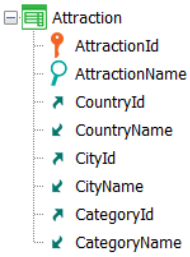
Actualización a la Base de datos con comandos específicos de procedimientos.

Cómo actualizar (update)

GeneXus

Previously: New Command

	Uniqueness check
New command	✓



AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
2	The Great Wall	3	1	2
3	Eiffel Tower	2	1	2
4	Forbidden city	3	1	2

3

New

AttractionId = 3
AttractionName = "Eiffel Tower"
CountryId = 2
CityId = 1
CategoryId = 3

When duplicate

for each Attraction
CategoryId = 3
endfor

endnew

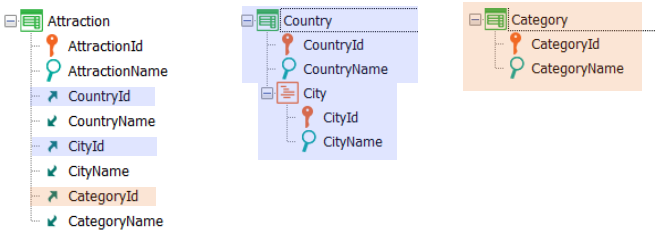
En el video en el que estudiamos el comando New para insertar un registro en una tabla por procedimiento, vimos que si el registro que queríamos insertar se encontraba duplicado por clave primaria o clave candidata, entonces podíamos elegir modificarlo, escribiendo un for each dentro de la cláusula when duplicate.

¿Pero, y si ya sabemos que el registro existe y lo que deseamos es justamente actualizarlo?
En el ejemplo, cambiarle la categoría a la atracción turística 3, Torre Eiffel, que sabemos existe.

Update

¿Existe un comando especial Update en procedimientos?

For each Command



Insert, Update, Delete

AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
2	The Great Wall	3	1	2
3	Eiffel Tower	2	1	2
4	Forbidden city	3	1	2

3

For each Attraction
Where AttractionName = "Eiffel Tower"

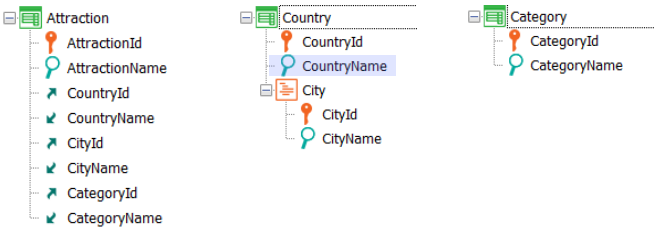
CategoryId = 3

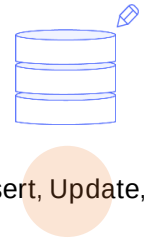
endfor

La respuesta es NO. Para actualizar se utiliza el conocido comando For each. Con esto estamos diciendo que el For each no solo se utiliza para consultar la base de datos, sino también para actualizarla.

En el ejemplo, si queremos cambiar la categoría del registro correspondiente a la Torre Eiffel, entonces alcanzará con escribir este For each. Estamos recorriendo la tabla Attraction, filtrando por AttractionName "Eiffel Tower" y para el registro encontrado, directamente asignándole valor al atributo que queremos modificar. Por supuesto, podríamos modificar el valor de casi todos los atributos del registro. Y no solo de él, sino también de todos los registros relacionados de la tabla extendida. Aquí sí pueden actualizarse muchos registros en una sola operación, a diferencia de lo que ocurría con el new.

For each Command





Insert, Update, Delete

AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
2	The Great Wall	3	1	2
3	Eiffel Tower	2	1	2
4	Forbidden city	3	1	2

CountryId	CountryName
1	Brazil
2	France
3	China

Francia

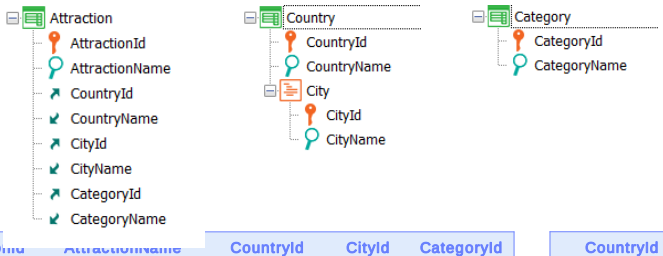
For each Attraction
Where AttractionName = "Eiffel Tower"

CategoryId = 3
CountryName = "Francia"

endfor

Por ejemplo, podríamos modificar el nombre del país (escribiéndolo en español –vemos que en la tabla está en inglés-). Entonces el for each se posiciona en el registro de AttractionName "Eiffel Tower", modifica su atributo CategoryId, y accede al registro del país relacionado, por tabla extendida, y en él modifica su atriburo CountryName, colocando "Francia", en español.

For each Command



Insert, Update, Delete

AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
2	The Great Wall	3	1	2
3	Eiffel Tower	2	1	3
4	Forbidden city	3	1	2

CountryId	CountryName
1	Brazil
2	Francia
3	China

For each Attraction
Where AttractionName = "Eiffel Tower"
AttractionId = 5
CategoryId = 3
CountryName = "Francia"

endfor

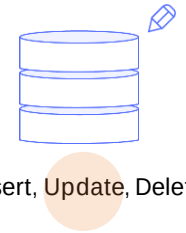
La siguiente pregunta es si todos los atributos de la tabla extendida pueden actualizarse o existen restricciones.

¿Por ejemplo, podemos actualizar atributos de la clave primaria? ¿Podríamos cambiarle el Id de atracción por 5? No, pueden actualizarse todos los atributos del registro y de los relacionados por tabla extendida a excepción de la clave primaria. El listado de navegación nos indicará este error.

Por tanto, si necesitamos cambiarle la clave primaria a un registro no tendremos otra alternativa que crear un registro nuevo con la clave nueva y eliminar el anterior.

For each Command

AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
2	The Great Wall	3	1	2
3	Eiffel Tower	2	1	2
4	Forbidden city	3	1	2
5	Eiffel Tower	2	1	3



Insert, Update, Delete

For each Attraction
Where AttractionName = "Eiffel Tower"

AttractionId = 5
CategoryId = 3

endfor

For each Attraction
Where AttractionName = "Eiffel Tower"

new
AttractionId = 5
CategoryId = 3
endnew

Delete

endfor

Así, por ejemplo, si lo que queremos es cambiarle a la atracción de nombre "Eiffel Tower" el id por 5 y la categoría por 3, ejecutamos un comando new, que queremos que tenga la misma tabla base, Attraction. Allí especificamos el nuevo valor de clave primaria, y para el resto de los atributos, queremos que asuma los mismos valores que los del registro del for each en el que nos encontramos, a excepción de CategoryId, que queríamos aprovechar y cambiarlo por 3.
Luego, lo que haremos será ejecutar el comando Delete, que elimina el registro en el que estábamos posicionados en el for each, es decir, en nuestro caso el de Id 3.

	Uniqueness check	Referential Integrity check	Rules/Events execution
Assignment in For each	✓		

Unique Index

AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
2	The Great Wall	3	1	2
3	Eiffel Tower	2	1	2
4	Forbidden city	3	1	2

```
For each Attraction
      Where AttractionId = 3

       AttractionName = "Louvre Museum"

endfor
```

```
For each Attraction
      Where AttractionId = 3

       AttractionName = "Louvre Museum"

When duplicate
      ....
endfor
```

Al igual que vimos con el caso del comando new, la actualización a través de For each realizará chequeos de unicidad de registros. En este caso solo aplicará cuando haya claves candidatas, es decir, cuando exista algún índice unique definido sobre algún atributo. Imaginemos que es el caso de AttractionName. Y que entonces estamos queriendo modificar en el for each el nombre de la atracción 3, para que pase a ser "Louvre Museum".
Antes de intentar hacer esa actualización, el for each chequeará, utilizando el índice unique, que no exista ya un registro con ese valor. Como en este caso sí existe, entonces no hará nada. Lo mismo que sucedía con el new.

Pero también, como en el new, no hará nada a menos que... hayamos programado cláusula when duplicate. En ese caso, se ejecutará su contenido. Volveremos sobre esto luego.

	Uniqueness check	Referential Integrity check	Rules/Events execution
Assignment in For each	✓	✗	

AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
2	The Great Wall	3	1	2
3	Eiffel Tower	2	1	2
4	Forbidden city	3	1	2

3

Category Table

Excepti on

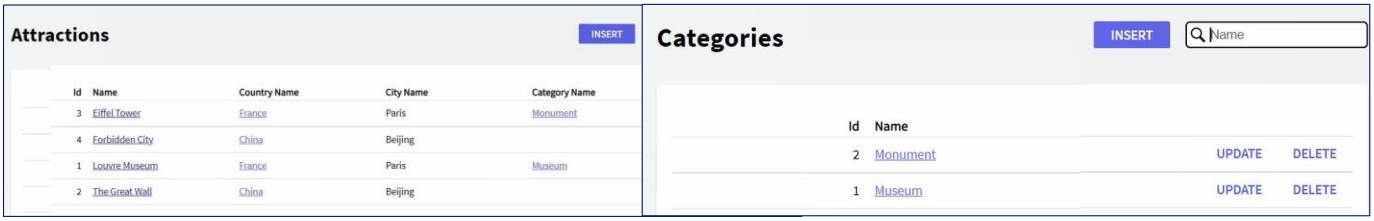
For each Attraction
Where AttractionName = "Eiffel Tower"

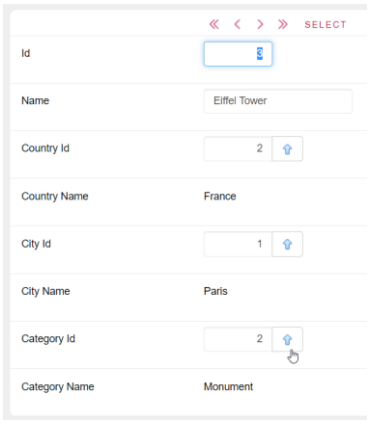
CategoryId = 3

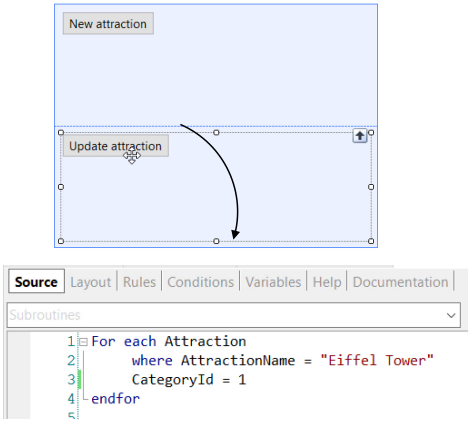
endfor

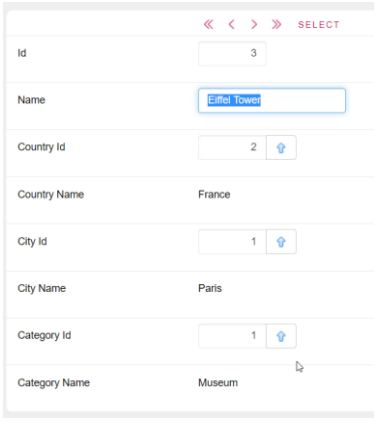
Por otro lado, al igual que vimos con el caso del comando new, la actualización a través de For each no realizará chequeos de integridad referencial, por lo que si la categoría 3 no existe en la tabla Category, el programa no lo controlará. Otra vez, si la base de datos tiene declarada la integridad referencial, ella sí la controlará, por lo que arrojará una excepción y el programa cancelará.

Veámoslo en la práctica.



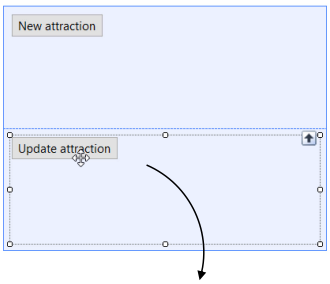


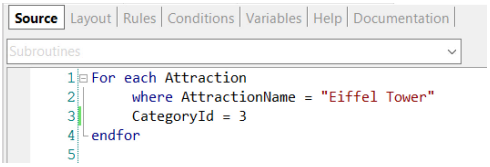


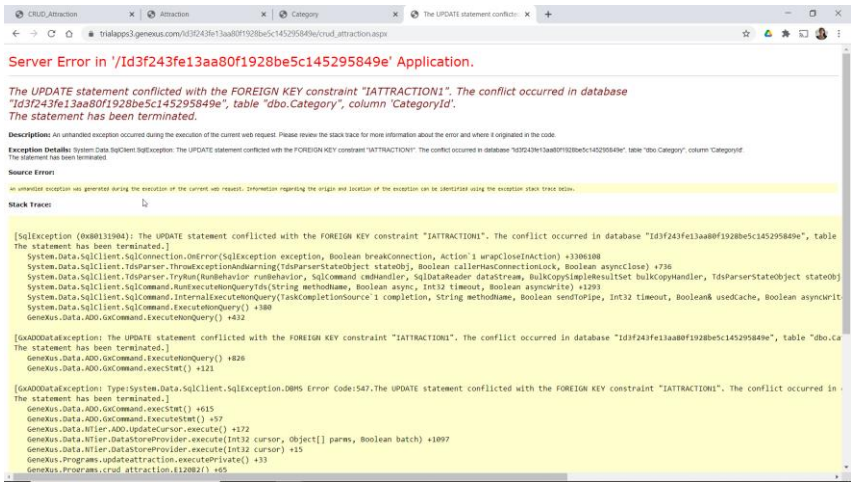


Tenemos las cuatro atracciones, la tercera es la Torre Eiffel, que tiene como categoría, la 2.
Y si vamos a observar los datos de categorías tenemos solamente 2 categorías: la 1 y la 2.

Bien, ahora vamos a GeneXus. Y vemos que tenemos este web panel, en el que hemos programado, en el evento asociado al botón, la invocación al procedimiento UpdateAttraction, en el que tenemos este for each, que lo que intenta hacer es cambiar la categoría de la Torre Eiffel por la 1, que sabemos que existe.







Ahora observemos qué sucede si en lugar de asignarle la categoría 1 que sabemos que existe, le asignamos la categoría 3, que no existe. Probemos.

El programa se interrumpió debido a que el for each no hizo control de integridad, e intentó realizar la actualización, pero la base de datos sí realizó el control. Por lo que arrojó esta excepción que no fue capturada por nuestro procedimiento como para hacer algo con ella. Esto se puede hacer y se nombrará en otro video.

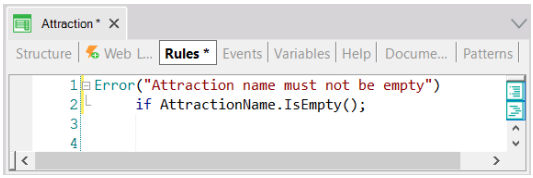
	Uniqueness check	Referential Integrity check	Rules/Events execution
Assignment in For each	✓	✗	✗

AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
2	The Great Wall	3	1	2
3		2	1	2
4	Forbidden city	3	1	2

For each Attraction
Where AttractionId = 3

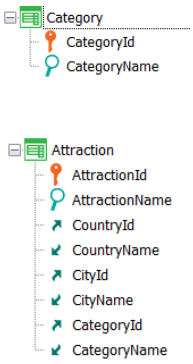
AttractionName = ""

endfor



Por supuesto, igual que como dijimos para el caso del new, la actualización por asignación dentro del for each de un procedimiento tampoco ejecutará absolutamente ninguna regla o evento de transacción. A diferencia de lo que ocurre cuando se actualiza a través de Business Component.

For each Command



CategoryId	CategoryName
1	Museum
2	Monument
3	Tourist Site

AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
2	The Great Wall	3	1	3
3	Eiffel Tower	2	1	2
4	Forbidden city	3	1	3
5	Christ the Redemmer	1	2	2

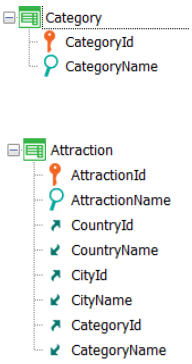
```
New
    CategoryName = "Tourist Site"
endnew

For each Attraction
    Where CountryName = "China" and CityName = "Beijing"
    Where CategoryName = "Monument"
    CategoryId = find( CategoryId, CategoryName = "Tourist
    COMMIT
endfor
```

Aquí vemos un ejemplo donde insertamos una categoría nueva, Tourist Site, en la tabla Category, e inmediatamente recorremos con un for each las atracciones turísticas del país China y ciudad Beijing, de categoría "Monument", cambiándoles esa categoría por la nueva, "Tourist Site".

Otra vez, este código solo en válido en el Source de un procedimiento. ¿Y qué pasa con el Commit? ¿Cuándo se commitea el registro insertado en Category y los dos registros modificados en Attraction?

Si no escribimos comando Commit en forma explícita en el Source, entonces, si tampoco modificamos el valor por defecto de la propiedad Commit on Exit del procedimiento, entonces GeneXus agregará un Commit al final. Esto porque ya habrá descubierto que en el Source se está queriendo actualizar la base de datos. En un procedimiento donde GeneXus no encuentra que la base de datos vaya a ser actualizada, no lo agrega, aunque la propiedad Commit on Exit esté en Yes.

For each Command

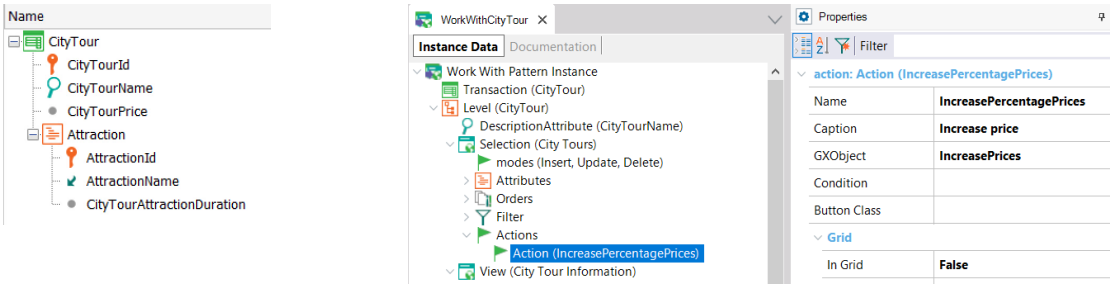


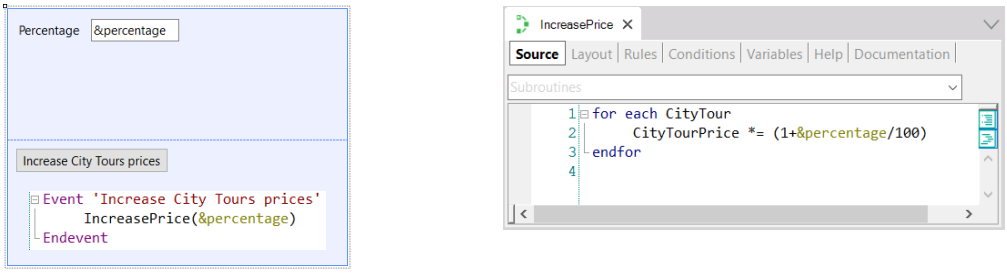
CategoryId	CategoryName
1	Museum
2	Monument
3	Tourist Site

AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
2	The Great Wall	3	1	3
3	Eiffel Tower	2	1	2
4	Forbidden city	3	1	3
5	Christ the Redemmer	1	2	2

```
New
    CategoryName = "Tourist Site"
Endnew
Commit

For each Attraction
    Where CountryName = "China" and CityName = "Beijing"
    Where CategoryName = "Monument"
    CategoryId = find( CategoryId, CategoryName = "Tourist Site" )
Endfor
Commit
```

Por supuesto, si quisiéramos que se realizara un Commit después de insertar la categoría, y luego después de modificar las atracciones, alcanzará con explicitarlo.





Hemos agregado una transacción CityTour para representar los tours por ciudades que se ofrecen al público. Cada city tour realizará un recorrido por un conjunto de atracciones turísticas, y tendrá un precio dado.
Le hemos aplicado el pattern work with y hemos agregado una acción que llamará a un web panel que hemos creado, que le pide al usuario un porcentaje, y llama a un procedimiento que lo que hará será incrementar el precio de todos los city tours en base a ese porcentaje.

Para ello lo que hacemos es recorrer la tabla de CityTour, y al atributo CityTourPrice asignarle como nuevo valor el que tenía por este factor de aquí, que es el que aplica el incremento. También podríamos haber utilizado directamente el operador "por igual" para no repetir el atributo.

Observemos que el listado de navegación nos informa qué atributo es el que está actualizando de la tabla CityTour.

Si ahora ejecutamos... vemos que tenemos dos city tours ingresados: uno de valor 300, otro de valor 200, y lo que vamos a hacer es ejecutar entonces ese web panel, vamos a decirle que queremos que aumente un 10%... Y ahora vemos cómo efectivamente lo ha hecho.

Summary

For each *BaseTransaction*
skip $expression_1$ **count** $expression_2$
order att_{11}, att_{12}, ... att_{1n} **[when** *condition***]**
order att_{21}, att_{22}, ... att_{2n} **[when** *condition* | **otherwise]**
using *DataSelector*(*parm*$_1$, ..., *parm*$_n$)
unique att_1, ..., att_n
where *condition* **[when** *condition***]**
where *condition* **[when** *condition***]**
where *att* **in** *DataSelector*(*parm*$_1$,..., *parm*$_n$)
blocking *NumericExpression*
...
$Attribute_1$ = $expression_1$
$Attribute_2$ = $expression_2$
...
$Attribute_N$ = $expression_N$
...
When duplicate
...
When none ...
endfor

	Uniqueness check	Referential Integrity check
Assignment	✓	✗

COMMIT

Transaction integrity	
Commit on exit	Yes

En suma, para actualizar registros específicamente por procedimiento contamos con el comando For each.

En su cuerpo, además de poder hacerse otras cosas, pueden actualizarse tanto atributos de la tabla base como de la tabla extendida. La única restricción es que la clave primaria de la tabla base del for each no puede actualizarse.

Por otro lado, habíamos visto que el único control programático que se realiza es el control de unicidad. Para el caso del for each, se controlará que ninguna clave candidata se repita. Si el for each encuentra que si realiza la asignación eso repetiría esa clave, entonces no hace nada, a menos que se programe la cláusula when duplicate.
En el código de esta cláusula bien puede programarse un new para insertar un registro nuevo, por ejemplo.

Sabemos que el for each no hace control de integridad referencial, por lo que intentará realizar la actualización que se haya especificado sin atender la existencia o no del registro referenciado. Esto es por motivos de performance. Pero las bases de datos en general sí realizan el chequeo, a menos que apaguemos esa funcionalidad, por lo que, de no apagarla, y fallar la integridad, arrojarán una excepción.

Por último: para que el registro quede commiteado en la base de

datos debemos asegurarnos de que el comando Commit se ejecute. En un procedimiento, por defecto, se coloca un Commit implícito al final (siempre y cuando se entienda que en el Source se está accediendo en algún lado a la base de datos para actualizarla). Pero podemos escribir explícitamente Commits en el Source, donde nos convenga.

No lo veremos aquí, pero opcionalmente puede especificarse una cláusula Blocking, que lo que hace es permitir hacer actualizaciones en bloque, en lugar de registro a registro. Es decir, procesará los registros en bloques de a N para reducir los accesos y mejorar la performance.

Por último, algo que no hemos dicho hasta el momento es que las redundancias no son mantenidas automáticamente cuando se hacen actualizaciones por procedimiento. Es responsabilidad del desarrollador hacerlo. Esto significa que si una redundancia depende de un atributo que está siendo actualizado, GeneXus no buscará o calculará el nuevo valor para almacenar en el atributo redundante. Tiene que hacerlo el desarrollador.

Sobre todo esto podemos encontrar más información en nuestro wiki.

GeneXus™

training.genexus.com
wiki.genexus.com