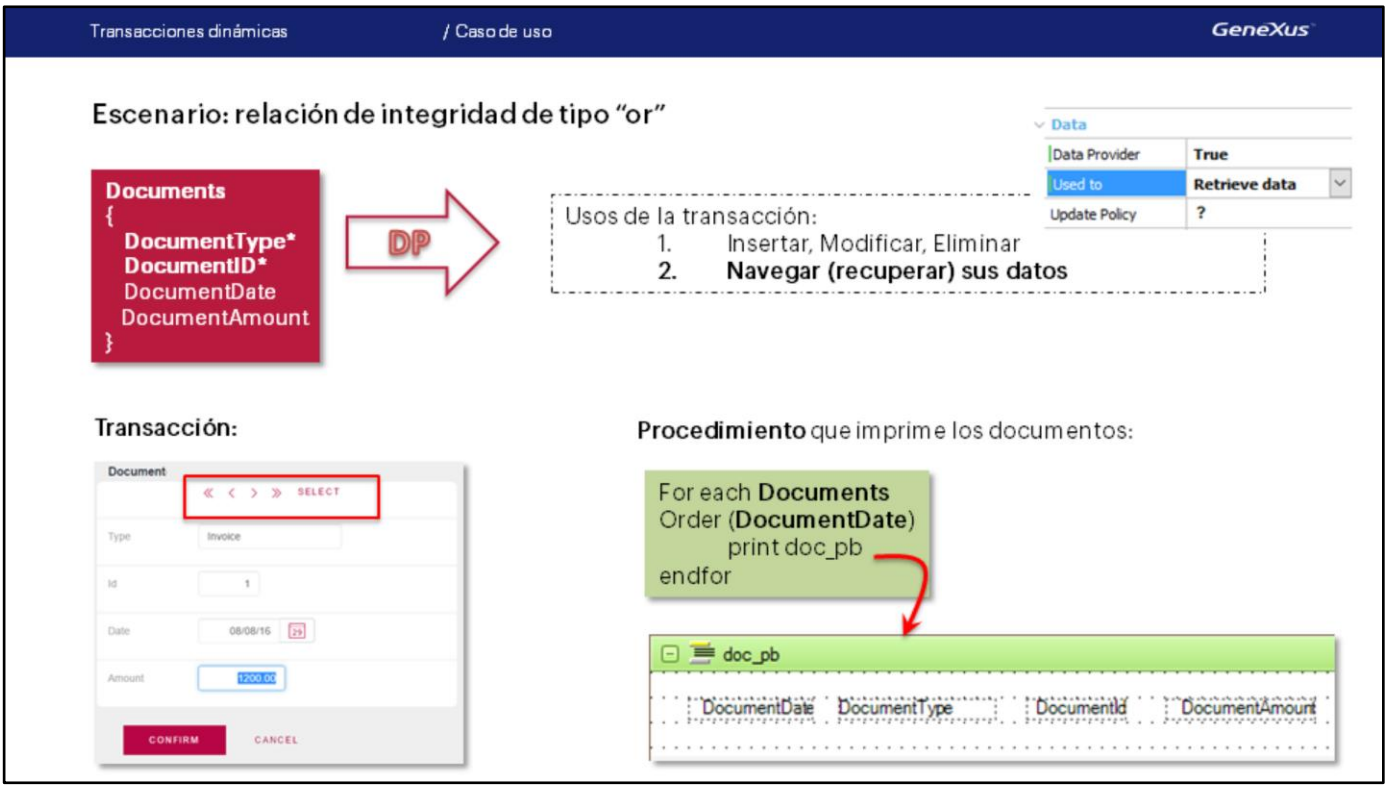
Es por ello que se crea la transacción Documents, con identificador compuesto por DocumentType y DocumentID. Esta transacción será como una "vista" que conjuga la información contenida en las tablas de Invoice y de Receipt. Es decir, no creará una tabla para contener la información sino que la tomará de las tablas correspondientes a Invoice y a Receipt, de idénticos nombres. Luego, la transacción Movement será una transacción común, que generará una tabla MOVEMENT que tendrá como una pseudo-llave foránea a DocumentType, DocumentID. ¿Por qué "pseudo"? Porque en verdad no existirá una tabla física, DOCUMENTS, con llave primaria DocumentType, DocumentID a la cual referir. Pero a nivel lógico sí existirá y como mencionaremos luego, los controles de integridad referencial se realizarán.



Cuando especificamos que la transacción tendrá Data Provider asociado para los datos, se habilita la propiedad "Used to" (la "Update Policy" está siempre habilitada). Si especificamos que ese Data Provider será utilizado para **recuperar los datos** (propiedad Used to: Retrieve Data) 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 de las tablas INVOICE y RECEIPT, asociadas a las transacciones de igual nombre.

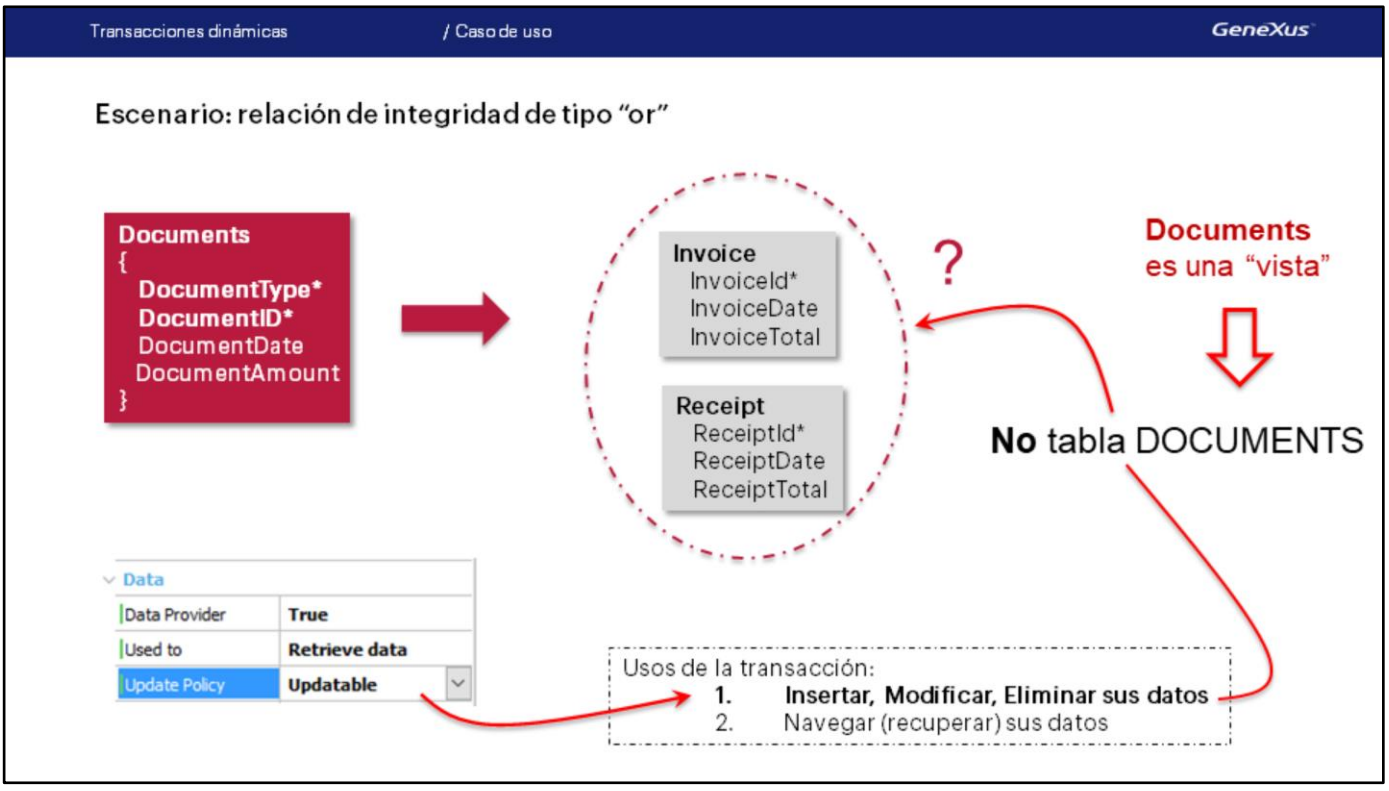


Así declararemos el Source del Data Provider. Tenemos un grupo Documents para devolver todos los documentos que son facturas, y otro para devolver todos los documentos que son recibos.

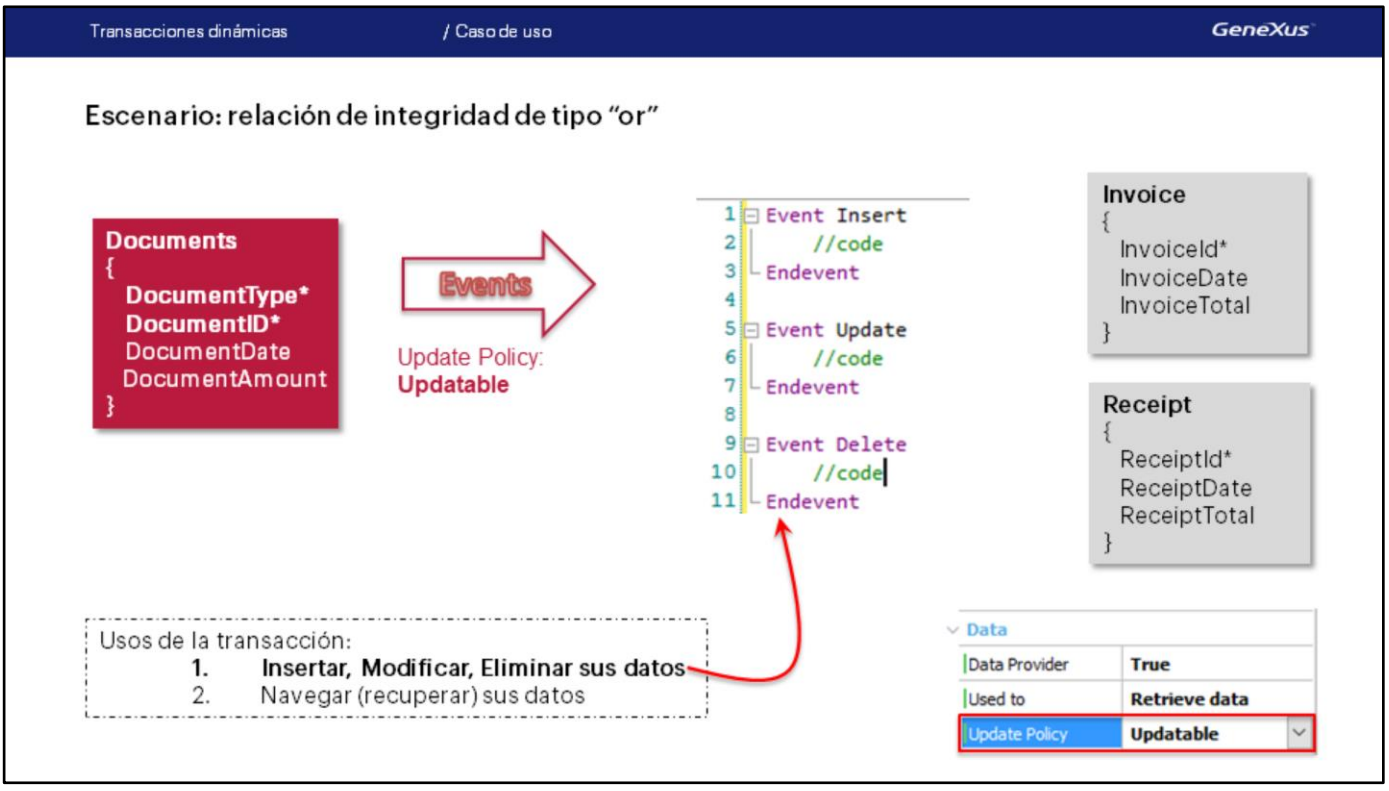


A partir de aquí toda vez que ejecutemos la transacción para navegar por sus datos, se ejecutará este Data Provider que será el que cargará la información apropiada en la pantalla de modo transparente para el desarrollador y para el usuario, quien nunca percibirá que se trata de una transacción sin tabla.

Luego, la transacción dinámica se utiliza como cualquier otra transacción. Por ejemplo si se quieren imprimir todos los documentos ordenados en forma descendente por fecha, se creará un procedimiento con el for each que se indica. En el printblock se colocan los atributos DocumentDate, DocumentType, DocumentId, DocumentAmount como siempre. La forma de obtener esos datos la resuelve GeneXus a partir del Data Provider que contiene su lógica.

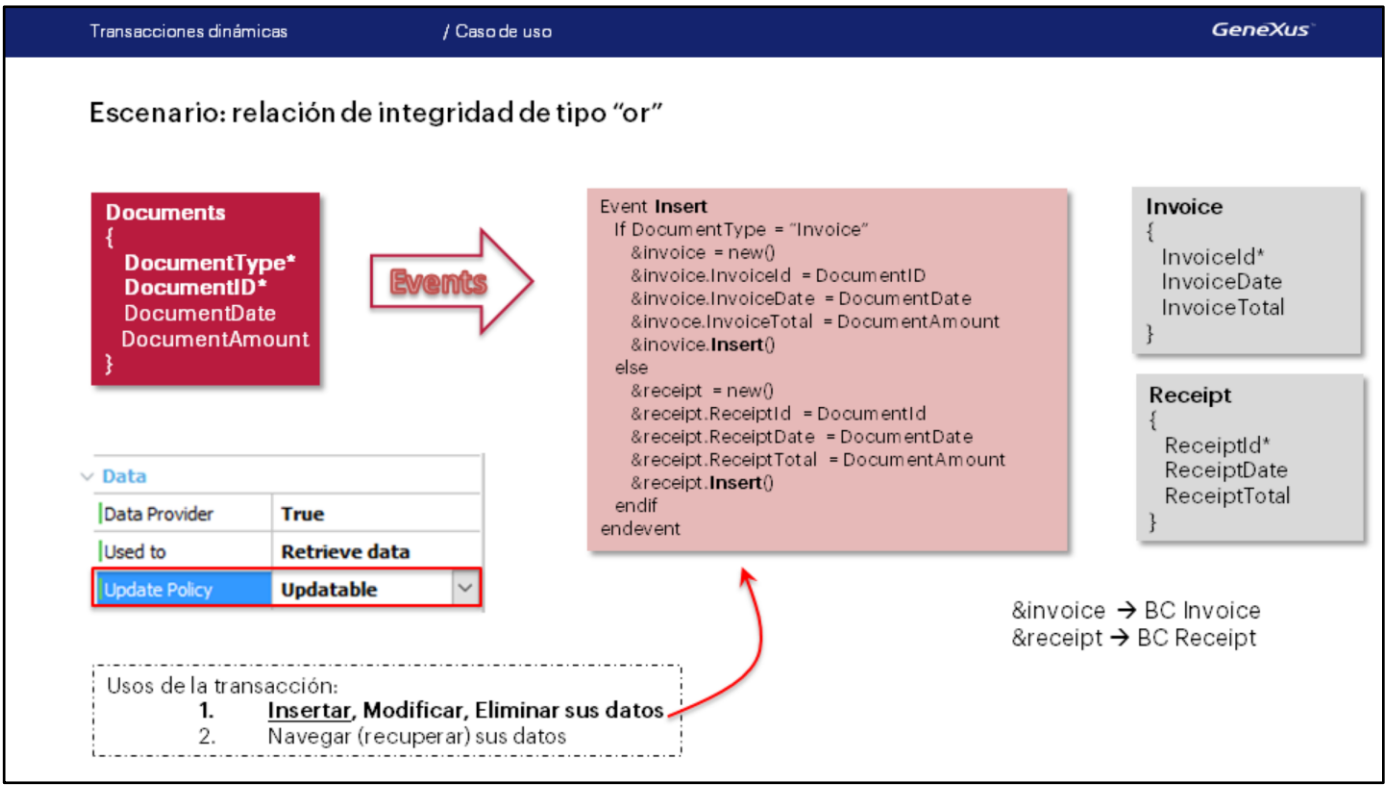


Ahora bien, las transacciones no se utilizan solamente para recuperar sus datos sino también para actualizarlos. ¿Cómo logramos hacer esto, dado que no tenemos tabla asociada a la transacción?



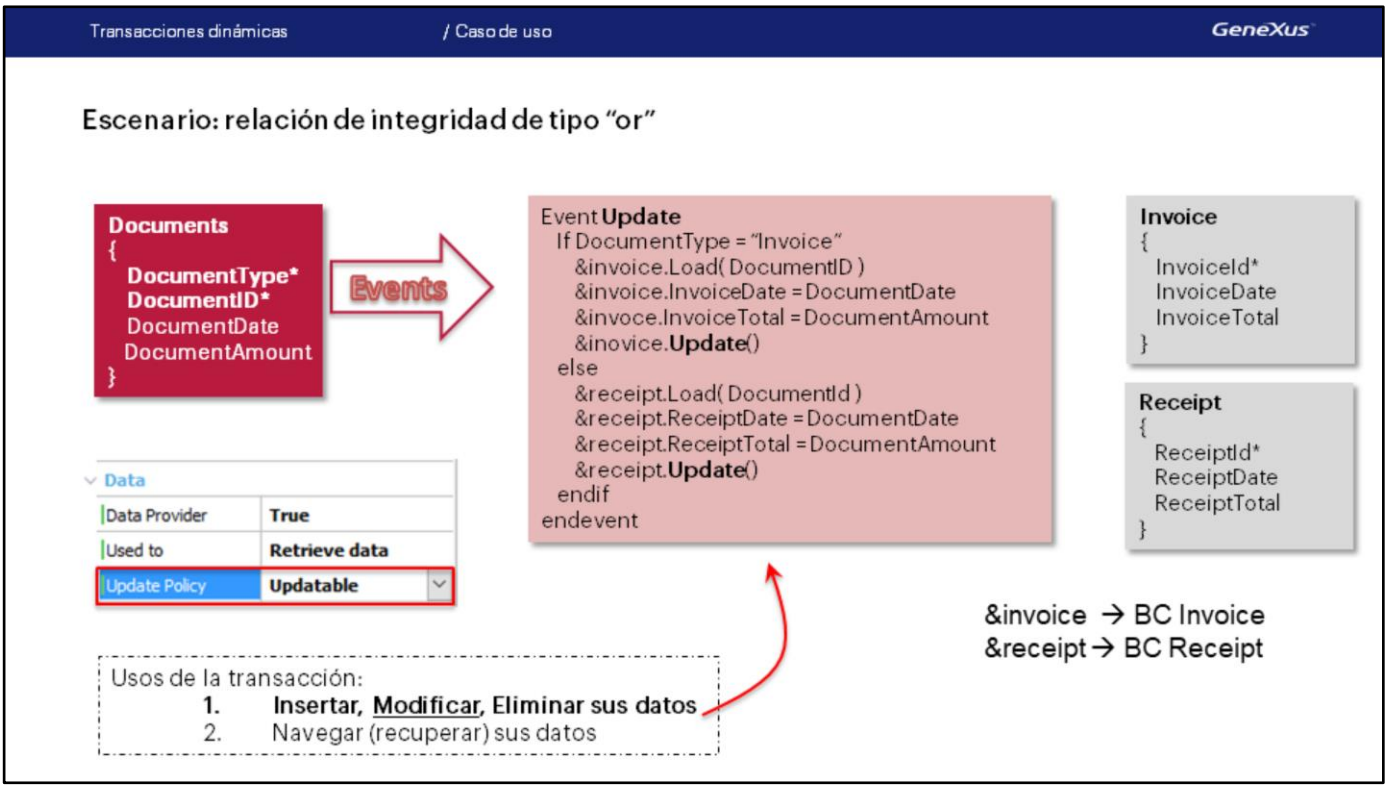
Si la propiedad Update Policy está con el valor "Updatable" se ofrecerán los eventos Insert, Update y Delete para programar cómo insertar, actualizar y eliminar la información que el usuario completó en la pantalla. Sólo el desarrollador sabrá qué debe hacer en cada caso con esa información.

Dependiendo de la realidad se querrá permitir realizar estas acciones o no. Nuestro caso parece ser uno en el que no deberían permitirse. Pero supongamos que sí.

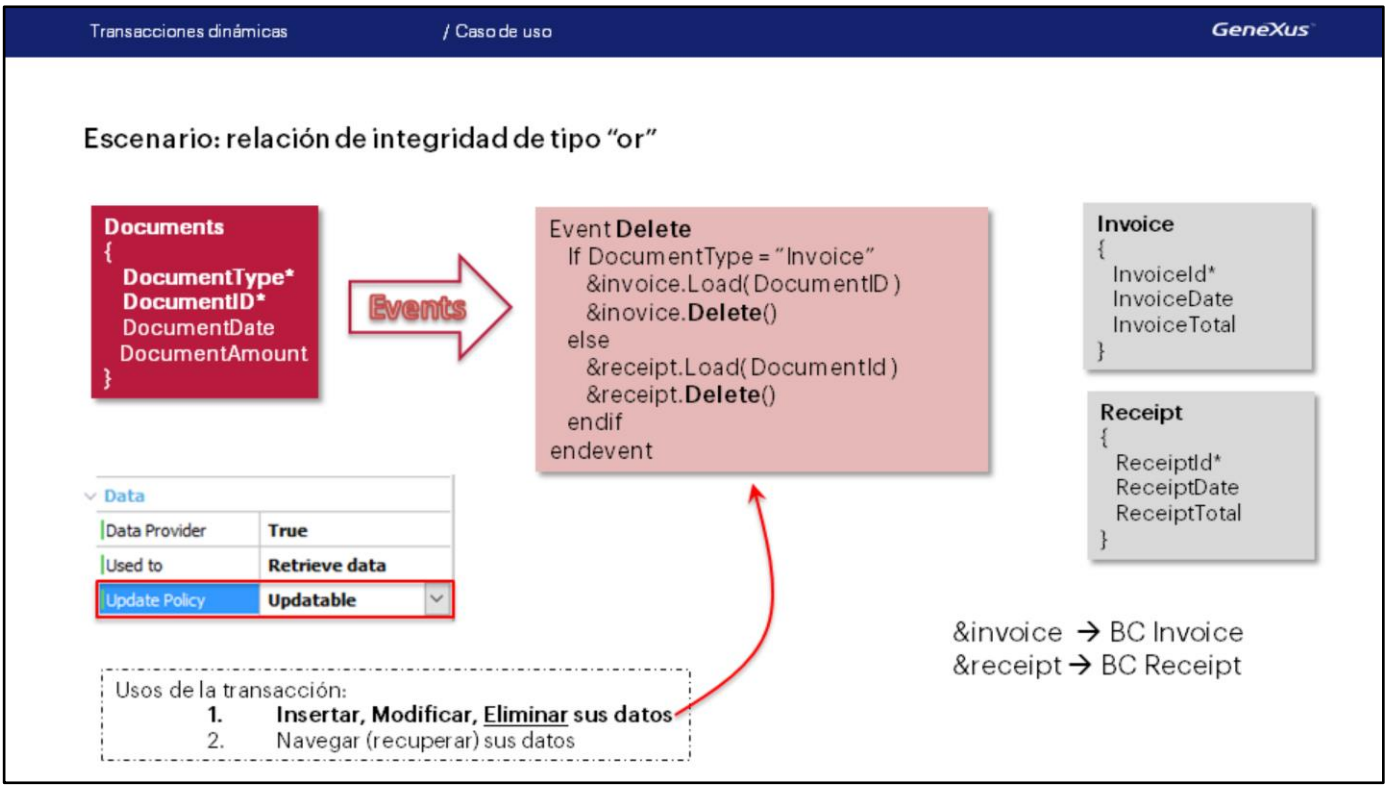


Cuando el usuario haya terminado de llenar los campos de la pantalla para insertar un nuevo movimiento y presione Confirm, tendremos que insertar un nuevo registro en la tabla Invoice o en la Receipt, dependiendo del valor que haya dado el usuario al campo DocumentType. Para ello utilizamos las variables &Invoice y &Receipt de los tipos de datos el Business Component Invoice y Receipt, respectivamente (que tendremos que haber obtenido a partir de las transacciones).

En lugar del método Insert del business component podríamos haber usado el Save. 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, es decir, hará su commit en forma implícita.



Aquí vemos cómo codificaríamos el Update. Como no sabemos qué campos cambió el usuario, asignamos todos los valores.



Y por último el Delete.

¿Reglas? ¿Eventos de disparo?

Rules

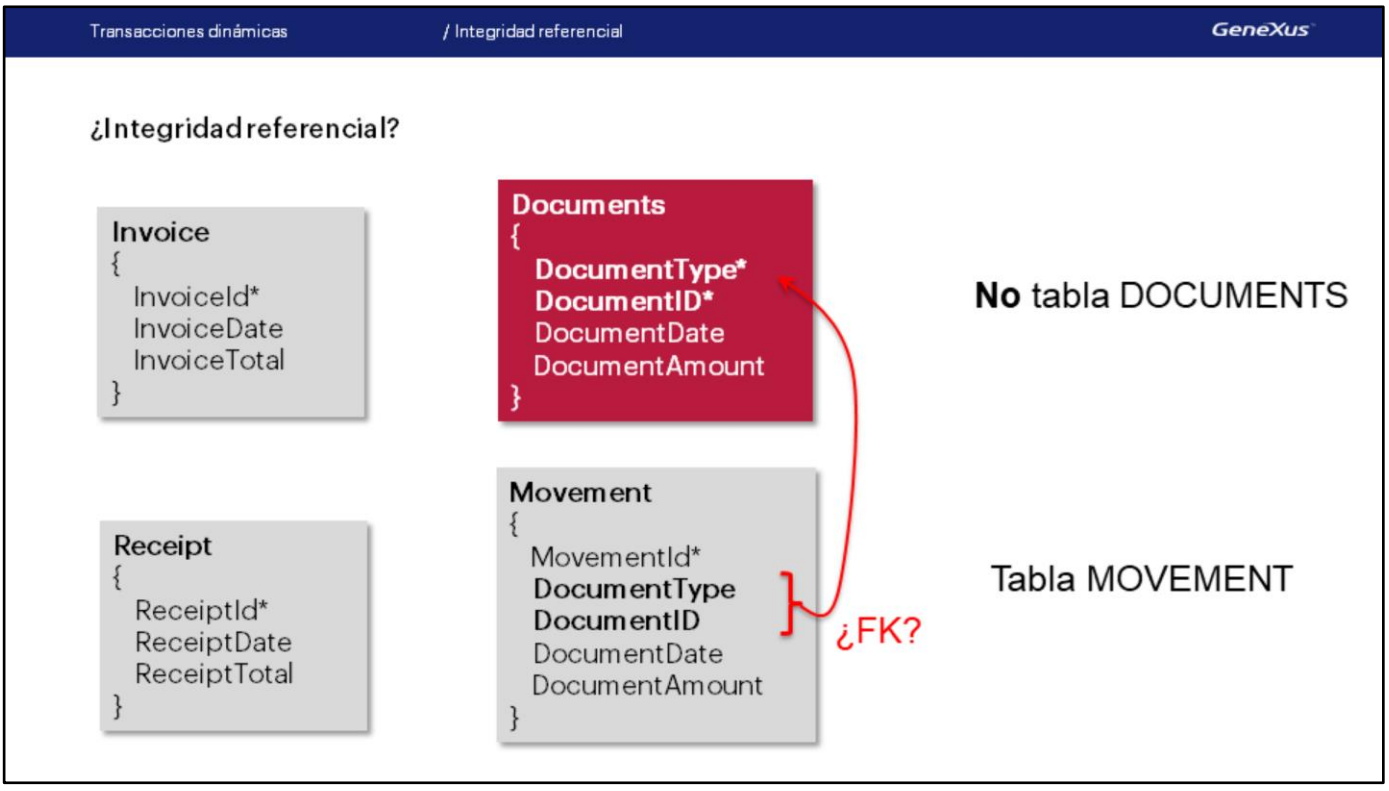
```
Error( "Invalid Document Type")
    if not (DocumentType = "Invoice" or DocumentType ="Receipt");

Default( DocumentDate, &Today );

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

- Se especifican y disparan igual que en una transacción estándar
- Árbol de evaluación y momentos de disparo son idénticos

Update Policy: **Updatable**	Eventos **Insert**, **Update** y **Delete** se ejecutan en el lugar donde en una transacción estándar se realizan las grabaciones

No nos hemos preguntado qué pasa si tenemos reglas especificadas a nivel de la transacción dinámica. ¿En qué momento se van disparando? ¿Qué pasa con el árbol de evaluación?

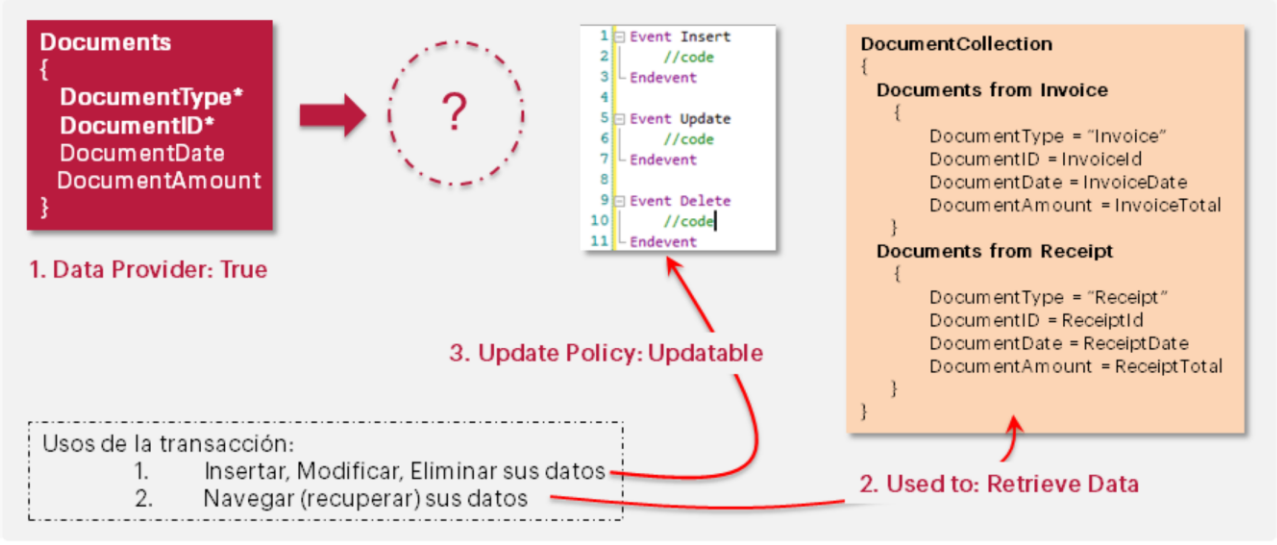
¿Qué pasa con los mensajes de éxito o fracaso de las operaciones Insert, Update, Delete? Del estilo del "Data was successfully added".



Ya que no se creará tabla DOCUMENT asociada a la transacción Document podríamos suponer que entonces en la tabla MOVEMENT asociada a la transacción estándar Movement el par de atributos DocumentType y DocumentID no podrán constituir la llave foránea que deberían. Entonces ¿GeneXus no podrá controlar la integridad referencial?

Como la integridad referencial debe asegurarse, GeneXus genera triggers de SQL para asegurarla. Por tanto podemos decir que {DocumentType, DocumentID} constituyen en Movement una “pseudo” llave foránea. En definivita, no se permitirá eliminar en Document facturas o recibos para los que haya un movimiento, y tampoco se permitirá ingresar un Movimiento que no exista como documento.

Resumen



Aquí vemos un resumen de las propiedades y sus efectos.

Resumen

1. Data Provider: True
2. Used to: Populate Data

DP (para inicializar)

3. Update Policy: Updatable

Permite (o no) las actualizaciones usuales (sobre las tablas de la transacción)

1. Data Provider: True
2. Used to: Retrieve Data

DP (para recuperar)

3. Update Policy: Updatable

Permite (o no) las actualizaciones, pero hay que programarlas en eventos especiales: **Insert**, **Update**, **Delete**

Aquí encontrará más ejemplos de uso de transacciones dinámicas y un desarrollo completo del tema: http://wiki.genexus.com/commwiki/servlet/wiki?28062,Dynamic%20Transactions.

Para el caso de proveedores de datos externos que manejen repositorios de datos con algún álgebra relacional (no tienen por qué ser bases de datos en el sentido de SQL) se utiliza otra solución que es importando el servicio (ej: Odata, CouchDB, otros no SQL). Al hacerlo GeneXus genera automáticamente la transacción y un **Data view**, que es un objeto creado por GeneXus utilizado con el fin de proveer la interfaz de comunicación entre la transacción y la "tabla" externa. Este será otro caso en el que una transacción no crea tabla en la base de datos propia. Se utiliza cuando se hace ingeniería inversa (utilizando DataBase Reverse Engineering Tool).

En este caso, como en el de transacciones dinámicas, el desarrollador utilizará BCs y For eachs como de costumbre, que internamente se traducirán en las invocaciones al servicio externo.

GeneXus™

The power of doing.

Videos	training.genexus.com
Documentation	wiki.genexus.com
Certifications	training.genexus.com/certifications