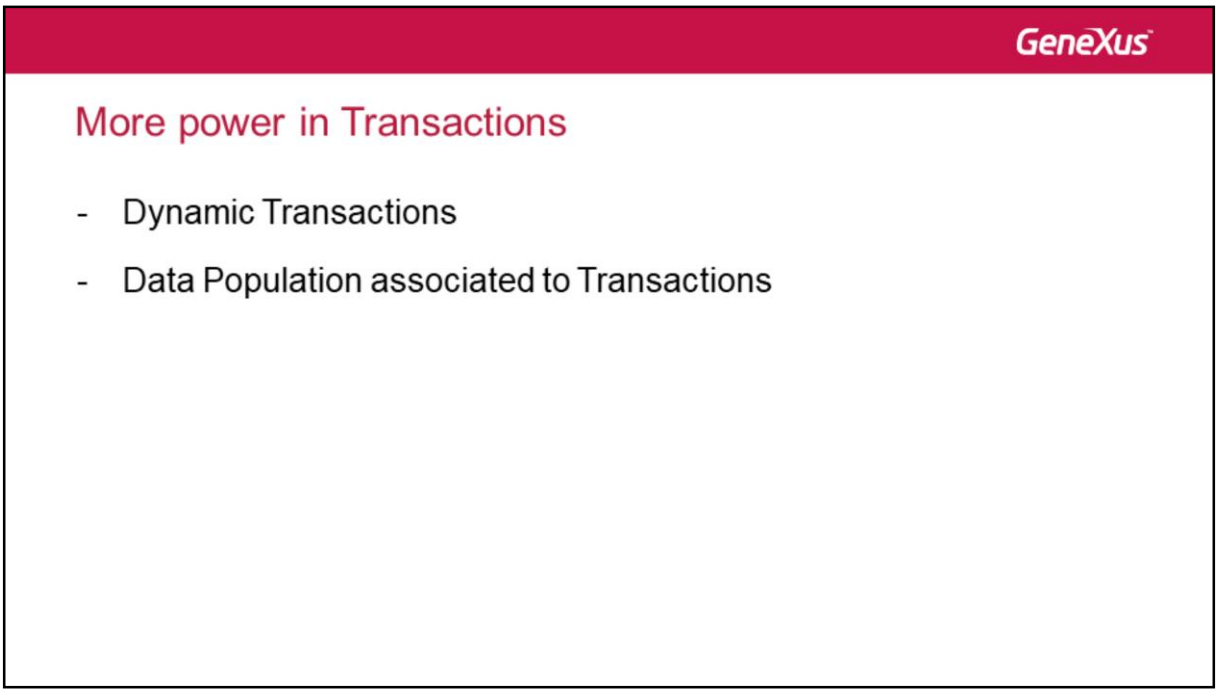
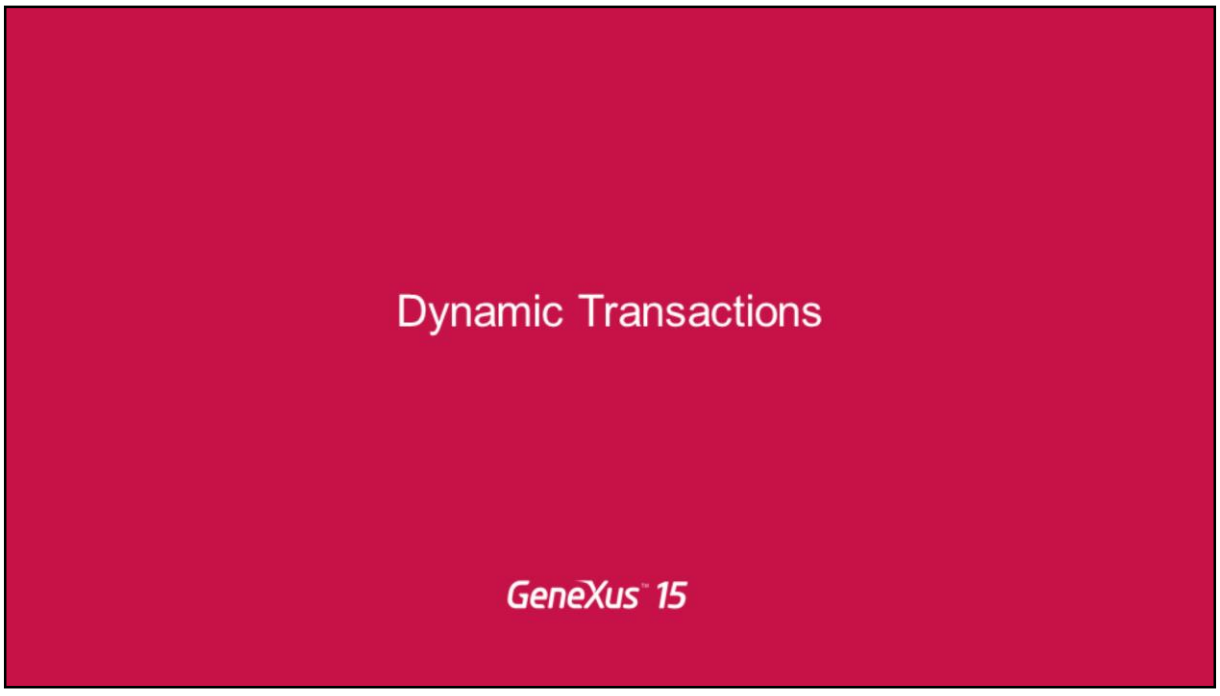
En GeneXus 15 tenemos mas poder en las transacciones

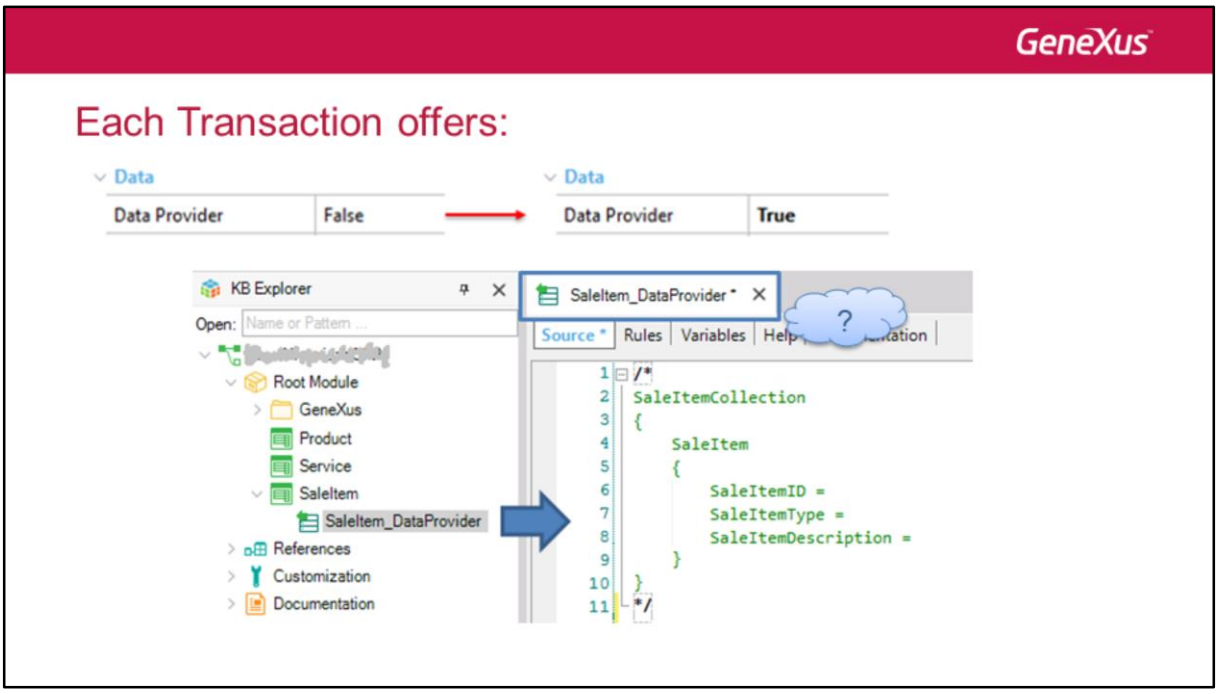


- **DYNAMIC TRANSACTIONS:** Las transacciones dinámicas son un nuevo concepto en GeneXus que ofrece flexibilidad y potencia para lograr diversos objetivos que veremos a continuación

- **DATA POPULATION:** En esta versión tenemos también la posibilidad de definir como queremos que se cargue la información en las tablas físicas asociadas a una transacción (común, no dinámica). Esto nos ahorra la necesidad de tener que definir programas de inicialización de las tablas.

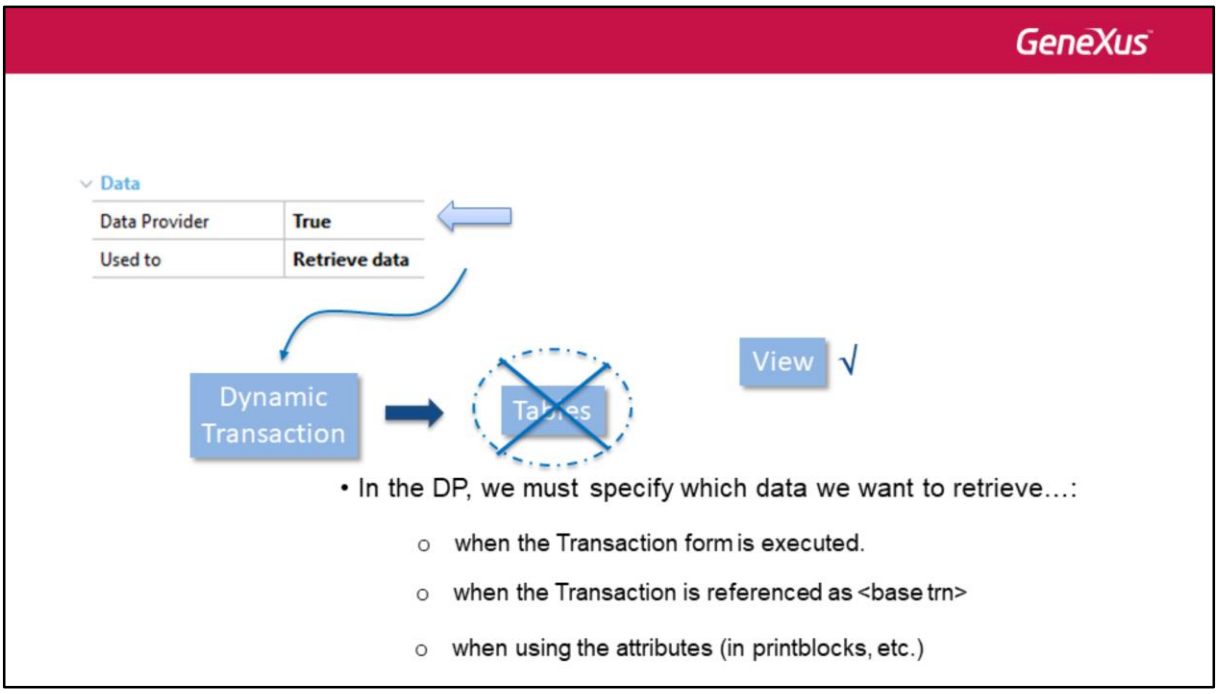


Las transacciones dinámicas son transacciones que no definen tablas físicas sino que ejecutan contra consultas de datos (que serán almacenadas como vistas en el DBMS)



Bajo a propiedad “Data” tenemos una propiedad “Data Provider” que por defecto se define como falso. Mientras esta propiedad esté en falso la transacción se comporta como conocemos hasta ahora.
Cuando esta propiedad se define en verdadero, entonces GeneXus crea un Data Provider – con el nombre <TransactionName>_DataProvider y que inicializa el código tal como vemos en el ejemplo

¿Para qué sirve este Data Provider?



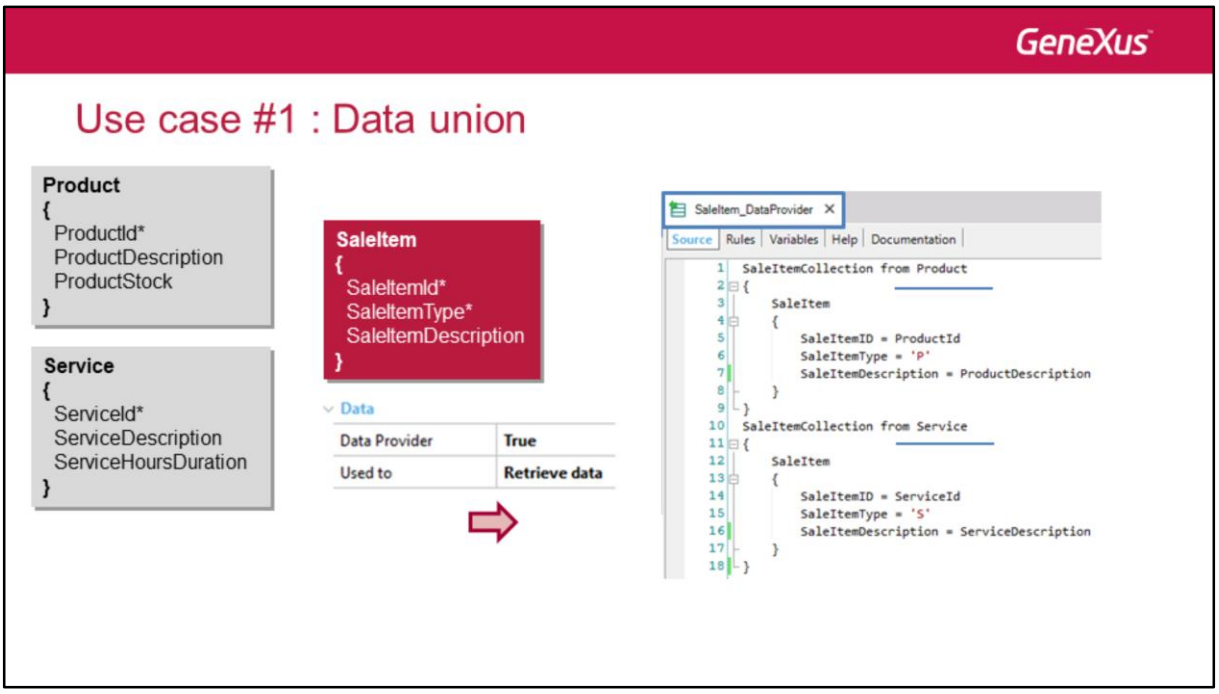
Una vez que fijamos la propiedad “Data Provider” en verdadero, tenemos acceso a una nueva propiedad “Used To”; esta propiedad nos permite definir el objetivo del data provider.

1. **Used-To = Retrieve Data** (recabar datos)

En este caso GeneXus entiende que la transacción es dinámica, lo cual significa que en vez de crear una (o varias) table(s) física(s) asociada(s) a la transacción, va a crear una vista que se almacena en la BD; en el Data Provider vamos a indicar qué datos queremos obtener

- Cuando se ejecuta la transacción
- Cuando se navegan los datos de la transacción como transacción base (i.e. en comandos For Each, Data Providers, Grids en paneles web o SD)
- Cuando se usan sus atributos en print-blocks, condiciones, etc.

Vamos a ver algunos casos de uso.



El primer caso de uso nos muestra una Unión.

Supongamos que una compañía vende productos y ofrece servicios –podría ser una empresa automotora o cualquier tipo de empresa.
Como podemos ver en la pantalla, se definen dos transacciones: Productos y Servicios para capturar la información de cada uno de ellos –como sabemos, hay muchas formas de modelar esta realidad donde una compañía vende productos y servicios, en este caso decidimos hacerlo como dos transacciones independientes ya que queremos capturar información diferente para cada uno.
A la misma vez nos interesa tener un catálogo completo de todo lo que la compañía ofrece.

Para resolver la unión –y poder ordenar todo lo que la empresa ofrece alfabéticamente– definimos la transacción “sale Item” con las siguientes propiedades:

- Data Provider=True
- Used To=Retrieve Data

GeneXus entiende que no debe crear una tabla física asociada a “Sale Item” y crea el Data Provider por defecto (basado en la estructura de la transacción). El próximo paso es completar la carga del DataProvider con la forma que queremos obtener los datos –en este caso navegando las tablas de Productos y Servicios.

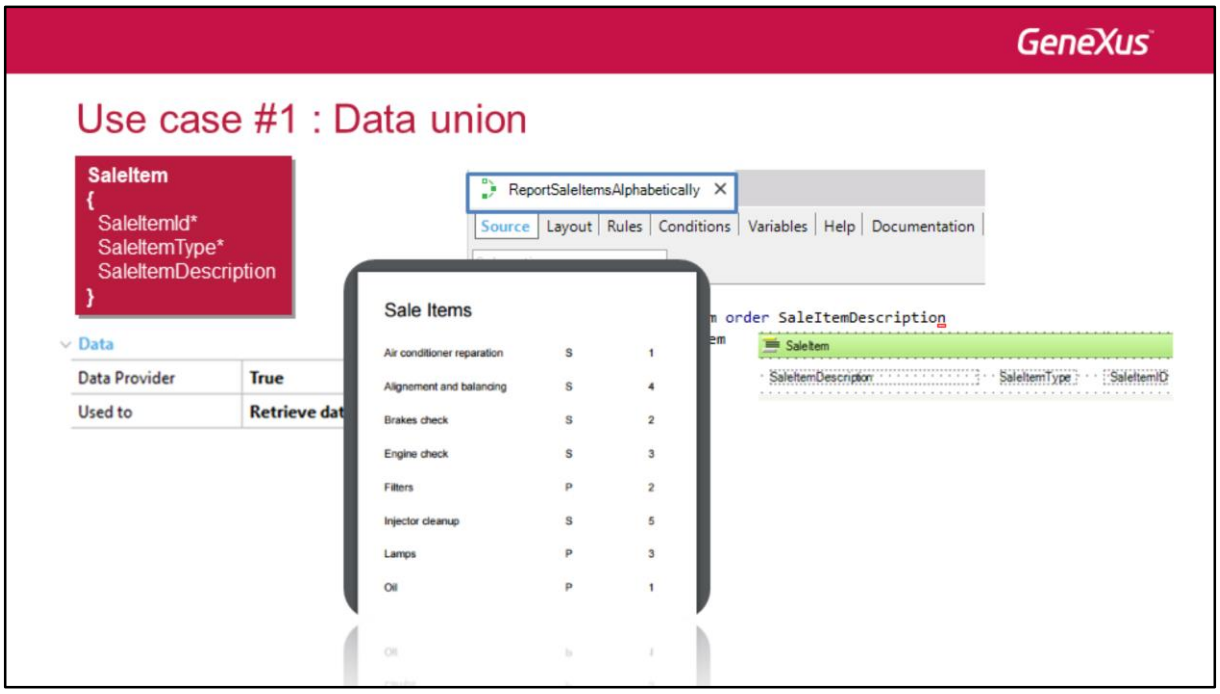
Nótese que la clave de SaleItem es compuesta – se define de esta forma ya que queremos combinar productos y servicios sin que haya conflicto con el identificador de cada uno de ellos, de esta forma podemos asignar a SaleItemType una “P” en caso de productos y “S” para servicios, logrando que la clave sea única.

En el ejemplo del Data Provider vemos cómo se carga primero la información de Productos y luego de los servicios.

Una vez que la transacción está definida, podemos trabajar con la transacción de la forma tradicional, utilizándola como transacción base para otras navegaciones.
Por ejemplo podríamos poner

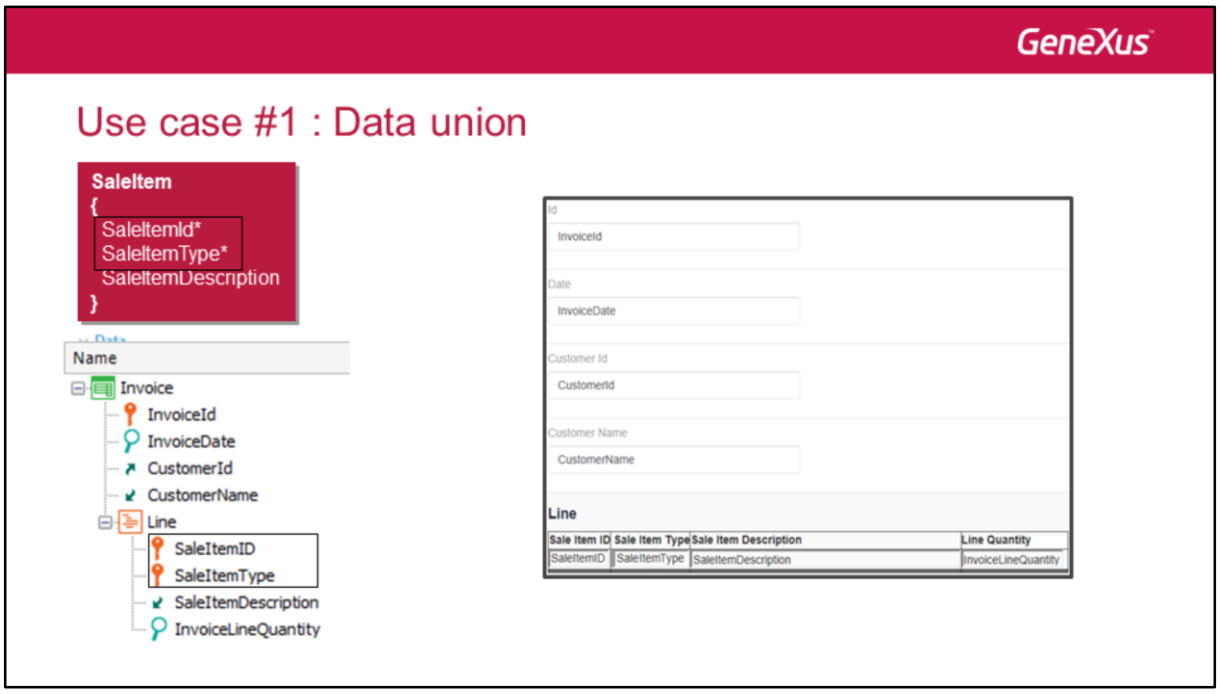
```
for each SaleItem
    order SaleItemDescription
```

```
    //do something
endfor
```



En este caso definimos un listado que muestra los SaleItems de nuestra compañía ordenados alfabéticamente.

Para lograrlo trabajamos de la misma forma que lo hacemos siempre, a pesar de que la transacción sea una transacción dinámica.



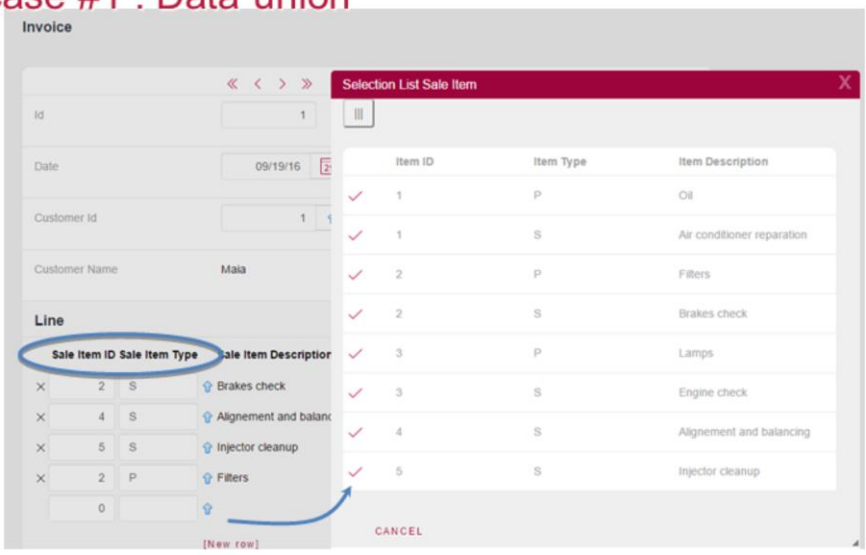
Supongamos también que a la hora de ingresar datos de facturación, la empresa quiere tener una sola lista de artículos –ya sean productos o servicios– en vez de tener que manejar dos listas independientes.

Podemos modelar esto de forma sencilla asignando al subnivel de Invoice los atributos de la transacción SaleItem – es decir, SaleItemID* SaleItemType* pueden ser usados como clave foránea (a pesar de que no refieran a una tabla física sino a una vista).

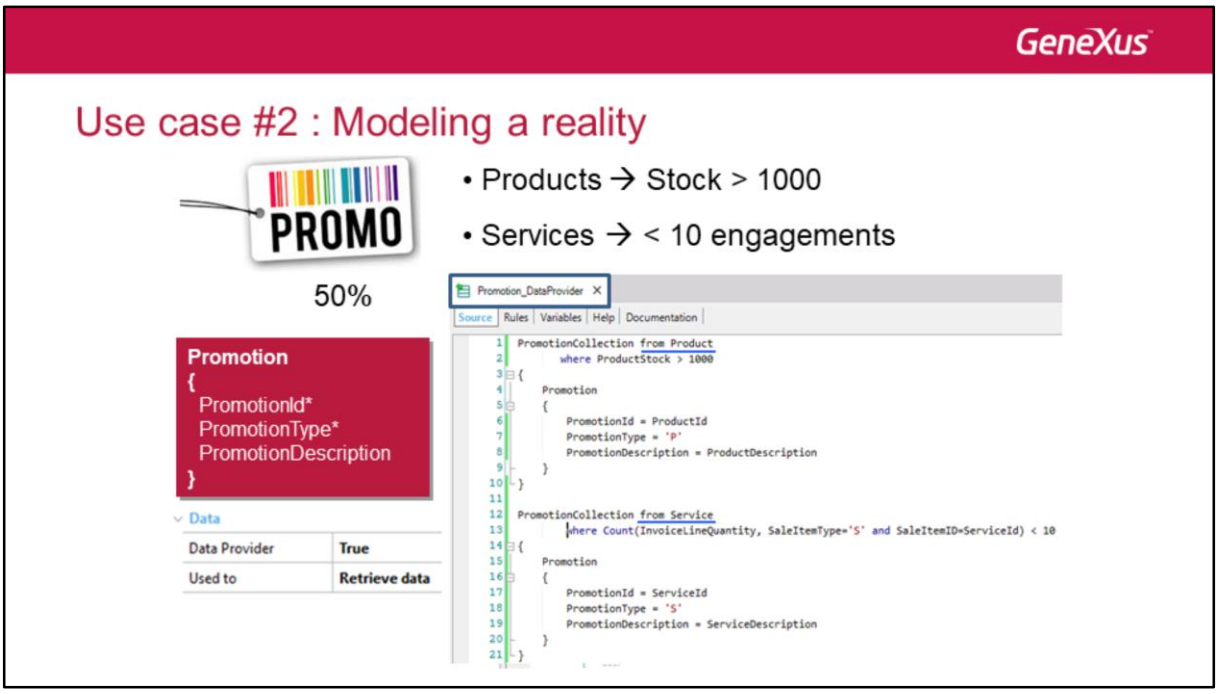
Este ejemplo no está resolviendo el tema de los precios de los productos y servicios – si hubiera información de precios

GeneXus

Use case #1 : Data union



Aquí podemos ver el form de la transacción Invoice en tiempo de ejecución
Note cómo en la grilla hay 4 artículos en la factura: 3 corresponden a servicios y 1 a productos.
También funciona el prompt que permite seleccionar el Sale-Item.



Veamos otro caso de uso.

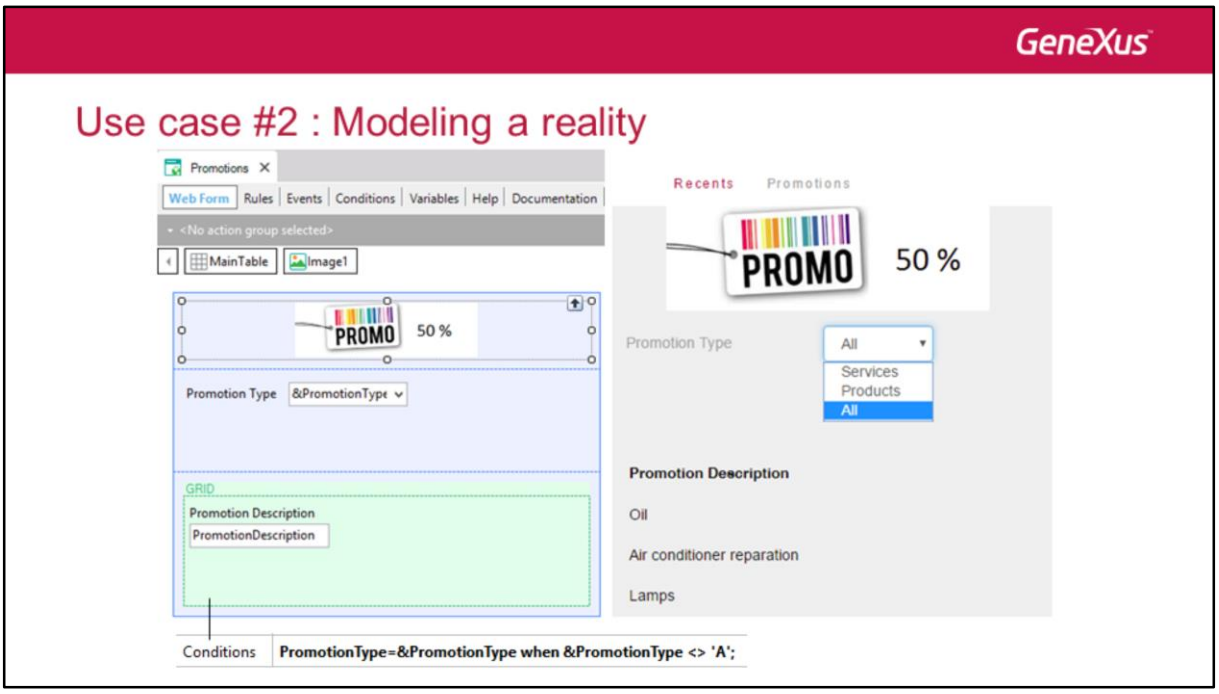
Supongamos que en la automotora, hemos definido promociones de la siguiente manera: se ofrece un 50% off de los productos para los cuales hay más de 1000 unidades de inventario, y para servicios que fueron contratados menos de 1- veces (esto sucede cuando los servicios son nuevos y no han sido muy usados aún, por tanto los primeros 9 clientes que contratan lo harán con un descuento)

Para lograrlo vamos a crear una transacción dinámica llamada “Promotion” donde vamos a tener los productos y servicios que califican para la promoción.

Definimos las propiedades de la transacción:
Data Provider= True
Used To = Retrieve Data

Nótese que la clave en la transacción Promotion es una clave compuesta al igual que en el caso de unión ya que aquí también estamos combinando productos y servicios.

En el Data Provider vemos cómo en el primer bloque obtenemos todos los productos con stock mayor de 1000 y en el segundo obtenemos los servicios que hayan sido facturados menos de 10 veces. Nuevamente una vez definido el Data Provider podemos trabajar con la transacción **Promotion** de forma natural.



Ahora definamos una página web que nos muestra los artículos en promoción.

Para esto definimos en el Web Panel una grilla que muestra el atributo PromotionDescription y que muestra productos y servicios que forman parte de una promoción.

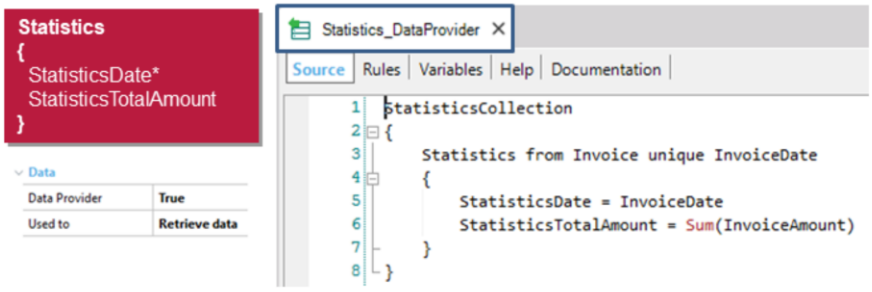
Esta realidad está modelada de forma óptima ya que el articulo (producto o servicio) dejará de estar en la lista una vez que cambie el inventario o la facturación del producto/servicio respectivamente.

Las transacciones dinámicas nos permiten modelar mejor nuestra realidad y simplifican el desarrollo haciéndolo mas consistente. También nos permiten capturar el conocimiento de forma modularizada, es decir, si cambia la forma en que la empresa define sus promociones solo habría que cambiar el Data Provider.

GeneXus

Use case #3 : Solving statistics

- For a given period, to know the amount billed per day.
- For a given date, to know the total amount billed.
- Best day of the year.



El tercer caso de uso tiene que ver con estadísticas. Muchas veces tenemos que resolver consultas del tipo:

- En un período de tiempo, conocer el importe facturado
- En una fecha dada, conocer el importe facturado
- Cuál fue el mejor día de facturación en el año

Es posible programar Data-Providers (o Procedimientos) para resolver cada una de estas consultas, pero la mejor forma es hacerlo en una transacción dinámica. Para ello vamos a definir las propiedades:

•Data Provider=True
•Used To=Retrieve Data

En el ejemplo se define una transacción dinámica **Statistics**

En el data Provider, se recorren las facturas (**Invoices**) agrupando por fecha (**Unique InvoiceDate**) y creando una entrada en la tabla **Statistics** para cada fecha. La estadística que se almacena es el total (sumarizacion) del importe facturado.

GeneXus

Use case #3 : Solving statistics

- For a given period, to know the amount billed per day

```
For each Statistics
    where StatisticsDate >= &InitialDate
    where StatisticsDate <= &FinalDate
          Print Printblock1 {StatisticsDate, StatisticsTotalAmount}
Endfor
```

- For a given date, to know the total amount billed.

```
For each Statistics
    where StatisticsDate = &Date
          Print Printblock1 {StatisticsDate, StatisticsTotalAmount}
Endfor
```

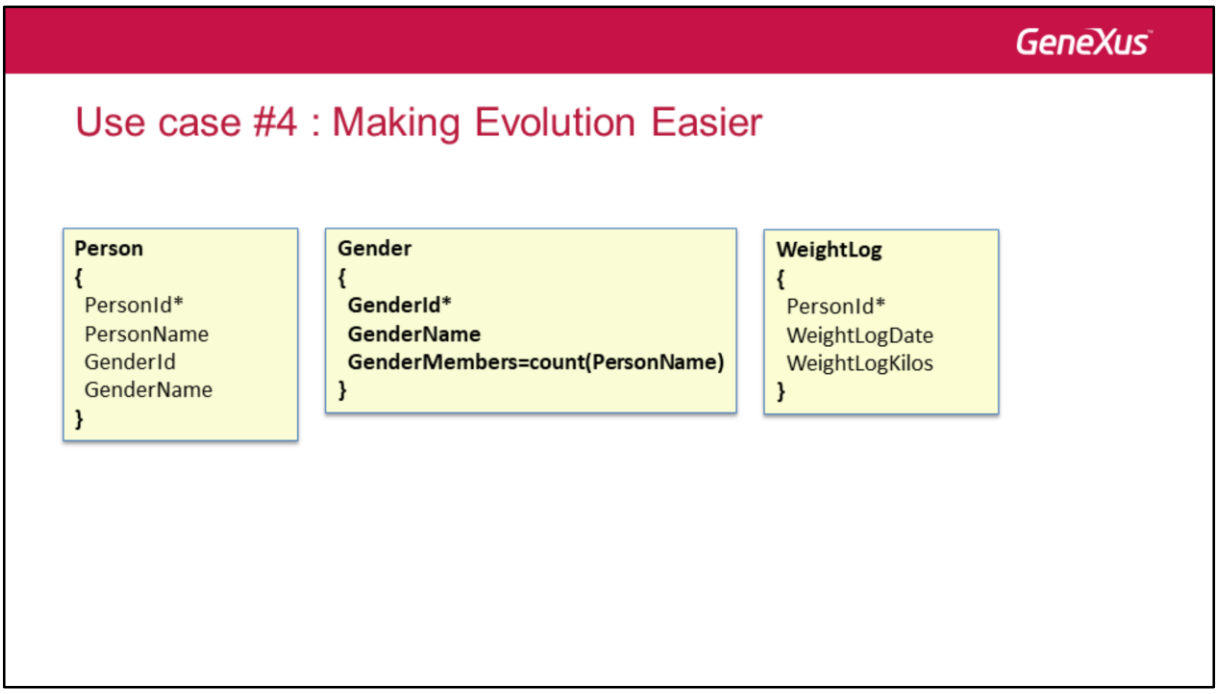
A la hora de trabajar /reportar las estadísticas vamos a navegar la transaction Statistics de forma normal.

Use case #3 : Solving statistics

- Best day of the year.

```
For each Statistics order (StatisticsTotalAmount)
    where StatisticsDate.Year() = &Today.Year()
          Print Printblock1 {StatisticsDate, StatisticsTotalAmount}
          exit
Endfor
```

```
&BestDay = max(StatisticsTotalAmount, StatisticsDate.Year = &Today.Year(),,StatisticsDate)
```



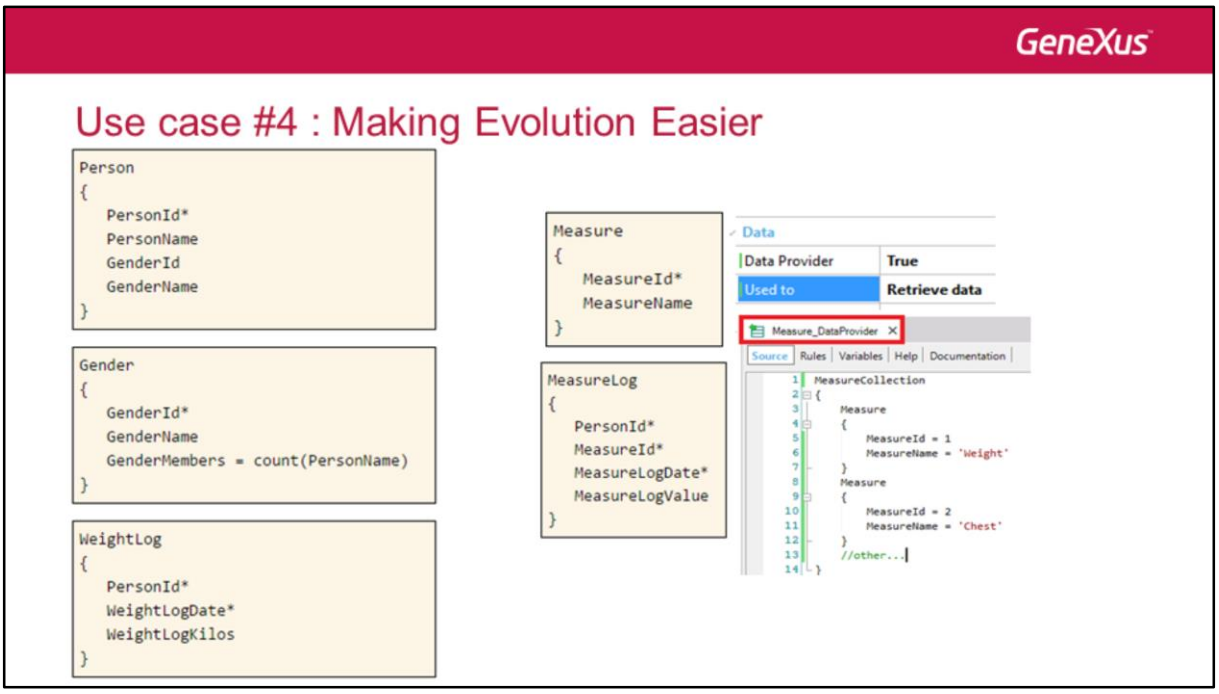
El caso de uso 4 tiene que ver con modelar una realidad de forma de facilitar la evolución.

Por ejemplo, aquí modelamos parte de una aplicación que permite registrar el peso de una persona.

- **Person**: Permite capturar la gene para la cual vamos a medir el peso, su nombre y su genero
- **Gender**: Permite definir los distintos géneros y tiene una fórmula que calcula cuánta gente hay definida para dicho género.
- **WeightLog**: Permite registrar el peso de cada persona en una fecha dada.

Ahora imaginemos que el Sistema se encuentra actualmente en producción y nos piden que registremos no solo el peso sino otras medidas como circunferencia de cintura.

Para esto necesitamos modificar la estructura de nuestras transacciones – También podríamos definir una nueva transacción para cada medida que queramos registrar pero es mejor y más extensible hacerlo en una transacción que capture todas las medidas, como veremos a continuación.



Definimos 2 nuevas transacciones:

Measure – permite definir las diferentes medidas (i.e. Peso, Circunferencia de Cintura, etc)

MeasureLog – permite registrar, para cada persona, el valor de la medida en una fecha dada.

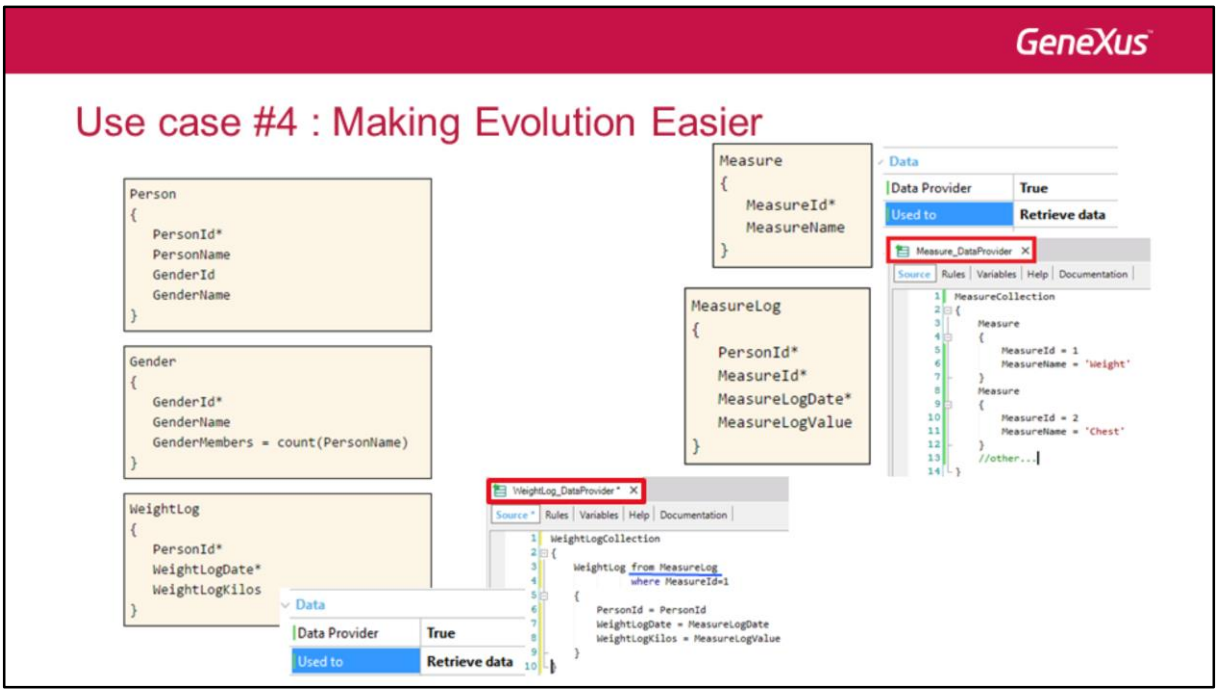
Nótese que la transacción **Measure** fue definida como dinámica como podemos ver en las propiedades del nodo “Data”.

Podemos ver también el Data Provider asignado a **Measure** y vemos cómo los valores no son obtenidos de tablas sino que son valores fijos. Esto es porque el desarrollador no quiere que los usuarios puedan editar/borrar o definir nuevas medidas

La transacción **WeightLog** no se necesitaría más ya que ahora todas las medidas se van a guardar en **MeasureLog** – que es una transacción tradicional con una tabla física asociada. Sin embargo la transacción **WeightLog** y sus atributos son referenciados en muchos lugares de la aplicación.

En vez de borrar la transacción **WeightLog**, una posibilidad es convertirla en una transacción dinámica

Es importante notar que si la transacción se define como dinámica, la información en la tabla física “**WeightLog**” no existiría más. Por tanto antes de realizar esta solución es importante migrar los datos (en este caso de **WeightLog** a **MeasureLog**)

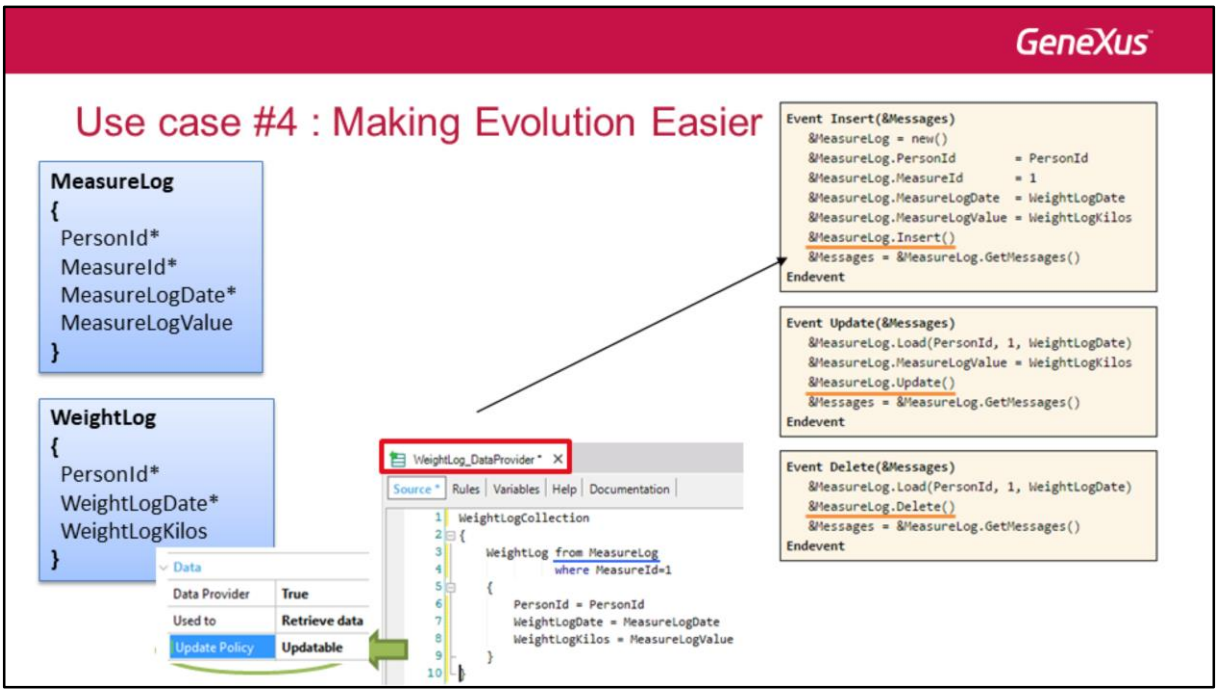


Luego de migrar los datos de **WeightLog** a **MeasureLog** podemos definir **WeightLog** como una transacción dinámica –en este momento **WeightLog** no estará asociada a una tabla física y se carga vía el Data Provider– cargando los valores a partir de **MeasureLog** cuando **MeasureId**=1.

Para esto necesitamos definir las propiedades de la transacción **WeightLog**:

- “Data Provider” property = True
- “Used To” property = Retrieve Data
- Completar el data Provider como se muestra en el ejemplo.

Una vez hecho esto, todo el código que referencia/navega **WeightLog** va a seguir funcionando de la misma manera que antes



Por defecto las transacciones dinámicas son de solo-lectura

Pero imaginemos que – en el escenario anterior – los usuarios estan acostumbrados ya a usar la transacción **WeightLog** y quieren poder continuar utilizándola para cargar el peso y también usar la **MeasureLog** en el resto de los casos.

Para lograr esto tenemos acceso a la propiedad "**Update Policy**" con valores Read Only / Updatable.

Al definir que la transacción **WeightLog** tiene "**Update Policy**" = **Updateable**, el formulario va seguir andando y permitir editar los datos – pero… ¿en que tabla física se guardarán los cambios?

En este caso el desarrollador necesita codificar los eventos Insert, Update y Delete en los eventos de la transacción **WeightLog**. En el ejemplo actual lo lógico es guardar esta información en la tabla **MeasureLog** y lo hacemos usando **BusinessComponents**.

Luego de aplicar los métodos Insert(), Update() y Delete() al Business-Component MeasureLog, obtenemos los mensajes de resultado – Si declaramos la variable &Messages como un parámetro en cada evento (como vemos en el ejemplo) entonces los mensajes son visibles en la transacción dinámica WeightLog de forma transparente como si fueran mensajes normales de una transacción.

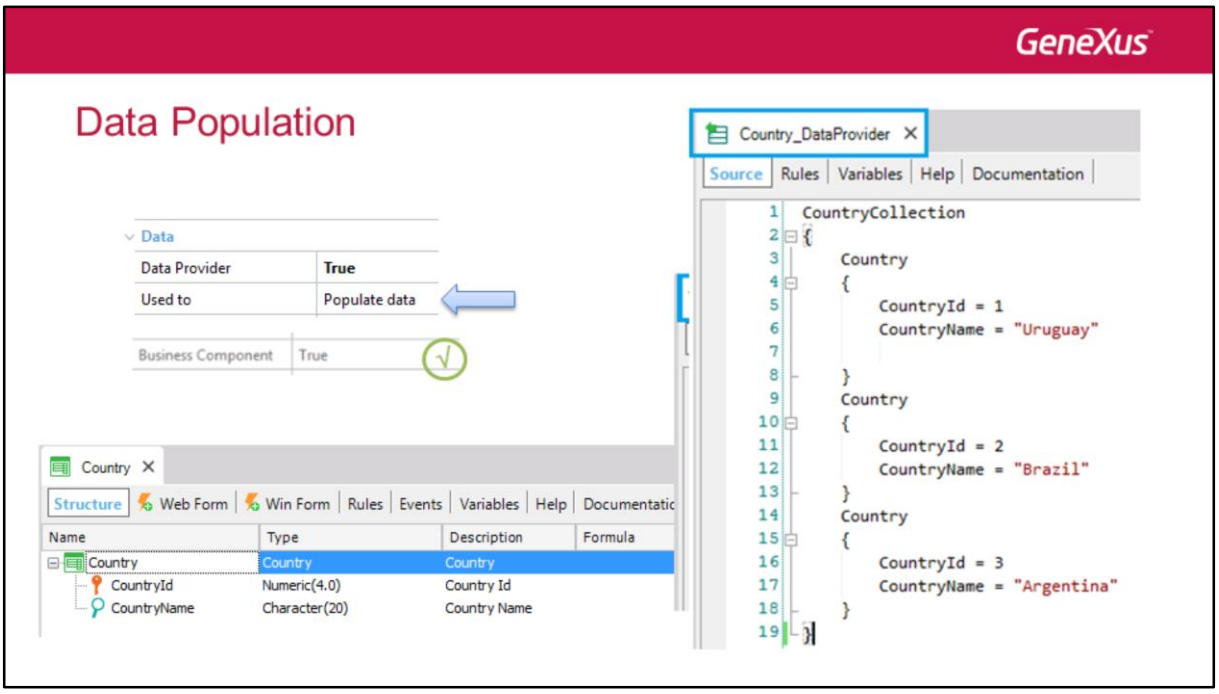
De esta forma, la transaccion Weight Log se puede utilizar de la misma exacta forma que antes, simplemente su implementacion es diferente.

Advantages

- Simplify Programming
- Easier for new developers
- Describe realities and intentions flexibly
- Systems evolve more easily

Data Population associated to Transactions

GeneXus 15



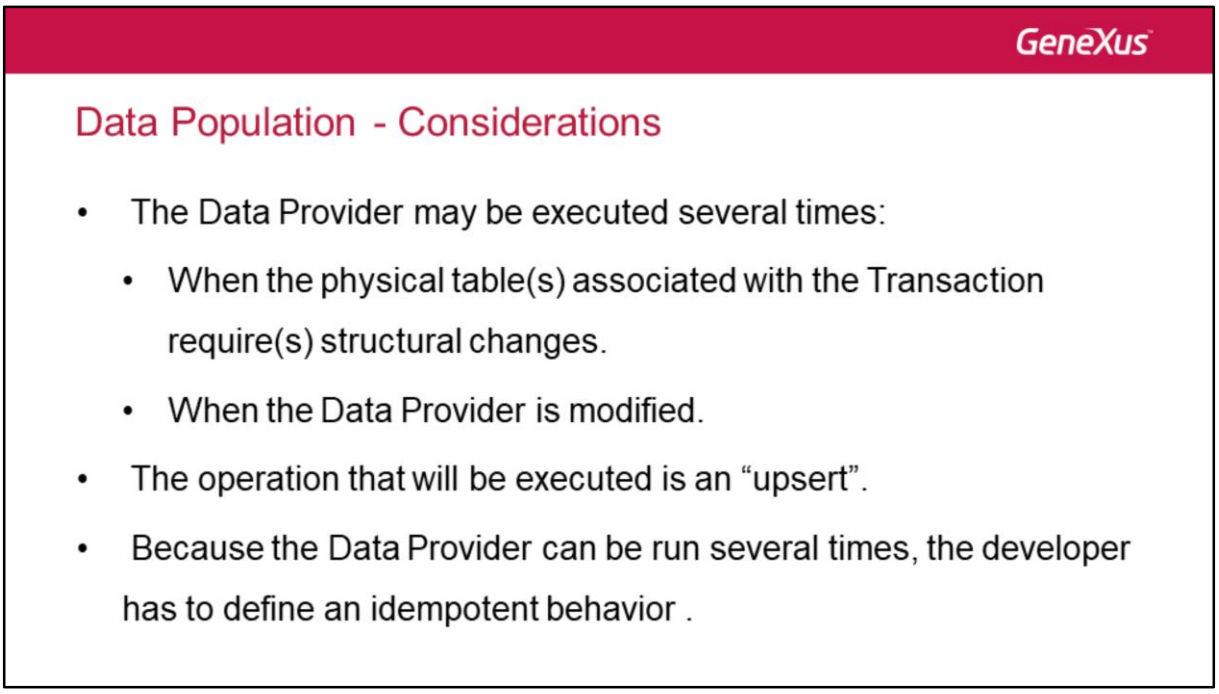
Veamos ahora qué pasa cuando definimos la propiedad “Used To” como “Populate Data”.

Por ejemplo consideremos la transacción Country.

Cuando la propiedad “Used to” es “Populate”, la transacción no se define como dinámica –es decir, vamos a tener una tabla física COUNTRY - sino que es una transacción tradicional pero que cuenta con un Data Provider para inicializar los datos.

Para realizar la inicialización GeneXus va a definir automáticamente la transacción como Business-Component.

La operación que se realiza al inicializar datos es un “upsert”, es decir, primero se va a intentar realizar un insert y si falla por clave duplicada se realiza un update.



Dado que el Data-Provider puede ejecutarse muchas veces el desarrollador debe definir un comportamiento “idempotente” (es decir, no importa la cantidad de veces que ejecute el Data Provider, el resultado debe ser el mismo que si se ejecuta una sola vez)
En el ejemplo que vimos esto se logra porque el desarrollador asignó la clave primaria (CountryId) explícitamente en vez de definirlo como un auto numérico. En caso de definirlo como un auto numérico se debe definir un índice Unique en una clave candidata (ej: CountryName) para evitar almacenar el mismo país múltiples veces.

La ejecución del Data-Provider es independiente del proceso de reorganización. Las tablas se crean/reorganizan y luego después de un build exitoso (F5), si es necesario se ejecuta el Data Provider (“si es necesario” depende de si hubo algún cambio o no en el código de inicialización o si se acaba de crear la tabla)

En la ventana de salida de GeneXus veremos una entrada que nos indica que se está ejecutando la inicialización de datos.

"========== Populate Data started ========== "

GeneXus™

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