Actualización de la base de datos

Business components vs Comandos específicos de procedimientos

GeneXus™

1. **Business Compontent**: Insert(), Update(), Delete(), etc.

Insert, Update, Delete

2. **Procedure**: New, Assignment in For each, Delete

Para actualizar la base de datos por código teníamos dos posibilidades:

Hacerlo utilizando el business component de la transacción, a través de sus métodos, o hacerlo exclusivamente dentro de un procedimiento, a través de los comandos New, For each con asignación directa de los atributos a ser modificados, y el comando Delete.

	Uniqueness Check	Referential Integrity Check	Rules/Events execution
Business Component	✓	✓	
New, Assignment in For each, 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

```
&attraction.Load(3)
&attraction.CategoryId = 100
&attraction.Update()
```

&attraction.GetMessages() →

Name	Type	Description	Is Collection
Messages		Messages	✓
Message			
Id	VarChar(128)	Id	
Type	MessageTypes, GeneXus	Type	
Description	VarChar(256)	Description	

Como vimos, con ambas opciones se controla la unicidad de registros, es decir, nunca se permite dejar en la base de datos registros con clave primaria o candidata repetida. En el ejemplo, nunca se permitirán dos atracciones con el mismo identificador. Y si, por ejemplo, AttractionName fuera clave candidata, tampoco se permitirían dos atracciones con el mismo nombre.
Esto es chequeado automáticamente por el Business Component o por el comando en el procedimiento antes de intentar la operación sobre la Base de Datos.

Hasta ahí lo que es igual. Ahora veamos las diferencias.

En lo que hace al aseguramiento de la integridad referencial, la inserción, actualización o eliminación vía Business Component chequeará antes de realizar la operación que la integridad referencial no se viole, tal como lo hace la transacción. Por ejemplo, si quisiéramos modificar la categoría de la atracción 3 por una inexistente, el Business component controlará previamente que exista la categoría 100 en la tabla Category y si no existe, entonces **no intentará** la actualización. Y además nos cargará un mensaje de error indicándolo en la colección de mensajes que nos devuelve si se lo pedimos.

	Uniqueness Check	Referential Integrity Check	Rules/Events execution
Business Component	✓	✓	
New, Assignment in For each, 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

```
For each Attraction
Where AttractionId = 3
    CategoryId = 100
endfor
```

Exception!

En cambio, las operaciones realizadas por los comandos específicos en procedimientos no realizarán control de integridad referencial alguno. Esto significa que si, por ejemplo, queremos actualizar la categoría de la atracción 3, ahora por asignación dentro de For each, no será consultada la tabla Category para averiguar si existe una categoría 100 antes de enviar la orden a la Base de Datos para que actualice ese registro. Por tanto, si la Base de Datos tampoco controla la integridad referencial, nos quedará un dato inconsistente. Y si sí la controla, como es el comportamiento por defecto, arrojará una excepción y el programa que estamos ejecutando cancelará.

	Uniqueness Check	Referential Integrity Check	Rules/Events execution
Business Component	✓	✓	
New, Assignment in For each, Delete	✓	✗	

Por tanto, por el momento la balanza parece inclinarse hacia el uso de Business Components.

Referential Integrity Check
✓
✗

```
For each Attraction
    &attraction.Load(AttractionId)
    &attraction.CategoryId = 100
    &attraction.Update()
endfor
```

vs

```
For each Attraction
    CategoryId = 100
endfor
```

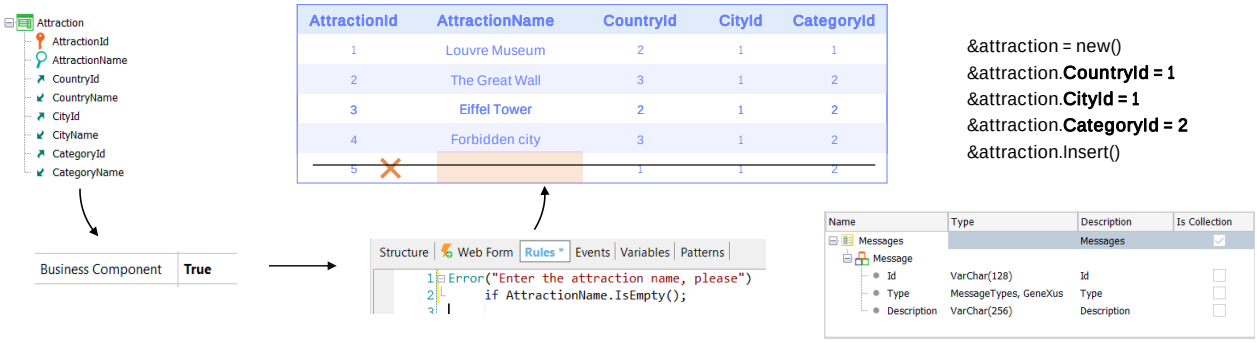
→

```
If find(CategoryId, CategoryId=100, 0) = 100
    For each Attraction
        CategoryId = 100
    endfor
endif
```

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
...	...	...	...	...

Sin embargo, si los registros a actualizar son millones o miles de millones, la cosa cambia un poco. Controlar por cada registro la integridad referencial antes de realizar la operación es una tarea que insume tiempo. Insignificante si se trata de pocos registros, pero muy importante si se trata de millones. En ese caso la actualización con comandos de procedimientos es LA solución si la performance se torna fundamental. En el ejemplo podríamos ejecutar el For each solo si estamos seguros de que existe la categoría 100. E incluso podríamos mejorar más la performance utilizando la cláusula blocking del for each.

	Uniqueness Check	Referential Integrity Check	Rules/Events execution
Business Component	✓	✓	✓
New, Assignment in For each, Delete	✓	✗	

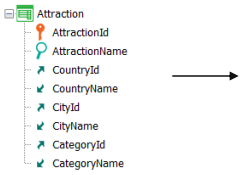


Y por último, ¿qué pasa con reglas y eventos definidos en la transacción?

Sabemos que la actualización por Business Components las y los ejecutarán (los que correspondan, claro, que no tengan que ver con al interfaz de la transacción).

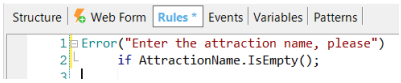
Así, si tenemos una regla de error que no permite ingresar una atracción sin nombre, e intentamos, por ejemplo, insertar una atracción nueva, autonumerada, a través del Business Component, ¿se permitirá? No, no se permitirá. Y el error quedará capturado en la colección de mensajes.

	Uniqueness Check	Referential Integrity Check	Rules/Events execution
Business Component	✓	✓	✓
New, Assignment in For each, 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
5		1	1	2

```
New
    CountryId = 1
    CityId = 1
    CategoryId = 2
endnew
```



En cambio, cuando se trata de comandos de actualización de la base de datos en procedimientos, la transacción no interviene en absoluto (o, bueno, solo interviene en la medida en que define la tabla de la base de datos, pero solo para eso). Entonces las reglas o eventos no son considerados para nada.

Por tanto, en el ejemplo, así exista regla de error, la inserción a través de comando New en un procedimiento no estará ni enterada, e insertará el registro con nombre vacío sin ningún problema.

Otra vez, aquí la balanza se inclina claramente hacia la utilización de Business Components en lugar de comandos de actualización en procedimientos, dado que nos aseguramos de que toda la lógica de datos se ejecute.

Podríamos repetirla en el procedimiento, pero sabemos los problemas que acarrea la duplicación.

Por otro lado, evidentemente ejecutar las reglas y eventos conlleva tiempo. Si la performance es un problema, entonces probablemente necesitemos actualizar la base de datos con comandos específicos de procedimientos.

	Uniqueness Check	Referential Integrity Check	Rules/Events execution	Scope
Business Component	✓	✓	✓	Any Object
New, Assignment in For each, Delete	✓	✗	✗	Procedure only!

Commit?

Rollback?

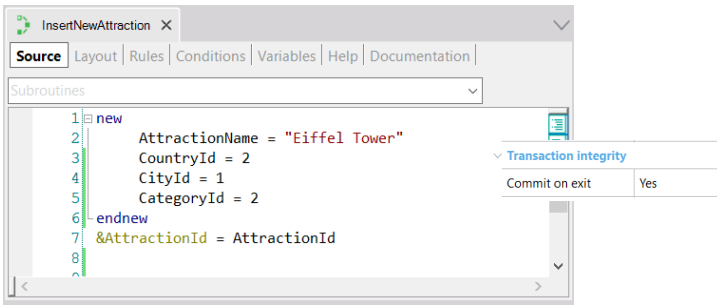
Una última ventaja de los Business Components frente a los comandos específicos de actualización, es que, mientras que los primeros pueden utilizarse prácticamente en cualquier objeto que admita código, por ejemplo, en eventos de Web Panels o Panels, los segundos solamente pueden programarse en procedimientos.

Sobre el Commit y Rollback vale lo mismo para ambas formas. Es decir, el desarrollador debe hacerse cargo de utilizarlos donde le convenga, más allá de que los procedimientos tengan la propiedad Commit on Exit por default en Yes.

Veamos un caso especial que puede llamar a confusión.

DEMO

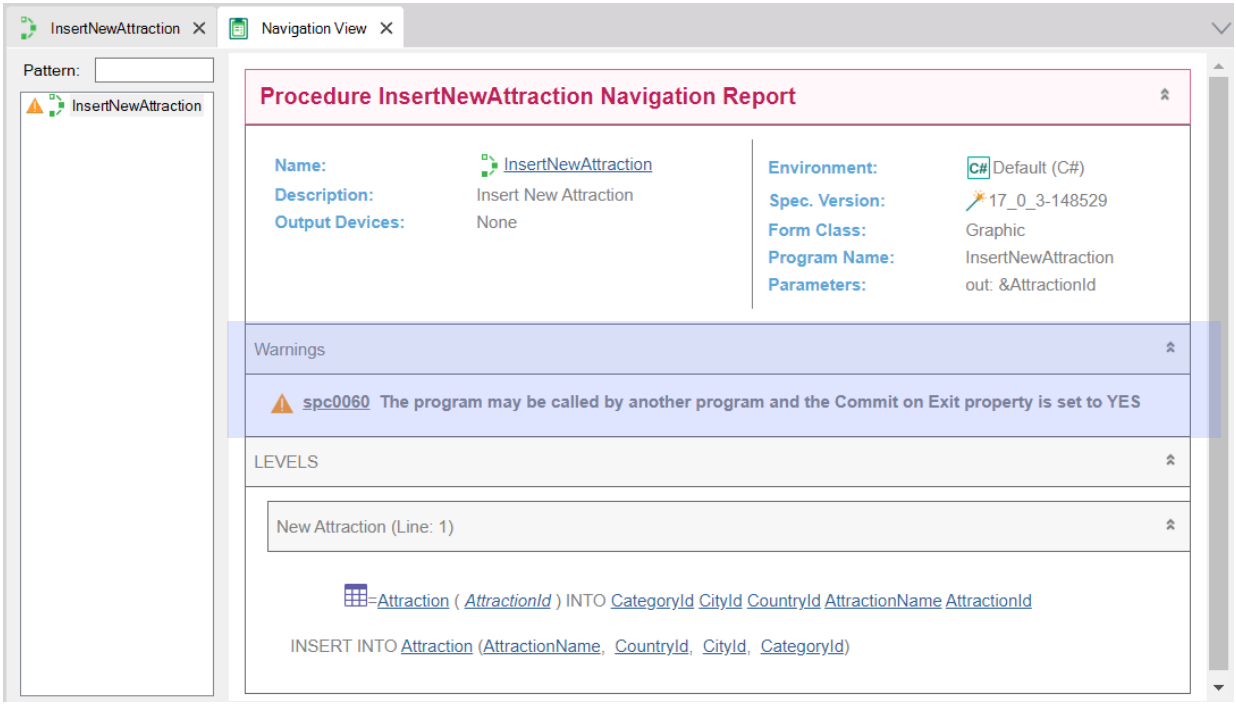

```
Event 'New attraction'
     InsertNewAttraction(&AttractionId)
     If not &AttractionId.IsEmpty()
          &text = "The attraction " + &AttractionId.ToString() + " was inserted"
     else
          &text = "The attraction could not be inserted"
     endif
     msg(&text)
Endevent
```



Supongamos que desde este Web Panel queremos llamar a este procedimiento que intentará insertar una atracción en la base de datos y nos devolverá su id, dado que es autonumerado.
Tenemos que decidir si insertamos el registro con el comando New o lo hacemos con el Business Component. Primero hagámoslo con el comando.

Como la propiedad Commit on exit está con su valor default, Yes, y claramente se está realizando una operación sobre la base de datos, entonces GeneXus coloca un commit implícito al final del código fuente del procedimiento.

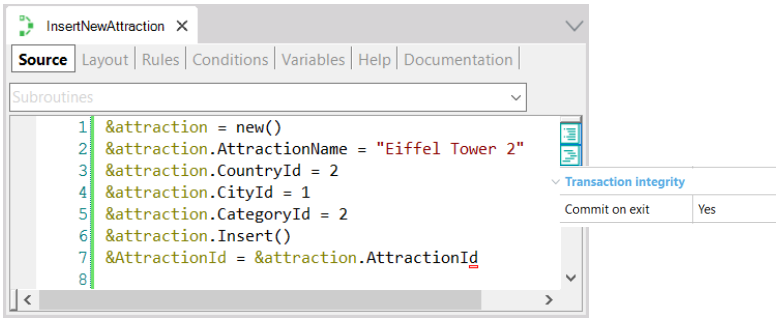
Por este motivo, sabemos que al volver de la ejecución del procedimiento, en esta sentencia si se pudo insertar el registro entonces ya se habrá commiteado.



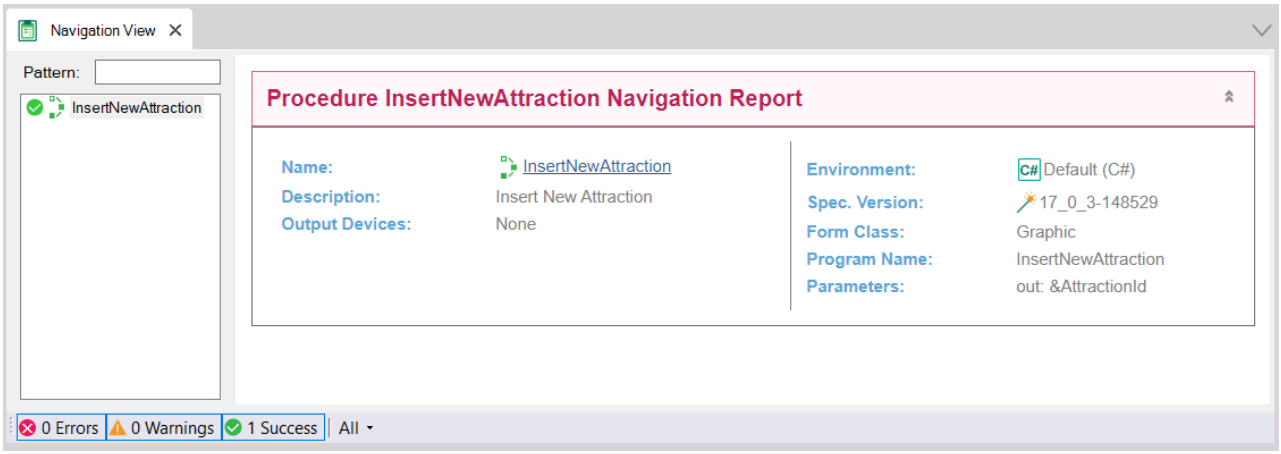
Y es por ello que el listado de navegación nos hace esta advertencia.

Veamos las atracciones que hay en la base de datos. Ahora ejecutemos. Volvamos a observar las atracciones. Tal como esperábamos, aquí encontramos la nueva atracción turística.


```
Event 'New attraction'
     InsertNewAttraction(&AttractionId)
     If not &AttractionId.IsEmpty()
          &text = "The attraction " + &AttractionId.ToString() + " was inserted"
     else
          &text = "The attraction could not be inserted"
     endif
     msg(&text)
Endevent
```



Pero ahora observemos qué hubiera pasado si en lugar de insertar con el comando New, lo hacemos con el Business Component (coloquémosle un 2 al nombre Eiffel Tower) y no explicitamos Commit, dado que tenemos la propiedad Commit on exit prendida.

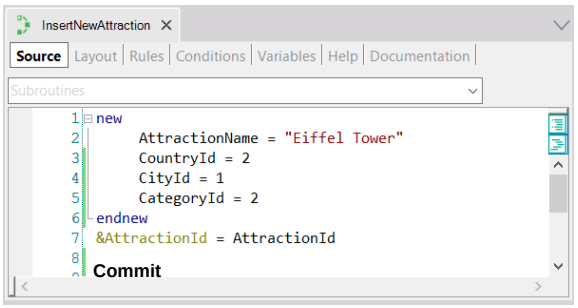


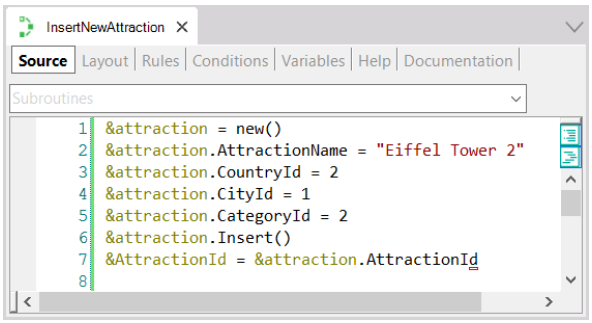
Si observamos el listado de navegación, ya vemos algo raro: no nos está haciendo la misma advertencia que en el otro caso, respecto a la propiedad Commit on exit.

Y luego, si ejecutamos, vemos que nos informa del siguiente número de atracción, lo que nos hace pensar que la insertó y commiteó correctamente, pero si vamos a buscarla a la transacción... ¡no está!

¿Qué pasó?

Commit on exit?





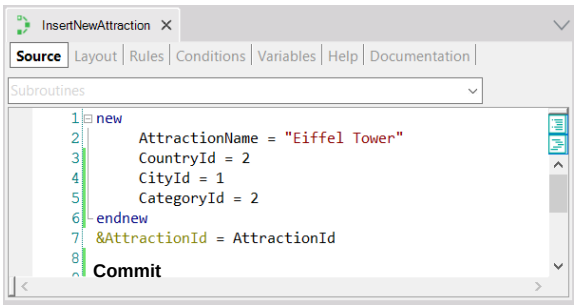
En videos anteriores dijimos repetidamente que el hecho de tener la propiedad Commit on exit de un procedimiento con su valor Yes no significaba que siempre GeneXus fuera a colocar un Commit al final del código fuente.

Dijimos que solo lo haría si encuentra que en el Source del procedimiento, en algún lado, se está queriendo acceder a la base de datos para hacer alguna operación de CRUD (Create, Update, Delete).

Así, en el primer caso claramente colocará un Commit, porque el comando New inequívocamente representa una operación de CRUD.

Sin embargo, no reconoce para el segundo caso una operación de ese tipo. Es que toma al Business Component por su estructura, como un SDT, y no logra interpretar el método Insert como corresponde. Por lo que no entiende que en el Source de este procedimiento se quiera hacer ninguna operación de CRUD. Y es por ello que no agrega el Commit al final del código fuente.

Commit on exit?



Así es que en este caso, por ahora no tendremos más remedio que explicitarlo.

Si ahora probamos... se comporta como esperábamos.

	Uniqueness Check	Referential Integrity Check	Rules/Events execution	Scope
Business Component	✓	✓	✓	Any Object
New, Assignment in For each, Delete	✓	✗	✗	Procedure only!

Commit?

Rollback?

Con esto vimos las principales diferencias entre realizar operaciones de CRUD con Business components y hacerlo directamente en procedimientos a través de los comandos específicos.

training.genexus.com
wiki.genexus.com
training.genexus.com/certifications