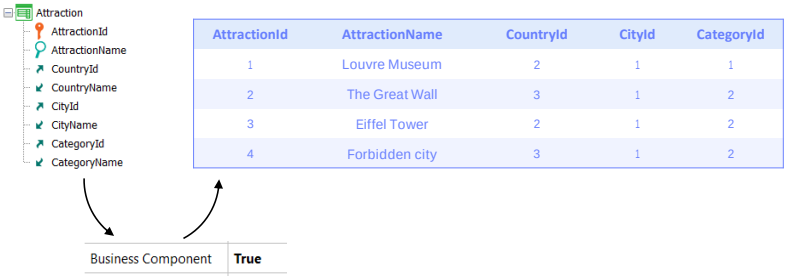
Atualização da base de dados

Business components vs Comandos específicos de procedimentos

GeneXus™

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

Insert, Update, Delete



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

Para atualizar a base de dados por código, tínhamos duas possibilidades:

Fazer isso utilizando o business component da transação, através de seus métodos, ou fazê-lo exclusivamente dentro de um procedimento, através dos comandos New, For each com atribuição direta dos atributos a serem modificados, e o 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, com ambas as opções é controlada a unicidade de registros, ou seja, nunca é permitido deixar na base de dados registros com chave primária ou candidata repetida. No exemplo, nunca serão permitidas duas atrações com o mesmo identificador. E se, por exemplo, AttractionName fosse chave candidata, também não seriam permitidas duas atrações com o mesmo nome.
Isto é verificado automaticamente pelo Business Component ou pelo comando no procedimento antes de tentar a operação na Base de Dados.

Até agora o que é igual. Agora vejamos as diferenças.

Quanto à garantia da integridade referencial, a inserção, atualização ou eliminação via Business Component verificará antes de realizar a operação que a integridade referencial não seja violada, tal como faz a transação. Por exemplo, se quiséssemos modificar a categoria da atração 3 para uma inexistente, o Business component controlará previamente que exista a categoria 100 na tabela Category e se não existir, **não tentará** a atualização. E também carregará uma mensagem de erro indicando isso na coleção de mensagens que nos retorna se solicitamos.

	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!

Ao contrário, as operações realizadas pelos comandos específicos em procedimentos não realizarão nenhum controle de integridade referencial. Isto significa que se, por exemplo, queremos atualizar a categoria da atração 3, agora por atribuição dentro de For each, não será consultada a tabela Category para saber se existe uma categoria 100 antes de enviar a ordem para a Base de Dados para que atualize esse registro. Portanto, se a Base de Dados também não controla a integridade referencial, ficaremos com um dado inconsistente. E se ela controla, como é o comportamento padrão, lançará uma exceção e o programa que estamos executando será cancelado.

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

Portanto, no momento, a balança parece inclinar-se para o 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
```

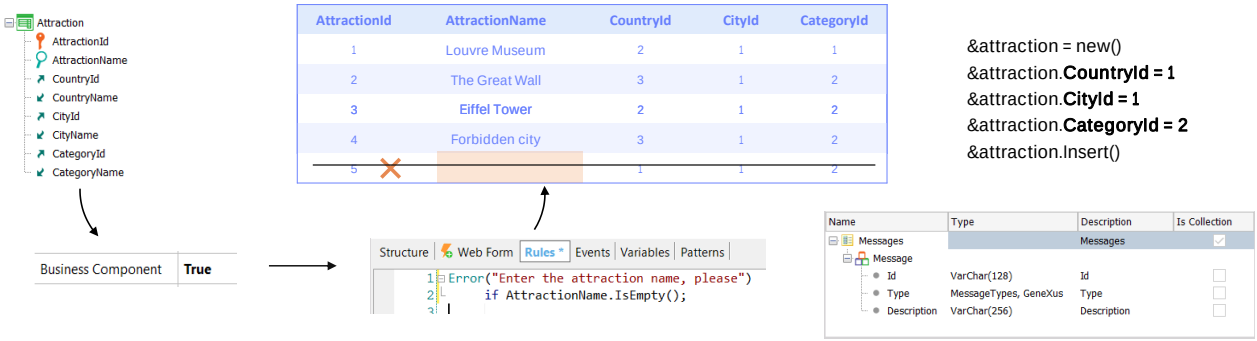
→

```
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
...	...	...	...	...

No entanto, se os registros a serem atualizados são milhões ou bilhões, a coisa muda um pouco. Controlar para cada registro a integridade referencial antes de realizar a operação é uma tarefa que leva tempo. Insignificante se trata-se de poucos registros, mas muito importante se trata-se de milhões. Nesse caso, a atualização com comandos de procedimentos é A solução se o desempenho se torna fundamental.
No exemplo, poderíamos executar o For each apenas se tivermos certeza de que existe a categoria 100. E até poderíamos melhorar ainda mais o desempenho utilizando a cláusula blocking do for each.

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

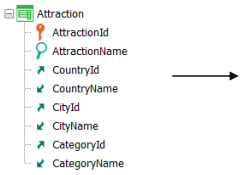


E, por último, o que acontece com regras e eventos definidos na transação?

Sabemos que a atualização por Business Components os executarão (aqueles que correspondam, claro, que não tenham a ver com a interface da transação).

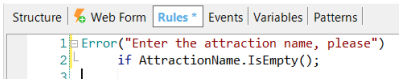
Assim, se temos uma regra de erro que não permite inserir uma atração sem nome, e tentamos, por exemplo, inserir uma nova atração, autonumerada, através do Business Component, será permitido? Não, não será permitido. E o erro será capturado na coleção de mensagens.

	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
```



Em vez disso, quando se trata de comandos de atualização da base de dados em procedimentos, a transação não interfere de forma alguma (ou, bem, apenas intervém na medida em que define a tabela da base de dados, mas apenas para isso). Portanto, as regras ou eventos não são considerados para nada.

Portanto, no exemplo, mesmo que haja uma regra de erro, a inserção através do comando New em um procedimento não ficará atenta, e vai inserir o registro com nome vazio sem nenhum problema.

Outra vez, aqui a balança se inclina claramente para a utilização de Business Components em vez de comandos de atualização em procedimentos, pois nos asseguramos de que toda a lógica de dados seja executada.

Poderíamos repeti-la no procedimento, mas sabemos os problemas que acarreta a duplicação.

Por outro lado, obviamente executar as regras e eventos leva tempo. Se o desempenho for um problema, então provavelmente precisaremos atualizar a base de dados com comandos específicos de procedimentos.

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

Commit?

Rollback?

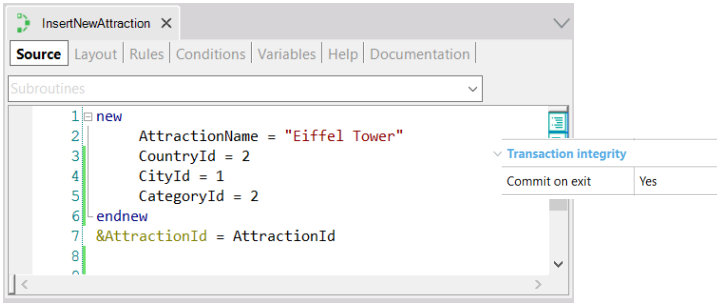
Uma última vantagem dos Business Components em comparação com comandos específicos de atualização é que, enquanto os primeiros podem ser utilizados praticamente em qualquer objeto que suporte código, por exemplo, em eventos de Web Panels ou Panels, os segundos só pode ser programados em procedimentos.

Sobre o Commit e Rollback vale o mesmo para ambas as formas. Em outras palavras, o desenvolvedor deve cuidar de utilizá-los onde lhe for conveniente, independentemente de que os procedimentos tenham a propriedade Commit on Exit por default em Yes.

Vejamos um caso especial que pode ser confuso.

DEMO

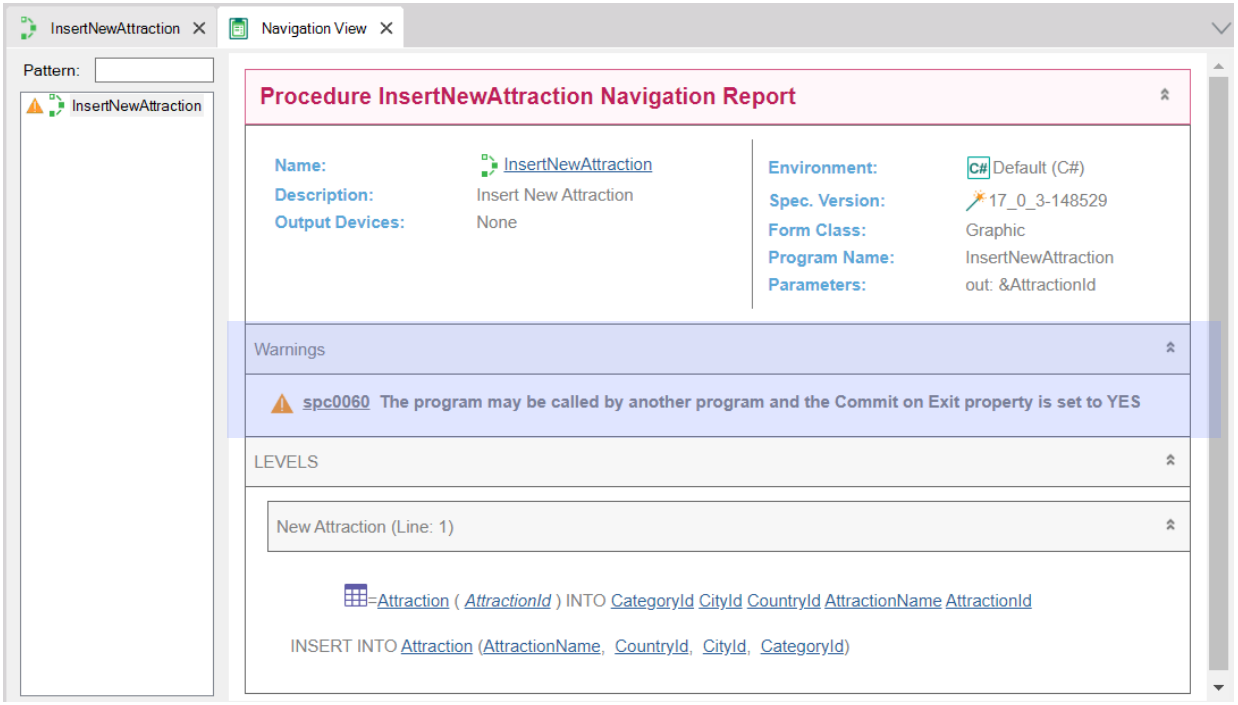

```
Event 'New attraction'
     InsertNewAttraction(&AttractionId)
     If not &AttractionId.IsEmpty()
          &text = "The attraction " + &AttractionId.ToString() + " was inserted"
     else
          &text = "The attraction could not be inserted"
     endif
     msg(&text)
Endevent
```



Suponhamos que a partir deste Web Panel queremos chamar este procedimento que tentará inserir uma atração na base de dados e nos devolverá seu id, uma vez que é autonumerado.
Temos que decidir se inserimos o registro com o comando New ou o fazemos com o Business Component. Façamos primeiro com o comando.

Como a propriedade Commit on exit está com seu valor default, Yes, e claramente se está realizando uma operação na base de dados, então GeneXus coloca um commit implícito no final do código fonte do procedimento.

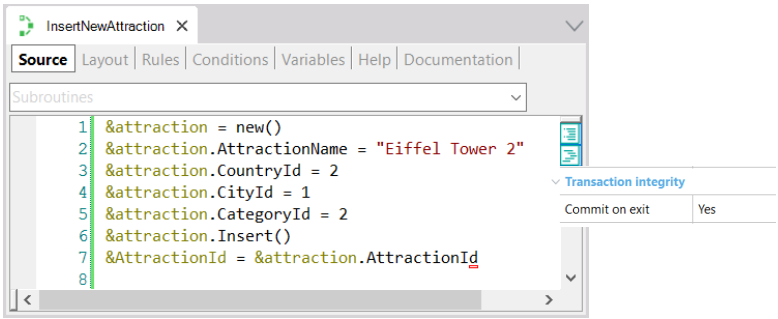
Por este motivo, sabemos que ao retornar da execução do procedimento, nesta sentença se foi possível inserir o registro, então ele já terá sido commitado.



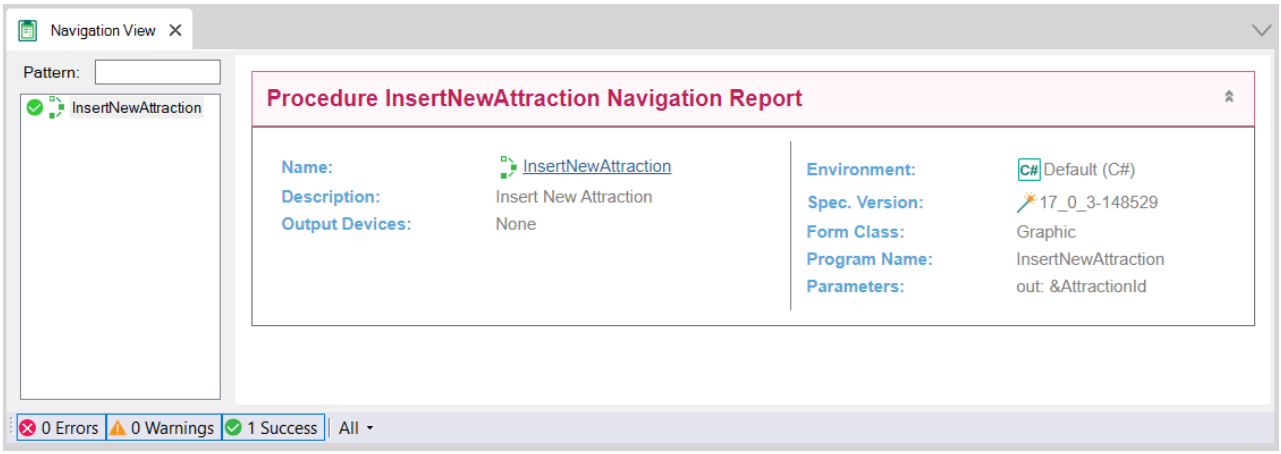
E é por isso que a lista de navegação nos dá este aviso.

Vejamos as atrações que existem na base de dados. Agora executemos. Voltemos a observar as atrações. Tal como esperávamos, aqui encontramos a nova atração 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
```



Mas agora observemos o que teria acontecido se em vez de inserir com o comando New, o fazemos com o Business Component (vamos colocar um 2 no nome Eiffel Tower) e não explicitamos Commit, já que temos a propriedade Commit on exit ativada.

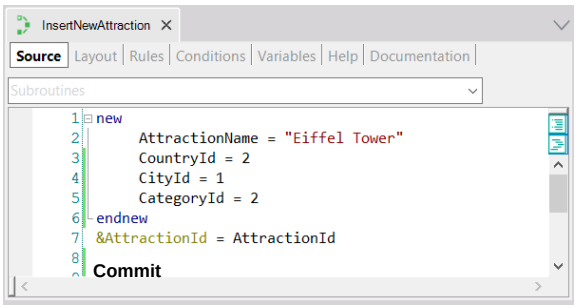


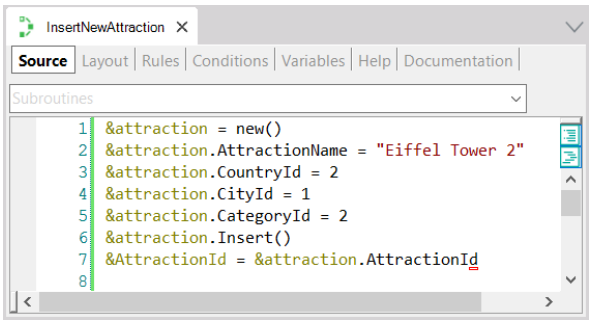
Se observamos a lista de navegação, já vemos algo estranho: não está nos dando o mesmo aviso que no outro caso, a respeito da propriedade Commit on exit.

E então, se executamos, vemos que nos informa o próximo número de atração, o que nos faz pensar que foi inserida e commitada corretamente, mas se vamos procurá-la na transação... não está!

O que aconteceu?

Commit on exit?





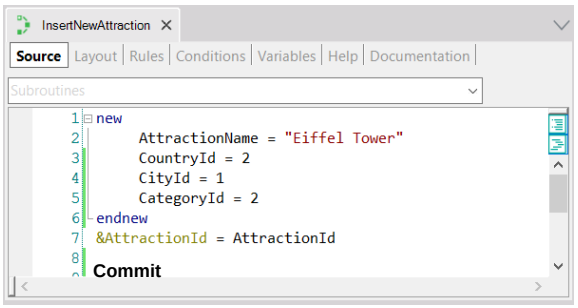
Em vídeos anteriores dissemos repetidamente que o fato de ter a propriedade Commit on exit de um procedimento com seu valor Yes não significava que GeneXus sempre colocaria um Commit no final do código-fonte.

Dissemos que só o faria se descobre que no Source do procedimento, em algum lugar, está querendo acessar a base de dados para fazer alguma operação de CRUD (Create, Update, Delete).

Portanto, no primeiro caso, claramente colocará um Commit, porque o comando New inequivocamente representa uma operação de CRUD.

No entanto, não reconhece para o segundo caso uma operação desse tipo. É que toma o Business Component por sua estrutura, como um SDT, e não consegue interpretar o método Insert como corresponde. Portanto, não entende que no Source deste procedimento deseja-se fazer qualquer operação de CRUD. E é por isso que não adiciona o Commit ao final do código-fonte.

Commit on exit?



Portanto, neste caso, por enquanto, não teremos escolha a não ser torná-lo explícito.

Se testamos agora... se comporta como esperávamos.

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

Commit?

Rollback?

Com isto vimos as principais diferenças entre realizar operações de CRUD com Business components e fazê-lo diretamente em procedimentos através dos comandos específicos.

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